



ELEKTROTEHNIŠKO-RAČUNALNIŠKA
STROKOVNA ŠOLA IN GIMNAZIJA
LJUBLJANA

Tiny Kingdoms

Strokovno poročilo

Mentor: Darjan Toth, prof.

Avtor: Gregor Kovač, R 4. B

Ljubljana, april 2020

Zahvala

Zahvalil bi se profesorju Darjanu Tothu za pomoč pri izdelavi svojega maturitetnega izdelka.

Povzetek

V tem strokovnem poročilu poročam o izdelavi svojega izdelka za četrti predmet na maturi, igre z naslovom Tiny Kingdoms. Najprej na splošno opišem, kako moj izdelek funkcioniра. Nato na kratko predstavim vso programsko opremo, ki sem jo uporabil. Potem podrobno opišem postopek izdelave. Dotaknem se vsega, od načrtovanja na papir in oblikovanja digitalnega načrta v začetnih stopnjah izdelave, do samega programiranja funkcionalnosti in grafičnega oblikovanja, ter popravljanja napak in implementiranja različnih dodatkov na koncu. Nato še izrazim svoje mnenje o končnem izdelku. Izpostavim tudi nekaj stvari, ki bi jih lahko še dodal. Zaključim z navedbo vsega znanja, ki sem ga pridobil med izdelavo.

Ključne besede: računalniška igra, programiranje, C#, Unity, Blender

Abstract

In this expert report I write about the development of my product for the fourth matura subject, a game titled Tiny Kingdoms. First I give a brief description of how my product functions. Then I shortly present all of the software I used. After that I precisely describe the whole development procedure. I touch on everything, from planning on paper and designing a digital plan in the first stages of development, to programming the functionality, designing the graphics, fixing errors and implementing different additions at the end. Then I express my opinion about the product. I also point out some things I could add. I conclude with listing all of the knowledge I gained during the development.

Key words: computer game, programming, C#, Unity, Blender

Kazalo vsebine

1. UVOD	7
2. OPIS IGRE	7
3. PROGRAMSKA OPREMA	8
3.1 UNITY	8
3.2 VISUAL STUDIO	9
3.3 BLENDER	9
3.4 GIMP	9
3.5 GARAGEBAND	9
4. POSTOPEK IZDELAVE	9
4.1 NAČRTOVANJE	9
4.2 OGRODJE	10
4.3 IGRALNA PLOŠČA	10
4.4 IGRALČEVO KRALJESTVO	10
4.5 GENERIČNA ZGRADBA	11
4.6 BARAKE	12
4.7 NEIGRALSKA KRALJESTVA	12
4.8 UREJENOST SKRIPT KRALJESTEV	14
4.9 UPRAVLJALEC Z NEIGRALKIMI OBJEKTI	14
4.10 TRGOVANJE	16
4.11 PRISTANIŠČE	16
4.12 NAPADANJE IN BRANJENJE KRALJESTEV	16
4.12.1 Napadalna vojska	17
4.12.2 Obrambna vojska	17
4.13 PAVZA	18
4.14 NASLOVNI MENI	18
4.15 SHRANJEVANJE IN NALAGANJE IGRE	18
4.15.1 Razred podatkov	19
4.15.2 Shranjevanje	20
4.15.3 Nalaganje	21

4.16	KONEC IGRE	21
4.17	LESTVICA REZULTATOV	22
4.18	GRAFIČNA PODOBA	22
4.19	ZVOČNI UČINKI	23
4.20	MANJŠI DODATKI, POPRAVKI IN RAZHROŠČEVANJE	23
4.20.1	Pisava in meniji	23
4.20.2	Prikaz statusa surovin	24
4.20.3	Sprememba perspektive	24
4.20.4	Indikator izgube življenskih točk	25
4.20.5	Popravek postavljanja zgradb	25
4.20.6	Prikaz potrebnih surovin	25
4.20.7	Tekoče premikanje zgradb	25
5.	ZAKLJUČEK	26
6.	VIRI IN LITERATURA	27

Kazalo slik

Slika 1: Izgled igre	8
Slika 2: Funkcija za postavljanje igralčevega kraljestva s komentarji	11
Slika 3: Prilagodljiv meni skripte generične zgradbe	12
Slika 4: Spremenljivke generične zgradbe s komentarji	12
Slika 5: Koda delovanja neigralskega kraljestva s komentarji, 1. del	13
Slika 6: Koda delovanja neigralskega kraljestva s komentarji, 2. del	13
Slika 7: Funkcija za postavljanje neigralskega kraljestva s komentarji	14
Slika 8: Funkcija za postavljanje nove neigralske zgradbe s komentarji, 1. del	15
Slika 9: Funkcija za postavljanje nove neigralske zgradbe s komentarji, 2. del	15
Slika 10: Meni za trgovanje	16
Slika 11: Funkcija za iskanje nove tarče s komentarji	17
Slika 12: Naslovni meni	18
Slika 13: Razred podatkov s komentarji, 1. del	19
Slika 14: Razred podatkov s komentarji, 2. del	19
Slika 15: Koda za iskanje neigralskih kraljestev s komentarji	20
Slika 16: Funkcija za shranjevanje igre s komentarji	20
Slika 17: Lestvica rezultatov	22
Slika 18: Model farme v Blenderju	23
Slika 19: Prva različica perspektive	24
Slika 20: Končna različica perspektive	24

1. Uvod

Četrti predmet na poklicni maturi je izdelek ali storitev. Jaz sem želel ustvariti nek programsko orientiran izdelek, saj mi gre programiranje veliko bolje od del, povezanih s strojno opremo. Odločil sem se, da mi bo najbolj ustrezano, če naredim računalniško igro, glede na to, da sem v tem dokaj vešč, ker imam že predhodne izkušnje z izdelavo iger. Po temeljitem razmisleku sem sklenil, da bom ustvaril minimalistično strateško igro z naslovom Tiny Kingdoms. V tej seminarSKI nalogi jo bom predstavil in podrobno opisal postopek izdelave.

2. Opis igre

Tiny Kingdoms je igra, postavljena v srednji vek. Igralec ima pogled na igralno površino iz izometrične perspektive in se lahko premika v vse smeri. Igralna plošča je naključno generirana in sestavljena iz manjših ploščic, ki predstavljajo različne vrste pokrajin. Le-te so travnik, gozd, gorovje in obala. Prve tri se generirajo v večjih skupinah, obala pa jih obdaja. Igralec v tej igri v osnovi gradi svoje kraljestvo. Na začetku igre ima možnost postaviti svoj grad na enega od prostih travnikov na igralni površini. Ko je le-ta postavljen, se igra začne. Igralcu so na razpolago različne surovine. To so žito, les, kamen, oglje, železo in zlato. Na začetku začne z naključno količino surovin. Z njimi lahko postavlja zgradbe. To so kmetija, rudnik, koča, barake in pristanišče. Vsako od njih, razen pristanišča, pa lahko tudi nadgradi za hitrejšo proizvodnjo materialov. Kmetija prinaša žito, koča les, rudnik pa kamen, oglje, železo in zlato. Na pristanišču pristanejo trgovske ladje, s katerimi lahko menjamo surovine. Barake izurijo meščane v vojake, ki se lahko pridružijo obrambni ali napadalni vojski. Na začetku igralec začne z določenim številom meščanov, ki pričnejo umirati, če jih pravočasno ne nahrani. Če nadgradi kraljestvo, se poveča število meščanov, hkrati pa se tudi razširi površina, na kateri lahko postavlja zgradbe.

Na igralni površini se naključno generirajo tudi druga kraljestva, ki imajo naključno število zgradb in naključno količino surovin. Vsakemu tujemu kraljestvu je dodeljen tudi status odnosa z našim kraljestvom. Ta je lahko od -50 do 50. Pod 0 pomeni, da je kraljestvo agresivno proti nam, 0 pomeni, da je nevtralno, nad 0 pa pomeni, da je z nami v prijateljskem odnosu. Če je proti nam agresivno, nas lahko v katerem koli trenutku napade, če pa je v prijateljskem odnosu, pa lahko z njim trgujemo. Vsako kraljestvo lahko podkupimo z zlatom in mu tako povišamo status odnosa.

Ko je neko kraljestvo napadeno, se proti njemu začne pomikati napadalna vojska. Le-to se bo skušalo braniti s svojo obrambno vojsko. Napadalci bodo šli od zgradbe do zgradbe in jih rušili ter tako pridobivali surovine. Če je moč obrambne vojske dovolj velika, bo pravočasno ustavila napadalno vojsko. Igralec igro izgubi, če mu nekdo drug uniči njegovo kraljestvo.



Slika 1: Izgled igre (Lastni vir)

3. Programska oprema

Za izdelavo igre Tiny Kingdoms sem uporabil pet ključnih aplikacij. To so Unity, Visual Studio, Blender, GIMP in GarageBand. V nadaljevanju jih bom natančneje predstavil.

3.1 Unity

Unity je program za izdelavo računalniških iger. Izdalo ga je podjetje Unity Technologies junija 2005, zadnja posodobitev pa je izšla februarja 2020. Ta program omogoča izdelavo 2D in 3D računalniških iger na relativno enostaven in vizualen način. Trenutno omogoča programiranje v jeziku C#, prej pa je podpiral tudi programska jezika Boo in UnityScript. Unity sam poskrbi za prikaz grafike in nam omogoča enostavno implementacijo grafičnega prikaza našega izdelka. Program podpira izdajo računalniških iger na večini večjih platform, kot so Windows, Mac, Linux, WebGL, PlayStation 4, XBox One, iOS, Android, Android TV in Oculus Rift. Unity lahko pridobimo v štirih različnih paketih, to so Unity Free, Unity Plus, Unity Pro in Unity Enterprise. Unity Free je brezplačen, drugi paketi pa so plačljivi in nam omogočajo dodatne funkcije. Jaz sem uporabil Unity Free.

"Ena najboljših stvari o uporabi Unityja je to, da se lahko zahvaljujoč hitrosti, s katero iteriraš, zelo hitro naučiš, kaj deluje in kaj ne." (Chayes, 2014)

Iz zgornjega citata lahko razberemo, da je ena od boljših funkcij Unityja možnost zelo hitrega testiranja igre. Z veliko izvedenimi ponovitvami lahko hitro odkrivamo različne napake. To mi je med izdelavo moje igre prišlo zelo prav.

3.2 Visual Studio

Visual Studio je integrirano razvojno okolje (IDE), ki ga je izdalo podjetje Microsoft leta 1997, zadnja posodobitev pa je izšla februarja 2020. Ta program omogoča izdelavo različnih računalniških programov, jaz pa sem ga uporabil le za urejanje kode. Visual Studio podpira 36 različnih programskih jezikov, kot so C, C++, Visual Basic, C# in JavaScript. Community verzija programa je brezplačna, na voljo pa sta tudi plačljivi različici Professional in Enterprise. Jaz sem uporabil Visual Studio Community.

3.3 Blender

Blender je brezplačen in odprtokoden program za izdelavo 3D grafik. Izdal ga je Ton Roosendaal januarja 1994, zadnja različica pa je izšla februarja 2020. Ta program omogoča izdelavo 3D modelov, animacij in vizualnih efektov. Jaz sem ga uporabil za izdelavo modelov.

3.4 GIMP

GIMP oziroma GNU Image Manipulation Program je brezplačen in odprtokoden program za obdelavo rasterskih slik. Izdala sta ga Spencer Kimball in Peter Mattis leta 1996, zadnja verzija pa je izšla februarja 2020. GIMP nam omogoča osnovne funkcije za obdelavo slik, kot so barvanje, brisanje in dodajanje grafičnih učinkov. Jaz sem ga uporabil za risanje ikon in tekstur za svoj izdelek.

3.5 GarageBand

GarageBand je program za obdelavo zvoka, ki se najpogosteje uporablja za izdelavo glasbe. Izdal ga je podjetje Apple januarja 2004, zadnja različica pa je izšla decembra 2019. Podprt je na napravah z macOS ali iOS operacijskim sistemom. Omogoča veliko različnih funkcij za obdelavo zvoka in ustvarjanje glasbe. Jaz sem ga uporabil za ustvarjanje zvočnih efektov svoje igre.

4. Postopek izdelave

4.1 Načrtovanje

Izdelavo svojega izdelka sem začel z načrtom. Najprej sem razmislil o tem, kaj želim, da bo moja računalniška igra delala. Po dobrem razmisleku in nekaj različnih domislicah sem prišel do ideje, ki je kasneje postala moj izdelek. Le-to sem najprej grobo opisal na list papirja in narisal nekaj skic za lažjo predstavo. Ko sem bil s tem osnutkom zadovoljen, sem se lotil pisanja bolj podrobнega načrta v urejevalniku besedil. Opisal sem vse podrobnosti delovanja svoje igre in na koncu narisal še diagram menijev in njihov potek. Ko sem zaključil z načrtovanjem, sem razmislil tudi o tem, kako se bom lotil izdelave svojega projekta. Odločil sem se, da bom najprej implementiral funkcionalnost in se nato ukvarjal z grafično podobo, končna faza bi bilo pa še razhroščevanje.

4.2 Ogrodje

Najprej sem se lotil postavitve in organizacije projektne datoteke in osnovne kode za premikanje igralčevega pogleda. Ustvaril sem nov projekt v Unityju in znotraj njega najprej naredil nekaj novih map za boljšo organizacijo med postopkom izdelave. Nato sem v prazno sceno postavil kocko in kamero. Potem sem začel programirati premikanje kamere, kar je bilo zelo lahko delo. Omogočil sem premikanje v štiri smeri in sicer s tipko "W" za gor, "S" za dol, "A" za levo ter "D" za desno. Določil sem tudi meje premikanja, saj se igralec ne sme premikati dlje kot do roba igralne plošče.

4.3 Igralna plošča

Kot sem omenil že v opisu igre je igralna plošča sestavljena iz manjših naključno generiranih ploščic. Najprej sem ustvaril objekt za generično ploščico in napisal del programa, ki ustvari igralno ploščo iz le-teh. Ta koda najprej naredi dvodimenzionalno tabelo celih števil, ki predstavljajo strukturo plošče. Vsako število pomeni tip ploščice. Nato program pregleda to tabelo in postavi ploščice glede na njeno vsebino. Ustvaril sem štiri objekte za različne tipe pokrajin, to so travnik, gozd, gorovje in obala. Zanje sem v kodi igralne plošče deklariral štiri spremenljivke. Ko sem program dodelal tako, da lahko postavlja vse različne tipe ploščic glede na števila v tabelo, sem se lotil algoritma za naključno generiranje igralne plošče. Najprej program na robe postavi številčno vrednost obale, v notranjosti pa naključno generira vrednost za eno od ostalih pokrajin. Nato gre še enkrat čez tabelo in vsako vrednost kopira v naključno število njenih sosedov. Ta postopek ponovi petkrat za lepši končni izgled. Nato ponovno pregleda tabelo in preveri, če obstaja zadostno število posameznih ploščic. Če jih je premalo, jih naključno generira še nekaj.

4.4 Igralčeve kraljestvo

Po ustvarjeni igralni plošči sem se lotil izdelave osnovnih zmožnosti igralčevega kraljestva. Začel sem s postavljanjem. Ko se igra začne, kraljestvo najprej sledi igralčevi miški. Ustvaril sem funkcijo, ki bere njegovo lokacijo in preverja, če se pod njim nahaja travnik. Glede na to ga premika po igralni plošči. Ko igralec klikne z miško, je kraljestvo postavljeno. Nato sem ustvaril spremenljivke za vse različne materiale, meščane in viteze. Njihove vrednosti prikazujem v levem zgornjem kotu okna igre. Želel sem, da lahko igralec odpre meni, ko klikne na kraljestvo. To sem storil s preprostim delom kode, ki preverja, če je uporabnik kliknil na kraljestvo in prikazuje meni glede na rezultat tega preverjanja. V meniju sem ustvaril nekaj gumbov. Eden se uporablja za nadgradnjo kraljestva, eden za postavitev drugih zgradb, eden pa za hranjenje meščanov. Dodal sem še kodo, ki začne manjšati število meščanov, če jih ne hranimo dovolj pogosto. Ustvaril sem osnovne funkcije zanke in se nato lotil izdelave generične zgradbe.

```

void Place()
{
    targetPos = new Vector3(0,0,0); // Inicializira se spremenljivka ciljne pozicije
    if (pauseMenu.GetComponent<PauseMenu>().pause == false) { // Kraljestvo se ne premika, če je vklopljena pavza
        ray = Camera.main.ScreenPointToRay(Input.mousePosition); // Ustvari se "žarek", ki potuje iz pozicije miške skozi kamerø
        if (Physics.Raycast(ray, out hit, Mathf.Infinity)) { // Preveri, če je "žarek" zadel kakšen objekt
            // Pošte se ploščica na mestu, ki ga je zadel "žarek"
            int tile = map.GetComponent<Map>().getTile((int)(Mathf.Round(hit.point.x / 10)), (int)(Mathf.Round(hit.point.z / 10)));
            if (tile == 1) {
                /* Če je ploščica travnik (označen z 1), ciljna pozicija postane točka, ki jo je zadel žarek,
                 * vendar se zaokroži na desetice, ker so ploščice velike 10x10
                */
                targetPos = new Vector3(Mathf.Round(hit.point.x / 10) * 10, 2.5f, Mathf.Round(hit.point.z / 10) * 10);
            }
        }
    }

    // Preverim, če je igralec pritisnil levi klik in če kraljestvo stoji na travniku
    if (Input.GetKey(KeyCode.Mouse0) && map.GetComponent<Map>().getTile((int)(Mathf.Round(transform.position.x / 10)),
        (int)(Mathf.Round(transform.position.z / 10))) == 1) {
        isPlaced = true; // Kraljestvo je zdaj postavljeno, zato nastavim spremenljivko isPlaced na true

        // Funkcijo buildingPlaced na igralni plošči označi mesto, kjer je bilo kraljestvo postavljen
        map.GetComponent<Map>().buildingPlaced((int)(Mathf.Round(hit.point.x / 10)), (int)(Mathf.Round(hit.point.z / 10)), -1);
    }

    if (targetPos.y == 0f)
        targetPos = prevHit; // Ciljna pozicija je enaka prejšnji ciljni poziciji, če nova pozicija ni določena

    // Če pozicija kraljestva ni enaka ciljni poziciji, ga je potrebno premakniti
    if (transform.position != targetPos){
        float nx = 0, nz = 0; // nx je premik v smeri x, nz pa premik v smeri z
        mspeed = 1f; // mspeed je velikost premika

        /* Preverjam, ali je x koordinate kraljestva večja ali manjša x koordinati ciljne pozicije
         * in jo temu primereno predznačim. Enako storim za z koordinato */
        if (targetPos.x > transform.position.x) {
            nx = mspeed;
        } else if (targetPos.x < transform.position.x) {
            nx = -mspeed;
        }
        else if (targetPos.z > transform.position.z) {
            nz = mspeed;
        }
        else if (targetPos.z < transform.position.z) {
            nz = -mspeed;
        }

        // Kraljestvo premaknem za nx v smeri x in za nz v smeri z
        transform.position = new Vector3(Mathf.Floor((transform.position.x+nx)), transform.position.y,
            Mathf.Floor((transform.position.z+nz)));
    }

    prevHit = targetPos; // Shramim ciljno pozicijo v spremenljivko za prejšnjo po
}
}

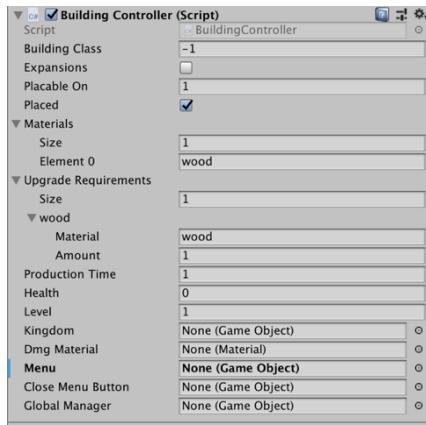
```

Slika 2: Funkcija za postavljanje igralčevega kraljestva s komentarji (Lastni vir)

4.5 Generična zgradba

Želel sem, da bi izdelava moje igre potekala čim bolj gladko in dinamično. Prav zaradi tega razloga sem se odločil za izdelavo generične zgradbe, ki bi jo lahko spremenil v kakršno koli drugo zgradbo le z nekaj manjšimi spremembami v Unityjevem urejevalniku. Tekom izdelave se je ta odločitev izkazala za zelo dobro.

Ustvaril sem objekt z nadomestljivim grafičnim modelom in skripto. Ta skripta vsebuje spremenljivke naprimer za identiteto zgradbe, razred zgradbe (le-ta se uporablja za postavljanje le na določeno pokrajino), tabelo materialov, ki jih zgradba proizvaja, strukturo zahtev za nadgradnjo (le-ta vsebuje ime materiala in količino tega materiala, ki jo potrebujemo), tabelo teh zahtev, življenske točke, nivo in starševsko kraljestvo. Vsebuje tudi funkcijo za postavljanje, ki je zelo podobna tisti, ki jo vsebuje kraljestvo, funkcijo za nadgradnjo in kodo za proizvodnjo materialov. Ko je bila generična zgradba popolnoma funkcionalna, sem v relativno kratkem času iz nje naredil še kmetijo, rudnik, kočo, barake in pristanišče. Postavljanje teh zgradb sem nato implementiral v igralčeve kraljestvo in ustvaril še zahteve materialov, ki jih potrebujemo za njihovo gradnjo.



Slika 3: Prilagodljiv meni skripte generične zgradbe (Lastni vir)

```
// Unikatna id številka zgradbe
[HideInInspector] public int id = -1;

// Tip zgradbe
public int buildingClass;
// -1 = kraljestvo, kmetija, barake
// -2 = koča
// -3 = rudnik
// -4 = pristanišče

/*
Nekatere zgradbe (naprimjer barake) imajo dodatno skripto.
Takrat spremenljivko expansions nastavim na true
*/
public bool expansions = false;

/* Spremenljivka, ki se uporablja za določanje tipa ploščice, na katero
lahko postavimo zgradbo */
public int placableOn;

// Spremenljivka, ki je true, če je zgradba že postavljena
public bool placed = true;

// Tabela surovin, ki jih proizvaja zgradba
public string[] materials;

// Spremenljivka za naključno generiranje surovine, ki jo bo dobilo kraljestvo
int randMaterial;

// Struktura zahtev za nadgradnjo
[System.Serializable]
public struct upgradeRequirement
{
    // Surovina
    public string material;

    // Potrebna količina te surovine
    public int amount;
};

// Rabela zahtev za nadgradnjo
public upgradeRequirement[] upgradeRequirements;

// Čas, v katerem se proizvede nova surovina
public float productionTime = 1f;
```

Slika 4: Spremenljivke generične zgradbe s komentarji (Lastni vir)

4.6 Barake

Za polno funkcionalnost barak sem potreboval še eno dodatno skripto. Le-ta bi skrbela za treniranje vitezov in njihovo postavitev v obrambno ali napadalno vojsko. Kodo sem napisal hitro, saj sem lahko uporabil nekaj manjših delcev iz skripte generične zgradbe. Igralec ima na razpolago gumb, s katerim prične trening. Število vitezov v barakah se začne večati, število meščanov v kraljestvu pa se manjša. Ko število meščanov pade na nič, se trening začasno ustavi. Ko imamo v barakah nekaj vitezov, jih lahko postavimo v obrambno ali napadalno vojsko s pomočjo dveh gumbov.

4.7 Neigralska kraljestva

Programiranje drugih kraljestev, ki jih ne upravlja igralec je bilo malo večji izziv. Začel sem z nekaj osnovami, ki sem jih prevzel iz igralčevega kraljestva. Obdržal sem enako

strukturo objekta, napisal pa sem novo skripto. Za začetek sem kopiral vsebino skripte igralčevega kraljestva in ji odstranil vse možnosti upravljanja z miško in gumbi.

Že predhodno sem razmislil, kako bo funkcionalita umetna inteligence neigralskih kraljestev. V rednih časovnih intervalih se ponavlja izvrševanje funkcij, ki skrbijo za različna opravila kraljestva. Najprej preveri količino vseh surovin in ukrepa, če jih je premalo. Če naprimer primanjkuje lesa, bo kraljestvo zgradilo novo kočo ali nadgradilo že eno od obstoječih. Nato bo nahranilo meščane, nato pa bo treniralo viteze. Če je napadalna vojska številčnejša od obrambne, bo nekaj vitezov dodeljenih v obrambno vojsko, enako pa velja tudi obratno. Nato bo generiralo naključno število in če to število ustreza pogoju, bo napadlo igralca. To se zgodi le v primeru, če je status odnosa z igralcem manjši od nič. Implementacija vseh teh ukazov ni bila zahtevna, temveč le časovno precej potratna. Za boljšo organiziranost in lažje delo sem različna opravila napisal v različne funkcije in jih samo klical drugo za drugo.

```
// Vse funkcije se kličejo le v primeru, da kraljestvo ne napada igralec
if (playerKingdom.GetComponent<KingdomController>().attackArmy.GetComponent<AttackArmy>().isAttacking == false)
{
    timer -= 1f * Time.deltaTime * globalManager.GetComponent<GlobalManager>().timeMultiplier;

    if (timer <= 0)
    {
        /* Funkcije se kličejo v rednih časovnih intervalih, ki jih
        dolga spremenljivka refreshTime */

        timer = refreshTime;

        for (int i = 0; i < 4; i++)
        {
            /* Preveri se količina vsakega materiala, če ga primankuje,
            kraljestvo postavi ustrezeno zgradbo */
            if (CheckMaterial(i))
                break;
        }

        // Kraljestvo nahraní meščane
        this.GetComponent<KingdomController>().FeedCitizens();

        // Naknadno se preveri še količina napadalnih v obrambnih vitezov
        CheckMaterial(4);

        /* Treniranje vitezov se izvaja takrat, ko barake obstajajo
        (objekt je shranjen v spremenljivki brc) */
        if (brc != null)
        {
            // Če ima kraljestvo premalo obrambnih vitezov, prične z njihovim treningom
            if (this.GetComponent<KingdomController>().defenceKnights <= 20 && this.GetComponent<KingdomController>().citizens >= 5)
            {
                brc.GetComponent<BarracksController>().isTraining = true;
                brc.GetComponent<BarracksController>().TrainDefence();
            }

            // Če ima kraljestvo premalo napadalnih vitezov, prične z njihovim treningom
            else if (this.GetComponent<KingdomController>().attackKnights <= 20 && this.GetComponent<KingdomController>().citizens >= 5)
            {
                brc.GetComponent<BarracksController>().isTraining = true;
                brc.GetComponent<BarracksController>().TrainAttack();
            }
            else
            {
                // Če je vseh vitezov dovolj, preneha s treningom
                brc.GetComponent<BarracksController>().isTraining = false;
            }
        }
    }
}
```

Slika 5: Koda delovanja neigralskega kraljestva s komentarji, 1. del (Lastni vir)

```
/* Če je status odnosa z igralcem manjši od 0, če je igralec postavil svoje
kraljestvo, če je naključno generirano število glede na odnos enako 1, če
vojska že ne napada in če je napadalnih vitezov več od 0,
kraljestvo napade igralca
*/
if (relationship < 0 && playerKingdom.GetComponent<PlayerKingdomController>().isPlaced == true &&
(int)Mathf.Floor(Random.Range(0, 100+relationship)) == 1 &&
this.GetComponent<KingdomController>().attackArmy.GetComponent<AttackArmy>().isAttacking == false &&
this.GetComponent<KingdomController>().attackKnights > 0)
{
    AttackPlayer();
}
} else
{
    // Če je kraljestvo napadeno in je naključno generirano število enako 1, se brani
    if ((int)Mathf.Floor(Random.Range(0,200)) == 1)
    {
        Defend();
    }
}
```

Slika 6: Koda delovanja neigralskega kraljestva s komentarji, 2. del (Lastni vir)

4.8 Urejenost skript kraljestev

Med programiranjem sem dobil idejo za večjo organiziranost in boljšo preglednost skript igralčevega in drugih kraljestev. Ustvaril sem tri različne skripte, eno za "generično" kraljestvo, eno za igralčovo kraljestvo in eno za neigralčovo kraljestvo. Skripta za "generično" kraljestvo vsebuje vse funkcije in spremenljivke, ki so skupne vsem kraljestvom, skripta za igralčovo kraljestvo vsebuje le funkcionalnosti igralčevega kraljestva, skripta za neigralčovo kraljestvo pa vsebuje kodo za druga kraljestva. Tako imata oba objekta skripto za "generično" kraljestvo in vsak še eno dodatno. Ta modifikacija mi je omogočila mnogo lažje delo v prihodnosti. Podobno sem kasneje storil tudi za igralčeve in druge zgradbe.

4.9 Upravljalec z neigralskimi objekti

Za začetno postavitev in tudi kasnejše postavljanje zgradb sem ustvaril nov objekt, katerega funkcija je prav to. Vsebuje funkciji za postavljanje kraljestva in zgradb. Funkcija za postavljanje kraljestva izbere naključen travnik na igralni plošči in nanj postavi kraljestvo ter mu inicializira nekaj začetnih parametrov. Nato se petkrat kliče funkcija za postavljanje zgradbe. Najprej se naključno generira tip zgradbe, nato pa se poišče ploščica v določeni bližini okoli kraljestva, ki ustreza zgradbi in ni zasedena. Nato se ta zgradba postavi in inicializira se ji nekaj osnovnih parametrov. Če ustrezne ploščice ne najdemo v 500 poskusih, se zgradba ne postavi. Isto funkcijo za postavljanje zgradb sem uporabil tudi v skripti za neigralska kraljestva, ko poskušajo postaviti novo zgradbo.

```
void PlaceKingdom(int id)
{
    float x, z;
    int cnt = 0;

    // Iskanje naključne ploščice, na katero se lahko postavi kraljestvo
    do
    {
        x = Mathf.Floor(Random.Range(0, map.GetComponent<Map>().xSize));
        z = Mathf.Floor(Random.Range(0, map.GetComponent<Map>().xSize));
        cnt++;

        /* Zanko prekinem po 500 ponovitvah, ker ne želim, da bi se zaradi
         * tega upočasnila igră */
        if (cnt >= 500)
            break;

        /* Funkcija getTile(x, z) vrne tip ploščice na koordinatah (x, z).
         * Kraljestvo lahko postavimo le na travniku (tip 1) */
        while (!map.GetComponent<Map>().getTile((int)(x), (int)(z)) != 1);

        if (cnt < 500)
        {
            /* Funkcija buildingPlaced(x, z, id) na mapi označi tip postavljene
             * zgradbe na koordinatah (x, z). S tem preprečim postavljanje več
             * zgradb na eno ploščico */
            map.GetComponent<Map>().buildingPlaced((int)(x),
                (int)(z), -1);

            /* Instantiate<GameObject> ustvari nov objekt iz spremenljivke
             * Kingdom tipa GameObject */
            GameObject tmp = Instantiate<GameObject>(kingdom, new Vector3(x * 10, 2.5f, z * 10), Quaternion.identity);

            // Za lažje delo v drugih skriptah vsakemu kraljestvu dodelim id številko
            tmp.GetComponent<NPCKingdomController>().id = id;

            // Ime kraljestvu priredim glede na id številko
            tmp.name = tmp.name + id;

            // Kraljestvu priredim tudi značko glede na id številko
            tmp.gameObject.tag = ""+id;

            for (int i = 0; i < 5; i++)
            {
                // Vsako kraljestvo začne z največ 5 naključnimi zgradbami
                PlaceBuilding(i, x, z, id, tmp, (int)Mathf.Floor(Random.Range(0, 3)));
            }
        }
    }
}
```

Slika 7: Funkcija za postavljanje neigralskega kraljestva s komentarji (Lastni vir)

```

/* Razlaga argumentov funkcije:
   id = unikatna id številka zgradbe
   kx = x koordinata kraljestva
   kz = z koordinata kraljestva
   kid = id številka kraljestva
   kdm = objekt kraljestva
   bcl = številka, po kateri se generira tip zgradbe
*/
public void PlaceBuilding(int id, float kx, float kz, int kid, GameObject kdm, int bcl)
{
    float x=0, z=0;
    int cnt = 0;

    // Maksimalna oddaljenost zgradbe
    float rng = kdm.GetComponent<KingdomController>().range/10f;
    do
    {
        // Generiranje naključnih koordinat zgradbe
        x = Mathf.Floor(Random.Range(kx-rng, kx+rng));
        if (x < 0)
            x = 1;
        if (x > map.GetComponent<Map>().xSize)
            x = map.GetComponent<Map>().xSize - 1;

        z = Mathf.Floor(Random.Range(kz-rng, kz+rng));
        if (z < 0)
            z = 1;
        if (z > map.GetComponent<Map>().xSize)
            z = map.GetComponent<Map>().xSize - 1;

        cnt++;
        // Kot pri kraljestvu tudi tukaj ustavim zanko, če se ponovi 500-krat
        if (cnt >= 500)
            break;
        // Preverjanje, ali je ploščica na koordinatah ustrezna
    } while (map.GetComponent<Map>().getTile((int)(x), (int)(z)) != bcl);
}

```

Slika 8: Funkcija za postavljanje nove neigralske zgradbe s komentarji, 1. del (Lastni vir)

```

if (cnt < 500) {
    GameObject tmp;
    // Switch stavek za generiranje ustrezne zgradbe
    switch(bcl)
    {
        case 0:
            // Ustvari se nova koča
            tmp = Instantiate<GameObject>(cottage, new Vector3(x * 10, 2.5f, z * 10), Quaternion.identity);
            /* Pokliče se že prej razložena funkcija buildingPlaced(x, z, id),
            enako velja tudi za ostale zgrade */
            map.GetComponent<Map>().buildingPlaced((int)(x), (int)(z), -2);
            break;
        case 1:
            // Ustvari se farma
            tmp = Instantiate<GameObject>(farm, new Vector3(x * 10, 2.5f, z * 10), Quaternion.identity);
            map.GetComponent<Map>().buildingPlaced((int)(x), (int)(z), -1);
            break;
        case 2:
            // Ustvari se rudnik
            tmp = Instantiate<GameObject>(mine, new Vector3(x * 10, 2.5f, z * 10), Quaternion.identity);
            map.GetComponent<Map>().buildingPlaced((int)(x), (int)(z), -1);
            break;
        default:
            /* V primeri, da spremenljivka bcl nekako ne ustreza nobeni
            zgradbi, se generira generična zgrada */
            tmp = Instantiate<GameObject>(genericBuilding, new Vector3(x * 10, 2.5f, z * 10), Quaternion.identity);
            map.GetComponent<Map>().buildingPlaced((int)(x), (int)(z), -1);
            break;
    }

    // Zgradbi se prideli unikatna številka
    tmp.GetComponent<BuildingController>().id = id;

    // Zgradbi se prideli značka, ki je enaka id številki kraljestva
    tmp.gameObject.tag = ""+kid;

    // Zgradbi se prideli starševsko kraljestvo
    tmp.GetComponent<BuildingController>().kingdom = kdm;

    // Zgradbi se prideli ime glede na id kraljestva in id zgradbe
    tmp.name = tmp.name + kid + id;

    /* Spremenljivka ids se uporablja za določanje id številke nove
    zgradbe. Vsakič, ko se le-ta ustvari, se ids poveča. */
    kdm.GetComponent<KingdomController>().ids++;

    // Poveča se maksimalna oddaljenost zgradbe od kraljestva
    kdm.GetComponent<KingdomController>().range += 1f;
}

```

Slika 9: Funkcija za postavljanje nove neigralske zgradbe s komentarji, 2. del (Lastni vir)

4.10 Trgovanje

Trgovanje je bilo časovno zahtevno za izdelavo sploh zaradi povezave vseh gumbov s kodo. Ko želimo trgovati z nekim trgovcom, se odpre meni, v katerem je seznam surovin, ki jih ponuja ta trgovec, seznam surovin, ki jih mi ponujamo, gumbi za dodajanje in odvzemanje surovin, gumb za izmenjavo surovin in gumb za podkupovanje. Trgovec najprej naključno določi surovine, ki jih ponuja glede na količino surovin, ki jih ima. Na podlagi tega določi tudi vrednost surovin, ki jih mi vložimo glede na svoje surovine. Če ima naprimer malo lesa in veliko žita, se mu zdi les več vreden in zato lahko vložimo veliko manj lesa kot žita, da sklenimo isto kupčijo. Trgovcu se ponudba ponastavlja v rednih časovnih intervalih, razen če igralec še zmeraj trguje. Če je trgovec kraljestvo in ima status odnosa manjši od nič, ne moremo trgovati, lahko pa ga podkupimo z zlatom, ki ga poveča. Z višjim statusom odnosa dobimo tudi boljše ponudbe.



Slika 10: Meni za trgovanje (Lastni vir)

4.11 Pristanišče

Pristanišče je posebna zgradba, saj ne proizvaja surovin. Njegova funkcija je čakanje na trgovske ladje, ki naključno pristanejo ob pomolu. Z ladjo lahko trgujemo, vendar nima statusa odnosa, zato imamo vedno enako dobre ponudbe. Zanjo sem uporabil enako kodo, kot za trgovanje s kraljestvi, vendar sem dodal nekaj novih vrstic kode, ki preverjajo, če starševski objekt vsebuje skripto kraljestva ali ladje. S tem sem preprečil napake pri iskanju določenih spremenljivk.

4.12 Napadanje in branjenje kraljestev

Implementacija napadanja in branjenja kraljestev je zahtevala pošten razmislek. Po nekaj različnih konceptih sem se odločil, da bo končna verzija zelo enostavna. Ko je neko kraljestvo napadeno, se nasprotnikova vojska premika od zgradbe do zgradbe in jih uničuje. Če uniči kraljestvo, je napad zaključen. Če napadeni pravočasno reagira, lahko pošlje svojo obrambno vojsko, ki bo poskušala uničiti napadalca.

4.12.1 Napadalna vojska

Za napadalno vojsko sem najprej napisal funkcijo, ki poišče kraljestvo, ki ga napadamo in v tabelo shrani vse njegove zgradbe. Nato sem ustvaril novo funkcijo za določanje vsake naslednje zgradbe. Deluje tako, da najprej iz tabele zgradb izbriše prejšnjo tarčo in nato poišče naslednjo najbližjo zgradbo.

```
void newTarget() {
    // Prejšnja tarča se izbriše iz tabele
    buildings[minB] = null;

    /* Deklarira in inicializira se spremenljivka ok, ki prekini napadanje,
     * če funkcija ne najde nove tarče */
    bool ok = true;

    /* Inicializira se spremenljivka za indeks zgradbe z najmanjšo
     * oddaljenostjo od vojske (minB) in spremenljivka za to razdaljo (minDis) */
    float minDis = -1;
    minB = 0;

    for (int i = 0; i < buildings.Length; i++) {
        if (buildings[i] != null) {

            /* Izračuna se oddaljenost posamezne zgradbe, nato pa se primerja
             * s prejšnjo najmanjšo oddaljenostjo. Tako se poišče zgradba,
             * ki je najbližje vojski */
            if ((ok == true) || minDis >= Mathf.Sqrt((transform.position.x - buildings[i].GetComponent<Transform>().position.x) *
                (transform.position.x - buildings[i].GetComponent<Transform>().position.z) *
                (transform.position.z - buildings[i].GetComponent<Transform>().position.z))) {
                minDis = Mathf.Sqrt((transform.position.x - buildings[i].GetComponent<Transform>().position.x) *
                    (transform.position.x - buildings[i].GetComponent<Transform>().position.z) *
                    (transform.position.z - buildings[i].GetComponent<Transform>().position.z));
                minB = i;
            }
            // Ker je v tabeli vsaj ena zgradba, se ok nastavi na false
            ok = false;
        }
    }

    if (ok == true) {
        /* Če je ok true, kar pomeni, da v tabeli ni več zgradb,
         * se napadanje ustavi*/
        isAttacking = false;
        timer = timerInterval;
    } else {
        // Določi se nova tarča
        target = buildings[minB];

        // Ponastavi se časovnik za napadanje
        timer = timerInterval;
    }
}
```

Slika 11: Funkcija za iskanje nove tarče s komentarji (Lastni vir)

Premikanje vojske sem izvedel s preprostimi pogoji, ki preverjajo lokacijo tarče v primerjavi z vojsko. Ko le-ta pride na lokacijo tarče, ji začne odvzemati življenske točke glede na število vitezov. Ko je življenskih točk enako ali manj kot nič, zgradbo uniči. Takrat kraljestvo napadalne vojske dobi naključno število surovin glede na tip pravkar uničene zgradbe. Pri uničenju kraljestva dobimo večje število vseh materialov. Takrat se vojska začne premikati nazaj proti svojemu starševskemu kraljestvu.

4.12.2 Obrambna vojska

Obrambna vojska deluje precej podobno kot napadalna, vendar je tarča vedno napadalna vojska. Premikanje proti njej in odvzemanje življenskih točk deluje na enak način. Ko igralec klikne na nasprotnikovo napadalno vojsko, se mu ponudi opcija, da jo napade z obrambno vojsko. Če klikne na to opcijo, se bo pojavila in pričela z napadom. Pri neigralskih kraljestvih sem ustvaril pogoj, ki preverja, če ga kdo napada. Če je temu res, ima vsak trenutek naključno možnost, da se začne braniti.

4.13 Pavza

V vsako računalniško igro spada tudi opcija začasne ustavitve. Izdelava le-tega je bilo precej lahko delo. Najprej sem ustvaril meni z gumboma za nadaljevanje igre in vrnitev na naslovni meni. V skripti sem preverjal za klik na tipko "escape". Če se to zgodi, kličem funkcijo za prikazovanje in skrivanje menija, ki hkrati nastavi globalni parameter za hitrost premikanja časa igre na 0 ali 1. 0 pomeni ustavitev vsega premikanja, 1 pa normalno premikanje. Nato sem moral še preprečiti interakcijo z objekti v igri takrat, ko je pavzni meni prikazan. To sem naredil s stavkom, ki iz skripte pavznega menija črpa informacijo, če je pavza aktivirana. Če je ta pogoj resničen, na objekte in drug uporabniški vmesnik ne moremo več klikati dokler pavze ne prekličemo.

4.14 Naslovni meni

Naslovni meni je prva stran, ki se nam prikaže, ko odpremo igro. V svojem primeru sem se odločil, da bo vseboval gume za novo igro, nadaljevanje prejšnje igre in izhod, kasneje pa sem dodal še lestvico rezultatov. Povezava gumbov z njihovimi funkcijami in nalagnjem drugih scen je bilo precej preprosto. Gumboma za novo igro in nalaganje prejšnje igre sem kasneje dodal še nekaj vrstic kode, ko sem implementiral shranjevanje, ki ga bom opisal v nadaljevanju.

Za lepši estetski videz sem dodal še ozadje, ki ga sestavlja nekaj igralnih ploščic, kraljestvo, farma in morje. Kameri sem dodal še animacijo, ki jo počasi premika levo in desno za izgled dinamičnosti.



Slika 12: Naslovni meni (Lastni vir)

4.15 Shranjevanje in nalaganje igre

Shranjevanje in nalaganje shranjene igre je bila ena od najbolj zahtevnih nalog med izdelavo mojega izdelka. Kode je precej veliko in ves čas je bila velika možnost, da se nekje zmotim in se tako podatki ne bodo pravilno shranili. Med implementacijo sem naletel na mnogo težav, vendar sem jih odpravil, čeprav z veliko truda in težavami.

4.15.1 Razred podatkov

Za shranjevanje igre sem najprej moral deklarirati nov razred, ki bi hrani vse potrebne podatke. Najprej sem v tem razredu deklariral strukturo za kraljestvo, ki hrani vse njegove podatke, kot so lokacija, identiteta in število posameznih materialov. Za zgradbe sem deklariral strukturo, ki vsebuje vse potrebne podatke in še eno strukturo, ki vsebuje podatke o zahtevah za nadgradnjo. Za igralno ploščo sem deklariral eno dvodimenzionalno tabelo. Ustvaril sem tudi eno spremenljivko za igralčeve kraljestvo in tabelo za neigralska kraljestva, vsa so tipa strukture kraljestva. Deklariral sem še tabelo za zgradbe.

Ustvaril sem tudi konstruktor razreda. To je funkcija, ki se kliče, ko ustvarimo novo instanco tega razreda. Vhodni podatki konstruktorja so igralna plošča, igralčeve kraljestvo, tabela neigralskih kraljestev in tabela zgradb. Ta funkcija nato prepiše podatke iz vseh vhodnih parametrov v prej deklarirane spremenljivke.

```
[System.Serializable]
public class SaveData
{
    // Struktura kraljestva
    [System.Serializable]
    public struct Kingdom
    {
        // Koordinate
        public float x, y, z;
        // Količina posameznih surovin
        public int wheat;
        public int wood;
        public int stone;
        public int coal;
        public int iron;
        public int gold;
        // Število meščanov
        public int citizens;
        // Število obrambnih in napadalnih vitezov
        public int attackKnights;
        public int defenceKnights;
        // Življenjske točke
        public float health;
        // Id strelcev, nazadnje postavljene zgradbe
        public int ids;
        // Id strelka
        public int id;
        // Odnos z igralčevim kraljestvom
        public int relationship;
    }
}
```

Slika 13: Razred podatkov s komentarji, 1. del (Lastni vir)

```

// Struktura zgradbe
[System.Serializable]
public struct building
{
    // Koordinate
    public float x, y, z;
    // Tip
    public int buildingClass;
    // Id številka
    public int id;
    // Ploščice, na katerih je lahko postavljena
    public int placableOn;
    // Surovine, ki jih proizvaja
    public string[] materials;

    // Struktura zahtev za nadgradnjo
    [System.Serializable]
    public struct upgradeRequirement
    {
        public string material;
        public int amount;
    };
    // Tabela zahtev za nadgradnjo
    public upgradeRequirement[] upgradeRequirements;
    // Čas proizvodnje surovin
    public float productionTime;
    // Življenske točke
    public int health;
    // Stopnja
    public int level;
    // Značka
    public string tag;
}

// Širina in dolžina igralne plošče
public int mapXSize, mapYSize;
// Tabela s podatki o ploščicah mape
public int[,] mapArray;

// Spremenljivka igralčevega kraljestva
public Kingdom playerKingdom;
// Tabela neigralskih kraljestev
public Kingdom[] NPCKingdom;

// Tabela zgradb
public building[] buildings;

// Pretekli "dnevi" igre
public float days;

```

Slika 14: Razred podatkov s komentarji, 2. del (Lastni vir)

4.15.2 Shranjevanje

Funkcija za shranjevanje igre najprej poišče vse podatke, ki jih je potrebno shraniti. Za igralno ploščo in igralčovo kraljestvo tu ni bilo veliko težav, saj v igri obstaja le ena instanca teh dveh objektov. Pri neigralskih kraljestvih in zgradbah je bil to malo večji problem, saj nisem vedel, kako naj v sceni poiščem vse prave objekte. Odločil sem se za malo manj optimizirano metodo, vendar je edina, ki je delovala. Najprej v tabelo shramim vse objekte v sceni. Nato pregledam to tabelo in izbrišem vse objekte, ki v imenu nimajo določene besede, ki je značilna za objekte, ki jih iščem (za neigralska kraljestva naprimer je ta beseda "NPCKingdom"). Potem tabelo normaliziram, da odstranim vse neinicializirane indekse, nato pa jo še sortiram po imenu objektov s pomočjo metode mehurčnega urejanja.

```

// Poisci vse objekte tipa GameObjects
NPCKingdom = FindObjectsOfType<GameObject>();
for (int i = 0; i < NPCKingdom.Length; i++)
{
    /* Preveri, če ima objekta vsebuje niz "NPCKingdoms", ki označuje
     * neigralska kraljestva */
    if (!NPCKingdom[i].gameObject.name.Contains("NPCKingdom"))
    {
        // Iz tabele izbriše vse objekte, ki niso neigralska kraljestva
        NPCKingdom[i] = null;
    }
    else
    {
        // Neigralska kraljestva pomakne na začetek tabele
        NPCKingdom[ind] = NPCKingdom[i];
        ind++;
    }
}

// Prilagodi velikost tabele glede na število preostalih elementov
System.Array.Resize<GameObject>(<ref> NPCKingdom, ind);

// Sortira tabelo glede na imena objektov
for (int i = 0; i < NPCKingdom.Length - 1; i++)
{
    for (int j = 0; j < NPCKingdom.Length - i - 1; j++)
    {
        if (String.Compare(NPCKingdom[j].gameObject.name, NPCKingdom[j + 1].gameObject.name) > 0)
        {
            GameObject tmp = NPCKingdom[j];
            NPCKingdom[j] = NPCKingdom[j + 1];
            NPCKingdom[j + 1] = tmp;
        }
    }
}

```

Slika 15: Koda za iskanje neigralskih kraljestev s komentarji (Lastni vir)

Funkcija nato deklarira in inicializira novo spremenljivko tipa "BinaryFormatter", ki se uporablja za serializacijo pri shranjevanju. Serializacija je pretvorba nekega objekta v bajte. S tem procesom objekt pretvorimo v obliko, ki je primerna za zapisovanje na pomnilnik. Ustvari tudi spremenljivko za pot binarne datoteke, v katero shranjujem. Nato deklarira in inicializira še spremenljivko tipa "FileStream", ki se uporablja za branje, pisanje in zapiranje datotek. Potem ustvari še novo instanco razreda podatkov, ki sem ga že opisal in nato shrani podatke na pomnilnik. Na koncu še zapre tok podatkov.

```

public void SaveGame()
{
    // Funkcija FindData() poišče vse objekte, potrebne za shranjevanje
    FindData();

    // Ustvari se nova spremenljivka tipa BinaryFormatter
    BinaryFormatter formatter = new BinaryFormatter();

    // Določi se pot do datoteke, kamor se shranjujejo podatki
    string path = Application.persistentDataPath + "save.tnkgdms";

    /* Ustvari se nova spremenljivka tipa FileStream. Argumenta konstruktorja
     * sta pot do datoteke in način, na katerega sistem odpre datoteko. V
     * mojem primeru ustvari novo.
     */
    FileStream stream = new FileStream(path, FileMode.Create);

    /* Ustvari se nov objekt podatkov, ki sem ga prej opisal. Argumenti
     * so objekti, ki sem jih poiskal s funkcijo FindData().
     */
    SaveData saveData = new SaveData(map, playerKingdom, NPCKingdom, buildings, gameOver);

    // Podatki se zapisa v ciljno datoteko
    formatter.Serialize(stream, saveData);

    // Tok podatkov se zapre
    stream.Close();
}

```

Slika 16: Funkcija za shranjevanje igre s komentarji (Lastni vir)

Funkcija za shranjevanje podatkov naj bi se poklicala, ko ustavimo igro in se vrnemo na naslovni meni. Povezava gumba v meniju za pavzo in funkcije je bilo lahko delo.

4.15.3 Nalaganje

Funkcija za nalaganje igre je v primerjavi s tisto za shranjevanje relativno preprosta. Najprej preverim, če datoteka obstaja na lokaciji, kamor sem jo shranil. Če je temu tako, se ustvari spremenljivka tipa "BinaryFormatter", spremenljivka tipa "FileStream" in instanca razreda za shranjevanje podatkov. V to instanco se prepišejo shranjeni podatki in funkcija jo vrne. Naredil sem še eno funkcijo katere naloga je, da na podoben način kot pri shranjevanju podatke prepiše, vendar tokrat v že obstoječe objekte. Pri nalaganju zgradb lahko pride do problema, da število zgradb v sceni ni enako številu shranjenih zgradb. Prav zato sem ustvaril novo funkcijo v upravljalcu z neigralskimi objekti, ki sprejme vse potrebne vhodne podatke za novo kreiranje zgradbe.

Na naslovнем meniju sta dva gumba, en za nalaganje prejšnje igre in en za novo igro. Informacijo o pritisnjenu gumbu sem želel prenesti med scenama. S tem namenom sem naredil nov objekt, ki se ne uniči in vedno hranja svoje podatke. V njegovi skripti sem deklariral spremenljivko tipa "bool", ki določa, ali smo kliknili prvi ali drugi gumb. Če izberemo novo igro, je njena vrednost 1, v nasprotnem primeru pa 0. V sceni igre nato preverim njeno vrednost in nato kličem ustrezeno funkcijo.

4.16 Konec igre

Seveda mora obstajati tudi način, kako končamo igro. V mojem primeru obstajajo trije različni izidi. Ali sovražniki pokončajo igralčeve kraljestvo, ali igralec uniči vsa neigralska kraljestva, ali pa vsi meščani v igralčevem kraljestvu umrejo od lakote.

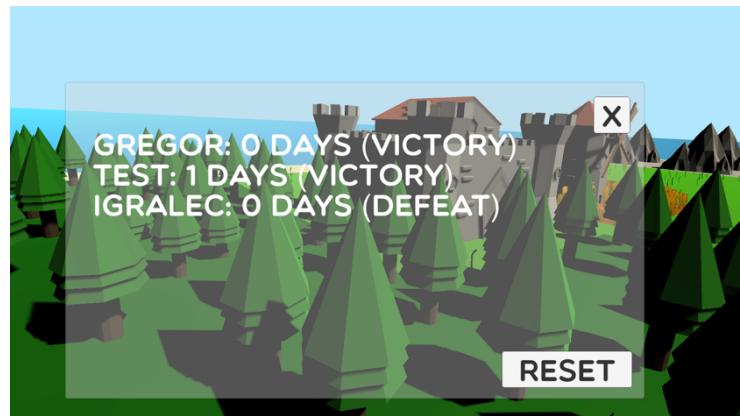
Najprej sem naredil nov objekt, v katerem se nahajajo trije različni meniji za vse načine konca igre. Vsebujejo besedilo in gumb za vračanje na naslovni meni. Ustvaril sem tudi novo skripto za ta objekt. Na začetku poišče vsa neigralska kraljestva na enak način, kot pri shranjevanju. Nato sproti med igro preverja, če je dolžina te tabele enaka nič. V tem primeru prikaže zmagovalni meni. Zapisuje si tudi vsoto igralčevih meščanov, napadalnih vitezov in obrambnih vitezov. V skripti za igralčovo kraljestvo sem dodal še kodo, ki ga uniči, če je ta vsota enaka nič ali manj. V skripti za konec igre sproti preverjam tudi, če igralčovo kraljestvo še zmeraj obstaja. Če ne obstaja, preverim ali je vsota manjša ali enaka nič in na podlagi tega prikažem ustrezni meni.

4.17 Lestvica rezultatov

Sredi izdelave sem dobil idejo, da bi implementiral lestvico rezultatov. Odločil sem se, da bom shranil čas, v katerem se je končala igra in ali je igralec zmagal ali izgubil. Najprej sem predelal meni za konec igre. Do sedaj sem imel tri različne menije, a sem zdaj obdržal le enega in spreminal besedilo glede na izid igre. Dodal sem mu še polje za vnos imena igralca. V skripti za konec igre sem dodal kodo, ki onemogoči shranitev rezultata in vračanje na naslovni meni, če je vnosno polje prazno. Nato sem v isti skripti ustvaril še števec in spremenljivko za število dni, ki ta števec spremeni v celo število.

Ustvaril sem nov razred za hranjenje rezultatov. Vsebuje spremenljivke za ime igralca, število dni in status (zmaga ali poraz). Ima še konstruktor, ki tem spremenljivkam nastavi vrednosti vhodnih podatkov. Funkcija za shranjevanje rezultatov v binarno datoteko je skoraj enaka tisti za shranjevanje igre.

Na naslovнем meniju sem ustvaril gumb, ki prikaže seznam rezultatov. Ob tem se kliče funkcija, ki najprej iz datoteke prebere vse rezultate in jih shrani v tabelo. Nato to tabelo sortira, najprej po statusu (zmagovalci imajo prednost pred poraženci), nato pa po času, s tem da je pri zmagovalcih pomemben čim nižji čas, pri poražencih pa čim višji. Vse podatke nato prikažem na tekstovnem objektu. Dodal sem še en gumb, ki ponastavi lestvico rezultatov. To naredi tako, da izbriše datoteko za hranjenje rezultatov, nato pa izbriše še trenutne vrednosti iz tabele.

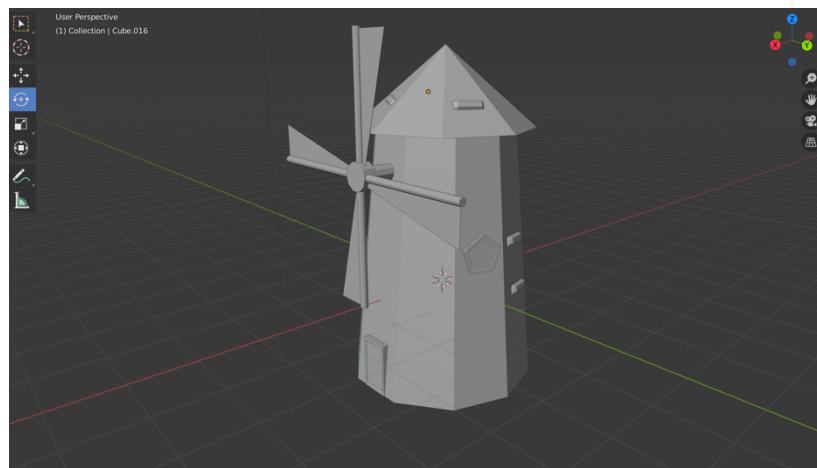


Slika 17: Lestvica rezultatov (Lastni vir)

4.18 Grafična podoba

Ko je večina moje igre že funkcionalna, sem se lotil izdelave grafične podobe. Uporabil sem orodje za 3D oblikovanje Blender in orodje za risanje GIMP. Oblikovanje prvega 3D modela je bil zame največji izziv, saj že dolgo časa nisem tega delal in sem se moral veliko stvari ponovno naučiti, pri vseh naslednjih pa je bilo delo veliko lažje in hitrejše. V Blenderju sem večino modelov sestavil iz valjev, ki imajo osnovno ploskev z 10 ali manj stranicami. V nadaljevanju sem jih naknadno oblikoval s pomočjo različnih orodij. "Extrude Region" iz poljubne ploskev potegne njeno kopijo in ji doda stranske ploskev. "Inset Faces" podvoji ploskev in kopijo pomanjša, ter njena oglišča poveže z oglišči prvotne ploskev. "Loop Cut" naredi novo stranico z začetkom v izbrani točki, ki leži na sredini neke stranice, vzdolž celotne površine modela. Po končanem oblikovanju, sem vsak model posebej uvozil v Unity.

V Unityju sem najprej za vsak model posebej nastavil pravilno velikost. Nato sem naredil nekaj tekstur kar v urejevalniku. Vse so enobarvne in večini sem odstranil sijaj, razen tisti, ki predstavlja kovino. Nato sem s pomočjo teh tekstur pobarval svoje modele, jih shranil in dodal v objekte namesto prejšnjega testnega modela, ki je bil le kocka.



Slika 18: Model farme v Blenderju (Lastni vir)

4.19 Zvočni učinki

Odločil sem se, da bom svojo igro popestril z nekaj preprostimi zvočnimi učinki. Naredil sem jih v programu GarageBand in jih uvozil v Unity. Ustvaril sem nov objekt, katerega namen je predvajanje zvokov. Dodal sem mu skripto, ki vsebuje spremenljivke za različne zvočne posnetke, nato pa sem napisal funkcijo, ki sprejme številko zvoka in ga predvaja. To funkcijo nato klicem iz drugih skript.

4.20 Manjši dodatki, popravki in razhroščevanje

Na tej točki je velika večina moje igre že funkcionirala, vendar še ni bila povsem pripravljena za oddajo. Želel sem dodati še nekaj dodatkov, ki bi predvsem vizuelno izboljšali izgled igre in jo tudi naredili navidezno bolj tekočo.

4.20.1 Pisava in meniji

Za takojšnjo nadgradnjo izgleda igre sem implementiral novo pisavo in lepše menije. Pisavo sem naložil preko spletne strani DaFont. Uporabil sem zastonjsko pisavo Multicolore.

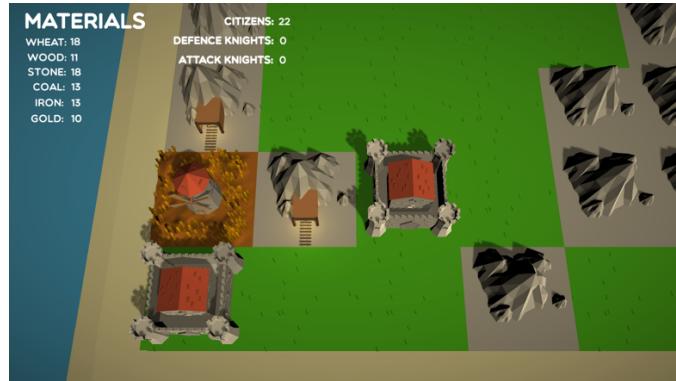
Prej so se vsi meniji odpirali v tridimenzionalnem prostoru zraven zgradb, kar ni izgledalo najlepše, poleg tega pa mi je povzročilo nekaj težav pri izdelavi. Tako sem jih premaknil v dvodimenzionalen prostor na fiksno mesto v oknu. Taki meniji izgledajo veliko lepše in bolj urejeno, hkrati pa zasedejo manj prostora in ne ovirajo igralca.

4.20.2 Prikaz statusa surovin

Do sedaj so se statusi surovin v meniju sproti spremnjali med igro, vendar so bile te spremembe težko opazne. Tako sem v skripto za igralčeve kraljestvo dodal kodo, ki preverja, ali so se statusi surovin, meščanov in vitezov spremenili in ali je bila ta sprememba pozitivna ali negativna. Če je bila pozitivna, bo številka za prikaz za trenutek spremenila svojo barvo na zeleno, če je bila negativna, bo pa svojo barvo spremenila na rdečo. Dodal sem še hiter prehod med belo in zeleno oziroma rdečo barvo za lepši videz. Tako je igralec veliko bolj jasno obveščen o stanju njegovih surovin.

4.20.3 Sprememba perspektive

Sredi razmisleka o morebitnih izboljšavah za moj izdelek sem se odločil, da bom spremenil perspektivo. Do sedaj je bil pogled na igro pod kotom 70° . Ko sem eksperimentaliral s kamero, sem ugotovil, da vsi modeli veliko lepše zgledajo iz izometrične perspektive. Tako sem kamero zavrtel za 45° po oseh X in Y, ter izbral ortografsko projekcijo v Unityjevem meniju.



Slika 19: Prva različica perspektive (Lastni vir)



Slika 20: Končna različica perspektive (Lastni vir)

4.20.4 Indikator izgube življenskih točk

Za bolj jasen potek napadanja in branjenja sem se odločil, da bom naredil indikator, ki bo prikazoval izgubo življenskih točk. Ugotovil sem, da bo najbolje, če zgradba utripa rdeče, ko je napadena. To sem izvedel tako, da sem podvojil vsak model in ga pobarval s posebno rdečo teksturo. Nato sem dodal kodo v skripte zgradb, ki preverja, če so se življenske točke zmanjšale. Če je temu tako, se prikaže rdeče pobarvan model za eno sekundo. Za lepši prikaz sem napisal še nekaj vrstic, ki prosojnost rdeče tekture višajo in nižajo.

4.20.5 Popravek postavljanja zgradb

Ena od večjih težav na katero sem naletel je bila v povezavi s postavljanjem zgradb in shranjevanjem. Pogosto se je dogajalo, da so druga kraljestva postavljala zgradbe na že zasedena mesta. Enaka težava se je pojavljala, ko je igralec poskušal postaviti zgradbo. Občasno se je zgodilo tudi, da so se pri nalaganju shranjene igre pod zgradbami pojavile ploščice napačnega tipa. Kar nekaj časa sem porabil, da sem odkril vzrok teh težav, a sem le ugotovil, da se to dogaja, ker se pri postavljanju igralčevega kraljestva in zgradb števila narobe zaokrožijo. Prej sem zaokroževal navzdol s funkcijo "Math.Floor", nato pa sem namesto nje uporabil funkcijo "Math.Round", ki število lahko zaokroži navzdol ali navzgor.

4.20.6 Prikaz potrebnih surovin

Želet sem dodati meni, v katerem se prikazujejo potrebne surovine za grajenje zgradb in njihovo nadgradnjo. Ustvaril sem nov objekt, ki vsebuje novo skripto in sam meni z dvema tekstovnima elementoma. Prvi naj bi vseboval surovine, ki jih imamo, drugi pa surovine, ki jih še nimamo. Nato sem vsem gumbom, s katerimi trošimo surovine dodal nov sprožilec dogodkov. Le-ta pokliče funkcijo za prikaz menija, ko gremo z miško čez gumb ali ko kliknemo nanj in funkcijo za skrivanje menija, ko z miško izstopimo iz gumba. V funkcijo za prikaz menija pošljem številko, s katero nato določim, katere surovine prikazati. Ustvarim še dve tekstovni spremenljivki in ju napolnim s podatki. Nato njuno vsebino kopiram v prej ustvarjena tekstovna elementa in prikažem meni. Poleg tega sproti berem še lokacijo miške na ekranu in meni premikam skupaj z njo. Z dodatkom prikaza potrebnih surovin igra zgleda veliko bolj uporabniku prijazna.

4.20.7 Tekoče premikanje zgradb

Do sedaj so se vse zgradbe med postavljanjem premikale z zelo velikim korakom, kar ni zgledalo najlepše. Želet sem, da bi potekalo bolj tekoče. V funkciji za postavljanje zgradb sem dodal spremenljivko za hranjenje nove lokacije. Sedaj se lokacija miške hrani vanjo. Nato se zgradba z manjšimi koraki premika proti temu položaju, dokler ga ne doseže. Dodal sem še eno spremenljivko, ki shrani prejšno lokacijo. Tako se zgradba premika proti njej, če se nova lokacija ne prebere (v primeri, da miška ni na ustreznih ploščicah). Zdaj postavljanje zgradb izgleda veliko bolj tekoče.

5. Zaključek

S svojim izdelkom sem zelo zadovoljen. Vanj sem vložil zelo veliko truda in tako je temu primeren tudi končni rezultat. Ena izmed edinih pomankljivosti, ki sem jih opazil, je optimizacija. Trenutno igra vsebuje nekaj blokov kode, ki bi na slabših napravah najverjetneje upočasnili delovanje. Razen tega pa vse ostalo dela, tako kot mora. Med izdelavo sem se veliko naučil, sploh o bolj naprednem delu s programom za izdelavo iger in o tridimenzionalnem oblikovanju. Menim, da mi je celoten proces zelo koristil, saj bom pridobljeno znanje vsekakor lahko uporabil tudi v prihodnosti.

6. Viri in literatura

- *Unity (Game Engine)*. 2020. Wikipedia. Dostopen prek: [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) (3. 3. 2020).
- *Microsoft Visual Studio*. 2020. Wikipedia. Dostopen prek: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio (3. 3. 2020).
- *Unity Store*. 2020. Unity. Dostopen prek: <https://store.unity.com/> (3. 3. 2020).
- *Blender (software)*. 2020. Wikipedia. Dostopen prek: [https://en.wikipedia.org/wiki/Blender_\(software\)](https://en.wikipedia.org/wiki/Blender_(software)) (3. 3. 2020).
- *GIMP*. 2020. Wikipedia. Dostopen prek: <https://en.wikipedia.org/wiki/GIMP> (5. 3. 2020).
- *Unity User Manual (2019.3)*. 2020. Unity Documentation. Dostopen prek: <https://docs.unity3d.com/Manual/> (29. 3. 2020).
- *Stack Overflow*. 2020. Stack Overflow. Dostopen prek: <https://stackoverflow.com/> (17. 3. 2020).
- *Unity Answers*. 2020. Unity. Dostopen prek: <https://answers.unity.com> (17. 3. 2020).
- *Unity Core Platform*. 2020. Unity. Dostopen prek: <https://unity.com/products/core-platform> (29. 3. 2020).
- Brackeys. 2017. *How to make RTS Camera Movement in Unit*. YouTube. Dostopen prek: <https://www.youtube.com/watch?v=cfjLQrMGEb4&t=616s> (31. 10. 2019).
- Neogrey Creative. 2013. *Multicolore*. dafont.com. Dostopen prek: <https://www.dafont.com/multicolore.font?l%5B%5D=1> (7. 3. 2020).
- *GarageBand*. 2020. Wikipedia. Dostopen prek: <https://en.wikipedia.org/wiki/GarageBand> (9. 4. 2020).
- Beaver Joe. 2019. *SAVE and LOAD DATA with Serialization - BINARY FORMATTER in Unity*. YouTube. Dostopen prek: <https://www.youtube.com/watch?v=-bWpH8iXI8> (9. 4. 2020).

Izjava o avtorstvu

Izjavljam, da je seminarska naloga Tiny Kingdoms v celoti moje avtorsko delo, ki sem ga izdelal samostojno s pomočjo navedene literature in pod vodstvom mentorja.

14. 4. 2020

Gregor Kovač