B2 d.o.o.

VIŠJA STROKOVNA ŠOLA

DIPLOMSKO DELO

Maks Jurkas

B2 d.o.o.

VIŠJA STROKOVNA ŠOLA

DIPLOMSKO DELO

Igra “Cryptic Dungeons”

Ljubljana, maj 2022 **Maks Jurkas**

IZJAVA O AVTORSTVU

Spodaj podpisan Maks Jurkas, študent B2 Višje strokovne šole v Ljubljani, izjavljam, da sem avtor diplomskega dela z naslovom Igra "Cryptic Dungeons", pripravljenega v sodelovanju z mentorjem Aleksandarjem Lazarevićem.

Izrecno izjavljam, da v skladu z določili Zakona o avtorskih in sorodnih pravicah (Ur. l. RS, št. 21/1995 s spremembami) dovolim objavo diplomskega dela na spletnih straneh šole ter objavo bibliografskih podatkov z abstraktom v sistemu Cobiss.

S svojim podpisom zagotavljam, da

- sta predloženi tiskana in elektronska verzija besedila diplomskega dela istovetni;
- je predloženo besedilo rezultat izključno mojega lastnega raziskovalnega dela;
- je predloženo besedilo jezikovno korektno (lektorirano) in tehnično pripravljeno v skladu z Navodili za pisanje diplomskih del B2 Višje strokovne šole, kar pomeni, da sem

  - poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam v diplomskem delu, citirana oziroma navedena v skladu z Navodili za pisanje diplomskih del B2 Višje strokovne šole, in
  - pridobil vsa dovoljenja za uporabo avtorskih del, ki so v celoti (v pisni ali grafični obliki) uporabljena v tekstu, in sem to v besedilu tudi jasno zapisal;

- se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku (Ur. l. RS, št. 55/2008 s spremembami).

V Ljubljani, dne 23.05.2022 Podpis avtorja:_____

KAZALO

KAZALO SLIK

POVZETEK

Predmet diplomske naloge je izdelava RPG (Role-Playing Game) igre z uporabo programskega jezika C# in Microsoft-ovega framework-a Windows Forms, ter Newtonsoft JSON framework-a za shranjevanje.

V šolskem letu 2020/21 smo pri predmetu Programiranje 2 delali s programskim jezikom C#, in sicer vizualno programiranje preko Windows Forms-ov. Pri odločanju za seminarsko nalogo sem se odločil izdelati nekaj svojega in sicer igro, kjer se spopadaš z raznimi nasprotniki, tako da lahko ustvariš njihov oklep in orožje ter s tem postaneš močnejši. Igralec začne z začetno opremo, kjer se mora spopadati z močnejšimi nasprotniki, da pridobi kovance in točke izkušenj in si s tem izboljša možnosti zmage nad končnim nasprotnikom prvega nivoja. Po prvi zmagi nad njim igralec pridobi material, katerega lahko po ponovnih spopadih s končnim nasprotnikom uporabi za izdelavo njegovega orožja in oklepa.

Igralec se nato počasi prebija skozi nivoje, nadgrajuje svojo opremo in zvišuje svojo moč in obrambo, dokler ne pride do zadnjega nivoja, kjer se lahko spopade s končnim nasprotnikom igre in odklene način igre, kjer se prikažejo vsi nasprotniki iz prejšnih nivojev. Ko izdela še zadnji meč in oklep, se igralcu odklene »Hard Mode« oz. težki način, kjer postanejo nasprotniki močnejši in igralcu dajo nove materiale, ki jih mora uporabiti za izdelavo končnega orožja in oklepa.

Pri tem projektu sem se odločil, da bom nadaljeval razvoj iger in poskušal narediti nekaj, kar bi pritegnilo pozornost publike in mi dalo veselje izdelave ter nadaljnjih projektov.

Moram priznati, da je bil projekt bolj uspešen na določenih področjih in rahlo nezadovoljiv na drugih, vendar bi ga z veseljem ponovil.

KLUČNE BESEDE:

Windows Forms, C# programski jezik, Role-Playing igra, Newtonsoft JSON, oprema, igra, programiranje

1 UVOD

1.1 OPREDELITEV TEME IN RAZLOGI ZA IZBIRO

Odločil sem se za izdelavo “Role-Playing” igre, ki bo temeljila na igralčevi opremi in delovala na princip “boljša oprema = močnejši igralec”.

Igro sem izdelal s pomočjo programa Visual Basic, v C# programskem jeziku, s pomočjo Windows Form-ov. Visual Basic je eden izmed najbolj priljubljenih programov za programiranje (vsaj za Microsoftove programske jezike. C# je relativno popularen programski jezik, ki se uporablja za več popolnoma različnih stvari, z močno podporo razvijanju iger za okolja Windows, poleg tega se je pa v zadnjih letih nabrala podpora za razvijanje iger za pametne telefone. Windows Form-i so v trenutnem stanju že precej stari in se uporabljajo vse manj, ampak so kljub temu še zmeraj precej uporabni in priljubljeni, saj so dokaj preprosti za učenje.

Za izdelavo igre sem se odločil, ker sem že dolgo želel pristopiti k izdelavi in razvoju iger, vendar nisem vedel, kje začeti. Ko smo pri predmetu Programiranje 2 delali z Windows Form-i sem si zamislil igro, ki bi jo lahko izdelal za seminarsko nalogo; po pogovoru s profesorjem pa sem se odločil to igro razširiti v diplomsko nalogo.

Že dolga leta sem oboževalec RPG (Role-Playing Game) žanra, kot so na primer: World of Warcraft, Monster Hunter, Slay The Spire, in podobne, a glede na to, da nimam nobenih izkušenj iz razivanja iger, sem se odločil za nekaj bolj preprostega, kjer bi iz teh iger črpal le osnovne lastnosti (izdelovanje opreme z materiali, ki jih igralec dobi od nasprotnikov, nakupovanje opreme, nadgradnja opreme), brez 3D grafike in sveta, po katerem bi se igralec premikal in spopadal z nasprotniki.

Zelo me navdušujejo igre, kjer se igralec premika po raznih različnih nivojih, z različnimi nasprotniki, vse z namenom da bi nadgradil svojo opremo in šel proti težjim in močnejšim nasprotnikom.

1.2 PREDSTAVITEV PROBLEMOV

Problemi, na katere sem naletel pri pisanju programa, so se nekoliko razlikovali od problemov, ki sem si jih zastavil pred začetkom programiranja.

Pričakoval sem, da bom naletel na probleme pri uporabljanju različnih lastnosti opreme skozi različne nivoje igre in pri shranjevanju podatkov pridobljenih po spopadu z nasprotnikom.

Predvideval sem, da bom naletel na številne probleme glede optimiziranja napisane kode, saj je bilo moje prvotno razmišljanje o uporabi opreme popolnoma drugačno in neoptimizrano glede na to, kar sem kasneje ugotovil in kako sem stvari na koncu zapisal.

Poleg ostalih problemov me je skrbelo, kako bom shranjeval podatke, ki bi jih kasneje lahko klical tudi iz drugega sistema. Moral sem se odločiti med serializacijo podatkov in shranjevanjem podatkov v podatkovno bazo (tu bi se moral odločiti še za ustrezno bazo podatkov – npr. MySQL, SQL, SQLite, itd.).

Manjši problemi, ki sem jih pričakoval, so bili uporaba posebnih efektov določenih nasprotnikov in določene igralčeve opreme, prikaz likov (igralec, orožje, oklep, nasprotniki, ozadja), odklepanje in zaklepanje prisvojene opreme in izvajanje spopada med igralcem in nasprotnikom.

1.3 CILJI PROJEKTA

Na začetku projekta sem si zastavil cilj, da ustvarim aplikacijo oz. igro, ki bi imela nekatere principe svetovno znanih RPG (Role-Playing Game) iger.

Želel sem uporabiti številne lastnosti iz raznih iger, ki sem jih igral in jih združiti v eno igro, a brez 3D grafik, premikanja po svetu, animacij, ipd. Prvotno sem nameraval implementirati pridobivanje opreme na način igre World of Warcraft, kjer igralec pridobi opremo, ko premaga končnega nasprotnika nivoja, ampak sem se kasneje odločil za drugačen pristop k pridobivanju opreme, in sicer sem uproabil metodo igre Monster Hunter, kjer igralec po zmagi nad končnim nasprotnikom nivoja pridobi materiale za izdelavo orožja in oklepa od njega. Premikanje skozi nivo, premagovanje nasprotnikov in premikanje do končnega nasprotnika sem si zamislil na način igre Slay the Spire, kjer ima igralec več možnosti pristopa h končnemu nasprotniku (bojevanje, preskakovanje boja…), tako da ima vsaka možnost svoje prednosti in slabosti.

Za like sem se odločil, da bodo 8-bitne sličice, ki bi jih sam izdelal.

1.4 UPORABLJENA SREDSTVA

Za izdelovanje likov sem uporabil program Adobe Illustrator, kjer sem s pomočjo mreže izdelal vse like, orožja, oklepe, gumbe in ozadja.

Za pisanje programa sem uporabil program Visual Basic in programski jezik C#.

V Visual Basic-u sem uporabil Windows Form-e za prikaz programa in Newtonsoft JSON framework za serializacijo oz. shranjevanje ter nalaganje igre.

1.5 REŠEVANJE GLAVNIH PROBLEMOV

1.5.1 PRENAŠANJE PODATKOV

Problemi pri tej nalogi so bili številni, največji problem je bil, ugotoviti kako prenašati podatke o igralcu in nasprotnikih iz ene Windows Form-e v drugo, saj se nova forma odpre s klicem »New«, ki kreira novo instanco Windows Form-a, kar pomeni, da so vsi podatki ponastavljeni., Ta problem sem odrešil s pomočjo kazalcev oz. referenc, tako da sem vse igralčeve podatke shranil v poseben dokument, iz katerega sem jih nato klical, poleg tega sem pa iz ene forme v drugo pošiljal določene podatke (npr. izbrani oklep in orožje) in sem tako odpravil problem shranjevanja oz.ne-shranjevanja podatkov, tako sem po nekaj dneh ugotavljanja lahko nadaljeval izdelavo svojega programa.

1.5.2 OPTIMIZACIJA

Ključni problem pri večini programov je optimizacija, z drugimi besedami: kako bi program zapisal s čim manjšim številom vrstic in ponavljanja, da bi delal hitreje in bolje kot prej.

Pri optimizaciji je bila uporaba kazalcev ključna, saj bi bil moj progam brez njih dolg več tisoč vrstic kode. Tako pa je ponavljanje kode minimalno in uporabljeno, le ko je to absolutno potrebno.

Kazalci delujejo tako, da v eni datoteki oz. eni formi kličemo podatke iz druge datoteke. V mojem primeru so bili vsi podatki shranjeni v eni datoteki (Data.cs) in vse forme črpajo podatke iz te datoteke.V objektu oz. class-u Podatki so shranjeni vsi podatki o igralcu, poleg tega pa so tu shranjeni tudi vsi podatki o orožjih, oklepih in zalogi materialov, ki jih je igralec pridobil pri igranju.

1.5.3 SHRANJEVANJE IN NALAGANJE IGRE

Poleg problema prenašanja podatkov, ko igralec igra, je bil eden od večjih problemov še shranjevanje igre, tako da bi po prekinitvi lahko igralec nadaljeval svojo prejšnjo sejo in se mu ne bi vsi podatki in pridobljena oprema zbrisala programa.

Tu sem se moral odločiti za način shranjevanja in nalaganja igre. Izbiral sem med serializacijo (shranjevanje objektov v neko datoteko) in uporabo podatkovne baze (MySQL, SQL, SQLite).

Najprej sem poskusil z uporabo podatkovne baze in se odločil za MySQL, saj sem bil s to bazo že seznanjen. Po številnih urah ugotavljanja, kako bi svoj progam povezal z bazo podatkov in iz baze bral oz. spreminjal vrednosti, se mi je celoten program obrnil na glavo in sem porabil velio časa, da sem vse nazaj zgradil v stanje pred izgubo podatkov. Naslednji dan sem ponovno poskusil vzpostaviti povezavo in uporabo baze, tokrat mi je uspelo, a se je pojavil nov problem; veliko kode in malo optimizacije, kar bi pomenilo, da bi bil program veliko bolj počasen pri dodajanju elementov in objektov v igro.

Ko sem uspešno vzpostavil povezavo s podatkovno bazo, sem to idejo zavrgel in se odločil za uporabo serializacije.

Serializacija je bila dokaj enostavna, saj je tehnologija pogosto uporabljena in dobro dokumentirana, kar je zelo poenostavilo problem shranjevanja.

Odločil sem se za Newtonsoft JSON serializacijo, kar shranjuje objekt Podatki v dokument s končnico .json.

Newtonsoft JSON je že precej razširjen in uporabljen »framework«, z leti razvijanja in optimiziranja in z dobro opisano dokumentacijo je njegova uporaba relativno enostavna.

2 DELOVANJE PROGRAMA

2.1 ZAČETNI ZASLON

Ob zagonu programa se nam prikaže začetni zaslon, kjer lahko uporabnik izbira med »Start«, »Load Game« in »Exit«.

Slika 1: Začetni zaslon

Če se uporabnik odloči začeti novo igro, pritisne na »Start«, kar bo programu povedalo, da se uporabijo privzeti podatki in nastavitve.

Če je uporabnik igro zagnal že kdaj prej in je shranil trenutno situacijo, lahko izbere opcijo »Load Game«, pri čemer se odpre pogovorno okno, kjer uporabnik poišče kreirano datoteko s končnico .json.

Ko uporabnik izbere želeno datoteko, se mu zažene igra z vsemi predhodno pridobljenimi materiali in napredki v igri.

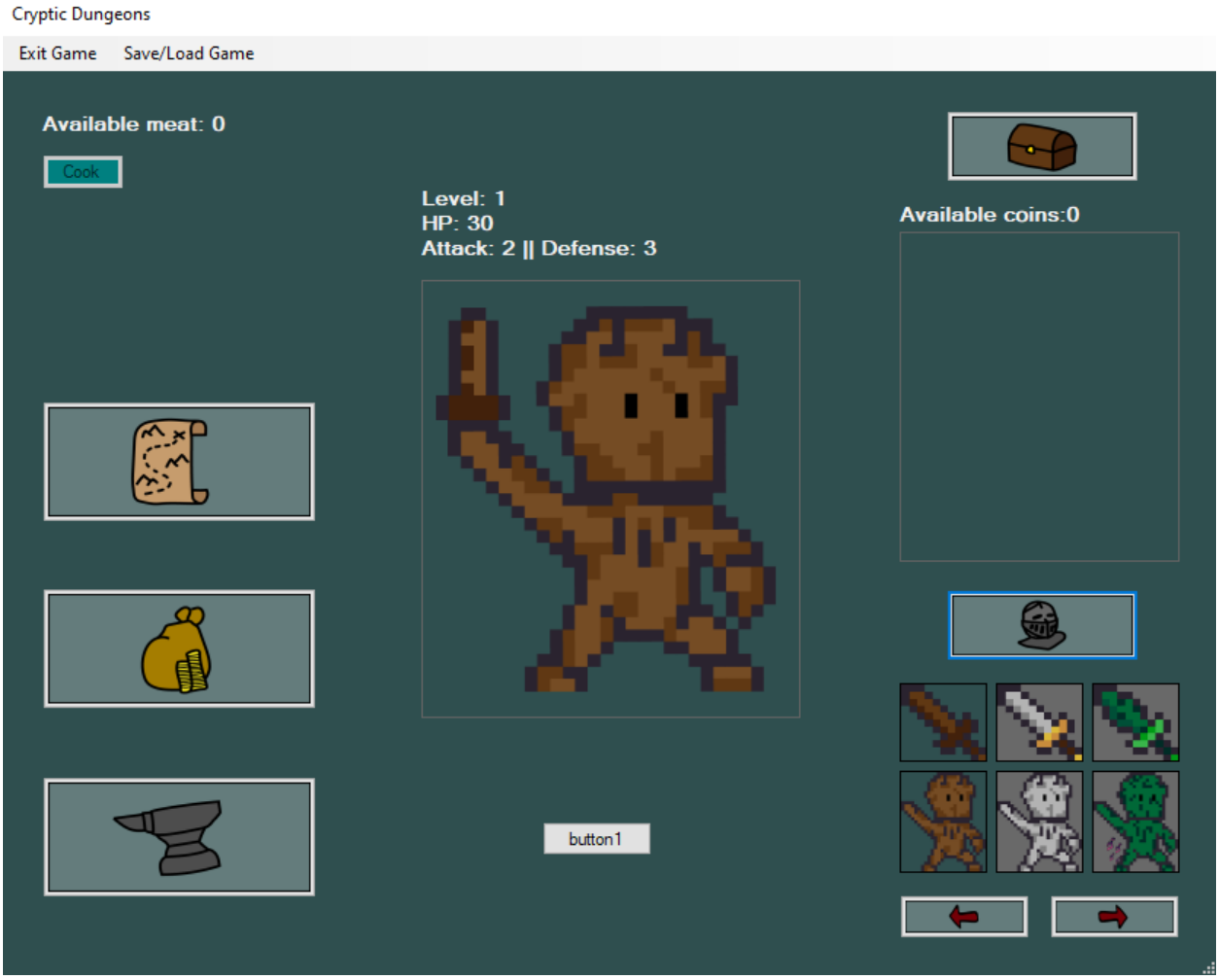
Tretja možnost na začetnem zaslonu je še »Exit«, ki aplikacijo, oz. program, zapre.

```
Home h = new Home(weapon, armor, ref p);
h.Width = this.Width;
h.Height = this.Height;
h.StartPosition = FormStartPosition.Manual;
h.Location = new Point(this.Location.X, this.Location.Y);
this.Visible = false;
h.ShowDialog();
```

Zgornja programska koda prikazuje odpiranje novega okna; tak zapis se ponavlja skozi celoten program, ko se odpira novo okno (prehod iz enega okna v drugega).

Zapis je namenjen temu, da se novo okno odpre na mestu trenutnega okna, in sicer z isto velikostjo.

2.2 DOMAČI ZASLON



Slika 2: Domači zaslon

Na domačem zaslonu vidimo na sredini igralca opremljenega s trenutno izbrano opremo, nad njim so podatki o igralčevih atributih (nivo igralca, življenjske točke, moč in obramba igralca).

Na levi strani vidimo gumbe za zemljevid, trgovino in kovača.

Na desni strani je prikazana še vsebina igralčeve torbe in kosi opreme, ki imajo sivo ozadje dokler jih igralec ne pridobi.

Na vrhu okna vidimo »Exit Game« in »Save/Load Game« ki se uporabljata za izhod iz igre in shranjevanje oz. nalaganje prejšnje seje igre.

Zanimivost pri kodi za to okno je v prikazovanju in skrivanju igralčeve opreme.

```
public void hideAvailableEquipment()
        {
            foreach (Control x in this.Controls)
            {
                if (x is Button && x.Visible == true)
                {
                    if ((string)x.Tag == "Equipment" || (string)x.Tag == "Equipment1" ||
                    (string)x.Tag == "Equipment2")
                    {
                        x.Visible = false;
                    }
                }
            }
        }
```

Ko uporabnik pritisne na gumb za opremo (železna čelada na desni strani zaslona), se izvede zgornja zanka, pri kateri gre program skozi vse kontrolne komponente programa in preveri, če je komponenta gumb in če je gumb trenutno skrit.

Nato še pregleda, če ima gumb značko »Equipment«. To značko imajo vse komponente, ki se uporabijo za prikaz opreme. Značka gumb skrije oziroma prikaže, če trenutno ni viden.

Domači zaslon je tudi edini zaslon, iz katerega lahko uporabnik shranjuje, nalaga in zapira igro.

- Shranjevanje igre:

```
p.saveName = Path.GetFullPath(saveFileDialog1.FileName);
File.WriteAllText(p.saveName, JsonConvert.SerializeObject(p));
```

Kot sem omenil že prej, sem za shranjevanje uporabil JSON serializacijo, kar pomeni, da se vsi želeni podatki shranijo v datoteko s končnico .json.

- Nalaganje igre:

```
p.saveName = Path.GetFullPath(openFileDialog1.FileName);
             JsonConvert.PopulateObject(File.ReadAllText(p.saveName), this.p);
```

Prvi stavek je podoben stavku za shranjevanje igre, njegova funkcionalnost je shranjevanje poti datoteke, da lahko kasneje uporabnik shranjuje v isto datoteko brez odpiranja dodatnega pogovornega okna.

Drugi stavek pretvori .json datoteko v podatke, ki zamenjajo trenutne uporabljene podatke.

2.3 IGRALČEVI PODATKI

Vse podatke, ki se uporabljajo skozi igro, sem zapisal v ločeno datoteko, iz katere jih program nato bere in kamor jih tudi zapisuje, ko se spremenijo.

Razred sem poimenoval »Podatki« in vsaka forma oz. datoteka ima na začetku zapisan kazalec, ki programu pove, od kod naj črpa podatke.

Inicializacija forme:

```
public Home(Weapons selectedWeapon, Armor selectedArmor, ref Podatki p)
{
      InitializeComponent();
      this.p = p;
```

- »Home« je domači zaslon, v oklepaju so pa zapisani podatki, ki se prenašajo v to formo iz prejšnjega odprtega okna.
- Zadnji del oklepaja (ref Podatki p) je kazalec na igralčeve podatke, zapisane v drugi datoteki.
- »InitializeComponent();« nakazuje zagon trenutnega okna in vseh njegovih komponent.
- »this.p = p;« pa pove programu nakaže spremenljivko, ki se uporablja za črpanje podatkov

Vsi podatki o orožjih, oklepih in nasprotnikih so zasebni, da jih igralec ne mora spreminjati in si igre prilagajati po svojih željah.

```
public class WoodenSword : Weapons
    {
        public WoodenSword()
        {
            weaponName = "Wooden Sword";
            weaponDmgMax = 3;
            weaponDmgMin = 2;
            isUnlocked = true;
            weaponImage = Properties.Resources.WoodenSword_Equip;
        }
    }
```

Zgornji zapis kode prikazuje javne lastnosti določenega objekta v igri.

Podatki o objektu so pa v bazi zapisani kot zasebni vnosi, katerim sem dodal javne lastnosti, ki omogočajo spreminjanje vrednosti.

Vsak objekt ima tu zapisane vse lastnosti, poleg tega pa še sliko objekta.

Spremenljivka »isUnlocked« se uporablja za prikaz opreme, ki jo je uporabnik pridobil (če je vrednost spremenljivke »true« pomeni, da je igralec kos opreme pridobil in mu je odklenjen).

Zapis orožij v objekt vseh orožij:

```
public class Weapon
    {
        public WoodenSword woodenS = new WoodenSword();
        public IronSword ironS = new IronSword();
        public PickleSword pickleS = new PickleSword();
        public GoblinSword goblinS = new GoblinSword();
        public MonkeMallet monkeM = new MonkeMallet();
        public DinoSword dinoS = new DinoSword();
    }
```

Po prvotnem zapisu želenega dela opreme v objekt, ga moramo še kreirati in vstaviti v polje vseh orožij.

Zapis vseh orožij v polje, ki ga program nato uporablja za klic želenega orožja:

```
public Weapon weapons = new Weapon();
```

Vsi materiali so ustvarjeni na enak način, zapis v polje za klic je pa drugačen, ker je njihova uporabnost drugačna.

Materiali so zapisani v razredu »Inventory«, kjer so shranjene tudi funkcije za pridobitev in uporabo materialov, poleg tega pa še funkciji za prikaz vseh igralčevih materialov ter zahtevano število materialov za izdelavo opreme.

```
public Item[] items = { new PickleJuice(), new PickleSkin(), new GoblinHide(), new
GoblinSkull(), new MonkeBones(), new MonkeCarapace(),
            new DinoTeeth(), new DinoClaws(), new DinoTail(),
            new IronOre(), new MythrilOre(), new EnhancementStone()};
```

Zapis materialov je narejen tako, da se vse zapiše v neko polje, kjer ima vsak posamezni material svojo identifikacijsko številko, ki se v programu uporablja za pridobitev in porabo materiala.

```
public void incrementQuantity(int i)
        {
            itemQuantity += i;
        }
public void decrementQuantity(int i)
        {
            itemQuantity -= i;
        }
```

V razredu za predmete sta zapisani zgornji funkciji, prva se uporablja za dodajanje, druga pa za odvzemanje predmetov iz igralčeve torbe.

Na koncu imamo še razred z recepti za orožja in oklepe, in sicer ima vsak recept zapisano potrebno količino materialov za izdelavo.

V razredu za recepte so še funkcije za preverjanje, če ima igralec dovolj materialov za izdelavo opreme, izdelavo želene opreme in za odklepanje izdelanega kosa opreme.

Podatki, ki niso potrebni za shranjevanje, imajo pred imenom »[JsonIgnore]«, kar programu pove, naj ga preskoči, ko izdeluje kopijo trenutnega stanja igre.

```
public class PickleJuice : Item
    {
        readonly public static int ID = 0;
        public PickleJuice()
        {
            itemName = "Pickle Juice";
            itemID = ID;
            itemQuantity = 0;
        }
    }
```

Tako so naslovljeni materiali: prvi podatek je identifikacijska številka, ki se uporablja za klic teh materialov, nato je spodaj nastavljen še itemID na to številko, saj je zgornji podatek le za branje.

```
        public int coins = 0;
        public int points = 0;
        public int str = 0;
        public int def = 0;
        public int playerMaxHP = 30;;
        public double exp = 0;
        public int maxExp = 10;
        public int level = 1;
        public int newLevel = 2;
        public bool didPlayerEat = false;
```

Zgoraj so navedeni igralčevi podatki in sicer:

- Kovanci
- Točke (za nadgrajevanje moči in obrambe)
- Moč
- Obramba
- Igralčeve življenjske točke
- Točke izkušenj
- Prag za točke izkušenj
- Igralčev nivo
- Igralčev novi nivo (uporabljeno za dodajanje novega nivoja)
- Ali je igralec pojedel (pred misijo)

2.4 ZEMLJEVID

Slika 3: Zemljevid

S pritiskom na gumb z ikono zemljevida na domačemu zaslonu nas program popelje do zemljevidnega okna, kjer lahko igralec izbira, kam bo šel.

Ko začnemo novo igro, imamo odklenjen le prvi nivo »Seabed«, kjer se igralec spopada z morskimi nasprotniki, poleg tega ima pa na voljo še »LowRank Grind«, kjer hitreje pridobiva točke izkušenj in kovance ter s tem postane močnejši, da lahko poskusi premagati končnega nasprotnika prvega nivoja.

Po izdelavi orožja in oklepa iz prvega nivoja (Pickle Sword in Pickle Armor) se uporabniku odklene naslednji nivo »Forest« ter neskončni spopad drugega nivoja, »MidRank Grind«.

Igra se nadaljuje iz nivoja v nivo, pri čemer je vsaka naslednja stopnja težja od prejšnje.

Poleg stopnjevanja težavnosti se z odklepanjem naslednjega nivoja igralcu onemogoči pridobivanje kamnov za nadgrajevanje opreme v prejšnjih nivojih.

Ko igralec premaga vse 3 nivoje in izdela vso opremo iz vseh nivojev, se mu odklene arena »Boss«, kjer se uporabnik spopade s končnim nasprotnikom igre.

Če mu ga uspe premagati, se odklene še zadnji nivo »Endless«, v katerem se pred igralcem prikažejo vsi nasprotniki, ki nastopajo v igri.

Poleg prvega nivoja ima igralec odklenjen tudi »Mining« oz. rudarjenje in »Hunting«, kjer se gre igralec na lov za hrano.

- Na vrhu sedi gumb za vračanje na prejšnje okno.
- Pod njim so nivoji, od zgoraj navzdol: misija (glavni spopad), neskončni spopad nivoja in neskončni spopad vseh nasprotnikov, na desni strani je končni nasprotnik igre.
- Na dnu sta še gumba za lov in rudarjenje.

Pri rudarjenju igralec pridobi rudo, ki jo potrebuje za izdelavo močnejše opreme.

Pri lovu igralec pridobi meso, ki mu zviša moč napada za izbrano misijo.

Koda za pošiljanje podatkov v drugo formo je tu drugačna, in sicer:

```
STS s = new STS(selectedWeapon, selectedArmor, boss, mob1, mob2, mob3, specialMob, ref p);
```

Poleg kazalca na igralčeve podatke se iz tega okna pošljejo še vsi nasprotniki, ki nastopajo na določenem nivoju.

2.5 GLAVNI SPOPAD

Slika 4: Glavni spopad - izbira

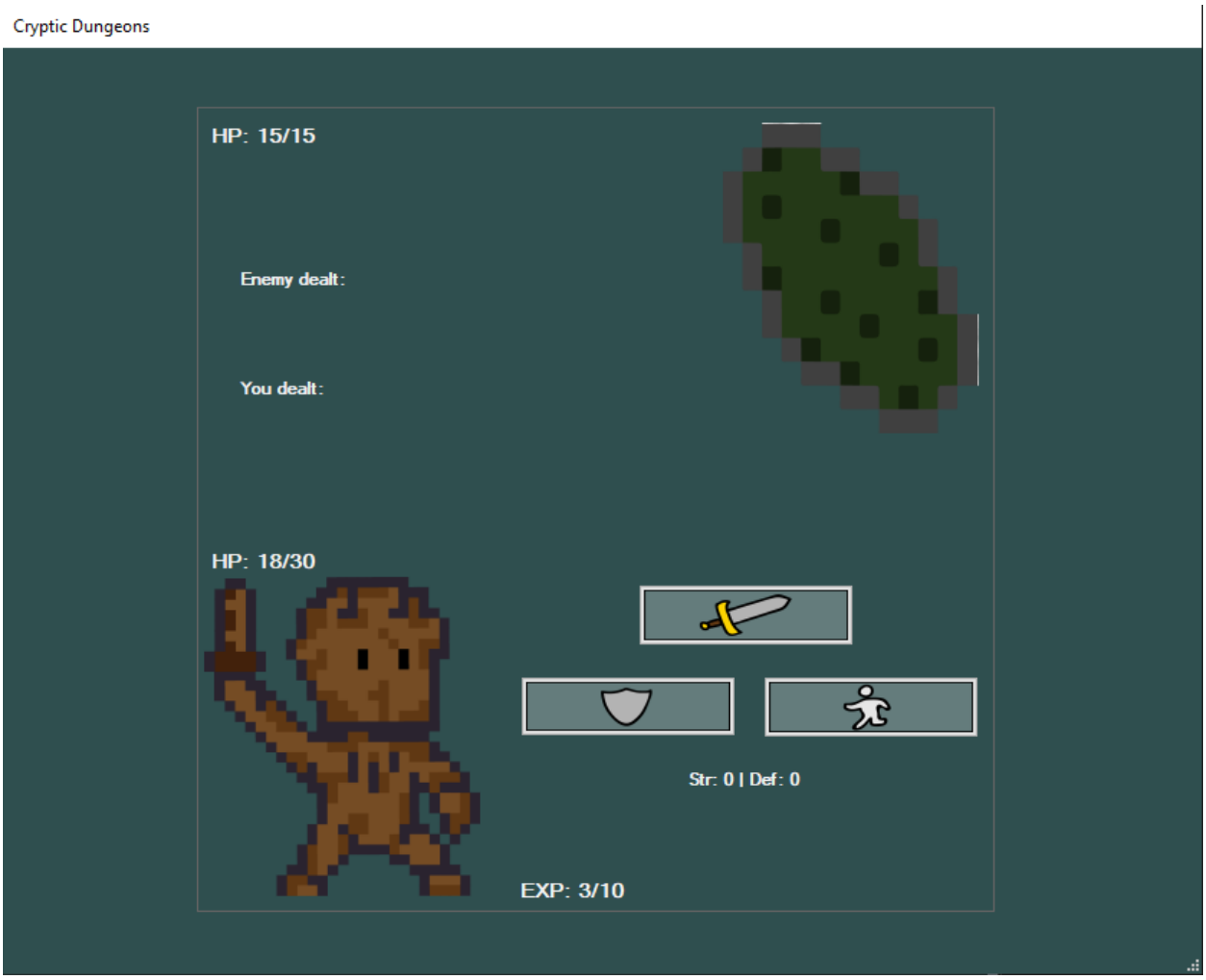
Ko igralec začne misijo, se mu odpre zgornja slika.

Tu ima izbiro med spopadom in naključnim izborom.

Naključni izbor ima 30% možnosti preskoka stopnje, 20% možnosti, da se mu zvišajo življenjske točke za 10 in 50% možnosti spopada.

Na tretji stopnji ima igralec možnost spopada s posebnim nasprotnikom, ki je sicer močnejši, ampak je tudi več vreden in ima možnost puščanja kamnov za nadgrajevanje za sabo.

Zadnja stopnja misije je spopad s končnim nasprotnikom, ki pušča za sabo materiale, če ga igralec premaga.



Slika 5: Glavni spopad - bitka

Če igralec izbere bitko, se mu prikaže ta zaslon.

Na vrhu se vidijo nasprotnikove življenjske točke, pod njimi se izpisuje igralčev in nasprotnikov napad, pod tem so pa še igralčeve življenjske pike.

Levo spodaj je prikaz igralca, desno zgoraj pa prikaz nasprotnika.

Desno od igralca vidimo 3 gumbe: gumb za napad, gumb za obrambo in gumb za beg iz boja.

Pod gumbi vidimo še atribute igralca, na dnu zaslona pa nam prikazuje še naše točke izkušenj.

Igra deluje tako, da igralec izbere enega izmed treh gumbov in poskuša zbiti nasprotnikove življenjske točke na 0, medtem ko sam poskuša preživeti spopad.

Če igralec nasprotnika premaga, se mu odklene naslednja stopnja, vse do končnega spopada, kjer ob zmagi pridobi materiale, ki jih lahko kasneje uporabi za izdelavo opreme.

Ob izbiri spopada se izvede funkcija za prikaz borbenega prizora, ki je podobna kot funkcija za prikaz in skrivanje opreme na domačem zaslonu, in sicer gre program skozi zanko, kjer preverja če so vsi argumenti pravilni in prikaže, kar je potrebno.

Ob vsaki zmagi se izvede funkcija, ki igralcu dodeli priigrane kovance in točke izkušenj, hkrati pa še preveri, če je igralec pridobil dovolj točk izkušenj za zvišanje njegovega nivoja.

Poleg pridobivanja zasluženih točk in kovancev program še zaklene trenutni izbor in odklene izbor naslednje stopnje (naključni izbor/spopad/posebni nasprotnik).

Če je igralec premagal posebnega nasprotnika, pridobi še kamen za nadgrajevanje opreme.

Če igralec premaga končnega nasprotnika nivoja, pa pridobi naključno količino izbranega materiala, ki ga lahko kasneje uporabi za izdelavo opreme.

Pri spopadu je igralčev napad izračunan s seštevanjem polovice igralčeve moči, naključnim številom med najnižjim in najvišjim napadom izbranega orožja in dodatnimi napadnimi točkami pridobljenimi iz uporabe mesa z lova:

```
x = p.str / 2 + r.Next(selectedWeapon.weaponDmgMin, selectedWeapon.weaponDmgMax + 1) + foodDmg;
```

Nasprotnikov napad pa se izračuna z naključnim številom med najnižjim in najvišjim možnim nasprotnikovim napadom, od česar pa se še odšteje polovična igralčeva obramba in tretjina obrambe izbranega oklepa:

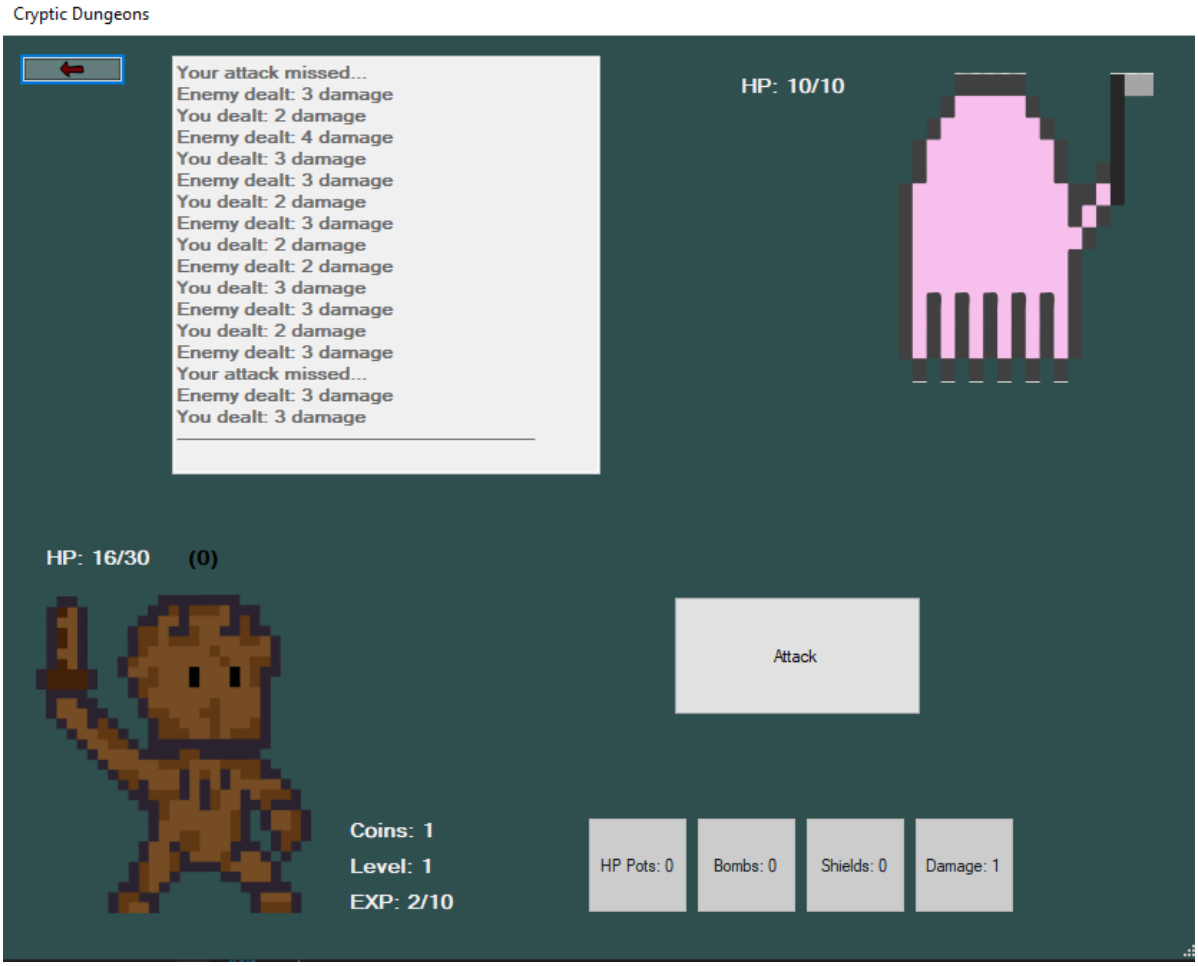
```
p.y = r.Next(selectedMob.minMobDmg, selectedMob.maxMobDmg + 1);
p.y = p.y - p.def / 2 - selectedArmor.armorDefense / 3;
```

Boj je zgrajen tako, da se takoj po igralčevem napadu izvrši še nasprotnikov napad.

Poleg možnosti za napad lahko igralec izbere obrambo, ob kateri ima možnost preslikati nasprotnikov napad nazaj na nasprotnika, ter istočasno zniža nasprotnikov oklep za polovico izbranega igralčevega oklepa.

Tretja možnost pri spopadu je še pobeg iz bojišča, kar pa ima 30% možnosti uspeha.

2.6 NESKONČNI SPOPAD



Slika 6: Neskončni spopad

Uporabnik pride do zgornjega okna, če izbere katerokoli »Grind« opcijo iz zemljevida.

Za spremembo od glavnega spopada, tu igralec ne more priti do končnega nasprotnika nivoja in ne more pridobiti njegovih materialov.

Ta način igranja je tudi drugačen od glavnega spopada, saj v tem primeru igralec nima možnosti obrambe ali bega z bojišča, temveč pridobi druge funkcije.

Pri neskončnem spopadu ima igralec na voljo 4 dodatne možnosti:

- Življenjski napitek (za zvišanje življenjskih pik)
- Eksploziv (za močne nasprotnike – naredi veliko močnejši napad)
- Ščit (se doda k igralčevim življenjskim pikam in nasprotnik lahko napade le ščit)
- Dodatne napadalne točke (zvišajo moč igralčevega naslednjega napada)

Za razliko od glavnega spopada lahko igralec tu hitreje pridobi točke izkušenj in kovance, saj so spopadi hitrejši.

Poleg hitrejšega pridobivanja moči pa mora biti igralec tu tudi bolj pazljiv, saj lahko izgubi točke izkušenj, če izgubi vse življenjske pike.

Igralec pa lahko iz spopada pobegne nazaj na domači zaslon kadarkoli želi.

Neskončni spopadi so določeni za vsak nivo in vsak ima drugačne nasprotnike.

Izbira nasprotnikov se izvede glede na izbrano stopnjo:

```
Type[] chooseMob = Assembly.GetAssembly(typeof(MobLowRank)).GetTypes().Where(TheType =>
                    TheType.IsClass && !TheType.IsAbstract &&
                    TheType.GetInterfaces().Contains(typeof(MobLowRank))).ToArray();
x = r.Next(chooseMob.Length);
ConstructorInfo ctor = chooseMob[x].GetConstructor(new Type[0]);
selectedMob = (Mob)ctor.Invoke(new object[] { });
```

Prvi stavek v tej kodi poišče nasprotnike, ki imajo značko »MobLowRank« pri zapisu. V tem primeru bodo izbrani nasprotniki za prvi nivo.

Naslednja dva stavka naključno izbereta enega izmed teh nasprotnikov in ga shranita.

Zadnji stavek zapiše izbranega nasprotnika v spremenljivko »selectedMob«, ki se uporabi za prikaz nasprotnika na zaslonu.

Po vsaki zmagi igralec pridobi nov predmet, ki mu pomaga pri naslednjem boju (napitek/eksploziv/ščit/dodatni napad), lahko pa uporabi le enega na spopad.

Ta način igre je namenjen pridobivanju izkušenj, da lahko uporabnik premaga končnega nasprotnika v glavnemu spopadu.

Ko igralčeve točke izkušenj presežejo prag, se zviša igralčev nivo. Igralec tako pridobi 2 točki, ki ju lahko spravi v moč ali obrambo, poleg tega se pa igralcu še zvišajo življenjske točke za 5 pik.

Zadnji nivo, ki ga igralec odklene, je neskončni spopad vseh nasprotnikov, kjer se mu prikažejo vsi nasprotniki, ki so nastopili v igri. Narejen je pa istem konceptu kot neskončni spopad nivoja, le da se tam prikažejo vsi nasprotniki in igralec lahko pridobi več točk izkušenj.

Po vsaki zmagi se izvede naslednja funkcija:

```
public void dropItems()
        {
            x = r.Next(12);
            if (x >= 0 && x <= 2)
                p.items.healthPot.incrementQuantity(1);
            else if (x >= 3 && x <= 5)
                p.items.shield.incrementQuantity(1);
            else if (x >= 6 && x <= 8)
                p.items.bomb.incrementQuantity(1);
            else
                p.items.sword.incrementQuantity(1);

            updateItems();
        }
```

Na tem mestu program naključno dodeli igralcu nek predmet, ki ga lahko uporabi v naslednjem spopadu.

Program tu namesto dodajanja količine izbranega predmeta preko npr. x = x + 1, uporabi funkcijo »incrementQuantity()«, ki je zapisana v datoteki s podatki.

Po dodelitvi predmeta igralcu se izvede še funkcija »updateItems()«, ki posodobi igralčeve predmete. Ta posodobitev se zgodi tudi po uporabi predmeta.

Predmet se odklene, če je njegova količina večja od 0.

Moč predmeta se prilagaja glede na izbran nivo. Primer:

```
private void btnShield_Click(object sender, EventArgs e)
        {
            if (p.items.shield.itemQuantity > 0)
            {
                p.items.shield.decrementQuantity(1);
                lockItems();
                updateItems();
                if (p.stageRank == 1)
                    p.shields += 6;
                else if (p.stageRank == 2)
                    p.shields += 10;
                else if (p.stageRank == 3)
                    p.shields += 15;
                lblShields.Text = "(" + p.shields + ")";
            }
        }
```

Pri uporabi predmeta se zniža njegova količina s funkcijo »decrementQuantity()«, takoj za tem se zaklenejo vsi predmeti ter se posodobijo njihove količine.

Za tem program preveri kateri nivo je izbran in prilagodi moč predmeta glede nanj.

2.7 KONČNI NASPROTNIK

Slika 7: Končni nasprotnik

Spopad s končnim nasprotnikom je mešanica glavnega borbenega prizora in neskončnega spopada.

Igralec ima tu na voljo 3 življenjske napitke, poleg tega pa ima na voljo še možnost napada in obrambe.

Ko igralec premaga in izdela orožje in oklep od končnega nasprotnika, se mu odklene še neskončni spopad z vsemi nasprotniki.

Končno orožje ima še poseben učinek, ki se lahko naključno sproži ob napadu in doda uporabniku dodatnih 5 napadalnih točk za 3 napade.

Pri končnemu spopadu so vpisane še dodatne funkcije za popestritev igre.

```
public void unlockItems()
        {
            if (p.items.healthPot.itemQuantity > 0)
                btnPotion.Enabled = true;
        }
```

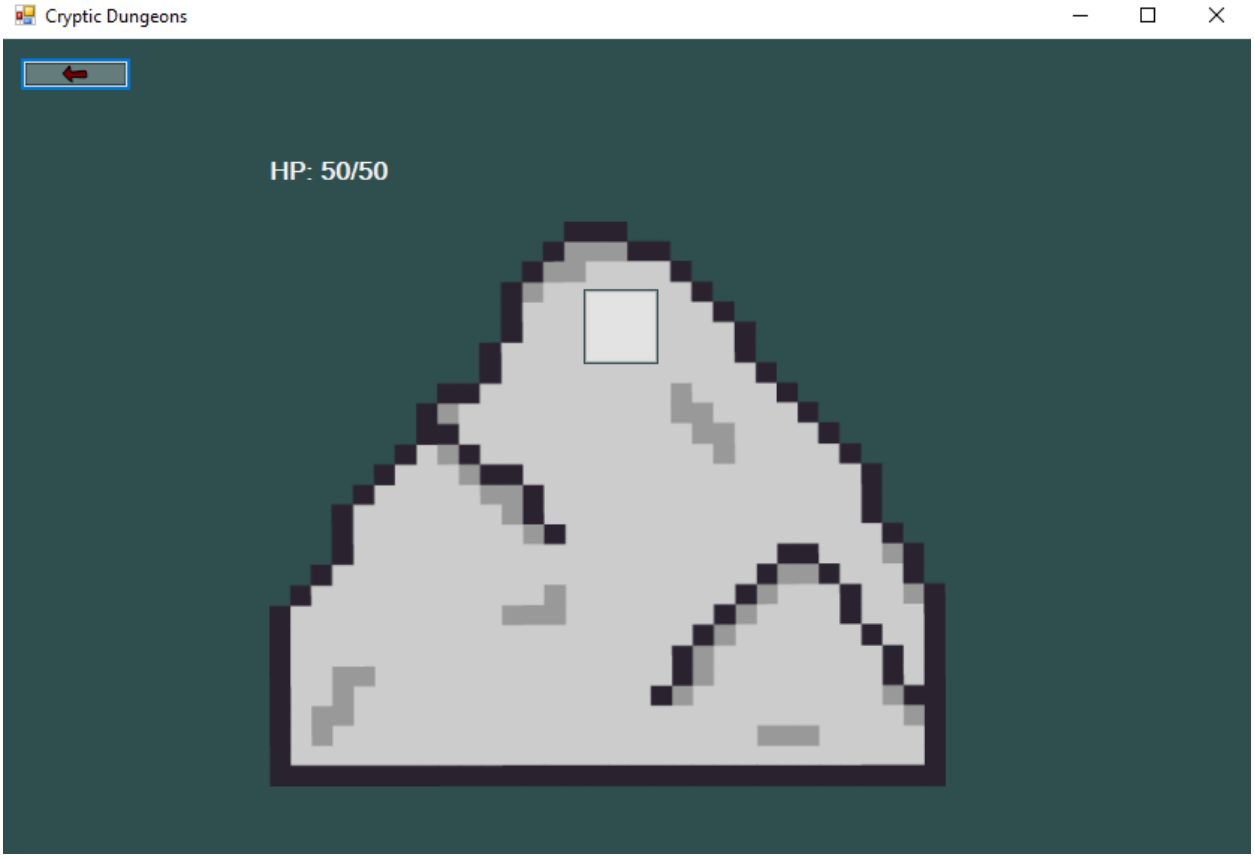
Zgornja funkcija igralcu odklene življenske napitke, če jih le ta ima.

Poleg bolj zanimivega načina igranja pri končnem nivoju, je tudi orožje, ki ga lahko igralec izdela z dodatno funkcijo.

```
if (fireAttack == true)
            {
                fireCount--;
                p.enemyHP -= 5;
                lblEnemyHP.Text = "HP: " + p.enemyHP + "/" + p.enemyMaxHP;
                txtDmgOutput.AppendText("Enemy burned for an additional 5 damage");
```

Zadnjemu in najmočnejšemu orožju, ki ga lahko igralec izdela, pripada zgornja funkcija, ki se naključno sproži za 3 napade. Če se aktivira posebni učinek orožja, bo ob vsakem igralčevem napadu, nasprotnik izgubil še dodatnih 5 življenskih točk.

2.8 RUDARJENJE



Slika 8: Rudarjenje

Pri pritisku na »Mining« iz zemljevida se uporabniku odpre zgornje okno.

Tu mora igralec pritisniti na gumbe, ki se prikazujejo na zaslonu, da pridobi rudo, potrebno za izdelavo določene opreme.

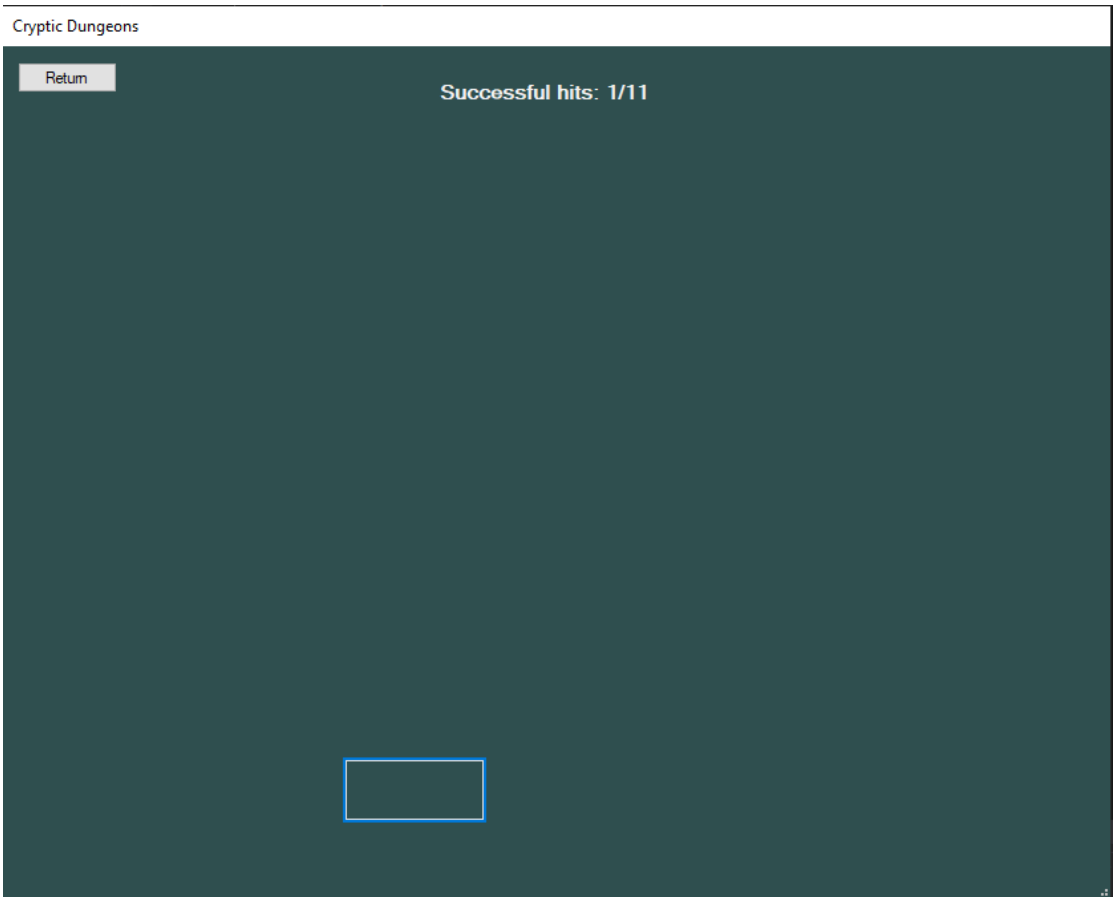
Ko igralec dovoljkrat zadene gumb, se kamen poruši in igralec pridobi material.

Ob uničenju se izvede stavek:

```
p.inventory.addItem(selectedOre, 1);
```

Ta stavek uporabi funkcijo »addItem()«, ki je zapisana v datoteki s podatki.

2.9 LOV



Slika 9: Lov

Ko želi igralec iti na lov, pritisne na zemljevidu gumb »Hunting«.

Ta igra deluje tako, da se na zaslonu prikaže gumb, ki se prestavi, ko ga igralec pritisne.

Ko igralec uspešno zadene gumb 11-krat, je konec igre in uporabnik pridobi meso, ki ga lahko nato uporabi za zvišanje napada za izbrano misijo.

```
public void randomizeButtonLocations()
        {
            xx = r.Next(20, 720);
            yy = r.Next(100, 580);

            button1.Location = new Point(xx, yy);
        }
```

Zgornji zapis kode se uporabi za naključno dodeljevanje naslednje pozicije gumba.

Spremenljivka »xx« predstavlja X koordinato, spremenljivka »yy« pa Y koordinato v programu.

2.10 TRGOVINA

Slika 10: Prikaz trgovine

Ko na našem domačem zaslonu pritisnemo na gumb za trgovino, se odpre nov Windows Form, ki prikazuje orožje in oklep, ki ju lahko kupimo s kovanci in železom.

Na zgornji sliki vidimo opremo, ki se odklene za nakup, ko so izpolnjeni vsi pogoji (pravilna količina kovancev in železa).

- Na vrhu je gumb »Return«, ki nas vrne na domači ekran.
- Pod njim je oprema, ki jo lahko kupimo (gumb »Buy« se odklene, ko ima igralec vse materiale)
- Na dnu se nam prikaže naše število kovancev.

Koda za pregledovanje razpoložljivosti izdelka:

```
public void isAvailable()
        {
            if (p.coins >= p.weapons.ironS.coinValue && p.weapons.ironS.isObtained ==
              false && p.inventory.requiredMaterials(9) >= 4)
                    btnBuyIronS.Enabled = true;
```

Ta funkcija se izvede ob inicializaciji forme in preveri, če ima igralec dovolj kovancev in rude za nakup želenega kosa opreme ter odklene gumb za nakup, če so izpolnjeni vsi pogoji.

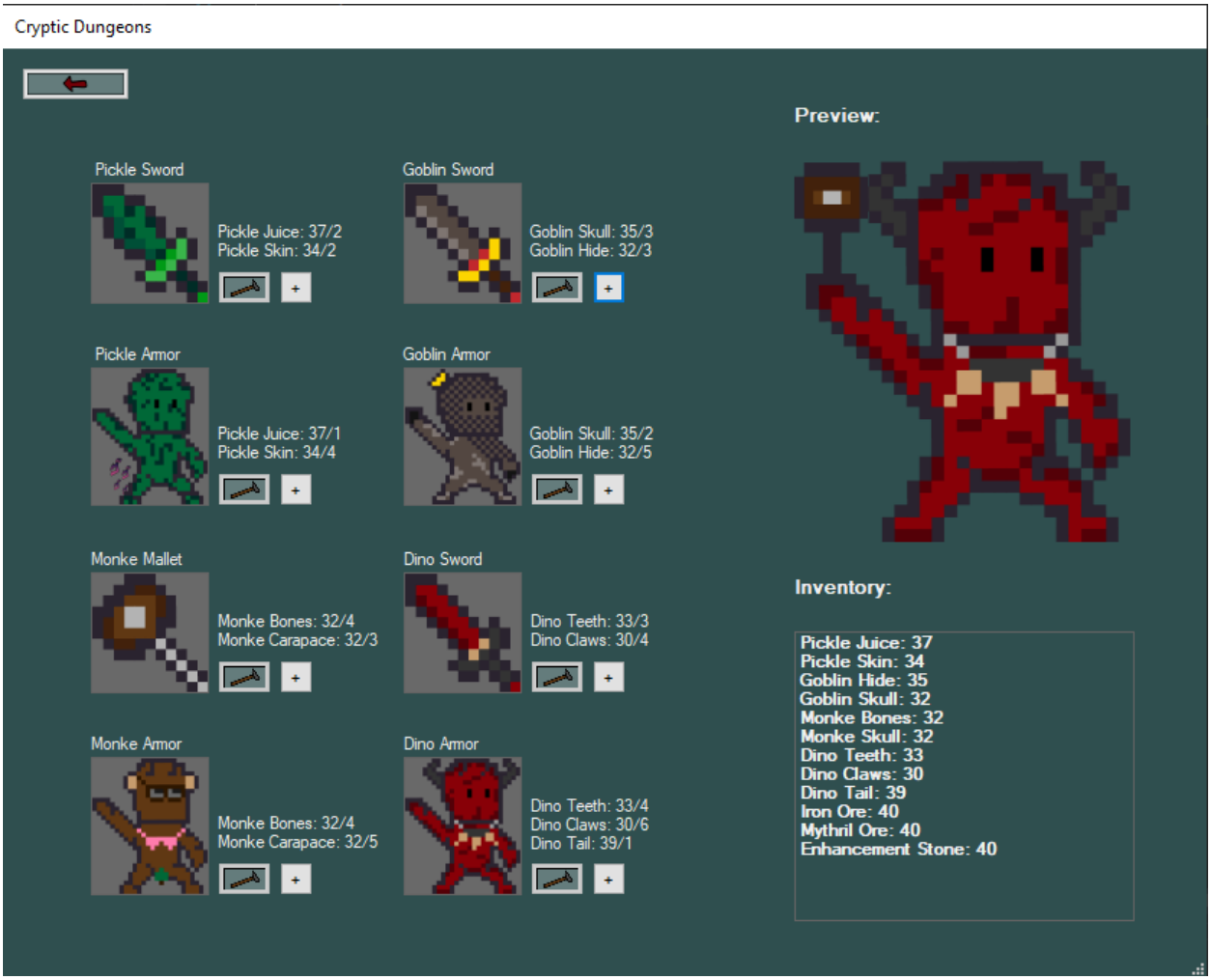
```
public void updateMaterials()
        {
            lblIronS.Text = "Coins: " + p.coins + "/35" + Environment.NewLine + "Iron
              Ore: " + p.inventory.requiredMaterials(9) + "/4";
            lblIronA.Text = "Coins: " + p.coins + "/60" + Environment.NewLine + "Iron
              Ore: " + p.inventory.requiredMaterials(9) + "/8";
        }
```

Zgornja koda se uporablja za posodobitev materialov, ki jih ima igralec v svoji torbi.

Ukaz »Enviroment.NewLine« se uporabi za preskok v novo vrstico.

V kodi se za prikaz potrebnih materialov uporablja funkcija »requiredMaterials()« iz datoteke s podatki, ki uporabi identifikacijsko številko materiala, da programu pove, kateri material naj preverja.

2.11 KOVAČ



Slika 11: Prikaz kovača

Ob pritisku na gumb z ikono nakovala na domačemu zaslonu se nam odpre okno, kjer lahko izdelujemo in nadgrajujemo opremo.

- Na vrhu zopet vidimo gumb za povrnitev na prejšnje okno.

- Pod njim imamo orožja in oklepe, ki jih lahko izdelamo v igri. Oprema je zaklenjena, dokler igralec ne pridobi vseh potrebnih materialov za izdelavo. Materiale igralec pridobiva na misijah, ti se shranijo v igralčevo torbo, prikažejo se pa tudi nad gumbom za izdelavo dela opreme, kar nam pove koliko materialov imamo in koliko jih še potrebujemo.

- Na desni strani imamo izgled igralca in igralčevo torbo z vso vsebino.

Pri kovaču je pregled razpoložljivosti izdelave drugačen kot pri trgovini, saj se tu uporablja še recept za izdelavo opreme.

V programu je zapisana funkcija »isCraftable()«, ki preverja, če ima igralec dovolj materialov za izdelavo želene opreme:

```
if (p.recipes.dinoS.isCraftable(ref p) == true && p.weapons.dinoS.isObtained == false)
            {
                btnCraftDinoS.Enabled = true;
                btnCraftDinoS.BackColor = Color.DarkSlateGray;
            }
```

V temu stavku program kliče funkcijo iz datoteke s podatki, ki primerja igralčeve materiale z materiali, ki so potrebni za izdelavo dela opreme.

Poleg tega se še preveri, če je uporabnik orožje ali oklep že pridobil, oz. izdelal in mu prepreči ponovno izdelavo.

Ko je pa igralec enkrat kos opreme izdelal, se mu prikaže gumb za nadgradnjo, če ima dovolj kovancev in kamnov za nadgradnjo .

Določen del opreme se lahko nadgradi do 5. stopnje.

Pri orožjih se zviša napad, pri oklepu pa obramba.

Koda za preverjanje razpoložljivosti za nadgradnjo:

```
if (p.weapons.pickleS.isUnlocked == true && p.weapons.pickleS.enchLevel < 5 &&
       p.inventory.requiredMaterials(11) >= 1 && p.coins >= 10)
                btnUpgradePS.Visible = true;
            else
                btnUpgradePS.Visible = false;
```

Tu program preveri, če je igralec opremo že pridobil, poleg tega pa še preveri, če je nivo nadgradnje manjši 5, če ima igralec dovolj kamnov za nadgradnjo ter če ima dovolj kovancev.

Če so vsi pogoji izpolnjeni, se prikaže gumb »+« za nadgradnjo.

2.12 RECEPTI ZA OPREMO

Za izdelavo želene opreme potrebuje igralec pridobiti določeno količino materialov.

Vsak del opreme ima svoj recept:

```
public class DinoArmorRecipe : Recipe
    {
        public DinoArmorRecipe()
        {
            recipe = new Dictionary<int, int>()
            {
                { DinoTeeth.ID, 4},
                { DinoClaws.ID, 6},
                { DinoTail.ID, 1 }
            };
        }

        override public void unlock(ref Podatki p)
        {
            p.armors.dinoA.isUnlocked = true;
        }
    }
```

Vsak recept za opremo je zapisan na način iz zgornjega primera kode, in sicer:

- Zaporedna številka posameznega materiala.
- Funkcija, ki odklene kos opreme.

Ko ima igralec enkrat dovolj materialov se ob obisku kovača odklene gumb za izdelavo izbrane opreme, ki je vezan na zgornji zapis kode.

```
public class Recipes
    {
        public PickleSwordRecipe pickleS = new PickleSwordRecipe();
        public PickleArmorRecipe pickleA = new PickleArmorRecipe();
        public GoblinSwordRecipe goblinS = new GoblinSwordRecipe();
        public GoblinArmorRecipe goblinA = new GoblinArmorRecipe();
        public MonkeArmorRecipe monkeA = new MonkeArmorRecipe();
        public MonkeMalletRecipe monkeM = new MonkeMalletRecipe();
        public DinoSwordRecipe dinoS = new DinoSwordRecipe();
        public DinoArmorRecipe dinoA = new DinoArmorRecipe();
    }
```

Zgornji zapis kode prikazuje kako se recepti generirajo, ko se igra zažene.

3 SKLEP

Projekt sem začel brez kakršnegakoli znanja o razvijanju iger, poznal sem le, do neke mere, programski jezik. Zato sem moral večino kode, ki sestavlja program, napisati s pomočjo podatkov in nasvetov na spletu.

Na srečo je splet že toliko razvit, da sem vse potrebno našel brez večjega truda in brez pomoči spletne strani StackOverflow (namenjena pomoči pri pisanju programske kode), mi projekta ne bi uspelo izdelati, vsaj ne tako optimizirano kot je končni produkt.

Skozi gradnjo projekta sem ugotovil, da sem se pravilno odločil, saj mi je projekt razširil obzorje.

Če bi moral projekt ponovno začeti, bi spremenil nekaj stvari.

- Spremenil bi program za izrisovanje likov. Za ta projekt sem uporabljal Adobe Illustrator, saj je relativno enostaven za uporabo, a mu manjkajo pripomočki za izdelovanje 8-bitnih likov. Če bi moral like še enkrat izdelati, bi uporabil program Aseprite, ki je namenjen izdelavi takšne grafike.

- Namesto izdelave programa, ki odpira razna okna, bi igro izdelal za HTML5 (naredil bi spletno aplikacijo), saj sta grafika in stil igranja bolj primerna za splet.

- Od samega začetka bi razmišljal bolj v smeri optimizacije, saj sem za projekt porabil precej ur, da sem spremenil stvari, ki bi jih lahko že prvič drugače zapisal.

- Pred začetkom izdelave programa bi se bolj pozanimal o izvedbi programa in bi si zapisal bolj podroben načrt, da bi izdelava projekta potekala bolj tekoče.

Kljub vsem problemov, na katere sem naletel, je bil projekt precej poučen, tako da se je moje znanje zelo povečalo. Zahvaljujoč predmetu Programiranje 2 in profesorju Aleksandaru Lazareviću bom z razvijanjem iger nadaljeval in upam, da bom nekoč izdelal nekaj, kar bom lahko delil tudi z ostalimi.

4 VIRI

W3Schools, dostopno 20. maj, 2021, https://www.w3schools.com/cs/index.php

Microsoft C# documentation, dostopno 15. maj, 2021, https://docs.microsoft.com/en-us/dotnet/csharp/

Stack Overflow, dostopno 23. maj, 2021, https://stackoverflow.com/

PRILOGE

NASPROTNIKI

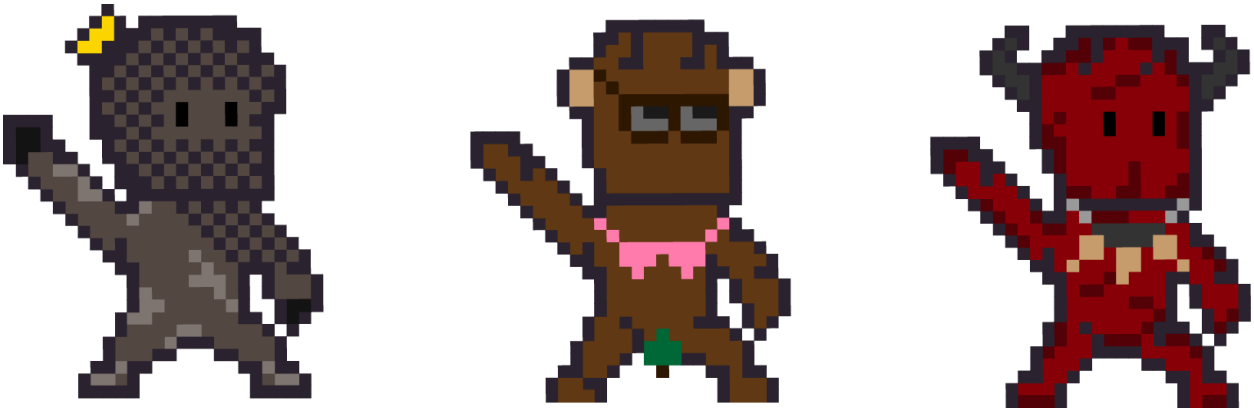
Slika 12: Nasprotniki 2. Nivoja

Slika 13: Nasprotniki 3. Nivoja

Slika 14: Končni nasprotnik

OKLEPI

Slika 15: Železni oklep, Bronasti oklep, "Pickle Armor"



Slika 16: Goblin oklep, Opičji oklep, Dinozaver oklep

OROŽJA

Slika 17: Lesen meč, Železen meč, "Pickle Sword"

Slika 18: Goblin meč, Opičji meč, Dinozaver meč