Izrada "pixelart" igre uloga u programskom alatu Unity

Tokić, Danijel

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Danijel Tokić

IZRADA PIXELART IGRE ULOGA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Danijel Tokić

JMBAG: 0016129661

Studij: Informacijski sustavi

IZRADA PIXELART IGRE ULOGA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Mladen Konecki

Varaždin, rujan 2021.

Danijel Tokić

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu će biti opisan žanr video igara zvan igre uloga te će biti opisane glavne funkcionalnosti takvih igara. Prvo će biti opisani alati pomoću kojih je izrađen praktični dio rada (Unity, Visual Studio, Pixilart), te nakon toga opisa samoga žanra i primjeri igara koji se nalazu u tome žanru. Zatim će biti opisan sam dizajn korišten za razvoj igre te na koji način se kreira i implementira taj dizajn unutar Unity alata. Nakon toga će biti opisan praktični dio projekta koji je će biti razdvojen po glavnim funkcionalnostima te će sve te funkcionalnosti biti dodatno razdvojene na manje dijelove. Ti dijelovi će biti opisani te će biti prikazani dijelovi koda koji su potrebni za te funkcionalnosti, također će kodovi biti opisani.

Ključne riječi: algoritmi, programiranje, računalne igre, Unity, igra uloga, pixelart

Sadržaj

1. Uvod

Razvoj igara svake godine napreduje sve više i više, te same igre postaju naprednije sa razvojem računalnih komponenti, a korisnici zahtijevaju sve bolje igre. Iako igre postaju sve zahtjevnije za naše uređaje te igre testiraju limite naših računala i dalje ljudi uživaju i u jednostavnim igrama koje nisu zahtjevne. Najbolji primjer je igra „Among Us“ koja je 2020. godine eksplodirala u popularnosti iako je jednostavna igra žanra pronađi uljeza [1]. Isto tako kako ljudi uživaju u jednostavnim igrama, uživaju i u igrama koje izgledaju kao da su stare tj. napravljene u starom stilu tako zvanom pixelart stilu, u kojem su bile rađene stare igre jer su 3D modeli bili previše zahtjevni za to vrijeme.

Kako se industrija razvoja video igara širila tako je nastalo i puno različitih žanrova, kao što su platformeri, akcijske igre, utrke, borbene igre te igre uloga. Kod žanra igri uloga postoje igre za jednog igrača kao što je na primjer „*The Elder Scrolls*“ serijal te postoje igre za više igrača kao što je „*World of Warcraft*“. Iako igre funkcioniraju na različite načine njihova ideja je ista, a to je da igrača stave u novi izmišljeni svijet gdje on kontrolira svojega lika te da postane dio toga svijeta. Iako taj žanr ne mora biti izrazito kompliciran za napravit u osnovi, takve igre postaju teške za izraditi jer se treba kreirati potpuno novi svijet koji će privući igrača te će mu biti zanimljivo istražiti taj novi svijet.

U ovome radu će biti opisane neke od najpopularniji igara žanra igri uloga, te će se prikazati na novoj kreiranoj igri neke od stvari koje su zajedničke skoro svim igrama ovoga žanra kao što je borba, razgovor sa ljudima toga izmišljenog svijeta,… te će biti prikazani neki dijelovi koda kako bi se prikazao način funkcioniranja Unity game engine-a.

2. Alati

Za implementaciju praktičnog dijela rada korišten je game engine Unity za kreiranje same igre, Visual Studio za razvoj koda te Pixilart web aplikacija za crtanje *pixelart* slika koje se unutar Unity-a pretvaraju u animacije.

2.1. Unity

Unity game engine je razvijen od strane Unity Technologies 2005. godine te je besplatan za korištenje u nekomercijalne svrhe [2]. Unity podržava razvoj 3D, 2D, VR (virtualno realnih) te AR (prividno realnih) igara. Bazira se na C# programskom jeziku, te sadrži posebnu klasu u C# (MonoBehaviour) iz koje nastaju sve klase za Unity Engine [3]. MonoBehaviour klasa posjeduje svoje metode kao što je Start koja se poziva pri instanciranju objekta, metoda kao što je Update koja se poziva nakon svakog frame-a (slike) u igri te mnoge druge metode [3].

Unity je trenutno jedan od najpopularnijih game engine-a gdje ga čak i veće kompanije koriste umjesto da razvijaju svoj game engine. Zbog svoje popularnosti među novim developerima također postoji i puno tečaja i videa na internetu koji pomažu učenju Unity-a te samom razvoju video igara [4].

Velikoj popularnosti Unity-a je pripomogao Unity Asset Store gdje drugi ljudi objavljuju svoje 3D modele, zvukove, animacije, slike i sl. [5]. Te stvari mogu drugi ljudi kupiti te koristiti u svojim projektima te na taj način se privuče više ljudi za koristiti Unity jer će na primjer osoba koja ne zna modelirati ali zna programirati moći samo koristiti modele koje je napravio netko drugi a on će samo napraviti kod.

2.2. Visual Studio

Microsoft Visual Studio je razvojno okruženje za pisanje programskih jezika razvijeno od strane Microsofta [6]. Visual Studio ima potporu razvoja svih programskih jezika za razne tehnologije, jezici koji su podržani su C#, F#, C++, Python,… te se mogu razvijati mobilne aplikacije, web aplikacije, računalni programi, video igre,…

Visual Studio je snažan alat ne samo za razvijanje koda već ima i snažan alat za pronalaženje pogrešaka koda te za samo testiranje koda. Također ima integrirano lakše

korištenje svih baza podataka te ima potporu za instaliranje raznih „produžetaka“ koji olakšavaju rad sa već postojećim tehnologija te dodaju dodatne potpore kao sa na primjer alatima za verzioniranje na primjer GitHub-om [6].

2.3. Pixilart

Pixilart je besplatna web aplikacija za kreiranje *pixelart*-a. Aplikacija je kreirana 2013. godine od strane Bryan Ware-a kojem je cilj bilo napraviti aplikaciju za *pixelart* dostupnu svima te napraviti platformu na kojoj drugi umjetnici *pixelart*-a mogu učiti jedni od drugih [7].

Pixilart aplikacija je jednostavna za korištenje te je za početnike bolji izbor nego neki profesionalni programi za crtanje. Iako je aplikacija jednostavna ona sadrži sve što je potrebno za razvoj *pixelart*-a. Mogu se koristiti slojevi, spremati animacije u posebnu datoteku ili ih spremiti kao niz slika, mogu se spremati i posebno napravljene palete boja za ponovno korištenje te osim klasičnih alata za crtanje postoje i dodatni alati kao što je „*Dithering Tool*“ koji se često koristi za pixelart.

3. Žanr igre uloga

Cilj igara uloga je prikazati novi izmišljeni svijet igraču, taj svijet može biti kreiran tako da kontroliramo lika koji se već nalazi u tome svijetu te mi donosimo odluke unutar svijeta za njega te se priča razvija prema tim odlukama npr. Igra „*The Witcher 3: Wild Hunt*“ ili svijet može biti kreiran, a lika mi kreiramo u potpunosti, gdje sami biramo kakav će naš lik biti te ponašanje našeg lika unutar toga svijeta npr. igra „*The Elder Scrolls V: Skyrim*“.

3.1. The Elder Scrolls V: Skyrim

Skyrim je igra smještena u izmišljenom svijetu „*The Elder Scrolls“* svemira gdje igra kreče tako da igračev lik nema niti ime niti izgled te nakon kratkog uvoda u igru kreira svoga lika gdje prvo bira svoj izgled (rasa, kosa, faca,…) te nakon toga započinje svoju avanturu. Nakon toga uvoda igrač je upoznat sa glavnom pričom igre te kreće u avanturu upoznavanja priče i likova koji će se pojavljivati i kasnije u priči. S obzirom da pričamo o igri uloga sa otvorenim svijetom i ne-linearnom pričom igrač ne mora prolaziti kroz glavnu priču nego ju može ignorirati u potpunosti i samostalno istraživati svijet.

Igra ima dobro razvijen sustav zadataka te sam razvoj lika. Umjesto da igrač mora apsolutno sve sam shvatiti kako igra funkcionira igrač zapravo saznaje sve usput u razgovorima unutar igre te zapravo igra nagrađuje igrača za istraživanje svijeta na primjer ako igrač želi kupiti knjige za magiju kod trgovca, trgovac će mu reći gdje se nalazi škola za magiju te igrač može otići tamo da nauči magiju.

Osim što igrač ima slobodu rješavanja zadataka i istraživanja svijeta, igrač ima i slobodu kreiranja načina borbe svog lika, gdje njegov lik može koristiti u borbi sve vrste magije i sva vrsta oružja te kako ih koristi postaje bolji s njima. Svaki način borbe ima svoje prednosti te se mogu i svi načini kombinirati, npr. Igrač može biti ratnik sa mačem i štitom ili čarobnjak sa mačevima. Ovaj način razvoja lika nije čest u igrama nego se većinom igre baziraju na tome da postoji neki broj različitih stilova borbe te igrač odabire stil borbe odmah na početku igre.

3.2. The Witcher 3: Wild Hunt

Iako *Witcher* 3 pripada istom žanru igara kao i *Skyrim*, igre imaju različite fokuse na samu priču te kreiranje lika. Unutar *Witcher*-a 3 igramo lika koji već postoji unutar svijeta te

nemamo izbor mijenjanja njegovog imena ili njegovog lika. Razlika između *Witcher*-a 3 i *Skyrim*-a je velika razlika u samoj priči, u *Skyrim*-u je glavna priča u manjem fokusu nego samo istraživanje svijeta i pronalazak drugim manjih priča unutar svijeta dok je u *Witcher*-u 3 bitnija glavna priča gdje kontroliranom liku pomažete ostvariti njegov cilj te upoznavanjem likova na tome putu si možete olakšati kasnije dijelove priče jer će vam ti likovi pomoći ovisno kako se ponašate prema njima pomoću izbora.

Također je razlika u igrama jer s obzirom da *Witcher* 3 ima već kreiranog lika kojim igrač upravlja igrač nema previše izbora u tome kako se taj lik bori. Sami glavni lik igre je *Witcher* tj. lovac na čudovišta te oni imaju svoj način borbe a to je dva mača, magija i pravljenje napitaka te igrač može nadograditi već postojeće vještine, ali ne može potpuno promijeniti njegov stil.

Unutar *Witcher*-a 3 istraživanje svijeta osim istraživanja dodatne priče svijeta služi i da pripremi igrača za glavnu priču gdje protivnici postaju teži i kompliciraniji za pobijediti. Rješavanjem dodatnih zadataka i istraživanjem svijeta igrač skuplja dodatne materijale kako bi mogao kreirati napitke te kako bi našao jače oružje i jači oklop.

4. Pixelart

Pixelart je poseban način crtanja koji je bio implementiran u starim igrama jer same slike (eng. *sprite*) ne zauzimaju previše memorije. Stare igre su koristile slike veličine 16x16 piksela te su postajale sa vremenom sve veće i veće. Danas se *pixelart* koristi kao posebna vrsta stila iako tehnologija nije ograničena da se *pixelart* mora koristiti.

Novi *game engine*-i moraju biti prilagođeni *pixelart* stilu jer se s novijim tehnologija zapravo pokušavaju izbjeći pikselizirana grafika te se takve postavke moraju isključiti kako bi se zapravo pikseli vidjeli. Osim grafike bitna je veličina slika te njihovo skaliranje unutar *Engine*-a. Prije samo kreiranja svih slika treba se odrediti jedinična mjera u pikselima jer se skaliranjem gubi stil *pixelart*-a, na primjer u ovom projektu jedinična mjera je 64x64 piksela te slika te veličine je jednaka jedinici 1 po visini i širini unutar *Engine*-a. Ukoliko želimo kreirati objekt koji je duplo manje visine a jednake širine njegova slika će biti veličine 64x32 piksela.

Osim tih postavki i veličine također se animacije ne smiju raditi na klasičan način 2D animacija. Klasične 2D animacije se rade slično kao i u 3D animacijama, gdje se uzme određeni dio slike te se pomoću rotacije, pomaka i vremena određuje nova pozicija te *engine* sam napravi pokret. Ukoliko se *pixelart* napravi na klasičan način pikseli neće biti na pravim pozicijama te će se izgubiti izgled *pixelart*-a, zbog toga se mora koristiti drugačiji način izrada animacija gdje se za svaku sliku animacije mora nacrtati nova slika te se izmjenjivanjem slika kroz određeni vremenski period dobije animacija. Klasični način 2D animacija ima „glatke“ animacije ali uništava stil *pixelart*-a gdje je bitnije da stil ostane isti nego da animacije izgledaju glatko.

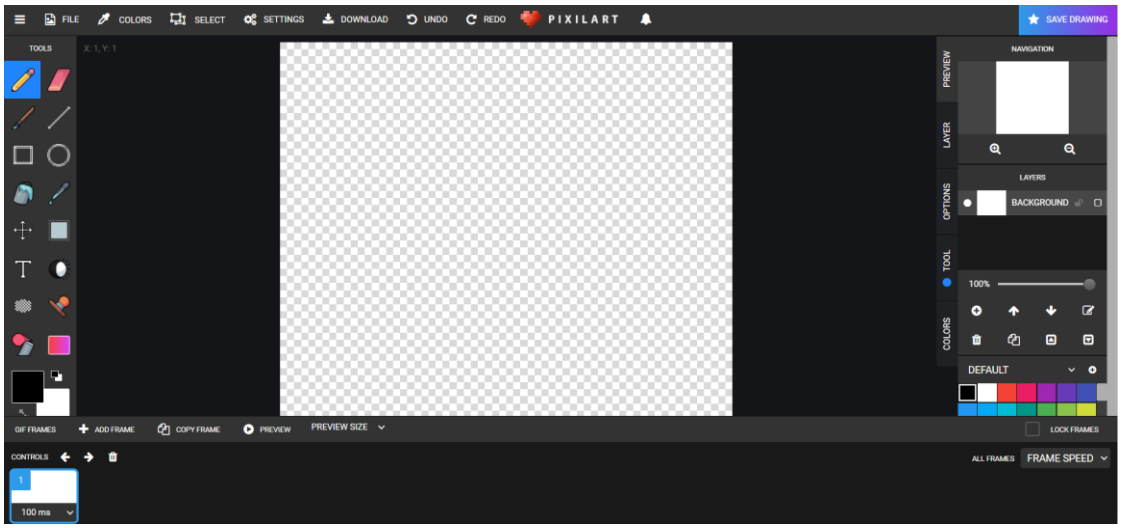
4.1. Izrada

Izrada *pixelart*-a je u teoriji jednostavna ali ovisi o samoj veličini i iskustvu crtača. Slike manjih veličina su teže za nacrtati jer je teško u tako maloj slici prikazati sve što treba. Za početnike je bolje uzeti malo veću podlogu za crtanje. U ovom projektu je korištena veličina 64x64 piksela kao jedinična mjera te su jednostavne ikone u sučelju napravljene veličine 32x32 piksela.

Osim crtanja likova bitno je razmišljati i o tome kako će se ti likovi „ponašati“, na primjer igrač u ovoj igri neće uvijek isto izgledati tj. igrač će moći namjestiti svoj izgled pri kreiranju svoga lika gdje će moći kombinirati drugačiju boju kože, boju očiju, izgled kose i boju kose.

Osim osnovnog izgleda lika također će lik imati i drugačije oklope i oružja kroz igru gdje svi ti oklopi i oružja moraju odgovarati modelu igrača. Zbog toga crtač mora napraviti sve te dijelove igrača odvojeno te će se unutar *Engine*-a ti dijelovi spojiti i kreirati igrača. S obzirom da igrač može odabrati da mu je lik muškog i ženskog spola, koji imaju drugačiju građu tijela, crtač mora za svaki oklop napraviti različite modele za svaki spol.

Pixilart alat nudi opciju gdje možemo crtati po slojevima što pomaže je lakše nacrtati sve potrebne dijelove tijela odvojeno za igru. Te se ti slojevi mogu posebno spremiti u svoje slike te se odvojeno implementirati u *Engine* igre.



Slika 1. Pixilart sučelje

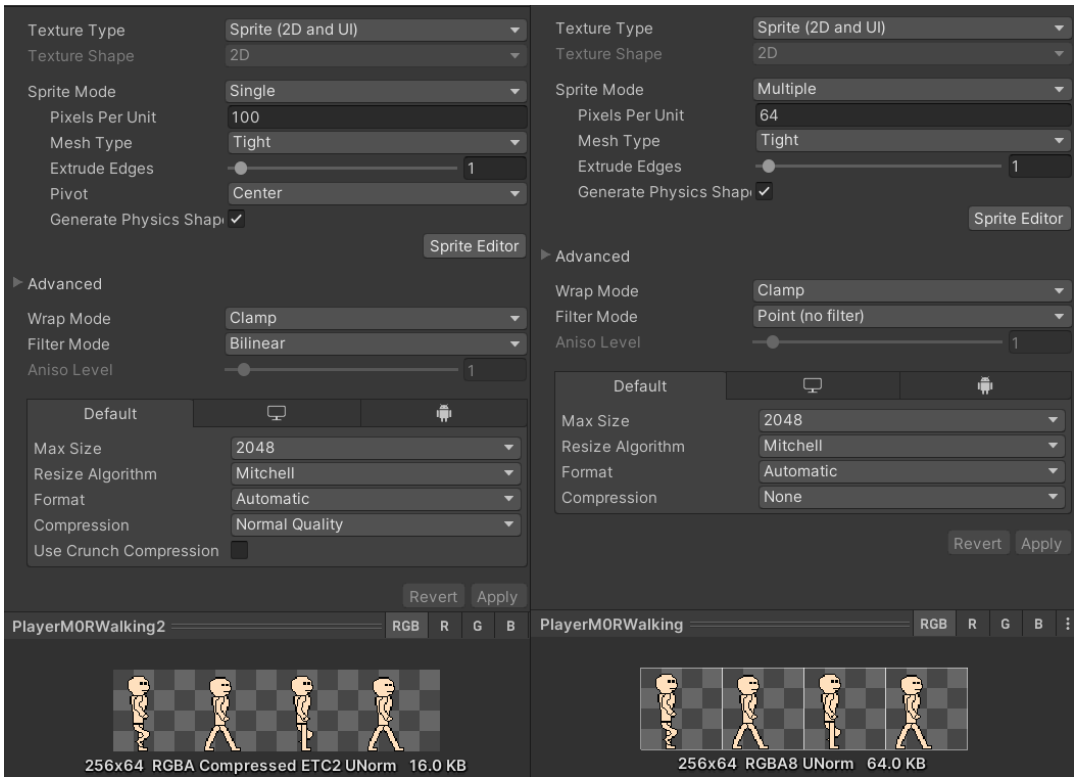
Na slici iznad je prikazan izgled alata Pixilart gdje možemo sa lijeve strane vidjeti sve alate koji se koriste unutar aplikacije, kao što je obična olovka, gumica za brisanje, alat za crtanje linija, alat za ispunjavanje boje,… S desne strane možemo vidjeti prikaz cijele slike, ispod toga slojeve i postavke za kopiranje slojeva, pomicanje,… te se ispod toga može vidjeti paleta boja gdje možemo dodati i spremiti vlastite palete kao datoteku. Traka na dnu aplikacije nam pokazuje slike te pritiskom gumba „*Preview*“ možemo vidjeti animaciju svih slika, također možemo promijeniti i vremenski razmak između svake slike.

Kada se nacrtaju sve slike animacije imamo više načina spremanja te animacije. Možemo ju spremiti sa „*.pixil*“ ekstenzijom te s time možemo vratiti cijeli proces nazad u aplikaciju te nastaviti rad nad animacijom. Osim toga možemo spremiti kao odvojene slike svih slojeva ili možemo spremiti odvojene slike samo određenog sloja. Spremanje odvojenog sloja

je korisno pri kreiranju na primjer novog oklopa gdje možemo na već postojećem modelu igrača samo dodati novi sloj oklopa te spremiti samo oklop.

4.2. Implementacija

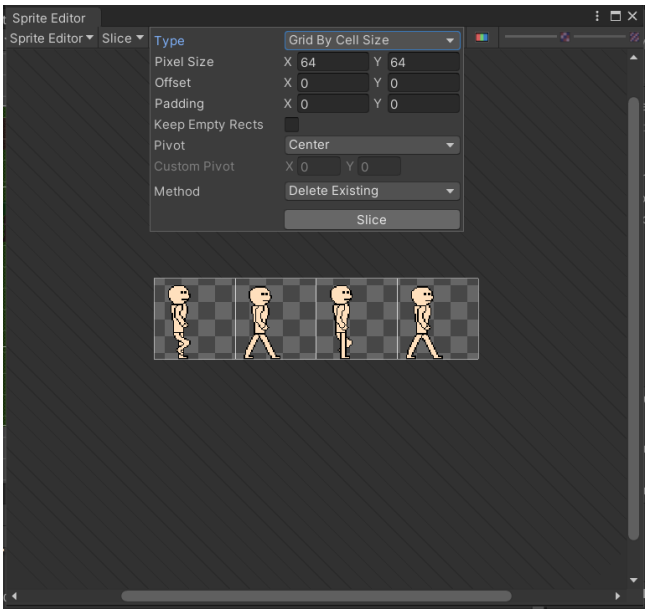
Nakon što je animacija nacrtana najbolje je slike animacije spremiti unutar iste slike gdje ćemo slike redom kojim se animacija odvija posložiti jednu pokraj druge, nebitno je da li je s lijeva na desno ili od gore prema dolje. Na taj način se olakšava spremanje i snalaženje među svim animacijama projekta, jer se te slike mogu odvojiti unutar Unity-a te će jedna animacija biti i dalje spremljena kao jedna datoteka.



Slika 2. Postavke slika za pixelart stil

Na slici iznad možemo vidjeti početne postavke (lijevo) te postavke koje se moraju postaviti (desno) kako bi animacija mogla imati *pixelart* izgled. Postavka „*Sprite Mode*“ ovisi o tome da li je unutar dodane slike nalazi jedna ili više slika, ukoliko je samo jedna postavka se ostavlja na „*Single*“ u suprotnom se postavlja na „*Multiple*“. Postavka „*Pixels per Unit*“ označava jediničnu mjeru slike što je objašnjeno na početku poglavlja, unutar ovog projekta

jedinična mjera je 64x64 te se ova postavka postavlja na 64. Postavka „*Filter Mode*“ se mora postaviti na „*Point (no filter)*“ kako *Engine* ne bi pokušao izgladiti sliku s čime bi ona izgubila *pixelart* izgled. Zadnja postavka koja se mora namjestiti je „*Compression*“ gdje ju moramo postaviti na „*None*“ kako bi se isključila kompresija slike.

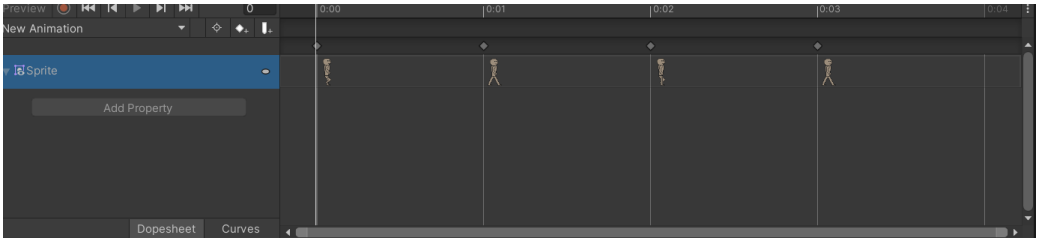


Slika 3. Rezanje slike

Zadnji korak namještanja slika je rezanje slike na dijelove što se radi samo u slučaju da je postavljena postavka „*Sprite Mode*“ na „*Multiple*“. Rezanje slike počinjemo na način da kliknemo gumb „*Sprite Editor*“ u postavkama slike te će nam se otvoriti prozor za uređivanje slike. Na slici iznad možemo vidjeti taj prozor te se klikom na gumb „*Slice*“ u gornjoj traci prozora otvara mali okvir za rezanje slike, odavire se postavka „*Type*“ te se postavlja način određivanja rezanja, u ovom primjeru je korišten način „*Grid by Cell Size*“ gdje mi određujemo veličinu svake slike unutar te će se slika izrezati na dijelove te veličine. Nakon odabira tog načina rezanja trebamo upisati veličinu svake slike, u ovom slučaju 64x64 te dodati pomak rezanja ukoliko je potrebno. Nakon što su postavke namještene samo se klikne gumb „*Slice*“ te će slika biti izrezana na dijelove. Nakon slika možemo vidjeti na koji način će slika biti razrezana s bijelim okvirima prije samog spremanja.

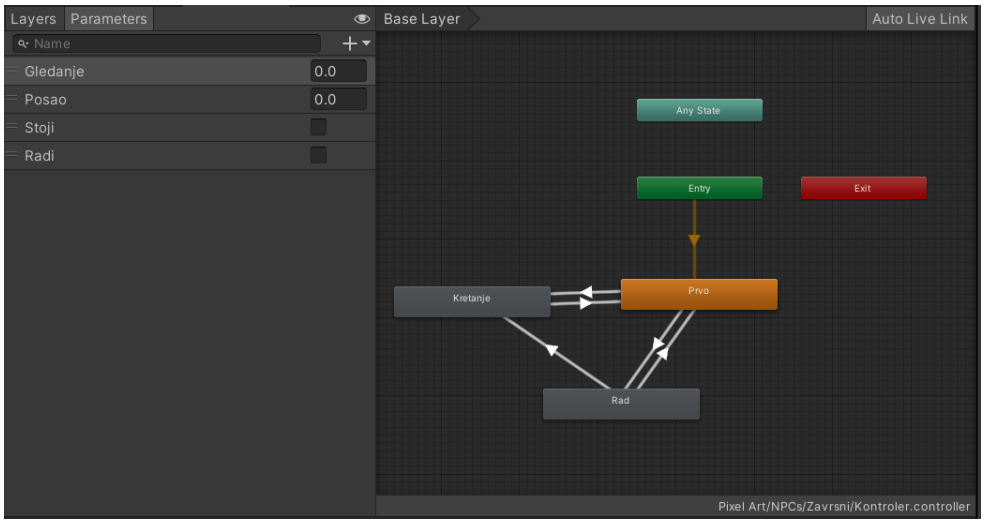
Slika 4. Izrezana slika

Nakon što je slika kreirana možemo pritiskom na strelicu s desne strane slike vidjeti kako je izrezana (slika iznad) te odabirom na sve manje slike te desnim klikom i odabirom „*Create->Animation*“ kreirat će se animacija gdje se izmjenjuju te slike u krug. Pregled animacije možemo vidjeti duplim klikom na kreiranu animaciju te će nam se otvoriti prozor za upravljanje animacijama (slika dolje). Unutar ovoga prozora možemo vidjeti koje se sve postavke objekta mijenjaju sa animacijom (u ovom slučaju samo slika) te kojom se brzinom izmjenjuju. Klikom na „*Add Property*“ možemo dodatne svari mijenjati tijekom animacije kao što je na primjer veličina objekta te možemo i postaviti razmake između svake slike kako bi animacija bila sporija ili brža.



Slika 5. Uređivanje animacije

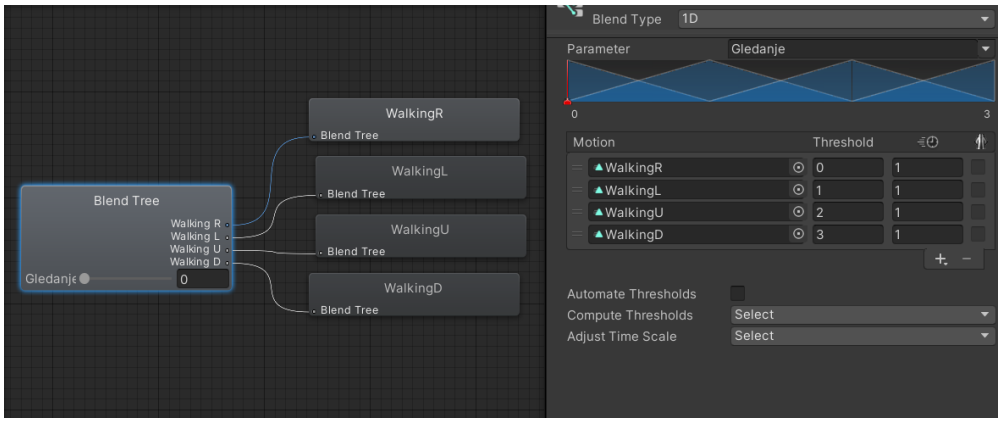
Ukoliko se animacija sama postavi na objekt to je jedina animacija koju će objekt imati te se na taj način implementiraju samo animacije na statične objekte. Ukoliko nam je objekt dinamičan te mu se izmjenjuju animacije ovisno o tome u kojem je stanju objekt potreban nam je „*Animator Controller*“ koji je mašina stanja gdje se ovisno o određenim varijablama i njihovim vrijednostima izmjenjuju animacije. Vrijednosti varijabli se mijenjaju pomoću skripti.



Slika 6. Animator Controller

Na slici iznad možemo vidjeti pozor za uređivanje „*Animator Controllera*“, ovaj kontroler je jednostavni kontroler za animacije NPC-a. S lijeve strane vidimo varijable koje kontroliraju stanje animacija to su „Gledanje“ decimalnog tipa koja određuje u kojem smjeru NPC gleda, „Posao“ decimalnog tipa koja određuje kojim poslom se NPC bavi, „Stoji“ *boolean* tipa koja određuje da li NPC stoji ili se kreće i „Radi“ *boolean* tipa koja određuje da li NPC radi trenutno ili ne radi.

Svaki okvir unutar kontrolera prikazuje određeno stanje, strelice između stanja prikazuju promjene stanja na primjer ukoliko je trenutno stanje „Prvo“ (tj. stajanje) te se varijabla „Stoji“ promjeni na netočno (eng. false), stanje animacije će se pomaknuti na stanje „Kretanje“ te ukoliko se varijabla „Stoji“ vrati na točno (eng. true) stanje će se vratiti na „Prvo“. Osim ručno kreiranih stanja postoje i predefinirana stanja kao što su „*Any State*“, „*Entry*“ te „*Exit*“ koja služe za dodatnu kontrolu. „*Entry*“ se pokreće pri instanciranju objekta te se odmah prebacuje u prvo definirano stanje, u ovom slučaj „Prvo“. „*Any State*“ služi kako bi se u bilo kojem trenutnu mogli prebaciti u neko stanje ukoliko je neki uvjet ispunjen. „*Exit*“ služi za gašenje kontrolera.



Slika 7. Blend Tree

Za jednostavnost organiziranje animacija postoji „*Blend Tree*“ što predstavlja jedno stanje, na slici iznad možemo vidjeti „*Blend Tree*“ za stanje „Kretanje“. To stablo sadrži četiri animacije koje predstavljaju kretanje u četiri različita smjera. Kada se stanje kontrolera prebaci na stanje „Kretanje“ unutar toga stanja u stablu se gleda varijabla „Gledanje“ te ovisno o njenoj vrijednosti (od 0 do 3) se određuje koja animacije će biti pokrenuta. Ovo stablo je jednodimenzionalno, a ukoliko je potrebno mogu se koristiti i dvodimenzionalna stabla gdje se uspoređuju vrijednosti dvije varijable. Stablo se inače koristi i za spajanje animacija ovisno o vrijednosti, ali se u *pixelart*-u to ne smije raditi.

Nakon što je kontroler kreiran on se samo dodaje na objekt te se pomoću posebno kreirane skripte i metoda za upravljanje kontrolerom izmjenjuju stanja te se dobije objekt sa dinamičnim animacijama. Također objekt može imati djecu koje svako ima svoj kontroler te se mogu kontroleri uskladiti što je slučaj za objekt igrača gdje svaki njegov dio tijela te svaki dio oklopa i oružje imaju svoj kontroler, ti kontroleri imaju iste varijable koje se usklađeno izmjenjuju na iste vrijednosti te se time dobiju animacije koje funkcioniraju kao jedna velika animacija.

5. Igra

Cilj ovog projekta je napraviti igru igranja uloga sa većinski otvorenim svijetom gdje su samo dijelovi instancirani (potrebno je učitavanje). Igra mora biti dovoljno jednostavna kako bi se mogla pokrenuti na mobilnim uređajima te treba biti jednostavna za igrati kako bi ju igrači bez iskustva u ovakvim igrama mogli igrati te kako bi i igrači upoznati s ovakvim igrama mogli uživati u njoj.

Prije samoga ulaska u svijet igrač kreira svoga lika te mu određuje izgled i njegovu klasu. Kod izbora klase ima tri opcije, a to su: ratnik, strijelac i čarobnjak. Ratnik je klasa koja koristi mač te se s njim bori i prima manju štetu nego ostale dvije klase jer nosi teški oklop. Strijelac je klasa koja koristi luk i strijele te može lakše izbjegavati protivnike i pobijedit ih iz daljine prije nego protivnici dođu do igrača, strijelac prima veću štetu od ratnika ali manju od čarobnjaka. Čarobnjak je klasa koja koristi magiju unutar borbe, čarobnjak ima najveću kontrolu nad protivnicima gdje ih može zaustaviti na mjestu te im raditi štetu dok bježi, osim toga što radi štetu čarobnjak je jedina klasa koja može sebe izliječiti bez napitaka.

Igra je smještena u državi izmišljenog svijeta fantazije gdje postoji magija i razna mitološka bića. Igrač će od samoga početka krenuti u avanturu glavne priče gdje će proći kroz sve glavne lokacije svijeta. Osim toga što će igrač proći kroz glavnu priču također će imati opciju da dodatno istraži svijet kako bi otkrio dodatne sitne i velike priče s kojima će bolje upoznati svijet, upoznati ljude ovoga svijeta te pronaći oklope i oružja koja ne bi mogao inače pronaći.

5.1. Funkcionalnosti

Neke funkcionalnosti igre su poprilično jednostavne te se zajedničke svim igrama uloga, kao što su leveli i život. Funkcionalnost kontroliranja inventara ovisi od igre do igre, neke igre imaju napravljeno da igrač ima ograničenu težinu nošenja doke neke igre imaju kao broj slobodnih pozicija te svaka stvar zauzima određen broj pozicija. U sljedećim poglavljima će biti opisane sve glavne funkcionalnosti igre te će biti prikazani odsječci koda.

5.1.1. Osnovne funkcionalnosti

Osnovne funkcionalnosti su funkcionalnosti primarno vezane za samoga igrača te su nešto s čime će se igrač susretati od početka do kraja igre. Neke funkcionalnosti su jednostavne kao što su leveli, dok su neke kompliciranije kao što su talenti.

Slika 8. Sučelje igre

Na slici iznad se nalazi slika igre te se može vidjeti sučelje igre. U gornjem lijevom kutu se nalaze osnovni podaci igrača (ime, level, život, mana, iskustvo), te se klikom na sliku lika otvara sučelje inventara. U donjem lijevom kutu se nalazi kontroler za kretanje lika. U donjem desnom kutu se nalaze napadi igrača (tamni su nedostupni), te područje za postavljanje napitaka. U gornjem desnom kutu možemo vidjeti gumb za pauziranje, gumb za otvaranje zadataka i mapu igre, te sat i kalendar.

5.1.1.1. Leveli i iskustvo

Leveli i iskustvo su ključni elementi da igrač postane jači u igri. Kako igrač pobjeđuje protivnike i kako rješava zadatke on skuplja iskustvo. Nakon što skupi određenu količinu iskustva igraču će se povećati level. Igrač može vidjeti svoje trenutno iskustvo i iskustvo potrebno za sljedeći level na glavnom sučelju igre u gornjem desnom kutu u obliku ljubičaste trake te svoj level pokraj imena isto u gornjem desnom kutu.

```
public int Level {
    get { return _level; }
    set {
        _level = value;
        PlayerSaveScript.Level = _level;
        SaveScript.SaveFile();
        PlayerData.UnlockAttack(value);
        attackControl.AttackButtonsUpdate();
    }
}
public int XP {
     get { return _xp; }
     set {
            _xp = value;
            PlayerSaveScript.XP = _xp;
            TrackXP();
            UIControler.UpdateInfo(false, false, true);
    }
}
```

Svojstva „Level“ i „XP“ služe za dodatnu kontrolu nad vrijednostima levela i iskustva igrača. Oba svojstva pri čitanju samo vraćaju vrijednosti dok pri izmjeni vrijednosti spremaju vrijednost u statične varijable skripte za spremanje igre te pri promjeni levela se igra i sprema automatski, osim što se igra sprema također se otključava napad igrača ako treba te se ažurira sučelje za napade. Kada se promjeni vrijednost iskustva osim što se vrijednost sprema također se poziva metoda TrackXP() koja provjerava da li je dovoljno iskustva skupljeno za novi level te se nakon te metode izvršava ažuriranje sučelja za prikaz iskustva.

```
public int XPToLvlUp() {
    return 100 * (int)Math.Pow(Level, 2);
}
private void TrackXP () {
    if (XP>=XPToLvlUp()) {
        _xp -= XPToLvlUp();
        TalentPoints++;
        playerData.MaxHP += playerData.HPPerLvl;
        playerData.BaseHP += playerData.HPPerLvl;
        HP = playerData.MaxHP;
        playerData.MaxResource += playerData.ResourcePerLvl;
```

```
            playerData.BaseResource += playerData.ResourcePerLvl;
            Resource = playerData.MaxResource;
            playerData.PrimaryStat += playerData.PrimaryStatPerLvl;
            PrimaryStat = playerData.PrimaryStat;
            Level++;
        }
    }
```

Metoda XPToLvlUp() računa koliko je iskustva potrebno za novi level, računa tako da trenutni level kvadrira i rješenje pomnoži sa dva. Metoda TrackXP() služi za provjeru da li je skupljeno dovoljno iskustva za novi level. Ako je skupljeno dovoljno to iskustvo se oduzima te se svi ostali podaci igrača povećavaju za određenu brojku koja ovisi o klasi igrača te se život i mana postavljaju na maksimalne vrijednosti.

5.1.1.2. Život

Život je podatak koji je ključan igraču jer ukoliko mu život dođe na 0 igrač umire te će mu se igra učitati zadnji puta kada je spremio igru. Život se smanjuje pri borbi kako igrač primi udarce od protivnika te se smanjuje i izvan borbe ukoliko ima loš efekt na sebi. Život se unutar borbe može vratiti s napicima ili sa napadima koji to rade dok se izvan borbe život sam polako vraća na maksimum. Igrač svoj trenutni život i svoj maksimalni život može vidjeti na glavnom sučelju igre u gornjem desnom kutu u obliku crvene trake.

```
    public override int HP {
        get { return _hp; }
        set {
            if (value > playerData.MaxHP)
                _hp = playerData.MaxHP;
            else if (value <= 0) {
                _hp = 0;
                Velocity = Vector2.zero;
                Death();
            }
            else
                _hp = value;
            playerData.CurrentHP = _hp;
            UpdateHP();
        }
    }
```

Svojstvo „HP“ pri čitanju samo vraća vrijednost života, dok pri postavljanju vrijednosti se prvo provjerava da li je nova vrijednost veća od maksimalne, te ukoliko je postavlja vrijednost „HP“-a na maksimum. Ako vrijednost nije veća od maksimuma nego je manja ili jednaka od 0 vrijednost te postavlja na 0 te se pokreće metoda Death() koja će „ubiti“ igrača. Ako vrijednost nije iznad maksimuma niti ispod minimuma onda samo spremamo vrijednost, te nakon toga spremamo podatak i za spremanje igre te pozivamo metodu UpdateHP() koja nam ažurira sučelje za prikaz života.

```
IEnumerator ResourcesOverTime(){
    while (true){
        if (Resource < playerData.MaxResource)
            Resource += ResourcePerSec;
        if (HP < playerData.MaxHP && !InCombat)
            HP += HPPerSec;
        yield return new WaitForSeconds(1);
    }
}
```

Metoda ResourcesOverTime() služi za regeneriranje mane i života tijekom vremena, metoda je napravljena od beskonačne petlje koja na kraju svakoga kruga čeka 1 sekundu (kada igra nije zaustavljena) te provjerava da li je mana puna, ako nije povećava ju, a za život provjerava da li je manji od maksimuma i da li je igrač izvan borbe, te ako je to točno povećava život.

5.1.1.3. Mana

Mana u igri služi za korištenje napada, kada igrač ostane bez mane ne može koristiti napade. Bitno je kontrolirati manu u igri gdje ju igrač neće prebrzo potrošiti već će ju pravilno regulirati. Mana se za razliku od života regenerira i usred borbe pošto je ključna za borbu. Neki napadi troše više mane dok neki manje ovisi o tome koliko je napad jak. Igrač može vidjeti svoju trenutnu manu i maksimalnu manu u glavnom sučelju igre u obliku plave trake.

```
public override int Resource {
    get { return _resource; }
    set {
        if (value > playerData.MaxResource)
            _resource = playerData.MaxResource;
        else
            _resource = value;
```

```
            playerData.CurrentResource = _resource;
            UpdateResources();
        }
    }
```

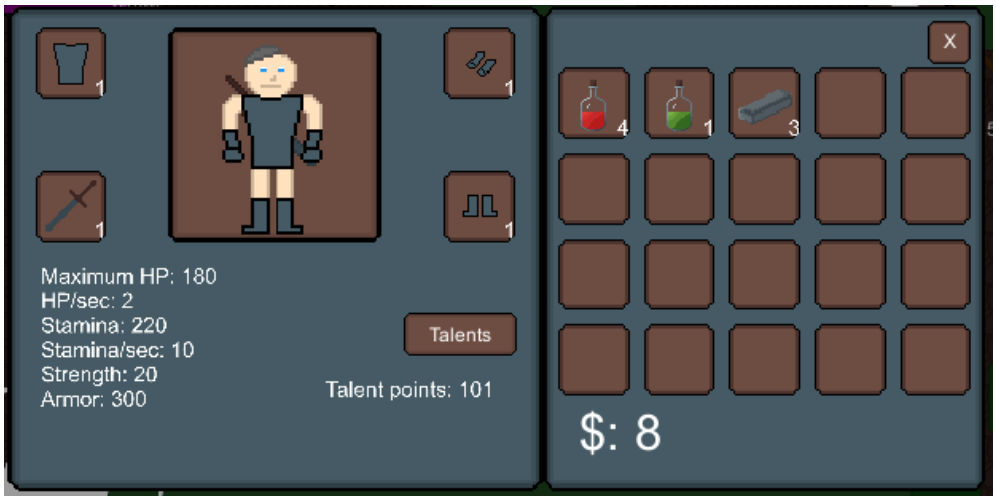
Svojstvo mane funkcionira slično kao i svojstvo za život, razlika ja u tome što se ne provjerava da li je mana pala ispod minimuma jer ne može te se na kraju ne ažurira sučelje za život već se ažurira sučelje za manu.

Za regeneriranje mane kao i za život koristi se metoda ResourcesOverTime() koja je objašnjena u poglavlju ranije.

5.1.1.4. Inventar

Inventar je napravljen u stilu inventara igre „*World of Warcraft*“ gdje igrač ima ograničen broj mjesta (u slučaju ove igre ima 20 mjesta) gdje svaki predmet zauzima jedno mjesto. Svako to mjesto, ovisno o predmetu, može sadržavati više istih predmeta na primjer svaki oklop i svako oružje samo zauzima jedno cijelo mjesto dok na primjer možemo spremiti 20 napitaka na jedno mjesto inventara dokle god su napici istoga tipa. Svaki predmet u inventaru je napravljen kao posebna lista predmeta što je realizirano posebnom skriptom koja sadrži tip predmeta, ID predmeta i količinu predmeta te su sve te liste predmeta spremljene u dodatnu listu tih predmeta što čini inventar.

Lista svi predmeta tj. inventar je realiziran kao posebna klasa koje nasljeđuje List<T> te ima svoje posebne metode. Te metode su AddItem(ItemList item) i RemoveItem(ItemList item). Dodavanje predmeta funkcionira na način da se prvo provjerava da li postoji taj predmet u inventaru, ako postoji samo količinu toga predmeta povećavamo do toga koliko trebamo ili do maksimuma. Ako smo prešli maksimum predmeta u jednom mjestu dodajemo novu listu predmeta u inventar sa preostalim predmetima ili ih sve dodajemo ukoliko taj predmet ne postoji u inventaru. Na kraju metoda vraća netočno (eng. *false*) ako nismo uspjeli staviti predmet u inventar ili točno (eng. *true*) ako jesmo. Micanje predmeta iz inventara funkcionira da tražimo taj predmet u inventaru te mu smanjujemo količinu, ako količina jednog mjesta nije dovoljna tražimo sljedeće mjesto istog predmeta te od tamo nastavljamo micati dok se ne makne tražena količina.



Slika 9. Sučelje inventara

Inventar se otvara klikom na sliku lika na glavnom sučelju igre u gornjem desnom kutu. Lijeva strana inventara nam prikazuje sliku lika, oružje i oklope koji su trenutno na liku, osnovne podatke te gumb za otvaranje talenata. Sa desne strane možemo vidjeti inventar lika sa svim predmetima u njemu i njihovom količinom te količinu novca koju lik ima.

5.1.1.5. Oklopi i oružja

Oklopi i oružja nasljeđuju klasu „*Item*“ koja nasljeđuje klasu „*ScriptableObject*“ od Unity *Engine*-a. Klasa „*ScriptableObject“* nam služi kako bi instance klasa mogli napraviti kao same datoteke unutar igre te ih na taj način koristiti. Klasa „*Item*“ sadrži osnovne podatke koji su isti za sve predmete unutar igre kao što je ID predmeta, ime predmeta, tip predmeta, cijena predmeta,… Klase za oklope i oružje sadrže dodatne podatke koji su posebni samo za njih a ne za sve predmete, na primjer klasa za oružje sadrži podatak za štetu koju radi oružje, dok klasa za oklope sadrži podatak koliku zaštitu pruža taj oklop.

Oružja unutar igre služe za napade igrača, jače oružje znači da ima veću štetu te će samim time napadi igrača raditi veću štetu protivnicima. Bez oružja igrač ne može niti koristiti napade te su oružja najbitnija za napad. Oklopi unutar igre služe za obranu igrača, jači oklopi znače više zaštite te će igrač primati manju štetu od protivnika. Igrač ima 3 dijela oklopa a to su oklop za tijelo, rukavice i cipele. Ukupna zaštita od napada se dobije zbrajanjem vrijednosti zaštite svih oklopa. Šteta koju protivnik radi igraču se računa funkcijom ispod.

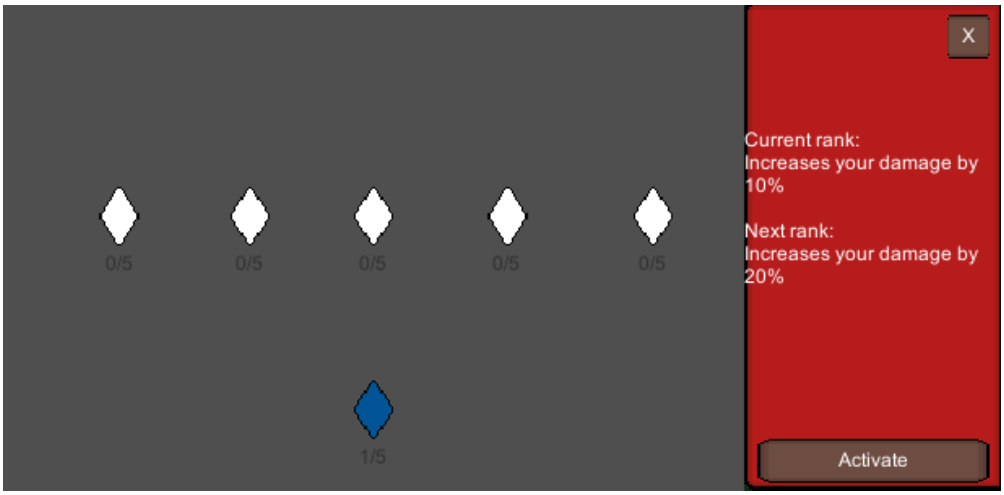
```
public static int CalculateDamage(int enemyDamage) {
    int dmg = 0;
    int armor = PlayerScript.PlayerInfo.ArmorStat +
PlayerScript.PlayerInfo.BonusArmor;
```

```
        float percentage = 1.0f - ((float)(armor > 1800 ? 1800 : armor) /
2000f);
        dmg += Mathf.RoundToInt((float)enemyDamage * percentage);
        return dmg;
    }
```

Funkcija prima brojku koju protivnik treba napraviti igraču, zatim se računa ukupna zaštita igrača zbrajanjem ukupne zaštite od oklopa i dodatnom zaštitom ako ju igrač ima. Nakon toga se od broja 1 oduzima zaštita podijeljena sa 2000 (zaštita ne može biti veća od 1800, na taj način će protivnik napraviti minimalno 10% štete). Nakon što je to izračunato dobije se postotak ukupne štete koju će protivnik napravit te se ukupna šteta mjeri sa izračunatim postotkom i to je brojka koju funkcija vraća.

5.1.1.6. Talenti

Talenti su bodovi koji ojačavaju igrača, ti bodovi se dobiju svaki puta kada igrač dobije jedan level. Talenti imaju 3 kategorije, napad, obrana i ostalo, svaka klasa lika ima drugačije talente (neki su im zajednički). Ti talenti ojačavaju igrača na različite načine te neki talenti imaju više razina kao na primjer prvi talent u napadu za ratnika je da radi 10% vise štete te ima 5 razina i može ako potroši 5 bodova na taj talent imati 50% više štete. Neki talenti su specifični na određene napade te će igrač trošiti bodove ovisno o tome na koji način igra i ono što misli da mu je potrebnije u tome trenutku. Bodovi su ograničeni te će igrač morati dobro organizirati svoje bodove.



Slika 10. Sučelje talenata

Na slici iznad možemo vidjeti sučelje za biranje talenata. Do sučelja se dolazi klikom na gumb „*Talents*“ u sučelju inventara te odabiranjem kategorije talenata. Unutar sučelja možemo vidjeti sve talente te kategorije i njihove maksimalne razine. Klikom na talent nam se s lijeve strane prikazuje opis trenutne razine i slijedeće razine odmah ispod. Klikom na gumb „*Activate*“ se igraču troši jedan bod i talentu se povećava razina.

5.1.2. Resursi igrača

Resursi igrača su predmeti i svari na koje će igrač pronalaziti tijekom igre te će mu služiti ta kreiranje novih predmeta ili pronalazak jačih stvari nego što trenutno posjeduje. Neke stvari su beskorisne same po sebi te ih treba predati likovima unutar igre kako bi mogao kreirati nešto novo.

5.1.2.1. Novac

Novac je osnova za kupovinu novih stvari i za kreiranje stvari. Novac se može dobiti ubijanjem protivnika, prodajom stvari te pronalaskom u škrinjama. Svaki predmet unutar igre ima svoju vrijednost, ta vrijednost je niža prilikom prodaje predmeta dok je vrijednost viša prilikom kupovine predmeta tako da treba biti pažljiv koje predmete će igrač čuvati a koje će prodavati. Zbog toga postoji talent u kategoriji ostalo za svaku klasu igrača koji poboljšava cijene gdje će predmeti pri prodaji imati veću vrijednost a pri kupovini imat manju vrijednost.

5.1.2.2. Resursi

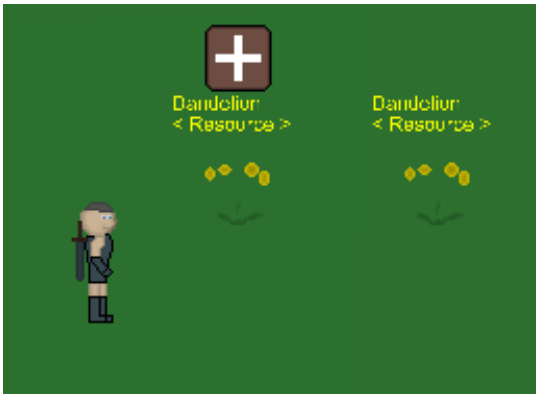
Resursi su predmeti unutar igre koje igrač može pronaći na više načina, ubijanjem protivnika, pronalaskom u svijetu, pronalaskom u škrinjama i kupovinom od likova unutar igre. Ti resursi pripadaju u više kategorija ovisno koja im je svrha, na primjer biljke koje se mogu pronaći u svijetu služe za kreiranje napitaka te se mogu skupiti i predati likovima u igri te će oni kreirati napitke za igrača. Osim napitaka igrač može skupljati na primjer kožu te dati likovima da kreiraju oklope od kože za igrača.

Resursi koji se mogu pronaći u svijetu kao što su biljke i metali imaju svoju posebnu skriptu. Ta skripta osim što sadrži podatke samog predmeta sadrži i vrijeme koje je potrebno da se predmet ponovno pojavi.

```
public void StartRespawn(int start) {
    sr = GetComponent<SpriteRenderer>();
    Color clr = sr.color;
    clr.a = 0;
```

```
        sr.color = clr;
        Canvas.Hide();
        StartCoroutine(Respawn(start));
    }
```

Funkcija iznad se pokreće nakon što igrač pokupi resurs u svijetu. Funkcija gasi komponente objekta i pravi ga prozirnim kako ga igrač više ne bi vidio a kako bi objekt mogao nastaviti odbrojavati kada će se ponovno pojaviti. Nakon što je objekt skriven pokreće se „*Coroutine*“ koji odbrojava kada se resurs treba ponovno pojaviti gdje će se nakon što vrijeme istekne ponovno moći objekt vidjeti.



Slika 11. Resursi u svijetu

Kada se igrač dovoljno približi resursu pronađenom u svijetu pojavljuje mu se gumb iznad resursa da ga pokupi i doda u inventar. Također mu iznad samoga resursa piše i njegovo ime kako bi igrač znao što uopće skuplja.

5.1.2.3. Škrinje

Škrinje su objekti koji se mogu pronaći u svijetu ili u tamnicama gdje one imaju svoj inventar koji igrač može pokupiti. Škrinje imaju statični inventar te se za razliku od resursa ne pojavljuju ponovno već kada se jednom pokupe predmeti iz škrinje oni se neće više pojaviti. Škrinje sadrže velike količine predmeta te su primarni način jeftinog nabavljanja jačih predmeta i veće količine novca unutar igre. Zbog toga škrinje nisu lako dostupne nego se većinom igrač mora „probiti“ do njih kroz veću količinu protivnika.

Slika 12. Sučelje škrinje

Na slici iznad možemo vidjeti sučelje škrinje nakon što ju otvorimo. S lijeve strane se nalazi inventar igrača, a sa desne strane inventar škrinje. Ukoliko je igraču pun inventar a želi predmete iz škrinje, može ostaviti svoje predmete u škrinji. Desni dio sučelja sadrži dva gumba, „*Take money*“ i „*Take all*“, prvi gumb će prebaciti samo novac iz škrinje igraču, a drugi gumb će prebaciti i predmete i novac iz škrinje igraču.

5.1.3. Borba

Borba je jedna od ključnih stvari igri uloga jer igrač u istraživanju svijeta nailazi na različite protivnike koji imaju različite napade. Neki protivnici su jači od drugih ali ovi slabiji većinom će se boriti u većim količinama odjednom. Borba u ovoj igri funkcionira u 4 smjera gdje se svi napadi izvršavaju prema gore, dolje, lijevo i desno. Te će se igrač morati dobro pozicionirati kako bi udario protivnike. Također će se i protivnici pomicati kako bi mogli udariti igrača.

5.1.3.1. Napadi

Svaka klasa igrača ima maksimalno 5 različitih napada na raspolaganju. Svaki taj napad funkcionira na različite načine, na primjer neki napadi samo naprave štetu protivniku dok neki mogu udariti više protivnika ili mogu staviti negativni efekt na protivnika. Osim napada koji samo rade štetu protivnicima postoje i neki napadi koji funkcioniraju na drugačiji način, na primjer strijelac ima napad da postane nevidljiv te ga protivnici ne mogu udariti ili čarobnjak ima napad da sebe izliječi.

Napadi unutar igre imaju više komponenti, a to je funkcija za pokretanje napada, područje u kojem je napad aktivan (eng. *HitBox*) i funkcija za računanje štete. Svi napadi se skaliraju sa jačinom oružja i dodatnim podacima igrača, ali osim toga neki napadi se bolje skaliraju od drugih, na primjer obični udarac ratnika se skalira sa cijelom štetom dok se na primjer udarac ratnika koji može udariti sve protivnike oko igrača se skalira sa jednom trećinom štete pošto može udariti više protivnika.

```
public override void Attack0(bool mobile = false) {
    if (CheckActive() && AttacksData[0].AttackUnlocked) {
       if (Inventory.Weapon == null)
            throw new GameError("You dont have weapon equiped");
       List<Effect> effectList = new List<Effect>();
       Talent bleed = Talent.SearchByStat(UpgradeStats.Attack0Bleed);
        if (bleed.CurrentRank > 0)
            effectList.Add(new Effect("Bleed", 5.0f, bleed.Strength,
EffectStat.HP, false));
        Resource = Attacks[0].Attack.Cast(Looking, Resource);
    }
}
```

Funkcija iznad se pokreće aktiviranjem napada, pri pokretanju funkcije prvo se provjerava da li je neki drugi napad već aktivan i da li je taj napad otključan ili ne. Nakon toga se provjerava da li igrač ima oružje na sebi, ako nema pojavljuje se ispis greške igraču. Ako ima oružje funkcija se nastavlja te se u ovome slučaju provjerava da li postoji talent za napad koji stavlja negativni efekt na protivnika, ako postoji kreira instancu toga efekta i sprema ga u listu efekata. Nakon toga se poziva funkcija toga napada za kreiranje *HitBox*-a ovisno u koju stranu igrač napada i koliko mane ima. Nakon toga funkcija ili vraća grešku da igrač nema dovoljno mane ili vraća podatak koliko mane treba imati nakon što se kreira *HitBox*.

```
public override int Cast(LookingSide looking, int resource = -1, float
speed = 2.0f) {
    if (!CheckCondition(resource))
        throw new System.Exception("Cant cast!");
    AnimationControler.UpdateAnimatorAttack(Data.AnimationID,
Data.ActiveTime);
    resource -= Data.AttackCost;
    GameObject attack = InstantiateHitBox();
    attack.GetComponent<HitBox>().Attack = this;
    switch (looking) {
        case LookingSide.Up:
```

```
                attack.transform.Rotate(0.0f, 0.0f, 90.0f, Space.Self);
                break;
            case LookingSide.Left:
                attack.transform.Rotate(0.0f, 0.0f, 180.0f, Space.Self);
                break;
            case LookingSide.Down:
                attack.transform.Rotate(0.0f, 0.0f, 270.0f, Space.Self);
                break;
        }
        StartCoroutine(Cooldown(Data.AttackCooldown));
        return resource;
    }
```

Funkcija iznad služi za kreiranje *HitBox*-a napada i vraća brojku koliko mane troši napad. Funkcija prvo provjerava da li igrač ima dovoljno mane da iskoristi napad, ako nema funkcija baca grešku koju igrač može vidjeti. Ako igrač može odraditi napad računa se koliko će igraču ostati resursa nakon odrađenog napada, instancira se *HitBox* napada i skripti *HitBox*-a se spremaju podaci napada, Nakon toga se instanciranom *HitBox*-u mijenja rotacija ovisno o tome u kojem smjeru gleda igrač te se nakon toga pokreće *Coroutine* za odbrojavanje kada će napad moć biti ponovno iskorišten i vraća se vrijednost mane.

```
    private void OnTriggerEnter2D(Collider2D collision) {
        if (CheckCondition(collision.gameObject)) {
            DamageTarget(collision.gameObject);
            Attack.Cancel(false);
            Destroy(gameObject);
        }
        else if (collision.CompareTag("Environment")) {
            Attack.Cancel(false);
            Destroy(gameObject);
        }
    }
```

Funkcija iznad se pokreće kada neki objekt uđe u *HitBox*, prvo se provjerava da li je objekt pravoga tipa (ako igrač napada da li je meta protivnik i ako protivnik napada da li je meta igrač), ako je objekt pravoga tipa radimo štetu meti, poslije toga prekidamo napad kako ne bi ostao aktivan te uništavamo *HitBox*. Ako meta nije točna i ako smo udarili zid ili nešto slično što ima oznaku „*Environment*“ napad prekidamo i uništavamo *HitBox*.

5.1.3.2. Efekti

Efekti su funkcionalnost igre koja ima pozitivne i negativne strane. Efekti utječu na igrača i na protivnika te traju određeno vrijeme. Efekti utječu na razne stvari objekta ovisno o samom efektu, na primjer efekt može tijekom vremena smanjivati život objektu ili mu usporiti kretanje.

Varijable efekta su naziv, trajanje, snaga, tip efekta i da li je efekt negativan ili pozitivan. Objekti mogu dobiti efekte od nekoga napada ili od napitaka, na primjer prvi napad od ratnika može napraviti da protivnik krvari te će mu se s vremenom smanjivati život dok na primjer napad od čarobnjaka koji ga liječi mu može povećati zaštitu na neko vrijeme. Igrač može vidjeti na sučelju sve efekte koji su trenutno aktivni na njemu te je bitno paziti na te efekte.

Slika 13. Aktivni efekti u sučelju

Svi efekti su prikazani u glavnom sučelju. Na slici iznad možemo vidjeti prikaz glavnoga sučelja gdje igrač ima negativni efekt na sebi koji traje još 5 sekundi i smanjuje mu život svake sekunde.

5.1.3.3. Napici

Napici služe igraču kako bi bio jači tijekom borbe, svaki napitak ima svoju snagu i svoj tip. Svaki tip napitka ima više razina gdje njegov efekt postaje jači. Na primjer najslabiji napitak za liječenje vraća igraču 10% života, srednji napitak za liječenje vraća 25% života, a veliki

napitak za liječenje vraća 75% života. Neki napici postavljaju efekt na igrača gdje mu na primjer povećavaju zaštitu na neko vrijeme. Ovisno o stilu igranja igrača igrač će koristiti različite vrste napitaka, na primjer za čarobnjaka se ne isplati napitak za povećanje zaštite jer se skalira po trenutnoj zaštiti koju čarobnjak ima nisku.

5.1.3.4. Protivnici

Protivnici su uvijek agresivni prema igraču te će ga automatski napasti čim se igrač dovoljno približi. Kada se igrač dovoljno približi protivnik će se kretati prema igraču i namjestiti će se da može udariti igrača. Ukoliko igrač krene bježati od protivnika protivnik će ga pratiti do određene udaljenosti, kada se igrač udalji previše protivnik će se vratiti na svoju prvobitnu lokaciju i život će mu se vratiti na maksimum.

Svaka vrsta protivnika ima drugačije napade koje će tijekom borbe iskoristiti. Neke napada će koristiti kad god može dok će neke napade kao što su napadi za vraćanje života iskoristiti kada mu će život nizak.

Ovisno o tome koji je protivnik u pitanju neki protivnici će se vratiti u život nakon nekoga vremena slično kao i resursi dok će neki protivnici ostati zauvijek mrtvi. Protivnici koji se neće više oživjeti su protivnici unutar tamnica ili neki jedinstveni protivnici unutar svijeta koji daju posebne predmete kako ih igrač ne bi mogao beskonačno ubijati.

5.1.3.5. Dobivanje stvari

Nakon što je protivnik ubijen on će ostaviti predmete za igrača koje igrač može pokupiti, ti predmeti se nalaze na podu te kada igrač bude blizu njih automatski će ih pokupiti te ukoliko nema mjesta u inventaru za njih oni će ostati na podu neko vrijeme. Zlato koje protivnici ostave će se automatski dodati igraču.

```
protected void DropItems() {
    foreach (var item in ItemDrops) {
        if (MathScript.RandomResult(item.DropChance)) {
            GameObject dropped = Instantiate(DroppedItem);
            ItemScript itmScr = dropped.GetComponent<ItemScript>();
            itmScr.Item = new ItemList(item.ItemType, item.ItemID,
MathScript.RandomNumber(1, item.Quantity), item.DropChance);
            itmScr.Position = gameObject.transform.position;
        }
    }
}
```

Funkcija iznad se aktivira kada protivnik umre te funkcija služi za ostavljanje predmeta koje igrač može pokupiti. Funkcija prolazi kroz sve predmete koje protivnik može ostaviti igraču te nakon toga računa nasumičnu šansu da li će protivnik ostaviti predmet ili ne, jer svaki od predmeta ima svoju šansu da bude ostavljen od 1% do 100%. Ako je šansa pogođena instancira se objekt na podu te se tome objektu dodaje lista predmeta sa nasumičnom količinom od 1 do maksimuma, te se na kraju taj objekt postavlja na poziciju gdje je protivnik umro.

5.1.4. Likovi

Likovi u igri su dio izmišljenoga svijeta i svaki lik ima svoje ime i svoje priču. Ti likovi služe da usmjeravaju igrača sa zadacima te mu služe za razmjenu i kreiranje predmeta. Osim toga igrač može pričati i sa likovima kako bi otkrio skrivene zadatke ili saznao priču svijeta. Likovi se primarno mogu nalaziti u selima i gradovima te nekolicina njih i u otvorenom svijetu. Ti likovi imaju svoj tip kao na primjer kovač koji može prodati predmete igraču koji su potrebni za ratnika i može kreirati predmete potrebne za ratnika.

Slika 14. Lik u svijetu

Kada se igrač približi liku pojavljuju se četiri gumba za aktivnosti s tim likom (ako lik nema aktivnosti gumbi se ne pojavljuju). Prvi gumb su zadaci koje lik ima za igrača, drugi gumb je otvaranje trgovine s likom te trgovina ovisi o tome kojega je tipa lik (u ovome slučaju je alkemičar te prodaje napitke). Treći gumb je za otvaranje sučelja za kreiranje predmeta te četvrti gumb je za pokretanje razgovora sa likom.

5.1.4.1. Trgovina

Trgovina je bitna za igrača jer služi da se igrač riješi predmeta koji mu nisu potrebni te da zaradi time novac. Osim toga igrač može i kupovati predmete od likova kako bi si nabavio potrebne predmete. Likovi imaju nasumične predmete u trgovini te se predmeti osvježavaju nakon 2 dana u igri te predmeti koji će se dodati ovise o levelu igrača.

Svaki lik ima svoj inventar predmeta te mu igrač može prodavati predmete dokle god lik ima mjesta u inventaru. Osim samog mjesta u inventaru bitno je i koliko lik ima novca jer nakon što lik ostane bez novca na može više kupiti predmete od igrača.

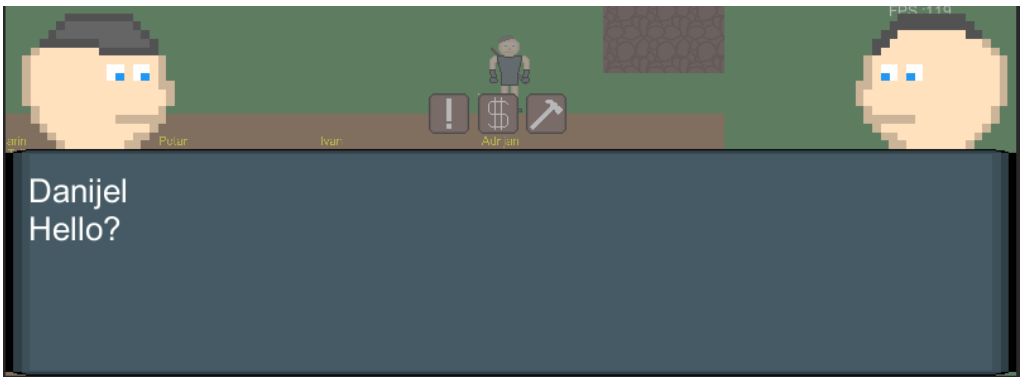
Slika 15. Sučelje trgovine

Na slici iznad možemo vidjeti sučelje trgovine sa likom. S lijeve strane se nalazi inventar igrača, a sa desne strane se nalazi inventar lika. Klikom na predmet igrača možemo vidjeti njegov opis, cijenu i gumb za prodaju, ukoliko igrač proda predmet dobiti će novac u vrijednosti predmeta. Klikom na predmet lika se također pojavljuje opis i cijena te gumb za kupnju, ukoliko se pritisne gumb igrač će platiti toliko novca liku i dobiti predmet. Ukoliko igrač pri kupnji nema novca transakcija se neće izvršiti, isto vrijedi i ukoliko igrač prodaje predmet a lik nema novca za kupiti predmet.

5.1.4.2. Pričanje

Pričanje sa likovima služi igraču kako bi mogao riješiti određene zadatke, kako bi otkrio nove zadatke ili kako bi samo popričao sa likovima da sazna nešto o samoj priči svijeta. Svaki lik u igri ima minimalno jedan dijalog gdje pozdravlja igrača ili nešto slično. Osim tih

ponavljajućih dijaloga lik može imati dijaloge potrebne za zadatke ukoliko je pričanje s tim likom potrebno za zadatak i ako je taj zadatak aktivan. Također može imati i dijaloge koji će otključati dodatne zadatke kod toga lika. Dijalog se pomalo učitava igraču i te se pritiskom može učitati cijeli, za razliku od nekih igara u ovoj igri se dijalozi igrača isto učitavaju sami tj. igrač ne bira sam dijaloge.



Slika 16. Sučelje dijaloga

Na slici iznad možemo vidjeti sučelje za razgovor sa likom. S lijeve strane se prikazuje glava igrača koja gleda u desno, a sa desne strane glava lika s kojim igrač priča koja gleda u lijevo. Prvi red teksta prikazuje ime osobe koja trenutno priča (u ovome slučaju igrač), a linije teksta ispod ispisuju trenutni dijalog. Pritiskom bilo gdje na ekranu se prelazi na slijedeći dijalog.

5.1.4.3. Zadaci

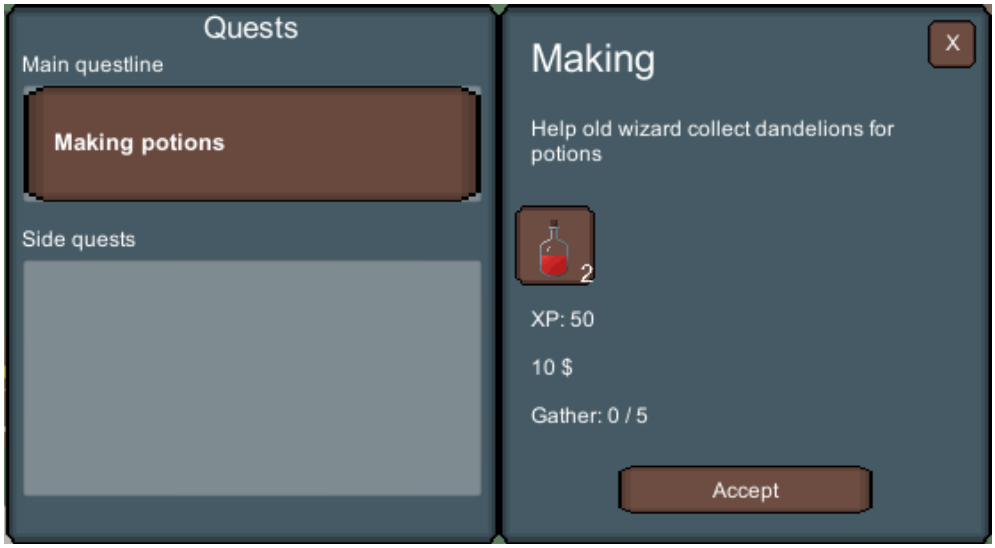
Zadaci su dio igre koji usmjerava igrača za prolazak kroz glavnu priču igre, manjih priča igre i za dobivanje dodatnih stvari, iskustva i novca da bi igrač postao jači. Postoje tri različita tipa zadataka, a to su zadaci gdje igrač treba ubiti određenog protivnika, zadatak gdje igrač treba pričati s određenim likom i zadatak gdje igrač treba skupiti određeni predmet. Nakon što igrač prihvati zadatak on postaje aktivan te se prikazuje igraču na mapi gdje mora ići da bi riješio zadatak, nakon što je zadatak riješen igrač se mora vratiti nazad do lika koji mu je dao zadatak kako bi ga završio i dobio nagrade.

Osim toga što zadatak ima određenu misiju i nagrade također zadaci i otključavaju druge zadatke čime se stvara serijal zadataka koji prolazi kroz priču. Svaki zadatak može imati listu zadataka koje otključava te nakon završetka zadatak se ti zadaci otključavaju. Osim

otključavanja zadataka rješavanjem drugih zadataka, oni se mogu otključati i sa razgovorima sa likovima gdje će likovi usmjeriti igrača prema zadatku.

```
public void FinishQuest(){
    State = QuestState.Finished;
    PlayerScript.PlayerInfo.XP += XPReward;
    PlayerScript.PlayerInfo.Currency += CurrencyReward;
    foreach (Quest quest in UnlockableQuests) {
        quest.Unlocked = true;
    }
    if (ItemReward != null) {
        foreach (ItemList item in ItemReward) {
            PlayerInventory.AddItemToInventory(item);
        }
    }
}
```

Funkcija iznad se pokreće u trenutku kada igrač završi zadatak kod lika te se stanje toga zadataka postavlja da je završen, igraču se daje iskustvo i novac, nakon toga se petljom čitaju svi zadaci koji trebaju biti otključani te se otključavaju i poslije toga ukoliko postoje nagrade predmeta i njih predajemo igraču.



Slika 17. Sučelje zadataka

Na slici iznad možemo vidjeti sučelje za zadatke. S lijeve strane možemo vidjeti popis svih zadataka koji su raspoređeni u dvije kategorije, zadaci glavne priče i zadaci manjih priča.

Klikom na gumb zadatka se otvara opis zadatka sa desne strane. U opisu zadatka možemo vidjeti njegovo ime, opis, predmete za nagradu, iskustvo koje će igrač dobiti, novac koji će igrač dobiti te količinu koju mora obaviti da bi zadatak bio riješen. Ispod toga se nalazi gumb za prihvaćanje zadatka, odbacivanje zadatka ili završetak zadatka (ovisno u kojem stanju je zadatak).

5.1.4.4. Kreiranje stvari

Kreiranje stvari olakšava igraču dobivanje predmeta koji su teški za dobiti ili preskupi za kupiti. Kreiranjem stvari igrač može skupiti resurse unutar igre ili ih kupiti od likova te time kreirati napitke, jače oklope,… Sami recepti za kreiranje stvari se otključavaju zajedno sa levelom igrača te će igrač kasnije u igri moći kreirati jače stvari, ali te stvari će naravno biti i skuplje. Nakon što igrač skupi stvari potrebne za kreiranje predmeta mora ih donijeti do lika i platiti mu za izradu. Nakon toga lik kreće raditi na proizvodnji predmeta koja traje neko vrijeme (neki predmeti se duže kreiraju od drugih) i kada proizvodnja završi igrač može doći do istog lika i pokupiti predmet.

```
IEnumerator Crafting() {
    while (true) {
        yield return new WaitForSecondsRealtime(1);
       AnimationController.UpdateAnimatorBoolean(BoolAnimParams.Working,
false);
        foreach (CraftRecipe recipe in ActiveCrafting) {
            if (recipe.CraftTime > 0) {
                recipe.CraftTime--;
       AnimationController.UpdateAnimatorBoolean(BoolAnimParams.Working,
true);
                if (recipe.CraftTime < 0) {
                    recipe.CraftTime = 0;
                }
            }
        }
    }
}
```

Funkcija iznad se nalazi u skripti za likove i funkcija ja uvijek upaljena te se kod funkcije ponavlja u beskonačnoj petlji. Na početku svakog kruga petlje funkcija prvo čeka jednu sekundu te se onda izvršava dalje. Nakon toga ažurira stanje animatora da lik ima animaciju kako ne radi, te poslije toga ulazimo u petlju koja prolazi kroz sve aktivna kreiranja predmeta.

Za svako aktivno kreiranje ako nije gotovo smanjujemo mu vrijeme kreiranja za 1 jednu sekundu i postavljamo stanje animatora da lik ima animaciju kako radi te ako je vrijeme kreiranja manje od 0 postavljamo ga na 0.

Slika 18. Sučelje kreiranja predmeta

Na slici iznad možemo vidjeti sučelje za kreiranje predmeta. S lijeve strane možemo vidjeti sve predmete koje možemo kreirati kod trenutnog lika. Klikom na gumb s lijeve strane se otvara opis kreiranja s desne stane. S desne strane možemo vidjeti predmet koji ćemo kreirati, resurse koji su nam potrebni za kreiranje, cijena kreiranja i vrijeme kreiranja (u sekundama) te gumb za pokretanje kreiranja. Ako je predmet kreiran umjesto gumba za kreiranje će se nalaziti gumb za uzimanje predmeta.

6. Zaključak

Izrada igara je dugotrajan posao gdje su potrebne različite vještine kao što je programiranje, dizajniranje sučelja, dizajniranje igre,… Srećom postoje razne pomoći na internetu i razne trgovine sa besplatnim stvarima koje olakšavaju izradu igara za početnike. Alat Unity olakšava učenje početnicima sa svojim tečajevima, jednostavnim sučeljem za razvoj, mnogim alatima unutar *Engine*-a te detaljnom dokumentacijom i forumima za pomoć.

Iako Unity nudi veliku pomoć svojim korisnicima i dalje je potrebno da korisnik sam nauči osnove programiranja te optimizaciju koda jer je to u igrama jako bitno. Velik broj početnika radi greške u kreiranju koda gdje cijeli kod stavljaju unutar Update metode koja se izvršava nakon svake slike igre te se samim time usporava igra zbog velike količine koda koji se nepotrebno izvršava.

Žanr igri uloga je jedan od najkompliciranijih žanrova za izradu, zbog velike količine funkcionalnosti, velikog svijeta kojega treba dizajnirati, puno kalkulacija koje se trebaju korektno izvršavati kako ne bi došlo do ogromnih brojki i stvari koje bi mogle „rušiti“ igru. Zbog toga čak i velike kompanije koje proizvode takve igre moraju nastaviti još godinama raditi na igri kako bi ih popravili i uklonili greške jer se neke greške teško primijete i u testiranju.

Popis literature

[1] M. Lugris, (09.09.2020.) *InnerSloth's Party Game Among Us Reaches 1.5 Million Simultaneous Players*, Preuzeto 01.09.2021 s https://web.archive.org/web/20200921123233/https://www.thegamer.com/among-us-1-5-million-players-simultaneous-innersloth-party-game/

[2] J. Brodkin, (03.06.2013.) *How Unity3D Became a Game-Development Beast,* Preuzeto 01.09.2021 s https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

[3] Unity Technologies, (30.08.2021.) *MonoBehaviour* [Dokumentacija], Preuzeto 01.09.2021 s https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[4] Unity Technologies, (2021.) *Welcome to Unity Learn,* Preuzeto 01.09.2021 s https://learn.unity.com/

[5] Unity Technologies, (2021.) *Unity Asset Store*, Preuzeto 01.09.2021 s https://assetstore.unity.com/

[6] Microsoft, (bez dat.) *Visual Studio 2019*, Preuzeto 01.09.2021 s https://visualstudio.microsoft.com/vs/

[7] B. Ware, (bez dat.) *An art community for everyone!*, Preuzeto 01.09.2021 s https://www.pixilart.com/about

Popis slika

Prilozi

Instalacija igre (.apk) je dostupna na: https://drive.google.com/file/d/1snzhSxCFvSJJg6kn6-MQCphZzLuq3X2R/view?usp=sharing