Izrada računalne igre simulacije farme

Fuks, Filip

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Fuks

Izrada računalne igre simulacije farme

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Fuks

Matični broj: 43134/14-R

Studij: Informacijski sustavi

Izrada računalne igre simulacije farme

ZAVRŠNI RAD

Mentor:

Dr.sc. Mladen Konecki

Varaždin, Siječanj 2020.

Filip Fuks

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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj završni rad obuhvaća izradu video igre u razvojnom alatu zvanom „Unity“. Žanr video igre koja će biti opisana u završnom radu je „Farming Simulator“. U nastavku će ukratko biti opisano što je Unity, koje su njegove karakteristike, što je „Farming Simulator“, mehanike toga žanra i inspiracije za ovaj završni rad.

U nastavku će biti opisan razvojni proces igre, stvaranje glavnoga lika gdje će biti opisane animacije, dizajn i skriptiranja mehanika. Opisani će također biti neprijatelji, svijet, sađenje i korištenje biljaka.

Naposljetku će biti iznošeni osobno mišljenje o radu s alatom i iskustvo općenitog razvoja video igre.

Ključne riječi: razvoj video igre, Unity, programiranje, game development, video igra, Farming Simulator, Unit

1. Sadržaj

2. Uvod

Još od malena sam imao veliki interest i ljubav za video igrama, još uvijek često znam spojiti svoju prvu konzolu koju sam posjedovao, Super Nintendo Entertainment System (SNES ukratko), i još uvijek imam jako lijepa sjećanja kada smo brat i ja zajedno sjedili pred televizijom i igrali video igru „Super Metroid“, u to vrijeme najbolje video igrice su (po osobnom mišljenju) bile iz Japana, igre poput Legend Of Zelda, Super Metroid, Chrono Trigger, Megaman... Doduše zapadnjačka kompanija koja mi pada na pamet kada god se sjetim SNES-a je RareWare sa svojim Donkey Kong serialom.

U to vrijeme nisam znao puno o video igricama, ali nastala je jedna velika želja u meni, a to je da jednom napravim video igru koja će druge usrećiti koliko je mene igra Super Metroid usrećila. I to me jednim dijelom inspiriralo da odaberem ovu temu za završni rad. Glavna inspiracija za završni rad su igre poput „Stardew Valley“ i „Harvest Moon“.

Odabir alata za razvoj igre nije bio težak. Unity kao alat je jako pristupačan i ima mnogo dokumentacije, veliku zajednicu i veoma je jednostavan za početnike. Naime nekada je to loše zato što mnogi ljudi naprave ne završene igre i kao takve ih prodaje, te je zato Unity na malo lošijem glasu kada dođe do zajednice razvoja video igara. Unity, doduše, je veoma dobar alat za izradu video igrica, neki primjeri bi bili sljedeće: Heartstone, Ori and the Blind Forest, Guns of Icarus... sve su to veoma kvalitetne, kompleksne i veoma dobre video igre. Uzevši sve to u obzir, odlučio sam uzeti Unity kao alat za izradu Farming Simulator igre koju radim za ovaj završni rad.

3. Unity

Unity je Pokretač (eng, game engine). Pokretač je program koji omogućava stvaranje video igre brzo i jednostavno. Pokretač je okvir za razvoj igara koji spaja sve aspekte izrade u jednu lijepo pakiranu cjelinu.

Unity je prvi puta pušten na tržište u lipnju 2005. Godine. U 2018. Unity je omogućio podršku na više od 25 platformi koje uključuju ali nisu limitirane na: mobilne telefone, Xbox One, PlayStation 4, Windows, Nintendo Switch...

Iako je Unity primarno Okvir za izradu video igara, prisvojen je od mnogih drugih industrija poput filma, automobilne industrije, arhitekture, inženjerstva i konstrukcije zbog mogućnosti simulacija i drugih iskustava koje je moguće replicirati pomoću Unity tehnologije.

Unity u svojim početcima je ciljao na demokraciju u izradi video igara, omogućivajući izradu mnogim ljudima koji nisu direktno u industriji video igrica.

Unity je dostupan u četiri verzije:

- Personal – besplatna verzija koja sadrži sve esencijalne funkcije za kreiranje video igre i namijenjena je za početnike koji se žele upoznati s Unityem.
- Plus – verzija koja se plaća 40$ mjesečno i omogućuje najnoviju verziju jezgrene unity razvojne platforme, premium resurse za učenje, promjenu ulaznog ekrana, Live analitiku, cloud dijagnostiku u stvarnom vremenu
- Pro – Sve što dolazi plus verziji, tri „sjedala“ ta Unity Teams Advanced prioritetnu pristup za Unity savjetnike za uspjeh, i personalizirane opcije za kupnju (jezgreni kod i premium podrška)
- Enterprise – ova verzija se ne može kupiti, ali se može stupiti u kontakt s Unity timom koji će pomoći osnovati tvrtku zasnovanu na Unity tehnologijama.

Unity podržava 2D ili 3D igre. Daju nam se pod-alati koji nam omogućuju mnoge stvari kao na primjer grafičko sučelje za kreiranje svijeta, animator, i Compiler koji provjerava točnost skripti. Ali uz to, još imamo i pristup unity tržištu (Unity Asset Store), ono nam omogućuje nabavu besplatnih dijelova koje možemo koristiti u svojem projektu.

Unity podržava nekoliko programskih jezika kao na primjer UnityScript što je Unity prilagođena verzija JavaScripta, te C# koji sam ja odlučio koristiti u ovom projektu.

S obzirom na to da Unity ima Jako puno mogućnosti za korištenje, na Unity stranici dostupan je opširan priručnik pod imenom „Unity User Manual“. Priručnik ima detaljno

objašnjene funkcije koje bi nam mogle zatrebati tijekom izrade naše igre. U priručniku je moguće i pronaći jednostavne primjere nekih jednostavnih vrsti igara.

4. Farming Simulator

Ova igra je žanra simulatora farme. Inspiracija za ovaj žanr dolazi od video igara tipa „Stardew Valley“ (Stardew Valley web stranica 2020), „Harvest Moon“ (Wikipedia, 2020), „Animal Crossing“ (Animal Crossing web stranica 2020), „Story of Seasons“ (Wikipedia 2020).

U ovim igrama fokus je na uzgajanju biljki ili životinja, ali uz to vrlo često imaju neki dodatni cilj poput druženja s ljudima, neke druge seoske aktivnosti (pecanje, planinarenje, preživljavanje u šumi) ili istraživanje pećina.

Video igra „Stardew Valley“ je glavna inspiracija za ovu igru, pa ću pomoću nje opisati mehanike igre. Glavni cilj igre je uzgajati voće i povrće, prvih dana, ne možemo imati puno biljaka, pa možemo ići po gradu malo razgovarati i upoznavati se s ljudima. Kasnije nam se otvori mogućnost pecanja i istraživanje pećina. U pećinama možemo pronaći rijetko sjeme čije plodove možemo prodati za velike novce ili ih koristiti da naš lik kojeg upravljamo postane jači. Naime u pećinama nismo sami, nego se mogu naći svakakve životinje i čudovišta koja čine skupljanje tih biljaka malo problematičnije.

U ovom radu, odlučio radi lakše prezentacije napraviti lako dostupno bilje koje donosi mnogo novaca s kojim možemo kupiti bijeg s otoka na koji smo se nasukali. Na otoku mogu se naći nekoliko neprijatelja, 3 vrste sjemena, ćupovi, ljudi za pričanje, pojačanja i luk i strijela. Svaka biljka ima neko dodatno svojstvo uz prodaju, ćupovi u sebi sadrže novčiće, strijele i srčeka koja mogu vratiti izgubljene živote igraća.

5. Struktura igre

U ovom dijelu ću opisati kako sam izradio sve elemente igre. Sve priložene slike bit će slikane u Unity verziji 2018.4.13f1.

5.1. Glavni Lik

Kada smo tek pokrenuli Unity po prvi puta mogli smo odabrati budemo li radili 2D ili 3D igru. S obzirom na to da sam odlučio napraviti 2D igru, tako sam odabrao 2D opciju. Unity nakon toga ponudi nekoliko primjera video igara koje su napravljene u 2D. Ako smo slučajno odabrali opciju za 3D igru, ne moramo potpuno ugasiti projekt i kreirati novi, moguće je promijeniti iz 2D u 3D opciju unutar „Project Settings“ prozora.

Nakon početne pripreme Unity-a počinjemo s potpuno praznim platnom. Kako bi mogli dobiti nekakve smislene objekte moramo samostalno napraviti slike (eng. Sprite) koje će dati izgled likovima, objektima i prostoru u igri. Unity u sebi posjeduje nešto što se zove „Asset Store“, to je tržište razno raznih slika i funkcionalnosti. Neke slike i funkcionalnosti se mogu kupiti, ali postoji mnogo besplatnih slika koje se mogu koristiti. S obzirom na to da je u ovom projektu kamera ortografska, morao sam naći zanimljiv set slika za glavnoga lika koji će imati sliku za svaki smjer kretanja.

Slika 1: Svi smjerovi kretanja glavnoga lika (vlastita izrada)

Svaka slika je bila odvojena, i postojale su samo slike za kretanje u desnu stranu. Kako bi omogućio da se Lik može okrenuti u lijevu stranu mogao sam napraviti jednu od dvije stvari. Prvo što sam mogao je programski okrenuti sliku lika za -1 na X osi, to bi proizvelo efekt kretanja u drugom smjeru. Druga način na koji sam mogao to popraviti je tako da ručno okrenem sliku u programu za obradu slike za što sam se odlučio ja. Sve slike su odvojene, i nalaze se u organiziranim mapama: Hodanje, Napad, Miran.

Kako bi stvorio glavnog lika, iz mape „Miran“ uzeo sam sliku gdje je glavni lik okrenut prema nama. Kako bi omogućili dana njega djeluje fizika i neprijatelji moramo mu dati „box collider“ i „Ridgidbody2D“ komponente. Collider mora biti niži od lika za glavu, ali bi bilo dobro i da se širina stavi da je manja od glavnoga lika jer će to učiniti igru prividno više „Fer“.

Nakon kreiranja objekta glavnog lika, dajemo mu tag „Player“ što će nam omogućiti da se referenciramo na glavnoga lika u kodu. Ali kako bi naš lik mogao biti udaren od raznih bića moramo napraviti posebni još jedan collider koji će se ponašati kao okidač (eng. trigger). Takvi slični Collideri moraju postojati i kako bi naš lik mogao napasti.

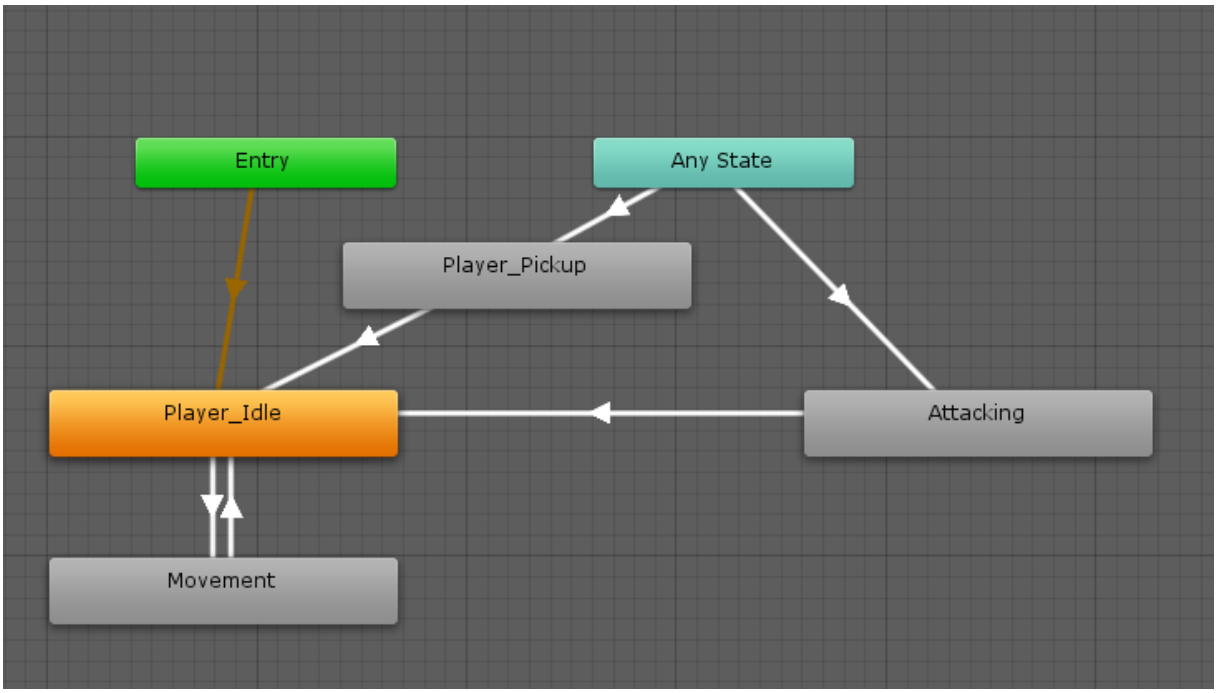
Kako bi naš glavni lik mogao napadati moramo mu dati posebni Hitbox (područje s kojim nanosi štetu). Stavljajući slike kako glavni lik napada možemo odrediti odokativno koje područje zahvati mač kada napadnemo s glavnim likom. Područje napada ćemo označiti pomoću „polygon Collider 2D“ komponente, i sve ih postaviti da imaju svojstvo „Okidača“, postaviti ih kao da su djeca igrivog lika, i postaviti da su neaktivni.

5.1.1. Animiranje Glavnog Lika

Unity za izradu animacija sadrži pod program koji se zove „Animator“. Da bi mogli koristiti Animator, moramo prvo dodati komponentu animatora na glavnoga lika. Nakon toga otvaramo potprogram „Animation“ gdje ćemo stvoriti animacije za našega lika. Otvorivši datoteke koje smo prethodno spomenuli (miran, hodanje, napad), i iz svake datoteke na glavnog lika staviti slike u Animation. Za svaku akciju moramo napraviti novi „clip“ animacije (kretanje gore, dolje, lijevo, desno, napad gore, dolje, lijepo desno, miran gore, dolje, lijevo, desno). Sve animacije postavimo da imaju 60 uzoraka, te označimo da se ne ponavljaju (eng. loop). Svaku sliku animacije stavimo na razmak od 0.05 sekundi kako bi postigli osjećaj fluidnosti.

Nakon što smo postavili animacije moramo ih povezati međusobno u Animatoru. Svaka animacija je predstavljena pomoću pravokutnika koje povezujemo tranzicijama koje se aktiviraju kada ako se zadovolji neki uvjet koji im mi zadamo. Kasnije ćemo opisati kako mijenjamo animacije unutar koda.

Animacija napada mora imati vrijeme izlaza koje korespondira duljini animacije napada. Ako je vrijeme izlaska pre rano, onda se animacija napada završi prije nego li se prikaže posljednja slika animacije. Ako je pre dugačko, onda se može dogoditi da se animacija ponovi nekoliko puta.



Slika 2: Animacijski kontroler za igrivog lika (izvor: Vlastita izrada)

5.1.2. Kontroliranje igrivog lika

5.1.2.1. Kretanje

Skripta za kretanje lika započinje sa „enum“, to je enumerator što je popis konstanti u kojima naš lik može biti. Igrač može biti u „walk“,“attack“,“interact“,“stagger“,“idle“ stanju, svako stanje nam može omogućavati ili onemogućavati pristup nekim metodama koje mi određujemo. Kada je igra tek pokrenuta, igrač se nalazi u „walk“ stanju. Za vrijeme svakog okvira koji prođe u igri (eng. Frame), očitava se „Update“ metoda i u njoj se poziva metoda „MoveCharacter“ ta metoda nam omogućava kretanje lika.

Prije nego li se lik može kretati mora imati neku brzinu i mjesto spremanja trenutne pozicije. Za kretanje lika koristimo funkciju „transform“, transform gleda poziciju našega lika te vektorski govori igraču koliko bi se trebao pomaknuti na ekranu, kojom brzinom kroz koliki vremenski razmak. Ali ako koristimo dva smjera kretanja istovremeno, lik će se kretati vidljivo brže nego u samo jednom smjeru. Da bi to negirali, potrebno je pozvati metodu „Normalize“ nad pozicijom našega lika (u našem slučaju „change“ varijabla). Nakon što normaliziramo možemo pokrenuti lika tako da pozovemo metodu od igračevog ridgid body-a, metodu „MovePosition“, unutar metode, navodimo da transformiramo poziciju, uz change, brzinu i vrijeme proteklo. Prethodna metoda će nam omogućiti kretanje lika, ali neće doći do promjene animacije. Kako bi promijenili animaciju, moramo unutar metode kretanja ući u metodu

„Animating“. Metoda animating započinje s if selekcijom, koja provjerava da naše kretanje (Vector3) nije jednak nuli. Ako nije jednak nuli, prosljeđujemo animatoru smjer kretanja našega igrivoga lika i prosljeđujemo da se bool „moving“ postavlja u „true“. Ako niti jedan gumb za kretanje nije pritisnuti onda moving se automatski postavlja na false.

```
void MoveCharacter(){
    change.Normalize();
    myRigidBody.MovePosition(
        transform.position + change * speed.RuntimeValue * Time.deltaTime );
        Animating();
}

void Animating(){
    if(change != Vector3.zero){
        animator.SetFloat("moveX", change.x);
        animator.SetFloat("moveY", change.y);
        animator.SetBool("moving", true);
    }else{
        animator.SetBool("moving",false);
}
}
```

Slika 3: Kod kretanja igrivog lika (izvor: Vlastita izrada)

5.1.2.2. Blisko napadnje

Kako bi naš igrač mogao napasti moramo postaviti bool za „attack“ u true, to će nam omogućiti da animator ode u „attacking“ animaciju. S obzirom na to da cijelo vrijeme prosljeđujemo smjer kretanja animatoru, s obzirom na to da znamo u kojem je smjeru lik okrenuti, znamo koje animacije treba upaliti. U Animation windowu postoji funkcija koja se zove „Record“ ona nam omogućuje snimanje promjena koje se događaju u igri kroz animacije. Kako bi postigli iluziju napada prvo ćemo započeti snimanje napada u bilo koju stranu te nakon početka snimanja upalimo pripadajući hitbox za napad koji se nalaze na našem liku, na kraju animacije, ugasimo hitbox i prekinemo snimanje. To će nam omogućiti da kada god se odrađuje animacija ta da se upali taj pripadajući hitbox. Kako bi onemogućili da igrač cijelo vrijeme napada i tako bude besmrtan moramo malo ga ograničiti, to ćemo učiniti tako da natjeramo lika da pričeka 0.35 sekundi nakon svakoga napada, i nakon toga se vraća u stanje hodanja.

```
private IEnumerator AttackCo(){
    animator.SetBool("Attacking", true);
    currentState = PlayerState.attack;
    yield return null;
    animator.SetBool("Attacking", false);
    yield return new WaitForSeconds(.35f);

    if(currentState != PlayerState.interact){
        currentState = PlayerState.walk;
    }
}
```

Slika 4: Kod za primarno napadanje (izvor: Vlastita izrada)

5.1.2.3. Napadanje na daljinu

Igrači lik unutar video igre može napadati i s lukom i strijelom ako nađemo potrebne stvari. Kada napadamo s lukom, postavljamo lika u „attack“ stanje, te zatim pozivamo metodu za stvaranje strijele „MakeArrow“. MakeArrow Metoda stvara privremeni vektor koji ima identične vrijednosti kao vektor našeg kretanja. Nakon toga se zatim se stvara nova strijela na poziciji gdje je igrači lik i strijeli se prosljeđuje privremeni vektor i metoda koja odredi u kojem smjeru će biti strijela okrenuta. Strijela zatim prima te dvije varijable i lansira se u smjeru koji je zadan s točnom orijentacijom. Nakon toga smanjujemo ukupnu količinu strijela koje igrač nosi za 1 i čekamo 0.35 sekundi.

```
private IEnumerator AttackSecondaryCo(){

    currentState = PlayerState.attack;
    yield return null;
    MakeArrow();
    playerInventory.arrow -=1;
    arrowMinus.Raise();
    yield return new WaitForSeconds(.35f);

    if(currentState != PlayerState.interact){
        currentState = PlayerState.walk;
    }
}

private void MakeArrow(){
    Vector2 temp = new Vector2(animator.GetFloat("moveX"), animator.GetFloat("moveY"));
    Arrow arrow = Instantiate(projectile, transform.position, Quaternion.identity).GetComponent<Arrow>();
    arrow.Setup(temp, ChooseArrowDirection());
}

Vector3 ChooseArrowDirection(){
    float temp = Mathf.Atan2(animator.GetFloat("moveY"), animator.GetFloat("moveX")) * Mathf.Rad2Deg;
    return new Vector3(0, 0, temp);
}
```

Slika 5: Kod za pucanje strijela (izvor: Vlastita izrada)

5.1.2.4. Skripta za udarce

Skripta za udarce je univerzalna skripta koju pridjeljujemo svim objektima koji mogu oštetit neprijatelje ili igrača. Skripta se zove „Knockback“ i zahtjeva prosljeđivanje 3 varijable, jačina guranja, trajanje ošamućuenosti i šteta. Skripta se aktivira ako se nešto nalazi ili uđe unutar zone okidača. Ako se collideri sa tagom player ili enemy nalaze unutar zone okidača, prvo izračunavamo koliko će napadnuti objekt biti odgurnut od objekta koji napada. Dohvaćamo fizičko tijelo objekta koji je napadnut te ga odmičemo za jačinu guranja. Nakon toga prosljeđujemo udarenom objektu upute koliko jako se mora odmaknuti od napadačkog objekta. Nakon što je objekt odgurnuti, oduzimamo životne bodove objektu. Ukoliko je neprijatelj u pitanju. Postavljamo ga u „stagger“ stanje kako igrač ne bi mogao više puta odjednom napasti neprijatelj, te se zatim prosljeđuje u „Knock“ metodu smjer i jačina odgurivanja, nanesena šteta i duljina u kojoj neprijatelj ostaje ošamućen.

Ukoliko je igrač taj koji je trebao primiti štetu, igrač se postavlja u „stagger stanje“ kako bi omogućili sekundu besmrtnosti nakon primanje štete, te da se odgurivanje korektno izvrši. Nakon toga prosljeđuje se Knock metodi vrijeme ošamućenosti te šteta igraću.

Knock metoda unutar skripte za igrača smanjuje ukupne životne bodove igrača za proslijeđenu razinu, podiže signal za refresh UI-a, nakon toga započinje se ko-rutina, koja postavlja igrača u idle poziciju nakon zadanog vremena ošamućenosti.

```
private void OnTriggerEnter2D(Collider2D other){
    if(other.gameObject.CompareTag("enemy") || other.gameObject.CompareTag("Player"))
    {
        Rigidbody2D hit = other.GetComponent<Rigidbody2D>();
        if(hit != null)
        {
            Vector2 difference  = hit.transform.position - transform.position;
            difference = difference.normalized * thrust;
            hit.AddForce(difference, ForceMode2D.Impulse);
            if(other.gameObject.CompareTag("enemy") && other.isTrigger)
            {
                hit.GetComponent<Enemy>().currentState = EnemyState.stagger;
                other.GetComponent<Enemy>().Knock(hit, knockTime, damage.RuntimeValue);
            }
            if(other.gameObject.CompareTag("Player"))
            {
                if(other.GetComponent<PlayerMovement>().currentState != PlayerState.stagger)
                {
                    hit.GetComponent<PlayerMovement>().currentState = PlayerState.stagger;
                    other.GetComponent<PlayerMovement>().Knock(knockTime, damage.RuntimeValue);
                }
            }


        }
```

Slika 6: Kod za udarce (Izvor: Vlastita izrada)

5.1.2.5. Skripta za žetvu

Skripta za žetvu sastoji se od 2 dijela. Ova skripta se postavlja na zone okidača s kojima igrač napada. Skripta je vrlo jednostavna. Ako se unutar zone okidača nalazi ćup ili biljka, poziva se metoda za razbijanje ćupa ili za žetvu biljke. Metoda žetve unutar skripte za biljku poziva metodu za stvaranje nasumičnih stvari. Nasumično iz popisa stvari računalo odabire što će biljka ostaviti s obzirom na postotke koje smo mi odredili. Nakon što su stvari odabrane, kažemo unityu da stvori te stvari ispod biljke koju smo poželi. Skripta je identična za ćupove.

```
public void Harvest()
{
    if(matured)
    {
        MakeLoot();
    }
}

private void MakeLoot(){
        for(int i = 0; i<iterations; i++){
        if(thisLoot != null)
        {
            Powerup current = thisLoot.LootPowerup();
            if(current != null)
            {
                Instantiate(current.gameObject, transform.position, Quaternion.identity);

            }

            Instantiate(seeds.gameObject, transform.position, Quaternion.identity);
        }
        }
        Destroy(this.gameObject);
```

Slika 7: Kod za žetvu (Izvor: Vlastita izrada)

5.2. Neprijatelj

Izrada neprijatelja je slična izradi igraćeg lika uz neke razlike koje ću opisati.

5.2.1. Stvaranje neprijatelja

Proces stvaranja neprijatelja malo se razlikuje od procesa stvaranja igraćeg lika. Unosimo potrebne slike neprijatelja u Unity, ali neprijatelj nema animaciju napada, nego samo animaciju hoda.

Slika 8:Kretanje neprijatelja (Izvor: Vlastita izrada)

Nakon što su dodane slike, napravljeno je novo drvo animacije, koje na sebi ima varijable kretanjeX, kretanjeY, loviti, kretati. Neprijatelj je prirodno u stanju čekanja „idle“, te ako se bool varijabla „kretati“ upali i postane istinita, onda se prelazi u kretanje animaciju. Ako „kretati“ postane netočno, onda se neprijatelj vrati u stanje čekanja. Neprijatelju stavljamo naznaku da je neprijatelj odnosno „enemy“.

5.2.2. Umjetna intaligencija

5.2.2.1. Stanje neprijatelja

Svaki neprijatelj može biti u jednom od 4 neprijateljska stanja. Ta stanja su određena u kodu, kada se s neprijateljem nešto dogodi, naravno, i stanje mu se promjeni. Kako bi mogli imati sveprisutno stanje za svakog neprijatelja posebno stvaramo nešto što se zove „enum“ metoda, i dajemo 4 stanja: čekanje, hodanje, napada, ošamućen. Kada se tek stvori, neprijatelj je u stanju čekanja. Ako igrač dođe u blizinu neprijatelja, neprijatelj odlazi u stanje kretanja, i kada se to dogodi, animatoru se šalje signal da je kretanje istina, te se preazi u animaciju hoda s obzirom na to u kojem se smjeru kreće neprijatelj. Ukoliko neprijatelj ošteti glavnoga lika, neprijatelj prelazi u napadačko stanje, koje onemogućuje neprijatelju da napravi više puta štetu igračem liku. Nakon 1 okvira neprijatelj se vraća u stanje kretanja te nastavlja svoju potjeru za glavnim igračem.

Ukoliko neprijatelj zaprimi štetu, prebacuje se u stanje ošamućenosti. Stanje ošamućenosti neprijatelju ne dopušta da nastavlja potjeru za igraćim likom i ne dopušta mu da zaprimi novu štetu dokle god je u tom stanju ne može zaprimiti novu štetu.

5.2.2.2. Umiranje

Kada neprijatelju se životni bodovi spuste na nulu ili niže, pokrećemo dvije metode. Prva metoda koja se pokreće se zove efekt smrti, to je metoda koja poziva animaciju umiranja. Animacija umiranja je prethodno napravljeni objekt koji služi samo da bi imali animaciju, objekt izvrši svoju animaciju te se zatim uništava. Druga metoda koja se poziva je napravi stvari. Kada neprijatelj umre, ima šansu da ostavi neke stvari iza sebe. Iz tablice za stvari uzima jednu od stvari koje ima ponuđene (strijele, srčeka ili novac). Nakon pozivanja metoda, objekt neprijatelja se isključuje.

5.2.2.3. Kretanje

Kako bi se neprijatelj mogao kretati morali smo napraviti detekciju kada igrači lik uđe u njegov vidokrug. Koristimo Metodu „FixedUpdate“ razlika je u tome što Update se koristi svaki okvir, dok FixedUpdatese događa cijelo vrijeme. Provjeravanje udaljenosti započinjemo s if selekcijom gdje gledamo je li igrači lik unutar vidokruga, vidokrug je globalna varijabla koju možemo namještati kako mi želimo posebno za svakog lika posebno. Ako se igrač ne nalazi u vidokrugu, neprijatelj ostaje ili prijelazi u stanje čekanja. Ako je igrač u vidokrugu, i neprijatelj nije u stanju ošamućenosti, neprijatelja pomičemo prema igraču tako da pomičemo njegov „RidgidBody“ prema poziciji igrača. Stanje neprijatelja se postavlja na „hodanje“ te se prosljeđuje animatoru zastavice da se neprijatelj kreće prema igraču.

```
void FixedUpdate()
{
    CheckDistance();
}

public virtual void CheckDistance()
{
    if(Vector3.Distance(target.position,
                        transform.position) <= chaseRadius
        && Vector3.Distance(target.position,
                            transform.position) > attackRadius)
    {
        if(currentState == EnemyState.idle || currentState == EnemyState.walk
            && currentState != EnemyState.stagger){
            Vector3 temp = Vector3.MoveTowards(transform.position, target.position, moveSpeed*Time.deltaTime);
            changeAnim(temp - transform.position);
            myRigidbody.MovePosition(temp);

            ChangeState(EnemyState.walk);
            anim.SetBool("chase", true);
            anim.SetBool("Move", true);
        }

    }else if(Vector3.Distance(target.position, transform.position) > chaseRadius){
            anim.SetBool("chase", false);
            anim.SetBool("Move", false);
        }
}
```

Slika 9:Kretanje Neprijatelja (Izvor: Vlastita izrada)

5.2.2.4. Napadi neprijatelja

Kako bi neprijatelj mogao napadati, na njega prvo stavljamo jedan novi box collider koji je malo veći od prvotnoga, te stavljamo da se taj collider ponaša kao okidač. Zatim stavljamo na neprijatelja skriptu koju smo stavili na zone napada našeg glavnog lika. Skripta za udarce je univerzalna skripta koja je zadužena za štetu koja se načini likovima i prethodno je bila detaljno objašnjena. S obzirom na to da smo dali skriptu neprijatelju, kada igrač uđe u zonu okidača na njega će se izvršiti sve prethodno objašnjene stvari koje se događaju kada netko primi štetu.

5.3. Kamera

Kamera ima vrlo bitnu ulogu u ovoj igri. S obzirom na to da je igra ortogonalna, moramo konzistentno pratit igraćeg lika. Ali radi boljeg dizajna moramo onemogućiti da se kamera kreče uvijek za likom, primjer restrikcije bi bilo prijelaz u novu sobu. Ako Igrač dođe ispred ulaza u sobu, kamera ne smije pokazati što se događa u sobi u koju igrači lik treba uči, ali kada igrači lik prođe kroz neku određenu crtu, onda kamera mora brzo se prebaciti u tu drugu sobu. Za kameru smo koristili smo komponentu s tržišta koja se zove „Cinemachine“. Cinemachine je zanimljiva komponenta jer omogućava lagano centriranje kamere na igraćega lika.

Kako bi omogučili Cinemachine moramo dodati „Cinemachine Brain“ komponentu na glavnu kameru, ta komponenta će računati kuda se mora glavna kamera pomaknuti s obzirom na to gdje se lik nalazi. S obzirom što Cinemachine radi na principu Zona Okidača, radimo novi objekt kojem ćemo dodjeliti ime sobe, te na njega staviti komponentu „Polygon Collider“ i postaviti da se ta komponenta tretira kao okidač. U taj objekt stavljamo sve neprijatelje, objekte i ostale objekte koji bi se nalazili u toj sobi kao da su djeca tog objekt. Zatim dodajemo jedan novi dijete objekt u taj glavni soba objekt koji se zove Cinemachine Virtual Camera. Na Cinemachine Virtual Camera stavljamo komponente „Cinemachine Virtual Camera“, „Cinemachine Confiner“ i „Cinemachine Impulse“. Virtual Camera govori Glavnoj kameri što da prati, u našem slučaju to će biti igrači lik te samo povlačimo igraćeg lika iz popisa svih objekata na follow polje u virtualnoj kameri. Cinemachine Confiner je Komponenta koja će limitirati glavnu kameru na granice koje mi zadamo. S obzirom na to da smo napravili da soba ima komponentu polygon collidera, stavljamo na polje „bounding shape 2D“ sobu u kojoj se nalazi virtualna kamera. Cinemachine Impluse je komponenta koja sama po sebi ne radi ništa, ali ako neprijatelj napravi štetu glavnom liku, ili glavni lik napravi štetu neprijatelju, kamera će se lagano zatresti kako bi dobili osjećaj za štetu.

Slika 10: Kamera kontroler(Izvor: Vlastita izrada)

5.4. Životna traka

5.4.1. Dizajn i implementacija

Kako bi napravili traku za životne bodove, moramo prvo kreirati novi UI objekt koji se zove „Canvas“. I dajemo mu novi UI objekt sliku, postavljamo je u gornji lijevi kut, te mu pridodajemo sliku koju želimo, dajemo ime tom objektu DržačŽivota, i dodajemo novi objekt koji će se ponašati kao dijete Držača i nazovemo ga kontejneri života. I zatim na kontejnere

stavljamo jednu komponentu „Horizontal Layout Group“ koja će centrirati samo srčeka da budu razvrstana ravnomjerno.

Trenutno imamo mjesto za postaviti srčeka, ali ne način na koji ćemo ih prikazivati. Za prikazivanje srčeka napravili smo novu skriptu koju ćemo staviti na Kontejner za srčeka. Kada skripta tek započinje pozivamo obije metode koje se nalaze u njoj. Prva metoda inicijalizira srca, a druga ažurira srca. Inicijalizacija srca počinje iteracijom koja postavlja ukupni broj srca na 3. Ako igrač primi štetu moramo pozvati metodu ažuriranja srca, ažuriranje srca provjerava koliko imamo sveukupno trenutno života i zatim pridjeljuje određenu sliku srca (puno, polu puno ili prazno srce).

```
void Start()
{
    InitHearts();
    UpdateHearts();
}

public void InitHearts(){
    for(int i = 0; i < heartContainers.RuntimeValue; i++)
    {
        hearts[i].gameObject.SetActive(true);
        hearts[i].sprite = fullHeart;

    }
}

public void UpdateHearts(){
    InitHearts();
    float tempHealth = playerCurrentHealth.RuntimeValue/2;
    for(int i = 0; i < heartContainers.RuntimeValue; i++)
    {
        if(i <= tempHealth-1)
        {
            //Full Heart
            hearts[i].sprite = fullHeart;
        }else if(i >= tempHealth)
        {
            //empty heart
            hearts[i].sprite = emptyHeart;
        }else{
            //half full heart
            hearts[i].sprite = halfFullHeart;
        }
    }
}
```

Slika 11:Skripta za upravljanje srcima (Izvor: Vlastita izrada)

Nakon što smo dodali ovu skriptu kao komponentu moramo predodrediti količinu maksimalnih srca u objektima. Za količinu srca moramo napraviti novi objekt koji će se biti child objekt kontejnera za srca, ti objekti će imati samo sliku punoga srca na sebi i bit će ugašeni

svi na početku. Ako se pokupi neko pojačanje da imamo više životnih bodova, onda se poziva inicijalizacija srca, te se stvara dodatno srce koje se stavlja s ostalim srcima.

Kako bi igra znala kada je neprijatelj napravio štetu igračem liku, moramo napraviti neki sistem signaliziranja. Zato smo napravili komponentu zvanu „SignalListener“, ta komponenta služi samo da bi pozivala metode iz drugih metoda kada se podigne signal. Signal smo napravili da je Skriptni Objekt. Skriptni Objekti su kontejneri za podatke koji se ne nalaze na sceni, nego u pozadini igre. U njima najčešće spremamo trenutno stanje koje trebamo prenositi između scena, kao na primjer količinu životnih bodova, preostale strijele, i općenito sve stvari koje se nalaze u torbi igraćega lika. Kada se šteta nanese igračem liku, ili igrači lik pokupi srčeko, podiže se signal za život koji poziva ažuriranje srca. Da to ne postoji, srca se ne bi nikada mijenjala, i ne bi nikada znali na kolikom smo zdravlju.

Novčići se nalaze kao dijete objekta „Canvas“. Napravili smo objekt i dali smo mu sliku koja će biti pozadina na kojoj će se mijenjati tekst. Tekst smo stavili kao dijete te pozadine i postavili smo da text ispisuje četiri nule. Na pozadinu za novčiće stavljamo skriptu slušača za signale koji prati ako igrač skupi novčić, i ako ga skupi, poziva se skripta za ažuriranje novčića. Skripta za ažuriranje novčića sadrži samo dvije varijable torbu igrača i prikaz novčića. Prikaz novčića je tekst koji mijenjamo s obzirom na to koliko imamo novčića, torbu ćemo kasnije opisati.

Strijele se nalaze kao dijete objekta „Canvas“. Napravili smo objekt i dali mu sliku koja će biti pozadina na kojoj će se mijenjati tekst. Tekst smo stavili kao dijete te pozadine i postavili smo da text ispisuje dvije nule jer nije predviđeno da igrači lik može imati više od 99, ali je polje automatski ekspanzivno. Na pozadinu za strijele stavljamo komponentu slušača za signale koji prati ako igrač pokupi strijele, i ako su strijele skupljene povećava se broj strijela koje imamo u torbi. Strijele su varijacije skripte „Powerup“, ta skript sadrži signal i signalizaciju. Kada hoćemo napraviti varijaciju poweupa, onda samo referenciramo njega. Nakon što se strijele dodaju u torbu, taj objekt se uništava te se odašilje signal da je došlo do akcije uzimanja strijela. Taj signal čuje pozadina za strijele čuje te onda ažurira strijele na točnu razinu i onemogućuje da imamo više strijela od trenutnoga maksimalnoga iznosa.

```
public class ArrowPickup : Powerup
{

    public Inventory playerInventory;
    public int maxArrow;
    // Start is called before the first frame update
    void Start()
    {
        CheckArrows();
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnTriggerEnter2D(Collider2D other){
        if(other.CompareTag("Player") && !other.isTrigger)
        {
            if(playerInventory.arrow < 10){
                playerInventory.arrow += 2;
                if(playerInventory.arrow > 10){
                    playerInventory.arrow = maxArrow;
                }
            }
            CheckArrows();
            Destroy(this.gameObject);
        }
    }

    public void CheckArrows(){
        powerupSignal.Raise();
    }
}
```

Slika 12:Skripta za strijele (Izvor: Vlastita Izrada)

5.5. Torba

S obzirom na to da je ovo igra nadahnuta igrom „Stardew Valley“ s toga moramo imati torbu u koju ćemo spremati stvari koje pokupimo s poda, bilo to biljke, ključevi, mačevi, luk i strijela ili ljekoviti napitak. Implementacija torbe je sljedeća. Prvo smo napravili novi canvas koji je isključen na početku te mu dali child objekt koji smo nazvali Torba. Game object torba ima u sebi naziv da znamo da smo trenutno u torbi, objekt koji se zove „Scroll view“, to je objekt kojeg unity nam pruža i on omogućuje da u njega stavimo mnoge gumbove i uvijek će svi biti dostupni jer možemo se kretati kroz taj pogled. Sadrži još i opis objekta na koji smo kliknuli, cijenu za koliko možemo prodati neki objekt na koji smo kliknuli, te gumb za prodaju.

Kako bi igrač mogao imati svoju torbu prvotno smo morali napraviti skriptni objekt koji će služiti kao torba. Skriptni objekt će u sebi imati samo listu „Torba objekata“, kada god pokupimo neki objekt koji može ići u torbu on će biti dodan u listu, te ćemo uvijek bit u mogućnosti doći do bilo kojeg objekta.

Objekt za torbu je skriptni objekt koji određuje sve o nekom objektu koji možemo pokupiti. Objekt ima 4 metode, use metoda koja poziva metodu iz nekog drugog objekta, smanjenje broja metoda koja smanjuje količinu preostalog objekta u torbi (skoro pa uvijek se poziva), saditi metoda, to je metoda koja posadi sjeme iz torbe na pod, prodaja metoda koja oduzme jedan objekt te daje igraču predodređeni iznos novaca (poziva se i signal za uzimanje jer potrebno je osvježiti brojač novaca).

```
    public void Use()
    {
        thisEvent.Invoke();
    }

    public void DecreaseAmount(int amountToDecrease){
        numberHeld -= amountToDecrease;
        if(numberHeld < 0)
        {
            numberHeld = 0;
        }
    }

    public void Plant(){
        if(farmland.RuntimeValue){
        target = GameObject.FindWithTag("Player").transform;
        Instantiate(seed.gameObject, target.position, Quaternion.identity);
        numberHeld -= 1;
        if(numberHeld < 0)
        {
            numberHeld = 0;
        }
        }
    }

    public void Sell(){
        playerMoney.coins += (int)sellValue;
        DecreaseAmount(1);
        pickupSignal.Raise();
    }
}
```

Slika 13: Objekt u torbi (Izvor: vlastita izrada)

Skripta koja upravlja torbom je najveća skripta kada dođe do torbe. Prvo što se dogodi kada je otvorena je da se očiste sva mjesta, zatim se ide kroz skriptni objekt igrače torbe, i za

svaki objekt u torbi se stavlja novi prozorčić koji predstavlja objekt, i postavljamo gumb za korištenje i prodaju da laž da se ne prikazuju.

Kada se trebaju napraviti prikazi objekata z torbi, prvo prolazimo kroz cijelu listu torbe igrača i za svaki objekt stvorimo novo mjesto za prikazivanje. S obzirom na to da pomični pogled ima automatsko sortiranje, ne moramo brinuti gdje će biti postavljeni. Ako pritisnemo na neki objekt, prvo dajemo doznaku torbi da smo izabrali objekt, zatim postavljamo opisni tekst na odgovarajući tekst, prodajnu cijenu isto postavljamo na odgovarajuću cijenu, ako je objekt moguće koristiti, postavljamo gumb za korištenje da je vidljiv, i na kraju, ako je moguće prodati objekt postavljamo gumb za prodaju da je vidljiv.

Ako igrač odluči iskoristiti objekt, taj objekt iskoristi sve svoje efekte zatim se izbrišu smanjuje količina za jedan (ili više ako je to navedeno), izbrišu se svi objekti u torbi, i ponovo se naprave pomoću prethodno opisane metode.

Ako igrač prodaje stvari, događa se skoro pa u potpunosti ista stvar kao i u tome kada se objekt iskoristi, ali, se poziva druga metoda na početku umjesto metode za iskoristiti jer je potrebno povećati količinu novaca igraćeg lika.

```
public void SetTextAndButton(string description, bool buttonActive)
{
    descriptionText.text = description;
    if(buttonActive){
        useButton.SetActive(true);
    }else{
        useButton.SetActive(false);
    }
}

public void SetSellButton(string sellForCoins, bool buttonActive)
{
        sellAmount.text = "" + sellForCoins;
        if(canSell.RuntimeValue == true){
            sellButton.SetActive(true);
        }else{
            sellButton.SetActive(false);
        }
}

void MakeInventorySlots(){
    if(playerInventory){
        for(int i = 0; i<playerInventory.myInventory.Count; i ++)
        {
            if(playerInventory.myInventory[i].numberHeld > 0){
            GameObject tmp =
            Instantiate(blankInventorSlot, inventoryPanel.transform.position, Quaternion.identity);
            tmp.transform.SetParent(inventoryPanel.transform);
            InventorySlot newSlot = tmp.GetComponent<InventorySlot>();
            if(newSlot){
                newSlot.Setup(playerInventory.myInventory[i], this);
            }
            }

        }
    }
}
```

Slika 14: Stvaranje objekata (izvor: vlastita izrada)

```
public void UseButtonPressed()
{
    if(currentItem)
    {
        currentItem.Use();
        //Clear all inventory slots
        ClearInventorySlots();
        //Refill all slots with new numbers
        MakeInventorySlots();
        if(currentItem.numberHeld == 0)
        {
            SetTextAndButton("", false);
            SetSellButton("",false);
        }

    }
}

public void SellButtonPressed()
{
    if(currentItem)
    {
        currentItem.Sell();
        //Clear all inventory slots
        ClearInventorySlots();
        //Refill all slots with new numbers
        MakeInventorySlots();
        if(currentItem.numberHeld == 0)
        {
            SetTextAndButton("", false);
            SetSellButton("",false);
        }

    }
}
```

Slika 15: Prodaja i korištenje objekata (izvor: Vlastita izrada)

5.6. Plijen

U igri je implementiran sustav plijena. Kada se unište bijeli ćupovi koji se nalaze po svijetu, ili kada se uništi čudovište, postoji šansa da iza sebe ostave novčiće, srčeka ili strijele. Novčiće smo prethodno objasnili. Čudovišta imaju najviše šanse za ostavljanje novčića i strijela, dok ćupovi ostavljaju većinom srčeka i novčiće ali nikada strijele.

Kada pokupimo srčeko trenutni životni bodovi se povećaju za jedno srce s time da ne možemo imati više od maksimalnog broja srca.

5.7. Dizajn scena/prostorija

5.7.1. Glavni izbornik (scena 1)

Za izradu glavnog izbornika napravio sam novu scenu, zatim sam u njoj kreirao Canvas. Stavio sam da Canvas ima dijete objekt sliku otoka, dva gumba i tekst. Gumb „New Game“ nas vodi na glavni otok i započinje igru, a gumb „Quit to Desktop“ izlazi iz video igre. Tekst prikazuje ime igre. Za tekst smo koristili posebni asset iz Unity Storea koji se koristi za oblikovanje teksta zvan „Text Mesh Pro“, taj asset je dobar jer koristi vertex grafiku za ispis slova.

Slika 16: Glavni izbornik (izvor: vlastita izrada)

5.7.2. Početni otok

Za dizajn nivoa sam koristio 2 asseta iz Unity Store-a, prvi (drva i neprijatelji) se zove tiny forest RPG, a drugi(prodavač, kućice i općenito ostatak svijeta) je iz „Zeldalike gfx“, kombinacija ta 2 asseta mi je omogućilo da izradim prostor koji želim.

Početni otok se sastoji od dvije sobe koje na sebi imaju zonu okidača u obliku poligona koja služi za ograničavanje kamere. Prva soba je sve osim poluotoka s kućicama, a druga soba je poluotok s kućicama. Između otoka i poluotoka nalazi se prijelaz iz sobe u sobu koji kada se prekorači dobijemo tranziciju kamere koju smo opisivali prethodno.

Po cijelom otoku raspršeni su neprijatelji, ćupovi i sjeme koje možemo pokupiti i posaditi na svojoj farmi koja se nalazi na najsjeverozapadnijem dijelu otoka. Farma je jedino mjesto gdje možemo posaditi sjeme. Na otoku se nalazi i jedna škrinja koja sadrži u sebi luk koji omogućuje pucanje strijela.

Slika 17: Početni otok (Izvor: Vlastita Izrada)

5.7.3. Spilja

Spilja je scena u koju možemo ući iz scene početnoga otoka. Ako igrač pronađe ulaz u spilju ekran će se zabijeliti, te zatim nakon prebacivanja scene od-bijeliti. U spilji možemo pronaći mnoge neprijatelje, i jedno posebno pojačanje koje povećava ukupne životne bodove koje možemo imati, puno ćupova i sva tri sjemena koja su dostupna u igri trenutno.

Za izgled spilje smo koristili Zeldalike GFX assete što mi je omogućilo da napravim spilju kakvu sam zamislio.Scena spilje se sastoji od 2 dijela, ulazni dio i glavna komora. U ulaznom dijelu imamo 3 čudovišta i 12 ćupova. Ako prekoračimo u drugu sobu kamera će napraviti brzi trzaj na granice glavne komore. U glavnoj komori možemo pronaći 4 neprijatelja posebno pojačanje 3 eliksira života i sva tri sjemena koje možemo posaditi na svojoj farmi.

Slika 18:Scena spilja (izvor: Vlastita izrada)

5.7.4. Kraj

Kada igrač skupi 2000 novčića, to mu omogućava bijeg s otoka na kojem je nasukan. Nakon razgovora s jednim sporednim likom upalit će se ekran koji pokazuje da smo došli do kraja igre. Sastoji se od Canvasa na kojem se nalazi poruka da je kraj.

Slika 19: Scena kraja igre (izvor: vlastita izrada)

6. Zaključak

Kada sam tek sakupljao potrebne materijale za igru i gledao kako bih mogao krenuti s projektom očekivao sam da ću u jednom tjednu napraviti igru. Naime nisam shvaćao da će biti toliko posla oko jako malih stvari koje inače u video igrama niti malo pažnje ne obraćamo. Toliko malih skripti koje je bilo potrebno napraviti da bi sve funkcioniralo zajedno kako je zamišljeno

Ova video igra je prvi pokušaj ozbiljnoga rada u Unity Game Engineu, radio sam neke omanje igre i pohađao par kursova, ali nikada nisam toliko duboko zalazio u općenitu izradu video igre, te nisam shvaćao na koliko malih stvari moramo paziti. Tijekom izrade naletio sam na mnogo grešaka koje su mi trebale čak i po nekoliko sati da popravim. Greške su bile nešto tipa da strijele ne lete uopće, greške oko kamere, greške oko prijelaza lika između scena i slične. Mnogi prijatelji su testirali igru i prijavljivali mi greške na koje su naišli. I kada sam dobio neku dojavu o grešci uvijek sam ispitivao točan način kako su došli do greške i kako ju reproducirati. U većini slučaja jako sam se zabavljao tijekom procesa otklanjanja grešaka jer sam dobivao osjećaj uspjeha kada sam uspješno otklonio grešku.

Općenito jako sam zadovoljan s performansom Unitya, dostupnosti materijala, i ljudima koji s radošću pomažu drugima oko njihovih projekata te objašnjavaju načine kako nešto popraviti. Unity zajednica je jako pristupačna i ugodna, i svakome bi preporučio tko želi započeti raditi video igre.

7. Popis Literature

[1] Unity Technologies The world's leading content-creation engine (21. Prosinca 2019). Dostupno na: https://unity.com/learn

[2] Animal Crossing – web stranica (23. Prosinca 2019). Dostupno na: https://animal-crossing.com

[3] Stardew Valley – web stranica (23. Prosinca 2019). Dostupno na https://www.stardewvalley.net

[4] Harvest moon (Video Game) - Wikipedia (24. Prosinca 2019). Dostupno na: https://en.wikipedia.org/wiki/Harvest_Moon

[5] Story of Seasons (Video Game) - Wikipedia (24. Prosinca 2019). Dostupno na: https://en.wikipedia.org/wiki/Story_of_Seasons

[6] Aleksandr – Unity Blog (2014, 3 rujan) *Documentation, Unity scripting languages and you (*25. Prosinca 2019). Dostupno na: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

[7] Unity (Game Engine) - Wikipedia (24. Prosinca 2019). Dostupno na: https://en.wikipedia.org/wiki/Unity_(game_engine)

[8] Brackeys (2019, 11 Kolovoza) *TOP DOWN MOVEMENT in Unity! (*21. Prosinca 2019*). Dostupno na:* https://youtu.be/whzomFgjT50

[9] Code Monkey (2018, 29 Srpnja) *Time Tick System (Unity Tutorial) (*21. Prosinca 2019). Dostupno na: https://youtu.be/NFvmfoRnarY

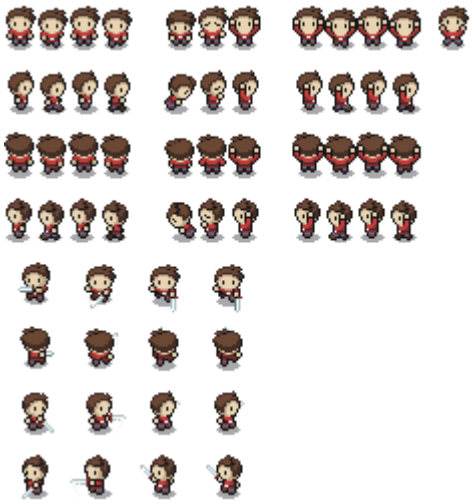
[10] CounchFerret makes Games (2018, 4 Prosinca) *Unity Top Down Colliders and Character Movement – Tutorial (*22.Prosinca 2019). Dostupno na: https://youtu.be/Cry7FOHZGN4

8. Popis slika

9. Popis Korištenih Resursa

1. ZeldaLike Tileset - https://opengameart.org/content/zelda-like-tilesets-and-sprites dostupno 21.12.2019
2. Tiny Forest RPG - https://assetstore.unity.com/packages/2d/characters/tiny-rpg-forest-114685 Dostupno 21.12.2019
3. Text Mesh Pro (asset za oblikovanje teksta) – https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126 Dostupno 2.1.2020

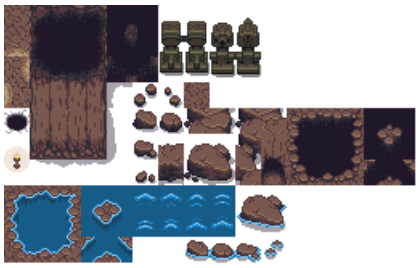
10. Prilozi



Slika 20: Sporedni Lik(izvor: Zeldalike Tileset)

Slika 21: Objekti(izvor: Zeldalike Tileset)

Slika 22: Vanjski Svijet(Izvor: Zeldalike Tileset)



Slika 23: Spilja(Izvor: Zeldalike Tileset)