Debreceni SZC Beregszászi Pál Technikum és Kollégium

4032 Debrecen, Jerikó u. 17.

OM kód: 203033/005

Telefonszám: +36 52 503 150

E-mail:
titkarsag@dszcberegszaszi.hu

Szakképesítés megnevezése: Szoftverfejlesztő és -tesztelő

SZAKDOLGOZAT

Készítette:
Basa Dávid,
Suba Dávid

DEBRECEN
2024

Tartalom

1 BEVEZETÉS

Mi Basa Dávid és Suba Dávid vagyunk. Jelenleg 20 évesek vagyunk, és ez a projekt az egyik legfontosabb kihívás volt eddig számunkra. Az ötletünk számos különböző túlélős játékból származik, amelyeket korábban játszottunk. Mivel mind ketten szerettünk volna, hogy saját játékot hozzunk létre, ez a feladat adott nekünk elegendő bátorságot és motivációt a projekt elkezdéséhez és kidolgozásához.

Miért választottuk a Unity-t? A Unity-t választottuk, mert már korábban mindketten ismerkedtünk a Unity játékfejlesztő platformmal. Első tapasztalatainkat még az iskolában szereztük, ahol egy bevezető programozási kurzus keretében megismerkedtünk az alapokkal. A Unity-t azonnal vonzónak találtuk, mert intuitív felhasználói felülettel rendelkezik, könnyen kezelhető, és rengeteg online tutorial és dokumentáció áll rendelkezésre hozzá. Miután elkezdtünk mélyebben megismerni a Unity-t, egyre inkább lenyűgözött a sokoldalúsága és a lehetőségek tárháza, amit nyújt a játékfejlesztők számára. A Unity által kínált erőteljes grafikus motor, a könnyen testre szabható fizikai motor és az integrált fejlesztői eszközök lehetővé teszik számunkra, hogy a kreativitásunknak csak a képzeletünk szab határt. Ahogy tanultunk és fejlődtünk a Unity használata során, egyre jobban éreztük, hogy képesek vagyunk megvalósítani az álmunkat, és létrehozni saját játékunkat. A projektünkben olyan elemeket szeretnénk felhasználni, mint a szörnyek, a megsemmisíthetetlen fa, az inventory rendszer, a tengeren belüli veszélyek és csodák felfedezése, valamint a hegyes vidékek felfedezése. Ezek az összetevők mind hozzájárulnak egy olyan világ létrehozásához, amelyben a játékosok teljes mértékben elmerülhetnek, és élvezhetik a felfedezés és a túlélés izgalmát. Összességében a Unity választása nemcsak technikai megfontolásokból történt, hanem azért is, mert úgy éreztük, hogy ez a platform a legmegfelelőbb eszköz számunkra, hogy álmainkat valóra váltsunk a játékfejlesztés területén. A Unity lehetővé teszi számunkra, hogy kreativitásunknak és ötleteinknek szárnyakat adjunk, és egy olyan világot hozzunk létre, amelyben mások is örömmel kalandoznak.

Ahhoz, hogy elkészítsük a játékot sok dolgot kellett megtanulnunk. Ilyen volt mindkettőnknek a modellezés tüzetesebb megismerése. A modellezés és animáció tüzetesebb megismerése számos új kihívást és tanulási lehetőséget kínált a projekt során. Bár korábban volt némi tapasztalatunk háromdimenziós tervező szoftverekkel, főként a Maya használatával, a projekt során számos új készséget és technikát sajátítottunk el. A projekt során megtanultunk a

csontozást és az ahhoz kapcsolódó animációkezelést is. A csontozás, más néven rigging, alapvető fontosságú a karakterek vagy objektumok mozgatását szolgáló vezérlőrendszer létrehozásában. Bár ez egy összetett és technikailag kifinomult folyamat, kulcsfontosságú annak biztosításában, hogy az animációs karakterek életre keljenek és hiteles mozgást mutassanak az animáció során. A megfelelő csontozással a karakterek könnyedén manipulálhatók és animálhatók, ami lehetővé teszi az animációs projektünk minőségi és meggyőző végeredményét. Ezáltal az animációink valósághűbbek és vonzóbbak lehetnek a nézők számára, és elősegíthetik a projekt sikerét és hatásosságát. A játékbeli ellenségekre valódi pénzt is költöttünk, körülbelül 30 ezer forintértékben. Ez az összeg a karakterek részletes modellezésére és animálására, valamint azok mozgásának és viselkedésének megvalósítására fordítódott.

2 OVERLOS WEBOLDAL

Úgy gondoltuk, hogy a játékhoz kell csinálnunk egy weboldalt. Két fő részből áll egy oldal, ahol le lehet tölteni a játékot és ahol olvashatsz érdekes információkat és megtudhatsz többet a játékunkról. A másik része egy webshop, ahol pénzért vehetsz tárgyakat, játék béli pénzt és plusz tartalmat a játékhoz. Ezen felől van egy főoldal, ami a weboldal megnyitásakor látszódik, ahonnan át navigálhatsz a játékoldalára vagy a webshopra. Ezen felül van egy regisztrációs és egy bejelentkezés oldal. Ha esetleg elfelejtené a jelszavát, arra is van megoldás a bejelentkezés oldalon megjelenik egy olyan gomb, hogy forget password annak a megnyomásával átírhatja a jelszavát. Az adatokat sql-ben tárólja. Az oldal technikai megvalósításához, php-t, javascript-et, html-t és css-est használtam.

2.1 Megvalósítás

2.1.1 Főoldal:

Sokat gondolkodtam, hogy hogyan nézzen ki a főoldal. Az első verzióban csak szimplán a játék weboldalát hozta volna be felül menüsorral, ahol válthatunk volna a webshopra. Webdizájnok után keresgélve felfedeztem, elég érdekes dizájnokat mint, például Bento UI, ami a japán bento boxokhoz hasonlóan különböző méretű részekre van bontva. Mikor elkezdtem ennek a kialakítását rá kellet jönnöm, hogy ez nem fog beválni mert nincs elég oldalam ahhoz, hogy jól nézzen ki. Így változtatnom kellet és végül egy „Slide show” dizájnt kapott. Ami automatikusan 7 másodperc után és manuálisan az oldalt látható nyilakkal is léptethető. A képernyő tetején egy úgymond butított verziója látható a teljes navigation bárnak. Ahol csak a cím és a login és sign up gombok láthatóak. A képernyő bal oldalán megjelenik az adott Slide címe egy rövid leírás és egy Learn more link, aminek a megnyomásával az adott oldal fog megnyílni. A jobb alsó sarokban az oldalhoz tartozó menüpontok láthatóak, ami itt a navigation bart helyettesíti természetesen ennek megnyomásával is lehet váltani a slidok között nem muszáj a nyilakat használni. A főoldal szerkezeti része az index.php-ban találhatóa html rész megmondja, hogyan jelenjen meg a weboldal. Ebben a részben nem sok php-t használtam mivel nem tartalmaz semmi olyan interakciót a kliens és szerver között, amit php-nak kellene kezelnie. Az egyetlen ilyen az egy $title = ’home’, $page = ’home’, include’./navbar.php’. A $title átadja az navbarnak

az oldal címét, a $page közli a navbárral, hogy milyen oldalon van, ezekről majd a navbar-nál részletesebben beszélek. Az include'./nabar.php' pedig meghívja az oldalra a navbart. A html-rész tartalmaz meta tageket és egy „linket", amivel a css-est hívja be az oldalra. Továbbá tartalmaz három fő „divet". Az első a „slider" itt található egy „lista" div, ami tartalmazza a sliderhez tartozó szerkezeti itemekte, elemeket. Például egy képet, ami az adott oldalhoz tartozó képet tartalmazza és a hozzá tartozó címet és szöveget egy „content" nevű divben tárólja. A második fő div az „arrows", ahol két gombot tartalmaz, amivel javascript segítségével balra vagy jobbra görgethetünk a képek között. A harmadik fő div a „thumbnail", ahol a sliderhez hasonlóan található egy kép, ami az adott oldalhoz tartozó képet tartalmazza és a hozzá tartozó címet. Az index.php végén egy „script" tag található, ami a javscriptet, hívja be az oldalba. A javscript az oldal viselkedéséért és az oldalon történő interakciókért felelős. A javascriptben interaktívvá tehetünk gombokat, linkeket vagy egyéb szerkezeti elemeket. Az index.js-ben meghívtam a slider listáinak elemeit, a két nyíl gombot és a thumbnail itemeket. Ezeket a document.querySelectAll-al() és a document.getElementById() függvényekkel hívtam be. Egy countItem nevű változóban megszámlálom az slider lista itemek hosszát, amit a showSlider() functionál használtam. Ebben a functionben bizonyos időközönként, azaz 7 másodpercenként az eppén következő slider list item kap egy active taget ezzel jelezve, hogy melyik aktív és a régiről pedig levesz ezt a taget. Agy másik részen ugyan ezt megteszi a thumbnail itemekkel is. Ezen kívűl van a két nyíl gombot is lekezeltem, ami megnyomásánál hozzá ad egyet az itemActive nevű változóhoz és meghívja a showSlide() nevű functiont így létetve a slidok között. Cursore kinézetet is állítottam, ha a kép felé viszi az egeret simából pointert csinál. Van egy függvény, ami az oldal responzivitásában segédkezik. Mégpedig a thumbnaileket egy 480px-el egyenlő vagy alacsonyabb szélességű eszközökön gombokká alakítja őket és megadja az elérési útját és hogy milyen oldal.php fájlt kell megnyitnia. A css-ről nem sok mindent tudok mondani ez felel az oldal kinézetéért és responzivitásáért. Különböző id-kal vagy class.okkal kell ellátnunk a html tagaket hogy külön külön az id-k segítségével vagy egy csoportba rendezve class-ok segítségével formázzuk, módosítsuk a kinézetét elhelyezkedését. A weboldal responzivitásáért a @media fele, ahol megadhatjuk, hogyan nézzen ki az oldal egy bizonyos képernyő szélesség elérésénél vagy a képernyő típusát is megkülönböztethetjük. Én az utóbbit használtam, mégpedig a @media only screen and (pointer: coarse) {} és a részlet az t csinálja, hogy ha a képernyő érintő képernyős az ebbe a részbe található stílus formázást jeleníti meg.

2.1.2 Sql:

Az sql-hez való csatlakozást a connect.php-ban kezeltem le. Azért csináltam neki egy külön php fájlt, mert így nem kell minden egyes php fájlba megírnom az sql connectiont, hanem csak require el behívom. Synatxis: require'./connect.php'. A $con = new mysqli() parancsal megadom az sql hez való csatlakozást a hozzá tartozó jelszót és felhasználó nevet és, hogy melyik adatbázisból szeretném a táblákat és a táblában lévő adatokat elérni. Ha esetlegesen valamilyen probléma adódna a csatlakozással egy hiba üzenetben kiírja a problémát. Majd a karakter kódolást utf-8-ra állítom.

2.1.3 Game:

Ezt az oldalt a game.php tartalmazza a játékhoz kapcsolódó információkat érdekességeket, amikről úgy véltem érdemes megosztani. A php-rész úgy, mint az főoldalnál tartalmaz egy $titlet, $paget és a include navabart ezeknek a funkcióját nem írom le még egyszer. A html-rész ugyanúgy tartalmaz egy game.css -re mutató hivatkozást. Itt is a @media only screen and (pointer: coarse) {} -t használtam a responzivitás beállításához. A body-rész tartalmaz egy headert amiben a php kapott helyet, tartalmaz egy main részt és egy footer részt. A main részbe írtam meg az oldal főbb szerkezeti részeit. Adtam az oldalnak egy fő címet, alatta egy párs oros bekezdésben összefoglalva leírtam a játékot és az elkészítéséhez vezető motivációnkat és a jövőbeli terveinket a játékkal kapcsolatban. Itt kapott helyet a játék letöltéséhez szükséges link. Ennek a megnyomásával letölthetted a játékot a számítógépedre vagy laptopodra. Itt kapott helyet egy rövid történet is a játék főtörténete, ha lehet így mondani. Az oldal alján csináltam egy ugyan olyan slidert mint a fő oldalon és a javascriptnek is az index.js-t hívtam meg mivel ugyan azt csinálja mind a kettő, így nem szerettem volna újra írni még egyszer csak más néven. A footerben megjelöltem a az alkotókat és figyelmeztetést tettem, hogy az oldalt copyright védi. A copyright védelem az esetleges lopások kizárásáért lenne felelős, de teljes védelmet nem biztosít viszont adatlopásnál perelhető a fél, ha kiderül az adatlopás.

2.1.4 Webshop:

Ez az oldal hozza be a webshopot, ha ezt a menüpontot választod. Ez az oldal a shop.php-ban kapott helyet. Ez már több php-t és javasriptet igényelt, mint a többi, de kezdem az elején. A htmlnek a head- részében található található egy ion-icon script, amivel egy weboldalról leszedhető ikonokat lehet használni ezzel színesítve a weboldal kinézetét. A body részben itt is helyet kapott egy header, ami tartalmazza a navbárnak szükséges adatokat. Helyet kapott egy

főcím, három fő div, egy bekezdés, aminek nagy szerepe van és ugyan úgy, mint a többinél egy footer és a javascrptet maghívő script tag. Ehhez az oldalhoz tartozik egy restapi-hoz hasonlító php fájl. Ez a php fájl a shopapi.php ez kezeli a webshop sql.hez vonatkozó kérelmeit, utasításait. A fájl megnyitásakor az elején látható a connect.php-nak a meghívása, a $con érték globálissá tétele és egy $method nevű változó, ami elkéri a klienstől, hogy a szerver milyen methódust hajtson végre $method = $_SERVER['REQUEST_METHOD']; Az oldal további részében switch($method)-dal kezelhetjük le a különböző methódusokat. Az első, amit lekezeltem az a 'GET' methódus. Ezzel a methódussal értékeket kaphatunk az sql szervertől a bizonyos feltételek megadása után. Jelen esetben megkértem, hogy az sql products nevű táblájából szedje ki az adatokat és tegye bele egy products nevű többe, majd az egy jsonként küldje el a webshopnak és tárolja el, ennek a kiíratásához javascriptet használtam. A következő a 'POST' methodus, aminek a segítségével új rekordokat adhatunk az sql adatbázisunkhoz. Viszont ez így mindenkinek elérhető így a shop.php-nak az ehhez tartozó #addd divben lekorlátoztam az elérést, hogy csak az admin nevű felhasználó tudja elérni ezt a methódust. Ezentúl bemeneti mezőket adtam meg, hogy hozzá is lehessen adni az értékeket. Különlegessége, hogy képet is lehet feltölteni, ami először nagyon meglepett, de elég könnyű és érthető forrásokat és videoókat találtam, ami elmagyarázta ennek a megvalósít. Így képes voltam elkészíteni a sajátomat. A szerver bekér egy fájlt, amit egy általa létrehozott mappában letáról és az elérési utvonalat linké változtatja, ami segíti, hogy ne legyen túl nagy az sql fájl. A következő methódus a Delete methódus, amit sajnos gyakorlati használatban nem teljesen tudtam megvalósítani, mivel a gomb amire meghívtam az ehhez szükséges eljárásokat mindenkinek látszódott és nem tudtam sehogy se lekorlátozni így kénytelen voltam kivenni az, viszont a kódot benne, hagytam. A javascriptben lekezeltem az 'GET' methodusból kapott értékeket elsőnek egy datába letaroltam a jsonból kapott adatokat, majd egy fetchAllProducts() functionnel egy allProducts[] nevű tömbben letaroltam az egészet. Egy renderProducts() nevű functionel kiírattam egy product-container nevű divbe, ami a container nevű divben kapott helyet egy webshopon belüli menüvel és egy kereső sorral, amiket javascriptel interaktívvá tettem. Mégpedig úgy, hogy a menusor gombjainak a textját kiszedtem és úgy futtattam le a renderproductot hogy csak azok az elemek jelenjenek meg, amik a gomb textjére hasonlítanak. A keresősorban pedig, ha elkezdi beírni valamilyen termék nevét akkor úgy módosítja a kiírt elemeket, hogy csak azok jelenjenek meg amik tartalmazzák azt. Az addToCart() funkcióban lekezeltem a kosárhoz adást. Megnyomunk egy gombot, hozzá adja a kosárhoz. Ez a kosár

ugyan úgy a container nevű divben kapott helyet, egy kosár gombbal és egy számlálóval, ami megszámolja, hány elemet adtál hozzá. Tovább ki is veheted az itemet a mellette lévő kuka gombbal. Van egy total count ami össze számólja a kosárban lévő itemek árát és kiírja azt. Van egy checkout nevű gomb, aminek a megnyomásával előhozhatjuk a fizetés formátumot, ami a buy nevű divben kapott helyet, ehhez viszont a bankapi.php tartozik ebben a php-ban kezelem le, hogy ha első vásárlás akkor adja hozzá a beírt adataidat az sql personal nevű táblájába. A vétel befejezése után egy p bekezdésben kiírja a megvet termékekhez, tartozó kódokat, amiket unityin belül beválthatsz arra, amit megvettél. Továbbá lekezeltem kisebb dolgokat, például a kosár csak akkor látszódik, ha megnyomod a kosár gombot. A kosár üres kosárral nem enged tovább a buy menüponthoz hiba kiírása után folytathatod a böngészést a webshopban. A css-ben úgy alakítottam ki az oldalt, hogy a balóldalon legyen egy menüsor középen jelenjen meg a webshop termékei és jobb szélen megjeleníthető legyen a kosár. A buy formátumot elrejtettem és csak akkor jelenik meg, ha a kosár tartalmaz valmait és a checkout gombra nyomunk. A resposivitást itt is @media only screen and (pointer: coarse) {} -rel oldottam meg. Érintőképernyős eszközön a menü átváltozik egy gombbá, aminek a megnyomásával legördül a menü középen a termékek láthatóak egymás alatt és jobb oldalon megnyitható a kosár.

2.1.5 Login:

A login oldalt a navbarban található login gombbal lehet előhozni. Megnyomásával egy loginformátum jelenik meg a képernyőn egy másik ablakban. A formátum a login.php nevű fájlban található. Ebben a fájlban található egy loginformátum, aminek a kitöltésével beléphetünk, ha nincs fiókunk, akkor van esélyünk az álján lévő register gombra nyomva, hogy regisztráljunk az oldalra. Ha már van fiókunk, abban az esteben a logingomb megnyomásával beléphetünk az oldalra. Ennek az ellenőrzésére az auth.php felel. Ami azt csinálja, hogy formba beírt adatokból kiszedi a felhasználó nevet megnézi, hogy létezik-e az adatbázisban ilyen felhasználó, ha igen $user változóba visszaadja a felhasználóhoz tartozó adatokat, amiket az oldal később feltud használni. Ha esetleg nem a főoldalon jelentkeztél be, akkor bejelentkezés után visszadob arra az oldalra, ahol a bejelentkezés előtt voltál. Egyéb funkciók, ha a remember mere rányomsz akkor 30 napig megtartja a bejelentkezésed az oldalra az oldal elhagyása után is.

2.1.6 Forget password:

Ha elfelejtetted volna a jelszavad, itt meg tudod változtatni a felhasználó neved beírásával megnézi, hogy létezik, ha igen megkért, jelszavakat ellenőrzi, hogy egyfórmák-e és módosítja az adatbázisban. A forget passwordot a bejelentkezésnél, érhetjük el.

2.1.7 Regisztráció:

A regisztráció a signup.php-ban kapott helyet ezt is a navbarban található signup gombbal érhetjük el ennek a megnyitásával egy másik ablakban előjön egy loginhez hasonlító formátum, kibővítve, miután beírtuk a szükséges adatokat a formátum elküldi őket a regauth.php -nak, ami megnézi, hogy létezik-e már az a felhasználónév és megnézi, hogy az email címmel nem e regisztráltak már korábban, mind a két estben hibaüzenetet ír ki, ha valamelyik teljesül: létezik a felhasználónév vagy, már regisztráltak erre az email-re. Ha nem a regauth INSERT INTO-van be insertálja az adatokat az sql users nevű táblába és megnyitja a bejeletkezési oldal és bejelentkezés után, böngészhetünk a weboldalon.

2.1.8 Setting:

Belépés után a login és a signup gomb eltűnik és helyette egy profil hoz hasonló gomb jelenik, meg aminek, ha velé visszük az egeret megjelenik egy legörülő menü, ahol rámehetünk a beállításokra vagy kijelentkezhetünk. Ha a beállítások gombra nyomunk akkor megnyílik a beállítások oldal, ahol ugyanúgy megtalálható a php változók, mint a többi oldalnál. Az oldalon található egy menüsor, amiben különböző műveletket hajtathatunk végre az oldallal. A profile settings fülben megváltoztathatjuk a felhasználó nevünket és email címünket, amit a settapi.php kezel a többi methodussal együtt. A felhasználónév vagy email változtatásnál bekéri az új értéket, amit elküld a php-nak, ami megnézi, hogy a felhasználónév vagy email nem használt-e már ha igen, hibaüzenetet ír és nem teljesíti a kérést. Amennyiben nem használt id alapján, amit a bejeletkezés utáni sessionel nyerhetünk ki, megkeresi a változtatni kívánt rekord helyét és megváltoztatja a kívánt rekordot. A change password fülben megváltoztathatod a jelszavad. Le ellenőrzi, hogy a két jelszó, amit beaírtál egyezik-e, ha igen módosítja a jeszót. A delet account fülben a felhasználó neved beírása után törli az adataidat.

2.1.9 Navigationbar:

A navigationbar a navabr.php ban található. Ennek a feladata az oldalak közötti váltakozás kezelése. A többi oldalról lekéri a $tiltle és $page változóka, amivel az oldalhoz igazítja a menüsor paramétereit.

2.2 Tapasztalatim

Érdekes kihívás volt számomra, nem gondoltam volna, hogy a webfejlesztés nehéz is tud lenni. Természetesen tapasztalat nélkül nehéz megvalósítani, de szerintem sokat fejlődtem az elmúlt időszakban. A weboldal és a css kialakításánál törekedtem a felhasználó barát használat megtartására és egy letisztul környezet kialakítására.
Felhasználói útmutató:
Az oldal megnyitásakor a főoldalon tudunk tovább navigálni a játék oldalára vagy a webshopra. Ezt a bal oldalt látható Learn more… felirat megnyomásával tehetjük. A játék oldalán játékról tudhatunk meg többet. A webshopon pedig tartalmat vehetünk a játékhoz. A webshop bal oldalán a menüsorban kereshetünk vagy kategória szerint rendezhetjük a termékeket. A termékeknél az Add to Cart gomb megnyomásával hozzáadhatjuk az általunk választott terméket a kosarunkhoz. Ami a jobb oldalt található. A megnyitásához nyomjuk meg a kosár gombot, ah szernénk megvásárólni a terméket nyomjuk meg a checkout gombot. Ilyenkor felugrik egy fizető oldal, ahova bekell írni az adatainkat és megkapjuk a termékhez tartozó kódot, amit a játékba beválhatunk. A navigációs bárban válthatunk oldalt, regisztrálhatunk és beléphetünk. Belépés után a gombok átváltoznak profil füllé, ahol beállíthatjuk a profilunkat vagy kijeletkezhetünk.

3 OVERLOS

A játékon belül a karakter mozgás elkészitésével kellett kezdenünk. A MoveControl osztály kulcsfontosságú elem a játékélmény szempontjából, mivel biztosítja a karakter mozgásának rugalmasságát és egyensúlyát, függetlenül attól, hogy a játékos belső vagy külső nézetből játszik-e. Ez az adaptabilitás és sokoldalúság kulcsfontosságú a játékélmény megfelelő biztosításában, hiszen minden játékos saját preferenciáinak megfelelően testreszabhatja a karakter mozgását. Az osztály átfogó funkcionalitást nyújt, kezdve az alapvető mozgással, mint például az ugrás és a sprintelés, egészen a speciális helyzetek kezeléséig, mint például a vízben való mozgás vagy a dőlések reakciója. Ez a sokoldalúság lehetővé teszi a karakternek, hogy minden környezeti kihíváshoz alkalmazkodjon, és élvezhesse a játék folyamatát.Az osztály nemcsak a karakter mozgását, hanem annak életszerűségét is figyelembe veszi. Például a gravitáció szabályozása és a környezeti tényezőkkel való interakció mind hozzájárulnak ahhoz, hogy a karakter mozgása természetes és valósághű legyen. Ezáltal a játékosok könnyedén bele tudnak merülni a játék világába anélkül, hogy zavarná őket a mesterséges vagy nem valósághű mozgás. Az osztály által biztosított funkcionalitás és életszerűség jelentősen javítja a játékélményt, és lehetővé teszi a játékosok számára, hogy teljes mértékben élvezzék a játékot és annak környezetét. Amikor karaktereket animáltam, sok szenvedésen mentem keresztül. Ez egy kihívást jelentő és nagyon nehéz folyamat volt, ugyanakkor hihetetlenül kielégítő is. Számtalan órát töltöttem azzal, hogy minden egyes animációt aprólékosan finomhangoltam, képkockáról képkockára, próbálva megörökíteni a mozgás és az érzelem lényegét. Voltak pillanatok, amikor frusztráltam, mert valami nem úgy nézett ki, ahogy szerettem volna, vagy az animációk nem folytak gördülékenyen. Folyamatosan haladnom kellett, finomítanom és csiszolnom kellett minden egyes animációt annak érdekében, hogy elérjem azt a minőséget és valószerűséget, amire törekedtem ez sok esetben sikerült azonban az emberében nem. De minden kihívás ellenére volt egyfajta elégedettség és sikerélmény minden animáció elkészültével. Nagyon jó vált látni, ahogy a karakter életre kel a képernyőn, ahogy mozog és viselkedik a tervek szerint. Ez egy szeretet munkája volt, és bár néha nagyon nehéz volt, nem cserélném el a tapasztalatot semmire sem. Türelmet, kitartást és figyelmet a részletekre tanított meg az animációban. Végül minden kemény munka meghozta gyümölcsét, és büszke vagyok arra, amit sikerült elérnem.

3.1 MapControl

A spawnolást a mapcontrol script kezeli. Ez a script felelős a térkép kezeléséért a Unity játékban. Beállítja a játékos helyzetét a spawn pontokhoz, aktiválja/deaktiválja a térképet, figyeli a játékos magasságát, és kezeli a játékos halálát.Van játék kezelőnk például a szünet beállító. A "PauseMenu" szkript egy játékmenü kezelő, amely lehetővé teszi a játékosnak a játék felfüggesztését és folytatását, valamint a játékmenübe való áttérést és a játék bezárását. A szkript fontos része a játékélménynek, mivel lehetővé teszi a játékosnak a szüneteltetést és az alapvető beállításokhoz való hozzáférést. Amikor a játékos megnyomja az Esc billentyűt, a szkript ellenőrzi, hogy éppen megnyitott-e más menü (például az inventárium vagy a térkép). Ha nem, akkor a játék felfüggesztődik vagy folytatódik a Pause() és Resume() metódusok segítségével. A felfüggesztés során a játékmenü megjelenik, és az idő leáll, míg a folytatásnál a játékmenü eltűnik, és az idő újra folytatódik. A loadMenu() metódus lehetővé teszi a játékosnak, hogy visszatérjen a fő menübe, és ehhez a jelenlegi játékot betölti újra. A QuitGame() metódus segítségével a játékos kiléphet a játékból. Ez a szkript fontos része a játékélménynek, mivel lehetővé teszi a játékosnak az alapvető műveletek elvégzését anélkül, hogy kilépne vagy zavaróan hatna a játékfolyamatra.A "GameTimeManager" osztály egy olyan eszköz a játékban, amely segíti a játékosokat abban, hogy mélyebben elmerüljenek a játék világában és élményt kapjanak az időjárási és napszaki változásokkal. Az osztályban található változók és beállítások lehetővé teszik a játékmenet számára, hogy finomhangolja az időjárást és a napszakokat, amelyek közvetlen hatással vannak a játékélményre. A "currentTime" változó tárolja az aktuális játékbeli időt, míg a "SunLatitude" és "SunLongitude" változók meghatározzák a napmozgás szögét és helyzetét az égbolton. Ezáltal a játékosoknak korlátozott befolyásuk lehet az időjárási viszonyokra és a napszakokra, ami a játék világának hangulatát és dinamikáját alakítja. Az osztály "UpdateTimeText()" módszere felelős az idő folyamatos frissítéséért és az időjárási viszonyok megfelelő beállításáért az időszerű játék béli idő alapján. Ez a módszer figyeli az aktuális időt és az ahhoz kapcsolódó változásokat, például a napkelte és napnyugta idejét, valamint az éjszakai hangulat megjelenítését.Az "UpdateLight()" metódus gondoskodik a nap- és holdfény megfelelő beállításáról az adott időjárási viszonyoknak megfelelően. Ez a metódus biztosítja, hogy a nap- és holdfény intenzitása és színe folyamatosan változzon az időjárás és a napszakok változásával, így élethűbb és atmoszférikusabb játékélményt nyújtva a játékosoknak. A "CheckShadowStatus()" metódus segítségével ellenőrzi az árnyékok státuszát az adott napszak és időjárási viszonyok alapján. Például biztosítja, hogy

a napkelte és napnyugta idején a napfény árnyékolást generáljon, míg éjszaka a holdfény legyen domináns az árnyékok kialakításában.Végül, a "SkyStar()" metódusfelelős az égbolt és a csillagok megfelelő megjelenítéséért az éjszakai időszakban. Ez a módszer biztosítja, hogy az éjszakai égbolt élethű legyen, és megfelelően reagáljon az időjárási viszonyok változásaira.

3.2 GameEventInterface

A "GameEventInterface" osztály számos funkciót lát el a játékban, amelyek közvetlenül befolyásolják a környezetet. A változók és metódusok összehangolt működése révén lehetővé teszi a dinamikus időjárási viszonyok kezelését és az azokhoz kapcsolódó események kiváltását a játék során. Az osztály egyik fő funkciója az idő és az időjárás kezelése, melyet a "GameTimeManager" osztállyal való együttműködésen keresztül valósít meg. Az időjárás szimulációja során a változók tárolják a napszakokhoz és időjárási viszonyokhoz kapcsolódó adatokat. Például a "SunLongitude" és "MoonLongitude" változók meghatározzák a nap és hold pozícióját az égbolton, míg a "StormIsActive" változó figyeli, hogy éppen vihar van-e.Az időjárási események véletlenszerűségen alapulnak, ami változatosságot és életszerűséget kölcsönöz a játéknak. Például a "ChangeEverthing()" metódusban történő véletlenszerű napszakváltás és ködszimuláció hozzájárul a játék világának életszerűségéhez és változatosságához. A "WeatherUpdate()" és "FogChance()" metódusok segítségével figyelik és irányítják a játékban zajló időjárási változásokat és eseményeket. Például a "WeatherUpdate()" metódus meghatározza, hogy mikor kell aktiválni a ködöt az adott időpontban, míg a "FogChance()" metódus döntést hoz arról, hogy milyen valószínűséggel kell generálni ködöt az adott időszakban.Ebben a scriptben van a referencia a „NormalClouds”-ra is felhők és az ahhoz köthető időjárás milyenségéhez. Ez a szkript felelős a felhők és azokhoz kapcsolódó időjárási hatások kezeléséért. Egyik legfontosabb funkciója a felhők paramétereinek dinamikus változtatása, ami lehetővé teszi különböző időjárási körülmények létrehozását a játékban. Ezek a változások magukban foglalják a felhők sűrűségét, alakját, erózióját, elhelyezkedését és eltűnését, melyek összességében hozzájárulnak a játék világának életszerűségéhez és változatosságához. Emellett a szkript lehetőséget biztosít az eső, villámlás és viharkülönböző típusú időjárási hatásainak aktiválására a játék során. Ezek az időjárási hatások dinamikusan reagálnak a játék eseményeire és a játékosok cselekedeteire, így növelve a játék teljesítményét,

ha esetleg nem lenne értelme az adott időjárási események fenntartásához, ilyen mikor a játékos a víz alatt van.

3.3 EffectScript

Az effektekről beszélve az effekteket az „EffectScript” script irányítja. Az osztályban számos változó található, amelyek tárolják az adott hatásokhoz szükséges adatokat és referenciákat. Például a "player" változó tárolja a játékos GameObjectjét, ahol a hatásokat megjelenítjük, míg a "RainObj" és "LightningObj" változók a felhőzápor és villámlás vizuális elemeket tartalmazzák. A "Components()" metódus inicializálja ezeket a változókat és beállításokat a szkript indításakor. A "EffectAllow()" metódus lehetővé teszi az összes hatás inaktívvá tételét egyetlen hívással. Az "Update()" metódusban a "LateUpdate()" függvény figyeli a játékos mozgását és a jelenlegi időjárási viszonyokat, és dinamikusan frissíti a csapadék helyzetét a játékoshoz viszonyítva. A "JustRain()", "JustThunder()", "Thunderstorm()" és "StormAndThunder()" metódusok lehetővé teszik különböző időjárási viszonyok aktiválását a játékban, például eső, villám vagy vihar.Az "RainChanges()" metódus lehetővé teszi az eső intenzitásának dinamikus változtatását időben, például amikor az esőerősség változik az idő múlásával vagy más játék események hatására.

3.4 ItemPlacement

Az időjáráson kívül még fontos volt a "ItemPlacement" osztály. Mivel alapvető fontosságú volt a játékbeli tárolórendszer megvalósításában és működésében. A jól strukturált és áttekinthető kód segíti az elemek hatékony kezelését és a játékmenet sima lefolyását. A megfelelően implementált metódusok és változók lehetővé teszik a rugalmas és testreszabott játékélmény kialakítását. Ez az egy kulcsfontosságú része az elkészített alkalmazásomban, mivel ez felelős az item elemek elhelyezéséért és kezeléséért a játékbeli tárolórendszerben. Ez az osztály számos fontos funkcióval és logikai állítással rendelkezik, amelyek nélkülözhetetlenek a játék megfelelő működéséhez. Továbbá e z az osztály felelős azért, hogy kezelje a játékos táskáját és a gyorselérési sávot, valamint biztosítsa a tárgyak felvétele és letétele közötti folyamatot. Rendkívül sok változóval rendelkezik. Az „instance” változó az osztály egy példányára mutat,

ami lehetővé teszi a más osztályokból való hozzáférést ehhez az osztályhoz. Tehát az Inventorys lista a játékos táskájában lévő tárgyakat és azok mennyiségét tárolja. Ez azért fontos, mert így könnyen nyomon követhetjük, hogy a játékos milyen tárgyakkal rendelkezik és azokból mennyi van. Az InventoryPlaces lista pedig a gyorselérési sáv helyeit tárolja. Ez a lista meghatározza, hogy a gyorselérési sávban mely pozíciókra kerülhetnek a tárgyak. Ez fontos a játék szempontjából, mert így a játékos könnyen hozzá tud adni vagy el tud távolítani tárgyakat a gyorselérési sávból.A ToolbarPlaces lista a gyorselérési sávban lévő tárgyak helyeit tárolja. Tehát ha a játékos hozzáad egy tárgyat a gyorselérési sávhoz, akkor annak a helyét ebben a listában tároljuk. Ez segíti a játékot abban, hogy megjelenítse a tárgyakat a megfelelő helyen a gyorselérési sávban.Az InventoryItemPlaces lista a gyorselérési sávban lévő tárgyakat és azok mennyiségét tárolja. Tehát minden tárgyhoz egy megfelelő helyet rendelünk a gyorselérési sávban, és itt nyomon követhetjük, hogy ezeknek a helyeknek milyen tárgyak vannak azokon. A ToolbarUIPlaces lista pedig a gyorselérési sáv UI elemeit tárolja. Ez a lista felelős azért, hogy megjelenítse a gyorselérési sávban lévő tárgyakat a felhasználói felületen. Így tudjuk biztosítani, hogy a játékos látja és kezelni tudja a gyorselérési sávban lévő tárgyakat. Az ItemContent és ToolbarContent változók a játékban lévő tárgyak tartalmát és a gyorselérési sáv tartalmát jelölik. Tehát az ItemContent például egy olyan tartalmat jelöl, amelyben a játékban lévő tárgyakat tároljuk és megjelenítjük, míg a ToolbarContent a gyorselérési sávban lévő tárgyakat tárolja és jeleníti meg.Az InventoryItem változó pedig egy prefabot jelöl, amely felelős az egyes tárgyak megjelenítéséért. Ez a prefab lehet például egy olyan objektum, amely tartalmazza a tárgy ikonját, nevét vagy egyéb részleteit, és amelyet dinamikusan hozunk létre és jelenítünk meg a játékban a játékban lévő tárgyakhoz vagy a gyorselérési sávban lévő tárgyakhoz. Természetesen változókon kívül rengeteg metódussal és eljárással rendelkezik. Elsőként a ChildUpdate() függvényről beszélve, ez felelős az eszköztár és az inventory frissítéséért. Amikor ez a függvény lefut, az összes eddigi elemet eltávolítja az eszköztárból és az inventory-ból, majd frissíti azokat az új adatokkal. Minden egyes elemet az eszköztárból (ToolbarPhysicContent) és az inventory-ból (ToolbarContent) tölt be, és ezeket tárolja az InventoryPlaces, ToolbarPlaces, InventoryItemPlaces és ToolbarUIPlaces változókban. A Update() függvény az egész játék futása során folyamatosan fut. Ez ellenőrzi az inputokat és a játék állapotát. Ha az input egy szám, akkor az ChangeSelectedSlot() függvény segítségével változtatja az aktuális kiválasztott slotot az eszköztáron. Ha a játékos megnyomja a "ZeroItem" billentyűt, akkor a kiválasztott slotot 10-re változtatja, és egy másik függvényt hív meg, ami az

adott akciót végezni fog.Amikor a játék szünetel, vagy az inventory vagy a térkép megnyitva van, a ScriptEnabler() függvény segítségével letiltja a játékmenetet, és egy "SomethingIsOpen" változó segítségével jelez, hogy valami meg van nyitva. Ha nincs semmi megnyitva, akkor visszaállítja a játékmenetet.Az ItemSelect() függvény a kiválasztott slotot kezeli az eszköztáron. Ha egy slot kiválasztva, akkor engedélyezi az adott sloton lévő elemek megjelenítését, és a többit letiltja. A HasChildren() függvény egyszerűen ellenőrzi, hogy egy adott játékobjektumnak vannak-e gyerekei vagy sem, és ezt a logikát a különböző helyeken használjuk a kód során, hogy meghatározzuk, hogy az adott objektum üres vagy sem. Az Add() függvény felelős az új elem hozzáadásáért az inventory-hoz. Először is megvizsgálja, hogy az elemet hozzá lehet-e adni egy már meglévő elemhez, ha az stackelhető (stackable) és van már egy azonos típusú elem az inventory-ban, akkor növeli a meglévő elem számát eggyel. Ha az elem nem stackelhető, vagy nincs még azonos típusú elem az inventory-ban, akkor megnézi, hogy az elemet hova lehet elhelyezni az eszköztáron (Toolbar). Ha van olyan hely az eszköztáron, ahol még nincs más elem, akkor létrehoz egy új objektumot az inventory-ban, beállítja annak adatait, és frissíti az UI-t az új elemmel. Ha az inventory megtelt (több, mint 60 elem), akkor beállítja a FullInv változót igazra. Az UpdateUI() függvény felelős az inventory UI-jának frissítéséért az új elem hozzáadása után. Létrehoz egy új objektumot az InventoryItem prefab-ból, beállítja annak adatait az új elemhez, majd hozzáadja az új elemet az Inventorys listához és az items listához. Az InventoryUpdate() függvény felelős az inventory UI-jának teljes frissítéséért. Ez azért fontos, hogyha például több elemet törölnek az inventory-ból egyszerre.A Remove() függvény eltávolítja az adott elemet az inventory-ból. Ha az eltávolítandó elem az eszköztárban található, akkor a ChangePictures() függvénnyel visszaállítja a képet az adott sloton. Az InventoryRemover() függvény felelős az adott elem eltávolításáért az Inventorys listából. Ez a függvény meghívódik, amikor az elemet eltávolítják az inventory-ból. A ChangePictures() függvény megváltoztatja az adott indexű elem képét és átlátszóságát az inventory UI-jában. A függvény először beállítja az elem képét az img paraméter által megadott képre, majd az átlátszóságot a Transparency paraméter alapján. Ezután az új képet és átlátszóságot beállítja az adott indexen található inventory elemre. Az InventoryRemover() függvény eltávolít egy elemet az inventoryból. Az items listából eltávolítja az adott InventoryItem objektumot. A Placer() függvény egy transzformációt ad vissza az adott indexű helyhez az inventoryban. Ha a megadott index egyenlő egy hely indexével az InventoryPlaces listában, akkor visszaadja az adott helyre mutató transzformációt. Ha nincs

találat, akkor null-t ad vissza. Az AddWeapon() függvény hozzáad egy új fegyvert az inventoryhoz az adott indexű helyre. Először megkeresi az adott indexű helyet az InventoryPlaces listában, majd létrehozza a fegyvert a megadott prefab alapján, és hozzáadja azt a megtalált helyhez. A CheckAllIndex() függvény ellenőrzi az összes elem indexét az inventoryban, és frissíti azokat, ha szükséges. Ez hasznos lehet például akkor, ha az elemeket átrendezik vagy törlik, és az indexek frissítése szükséges a megfelelő működéshez. A CheckAllIndexToolbar() függvény ugyanezt teszi az eszköztárban (Toolbar), azaz ellenőrzi az összes elem indexét az eszköztáron, és frissíti azokat szükség esetén. A ListRemoveer() függvény eltávolítja az adott indexen található elemeket a két listából, az InventoryItemPlaces-ből és a ToolbarPlaces-ből. A ListAdder() függvény hozzáad elemeket a megadott indexre a listákhoz. Az InventoryItemPlaces listához hozzáadja az item-et az a indexű helyre, a ToolbarPlaces listához pedig a place transzformációt. Emellett eltávolítja és hozzáadja az a indexű elemet a ToolbarUIPlaces listából a toolbar paraméter alapján. A Death() függvény a játékos megölésért ellenőrzi. Először leejti az összes elemet az inventoryból, majd törli az összes elemet mind a két listából (Inventorys és items). Ezután inaktívvá teszi a Toolbar tartalmát, aktívvá teszi a Backup-ot, és letiltja a játékost.A Resurect() függvény újraéleszti a játékost. Aktívvá teszi a Toolbar tartalmát, a játékost, és letiltja a Backup-ot. A AnimationCheck() függvény ellenőrzi, hogy az adott helyen van-e animáció. Ha van, nem csinál semmit, ha nincs, akkor a MoveControl-nak változtatja a cselekvést. Végezetül ez az osztály fontos szerepet játszik abban is, hogy biztosítsa az UI frissítését az új elemek hozzáadása vagy eltávolítása után, valamint az inventory teljes frissítését, ha szükséges. Ezen kívül figyelmet fordít az olyan eseményekre, mint a játékos halála vagy újjáélesztése, és megfelelően válaszol ezekre az eseményekre az inventory és a Toolbar sáv állapotának kezelésével.Az osztály rendkívül sokoldalú és alapvető fontosságú a játék teljes működésében, és minden egyes része szorosan kapcsolódik a többihez annak érdekében, hogy a játékélmény zavartalan legyen és a játékosok könnyen kezelhessék az elemeket és tárgyakat. Ezen itemeket az item osztály kezeli. Az item osztályban adunk meg minden olyan dolgot amit, a játékban gyűjthetnek vagy használhatnak a játékosok. Ez a kód egy új tárgyat hoz létre, amit majd a játékban fel tudunk használni vagy meg tudunk jeleníteni a UI felületen. Az „ItemName” csak egyszerűen a tárgy neve, például "Heal +100" vagy "Sword" ez mutatja meg a játékosoknak, hogy mi a pontos neve a náluk lévő objektumnak. Az „InventoryPlacement” azt mondja meg, hogy az adott tárgy hova kerül a játékos táskájában vagy felszerelésében. Lehet például olyan,

ami bekerül a Toolbar-ba vagy olyan, ami csak a fő Inventoryba. A következő érték a „stackable” ami azt mutatja meg, hogy a tárgyak egymásra rakhatók-e az inventoryban vagy sem. Például a koponyák általában egybe gyűjthetőek, míg például az egy fegyver biztos, hogy nem ilyen tárgy. Az Icon az kép, amely megmutatja a tárgyat a játékosnak a képernyőn a toolbar-ban vagy a fő inventory-ban. Természetesen ebben szerepel az adott tárgy Prefab-ja ami az az előre elkészített objektum a játékban, amelyet a tárgy használ, például egy koponya, amit a játékos gyűjt vagy egy gyógyuló főzettek, amit használ. A „MaxItem” azt mutatja, meg hogy mennyi darabot lehet egy adott tárgyból egy időben magánál hordani, ha az a tárgy egybegyűjthető. A „DropLocation” az a hely a játékban, ahol a tárgy megjelenik, ha a játékos kidobja. A „Toolbar” azt mondja meg, hogy az adott tárgyat lehet-e rakni a Toolbarba vagy sem, ez fontos abból a szempontból, hogy a felszedés során elég meghatározó hogy hová tegye a tárgyat. A „quantity” azt mutatja, meg hogy hány darab van az adott tárgyból, ez a maxitem szempontjából fontos valamint ezzel könnyítjük meg a maximum inventory kiszámításához kellő adatot. A „soulPrice” azt mutatja hogy mennyibe kerül ez a tárgy hogyha a” Soulshop”-ban vásárolnád meg. A „code „azt az azonosítót vagy kódot tartalmazza, amit a webshopban lehet venni. vagy azonosító a tárgyhoz. A potion egy olyan jelző, ami megmondja, hogy ez a tárgy megiható-e az inventory-ból vagy pedig nem lehetséges. A heal az érték, amely a fentebb említett potion nevű jelzőhöz tartozik. Ez meghatározza, hogy mennyit gyógyít, de nem mellesleg ez add továbbá értéket a sebezhetetlené tévő főzetnek is.

3.5 ItemDefault

Az elemek felvétele legtöbbször az "ItemDefault" osztállyal történik meg. Ennek az osztálynak a célja, hogy kezelje és irányítsa azokat a tárgyakat, amelyeket a játékosok fel tudnak venni és használni a játékban. Ehhez az osztályhoz mint mindegyikhez is rengeteg dolog szükséges. Van egy Rigidbody változó, ami a tárgyhoz tartozó fizika komponenst tartalmazza. A coll változó egy olyan alkatrészt tárol, ami a tárgyhoz kapcsolódó collider-t reprezentálja, ami az objektum fizikai hatásainak kezelésében segít. Az outlines változó felelős azért, hogy ha az objektumhoz tartozik egy külső keret, az aktív legyen vagy sem. Vannak olyan logikai változók is, mint az interactable, ami megmondja, hogy az adott tárgy interaktív-e vagy sem ez az objektum felvevésénél fontos, valamint a PickedUp, ami azt jelzi, hogy a játékos felvette-e már az adott tárgyat. Majd, néhány gombot is beállítunk a játékmenethez: A PickupKey és a DropKey

változók a felvételhez és a letételhez használt billentyűkódokat tárolják. Az Update függvényben ellenőrizzük, hogy van-e interakció az objektummal, ezt a kamerán lévő colliderel segitségével tehetjük meg. Mely ellenőrzi hogy az adott tárgy rendelkezik item nevű layer-rel vagy taggal. Ha ez a tárgy rendelkezik vele és ütközik-e a kamera collider-évl, ezt az ütközést az OnTriggerEnter, OnTriggerStay és OnTriggerExit függvényekben kezeljük. Ha a játékos közelében van egy tárgy, akkor az objektum körvonalát aktiváljuk, hogy jelezzük, hogy interakcióba lehet lépni vele. akkor megnézi, hogy a játékos lenyomta-e a felvétel számára meghatározott gombot. Ha igen, és van hely a játékos inventory-ába, akkor megnézi, hogy az adott tárgy milyen kritériumoként mehet bele az inventory-ba. Ha az objektum már fel van véve, akkor ellenőrizzük, hogy lenyomták-e a letétel gombot, és ha igen, letesszük az objektumot. Az ItemPlacement osztály egyéb metódusait is hívjuk a PickItUp és DropItDown függvényekben, hogy azok megfelelően kezeljék a tárgyak felvételét és letételét. Összességében ez a kód részletesen irányítja és kezeli a játékos által felvehető és használható tárgyakat. Minden felszedés után a tárgy egy neki megfelelő „Inventoryitem” scriptben találja meg a maga helyét. Ezen script az inventory toolbar-ban és a main Inventory-ban található meg. Hogy a megfelelő értékeket lássanak, a felhasználok rengeteg változóra van szükség. Az„image” ,ez a változó egy referenciát tárol egy kép objektumra. Az Image változó felelős azért, hogy megjelenítse a képet a felhasználói felületen. Ez a változó konkrétan az InventoryItem-hez tartozó képet tárolja. Az InventoryTracker ez egy másik kép objektum referenciáját tárolja. Ez az objektum segít nyomon követni az InventoryItem pozícióját az inventory rendszerben, lehetővé téve a tárgyak helyes megjelenítését és kezelését. Az Itemname egy string változó, ami tárolja az InventoryItem nevét. A név megkönnyíti az azonosítást és a tárgyak közötti kommunikációt. Például egy fegyver esetében ez lehet "", vagy egy kulcs esetében "Aranykulcs". Az item egy referencia az Item osztály egy példányára. Az Item osztály tárolja a tárgy teljes információit, például a nevét, típusát, tulajdonságait stb. Ez a változó teszi lehetővé az InventoryItem számára, hogy hozzáférjen a tárgyhoz kapcsolódó minden adathoz és viselkedéshez. A counter, ez egy egész szám, amely tárolja az InventoryItem darabszámát. Például egy kulcsnál ez lehet a mennyiség, ha több ugyanolyan kulcsot is tarthatsz. A privilousCounter egy egész szám, ami tárolja az előző darabszámot, amit az InventoryItem kezelésében használnak. Segít nyomon követni, hogy változott-e a darabszám az előző ellenőrzés óta. A textcount ez egy referenciát tárol egy TextMeshProUGUI objektumra. A TextMeshProUGUI osztály lehetővé teszi számunkra a szöveges információk megjelenítését a felhasználói felületen. Ebben az esetben ez

az objektum a darabszámot jeleníti meg az InventoryItem mellett. Az ezekre épülő metódusok és eljárások a következők. A „CountItemIndex()” metódus, ez a metódusfelelős az InventoryItem indexének meghatározásáért az InventoryItem szülőjében (általában egy lista vagy konténer). A metódus végig megy a szülő minden gyermekén, és megtalálja az aktuális GameObject indexét. Ha nem találja meg, az index értéke -1 marad. Az import() metódus, ez a metódus inicializálja az image és textcount változókat, amelyek hivatkozást tárolnak a képre és a számláló szövegre, amelyek az InventoryItem-hez tartoznak. Az inicializálás a transzformálás segítségével történik a gyermek objektumok között. Az OnBeginDrag() metódus meghívódik, amikor az InventoryItem húzása elkezdődik az egér által. Létrehoz egy másolatot az eredeti objektumról, amelyet húzni szeretnénk, és beállítja annak a helyét a húzási helyre. A raycastTarget értékeket is módosítja, hogy megakadályozza a kattintásokat az InventoryItem-ekre húzás közben. Az OnDrag() metódus ennek a folytatása, tehát meghívódik, amikor az InventoryItem-et húzzuk. Frissíti az objektum pozícióját a kurzor helyzetével. Az OnEndDrag() metódus ez a metódus a vége az onDrag methódusnak, ez akkor hívódik meg, amikor az InventoryItem húzássa véget ér. Ellenőrzi, hogy hova engedtük el az InventoryItem-et, majd visszaállítja a raycastTarget értékeket, hogy újra engedélyezze a kattintásokat. A HandleDrop() metódus, akkor kezeli az InventoryItem-et, amikor azt elengedjük egy másik GameObject-en. Ellenőrzi, hogy hova helyeztük el az InventoryItem-et, és ez alapján meghívja a megfelelő függvényeket vagy cselekvéseket. A Placing() metódus a helyére teszi az InventoryItem-et a húzás befejezése után. Visszaállítja az InventoryItem szülőjét a parentAfterDrag változó értékére, majd beállítja az objektum testvérek közötti indexét a itemIndex változó értékére. Ha az InventoryItem címkéje "Inventory-Items", akkor beállítja az „ObjName” szövegét a „Itemname” változó értékére. A Placinger() metódus helyére teszi az InventoryItem-et. Beállítja az InventoryItem szülőjét a saját szülőjére, majd beállítja az objektum testvérek közötti indexét a itemIndex változó értékére. Change() metódus megváltoztatja két objektum testvérek közötti indexét. Először meghatározza a cselekvő és a cél objektumok indexét. Ha a cselekvő objektum indexe kisebb, akkor a cselekvő objektum kerül a cél objektum elé, ellenkező esetben a célobjektum kerül a cselekvő objektum elé. Ezután frissíti az objektumok itemIndex változóit és cseréli a neveiket. A ChangeLocation() metódus megváltoztatja két objektum szülőjét. Először eltárolja a cél objektum szülőjét, majd áthelyezi a célt az adott objektum szülőjére, és az adott másik objektumot a cél objektum eredeti szülőjére helyezi. A RecreateScript() metódus, újra létrehozza az InventoryItem-et a jelenlegi helyén. Először visszaállítja az InventoryItem

szülőjét a saját szülőjére, majd beállítja az objektum testvérek közötti indexét a itemIndex változó értékére. Ezután hozzáad egy új InventoryItem komponenst az objektumhoz, majd törli az eredeti InventoryItem komponenst. A LocationSwitch() metódus megváltoztatja két objektum indexét a helyükön. Ha az első index kisebb, akkor először eltávolítja az első objektumot, majd hozzáadja a második helyére, és ugyanezt teszi a második objektummal. Ellenkező esetben először eltávolítja a második objektumot, majd hozzáadja az első helyére, majd ugyanezt teszi az első objektummal.

3.6 Soul Shop

Ezen scriptek elemit használja továbbá használja még a „soulshop” is mely lehetőséget biztosit ezen itemek meg vételére a játékban. A soulshop egy olyan objcum, amely egy olyan játékobjektumra vonatkozik, amelynek van egy Collider komponense és a fő célja az, hogy érzékelje, amikor egy másik játékobjektum belép az őt körülvevő kollíder területébe.Az OnTriggerEnter() és OnTriggerStay() metódusok azokat az eseményeket kezelik, amikor egy másik játékobjektumot érzékel kollíder. Ezekben a metódusokban ellenőrizve van, hogy a belépő objektum címkéje "MainCamera" és hogy lenyomták-e az "E" billentyűt, valamint azt is, hogy a SoulShop osztály SoulShopActive tulajdonsága igaz-e. Ha minden feltétel teljesül, akkor meghívódik a soulShop változóhoz tartozó Switcher() metódus,. A soulshopban is számos változó található. A SoulShopActive egy olyan statikus bool változó, ami jelzi, hogy éppen aktív-e a SoulShop. A SoulShopPaying egy egész szám változó, ami tárolja a lehetséges fizetendő lelkek mennyiségét, ha esetleg valaki nem csak egy dolgot szeretne venni. A SoulObjects egy olyan Item tömb, amely tárolja az itt kapható elérhető tárgyakat. A SoulContent egy olyan transform, amely tartalmazza a boltban elérhető tárgyak listáját. A Page1, Page2: GameObjectek, amelyek a boltban elérhető két oldalt reprezentálják. Az InputField: Egy TMP_InputField, amely azt a szövegmezőt jelzi mely activálja a kodót a webshopból. Az AlertObj egy GameObject, amely egy figyelmeztetést add ki hogy sikeres volt e a vásárkás vagy az adott kód be aktiválását jelenít meg a játékosnak. A Start() metódus inicializálja a boltot és a különböző változókat. Létrehozza a boltban elérhető tárgyakat és beállítja a megfelelő pozíciójukat a kosárban. Az AddToShop() metódus új tárgyakat ad a boltba, létrehozva egy új objektumot a SoulContent transzformáció alatt és beállítva a megfelelő nevüket, árát és ikonjukat. A Switcher() metódus állítja be a bolt állapotát (aktiválja

vagy deaktiválja), beállítja a megfelelő láthatóságot és letilja vagy engedi a játékos mozgását a bolt megnyitásakor. Az Update() metódus figyeli, hogy lenyomták-e az "Escape" billentyűt, és ha igen, bezárja a boltot. Az Off() metódus bezárja a boltot és törli a kiválasztott tárgyakat. A Buttonchange() metódus váltakozik a bolt két oldala között. A CheckTheTexts() metódus ellenőrzi a beírt kódokat és hozzáadja a megfelelő tárgyat az inventáriumhoz. A Pay() metódus kezeli a fizetést a boltban kiválasztott tárgyakért, ellenőrzi, hogy elegendő lelket rendelkezik-e a játékos, majd frissíti az inventáriumot és törli a kiválasztott tárgyakat a kosárból. AddToBasketText(): Ez a függvény hozzáad két új szöveges objektumot a kosárba. Az első objektum a termék nevét, a második pedig az árát jeleníti meg. A DestroyBasketItem() függvény törli az adott nevű elemet mindkét kosárból az index alapján. Ezt követően frissíti a többi elem indexét, hogy azok megfeleljenek a változásnak. Az UpdateIndex Ez a függvény frissíti a termékek indexét a kosárban az eltávolított elem után. Így biztosítja, hogy az indexek helyesen álljanak be a többi elemhez képest. A DeselectItems függvény visszaállítja az összes kiválasztott elemet a kosárban. A kiválasztott elemeket általában vizuálisan kiemelik vagy más módon jelölik, és ez a függvény biztosítja, hogy ezek a jelölések eltűnjenek. A NextItemPlace függvény visszaadja a következő üres hely indexét a kosárban. Ez segíti a rendszernek abban, hogy tudja, hol kell elhelyezni az új elemeket a kosárban. Az AddItemsToInventory függvény hozzáadja a kiválasztott elemeket a készletbe a kosárból. Amikor egy vásárló befejezte a vásárlást, és a kosár tartalmát át kell helyezni a készletbe vagy a vásárolt elemek listájába, ezt a függvényt használják.

Ezen Soulshophoz a soulshop item osztály járul, hozzá mivel ez határozza meg az egyes elemek viselkedéséért a lélek boltban. A backgroundv2, v1, v2 változók tárolják az elem hátterének színét. A backgroundv2 az aktuális hátterszínre hivatkozik, míg a v1 és v2 a kattintásra és a nem kattintásra meghatározott színeket jelölik. A clicked változó jelzi, hogy az adott elemre kattintottak-e vagy sem. A SoulShopObj változó tárolja a lélek bolt objektumot, amelynek a része az adott elem. Az item a változó tárolja az elemhez tartozó adatokat, mint például az elem neve és ára. Az ItemIndex változó tárolja az elem indexét a lélek boltban. A Start() függvényben inicializáljuk az elem hátterének színét és a szükséges változókat. A Click() függvény reagál az elemre való kattintásra. Ha az adott elemre kattintanak, akkor kiválasztódik, a hátteres szín megváltozik, és hozzáadjuk az elem árát a lélek boltban levő összes többi elem árához. Ha az elem már kiválasztott, akkor a kiválasztás törlődik. Az Update() függvény ellenőrzi, hogy a lélek bolt aktív-e, és ha nem, akkor törli a kiválasztást az elemről. A DeselectItem() függvény

eltávolítja az elem kiválasztását, visszaállítja az alapértelmezett hátterszínt, csökkenti a lélek boltban levő összegzett árat, és eltávolítja az elemet a kosárból. Más megjelenitett tárgyakhoz az area nevü osztályból haszbáltuk a scripteket. Ez az Area osztály felelős az adott területen található gyerekelemek működésének kezeléséért. A Coll, layerMask, count, PlayerIsThere változók segítenek a terület kezelésében és az ott lévő elemek kezelésében. A Coll tárolja a terület colliderét, a layerMask azonosítja a területen lévő objektumokat, a count számolja az adott területen lévő gyerekelemek számát, míg a PlayerIsThere jelzi, hogy a játékos az adott területen van-e vagy sem. A Start() függvényben inicializáljuk a változókat, beállítjuk a layerMask értékét, majd meghívjuk a DelayedStart() függvényt egy kis késleltetéssel. A DelayedStart() függvényben meghívjuk a ChildrenChange() függvényt a gyerekelemek állapotának beállítására. Az OnTriggerEnter(), OnTriggerStay(), és OnTriggerExit() függvények az adott területbe való belépésre, ott tartózkodásra és kilépésre reagálnak. Ha a játékos belép a területre vagy ott tartózkodik, akkor meghívják a ChildrenChange() függvényt az elemek megjelenítésére. Ha a játékos kilép a területről, és nincs más játékos a területen, akkor a gyerekelemek inaktívvá válnak. A ChildrenChange() függvény felelős a gyerekelemek állapotának megváltoztatásáért. Ha csak egy gyerekelem van, akkor az aktív/inaktív státuszát változtatjuk. Ha több gyerekelem van, akkor mindegyikük állapotát megváltoztatjuk. Általában ennek a gyermekei a spawnobject és sp1 tárgyak és enemy spawnolását segítettek.

3.7 SpawnEnviramentObj

A SpawnEnviramentObj osztályfelelős környezeti objektumok generálásáért egy adott területen belül. A SpawningOba tömbben tárolódnak azok az objektumok, amelyeket létre kell hozni és elhelyezni a területen. A Ground és Water rétegmaszkok határozzák meg, hogy mely rétegeken lehet generálni objektumokat. A width és a height a collider méretei az adott területen. A counter, maximum: Ezek a változók segítenek követni, hogy mennyi objektumot hoztunk már létre, és mennyi a maximum, amit létre kell hoznunk. A max Az adott collider maximális magassága. A spawnPosition a véletlen helyen létrehozandó objektumok helyét tároló vektor. A Coll az adott terület colliderének referenciája. A Start () függvényben inicializáljuk a változókat, beállítjuk a rétegmaszkokat, és meghívjuk az ObjectGenerate() függvényt, hogy létrehozzuk a környezeti objektumokat. Az ObjectGenerate() függvény a következőket teszi, az InWater változó alapján meghatározza, hogy az objektumok vízben helyezkednek-e el vagy

sem. Egy while ciklusban generál véletlenszerű helyet az objektumoknak az adott területen belül. Ellenőrzi, hogy az adott helyen van-e víz (Watertest függvény), és ha igen, új helyet generál. Növeli a counter változót, és addig folytatja a ciklust, amíg el nem éri a maximális értéket. A Watertest() függvény meghatározza, hogy az adott helyen van-e víz vagy sem, és az InWater változó alapján tér vissza igaz vagy hamis értékkel. Az ehhez kapcsolódó területeken spawnolnak az ellenességek a sp1 nevü scripttel. A Sp1 osztály felelős a játékon belül az ellenségek spawnolásáért az adott területen belül. Az enemyPrefab a tömbben tárolódnak azok a prefabelemek, amelyek az ellenségek megjelenítéséért felelősek. A spawnRate az ellenségek spawnolási sebessége, amely meghatározza, hogy milyen gyakran jelennek meg az új ellenségek. A spawnCount és MaxSpawn változók segítenek követni, hogy hány ellenséget hoztunk már létre, és mi a maximális létrehozható ellenségek száma. A canSpawn a változó vezérli, hogy folytatható-e az ellenségek spawnolása. A Ground és Water a rétegmaszkok határozzák meg itt is hogy , hogy mely rétegeken lehet generálni objektumokat. A width és height a collider méretei az adott területen. A CanThereSpawn változó jelzi, hogy az adott helyen lehet-e spawnolni ellenséget. A spawnPosition itt is a létrehozandó ellenség helyét tároló vektor. Title: Egy üres játékobjektum, amely tartalmazza az ellenség prefabejeit csoportosítva. Az Sp1 osztályban a SpawnEnemies metódus egy IEnumerator, ami lehetővé teszi az ellenségek folyamatos spawnolását a játék futása közben. Ez a metódus újratöltődik egy véletlenszerű időközönként, amelyet a spawnRate változó határoz meg. Amíg a canSpawn változó igaz és a spawnCount kevesebb, mint a MaxSpawn, addig a metódus meghívja a Spawning metódust, ami létrehoz egy új ellenséget a megfelelő helyen, majd növeli a spawnCount értékét. A Spawning metódusban először véletlenszerűen kiválaszt egy ellenség prefabot, majd létrehozza azt a GetValidSpawnPoint metódus által visszaadott érvényes spawn ponton. Ezután az ellenség méretét és szülőjét beállítja, majd növeli a spawnCount értékét. A GetValidSpawnPoint metódus véletlenszerű spawn pontot generál az adott területen belül, majd ellenőrzi, hogy az a hely érvényes-e az ellenség spawnolására a Watertest metódussal. Ha nem megfelelő a hely, új spawn pontot generál, amíg talál egy megfelelőt. A Watertest metódus ellenőrzi, hogy a megadott spawn pont megfelelő-e az ellenség spawnolására, figyelembe véve, hogy van-e víz az adott pont közelében. Végül a NewSpawnRate metódus frissíti a spawnRate változót egy új véletlenszerű értékkel, hogy változatosabbá tegye az ellenségek spawnolási sebességét. A sok ellenség, tárgy és környezet miatt nem árt a nem látható tárgyakat eltüntettetni a nem látható tárgyakat a teljesitmény hibák Az LagSolution osztály olyan mechanizmust valósít meg, amely

lehetővé teszi az összes gyermekjáték objektumának scriptjeinek kikapcsolását, ha egy bizonyos feltétel teljesül. Nézzük részletesen: scripts: Egy tömb, amelyben tároljuk az objektum összes MonoBehaviour típusú scriptjét. enable: Egy logikai változó, amely jelzi, hogy az objektum scriptjei jelenleg engedélyezve vannak-e vagy sem. A Start metódusban először lekérdezzük az objektum összes MonoBehaviour típusú scriptjét és tároljuk azokat a scripts tömbben. Ezután egy OverlapSphere-t használunk, hogy ellenőrizzük, vannak-e környező collider-ek az objektum körül. Ha nincs egyetlen collider sem a közelben, meghívjuk az OverDisable metódust. Az OverDisable metódusban először beállítjuk az enable változót hamisra. Ezután végigmegyünk az összes tárolt scripten és kikapcsoljuk azokat, kivéve az LagSolution osztályt és az esetlegesen meglévő collidereket. Emellett minden gyermekjáték objektumot kikapcsolunk és letiltjuk az esetleges MeshRenderer és Rigidbody komponenseket. Az OverEnable metódusban először ellenőrizzük, hogy az enable változó hamis-e. Ha igen, akkor minden scriptet engedélyezünk az LagSolution osztályon kívül, bekapcsoljuk a MeshRenderer és a Rigidbody komponenseket, valamint minden gyermekjáték objektumot aktiválunk. Ezután beállítjuk az enable változót igazra. Az ellenségeink beállítása se volt egyszerű mivel volt ahol 2 rétegü animációt és volt ahol csupán egy rétegű animációt kellett használni. FinalOneLayerEnemyScript osztály felelős az egyrétegű ellenségek viselkedéséért a játékban. Az Enemy változóban tároljuk az Enemy osztály példányát, amely részletezi az ellenség tulajdonságait. Agent egy NavMeshAgent komponens, amelyet az ellenség mozgatására használunk.A Ground és Player: Rétegmaszkok, amelyek meghatározzák, hogy hol található a talaj és a játékos. A player játékos transzformációját tároló változó. A Damagers Egy lista az ellenség által kibocsátott károkat végző objektumok transzformációiról. Heal: Az ellenség életének mennyisége. A CurrentHeal a z aktuális gyógyulási állapotot jelző változó. A healDifference: Egy logikai változó, amely azt jelzi, hogy van-e különbség az előző és az aktuális gyógyulási állapot között. Az animator az ellenség Animator komponense. A currentAnimation Az aktuális lejátszott animáció neve. Az attack: Az aktuális támadás animáció neve. A NextAttack egy logikai változó, amely jelzi, hogy a következő támadást kell-e végrehajtani. A sightRange, sightAngle és az attackRange a látótávolság, látószög és támadási távolság az ellenség számára.

A walkPoint, walkPointSet, walkPointRange a séta célállomását, a célállomás beállítását és a séta célállomásának hatótávolságát tároló változók. A fly egy logikai változó, amely jelzi, hogy az ellenség repülni tud-e vagy sem. A Canthreath egy logikai változó, amely jelzi, hogy az

ellenség képes-e fenyegetni vagy sem. Spawn: Egy változó, amely jelzi, hogy az ellenség már-e inicializálódott-e vagy sem. Death: Egy logikai változó, amely jelzi, hogy az ellenség meghalt-e vagy sem. Az osztályban számos privát és nyilvános metódus található a különböző viselkedések és funkciók kezelésére, például Patroling, ChasePlayer, SearchPlayer, GetHit, FieldViewCheck stb. Ezek a metódusok felelősek az ellenség mozgásának, támadásának, keresésének, sebezhetőségének, látótávolságának stb. kezeléséért. Az osztályban használt funkciók közé tartozik továbbá az ellenség gyermekeinek ellenőrzése és a hozzájuk tartozó scriptek hozzáadása (AddScriptToChildrenWithCollider), valamint az ellenség megtalálása a látótávolságon belül és a látószögben (FieldViewCheck). Az ellenség támadási animációjának kezelése során a támadások véletlenszerűen választhatók ki a rendelkezésre álló támadások közül (SwitchAttackMethod). Az FinalTwoLayerEnemy osztályban számos funkció került hozzáadásra vagy finomításra az előző verzióhoz képest. Itt a támadási animációk időzítése Coroutine-ok segítségével történik, lehetővé téve az időzítés finomhangolását és a támadások dinamikusabb kezelését. Emellett két külön animációs réteget használ, ami lehetővé teszi a különböző animációk független kezelését és rétegezését, ezáltal bonyolultabb viselkedést valósítva meg. A kód több ponton is modulárisabb és áttekinthetőbb, például az AddScriptToChildrenWithCollider és a DamageCheck függvények újrafelhasználhatóak és hatékonyak. Ezáltal az FinalTwoLayerEnemy osztály egy átfogóbb és rugalmasabb megoldást kínál az ellenség karakterek viselkedésének implementálására a játékban.

4 ÖSSZEGZÉS

Létrehoztunk egy valódi fizikával rendelkező játékot túlélő, RPG játékot, időjárási viszonyokkal leküzdhető ellenfelekkel érdekes környezettel, tájjal, amit fel lehet fedezni. Gyűjthető soul-lal, amiből különböző tárgyakat és fegyvereket vehetsz. Létrehoztunk mellé egy weboldalt, ahol többet tudhatsz meg a játékról és a webshopban vehetsz termékeket a játékhoz ezzel is támogatva minket. Sok mindent terveztünk még a játékhoz és a weboldalhoz is például: armpur, skill level, több élőlény, több fegyver. A weboldalhoz pedig: egy oldal rólunk a kódok finomhangolása, működő e-mail rendszer. Ezeket főként a tapasztalatlanságunk és az időhiány miatt nem tudtunk megcsinálni.

5 IRODALMIJEGYZÉK

Játék modellek:

https://www.cgtrader.com/3d-models/character/woman/base-mesh-4f9c9a15-5498-46d2-81cf-a72efa414188

https://www.cgtrader.com/3d-models/animals/other/giant-4dfb9b7e-1451-452b-a1fb-9c9cb9a9357c

https://www.cgtrader.com/3d-models/animals/mammal/low-poly-black-wolf-rigged-animated-3d-model

https://www.cgtrader.com/3d-models/animals/mammal/raja-the-white-bengal-tiger-3d-model

Video:

https://www.youtube.com/watch?v=USmwEDVM2Qc&t=6435s

https://www.youtube.com/watch?v=j7GG009J9uc&t=973s

https://www.youtube.com/watch?v=zZ6vybT1HQs

https://www.youtube.com/watch?v=lkIFF4maKMU&pp=ygUKamF2YXNjcmlwdA%3D%3D

https://www.youtube.com/watch?v=lfmg-EJ8gm4&t=5s&pp=ygUKamF2YXNjcmlwdA%3D%3D

Formázás:

https://www.youtube.com/watch?v=0VMlEhnQXsA

https://www.youtube.com/watch?v=ibKNiyK1Nws&t=2s

https://www.youtube.com/watch?v=lnX8WwiGghA