VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Kursinis darbas

Kompiuterinių žaidimų kūrimas naudojant Unity platformą

(Development of computer games using the Unity platform)

Atliko: 3 kurso 2 grupės studentas
Tautvaldas Gumbis

Darbo vadovas:
dr. Igor Katin

Vilnius, 2024

TURINYS

ĮVADAS

Šiais laikais kompiuteriniai žaidimai tapo itin populiaria veikla, o jų kūrimui žmonės dažniausiai naudoja žaidimo kūrimo varikliukus. Šiais laikais *Godot, Unreal Engine, GameMaker* ir, žinoma, *Unity* yra bene dažniausiai naudojami varikliukai. Būtent pastarosios platformos suteikiami įrankiai žaidimų kūrimui ir bus apžvelgiami šiame darbe. Unity leidžia rinktis iš keleto projektų tipų kuriant žaidimą, priklausomai nuo įrenginio, kuriam bus skirtas žaidimas (telefonui, kompiuteriui, ar žaidimas su virtualios realybės akiniais). Taip pat vertėtų atkreipti dėmesį ir į žaidimo dimensiją (ar žaidimas kuriamas 2D, ar 3D formatu).

Šiame darbe kuriamas žaidimas bus 3D erdvėje, veikėjų vizualus perteikiant 2D formatu. Žaidimo idėja – keliaujantis burtininkas - daktaras, gydantis įprastus gyventojus renkant specifinius ingredientus pateiktame žemėlapyje. Tačiau žaidėjo valdomam pagrindiniam veikėjui trukdo įvairūs monstrai, atimdami krepšius su ingredientais ar puldami patį herojų. Pagrindinis veikėjas turi dvi atakas, galinčias jį apsaugoti nuo šių monstrų. Šiame darbe bus apžvelgiama, kaip galima tai pasiekti naudojant įvairius Unity įrankius bei komponentus, reikalingus siekiant įgyvendinti ne žaidėjo valdomų veikėjų veiksmus. Ties tuo žaidimas nesibaigs, kursinio darbo projektui bus siekiama praplėsti žaidimo funkcionalumą iš vieno žaidėjo žaidimo į galimybę žaisti internete su draugais ar nepažįstamais žmonėmis.

Problematika: Norint sukurti kuo įvairiapusiškiau geresnį bei žaidėjui patrauklesnį žaidimą, privalu naudoti kompleksines technologijas bei įrankius. Tinkamo varikliuko pasirinkimas irgi yra labai svarbus žingsnis, skirtingose platformose yra skirtingi privalumai bei trūkumai.

Kursinio darbo tikslas: Naudojant Unity platformą sukurti 3D erdvinį žaidimą su monstrų, žaidėjų ir ingredientų sistemomis, lyginant Unity ypatumus su kitų žaidimų kūrimo platformų galimybėmis.

Kursinio darbo uždaviniai:

- Išanalizuoti Unity platformos pranašumus pradedančiųjų žaidimų kūrėjų atžvilgiu;
- Išanalizuoti Unity navigacijos sistemą ir jos sąveiką su žaidėjo nevaldomais veikėjais;
- Sukurti žaidėjo valdomo veikėjo ir kitų žaidimo objektų sąveikas;

1. Žaidimų kūrimo platformų analizė

Lyginant technologines platformas visuomet yra svarbu, jog naudojamos platformos (Unity, Unreal Engine ar Godot) būtų paprastos naudoti, tačiau turėtų įvairių funkcijų, didelę žmonių bazę, leidžiančią rasti daugiau informacijos apie naudojamą technologiją.

1.1. Unity platforma

Ši žaidimų kūrimo platforma yra viena žinomiausių. Jau 2005 metais Unity buvo pristatyta viešai, o tai leido palaipsniui sukurti stabilų produktą, su kuriuo buvo sukurti tokie populiarūs žaidimai, kaip „Assasin‘s Creed: Identity ” bei „Deus Ex: Fall ” [ES17]. Laikui bėgant, Unity varikliukas tapo vienu pagrindinių žaidimų kūrėjų pasirinkimų. Unity programavimo kodams naudojama C# programavimo kalba.

Unity žaidimų varikliukas yra populiarus ir turi gerą kūrėjų sukurtą dokumentaciją. Internete didelė bendruomenės dalis diskutuoja bei dalinasi vaizdo įrašais įvairiausiais klausimais bei temomis [NBH+20]. Unity yra dažniau vertinama kaip pradedančiųjų žaidimų kūrėjų platforma dėl draugiškos vartotojo sąsajos ir didžiausios bendruomenės [Law20]. Tačiau su patirtimi žaidimo varikliuko galimybės tikrai neapriboja vartotojo kurti įvairiausias žaidimų mechanikas, Unity turi labai daug įrankių ir išteklių tam, jog būtų galima įgyvendinti norimas idėjas.

Šiuo varikliuku galima kurti žaidimus efektyviai įvairioms platformoms – kompiuteriams, konsolėms, mobiliesiems telefonams, virtualiai realybei. Tai leidžia Unity platformai išlikti visapusiškai ir pasiekti bet kokią auditoriją.

Unity gali būti tinkamesnis pasirinkimas pradedantiesiems kūrėjams, ieškantiems platformos su gausia bendruomene. Paprasta vartotojo sąsaja lengvina įsitraukimą į varikliuko daugybę siūlomų funkcijų.

1.2. Unreal Engine platforma

Unreal Engine daugiausia naudoja C++ kalbą ir pirminė varikliuko versija pasirodė 1998 metais.

Šis varikliukas turi išskirtinumą tuo, jog siūlo vartotojams galimybę sukurti labai aukštos kokybės fizinės aplinkos, veikėjų ir pasaulio kraštovaizdžio kūrimo funkcijas [SMJ22].
Vis dėlto, tai yra sudėtingesnis varikliukas, dažnu atveju naudojamas didelių kompanijų su daug patyrusių darbuotojų, siekiant atkurti kuo realistiškesnių grafikų žaidimus, pavyzdžiui, „Hogwarts Legacy".

1 pav. Nuotrauka iš „Steam" parduotuvėje esančio „Hogwarts Legacy" žaidimo, kurto Unreal Engine varikliuku.

Ši platforma taip pat yra labai populiari, turi savo dokumentaciją bei bendruomę, tad kilus techniniams klausimams interneto forumuose tai gali padėti rasti atsakymus [NBH+20]. Unreal Engine suteikia galimybę kurti vizualiai stulbinančią aplinką ir yra tinkamas projektams, kuriems reikalingos aukščiausios kokybės grafikos.

Unreal Engine optimizacija ir integruoti įrankiai padeda lengviau kurti ne tik žaidimus, tačiau ir vizualizacijas ar vizualizacijų programinę įrangą įvairiais mąsteliais. Unreal Engine naudojamas ir kaip perteikimo varikliukas tam tikruose filmuose [BPS+20].

1.3. Godot platforma

Vienas iš pasirinkimų, norint kurti žaidimus, yra platforma Godot. Tai atviro kodo žaidimo varikliukas, reiškiantis, jog vartotojai turi didelę laisvę plėsti platformą įvairiais savo kuriamais patobulinimais [RM24]. Kadangi tai yra atviro kodo projektas, augant bendruomenei, Godot galimybės gali sparčiai plėstis.

Godot, kaip ir Unity, yra rekomenduojama mažiau patyrusiems žaidimų kūrėjams, lyginant su Unreal Engine, kuris reikalauja daugiau patirties norint pasiekti gerų rezultatų [SP23].

Be to, Godot integracija su sustiprininto mokymosi agentais turi potencialo kurti aplinką, palankią dirbtinio intelekto agentų mokymui ir sudėtingų sprendimų priėmimo procesų tyrimui. Sujungiant žaidimų kūrimo ir dirbtinio intelekto technologijas, Godot tampa universaliu įrankiu tyrėjams ir kūrėjams, besidomintiems interaktyvaus turinio kūrimu ir dirbtinio intelekto ribų tyrinėjimu [BDS+21].

Vis dėlto, Godot buvo išleista 2014 metais, tad lyginant su 2005 išleista Unity ar Unreal Engine pirmąja versija, išleista 1998 metais, yra nemažas laiko skirtumas. Dėl šios priežasties, Godot, nors yra sparčiai auganti ir didelę žmonių bazę turinti platforma, nėra tokia didelė bei neturi tokios plačios dokumentacijos.

1.4. Platformų palyginimas

Taigi, yra trys populiariausios šių laikų žaidimo kūrimų platformos. Lentelėje Nr. 1 pateikiami palyginamieji rezultatai.

Platforma	Sukūrimo metai	Sudėtingumas	Mokymosi išteklių kiekis	Pagrindinė programavimo kalba
Unity	2005	Vidutinis	Didžiausias	C#
Unreal Engine	1998	Sunkus	Didelis	C++
Godot	2014	Vidutinis	Vidutinis	GDScript

Lentelė Nr. 1 Žaidimų kūrimų platformų palyginimas

Lentelėje Nr. 1 yra vaizduojami duomenys, lyginant varikliuko sudėtingumą, kiek informacijos įmanoma rasti apie varikliuką bei kokia programavimo kalba daugiausia vertėtų dirbti naudojantis platforma.

Pagrindiniai kriterijai nustatytam darbui – sukurti paprastą 3D žaidimą su įvairiomis mechanikomis – sudėtingumu ir mokymosi išteklių kiekiu. Kadangi žaidimas kuriamas vieno žmogaus, būtų sunku išnaudoti Unreal Engine varikliuko suteikiamas galimybes, nors pačių mokymosi išteklių yra nemažai. Godot yra tinkamesnis pasirinkimas tokiai užduočiai, tačiau, lyginant su Unity, sudėtingumas abiejų platformų yra itin vienodas, tačiau Unity turi didesnę bendruomenę ir aibę mokymosi šaltinių, padėsiančių įgyvendinti įvairiausias norimas žaidimo taisykles. Taigi galima teigti, jog pradedantiesiems žaidimų kūrėjams, Unity platforma būtų optimaliausias pasirinkimas.

2. Žaidimo kūrimo naudojant Unity varikliuką metodika

Šiame skyriuje apžvelgiama metodinė dalis, kaip yra kuriami žaidimai, naudojant Unity platformą bei kaip galima pasiekti įvade aprašytas žaidimo mechanikas.

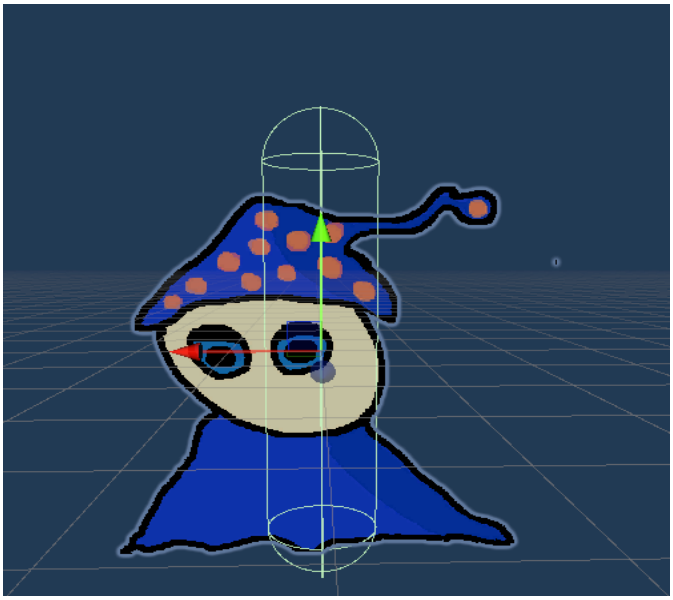
2.1. Žaidimo objektai

Unity platformoje viskas susideda iš žaidimo objektų, kuriuos sukūrus, jiems ar jų vaikams, yra priskiriami komponentai, pavyzdžiui, parašyti kodai, nusakantys tam tikrą logiką, kuria komponentas turėtų vadovautis, ar tokie komponentai, kurie tikrina susidūrimus (angl. collider) ar turi informaciją apie veikėjo poziciją, dydį, rotaciją (angl. transform).

Toliau bus apžvelgiami pagrindiniai žaidimo objektai, kurie sukurtų žaidimui siekiamą funkcionalumą ir įgyvendintų žaidimo idėją.

2.1.1. Žaidėjo objektas

Žaidėjo objektą sudaro transformacijos komponentas, kuris laiko poziciją, rotaciją ir objekto dydį. Taip pat priskiriamas kapsulės susidūrimo komponentas, kuris tikrina, ar žaidėjas nesiliečia su kitais objektais, turinčiais tam tikrą susidūrimo komponentą. Svarbu paminėti, jog Unity leidžia nustatyti apimtį, pagal kurią bus tikrinami susidūrimai nepaisant nuo to, koks yra vaizdo generatorius (pavyzdžiui, tinklo ar žaidėjo objekte naudojamo nuotraukų).



2 pav. Žaidėjas ir kapsulės susidūrimo komponentas (žalias).

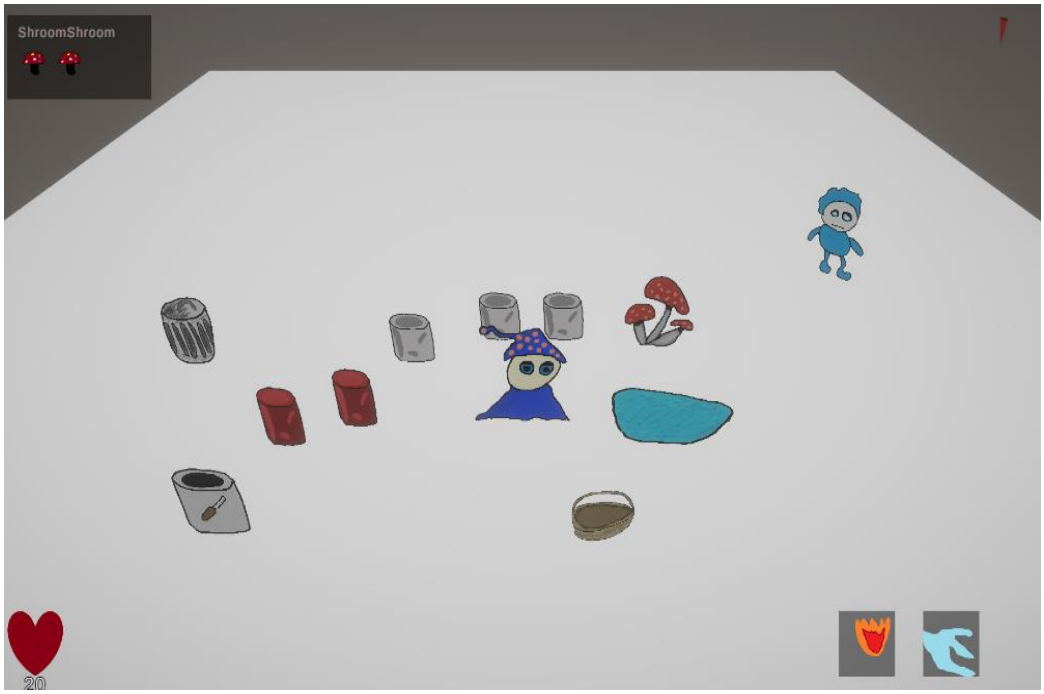
Kaip matoma antrajame paveikslėlyje, nustačius aukščio ir spindulio reikšmes, Unity automatiškai sukuria kapsulę žaliais kontūrais, pagal kurią automatiškai bus apskaičiuojami tam tikri galimi susidūrimai su kitais objektais, pavyzdžiui, monstrais.

Nepaisant transformacijos komponento, vaiko objekte galima suteikti išvaizdą, pridedant nuotraukų atvaizdavimo komponentą. Įkėlus nuotrauką tarp Unity projekto failo ir pasirinkus nuotraukos struktūra kaip 2D, redaktoriuje ją galima tempti į laukelį „Sprite“ ir taip gauti išvaizdą norimam veikėjui.

Visgi pagrindinis komponentas tokiam objektui kaip žaidėjas bus suprogramuotas kodas, leidžiantis manipuliuoti kitais komponentais. Trečioje kursinio darbo skiltyje bus apžvelgiamos įdomiausios kodo dalys apie įgyvendinimą.

2.1.2. Pacientų ir darbo objektai

Viena pagrindinių žaidimo mechanikų – surinkti žemėlapyje išmėtytus resursus į krepšį, sukuriant norimą receptą, kuris sudarys vaistus pacientams. Kairiajame ekrano kampe atvaizduotas receptų sąrašas su pavadinimu bei ingredientais, kuriuos veikėjas turės surinkti, norint pagaminti pageidaujamą produktą.



3 pav. Receptai atvaizduojami viršutiniame kairiame ekrano kampe.

Surinktus objektus, tokius kaip grybas, galima pasidėti ant nuotraukos viduryje matomo pilko stovo ar ant jo uždėti krepšį, tokiu būdu objektus iškart sudedant bendrai. Kai kuriuos objektus, tokius kaip grybas, galima įvairiai susmulkinti ant tam skirto akmens su peilio ženklu. Galiausiai krepšį su jau sudėtais ingredientais galima atiduoti ateinantiems pacientams (nuotraukoje žydras žmogeliukas) ir, jeigu neturime pilnų gyvybių, atiduotas tinkamas receptas grąžins žaidėjui dalį prarastų gyvybių. Žaidime taip pat matomas šiukšliadėžės objektas, į kurį galima nunešti nereikalingus objektus ar neteisingus receptų krepšius. Ant raudono stalo galima kaitinti vandenį ar kitus objektus, kuriuos ateityje bus lengviau pridėti.

2.1.3. Monstrų objektai

Žaidimų kūrime itin svarbu sudominti žaidėjus netikėtumais. Amerikiečių vaizdo žaidimų dizainerio bei autoriaus Jesse Schell knygoje apie žaidimų dizainą yra pabrėžiama, jog svarbu, kad žaidėjai turėtų problemas, kurias jie galėtų spręsti. Kaip gali kilti tokios problemos, kaip gali atsirasti naujų problemų žaidimo eigoje, jog išlaikyti įdomumą [Sch08]. Tam savo žaidime galima kurti įvairius monstrus, kurie skirtųsi pagal lygį, savo veiksmus bei išvaizdą. Pirmą kartą žaidžiantis žmogus stebės, kaip skirtingi monstrai trukdo, o žaidimo eigoje jau žinant jų galias, galės atitinkamai reaguoti susidūrus su jais.

Monstrai šiuo metu žaidime yra du – vienas jų atakuoja žaidėją, o kitas bando pavogti krepšį, tikėdamasis, kad ten bus kokių nors jau įdėtų ingredientų, taip trukdant žaidėjui lengvai pasiekti tikslą. Monstrams galima sukurti bendrą tėvinį kodą, kuris nurodytų visų monstrų bendrą veikimo principą, pavyzdžiui, jeigu žaidėjas panaudoja šaldančius burtus, kiekvieno monstro medžiaga nustatoma iš paprastos į sušąlusią, o kai monstrai yra užšaldyti, jie negali vykdyti jokių veiksmų.

4 pav. Monstras, vagiantis krepšius iš žaidėjo.

Kiekvienas monstras, paveldintis tėvinį kodą, turi dar ir savo atskirą kodą, nusakantį specifinius veiksmus. Pavyzdžiui, monstras, vagiantis krepšį iš žaidėjo, pradės judėti link krepšio tik tada, kai jį aptiks žaidime, tuomet, priėjus iki kažkokio atstumo, jis galės tą krepšį paimti ir pradėti nešti į savo atsiradimo vietą, kur jis krepšį sunaikins. Žaidėjas, kadangi turi galimybes daryti žalą monstrams burtais, įveikęs šį monstrą gali atgauti savo krepšį, nes monstrui mirus krepšys išmetamas ant žemės.

2.1.4. Burtų objektai

Pagrindinis valdomas veikėjas privalo turėti galimybių apsiginti prieš jam trukdančius monstrus. Tam reikia sukurti tam tikrą sistemą, darančią žalą monstrams, kad tai juos veiktų. Burtai turės savo daromą žalą monstrams, sklidimo erdvėje greitį, gali turėti tam tikrą specialų efektą.

2.1.5. Valdymo objektai

Žaidimo eigai kontroliuoti dažniausiai naudojami tušti objektai, turintys tik transformacijos komponentą ir kokį nors savo kodą. Pavyzdžiui, objektas, skirtas tikrinti, ar žaidėjas yra gyvas, ar ne. Jeigu ne, tuomet šis valdymo objektas tiesiog laikinai deaktyvuos žaidėjo objektą ir žaidėjo gyvybių nuotrauką. Praėjus nustatytam laikui, žaidėjas būtų prikeliamas ir nustatomas vėl maksimalus gyvybių kiekis.

Tokiu objektu taip pat dažnai yra kuriamas koks nors pagrindinis valdymo objektas, kuris tikrins, kokioje žaidimo stadijoje yra pagrindinis herojus. Šio žaidimo atveju, bus implementuota, jog būtų keturios stadijos: laukimo, atgalinio skaičiavimo iki žaidimo pradžios, lygio žaidimo ir lygio pabaigos. Laukimo stadijoje galima atskleisti kokią nors pirminę žinutę žaidėjui, paaiškinančią žaidimo taisykles, o jam su jomis susipažinus, paleisti skaičiavimą ir pradėti žaidimą. Žaidimui pasibaigus, šis valdymo objektas siučia įvykį kitiems objektams, pranešant apie pabaigą ir yra rodomas žaidimo rezultato vaizdas, mūsų atveju – kiek pacientų buvo išgydyta ir pasirinkimai žaisti kitus lygius – grįžti į meniu ar išeiti iš žaidimo.

2.2. Unity dirbtinio intelekto bibliotekos analizė

Taigi Unity platformoje žaidimą sudaro daugybė tokių žaidimo objektų, kurie kaip nors sąveikauja tarpusavyje. Tačiau vertėtų apžvelgti, kaip tokie monstrai ar paprasti žaidėjo nevaldomi veikėjai randa kelią iki jiems reikiamų vietų. Tam Unity naudoja “NavMesh” klasę, kuri atlieka erdvines užklausas, tokias kaip kelio ieškojimas bei vaikščiojimas [UT24a].

2.2.1. NavMesh Agent

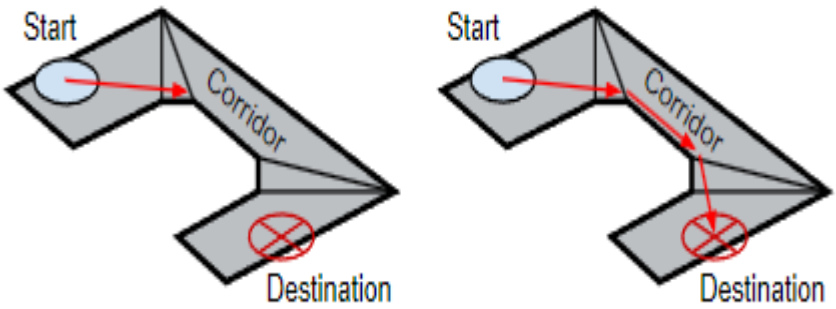
Žaidimo objektams, kurių nevaldys žaidėjas, tačiau jie ir nebus statiški, bus naudojami Unity „NavMesh“ agentai. Jie yra reprezentuojami kaip cilindrai, turintys tam tikrą aukštį ir spindulį. Ši cilindro forma skirta rasti susidūrimus su kitais agentais ar kliūtimis [UT23a]. Unity varikliuke galima Nav Mesh Agent komponentui nustatyti skirtingus kintamuosius, tokius kaip greitis ar automatinis stabdymas, jog agentas prieš kažkokią kliūtį pristabdytų, kokiu spinduliu jis vengtų kliūčių ir daugiau.

2.2.2. NavMesh Obstacles

Šiam komponentui svarbiausias parametras yra tai, ar objektas yra statinis, nejudantis, ar objektas yra judantis. Kai mes pasirenkame, jog objektas bus statinis, tuomet agentas galės judėti iš anksto planuodamas kliūties apėjimą. Tačiau jeigu kliūtis yra judanti, tada paliekama šį rėžimą išjungtą ir agentas bandys tiesiog sukti, jog nekiltų susidūrimų. Šis pastarasis veikimo principas yra labai paprastas ir turi nedidelį spindulį, tad jei žaidime yra daug tokių objektų, agentui gali būti sunkiau rasti savo tikslą. Ar objektas juda, Unity nustato tai pagal mūsų nustatytą parametrą komponente, kuomet objektas pajuda, tada keičiasi ir vietą, kurią reikės apeidinėti agentui. Šio skaičiavimo rezultatas įvyks kitame kadre [UT23b].

2.2.3. Navigacijos sistemos principas Unity

Turint tam tikrą veikėją ir tikslą yra būtina nustatyti ribas, kuriose galima stovėti ir vaikščioti. Jau minėta, jog agentai yra reprezentuojami kaip cilindrai. Unity sistemoje, vaikščiojimo zona yra automatiškai sukuriama pagal geometriją scenoje, testuojant vietas, kur agentas-cilindras galėtų stovėti. Tuomet šios vietos yra sujungiamos į tam tikrą paviršių. Gautas paviršius agentui yra navigacijos tinklas. Šis tinklas saugo paviršių kaip aibę daugiakampių. Kadangi yra tinklas, veikėjas gali ieškoti šiais daugiakampiais kelio nuo pradžios iki pabaigos, ir Unity kaip savo kelio paieškos algoritmą naudoja A*. Unity dokumentacijoje kelias nuo pradinio daugiakampio iki tikslo yra vadinamas koridoriumi.



6 pav. Unity dokumentacijos koridoriaus iliustracija. Agentas juda daugiakampiais nuo starto iki finišo.

Agentas stengsis neatsimušti į jokias kliūtis, apskaičiuojant tinkamą pagreitį ir apsaugant nuo galimų atsimušimų su kitais agentais ar objektais ateityje. Kai jau yra apskaičiuoti reikiami posūkiai ir pagreitis, tuomet agentas pradeda judėti. Navigacija veikia globaliai ir lokaliai. Globali navigacija suranda kelią – koridorius, o lokaliai yra apskaičiuojama agento judėjimo kryptis, greitis. [UT23c]

3. Žaidimo įgyvendinimas

Unity platformoje viskas susideda iš žaidimo objektų, kuriuos sukuriama ir jiems ar jų vaikams, priskiriami kokie nors komponentai, tokie kaip parašyti kodai, nusakantys tam tikrą logiką, pagal kurią komponentas turėtų vadovautis, ar tokie komponentai, kurie tikrina susidūrimus (angl. collider), turi informaciją apie veikėjo poziciją, dydį, rotaciją (angl. transform).

Kursinio projekto darbe yra numatoma detaliau aprašyti žaidimo įgyvendinimą, patobulinti žaidimą sukuriant sistemą, leidžiančią žaisti žaidimą keliems žaidėjams prie vieno kompiuterio ar susijungiant internetu.

Žemiau aprašyti pagrindiniai žaidimo objektai, kurie sukuria žaidimui siekiamą funkcionalumą ir įgyvendina žaidimo idėją.

3.1. Žaidėjo objekto kodas

Naudojantis Unity redaktoriumi, patogu objekto kode rašyti prieš nusakant, ar laukas yra privatus, ar viešas, priežodį [SerializeField], jeigu kiltų noras keisti reikšmę bežaidžiant žaidimą. Tai yra itin patogu testuojant tokias reikšmes, kaip žaidėjo ėjimo greitis.

Žaidėjo kodo pačioje pradžioje prenumeruojame pagalbinio objekto, sukurto apdoroti įvedimą siunčiamus įvykius.

```
   private void Start(){
        currentHealth = playerMaxHealth;
        gameInput.TakeActionEvent += GameInput_TakeAction;
        gameInput.AlternateActionEvent += GameInput_AlternateActionEvent;
        gameInput.ActivateFireballEvent += GameInput_FireballPerformed;
        gameInput.ActivateIceWaveEvent += GameInput_IceWavePerformed;

    }
```

Kuomet žaidėjas paspaus mygtuką, skirtą iššauti ugnies burtus, pagalbinėje GameInput klasėje bus išsiunčiamas įvykis, kurį gaus žaidėjo kodas, nes buvo įvykdyta prenumerata.

```
    private void
Fireball_performed(UnityEngine.InputSystem.InputAction.CallbackContext obj){
        ActivateFireballEvent?.Invoke(this, EventArgs.Empty);
    }
```

Tuomet žaidėjo kode esanti funkcija GameInput_FireballPerformed bus įvykdoma.

```
  private void GameInput_FireballPerformed(object sender, System.EventArgs e){
        if (fireballCooldownTimer <= 0)
        {
            if(!HasMedObject()){
                    Vector3 targetPosition = transform.position +
transform.forward*5;
                    Fireball newFireball = Instantiate(fireballPrefab,
holdingPoint.position, Quaternion.identity);
                    newFireball.Activate(targetPosition);
                    fireballCooldownTimer = newFireball.abilityData.cooldown;
            }
        }
    }
```

Pirmiausiai yra patikrinama, ar žaidėjas jau gali naudoti tam tikrus burtus. Tuomet tikrinama, ar žaidėjas neturi jokio objekto, tokio kaip krepšys ar ingredientas rankose. Jeigu šios sąlygos tenkinamos, yra apskaičiuojama pozicija (čia – targetPosition), į kurią turėtų skrieti kerai. Yra sukuriamas kerų objektas, jis yra aktyvuojamas ir paleidžiamas pagal apskaičiuotą kryptį, suteikiant specifinį greitį. Kerų žala ir greitis bus apžvelgiama 3.3. skyriuje.

3.2. Atakuojančio monstro kodas

Metodinėje dalyje buvo nuodugniau nagrinėjama Unity navigacijos sistema. Taip yra todėl, kad ją suprantant yra lengva nustatyti monstrų veikimo principus.

```
    private new void Start(){
        base.Start();
        agent = GetComponent<NavMeshAgent>();
    }
```

Startavus žaidimą, inicializuojamas monstro agento komponentas. Funkcija, jog monstras judėtų link žaidėjo, atrodys labai paprastai.

```
    private void MoveTowardsPlayer(){
        agent.speed = monsterData.speed;
        player = FindObjectOfType<Player>();

        if(player != null && agent.enabled){
```

```
            agent.SetDestination(player.transform.position);
        }
    }
```

Agentui yra priskiriamas toks greitis, koks yra jo objekte priskirtam ScriptableObject, apie kurių veikimo principą nagrinėjama 3.3 skyrelyje. Tuomet surandamas žaidėjo objektas – jeigu jis yra scenoje, tuomet su UnityEngine.AI biblioteka naudojama SetDestination funkcija, kuri, naudojant A* algoritmą, eis links žaidėjo pozicijos.

```
public override void Act(){
        Player player = FindObjectOfType<Player>();
        if(player != null)
        {
            float distanceToPlayer = Vector3.Distance(transform.position,
player.transform.position);
            if(distanceToPlayer <= monsterData.range)
            {
                player.TakeDamage(monsterData.damage);
            }
        }
    }
```

Aptikus žaidėją, yra skaičiuojama su Vector3.Distance funkcija, kuri grąžina atstumą tarp dviejų pozicijų, ar monstras gali sąveikauti su žaidėju. Jeigu atstumas neviršija nustatyto monstro atstumo sąveikai, tada žaidėjui yra nuimamos gyvybės.

```
    protected override void Update(){
        base.Update();

        if(!IsFrozen()){
            attackTimer += Time.deltaTime;
            if (attackTimer >= attackCooldown) {
                Act();
                attackTimer = 0f;
            }
            MoveTowardsPlayer();
        }

    }
```

Tam, kad monstras visą laiką vykdytų šiuos veiksmus, naudojama Unity funkcija Update, kuri paleidžia šį kodą su kiekvienu einančiu kadru. Patikrinama, ar monstras nėra užšalęs – jei taip yra ir praėjo nustatytas laikas, kas kiek monstras gali atakuoti, tuomet jis bando atakuoti žaidėją. Taip pat jei monstras nėra užšalęs, jis nuolatos judės žaidėjo link.

3.3. Burtų ir monstrų reikšmių veikimo principas

Sukuriamas tėvinis skriptas burtams, kuriam bus naudojama Unity Scriptable Objects sistema. Ši sistema buvo taip pat naudojama ir monstrų duomenims nusakyti. Galima sukurti klasę, kuri praplėstų Scriptable Object klasę. Nustačius priežodį [CreateAssetMenu()], redaktoriuje galima kurti failus, kurie yra ScriptableObject tipo.

```
5   [CreateAssetMenu()]
6   public class AbilitySO : ScriptableObject{
7       public GameObject prefab;
8       public int damage;
9       public float cooldown;
10      public float range;
11      public float speed;
12      //animations, vfx
13  }
```

5 pav. Programuojama burtų informacija Scriptable Object sistema.

Unity leidžia susieti kokius nors išsaugotus žaidimo objektus, kad būtų galima turėtu burtų išvaizdą ar kitus efektus, nustatyti žalą, trukmę, kas kiek laiko galima leisti specifinį burtą, ilgį, kiek laiko burtas gali nukeliauti bei greitį, ar pridėti kokių nors dar mums reikalingų duomenų naudojant Unity Scriptable Object. Taip ir galima pridėti monstrams skirtingą kiekį gyvybių, jų žalą, greitį ir kitas savybes iš Unity redaktoriaus, sukuriant lengvą vartotojo sąsają žaidimų kūrėjui.

Burtai šiuo metu žaidime yra suprogramuoti du – vienas yra ugnies, kuris daro didesnę žalą ir skrenda greičiau, o kitas yra šalčio, kuris turi mažesnę žalą, bet užšaldo monstrus ir panaikina jų galimybę vykdyti kažkokius veiksmus nustatytam laikui. Kursinio projekto darbe bus apžvelgiamos jų sąveikos su monstrais.

REZULTATAI IR IŠVADOS

Išvados:

- Unity platforma turi panašų sudėtingumą su Godot žaidimų kūrimo varikliuku, bet turi daugiau mokymosi šaltinių suteikiant jai pranašumą pradedančiųjų žaidimų kūrėju atžvilgiu.
- Unity žaidimų kūrimo varikliukas yra lengvesnis naudoti negu Unreal Engine, todėl pradedantieji žaidimų kūrėjai turėtų rinktis Unity.
- Unity AI bibliotekos suteikiama navigacinė sistema leidžia lengvai implementuoti nevaldomų veikėjų kelio paieškos algoritmą.

Rezultatai:

- Sukurta žaidėjo sistema, leidžianti žaidėjams valdyti pagrindinį veikėją, naudotis įvairiais kerais prieš monstrus, rinkti įvairius ingredientus, esančius žemėlapyje.
- Sukurta monstrų sistema, leidžianti lengvai redaktoriuje keisti monstrų gyvybes, daromą žalą ir kitas reikšmes naudojant Unity ScriptableObject.
- Implementuotas atakuojantis monstras, kuris, jeigu nėra paveiktas šaldymo kerų, eis link žaidėjo valdomo veikėjo ir jį puls.

ŠALTINIAI

[BDS+21] Edward Beeching, Jilles Dibangoye, Simonin Olivier & Christian Wolf. (2021). Godot Reinforcement Learning Agents.

[BPS+20] Ekaterina Bogdanova, Pavel Pugachev, Ivan Saldikov et al.. (2020). Visualization of neutron characteristics distribution of debris particles. Scientific Visualization.

[ES17] Christopoulou Eleftheria, Xinogalos Stelios. (2017). Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. International Journal of Serious Games.

[Law20] H. A. J. Al Lawati, The Path of UNITY or the Path of UNREAL? A Comparative Study on Suitability for Game Development, Journal of Student Research, 2020.

[NBH+20] Paul Ryan Nesbit, Adam Boulding, Christopher Hugenholtz et al., Visualization and Sharing of 3D Digital Outcrop Models to Promote Open Science, Article in GSA Today

[RM24] Ranaweera, M.; Mahmoud, Q.H. Deep Reinforcement Learning with Godot Game Engine. Electronics 2024. https://www.mdpi.com/2079-9292/13/5/985

[Sch08] J.Schell**.** The Art of Game Design: A Book of Lenses. 2008, p. 36-38.

[SMJ22] Jonas Vedsted Sørensen, Zheng Ma, and Bo Nørregaard Jørgensen. Potentials of game engines for wind power digital twin development: an investigation of the Unreal Engine, , SDU Center for Energy Informatics, 2022.

[SP23] Sobota Branislav, Pietrikova Emília. (2023). The Role of Game Engines in Game Development and Teaching. https://www.researchgate.net/publication/373282820_The_Role_of_Game_Engines_in_Game_Development_and_Teaching

[UT23a] Unity Technologies. Unity Documentation. *About Agents.* [Online] 10 19, 2023. https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/AboutAgents.html.

[UT23b] Unity Technologies. Unity Documentation. *About Obstacles.* [Online] 10 19, 2023. https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/AboutObstacles.html.

[UT23c] Unity Technologies. Unity Documentation. *NavInnerWorkings.* [Online] 2023. https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/NavInnerWorkings.html.

[UT24a]Unity Technologies. Unity Documentation. *NavMesh.* [Online] 06 06, 2024. https://docs.unity3d.com/ScriptReference/AI.NavMesh.html.