VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Kursinis projektas

Kompiuterinių žaidimų kūrimas naudojant Unity platformą

(Development of computer games using the Unity platform)

Atliko: 4 kurso 2 grupės studentas
Tautvaldas Gumbis

Darbo vadovas:
dr. Igor Katin

Vilnius
2025

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).

Kursiniame darbe buvo kuriamas žaidimas 3D erdvėje, veikėjų vizualus perteikiant 2D formatu. Žaidimo idėja šiam kursinio darbo projektui išlieka ta pati – keliaujantis burtininkas - daktaras, gydantis įprastus gyventojus renkant specifinius ingredientus pateiktame žemėlapyje. Žaidėjo valdomam pagrindiniam veikėjui trukdo įvairūs monstrai, atimdami krepšius su ingredientais ar puldami patį herojų.

Jau kursiniame darbe šiek tiek apžvelgėme, kaip galima kurti atakas, galinčias apsaugoti jį nuo tam tikrų monstrų. Šiame darbe bus apžvelgiama, kaip sukurti naujų monstrų tipų, atakų, išsaugoti žaidimo būseną, kaip padaryti žaidimo nustatymų meniu naudojant įvairius Unity įrankius bei komponentus.

Ties tuo žaidimas nesibaigs, baigiamajame bakalauro darbe 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 projekto tikslas: Naudojant Unity platformą tobulinti žaidimą pridedant naujų monstrų, atakų, nustatymų, žaidimo būsenos išsaugojimą, palyginti Unity ypatumus su kitų žaidimų kūrimo platformų galimybėmis.

Kursinio darbo projekto uždaviniai:

- Išanalizuoti Unity platformos pranašumus pradedančiųjų žaidimų kūrėjų atžvilgiu;
- Sukurti naujų žaidėjo valdomo veikėjo ir kitų žaidimo objektų sąveikas (naujas monstras, ataka);
- Sukurti galimybes pasikeisti nustatymus;
- Sukurti žaidimo būsenos išsaugojimą, jog nebūtų prarandamas žaidimo progresas;

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.

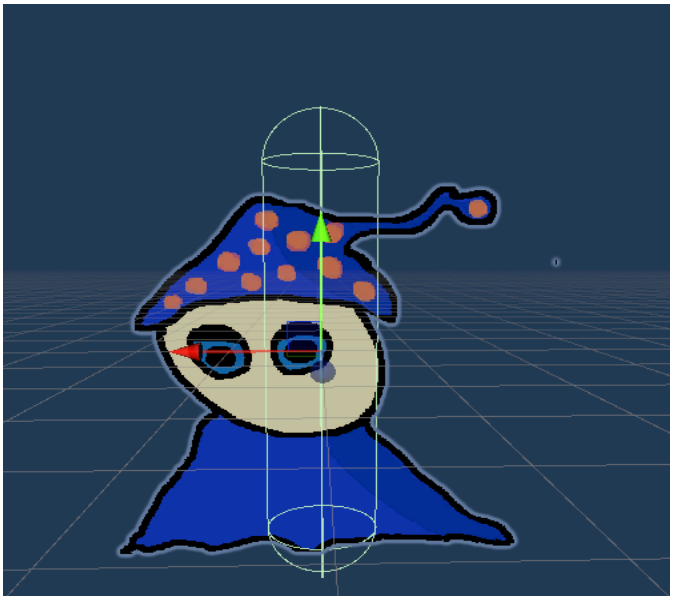
2.1. Žaidimo objektai

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).

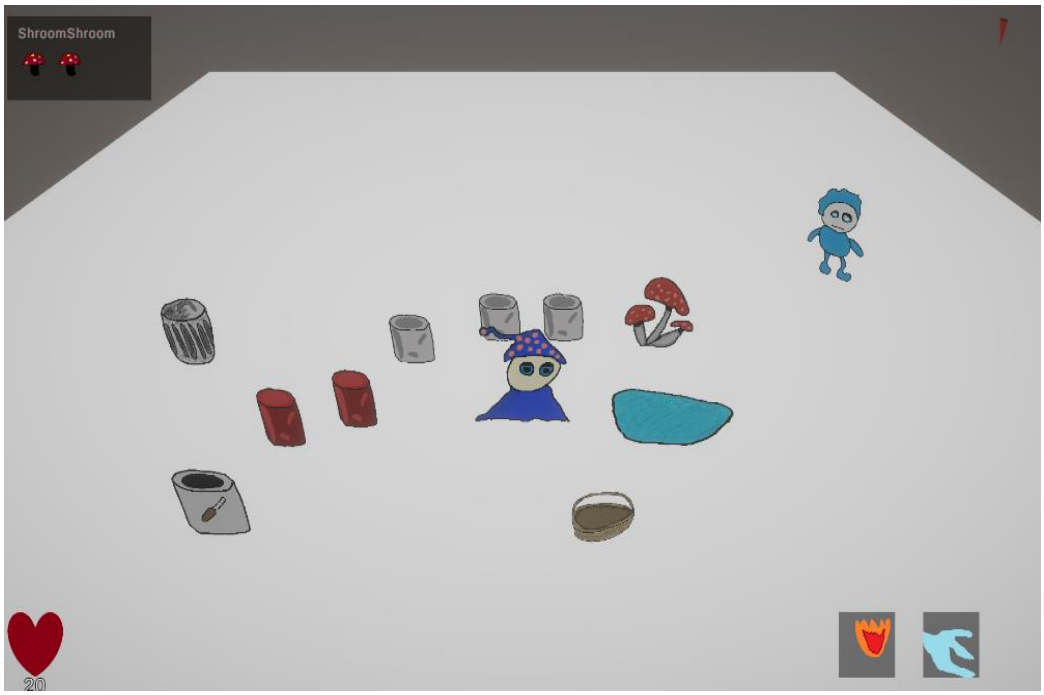
Kaip matoma antrajame paveikslėlyje, nustačius aukščio ir spindulio reikšmes, Unity automatiškai sukuria kapsulę žaliais kontūrais, pagal kurią automatiškai bus apskaičiuojami tam tikri galimi susidūrimai su kitais objektais, pavyzdžiui, monstrais.

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 trys – vienas jų atakuoja žaidėją, kitas bando pavogti krepšį, tikėdamasis, kad ten bus kokių nors jau įdėtų ingredientų, taip trukdant žaidėjui lengvai pasiekti tikslą. Taip pat yra sukurtas monstras, prieš kurį ugnies ataka neveikia. 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, šiuo atveju – kiek pacientų buvo išgydyta ir pasirinkimai žaisti kitus lygius – grįžti į meniu ar išeiti iš žaidimo.

2.1.6. Žaidimo būsenos objektas

Žaidimo būsenai išsaugoti, sukurti naują ar užkrauti jau seniau sukurtą taip pat reikalingas objektas. Tiesa, jis bus panašus į jau aprašytus valdymo objektus. Pagrindinis skirtumas yra tas, jog norint, kad tinkamai veiktų žaidimo būsenos saugojimas, jis turi egzistuoti ir tuomet, kuomet yra keičiamos scenos.

Norint, jog objektas nebūtų sunaikintas kai yra keičiamos scenos, reikalinga naudoti DontDestroyOnLoad funkciją skripte. Taigi šis objektas bus tiesiog paprastas objektas turintis transformacijos komponentą bei skriptą, kurio įgyvendinimas plačiau bus apžvelgtas „Žaidimo įgyvendinimo“ dalyje.

2.1.7. Vartotojo sąsajos objektas

Unity variklyje kaip žaidimo objektą galima sukurti Canvas, tai yra objektas, kuris perteiks vartotojui vartotojo sąsajos elementus scenoje. Į jo vidų galima dėti visus su vartotojo sąsaja susijusius elementus.

5 pav. Vartotojo sąsajai priskiriami žaidimo elementai

Projekte atvaizduojami receptai, žaidėjo burtai, gyvybės, lygio laikrodis, meniu, kuris pasirodo sustabdžius žaidimą, nustatymų meniu ir visa tai yra pasiekiama kuriant objektus Canvas viduje. Canvas vidus kaip komponentus turės vietoj standartinio transformacijos komponento RectTransform komponentą, pagal kurį galima keisti objekto plotį, aukštį, sukimosi tašką ir dar kitus

kintamuosius. Taip pat labai svarbi dalis vartotojo sąsajai yra sukurti tinkamas klases, kurių tam tikri metodai būtų vykdomi paspaudus mygtuką. Tam galima pridėti kaip komponentus mygtukus. Kiti dažnai naudojami vartotojo sąsajos komponentai yra nuotraukos ar tekstai.

3. Žaidimo įgyvendinimas

Unity platformoje viskas susideda iš žaidimo objektų, kurie yra sukuriami 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 bus patobulinamas žaidimas sukuriant galimybę žaidėjui pakeisti nustatymus, ką kuris mygtukas daro. Taip pat žaidimas galės gebėti išsaugoti lygį, kuriame žaidėjas baigė žaisti, pridedama nauja galimybė atakuoti monstrus.

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

3.1. Žaidimo eiga

Nustatymai pasiekiami jau esant žaidime. Dabar žaidėjo valdomas veikėjas galės judėti tik po nustatyto laikmačio. Prasidedant žaidimui, žaidimas tampa pradėto lygio fazėje, kuomet žaidėjas gali vykdyti savas užduotis.

```
    private void Update(){

        switch(gameState){

            case GameState.CountdownTime:
                countdown -= Time.deltaTime;
                if(countdown <= 0){
                    gameState = GameState.LevelStarted;
                    GameStateChangedEvent?.Invoke(this, EventArgs.Empty);
                }
                break;
            case GameState.LevelStarted:
                levelDuration -= Time.deltaTime;
                timePassed += Time.deltaTime;
                if(levelDuration < 0f){
                    gameState = GameState.LevelEnded;
                    GameStateChangedEvent?.Invoke(this, EventArgs.Empty);
```

```
            }
            break;
        case GameState.LevelEnded:
            Time.timeScale = 0;
            break;

    }
}
```

Čia svarbu yra tai, jog levelDuration yra lengvai keičiamas laukas (naudojantis [SerializeField], kurio veikimas jau buvo apžvelgtas kursiniame darbe), tad kiekvienam lygiui lengvai galima pakeisti jo trukmę priklausant nuo lygio keliamų uždavinių.

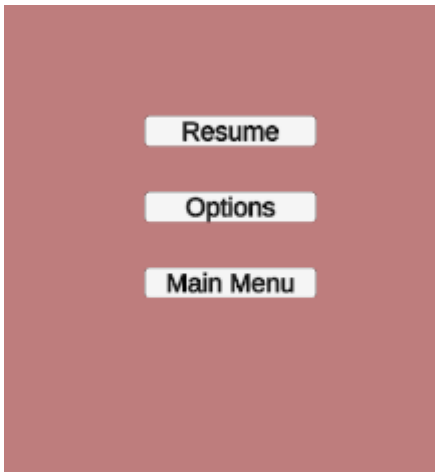
3.2. Žaidimo stabdymas bei nustatymai

Unity leidžia lengvai stabdyti žaidimą pasinaudojant Unity varikliuko klase „Time“. Su Time.timeScale funkcija galima nustatyti skalę, kokia turėtų būti laiko tėkmė žaidime. Jeigu skalė bus lygi vienetui, tuomet žaidime laikas eis kaip ir realiame gyvenime. Jeigu skalė būtų lygi 0.5, tuomet laikas žaidime eitu dvigubai lėčiau negu realiame gyvenime. Tad tokiu būdu galima keisti skalę tarp vieneto ir nulio norint lengvai sustabdyti žaidimą.

```
private void GameInput_PauseGameEvent(object sender, System.EventArgs e){
    PauseGame();
}
public void PauseGame(){
    isPaused = !isPaused;

    if (isPaused){
        Time.timeScale = 0;
        GameManagerPausedEvent?.Invoke(this, EventArgs.Empty);
    } else {
        Time.timeScale = 1;
        GameManagerUnpausedEvent?.Invoke(this, EventArgs.Empty);
    }
}
```

Sustabdžius žaidimą, jog žaidėjas suprastų, kad žaidimas buvo sustabdytas, sukuriama vartotojo sąsaja pridedami lentelę su keliais pasirinkimais esančiais pauzės meniu.

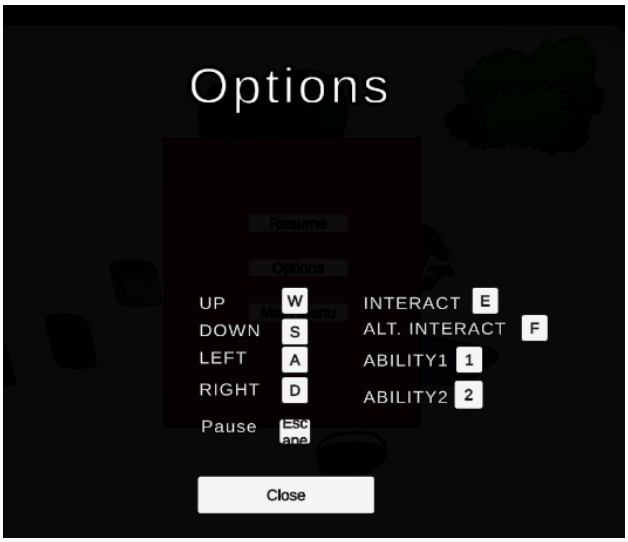


6 pav. Pauzės meniu

Žaidėjas gali pasirinkti tęsti žaidimą (tuomet yra vėl kviečiama PauseGame funkcija), galima paspausti Options, kuris atidarytų nustatymų meniu bei MainMenu, kuris grąžintų žaidėją į pradinį žaidimo ekraną.

```
private void Awake(){
        resumeButton.onClick.AddListener( () => {
            GameManager.Instance.PauseGame();
        });
        optionsButton.onClick.AddListener( () =>{
            OptionsUI.Instance.Show();
        });
        mainMenuButton.onClick.AddListener( () => {
            Loader.Load(Loader.Scene.MainMenuScene);
        });
    }
```

Pasirinkus nustatymų meniu iššoka nauja lentelė su pasirinkimais, leidžiančiais pakeisti žaidėjo valdymui naudojamus mygtukus:



7 pav. Nustatymų meniu

Paspaudus ant tam tikro kvadratėlio (pavyzdžiui W) galima pakeisti jį į bet kokį kitą mygtuką.

```
public void RebindKeybind(Keybinds keybinds, Action onActionRebound){

    InputAction inputAction;
    int bindingIndex;

    switch (keybinds){
        default:
        case Keybinds.Move_Up:
            inputAction = playerInputSystem.Player.Movement;
            bindingIndex = 1;
            break;
        case Keybinds.Move_Down:
            inputAction = playerInputSystem.Player.Movement;
            bindingIndex = 2;
            break;
        case Keybinds.Move_Left:
            inputAction = playerInputSystem.Player.Movement;
            bindingIndex = 3;
            break;
        case Keybinds.Move_Right:
            inputAction = playerInputSystem.Player.Movement;
            bindingIndex = 4;
            break;
        case Keybinds.Interact:
            inputAction = playerInputSystem.Player.Take;
```

```
                bindingIndex = 0;
                break;
            case Keybinds.AltInteract:
                inputAction = playerInputSystem.Player.AlternateInteraction;
                bindingIndex = 0;
                break;
            case Keybinds.Pause:
                inputAction = playerInputSystem.Player.Pause;
                bindingIndex = 0;
                break;
            case Keybinds.Ability1:
                inputAction = playerInputSystem.Player.Fireball;
                bindingIndex = 0;
                break;
            case Keybinds.Ability2:
                inputAction = playerInputSystem.Player.IceWave;
                bindingIndex = 0;
                break;
        }

        inputAction.PerformInteractiveRebinding(bindingIndex).OnComplete(callback
=> {
            callback.Dispose();
            onActionRebound();

            PlayerPrefs.SetString(PLAYER_BINDINGS,
playerInputSystem.SaveBindingOverridesAsJson());
            PlayerPrefs.Save();
        }).Start();
    }
```

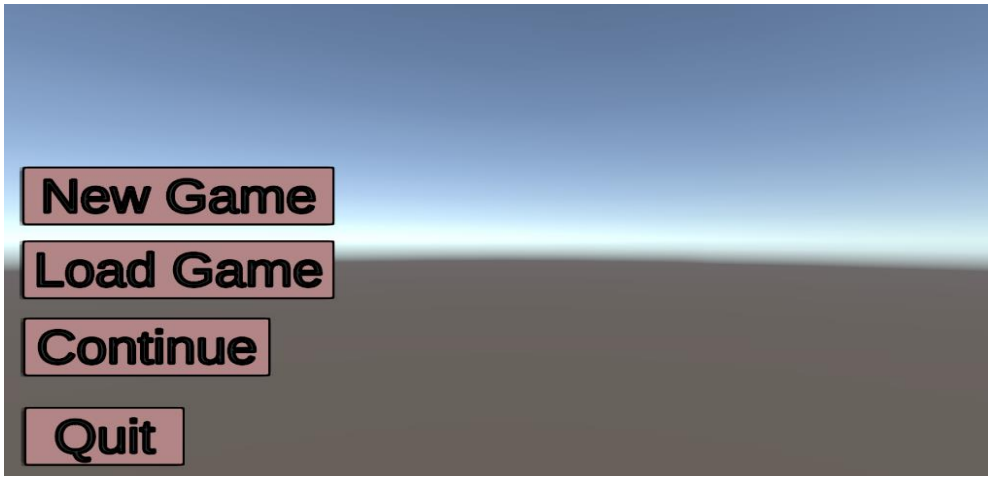
Priklausant nuo Keybind reikšmės, ar tai būtų pauzės mygtukas, ar judėjimo, priskiriama InputAction esanti PlayerInputSystem reikšmė ir taip pat priskiriama nauja indekso reikšmė. Pavyzdžiui, vaikščiojimas PlayerInputSystem žaidimo projekte yra traktuojamas kaip blokelis, į kurį įeiną vaikščiojimo kryptys, todėl indeksai šiom kryptimis prasideda nuo 1 iki 4.

Toliau pasinaudojant Unity InputAction galima naudoti PerformInteractiveRebinding(bindingIndex), tuomet žaidimas lauks vartotojo įvesto naujo klavišo. Jį įvedus, yra atlaisvinami naudojami resursai, ir naudojant onActionRebound() nurodoma, jog pavyko pakeisti mygtuką.

PlayerPrefs klasė Unity saugo informaciją žaidėjo registre. Pasinaudojus ja, galima išsaugoti nustatymuose pakeistus mygtukus, ir tuomet žaidėjui išėjus iš žaidimo ir grįžus, nereikės keisti mygtukų per naujo.

3.3. Loader klasė

Žaidimai dažnai naudoja daug scenų vaizduoti skirtingiems lygiams ar pradinio meniu ekranui.



8 pav. Pradinis žaidimo ekranas

Tam yra reikalinga klasė, kuri gebėtų perjungti žaidimo scenas. Jai suteikiamas pavadinimas Loader.

```
public static class Loader{
    public enum Scene{
        MainMenuScene,
        Level1_1, Level1_2, Level1_3, Level1_4, Level1_5,
        Level2_1, Level2_2, Level2_3, Level2_4, Level2_5,
        Level3_1, Level3_2, Level3_3, Level3_4, Level3_5,
        Level4_1, Level4_2, Level4_3, Level4_4, Level4_5,
        LoadingScene
    }
    private static Scene targetScene;
    private static AsyncOperation asyncOperation;

    public static void Load(Scene targetScene){
        Loader.targetScene = targetScene;
```

```
        SceneManager.LoadScene(Scene.LoadingScene.ToString());
    }
    public static void LoadAsync(Scene targetScene){
        Loader.targetScene = targetScene;

        asyncOperation =
SceneManager.LoadSceneAsync(Scene.LoadingScene.ToString());
    }
    public static void LoaderCallback(){
        if (asyncOperation != null && asyncOperation.isDone){
            asyncOperation = SceneManager.LoadSceneAsync(targetScene.ToString());
        }
        else{
            SceneManager.LoadScene(targetScene.ToString());
        }
    }
}
```

Pirmiausiai aprašomos visos scenos, kurios yra sukurtos Unity projekte. LoadAsync funkcija yra naudojama tuomet, jeigu norima keisti sceną nesunaikinant tam tikro objekto. Kai kviečiama naują sceną, pavyzdžiui Level1_1, tuomet įjungiame LoadingScene, tam, jog žaidėjas suprastų, kad yra kraunamas žaidimo turinys. Pagalbinė funkcija LoadingScene scenoje esančiame žaidimo objekte (kviečianti Loader.LoaderCallback();) tikrins, ar norima scena jau yra užsikrovusi. Tuomet pilnai užkrautas Level1_1 yra parodomas žaidėjui. Taip galima pereiti iš MainMenu ekrano į žaidimą, bei rodyti ekrane žinutę, jog scena yra kraunama.

3.4. Žaidimo išsaugojimas

Žaidimo išsaugojimui svarbu yra žinoti, kokią informaciją apie žaidimą reikalinga saugoti.

```
public class GameData
{
    public long lastUpdated;
    public string currentLevel;
    public int deliveriesMade;

    public GameData(){
        this.deliveriesMade = 0;
        this.currentLevel = "Level1_1";
    }
    public string GetCurrentLevel(){
```

```
        return currentLevel;
    }
    public string GetLastPlayedDate()
    {
        DateTime lastPlayedDate = DateTime.FromBinary(lastUpdated);
        string lastPlayedDateString = lastPlayedDate.ToString("yyyy-MM-dd HH:mm");
        return lastPlayedDateString;
    }
}
```

Tam apibrėžiama GameData klasę, kurioje galima grąžinti esamą lygį, kiek kartų buvo įvykdyta tam tikra užduotis (šiuo atveju padaryti pristatymai) bei tokią informaciją, kuomet žaidimas buvo žaistas paskutinį kartą.

Sukuriama klasė DataManager, kurioje bus metodai, leidžiantys sukurti naujo žaidimo informaciją, išsaugoti žaidimą bei užkrauti žaidimą.

```
public void NewGame(){
        this.gameData = new GameData();
    }
```

Naujam žaidimui sukurti pirmiausiai yra sukuriamas naujas GameData objektas, kurį galima redaguoti. Jeigu nebuvo pradėti žaidimai, kurie buvo vėliau išsaugoti ar meniu yra pasirenkamas naujas žaidimas, būtent ši funkcija yra kviečiama. Žaidimo išsaugojimo funkcija veikia sudėtingiau.

```
    public void SaveGame(){
        if(this.gameData == null){
            Debug.LogWarning("Klaida: Nebuvo rasti duomenys, kuriuos galėtume
išsaugoti");
            return;
        }
        gameData.lastUpdated = System.DateTime.Now.ToBinary(); // Išsaugome laiką,
kada buvo išsaugotas žaidimas, tam, kad galėtume implementuoti “Continue” iš meniu.

        fileDataHandler.Save(gameData, selectedProfileID);
    }
```

Sukuriama papildoma klasė FileDataHandler, kuri būtų atsakinga už darbą su failais tam, kad palaikytų tinkamus objektinio programavimo standartus. Metodui Save yra paduodama informacija, kurią saugosime bei profilio id.

```
public void Save(GameData data, string profileId){
```

```
        if(profileId == null){
            return;
        }
        string fullPath = Path.Combine(dataDirPath, profileId, dataFileName);
        try{
            Directory.CreateDirectory(Path.GetDirectoryName(fullPath));

            string dataToStore = JsonUtility.ToJson(data, true);

            if(useEncryption){
                dataToStore = EncryptDecrypt(dataToStore);
            }
            using (FileStream stream = new FileStream(fullPath, FileMode.Create)){
                using(StreamWriter writer = new StreamWriter(stream)){
                    writer.Write(dataToStore);
                }
            }
        }catch (Exception e){
            UnityEngine.Debug.LogError("Klaida bandant išsaugoti žaidimo
informaciją į failą esantį " + fullPath + "\n" + e);
        }
    }
```

Pirmiausiai patikrinama, ar profileID nėra lygus nuliui, nes jeigu taip, tuomet yra klaidų su profiliu. Sukuriama string fullPath, tai yra direktorija, kurioje bus saugoma žaidimo informacija. Parametras dataDirPath yra direktorija, profileID – konkretaus žaidimo būsena (jų žaidime yra galimos trys), dataFileName – failo pavadinimas, kuriame bus saugomi duomenis. Sukūrus direktoriją, žaidimo duomenys yra paverčiami į JSON formatą, naudojant Unity klasę JsonUtility. Taip pat sukuriamas dar vienas metodas EncryptDecrypt (paprastas šifravimas su XOR ir slaptažodžiu), ir su [SerializeField] Unity variklyje galima pasirinkti, ar reikia naudoti šifravimą. Šifravimas gali būti reikalingas tam, jog žaidėjai negalėtų lengvai pakeisti faile lygio iš pradinio į paskutinius. Naudojant FileStream ir StreamWriter yra atidaromas rašymo srautas faile ir išsaugoma norimą informacija.

Išsaugojus informaciją dabar galima ją ir užkrauti. Tam sukuriama FileDataHandler funkcija Load, kuriai kaip parametras paduodamas norimos užkrauti žaidimo būsenos ID.

```
  public GameData Load(string profileId){
        if(profileId == null){
            return null;
        }
```

```
        string fullPath = Path.Combine(dataDirPath, profileId, dataFileName);
        GameData loadedData = null;
        if(File.Exists(fullPath)){
            try{
                string dataToLoad = "";
                using (FileStream stream = new FileStream(fullPath,
FileMode.Open)){
                    using(StreamReader reader = new StreamReader(stream)){
                        dataToLoad = reader.ReadToEnd();
                    }
                }
                if(useEncryption){
                    dataToLoad = EncryptDecrypt(dataToLoad);
                }
                loadedData = JsonUtility.FromJson<GameData>(dataToLoad);
            } catch (Exception e){
                UnityEngine.Debug.LogError("Klaida užkraunant informaciją iš failo
" + fullPath + "\n" + e);
            }
        }
        return loadedData;
    }
```

Pirmiausiai patikrinama, ar failas toje direktorijoje egzistuoja. Tuomet yra bandoma skaityti duomenis iš failo su FileStream ir StreamReader. Jeigu šifravimas buvo naudotas, tuomet duomenys yra dešifruojami. JSON tekstą su JsonUtility klasės metodu FromJson galima grąžinti kaip C# GameData objektą, ir turėti savo sukurtos klasės informaciją. Galiausiai grąžinama loadedData, kuri turės išsaugotą paskutinį žaidėjo žaistą lygį.

3.5. Vėjo ataka

Sukurtas naujas atakos tipas, imituojantis vėjo ataką. Dabar žaidime galima rinktis tarp ugnies, ledo ir vėjo atakų, kiekviena jų turi skirtingą veikimo principą, daroma žalą, greitį bei trukmę, kas kiek ataką galima naudoti.

```
    protected override void OnTriggerEnter(Collider other)
        {
            Monster monster = other.GetComponent<Monster>();
            if (monster != null){
            Vector3 knockbackDirection = (monster.transform.position -
transform.position).normalized;
```

```
            monster.ApplyKnockback(knockbackDirection, knockbackForce);
            monster.TakeDamage(abilityData.damage);
            }
            Destroy(gameObject);
        }
```

Kuomet aptinkamas monstras, yra apskaičiuojama priešinga kryptis, iš kurios atskriejo vėjo ataka, kuria monstras bus pakreiptas. Apskaičiavus šią kryptį, ji yra paduodama kaip parametras funkcijai ApplyKnockback esančiai Monster klasėje. Taip pat kaip parametras paduodamas atmušimo stiprumas.

```
public void ApplyKnockback(Vector3 direction, float force){
        Rigidbody rb = GetComponent<Rigidbody>();
        if (rb != null){
            rb.AddForce(direction * force, ForceMode.Impulse);
        }
    }
```

Monstro klasėje esantis metodas ApplyKnockback atrodo štai taip. Paėmus tam tikro monstro Rigidbody komponentą, galima pritaikyti Unity variklio Rigidbody.AddForce metodą. Kaip parametrus reikia paduoti kryptį bei rėžimą. Šiuo atveju paduodama apskaičiuota kryptis vėjo atakos kode su stiprumu, bei ForceModeImpulse rėžimas, kuris pagreitį skaičiuoja pagal stiprumo ir masės dalybą.

3.6. Ugnies monstras

Implementuotas naujas monstras, kuriam ugnies burtai nedaro žalos. Kadangi buvo sukurta vėjo ataka, patobulindama galimybės žaidėjui rinktis atakos, galima pridėti variacijas su tokiais monstrais, kurie turėtų specialių gebėjimų pasipriešinti žaidėjo atakomis. Vienas tokių galėtų būti ugnies monstras.

```
public override void TakeDamage(int damage, string sourceTag = null){

        if(sourceTag == Fireball.FireballTag){
            Debug.Log("Ugnis nedaro žalos šiam monstrui");
            return;
        }
        base.TakeDamage(damage);
    }
```

Tokiai implementacijai paprasčiausiai galima pasitelkti naudoti Unity variklio žymas. Kiekvienam burtui variklyje priskiriame žymą, šiuo atveju ugnies ataka turės žymą pavadinimu „Fireball“.

```
public virtual void OnTriggerEnter(Collider other)
{
    Monster monster = other.GetComponent<Monster>();
    if (monster != null)
    {
        monster.TakeDamage(abilityData.damage, gameObject.tag);
        Destroy(gameObject); // Sunaikina burtų objektą
    }
}
```

Tuomet burtų klasėje, kuomet burtai susiduria su tam tikru monstru, paduodama Monster klasės funkcijai TakeDamage kaip parametrai žala, kuri turėtų būti padaryta monstrui bei burtų žyma. O kadangi kiekvienas specifinis monstras paveldi iš Monster klasės, galima specifinėse monstrų klasėse perrašyti TakeDamage metodą, jog jis tikrintų žymas ir pagal tai būtų galima sukurti monstrus, kuriem žalos nedarytų ir kitos burtų atakos.

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.

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.
- Sukurta nustatymų sistema, leidžianti žaidėjams pakeisti esamus klaviatūros mygtukus į kitokius, taip pat išsaugant žaidėjo pasirinktus naujus pakeistus klavišus.
- Sukurta pagrindinio meniu scena, lygių scenų šablonai, klasė, padedanti pakeisti scenas.
- Sukurta žaidimo eigos kodas, žaidimas prasideda po kelių sekundžių, vyksta nustatytą lygio trukmę, galiausiai išmetamas žaidimo baigimo meniu su pasirinkimais eiti į sekantį lygį ar grįžti į meniu.
- Sukurta žaidimo būsenos išsaugojimo sistema, leidžianti vartotojui sukurti tris profilius, išsaugant paskutinį žaistą lygį. Galima pasirinkti žaisti anksčiausiai žaistą būseną iš meniu, sukurti naują būseną, pasirinkti iš trijų jau sukurtų būsenų.
- Sukurta vėjo ataka, atmušanti monstrus nuo žaidėjo.
- Sukurtas ugnies monstras, kuriam ugnies ataka nedaro jokios žalos.

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