Střední průmyslová škola elektrotechnická
a Vyšší odborná škola Pardubice

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNICKÁ

MATURITNÍ PRÁCE – PROGRAMOVÁNÍ

3D BUDOVATELSKÁ HRA

březen 2023 Roman, Šťulík, 4.E

„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.

Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejích ukázek pro výuku.“

V Pardubicích dne ....

podpis

Střední průmyslová škola elektrotechnická a Vyšší odborná škola Pardubice

ZADÁNÍ MATURITNÍ PRÁCE

Maturitní zkouška – profilová část – Maturitní projekt

Obor:	18-20-M/01 Informační technologie	**Školní rok:**	2022/2023
Jméno a příjmení žáka:	Roman Šťulík	**Třída:**	4.E
Téma maturitní práce:	3D budovatelská hra		
Vedoucí maturitní práce:	Mgr. Čestmír Bárta		
Pracoviště vedoucího:	SPŠE a VOŠ Pardubice		

Kategorie maturitní práce: Programování

Téma maturitní práce (popis):

Vytvořte 3D hru pro jednoho hráče s pohledem shora v herním enginu Unity. Bude se jednat o budovatelkou hru pro jednoho až dva hráče na jednom počítači. Cílem bude na náhodně generované mapě těžit a zpracovávat suroviny. Pomocí získaných materiálů stavět a vylepšovat automatickou obranu a pomocí ní odrážet vlny nepřátel. Hra nebude obsahovat konec pouze cíl, po kterém může volně hráč pokračovat do nekonečna.

Hlavní body – specifikace maturitní práce:

1. Náhodná generace mapy
2. Nepřátelský pathfinding
3. Možnost pro hráče měnit terén
4. Těžba a zpracovávání surovin
5. Pohyb hráče
6. Vyrábění a vylepšování pracovních stanic

Způsob zpracování maturitní práce:

Maturitní práce musí být zpracována v souladu se zákonem č. 121/2000 Sb., autorský zákon.

Maturitní práce je tvořena praktickou částí (viz téma a specifikace maturitní práce výše) a písemnou prací.

Maturitní práce je realizována žákem převážně v rámci výuky ve 4. ročníku. Lze pokračovat na projektu z nižších ročníků.

Podrobné pokyny ke zpracování maturitní práce příslušné kategorie jsou uvedeny v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Zpracování písemné práce musí odpovídat požadavkům uvedeným v dokumentu *MP_Formalni_stranka_dokumentace*. Písemná práce musí být rovněž zpracována v souladu s normou pro úpravu písemností [ČSN 01 6910 (2014)], s citační normou [ČSN ISO 690], se základními typografickými pravidly, pravidly sazby, gramatickými pravidly a pravidly českého pravopisu.

Pokyny k rozsahu a obsahu maturitní práce:

Rozsah a obsah praktické části maturitní práce je určen tématem a specifikací maturitní práce, rozsah písemné části maturitní práce je minimálně 15 normostran vlastního textu. Do

uvedeného rozsahu se nezapočítávají úvodní listy (titulní list, prohlášení, zadání, anotace, obsah…), závěrečné listy (seznam literatury…) a přílohy. Podrobné pokyny k rozsahu a obsahu maturitní práce příslušné kategorie jsou uvedeny v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Kritéria hodnocení maturitní práce a její obhajoby:

Maturitní práce a její obhajoba u maturitní zkoušky je hodnocena bodově. Celkový dosažený počet bodů je součtem bodů přidělených vedoucím práce a oponentem (maximální dosažitelný počet je 100 bodů). Výsledná známka u maturitní zkoušky je stanovena přepočtem celkového počtu bodů na známku pomocí tabulky uvedené níže.

V případě nesplnění tématu maturitní práce a v případě plagiátorství bude práce hodnocena stupněm „nedostatečný".

Tabulka bodového vyjádření hlavních kritérií a obhajoby:

- praktická část – zpracování tématu maturitní práce .....[max. 25 bodů]
- písemná část – zpracování, formální a obsahová stránka .....[max. 15 bodů]
- obhajoba maturitní práce před zkušební komisí .....[max. 10 bodů]

Podrobná kritéria pro hodnocení maturitní práce příslušné kategorie jsou uvedena v dokumentu *MP_Kriteria_hodnoceni*.

Tabulka přepočtu celkového počtu bodů na známku:

[0 .. 60 bodů]..... [5]
[61 .. 70 bodů]..... [4]
[71 .. 80 bodů]..... [3]
[81 .. 90 bodů]..... [2]
[91 .. 100 bodů]..... [1]

Požadavek na počet vyhotovení maturitní práce a její odevzdání:

Kompletní maturitní práce (praktická i písemná) se odevzdává ve stanoveném termínu vedoucímu maturitní práce. Písemná práce se odevzdává v jednom tištěném vyhotovení obsahující podepsaný přenosný nosič CD/DVD/SD s písemnou práci a sadou kompletních dat v elektronické podobě dle pokynů v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Termín odevzdání maturitní práce: 24. března 2023

Délka obhajoby maturitní práce před zkušební maturitní komisí: 15 minut

Pardubice 30. září 2022

Petr Mikuláš

Mgr. Petr Mikuláš, *ředitel školy*

Anotace

Hra zaměřená na budování základny a její obranu před nepřáteli. Obsahuje náhodnou generaci mapy. Možnost hrát v módu pro jednoho nebo dva hráče na jednom počítači.

Klíčová slova: budování, pro dva hráče, pohled shora, unity

Annotation

Game focused on building and defending base from enemies. Contains random map generation. It can be played in single-player or two-player on one computer mode.

Keywords:building, two player mode, top down view, unity

Obsah

Úvod

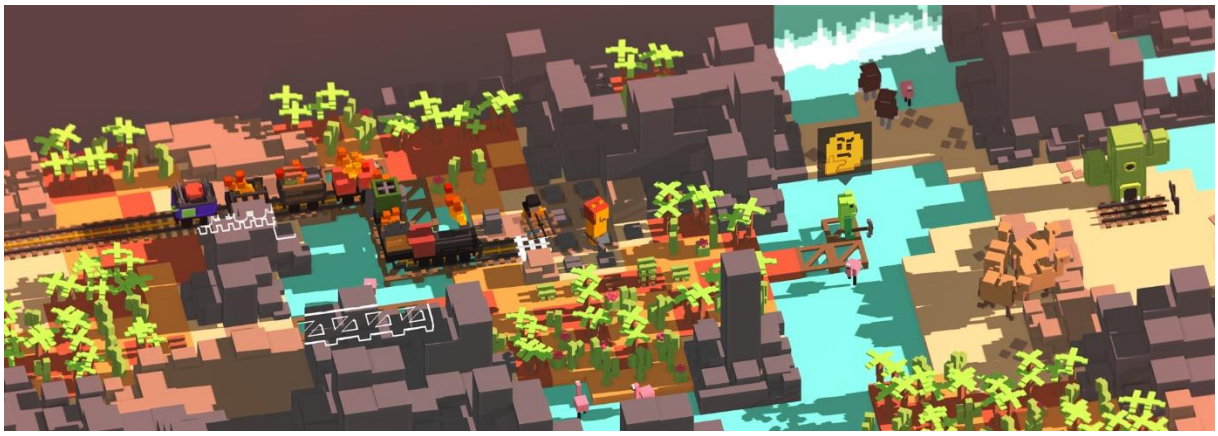
Jako maturitní projekt jsem se rozhodl vytvořit hru, protože to je něco, co jsem chtěl vždycky zkusit. V minulosti jsem začal pracovat na hře, která se hraje ve třech dimenzích a tak jsem věděl, že to rozhodně není něco co by jsem chtěl dělat. Nakonec jsem se rozhodl pro 3D hru s pohledem shora, protože to nejvíce vyhovovalo žánru hry.

Žánr a téma hry jsem si vybral při hraní hry Unrailed, která je uvedená v podobných projektech. Napadlo mě, jak by asi vypadala hra s herním stylem Unrailed zaměřená na budování a obranu základny. Věděl jsem, že chci dělat hru, ve které se konstantně něco děje a hráč si nemůže ani na chvíli odpočinout.

Na tom, že hra bude obsahovat multiplayer jsem se rozhodl po zvážení, co bych od podobné hry asi chtěl a očekával. Navíc možnost hrát hru s kamarády ji alespoň podle mě mění z hry, kterou člověk zapne jednou na něco co bude hrát opakovaně, když nemá s kamarády co dělat. Multiplayer také přidává extra vrstvu napětí mezi hráči ve chvílích, kdy se něco nepovede nebo se rozhodnout mezi sebou soutěžit.

1 Analýza obdobných projektů

1.1 Unrailed



Obrázek 1 Unrailed

Unrailed je 3d hra s pohledem shora ve které má hráč za úkol získávat suroviny a z nich na vagónu vlaku vyrábět koleje. Vlak se během hry zastaví, pouze pokud dojede do stanice a hráč tak musí pokládat koleje rychleji než se vlak pohybuje. Ve hře se nacházejí vylepšení, která jsou dostupná nakoupením nových vagónů nebo vylepšením těch stávajících. Hra je pro jednoho až čtyři hráče.

1.1.1 Kladné stránky

+ jednoduchá hratelnost

+ multiplayer

+ příjemná low poly grafika

+ dobrá optimalizace

+ náhodně generovaná mapa

1.1.2 Záporné stránky

- v režimu pro jednoho hráče je obtížnost velmi vysoká

- hra nenabízí konec

- hra může působit repetitivně

1.2 Factorio

Obrázek 2 Factorio

Factorio je 2d hra s pohledem shora zaměřená na vybudování továrny za účelem postavení rakety a následného opuštění planety. Základním úkolem hráče je těžba a zpracovávání surovin. Nejdůležitějším prvkem je pak automatizace výrobních procesů v továrně. Ve hře se nachází nepřátelé proti kterým musí hráč svou továrnu uchránit.

1.2.1 Kladné stránky

+ velká rozmanitost surovin a nástrojů

+ nepřátele kteří jsou zdrojem dodatečného obsahu

+ hra obsahuje konec ale i obsah pro hráče kteří ještě přestat nechtějí

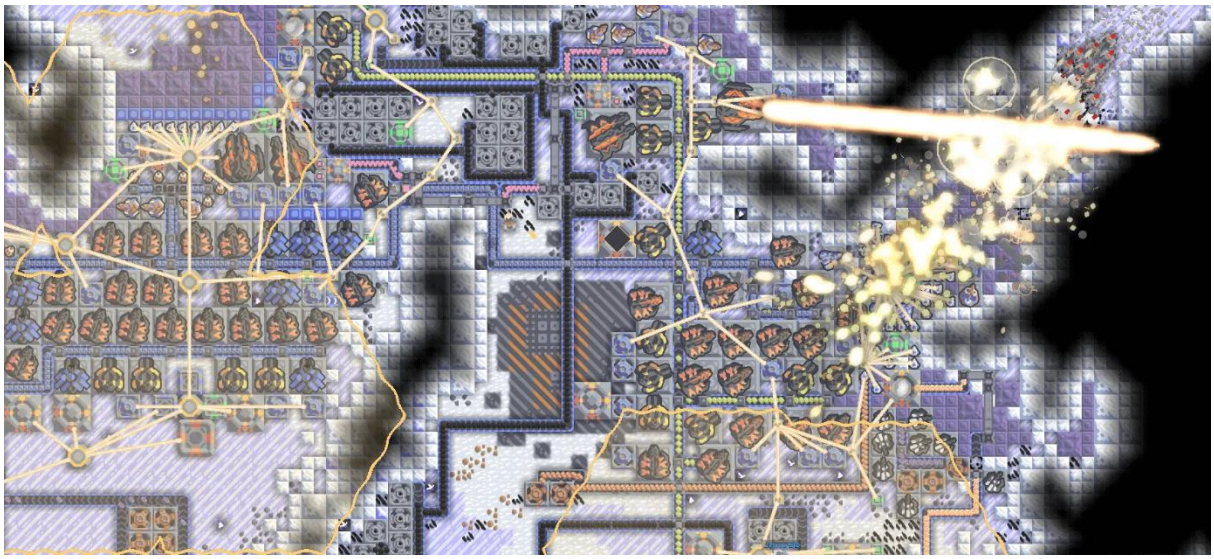
+ náhodně generovaná mapa

+ díky představováním nových herních mechanik hra odolává repetetivnosti

1.2.2 Záporné stránky

- vyšší složitost převážně v pozdějších fázích

1.3 Mindustry



Obrázek 3 Mindustry

Mindustry je 2d hra s pohledem shora zaměřená na obranu základny před vlnami nepřátel. K obraně základny je potřeba postavit automatickou obranu ze surovin které se nachází na mapě. Hra obsahuje předem vytvořené levely které jsou připravené tak aby hráči postupně představovali nové technologie. Po poražení určitého počtu vln nepřátel na každé mapě se další vlny zastaví a hráč získá volnou ruku nashromáždit tolik surovin kolik jen chce. Získané suroviny se využívají na výzkum nových technologií.

1.3.1 Kladné stránky

+ velké množství nejrůznějších staveb

+ vlny nepřátel přidávají pocit naléhavosti

+ velké množství kvalitních předem vytvořených levelů

+ několik herních módů (PVP, sandbox, hráč proti počítači, obrana)

1.3.2 Záporné stránky

- po skončení nepřátelských útoků se hra stává nudnou

- velmi špatné ovládání na mobilních telefonech

2 Návrh projektu

2.1 Cílové skupiny

Lidé, kteří nemají čas na dlouhodobé budování a chtějí krátký a intenzivní zážitek.

2.2 Základní prvky projektu

- Náhodné generování mapy
- Nepřátelský pathfinding
- Nepřátelské jednotky
- Ovládání a pohyb hráče
- Multiplayer
- Interakce hráče s okolím

3 Použité technologie

3.1 Unity

Pro mou práci jsem si vybral herní engine Unity. Unity jsem si vybral z důvodu mých předchozích zkušeností s ním. Dalšími důvody pro zvolení Unity byla jeho přístupnost a množství návodů na internetu.

3.1.1 Základní informace

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Unity poskytuje možnosti vývoje pro 2D i 3D hry libovolného žánru a zaměření. Kromě grafického prostředí pro tvorbu, podporuje také tvorbu skriptů v jazyce C#. Spíše vhodné pro tvorbu menší indie titulů. [5]

3.1.2 Alternativy

- Unreal Engine – Hlavní konkurent unity vyvíjený společností Epic Games. Využívá programovací jazyk C++. Spíše uzpůsobený pro velká studia.
- Source Engine 2 – Herni engine vyvíjený společností Valve. Využívá programovací jazyk C++. Ne tak univerzální jako unity.

3.2 Blender

Pro vytváření 3D modelů jsem využíval software Blender. Vybral jsem si ho z důvodu předchozích zkušeností s ním a možností použití 3D modelů, které jsem vytvořil dříve.

3.2.1 Základní informace

Blender je program pro tvorbu, texturování a animování 3D modelů vyvíjený společností Blender Foundation. Využívá se k tvorbě animací, modelů pro 3D tisk, prostředí pro virtuální realitu a dalších. [3]

3.2.2 Alternativy

- 3D Studio MAX – Profesionální program pro tvorbu 3D grafiky. Využívá se v postprodukci při výrobě reklam, filmů a videoherní grafiky. [1]
- Autodesk Maya – Program pro tvorbu assetů pro 3D animace, počítačové hry a filmy. [2]

3.3 Json.NET

Json.NET je open source knihovna pro serializaci a deserializaci JSON a XML od společnosti Newtonsoft. Knihovna umí převádět mezi JSON a XML, generovat JSON který je přehledný a automaticky serializovat a deserializovat do nejrůznějších datových struktur. [4]

Knihovnu využívám pro veškerou práci s JSONem, jako například importování receptů ze souboru.

```
public static void loadRecipes(){
    string json = Resources.Load<TextAsset>("json/receptyXD").text;
    recipes = JsonConvert.DeserializeObject
              <Dictionary<string, Recipe>[]>(json);
}
```

4 Náhodné generování mapy

Mapa je plochá čtvercová oblast s pevně danou velikostí, která je rozdělena do stejně velkých bloků. Každý blok je obsazen maximálně jednou entitou. Při generování mapy je potřeba následující:

- Základní parametry mapy
- Vytvoření terénu
- Ohraničení mapy
- Vytvoření ložisek surovin
- Zajištění generování entit ve volném prostoru
- Pracovní stanice
- Určení pozic základen

4.1 Základní parametry mapy

Velikost je pevně určená a určuje kolik bloků bude tvořit obě strany čtverce mapy.

Scale je vlastnost, která jaké oddálení bude použito pro noise generator, který určuje, jaký tvar bude terén mít.

Aby mapa působila dojmem náhodné generace používá se osmimístný seed, který může být zadán hráčem v menu hry nebo vytvořen náhodným generátorem čísel.

4.2 Vytvoření terénu

Před vytvořením terénu umístíme podklad, na kterém budou všechny entity umístěny. Následně se prochází všechny pozice na mapě a každé je přiřazena hodnota z šumu. Pozice čtené hodnoty je odsazen od středu šumu podle seedu, o první polovinu seedu na ose X a o druhou polovinu seedu na ose Y, šum je také "oddálen" pomocí hodnoty scale. Podle hodnoty šumu se určuje, jestli daný blok bude prázdný ne na něm bude nezničitelná skála nebo kamenné ložisko.

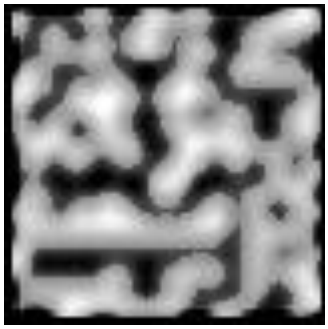
```
private float getSample(float x ,float y , int AreaSize){
    float xCoor = (float)(x/ AreaSize) * scale + offsetX;
    float yCoor = (float)(y/ AreaSize) * scale + offsetY;
```

```
        float sample = Mathf.PerlinNoise(xCoor,yCoor);
        return sample;
    }
```

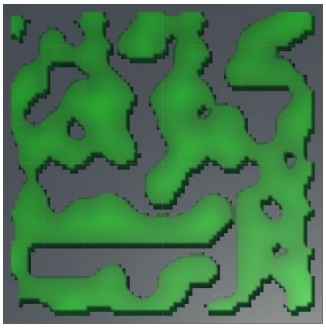
4.3 Ohraničení mapy

Aby hráč nemohl vypadnout z mapy musí být ohraničená. Ohraničení může být tvořeno neviditelnou bariérou nebo nezničitelnou skálou. Z estetického hlediska je skále lepší možností. Aby se hraniční zeď vytvořila s ohledem na existující terén upravuje se hodnota šumu pokud je bloku, pro něhož je určená v blízkosti okraje mapy. Pokud se blok nachází na samém okraji mapy umístí se na něj vždy skála.

```
if(Mathf.Abs(x) > 0.9*(MapSize/2)){
    int borderDistance = (MapSize/2) - (int)Mathf.Abs(x);
    sample += sample * 1f/borderDistance;
    if(borderDistance == 1){
        sample = 1f;
    }
}
```



Obrázek 5 Noise map



Obrázek 4 Generated map

4.4 Vytvoření ložisek surovin

Pro vytvoření ložisek je potřeba určit místo kde se budou nacházet, o jakou surovinu se bude jednat a následně vygenerovat jednotlivé objekty suroviny.

4.4.1 Určení pozice ložiska

Pozice ložiska je určena vzdáleností od středu mapy a směrem ve kterém se má ložisko vygenerovat. Po každém ložisku se úhel směru změní o 60 stupňů. K určení vzdálenosti od

středu se používají jednotlivá čísla seedu. Pokud je vzdálenost stejná jako u předchozího ložiska použije se opačná hodnota čísla seedu. Pro každé druhé ložisko se přičítá offset.

```
Vector3 OrespawnPoint = new Vector3(Mathf.Cos(radian), 0 , Mathf.Sin(radian)) *
                        (seedN * (size/32) + centeroffset);
OrespawnPoint =new Vector3( Mathf.Round(OrespawnPoint.x) , 2 ,
                          Mathf.Round(OrespawnPoint.z));
```

4.4.2 Vybrání suroviny

Druh vygenerované suroviny závisí na poměru vzdálenosti ložiska od hráčovy základny a vzdálenosti mezi dvěma základnami. Aby byla každá surovina zastoupená stejným množstvím ložisek není ji po určitém počtu zvolení možně vybrat.

```
int oreIndex = (int)Mathf.Round(
(Vector3.Distance(playerLocation,oreLocation) /
(Vector3.Distance(-playerLocation , playerLocation)) * (list.Count-1) ));
if(oreIndex>= list.Count)oreIndex = list.Count-1;
```

4.5 Zajištění generování entit ve volném prostoru

Aby se základny nebo ložiska surovin nenacházeli uvnitř skály, testuje se po jejich umístěním, jestli se nacházejí na volném bloku. Pokud se není daný blok volný spustí se funkce, která testuje pozice ve čtyřech směrech v postupně se zvadající vzdálenosti. Pokud funkce objeví volný prostor je dané ložisko nebo základna přemístěna tam.

4.6 Určení pozic základen

Pozice základen je zvolena pomocí seedu. Souřadnici se získá tak, že od zbytku po dělení poloviny seedu velikostí mapy se odečte polovina velikosti mapy. Pro každou souřadnici se proces dělá zvlášť. Pro osu X se použije první polovina seed a pro osu Y druhá.

```
int spawnPointX = (int)offsetX % size -size/2;
int spawnPointY = (int)offsetY % size -size/2;
```

Takto se získají souřadnice jedné základny. Pro druhou jsou hodnoty na obou osách opačné. Pokud se opačné pozice nacházejí příliš blízko ne daleko od sebe jsou odsezeny. Odsazení probíhá přičtení vzdálenosti mezi pozicemi vynásobenou multiplierem

```
Vector3 move = (-position - position) * multiplier;
return position += new Vector3(move.x , 0 , move.z);
```

5 Nepřátelský pathfinding

Pathfinding slouží k najití ideální cesty mezi základnami po které se budou pohybovat nepřátelské jednotky. Pathfinding musí umět následující:

- Najít ideální cestu
- Detekovat překážky na cestě
- Vyhnout se překážce
- Zajistit průchodnost mapy

5.1 Najití ideální cesty

Pro najití cesty se používá A* algoritmus. Cesta se nehledá neustále ale nalezená cesta je pouze předána nepřátelským jednotkám, které ji slepě následují.

5.1.1 A* algoritmus

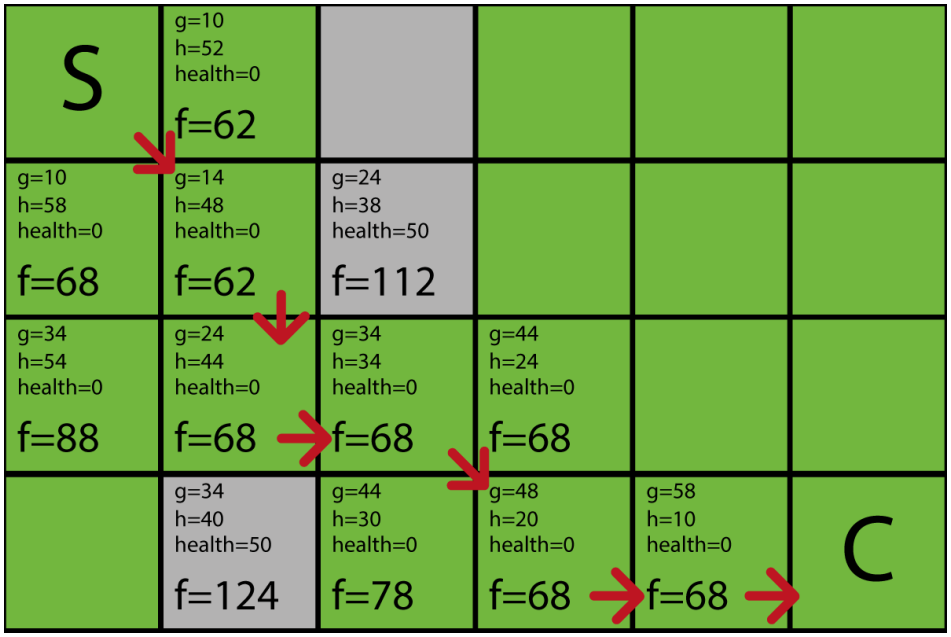
A* je algoritmus používaný pro hledání optimálních cest. Funguje na principu best–first (nejlepší možnost se zkouší jako první) a hodnocení jednotlivých uzlů. Před začátkem hledání cesty je tedy potřeba vytvořit mapu uzlů které budou sloužit jako záchytné body. Každý blok na mapě představuje jeden uzel. Každý uzel má hodnotu, kterou lze spočítat jako $f(n) = g(n) + h(n)$ kde g(n) je cesta od počátku k uzlu a h(n) je nejkratší teoretická cesta od uzlu k cíli.

Kromě hodnot z vzorce se přičítá i obtížnost pohybování se po daném poli. Obtížností je myšleno, jestli se na daném bloku nachází překážka a pokud ano kolik má životů. Vzorec je tedy $f(n) = [g(n) + health(n)] + h(n)$

Cesta od začátku k současnému uzlu se počítá jako součet ceny rodičovského uzlu, životů překážky na uzlu, cena za přesun na jiný uzel, která je 10 v případě rovného posunu a 14 v případě diagonálního a překážek na vedlejších uzlech v případe diagonálního pohybu.

Samotné hledání cesty probíhá tak, že algoritmus se nachází na nějakém uzlu. Jako první se zkontroluje, jestli je daný uzel cílem cesty. Pokud není spočítají se hodnoty všech sousedních uzlů a uloží se současný uzel jako rodič pak se uzly uloží do seznam

spočítaných uzlů. Následně se ze seznamu vybere uzel s nejmenší hodnotou a cyklus se opakuje. Jakmile algoritmus dojde na konec pomocí hodnoty rodiče v uzlech se po vlastních stopách dostane zpět na začátek, přičemž po cestě ukládá každý uzel do cesty.



Obrázek 6 Pathfinding

5.2 Detekování překážek na cestě

Detekování překážek, které jsou přímo na cestě probíhá při vytvoření mapy uzlů a je zapsáno do hodnoty *health*. Další detekce překážek je potřeba pouze při diagonálním pohybu. Připočítávají se při ní hodnoty *health* uzlů sousedících s uzlem kam se přesouváme a zároveň uzlem na kterém se nacházíme.

Aby bylo možné detekovat vznik nové překážky za běhu, ukládají se pozice všech bloků, které ovlivňují průchodnost cesty. Při pokládání pracovních stanic a další pevných objektů se potom porovnává jejich pozice s uloženými pozicemi.

```
Vector2 Coordinates = grid.Coordinates(position);
grid.Nodegrid[(int)Coordinates.x, (int)Coordinates.y].walkAbility =
             (int)stationPlaced.GetComponent<ObHealthBar>().currentHealth;
int index = 0;
if(pathFindingOb.GetComponent<PathFindingColtroller>().dictPath.
                                TryGetValue(position, out index)){
    pathFindingOb.GetComponent<PathFindingColtroller>().changePath(index,
index+1);
}
```

5.3 Vyhnutí se překážkám

Vyhnutí se překážkám při generování cesty probíhá hledáním cesty kolem. Pokud cesta kolem neexistuje nebo je moc dlouhá v porovnání s množstvím životů překážky rozhodne se algoritmus, že nejlepší je překážku zničit.

Pokud se nová překážka objeví za běhu spustí se funkce, která má za úkol najít cestu kolem překážky. Aby cesta vypadala přirozeně hledá se nová trasa mezi uzlem o 2 před překážkou a o 3 za ní. V případě, že neexistuje žádná lepší cesta může se algoritmus rozhodnout novou překážku rozbít.

5.4 Zajištění průchodnosti mapy

Průchodností mapy je myšleno, aby existovala cesta mezi základnami, která nevede přes nezničitelnou skálu nebo ven z mapy. Vzhledem k náhodné generaci pomocí šumu se může stát, že se mapa rozdělí na 2 poloviny a stane se tak neprůchozí. V takovém případě nemá algoritmus jinou možnost, než zvolit cestu skrz terén. Pokud vede výsledná trasa skrz terén je v kroku kdy se algoritmus vrací ve vlastních stopách zničen.

```
if(checking.walkAbility > 1000){
      Vector2 Coordinates = grid.Coordinates(checking.location);
      grid.Nodegrid[(int)Coordinates.x,(int)Coordinates.y].walkAbility = 0;
      RaycastHit hit = new RaycastHit();
      Physics.Raycast(new Vector3(checking.location.x , 2,checking.location.z),
                      Vector3.down, out hit , 1f);
      Destroy(hit.collider.gameObject);
}
```

6 Nepřátelské jednotky

Nepřátelské jednoty jsou generovány jejich základnou, poté se vydávají k hráčově základně a ničí vše co jim stojí v cestě. Aby tento proces fungoval je potřeba zajistit vytváření jednotek v pravidelných intervalech, jejich pohyb a schopnost zničit objekty a stavby, které jim stojí v cestě.

6.1 Nepřátelská základna

Nachází se na opačné straně mapy než ta hráčova a její zničení je také hráčovým hlavním cílem. Zajišťuje generovaní generování nepřátelských jednotek.

Při generování mapy se spustí cyklus nepřátelských vln. Před první vlnou je odpočet, jehož délka závisí na obtížnosti, ale v základu je 120 vteřin. Při generování vlny nepřátel je třeba zvolit z jakých jednotek se bude vlna skládat. Jednotky se volí náhodným generátorem čísel. Aby byla úroveň obtížnosti vln správná mají jednotky číselný údaj určující jejich sílu a při jejich vybrání je tato hodnota odečte od číselné hodnoty obtížnosti momentální vlny. Pokud je úroveň nepřátelské jednoty větší než zbývající hodnota úrovně postupuje funkce skrz seznam nepřátel směrem k jednotkám s nižší obtížností, dokud nenarazí ne nějakou, která odpovídá zbývající hodnotě obtížnosti.

```
private int chooseEnemy(){
    int r = Random.Range(0, Enemies.Length);
    if(Enemies[r].GetComponent<movement>().Cost <= level-spawned){
        return r;
    }else{
        for(int i = r; i >= 0; i--){
            if(Enemies[i].GetComponent<movement>().Cost <= level-spawned)
                return i;
            }
        }
    }
```

6.2 Pohyb jednotek

Nepřátelské jednotky se pohybují po cestě vytvořené pathfinding algoritmem. Vždycky se pohybují směrem k dalšímu bodu na cestě. Když se dostanou tam, kam směřují zavolá se funkce *nextNode*.

6.2.1 Next node

Slouží ke změně momentálního cíle jednotek. Nejdříve zkontroluje, jestli se jednotka nenachází na konci cesty nebo na dostřel od svého cíle (hráčovy základny). Pokud je na dostřel přestane se pohybovat a začne na základnu útočit. Jestli že se nenachází na konci cesty zkontroluje se hodnota průchodnosti uzlu, které je v cestě o hodnotu dostřelu před jednotkou. Když je hodnota větší než 0 znamená to, že se na uzlu nachází objekt, který je potřeba zničit. Pokud je hodnota 0 je jednotce přiřazena další pozice v cestě, na kterou se má přesunout.

```
public virtual void GetnextNode(){
    if(index+ attackSc.range+1 < spawnScript.path.Count){
        if(spawnScript.path[index+ attackSc.range].walkAbility > 0){
            attack();
        }
        index++;
        currentNode = nextNode;
        nextNode = spawnScript.path[index].location;
    }else{
        end = true;
        attack();
    }
}
```

6.2.2 Pohyb

Pohyb je fyzikální a funguje na principu přiřazování rychlosti komponentu rigidbody. Každý snímek, pokud jednotka zrovna neútočí do nějakého objektu, se pohybuje po přímé lince mezi předchozím a následujícím bodem na trase k hráčově základně. Pokud se před jednotkou při pohybu objeví předmět pokusí se ho jednotka zničit.

6.3 Útok

Pro útok potřebují jednotky pozici cíle, tu dostávají ze scriptu zajišťujícího pohyb. Může se jednat o překážku na cestě nebo náhodný objekt, který se objevil před jednotkou (např. hráč nebo jiná jednotka). Když zná jednotka lokaci pokusí se zjistit o jaký objekt se jedná a získat referenci na něj pomocí raycastu. V případě že úspěšně identifikuje objekt přidá ho do seznamu cílů, zastaví pohyb a začne na něj útočit. Útok přestává, pokud je

objekt zničen nebo mimo dosah. Než se ale jednotka může znovu začít pohybovat kontroluje jestli je seznam cílů prázdný.

```
public void addToTarget(Vector3 location){
    hits = Physics.RaycastAll(location + new Vector3(0,5f,0), Vector3.down,5f);
    if(hits.Length > 0){
        foreach(RaycastHit hit in hits){
            if(hit.collider.transform.GetComponent<HealthBar>() != null){
                targets.Add(hit.collider.transform.GetComponent<HealthBar>());
            }
        }
        startAttack();
        }
    }
```

7 Ovládání a pohyb hráče

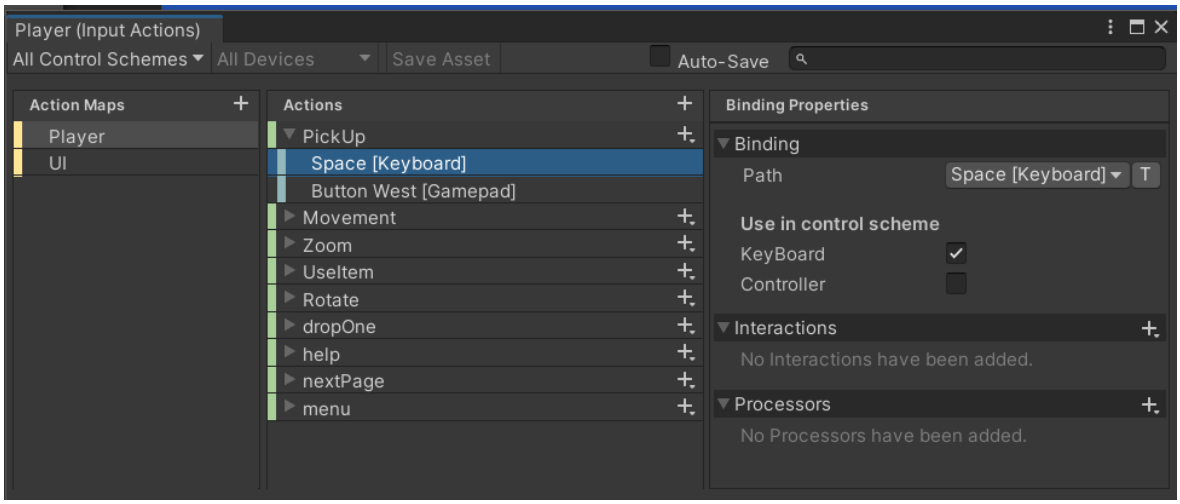
Hráče představuje samotná postava, kterou hráč ovládá ale i kamera a Canvas který zajišťuje uživatelské rozhraní nesdílené mezi hráči. K ovládání postavy hráče ale i dalších komponent hry se využívá nový unity input systém.

7.1 Unity input systém

V první fázi pracoval projekt se starým Input managerem, který je starý způsob pro řešení vstupů. Později jsem pro jednodušší implementování ovládání na ovladači a multiplayeru přešel na nový Input system package.

7.1.1 Input system package

Jedná se nový způsob získávání vstupů. Hlavní změnou je zjednodušení implementace ovládání pomocí různých zařízení (klávesnice, ovladač atd.) a přizpůsobení ovládání.



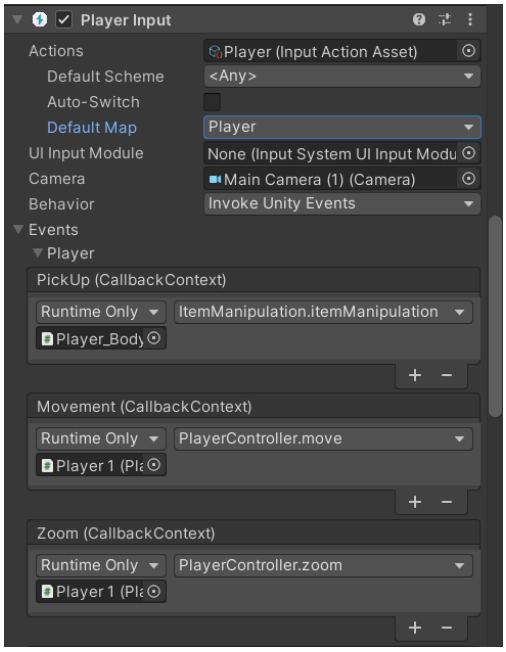
Obrázek 7 Action mapy

V novém systému se definují takzvané action mapy, které slouží ke sdružení ovládání v určitých situacích ve hře například action mapa pro pohyb pěšky, v autě nebo navigaci v menu. Uvnitř map se nachází actions ke kterým se přiřazují bindings. Do binding se nastavuje při jakém inputu se má daná action spustit.

Objektu, který chceme ovládat pomocí action se přiřadí *Player Input* komponent. Uvnitř komponentu definujeme action mapu a poté které funkce nebo eventy se mají

spustit, když se každá akce aktivuje. Druhou variantou jak spustit funkci pomocí action je definovat její zavolání přímo ve scriptu. Pro definování stačí přistoupit k objektu PlayerInputActions a přidat k akci funkci kterou chceme, aby spouštěla.

```
playerInputActions = new PlayerInputActions();
playerInputActions.UI.move.started += move;
playerInputActions.UI.Use.started += use;
playerInputActions.UI.Ecs.started += esc;
```



Obrázek 8 Player input

7.2 Pohyb hráče

Pohyb hráče je fyzikálně založený. Při zmáčknutí klávesy se spustí akce, která přečte hodnotu 2D vektoru. Hodnota následně slouží jako směr, kterým se objekt hráče pohybuje po ploše mapy. Podle směru se také natočí postava hráče tak aby směrovala tam kam se hráč pohybuje.

```
public void move(InputAction.CallbackContext context){
    input2 = context.ReadValue<Vector2>();
}
```

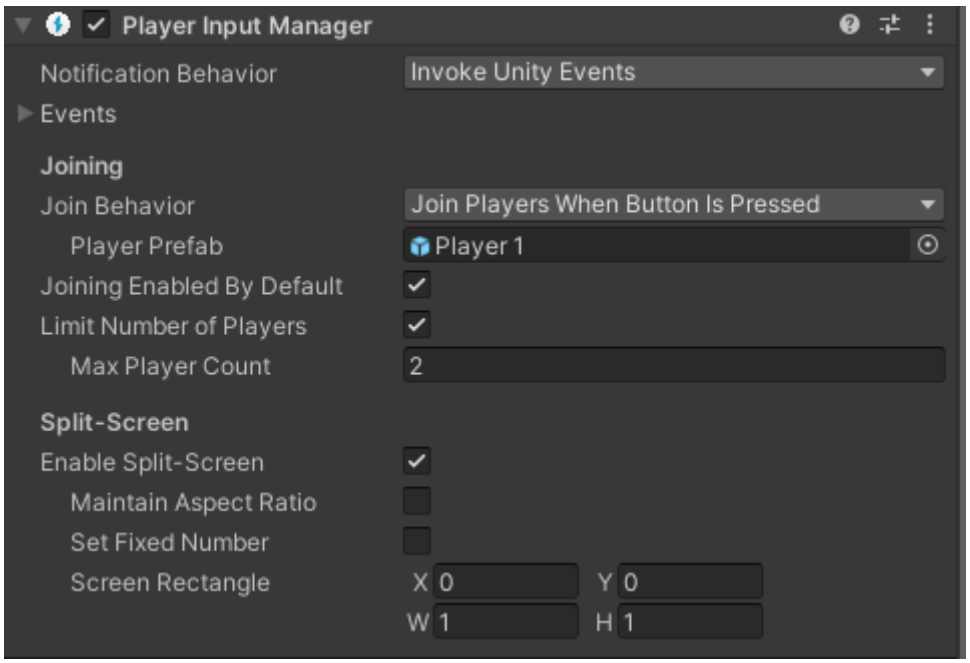
```
if(input2  != new Vector2(0,0)){
      input  = new Vector3(input2.x , 0 , input2.y).normalized * walkSpeed;
```

```
        playerRigidBody.velocity = input;
    }
```

8 Multiplayer

Ve hře se nachází lokální multiplayer. Funguje na principu splitscreen (rozdělené obrazovky) kdy jeden hráč vidí svou postavu na horní polovině obrazovky a druhý na té spodní. Rozdělení obrazovky a ovládání dvou hráčů naráz zajišťuje komponent *Player Input Manager*. Pro připojení druhého hráče stačí, stisknou tlačítko na jiném zařízení, než které používá první hráč. V takovém případě komponent detekuje input od nového zařízení a automaticky vytvoří druhý objekt hráče a rozdělí obrazovku. Množství hráčů není teoreticky omezeno na dva, ale při větším počtu se objevují problémy s čitelností UI jednotlivých hráčů. Každý hráč ovládá pouze dění na jeho vlastní polovině obrazovky. Výjimkou jsou menu otevřená přes rozdělení obrazovek, ty ovládají oba hráči současně.

```
if(Container.childCount == 2){
    for(int i = 0; i<Container.childCount; i++){
       Container.GetChild(i).GetChild(1).GetComponent<Camera>().rect =
      new Rect(0,0.5f*i,1,0.5f);
       Container.GetChild(i).GetChild(2).GetComponent<CanvasScaler>().reference
Resolution = new Vector2(3840,2160);
       Container.GetChild(i).GetChild(2).GetComponent<CanvasScaler>().matchWidt
hOrHeight = 0;
    }
}
```



Obrázek 9 Player input manager

9 Pracovní stanice

Aby byly pracovní stanice standardizované všechny používají třídu Station nebo jednu z jejích rozšiřujících tříd pro ovládání a sloty pro uchovávání předmětů.

9.1 Slot

Slot slouží k pokládání předmětů do stanice. Sloty akceptují jen specifické předměty. Když se hráč pokusí položit do slotu předmět spustí se funkci *chooseSlot*. Funke ověří, jestli lze předmět na slot položit, pokud to není možné funkce se zavolá na jiném slotu. Slot také může informovat stanici o položení nebo odebrání předmětu zavoláním funkce *notify*.

9.2 Station

Třída *Station* zajišťuje ovládání a základní funkčnost stanic. Použít stanici lze přímo funkcí *Activate* nebo ji může aktivovat jeden ze slotů funkcí *notify*. Při aktivaci stanice nejdřív ověří, jestli kombinace předmětů ve slotech odpovídá nějakému receptu. Po nalezení receptu se předměty ze slotů odeberou. Následně se do slotů umístí výstupní recepty. Pokud se stanice nevyužívá výrobě předmětů má vlastní zděděnou verzi activate.

9.3 Station Placement

Všechny stanice se pokládají stejným způsobem. Pokud hráč drží v ruce předmět stanice a použije ho, objeví se před ním zástupná stanice.

```
public virtual Vector3 placeVector(){
    Vector3 v3 = player.position + player.forward*2.5f;
    return new Vector3(Mathf.Round(v3.x), 1f , Mathf.Round(v3.z));
}
```

Zástupná stanice kontroluje, jestli je místo, na kterém se nachází volné. V pravidelných intervalech se kontroluje pozice a rotace hráče a pokud se změnily přemístí se pozice zástupné stanice. Pokud hráč použije předmět stanice znovu zástupná stanice zmizí a na jejím místě se objeví položená stanice. Při položení stanice se zkontroluje, jestli

není v cestě nepřátelských jednotek, případně se zavolá funkce, která vygeneruje cestu kolem.

10 Interakce hráče s okolím

Hráč potřebuje interagovat s několika druhy objektů:

- Suroviny a předměty na zemi
- Pracovní stanice
- Nástroje

10.1 Suroviny a předměty na zemi

Interakce se surovinami a všemi dalšími předměty ve hře probíhá stejně a tudíž je třeba zajistit základní operace s předměty, kterými jsou: sebrání předmětu ze země, položení na zem nebo pracovní stanici, prohození s předmětem který leží tam kam se snažím položit. Všechny tyto operace spouští akce *Pick up*. Akce spouští stejnou funkci bez ohledu na to, co chce hráč dělat takže nejdříve je potřeba zjistit co by to mohlo být. K tomu slouží funkce detekující objekty, které se nachází před a pod hráčem.

```
private void groundManipulation(bool forward = true){
        GameObject onGround = detect(true, forward);
        if(onGround != null){
            if(onGround.layer == LayerMask.NameToLayer("Station")){
                    if(!machineInteraction(onGround) && detectInFront)
                            groundManipulation(false);
            }else{
                if(DropItem(onGround, transform.position, carriedItems))
                      PickUp(onGround);
            }
        }else{
           DropItem(null , transform.position, carriedItems);
        }
}
```

10.1.1 Detect

Protože hráč pravděpodobně kouká na to, s čím chce interagovat tak se uvnitř funkce se nejdřív pošle raycast směrem dopředu. Pokud něco trefí funkce vrátí odkaz na trefený objekt, kterým může být buď slot pracovní stanice nebo komínek předmětů. Pokud

ne vyšle se druhý raycast tentokrát směrem dolů, který také vrací slot nebo komínek. Detect se taky s omezenou funkčností spouští periodicky. V tu dobu pouze detekuje komínky s předměty. Pokud najde nějaký se stejnými předměty, jaké hráč nese, sebere z něj všechny předměty nebo alespoň tolik aby nesl maximum, co může.

10.1.2 Předměty na zemi

Pokud funkce Detect vrátí objekt komínku předmětů nebo nic, program pokračuje do části, kde se snaží položit předměty na zem a popřípadě sebrat ty, které tam jsou. Pokud detekoval komínek se stejnými předměty, jaké drží. Prostě je přidá do komínku. V případě, že předměty jsou odlišné, zkusí je vyměnit s těmi, co drží v ruce, když ani to nejde, nemůže nic dělat a proces se zruší.

10.1.3 Předměty a pracovní stanice

Pokud funkce detect vrátí slot pracovní stanice jako první program vybere slot, do kterého by se měli předměty položit. Při vybírání slotu se jako první kontroluje ten, co hráč dostal od funkce detect. Jako první se zkontroluje, jestli slot přímá předměty, které hráč drží. Potom se zkontroluje, jestli je slot prázdný nebo se předměty ve slotu mohou prohodit s těmi, co drží hráč. Pokud kontrola slot zamítne, přesune se na další slot, jestliže žádný další slot není s předměty se nic neděje a proces se zruší.

10.2 Pracovní stanice

Při aktivaci funkce pro použití předmětu v ruce se nejdřív zkontroluje, jestli postava hráče nekouká směrem na stanici pomocí raycastu. Pokud raycast vrátí objekt, který má připojený komponent *Station* zavolá funkci *Activate*. Ve speciálních případech jako je třeba natahování elektrického plotu probíhá obsluha stanice jinou specifickou cestou.

10.3 Nástroje

Když raycast mířící na stanice žádnou netrefí a zároveň hráč drží předmět s třídou *tool* nebo jednou z jeho rozšiřujících tříd zavolá se její funkce *use*. Nástroje mohou sloužit

k hodně různým věcem ale většina je buď nástroj na těžbu surovin, nebo stanice pro položení na zem.

Nástroj, který patří do vrstvy tool se při sebrání umístí na jiné místo vzhledem k postavě hráče než obyčejné předměty nebo nástroje nepatřící do vrstvy. Takové nástroje mají taky komponent, který určuje, jakou rotaci budou mít po sebrání.

```
public Vector3 rotation;

public void setRotation(){
    transform.eulerAngles += rotation;
}
```

Závěr

Cílem projektu bylo vytvořit hru v herním eginu unity, jejíž tématem je budování a obrana základny. A to se povedlo. Spolu s tvorbou samotné hry jsem také vytvářel grafický obsah a 3D modely.

Projekt měl pro mě velký význam, protože jsem se naučil hodně o vývoji her a problémech, které to přináší. Také jsem se zlepšil v navrhování a tvorbě 3D modelů v závislosti na zasazení do hry. Nejdůležitější věc, co jsem se naučil je postupovat při tvorbě po malých krocích a nespěchat k hotovému projektu, ale dát si záležet na každém jeho aspektu.

Do budoucna bych rád rozšířil hru o několik malých vylepšení jako podrobnější nastavení, možnost vytvořit vlastní mapu nebo přidávání druhů nepřátel a samozřejmě ukládání postupu. Co se složitějších úprav týká rád bych se pokusil přidat do hry možnost hrát hru s více hráči prostřednictvím internetu a také vytvořit verzi hry pro mobilní telefony.

Seznam použité literatury a zdrojů obrázků

1. 3D Studio MAX: 3D Studio MAX. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-03-14]. Dostupné z: https://cs.wikipedia.org/wiki/3D_Studio_MAX
2. Autodesk Maya: Autodesk Maya. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-03-14]. Dostupné z: https://en.wikipedia.org/wiki/Autodesk_Maya
3. Blender: Blender. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-03-14]. Dostupné z: https://cs.wikipedia.org/wiki/Blender
4. Newtonsoft. *Newtonsoft: Json.net* [online]. 2023 [cit. 2023-03-22]. Dostupné z: https://www.newtonsoft.com/json
5. Unity (herní engine). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/Unity_(herní_engine)

Seznam obrázků