2024

Křesťanské gymnázium

Kozinova 1000

102 00 Praha 10 – Hostivař

Vymítač virů

ANNA LIBICHEROVÁ

Prohlášení

Prohlašuji, že jsem maturitní práci zpracoval/a samostatně a že jsem uvedl/a všechny použité informační zdroje a literaturu v seznamu použitých zdrojů.

Poděkování

Ráda bych poděkovala vedoucí našich maturitních projektů Mgr. Ivoně Spurné za podporu a její cenné rady.

Abstrakt

Tato maturitní práce se zabývá průběhem tvorby a funkcí mé 2D RPG videohry. Hra s názvem “Vymítač virů” využívá prvky gamifikace k obohacení vědomosti studentů a lepšímu porozumění učiva a zároveň poskytuje možnost volnočasové aktivity. Výsledky přinášejí cenné poznatky o tvorbě v Unity Engine, digitální umělecké tvorbě a programování v C sharp.

Klíčová slova

RPG, videohra, pixel, 2D, vir

Abstract

In English

This graduation thesis explores the process of creating and the functions of my 2D RPG video game. The game titled "Vymítač virů" utilizes gamification elements to enrich students' knowledge and enhance their understanding of the curriculum, while also providing an opportunity for leisure activities. The results yield valuable insights into working with the Unity Engine, creating digital arts and programming in C sharp.

Key words

RPG, videogame, pixel, 2D, virus

Obsah

Souhrny objektů

Seznam obrázků

Seznam zkratek

RPG	Hra na hrdiny
2D	Dvoudimenzionální
UI	Uživatelské rozhraní
NPC	Nehratelná postava

Úvod

Cíle této práce jsou definovány vytvořením funkční RPG hry v Unity, která bude nejen vizuálně atraktivní, ale také nabídne zajímavý a poutavý příběh. Součástí cílů je i zdokonalení dovedností v oblasti programování, grafického designu a tvorby herního obsahu. Práce se také zaměří na technické aspekty vývoje hry v Unity a možnosti aplikace v různých vzdělávacích kontextech. Přínos této práce spočívá jednak v konkrétním vytvoření hry, která může být následně prezentována a sdílena, a zároveň v posílení dovedností v oblasti herního designu a programování v prostředí Unity.

Teoretická část

Vliv edukativních videoher na vzdělávání studentů

Gamifikace je proces aplikace herních prvků a technik do nehracího prostředí. Přináší mnoho výhod v oblasti vzdělávání. Zvyšuje motivaci studentů a poskytuje okamžitou zpětnou vazbu, což pomáhá studentům vidět svůj postup a identifikovat oblasti k zlepšení. Tato metoda také podporuje spolupráci mezi studenty a zlepšuje jejich sociální dovednosti. Výzkumy naznačují, že gamifikace může vést k vyšší úspěšnosti žáků a zlepšení jejich výkonu.

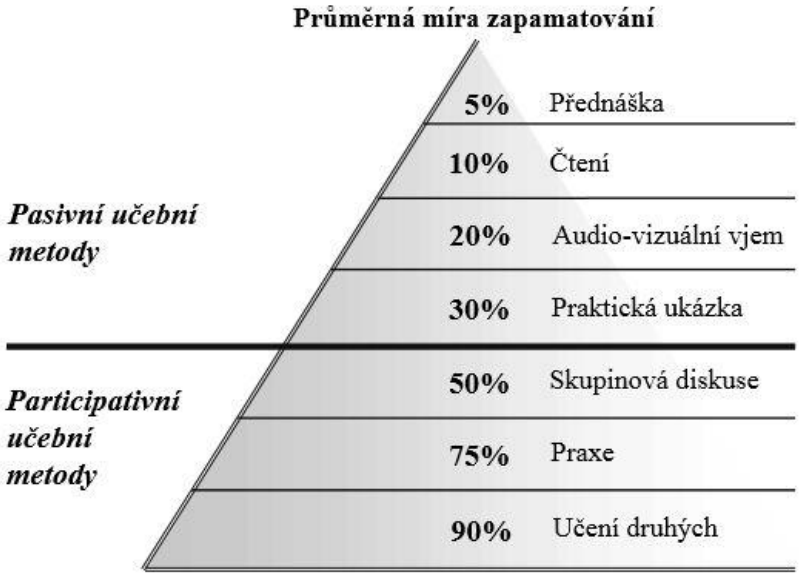
„Game-based Learning“ (vzdělávání založené na hrách) je metoda výuky, která využívá prvky her nebo celé herní prostředí k dosažení vzdělávacích cílů. Zvyšuje zapojení studentů díky zábavnosti a pestrosti herních prvků. Tato metoda usnadňuje výuku a poskytuje studentům možnost zapojit se do vzdělávacího procesu s radostí a zájmem. Dále podporuje rozvoj kritického myšlení a kreativity studentů, neboť je motivuje k řešení problémů a hledání nových způsobů řešení. Stejně jako gamifikace, i „Game-based Learning“ umožňuje poskytnutí okamžité zpětné vazby a individualizaci učení, což umožňuje studentům postupovat vlastním tempem a zaměřit se na oblasti, které potřebují nejvíce zlepšit. (1)

Definice videohry

Videohry jsou vnímány jako forma zábavy, charakterizovaná různými kritérii, včetně pravidel, cílů, průběhu a soutěže. Tyto kritéria přispívají k vytváření různých žánrů her. Jako příklad lze uvést hrací kostku, kde jedna kostka obsahuje specifická pravidla, omezené pohyby, a je omezena v možnostech. Hlavním cílem této hry je vyřešit danou situaci, přičemž průběh hry je ovlivněn náhodou, a soutěž může spočívat v rychlosti, s jakou hráč danou situaci zvládne. Například desková hra "Člověče, nezlob se!" má definovanou herní mapu, specifická pravidla, cíl spočívající v dosažení určitého místa na mapě, a soutěžící interagují s ostatními hráči během hry. Díky rozmanitosti žánrů videoher lze vytvořit edukativní hru zaměřenou na libovolné téma ve vzdělávání. (1)

Pyramida učení

Pyramida učení je schéma vytvořené od Národní výcvikové laboratoři (National training Laboratories) v Betelu, Maine ukazuje míru retence informací podle způsobu příjmu. Člověk si si zapamatuje 10 % toho, co přečetl, 20 % toho, co uslyšel a 30 % toho, co uviděl. Vyšší procenta zastupují participativní učební metody. Jde vidět, že metody, kdy se žák zapojí do učiva jsou nejúčinnější. Aplikace této pyramidy na edukativní videohry může posílit proces učení tím, že hráče postupně zapojuje do aktivit a umožňuje jim získávat znalosti a dovednosti prostřednictvím interakce s herním prostředím. (1)



Obrázek 1 – Pyramida (5)

2D hry v Unity

Unity poskytuje mnoho funkcí pro efektivní vytváření 2D videoher. Slouží primární pro zpříjemnění práce.

Sprite editor

Unity obsahuje nástroj jménem „Sprite editor“, který slouží pro tvorbu a úpravu 2D grafik, takzvaných „spritů“, přímo v editoru. Lze vytvářet, importovat a upravovat své sprite

atlasy. Z jednoho atlasu je možné vyříznout několik menších obrázku. Díky této schopnosti máme v Unity knihovnu grafik, které snadně a rychle můžeme vložit do své hry.

Tilemap editor

Tilemap editor slouží pro práci se sprity jako s dlaždicemi. Je to efektivní způsob, jak vytvářet pozadí a různé levely 2D hry. Můžete vytvářet složité levely pomocí různých typů dlaždic a vrstev. Tilemap editor nabízí spoustu užitečných nástrojů jako například štětec, kapátko, nástroj na označování a kopírování a guma. Díky nim můžu kopírovat vzory, či smazat velké části za pár sekund.

Prefabs

Prefabrikáty jsou předdefinované objekty nebo skupiny objektů, které můžete vytvořit a uložit do knihovny, aby byly znovu použitelné v různých scénách vaší hry. Prefabrikát obsahuje nastavení objektu, včetně jeho komponent, umístění, velikosti, materiálů a dalších vlastností.

Funkce uživatelského rozhraní

Dále Unity nabízí nástroje pro návrh uživatelského rozhraní. Můžete vytvářet tlačítka, textová pole, obrázky a další prvky UI pro jednoduší interakci hráče se hrou.

Funkce animace

Unity podporuje animace pro 2D sprite grafiku. Můžete vytvářet animace pohybu, změn stavu a další pomocí editoru animací. Editor sleduje změnu v lokaci, stavu, změnu grafiky a mezi změnami automaticky vytváří přirozené přechody.

Text mesh pro

Text Mesh v Unity je komponent, která umožňuje zobrazovat text ve 3D prostoru. Jedná se o alternativu klasického 2D textu, který se zobrazuje na ploše. Text Mesh umožňuje vytvářet text, který může být umístěn ve scéně ve formě 3D objektu. Tato komponenta umožňuje nastavit různé vlastnosti textu, jako je font, velikost, barva, zarovnání, tloušťka písma atd. Text Mesh je často využíván v 3D hrách pro zobrazování informací, dialogů, uživatelského rozhraní atd. (4)

C Sharp

C Sharp běžně známý jako C# je programovací jazyk vyvinutý od Microsoftu. Je užíván na tvorbu webových aplikací, aplikací na plochu počítače, mobilní aplikace, hry a mnoho dalšího. Jednou z hlavních charakteristik C# je jeho objektově orientovaný přístup, který umožňuje strukturování kódu pomocí tříd, objektů, dědičnosti a polymorfismu.

Třídy a objekty

Třída je jakýmsi návrhovým vzorem, definujícím vlastnosti a chování objektu. Každá třída může obsahovat atributy (data) a metody (operace), které spolu logicky souvisejí. Objekt je instance třídy, což znamená konkrétní reprezentaci abstraktního návrhu definovaného ve třídě. Tyto objekty mohou obsahovat data a metody které jsou specifikovány ve třídě. V následující ukázce třída „Objekt“ má v sobě místa pro informace, které instance „mujObjekt“ doplňuje.

```
class Program
{
    static void Main()
    {
        class Objekt
        {
            public string Jmeno { get; set; }
            public int Vek { get; set; }

            public void VypisInformace()
            {
                Console.WriteLine($"Objekt: {Jmeno}, Věk: {Vek}");
            }
        }

        Objekt mujObjekt = new Objekt
        {
            Jmeno = "MujObjekt",
            Vek = 25
        };
        mujObjekt.VypisInformace();
    }
}
```

Dědičnost

Dědičnost umožňuje třídě (nazývané "podtřída" nebo "odvozená třída") dědit atributy a metody od jiné třídy. Dědičnost přispívá k znuvupoužitelnosti kódu, hierarchické organizaci tříd a efektivnějšímu návrhu programu. Střípek kódu ukazuje nadřazenou třídu, která nese proměnou „Jmeno“ a podřazenou třídu která díky dědičnosti má danou informaci. (3)

```
class RodicovaTrida
{
    protected string Jmeno = "jméno";
}
class PodrizenáTrida : RodicovaTrida
{
    public void VypisJmeno()
    {
        Console.WriteLine($"Hodnota atributu Jmeno: {Jmeno}");
    }
}
```

Proměnné

Proměnné sdružují hodnoty dat. Existuje mnoho typů, například:

- int - celočíselný datový typ
- float - ukládájí negativní a kladné čísla s desetinnou čárkou
- char - 1 písmeno, “a” nebo “A”
- string - text ohraničený dvojitými uvozovkami, “Hello world”
- bool - 2 stavy, pravda / nepravda (2)

```
using System;


class Program
{
    static void Main()
    {
        int cislo = 10;
        float cisloSDesetinnou = 3.14f;
        char znak = 'a';
        string retezec = "Hello world";
        bool pravdaNeboNepravda = true;


        Console.WriteLine("Cislo: " + cislo);
        Console.WriteLine("Cislo s desetinnou: " +
cisloSDesetinnou);
        Console.WriteLine("Znak: " + znak);
        Console.WriteLine("Retezec: " + retezec);
        Console.WriteLine("Pravda nebo nepravda: " +
pravdaNeboNepravda);
    }
}
```

Praktická část

Game Manager

Vývoj herního prostředí ve hře vytvořené v Unity zahrnuje mnoho důležitých prvků, z nichž jeden z nejzásadnějších je manažer hry (Game Manager). Herní manažer je klíčovým prvkem, který řídí a koordinuje celkový chod hry a interakci mezi různými herními funkcemi. Game Manager obsahuje různé části kódu, jelikož je dokáže realizovat v herním prostředí. Nese například různé proměné, jako měnu, životy a další údaje, které ovlivňují další prvky ve hře.

```
    public List<Sprite> playerSprites;
    public List<Sprite> weaponSprites;
    public List<int> weaponPrices;
    public List<int> xpTable;
    public int pesos;
    public int experience;
```

Ukládání stavu hry

Metoda „SaveState“ slouží k uložení aktuálního stavu hry. Nejprve se vytvoří řetězec obsahující data k uložení, jako jsou měna hráče, zkušenosti a úroveň zbraně. Poté se tento řetězec uloží do „PlayerPrefs“, což je mechanismus v Unity pro ukládání trvalých dat.

```
public void SaveState()
{
    string s = "";
    s += "0" + "|";
    s += pesos.ToString() + "|";
    s += experience.ToString() + "|";
    s += weapon.weaponLevel.ToString();
```

```
    PlayerPrefs.SetString("SaveState", s);
}
```

Funkce „LoadState“ je volána při načtení nové scény a slouží k načtení uloženého stavu hry. Nejprve se kontroluje, zda existuje uložený stav hry. Pokud ne, metoda končí. Pokud ano, uložený řetězec „PlayerPrefs“ se rozdělí na jednotlivé části a na základě těchto dat se aktualizuje měna hráče, zkušenosti a úroveň zbraně.

```
public void LoadState(Scene s,LoadSceneMode mode)
{
    SceneManager.sceneLoaded -= LoadState;

    if(!PlayerPrefs.HasKey("SaveState"))
        return;

    string[] data =
PlayerPrefs.GetString("SaveState").Split('|');

    pesos = int.Parse(data[1]);
    experience = int.Parse(data[2]);
    if(GetCurrentLevel() != 1)
        player.SetLevel(GetCurrentLevel());
    weapon.SetWeaponLevel(int.Parse(data[3]));
}
```

Fyzika prostředí a postav

Následující skripty hrají klíčovou roli k přirozenému fungování hráče při interakci s okolím a vytvoření poutavého herního zážitku. Slouží jako stavební kámen pro složitější úkoly.

Pohyb

Tato sekce popisuje implementaci mechanismu pohybu hráče ve 2D herním prostoru ve vývojovém prostředí Unity. Program získá z Unity předurčené klávesy pro pohyb horizontální a vertikální. Tradičně se používají klávesy “w”, “a” pro osu x a “s”, “d” pro osu y. Princip spočívá na počítání rozdílu velikosti vektoru v stávající poloze hráče a vstupu uživatele. Pokud objekt s komponentem „Box Collider 2D“ přijde ke kolizi s předměty na vrstvě “Layer” a “Blocking” nemůže projít dále. Tato funkce přidává možnost “zdí”, které ohraničují hrací pole. Pokud není detekována kolize, provede se pohyb hráče v dané ose.

```
private void FixedUpdate()
    {
        float x = Input.GetAxisRaw("Horizontal");
        float y = Input.GetAxisRaw("Vertical");
moveDelta = new Vector3(x, y, 0);
hit = Physics2D.BoxCast(transform.position, boxCollider.size, 0,
new Vector2(0, moveDelta.y), Mathf.Abs(moveDelta.y *
Time.deltaTime), LayerMask.GetMask("Actor", "Blocking"));
        if (hit.collider == null)
        {
           transform.Translate(0, moveDelta.y * Time.deltaTime,
0);
        }

        hit = Physics2D.BoxCast(transform.position,
boxCollider.size, 0, new Vector2(moveDelta.x, 0),
Mathf.Abs(moveDelta.x * Time.deltaTime),
LayerMask.GetMask("Actor", "Blocking"));
        if (hit.collider == null)
        {
            transform.Translate(moveDelta.x * Time.deltaTime, 0,
0);
```

```
}
```

Camera motor

Tento skript hýbe s kamerou podle pohybu hráče. Počítá rozdíl mezi aktuální pozicí kamery a pozicí hráče. Zkontroluje, zda rozdíl na ose X překročil stanovené omezení a zjistí, na které straně hráče je kamera na ose. Vypočítá „delta.x“ a stejným způsobem i „delta.y“ a posune podle této hodnoty kameru.

```
private void LateUpdate()
{
    Vector3 delta = Vector3.zero;

    float deltaX = lookAt.position.x - transform.position.x;
    if (deltaX > boundX || deltaX < -boundX)
    {
        if(transform.position.x < lookAt.position.x)
        {
         delta.x = deltaX - boundX;
        }
        else
        {
         delta.x = deltaX + boundX;
        }
    }

        float deltaY = lookAt.position.y - transform.position.y;
    if (deltaY > boundY || deltaY < -boundY)
    {
        if(transform.position.y < lookAt.position.y)
        {
         delta.y = deltaY - boundY;
```

```
        }
        else
        {
         delta.y = deltaY + boundY;
        }
    }
    transform.position += new Vector3(delta.x, delta.y, 0);
}
}
```

Kolize

Kolize slouží pro vymezení prostoru, ale i pro interakci s uživatelem. Při kolizi dvou komponentů „Box Collider 2d“ může vzniknout několik různých výsledku, které ovlivní průběh hry. Hlavní funkce je změna stavu, například grafická nebo aktualizace několik proměnných, které ukazují předměty ve vlastnictví hráče, úspěšnost, či vlastnosti uživatele.

Při začátku systém se ujistí, že objekt obsahuje komponent „BoxCollider2D“, bez kterého funkce nemůže pokračovat. Komponent „BoxCollider2D“ je neviditelný box na objektu ve hře, který definuje plochu okolo něho. Speciální funkce „OverlapCollider“ hlédá další objekty, se kterými se BoxCollider2D překrývá. Pokud tato funkce nenajde nic, děj pokračuje. Pokud je s nějakým jiným objektem v kolizi nastane funkce „protected virtual void OnCollide“. Tato funkce je definována třídou „protected virtual void“, což znamená, že ji můžou užívat další skripty, které z ni dědí a můžou výsledek této funkce přepsat. Díky této vlastnosti můžeme dědičným skriptům udávat jiné schopnosti, jako například teleportaci nebo změna stavu při kolizi.

```
    protected virtual void Start()
    {
        boxCollider = GetComponent<BoxCollider2D>();
    }
    protected virtual void Update()
    {
```

```
        boxCollider.OverlapCollider(filter, hits);
        for (int i = 0; i < hits.Length; i ++)
        {
            //jestli objekt nekoliduje s ničím tak ...
            if (hits[i] == null)
                continue;
            OnCollide(hits[i]);
            hits[i] = null;
        }
    }
    protected virtual void OnCollide(Collider2D coll)
    {
        Debug.Log(coll.name);
    }
```

Příklad dědičného skriptu kolize je teleportér. Při kolizi přepisuje funkci „OnCollide“ vytváří podmínku, jestli objekt, se kterým je teleportér v kolizi, nese jméno „Player“, načte správce hry jednu ze scén z množiny se jménem „sceneNames“ a načte ji. Nová scéna reprezentuje další žalář, do kterého se hráč dostane při splnění úkolu v předešlém.

```
using UnityEngine;
public class teleporter : collidable
{
    public string[] sceneNames;
  protected override void OnCollide(Collider2D coll)
    {
        if(coll.name == "Player")
        {
            //teleportace hráče, vebere náhodnou scénu v array ( poli) stringu sceneName(s)
```

```
            GameManager.instance.SaveState();
                string sceneName = sceneNames[Random.Range(0,
sceneNames.Length)];
            //loaduje scénu
             UnityEngine.SceneManagement.SceneManager.LoadScene(s
ceneName);
        }
    }
}
```

Další skript je pro truhlu, jejíž funkce je je při kolizi darovat hráči měnu. Při setkání s hráčem se stav změní na „vybraná“ a změní se ilustrace truhly na prázdnou. Přičte se měna v správce hry a hráči se ukáže text, že odměnu opravdu získal.

```
public class chest : seber
{
public Sprite emptyChest;
public int pesosAmount = 5;
    protected override void OnCollect()
        if(!collected)
        {
            collected = true;
            GetComponent<SpriteRenderer>().sprite = emptyChest;
            GameManager.instance.pesos += pesosAmount;
             GameManager.instance.ShowText("+" + pesosAmount + "
imunity!", 25, Color.white, transform.position, Vector3.up * 20,
3.0f);
        }
```

```
    }
}
```

Zápasy s protivníkem

Pro funkční zápasy potřebujeme zbraň, hráče, nepřítele a funkci životů. Zbraň dědí ze skriptu kolize a při útoku objektu se štítkem „fighter“ a jiným jménem než „Player“ posílá zprávu objektu, se kterým je zbraň v kolizi, že si má ubrat životy.

```
protected override void OnCollide(Collider2D coll)
    {
        if (coll.tag == "fighter")
        {
            if (coll.name == "Player")
            return;

        Damage dmg = new Damage()
        {
           damageAmount = damagePoint[weaponLevel],
           origin = transform.position,
           pushForce = pushForce[weaponLevel]
        };
        coll.SendMessage("ReceiveDamage", dmg);
```

Funkce „ReceiveDamage“ ukladá metody které se mají provést při kolizi. Aby nemohl hráč nebo nepřítel být zraněn několikrát za sekundu, je zde vytvořena podmínka, která počítá rozdíl mezi

poslední a aktuální kolizí objektu se zbraní a porovnává, jestli je větší než předem určený časový úsek. Pokud podmínka platí, resetuje se čas a objektu se odebere zdraví. Pokud má objekt méně nebo rovno nuly bodů zdraví, nastává funkce „Death()“, která značí smrt.

```
protected virtual void ReceiveDamage(Damage dmg)

{

    //počítá jestli můžeme dostat hit

    if(Time.time - lastImmune > immuneTime)

    {

      lastImmune = Time.time;

      hitpoint -= dmg.damageAmount;
     pushDirection = (transform.position - dmg.origin).normalized
* dmg.pushForce;

       GameManager.instance.ShowText(dmg.damageAmount.ToString(),
25, Color.red, transform.position, Vector3.zero, 0.5f);



     if(hitpoint <=0)

    {

        hitpoint = 0;

        Death();

    }

    }

}
```

Pro ztížení hry se musí limitovat, jak často může hráč udeřit zbraní. Při stisknutí levého tlačítka myši skript vypočítá, jestli rozdíl aktuálního času a času minulého stisknutí myši je větší než čas do opětovného použití, který jsme si definovali na začátku. Pokud podmínka platí, hráč může udeřit.

```
protected override void Update()
 {
     base.Update();
     if(Input.GetMouseButtonDown(0))
     {
         if(Time.time - lastSwing > cooldown)
         {
             lastSwing = Time.time;
             Swing();
         }
     }
 }
```

Nepřítel se pohybuje jiným způsobem než hráč. Pro hráče je vstup w, a, s, d, ale pro nepřítele je vstup samotná lokace hráče. Definujeme si vzdálenost, ve které si nás vir všimne a další vzdálenost ve které nás protivník nahání. Pokud je vir ve stejné lokaci jako hráč, nastává chyba, kdy se nepřítel stále snaží dostat k naší lokaci i když v ní již je. Proto existuje podmínka, kdy se stav „chasing“ nastaví jako „false“ pokud má vir i hráč stejné souřadnice. Bool „chasing“ se zapíná, pokud je hráč v takové blízkosti viru, ve které ho začíná pronásledovat.

```
private void FixedUpdate()
    {
     if(Vector3.Distance(playerTransform.position,
startingPosition) < chaseLenght)
        {
     if(Vector3.Distance(playerTransform.position,
startingPosition) < triggerLenght)
```

```
                {
                  chasing = true;
                }
                if (chasing)
                {
                 if (!collidingWithPlayer)
                 {
      UpdateMotor((playerTransform.position
transform.position).normalized);
                 }
                }
        }
```

Plovoucí text

Pro zlepšení komunikace s hráčem prostřednictvím videohry a efektivnějšího sdělování informací jsem vyvinula funkci plovoucího textu. Tato funkce umožňuje zobrazení textu na obrazovce s možností úpravy různých parametrů, včetně fontu, velikosti, doby zobrazení, barev a dalších vlastností. Pro aktivaci této funkce stačí provést jednoduchý vstup, obvykle v reakci na určitou událost, jako je například kolize herních objektů.

Nejdříve definujeme funkci „Show()“ a „Hide()“. Při první funkci se text nastaví stav na aktivní a při druhé na neaktivní. Dále metoda „UpdateFloatingText()“ aktualizuje plovoucí text na obrazovce v každém snímku hry. Nejprve kontroluje, zda je plovoucí text aktivní. Pokud není, funkce končí. Poté kontroluje, zda uběhla doba trvání zobrazení textu od jeho posledního zobrazení. Pokud ano, plovoucí text je skryt voláním metody „Hide()“. Nakonec aktualizuje pozici plovoucího textu na obrazovce pomocí vektoru pohybu a času, který uplynul od poslední aktualizace.

```
public void Show()
    {
        active = true;
        lastShown = Time.time;
        go.SetActive(active);

    }

    public void Hide()
        {
        active = false;
        go.SetActive(active);
        }

    public void UpdateFloatingText()
    {
        if (!active)
            return;
        if (Time.time - lastShown > duration)
            Hide();

        go.transform.position += motion * Time.deltaTime;
    }
```

Tato funkce slouží k zobrazení nového plovoucího textu na obrazovce. Přijímá různé parametry, jako je zpráva „msg“, velikost písma „fontsize“, barva textu „color“, pozice na obrazovce „position“, pohyb textu „motion“ a doba trvání zobrazení „duration“.

```
public void Show(string msg, int fontSize, Color color, Vector3
position, Vector3 motion, float duration)
{
    FloatingText floatingText = GetFloatingText();
```

```
    floatingText.txt.text = msg;
    floatingText.txt.fontSize = fontSize;
    floatingText.txt.color = color;
floatingText.go.transform.position                                    =
Camera.main.WorldToScreenPoint(position);
    floatingText.motion = motion;
    floatingText.duration = duration;
```

Při funkci „Show()“ vytváří novou instanci plovoucího textu a nastavuje mu uvedené parametry.

```
    floatingText.Show();
}
private FloatingText GetFloatingText()
    {
        FloatingText txt = floatingTexts.Find(t => !t.active);

        if(txt == null)
        {
        txt = new FloatingText();
        txt.go = Instantiate(textPrefab);
        txt.go.transform.SetParent(textContainer.transform);
        txt.txt = txt.go.GetComponent<TMPro.TextMeshProUGUI>();

        floatingTexts.Add(txt);
        }
        return txt;
    }
}
```

Pomocí funkce plovoucího textu jsem vytvořila nehratelné postavy NPC, které uživatele provázejí videohrou. Skript nabízí množinu textu, kterou podle pořadí pomocí funkce indexů přehrává. Pro snadnou čitelnost je zde doba „cooldown“, na kterou zůstane text na obrazovce, než se zobrazí další v množině.

```
public class Npcsprechen : collidable
{
    public string[] messages;
    public float cooldown = 4.0f;
    private float lastShout;
    private int currentIndex = 0;

    protected override void Start()
    {
        base.Start();
        lastShout = -cooldown;
    }

    protected override void OnCollide(Collider2D coll)
    {
        if (Time.time - lastShout > cooldown)
        {
            lastShout = Time.time;
GameManager.instance.ShowText(messages[currentIndex], 25, Color.white, transform.position + new Vector3(0, 0.18f, 0), Vector3.zero, cooldown);
            currentIndex = (currentIndex + 1) % messages.Length;
        }
    }
}
```

Uživatelské rozhraní

Uživatelské rozhraní (UI) slouží k vytvoření a zobrazení různých prvků, které umožňují interakci hráče s hrou a poskytují mu potřebné informace.

Menu

Menu zobrazuje hráči zkušenosti, jeho aktuální zdraví, úroveň zbraně, počet měny a také poskytuje hráči možnost si koupit novou zbraň. Slouží k zorientování hráče, jak úspěšný je ve své videohře. V Unity jsem si pomocí elementů uživatelského rozhraní vytvořila kostru mého menu. Pomocí skriptu poté můžu s textovými poli pracovat jako s proměnnými, jejichž hodnota se bude měnit podle úspěšnosti hráče. Na začátku skriptu si definujeme veřejné třídy textového typu a obrázkového. Manažer hry nese všechny informace, ke kterým má skript menu přístup. Pokud se tedy změní nějaká informace v manažeru hry, menu se obnoví a ukáže nové informace.

Obrázek 2 - Kostra menu

Na začátek pomocí manažeru hry zjistíme aktuální úroveň hráčovi zbraně. Pokud je úroveň zbraně stejná jako počet zbraní v manažeru, napíše se místo textu ceny na tlačítku objeví „MAX“. Toto naznačuje uživateli, že již není další zbraň pro další nákup. Jinak zobrazí cenu zbraně podle pořadí v herním manažeru. Dále jsou proměnné ve formátu celých čísel, přeměněny na text, aby mohli být použité v menu.

```
public class Charactermenu : MonoBehaviour
{
                                        weaponSprite.sprite        =
GameManager.instance.weaponSprites[GameManager.instance.weapon.w
eaponLevel];
                  if(GameManager.instance.weapon.weaponLevel    ==
GameManager.instance.weaponPrices.Count)
            upgradeCostText.text = "MAX";
        else
                                          upgradeCostText.text      =
GameManager.instance.weaponPrices[GameManager.instance.weapon.we
aponLevel].ToString();
levelText.text                                                      =
GameManager.instance.GetCurrentLevel().ToString();
hitpointText.text                                                   =
GameManager.instance.player.hitpoint.ToString();
pesosText.text = GameManager.instance.pesos.ToString();
int currLevel = GameManager.instance.GetCurrentLevel();
```

Obrazovka při smrti

Při funkci „OnDeath()“ manažer hry spouští animaci, která ukáže obrazovku při smrti, neboli zneškodnění. Menší menu nabízí tlačítko s textem „Uzdravit se“. Při stisknutí se animace obrazovky schová a pomocí manažeru scén se načte úvodní místnost, jak je napsané „Respawn()“ funkci.

Obrázek 3 - Menu při smrti

```
    protected override void Death()
    {
        isAlive = false;
        GameManager.instance.deathMenuAnim.SetTrigger("Show");
    }
public void Respawn()
{
    deathMenuAnim.SetTrigger("Hide");
    UnityEngine.SceneManagement.SceneManager.LoadScene("Main");
    player.Respawn();
}
```

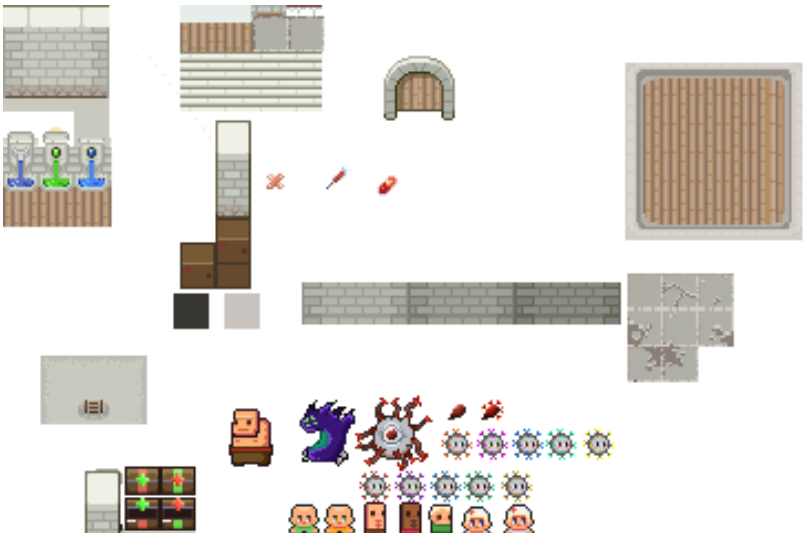
Estetika

Následující funkce zkrášlují mou video hru a slouží také jako nástroj pro lepší pochopitelnost a hratelnost.

Ilustrace

Pro zdokonalení estetiky a dynamizace videoher jsou klíčové prvky jako kvalitní ilustrace a animace. V mé práci jsem využila program Clip Studio Paint, který poskytuje široké možnosti pro pixelové umění ve vysoké kvalitě. Díky jeho funkcím pro práci s vrstvami a flexibilnímu měnění rozlišení jsem mohla efektivně upravovat grafiku a přizpůsobovat ji podle potřeby, a to s ohledem na všechny estetické a vizuální požadavky. V Unity je k dispozici funkce Sprite Editor, která umožňuje snadné vystřihování jednotlivých obrázků z velkého atlasu ilustrací.

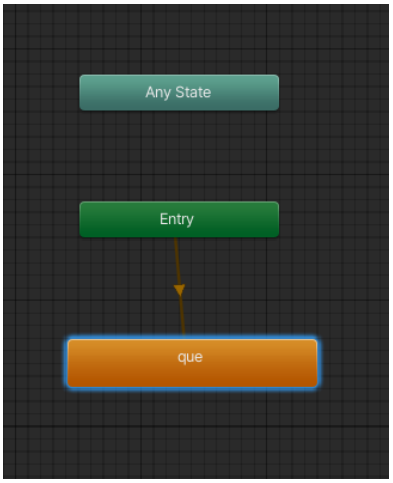
Tato funkce usnadňuje práci s grafikou, jelikož nám umožňuje efektivně vytvářet a upravovat sprity podle potřeby, což přispívá k lepšímu vizuálnímu zpracování našich her. Pro tvorbu prostředí jsem využila atlas, z něhož jsem čerpala většinu ilustrací. Většina spritů je ve velikosti 16x16 pixelů, což usnadňuje formátování a zachování jednotného vzhledu. Dále jsem využila 2D Tileset, která mi umožňuje pracovat s vyřezanými sprity jako s paletou. To mi poskytuje možnost rychle kopírovat vzory a snadno vyplňovat určité oblasti prostředí svým motivem.



Obrázek 4 - atlas ilustrací

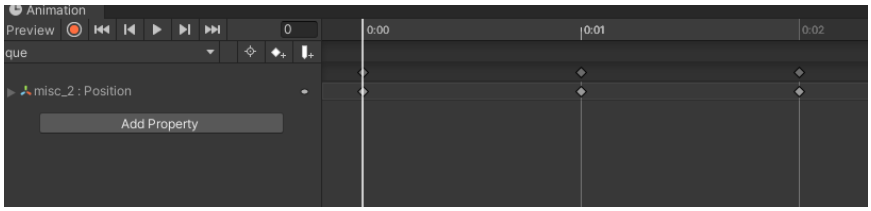
Animace

V Unity se animace vytvářejí pomocí animačního ovládacího panelu a krátkých animačních klipů. Animační kontrolér určuje vstupní událost, při které se animace spustí, například stisknutí myši pro útok, stisknutí tlačítka nebo automatické spuštění animace při načtení scény.

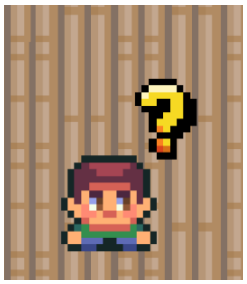


Obrázek 5 - Ovladač animací

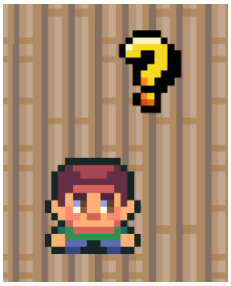
Krátké animační klipy jsou vždy přiřazeny konkrétnímu objektu ve scéně. Například může jít o animaci, při které se otazník zvyšuje a snižuje na ose Y. Tímto způsobem lze vytvořit animaci "létajícího" otazníku. V mé hře jsem tuto animaci využila k přilákání hráče. Animace se často používají i u vysvětlivek nebo u NPC postav. Dále jsou animace využívány k oživení hry; například virusy mohou problikávat, aby byly výraznější vůči pozadí a snadněji rozpoznatelné.



Obrázek 6 - Úsek animace



Obrázek 8 - Animace 1.



Obrázek 7 - Animace 2

Průběh hry

Videohru jsem rozdělila do několika částí, aby hráče postupně seznámily s fundamentálními herními možnostmi a postupně jim představovaly základní mechaniky a prostředí hry. Hra začíná jednoduchým příběhem, který hráče vtáhne do děje. Tento úvodní segment slouží k tomu, aby hráči získali povědomí o tom, jak hra funguje, a aby se s ní snadno seznámili a mohli pokračovat do náročnějších částí. Dále jsou konfrontováni s různými hádankami a souboji, které postupně zvyšují obtížnost hry. Nakonec hráči dosáhnou odměněného konce, kde se jejich úsilí a dovednosti vyplatí.

Obrázek 9 - Font Asset Creator

Tutoriál

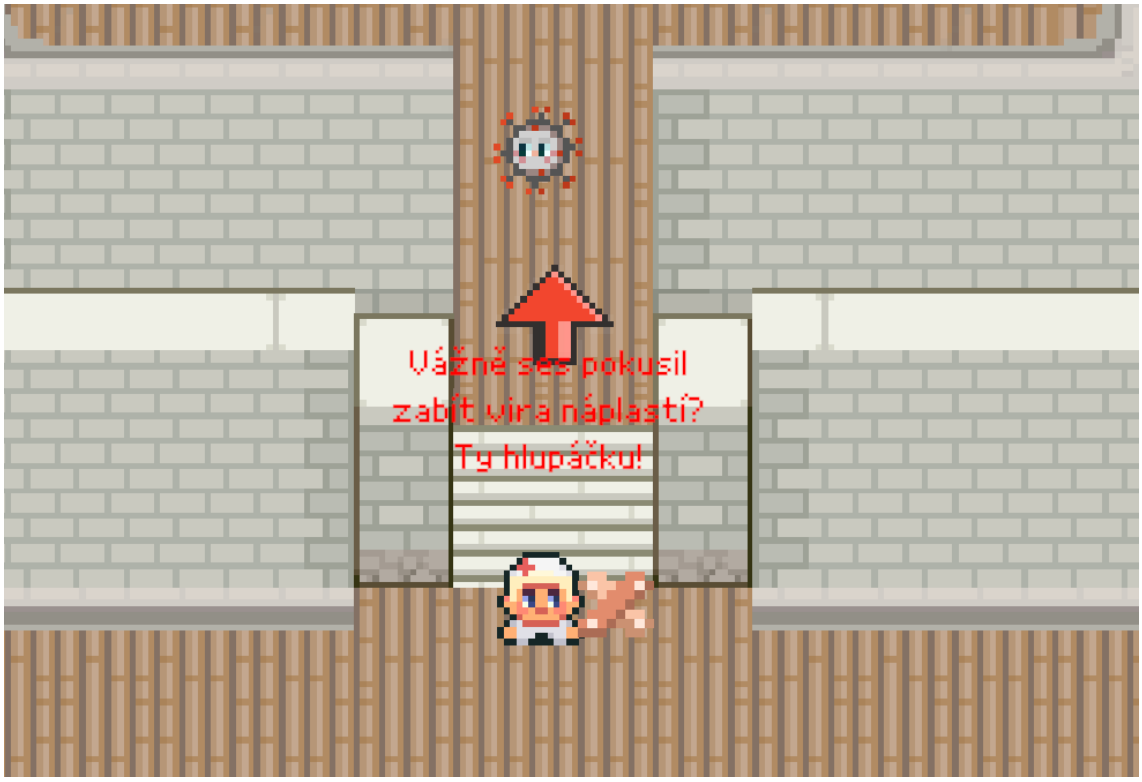
Tutoriál jsem rozdělila do tří segmentů, pro snazší pochopitelnost pro hráče. První část seznamuje hráče se základním ovládáním, čili jak se hráč má pohybovat a útočit. Pro pohyb jsem zvolila klávesové písmena w, a, s, d, které jsou typické pro většinu her. Stejně tak běžně

používané je levé stisknutí myši pro útok. Dále průvodce nám udává úkol vyzkoušet tři funkce – rozbíjení boxů, uzdravování se, a vybírání truhly.

Obrázek 10 - Tutoriál 1. část

Obrázek 11 - Tutoriál 2. část

Hráče dále seznamujeme s mechanismy získávání lepších zbraní prostřednictvím sběru měny a různých typů nástrojů. Textové instrukce hráčům vysvětlují, že strategické sbírání herní měny a nalezení různých typů nástrojů je klíčem k vylepšování jejich výzbroje.



Obrázek 12 - Tutoriál 3. část

Ve třetím segmentu se uživatel dostane ke svému prvnímu viru. Při souboji s nepřítelem hráče informujeme, že si má špatnou zbraň, což je zároveň vtipným způsobem naznačení, že virus nelze porazit pouhou náplastí, ale jen účinnými léčivy, které jsou představeny v druhé části tutoriálu. Toto hráči ukazujeme, aby si uvědomil význam správné přípravy pro porážení protivníků. Jediným způsobem, jak opustit tuto část hry, je nechat se zneškodnit, což hráči videohra napovídá. Následně se hráč ocitne ve výchozí místnosti, kde začíná další dobrodružství.

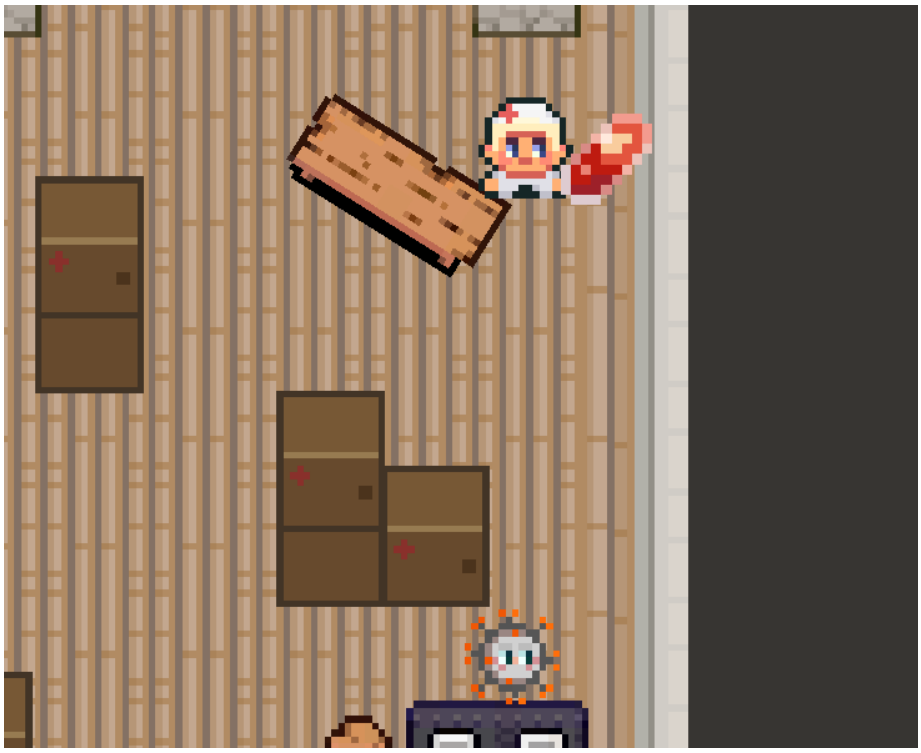
Žalář

Při vchodu do žaláře je hráči připomenuto si zakoupit lepší zbraň, jelikož ho dále čekají větší nebezpečí, na které je potřeba účinná zbraň. Současně je uživateli sdělen krátký příběh, o tom, co se stalo v nemocnici, kterou má hráč zachránit. Postupně se v dalších částech nemocnice hráč dozvídá o skutečných následcích této katastrofy.

Obrázek 13 - Žalář - NPC

Obrázek 14 - Žalář - příběh

Hráč postupně prochází místnostmi, s gradující obtížností. Viry, se kterými se hráč setkává v pozdějších fázích, jsou rychlejší, silnější a mají delší životnost. Hráč je připraven díky svým silnějším zbraním, které mu umožňují efektivněji zneškodňovat tyto silnější viry. Tento postupný nárůst obtížnosti dodává hře dynamiku a udržuje zájem.



Obrázek 15 - Žalář - nepřítel

Obrázek 16 - Žalář - 2. nepřítel

Prvním z hlavních nepřátel, kterým hráč čelí, je virus s omezenou pohyblivostí. Tento protivník není schopen pohybu, avšak vytváří kolem sebe rotující formace ničivých bublin, které se

pohybují v kruzích s variabilní rychlostí. Tento dynamický prvek představuje pro hráče výzvu, neboť ztížená manévrovatelnost těchto bublin vyžaduje zvýšenou pozornost a zručnost hráče při úniku či eliminaci této hrozby.

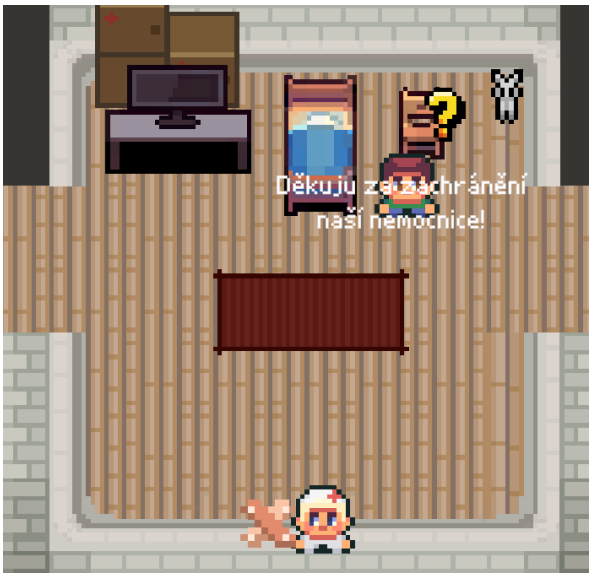
Obrázek 17 - Žalář - 1. hlavní nepřítel

Po dalším objevování nemocnice se hráč setká s posledním hlavním nepřítelem, který se objevuje těsně před koncem hry. Je na rozdíl od předešlého schopný pohybu a je obklopen rotačními škodlivými bublinami. Tento nepřítel představuje nejtěžší výzvu, kterou hráč musí překonat, aby dokončil videohru.

Obrázek 18 - Žalář - 2. hlavní nepřítel

Konec

Po úspěšném dokončení videohry a zachránění nemocnice, je hráči představena závěrečná odměna. Sleduje obnovenou a funkční podobu nemocnice, kterou bránil před viry. Tento závěr je navržen tak, aby uživateli poskytl dobrý pocit po splnění poslání. Obnově nemocnice je věnována zvláštní pozornost, aby hráč pochopil význam své práce a byl odměněn za své úsilí.

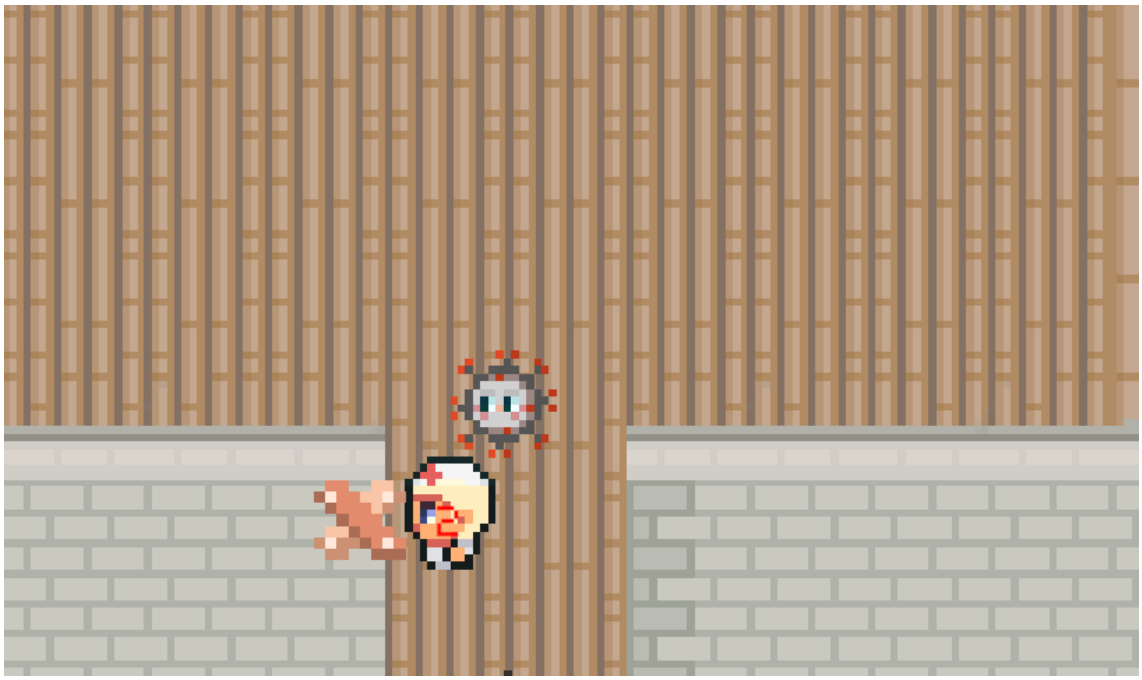


Obrázek 20 - Konec - zachráněná nemocnice

Obrázek 19 - Konec - zachráněná nemocnice 2

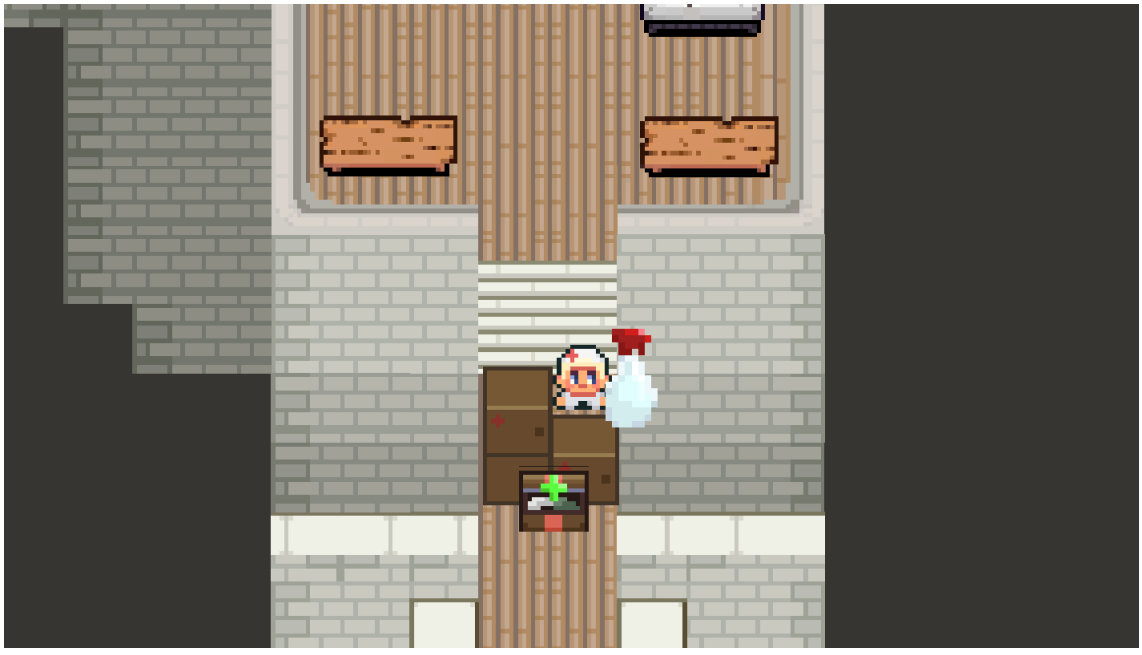
Výsledky

Obrázek 21 - Výsledek - Tutoriál



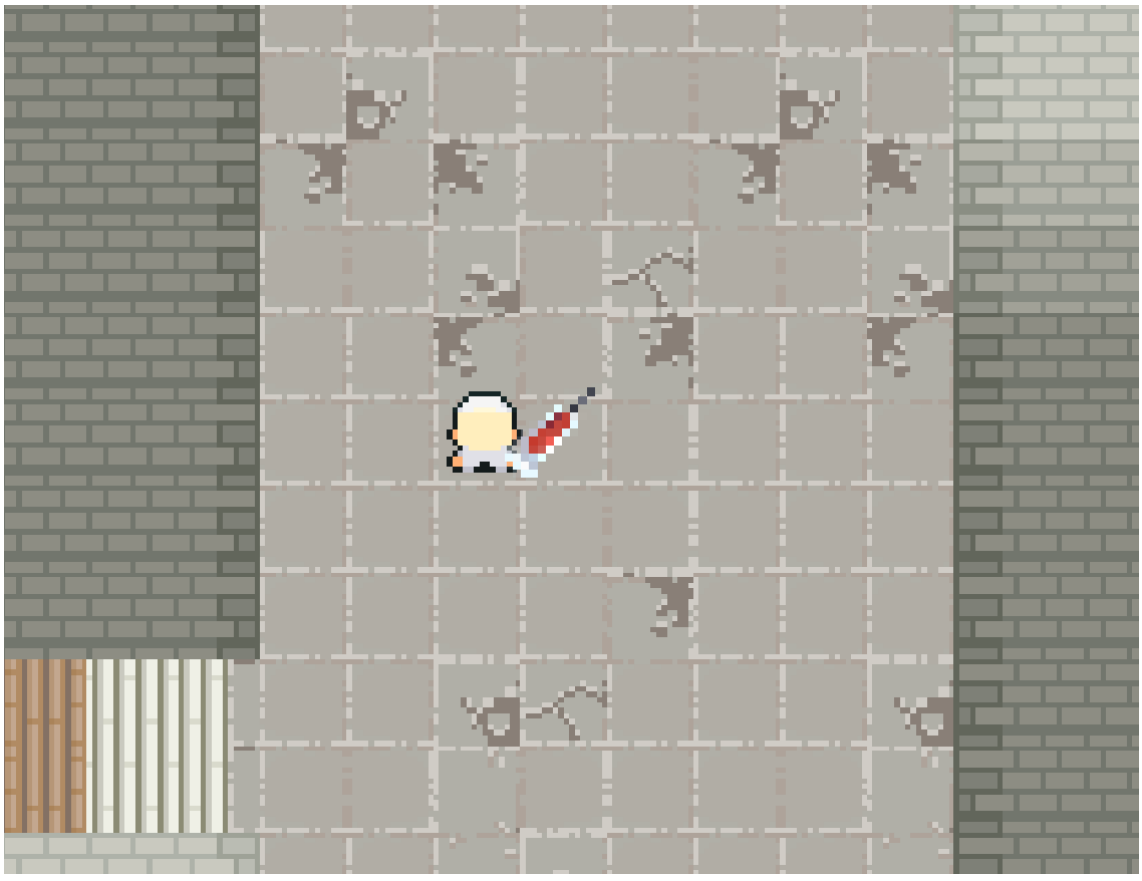
Obrázek 22 - Výsledek - Útok nepřítele

Obrázek 23 - Výsledek - Uzdravovací umyvadlo

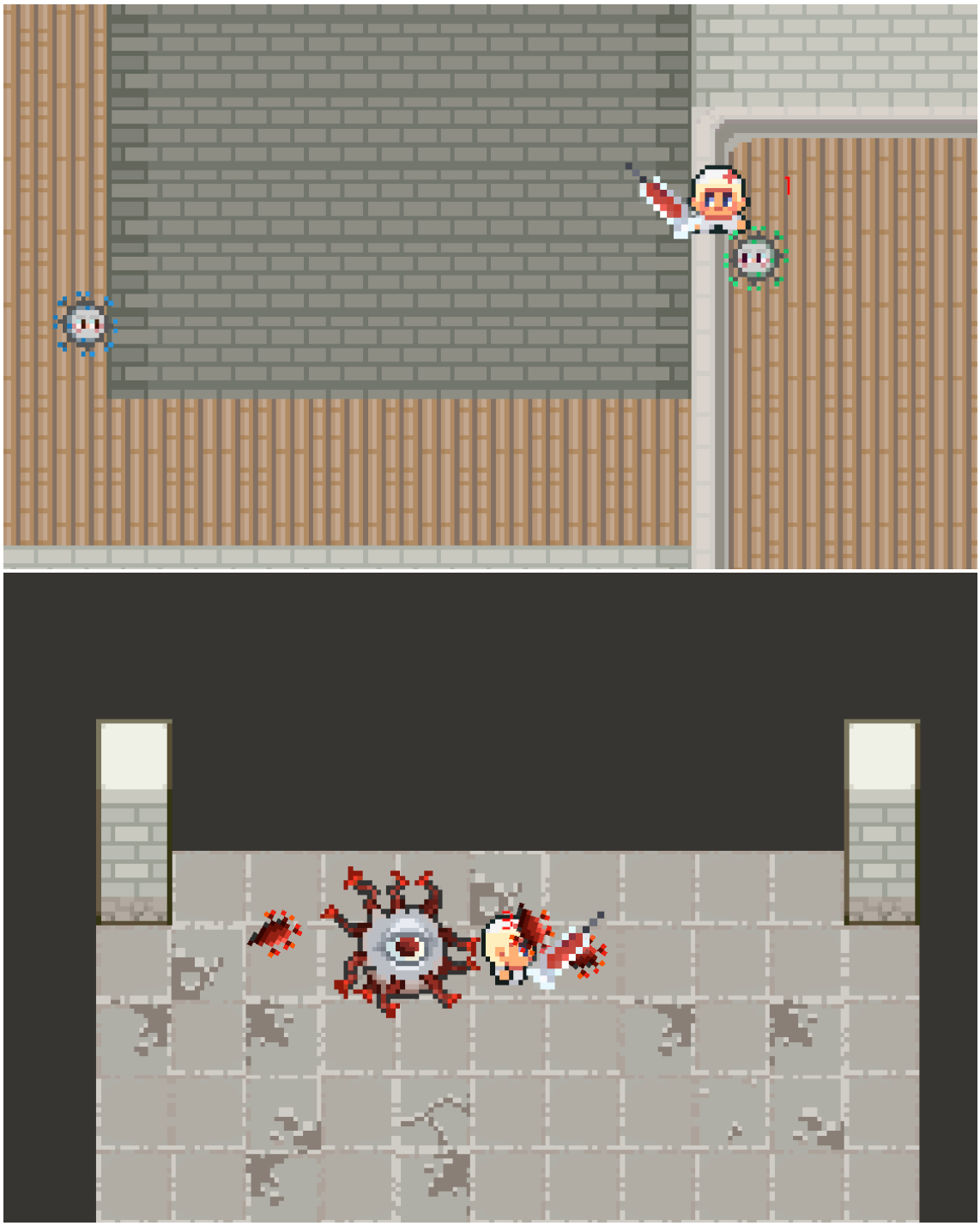


Obrázek 24 - Výsledek - Překážka

Obrázek 25 - Výsledek - Boj s hlavním nepřítelem



Obrázek 26 - Výsledek - Neviditelné bludiště



Obrázek 27 - Výsledek - Boj s hlavním nepřítelem 2.

Obrázek 28 - Výsledek - Závěr hry

Závěr

Práce na tomto projektu mi poskytla cenné zkušenosti a dovednosti v oblasti herního vývoje. Díky vývoji mé RPG pixelové videohry v Unity jsem se naučila základy programování v jazyce C#, ale také plně ovládla nástroje a techniky v aplikaci Unity Engine. Tento projekt mi umožnil rozvinout svou kreativitu a zdokonalit dovednosti v digitálním umění v oblasti Clip Studio Paint.

Citovaná literatura

1. Code name: UNICORN. Making video games shine. Brno, 2019. Bakalářská práce.

2. C# Tutorial. *W3Schools* [online]. 2024 [cit. 2024-03-21]. Dostupné z: https://www.w3schools.com/cs/index.php

3. **C Sharp**. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2024, 2001-2024 [cit. 2024-03-21]. Dostupné z: https://cs.wikipedia.org/wiki/C_Sharp#C%C3%ADle_jazyka

4. UNITY. Unity - Documentation. Unity - Manual [online]. [cit. 2024-04-02]. Dostupné z: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

5. [online]. In: . [cit. 2024-04-02]. Dostupné z: https://www.google.com/url?sa=i&url=https%3A%2F%2Fjakserychlenaucit.cz%2Fjak-se-nejlepe-ucit-pyramida-zapamatovani%2F&psig=AOvVaw31mHKB3rMiRpoojQCWlxX_&ust=1706435249292000&source=images&cd=vfe&opi=89978449&ved=0CAUQjB1qFwoTCPjqiZKl_YMDFQAAAAAdAAAAABAD