STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST
Obor SOČ: M18

Mobilní aplikace – hra v Unity

Vypracoval: Róbert Kohout

Třída: IT4

Škola: Střední škola spojů a informatiky, Tábor, Bydlinského 2474

Kraj: Jihočeský

Konzultant: Ing. Dana Almášiová

Čestné prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil (a)jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ. Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V……dne……. Podpis……..

Obsah

Anotace

Cílem této práce bylo naprogramovat funkční mobilní hru v Unity, která je ovlivňována časem v našem světě. Přidat do ní funkce zlepšování a posouvání se kupředu, objevování nových lokalit a nepřátel.

Klíčová slova

Mobilní hra, Unity

Annotation

The goal of this work was to program a functional mobile game in Unity that is influenced by time in our world. Add to it the functions of improving and moving forward, discovering new locations and enemies.

Keywords

Mobile game, Unity

Úvod

Mým cílem bylo vypracovat mobilní hru, jednoduchou klikačku, která bude ovlivňována realným časem a podle toho se bude měnit i její samotná hratelnost, otevírat různé zákoutí a cesty, co prozkoumat. Hraní přes den i noci má své výhody a nevýhody, na které hráč musí přijít a využívat jich. Komponenty, co jsem implementoval, a jsou důležité pro hratelnost, je jím nastavení reálného času, jeho zpracování a vložení do hry. Další je automatické generování nepřátel. Automatické vyhledání nepřátel, následné zabití, spuštění animace a přičtení peněz hráči. Přenášení dat hráče mezi jednotlivými scénami, aby neustále nemizely. Ukládání a načítání správných dat a progresem hráče. Vytvoření NPC (Non-Player Character), naprogramování jeho schopností, dosazení jeho animací. Dopočítávání stáří a následné vyhodnocování NPC, které za nás v pozdější fázi hry zabíjí nepřátele a značně nám usnadňuje samotnou hru.

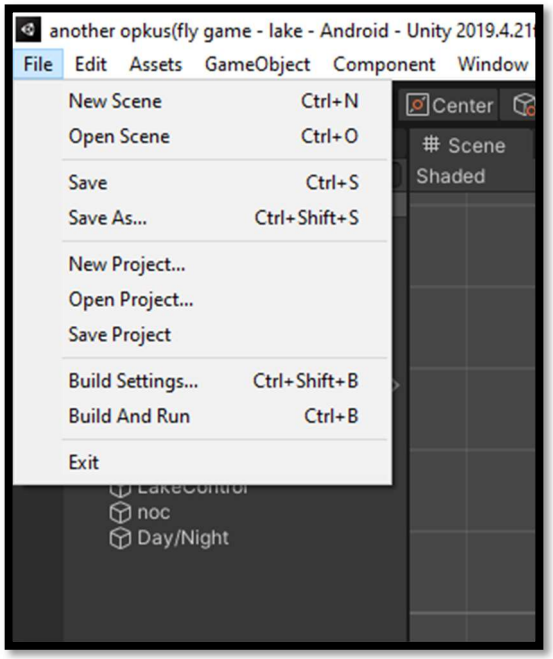
1. Specifika tvorby aplikace pro mobily

1. Budeme potřebovat samotný nápad a námět hry. To, jak by měla asi vypadat, nějaký žánr a nebo styl, jak by se mohla hrát a základní mechaniku, na které budeme stavět a rozšiřovat její možnosti. Moji volbou byla jednoduchá klikačka, náhodně na displeji se budou spawnouvat nepřátelé a hráč je pomocí kliku zabije, tím získá peníze a bude se moci vylepšovat, zabíjet více nepřátel, získá nové možnosti, více peněz a tak dále.

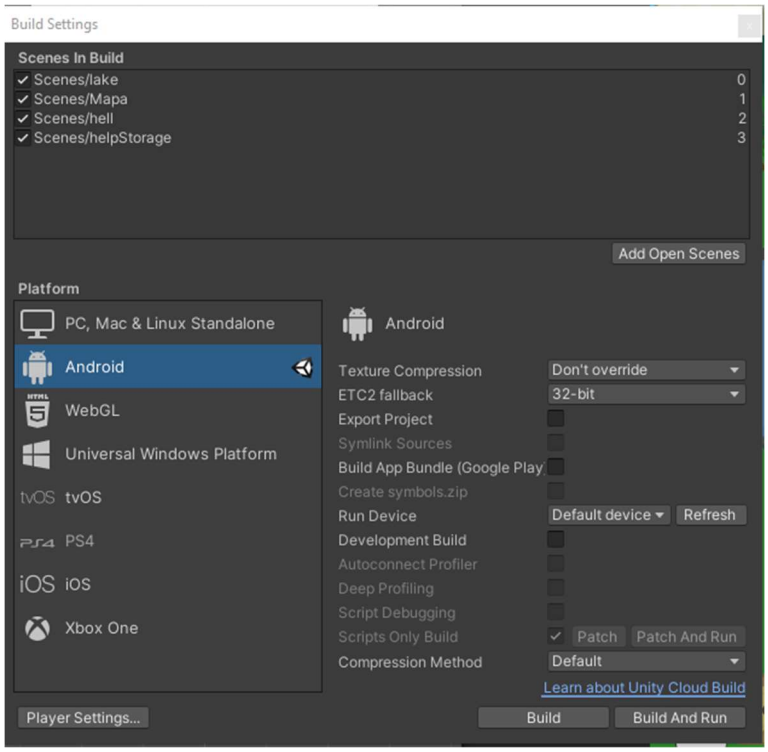
2. Budeme potřebovat nějaký herní engine. Možností máme opravdu hodně například Unity, Unreal Engine, Solar2D, Build box. Herní engine nám už takhle ušetří spousty práce. Ovšem nějaký čas nám zabere naučit se s ním pracovat. Já jsem si vybral Unity, protože je nejznámější, má mnoho, ale opravdu mnoho možností, bylo v něm například vytvořeno i Pokemon Go.

3. Musíme umět programovat. Jelikož jsem si vybral Unity, tak ten k programování používá programovací jazyk C# s využitím vlastních knihoven. To znamená, že když chcete využít celý potenciál tohoto enginu, tak je dobré se naučit i tyto knihovny a příkazy, protože pak krásně a úhledně spolupracují mezi sebou a člověk si může hodně ulehčit práci.

4. Když vytváříme mobilní hru, tak musíme si vybrat, na jaký OS ji budeme vytvářet, jestli na Android a nebo na iOS, rozdíly vytváření hry na tyto dva OS nejsou příliš velké, jde jen o drobnosti v kódu. Ovšem když se kód píše dobře, je možné to psát na oba OS najednou. Takže ve výsledku to není až tak podstatné. Já si vybral android, protože to je můj OS na mobilu. To mi umožňuje okamžitě provádět případné testy hry přímo v praxi.



Jakmile máme vybráno, na jaké OS budeme hru vytvářet, klikneme na File a Build Settings… Vyskočí nám okno, kde jsem si nastavil na Android. Na



pravé straně se nám ukáže text „No Android module loaded.“ a pod ním tlačítko Install with Unity Hub. Klikneme na tlačítko a nainstalujeme Android module. Když máme nainstalovaný Android modul, už nám nebrání nic v tom, abychom se pustili do vytváření naší samotné hry.

2. Postup tvorby aplikace a její instalace na mobil

2.1 Generace nepřátel

Začal jsem s tím, že jsem vytvořil script na automatickou generaci nepřátel.

```
if (pocet < Player.MaxSpawn)
{
    float x = Random.Range(0.1f, 0.8f);
    float y = Random.Range(0.35f, 0.9f);
    Vector3 pos = new Vector3(x, y, 10.4f);
    pos = Camera.main.ViewportToWorldPoint(pos);

    GenerateFlys(pos);
void GenerateFlys(Vector3 vec)
{
    if (Player.JeDen)
    {
        pocet += 1;
        Instantiate(fly, vec, Quaternion.identity);
        Debug.Log("Max = " + Player.MaxSpawn + "/ Počet = " + pocet);
```

V prvních dvou řádkách si deklaruji proměnou, kde ji následně inicializuji náhodně generovaným číslem v rozsahu od 0,1 až 0,8 na X hodnotě a 0,35 až 0,9 na Y hodnotě. Čísla 0,1 až 0,8 představují procentuální hodnoty na obrazovce, tyto hodnoty nesmí překročit číslo 1 a nebo menší než 0, protože by se pak už objekt objevoval mimo scénu a nebyl by vidět. Já tam má nastavení 0,1 kvůli velikosti objektu, aby nebyla část objektu mimo scénu a 0,8 je kvůli velikosti objektu a šířce postranního panelu na pravé straně. U hodnoty Y jsem nastavil hodnoty, které potřebuji, aby odpovídali mým potřebám a spawnutí nepřátelé nepřekáželi ve scéně. Na třetím řádku deklaruji hodnotu pos. Deklaroval jsem ji jako Vector3, což je vnitřní datový typ z knihoven Unity, ve které se ukládají 3 proměné. Tento datový typ složí k určování polohy objektu ve scéně. V Unity je ještě jeden takový datový typ a to je Vector2, rozdíl mezi nimi je ten, že Vector3 složí k určování polohy ve scéně u 3D hrách, jelikož má v sobě x, y, z a Vector2 slouží k určování polohy ve 2D hrách, neboli x a y já zde využívám Vector3 kvůli ručnímu nastavení vrstev a hodnoty x a y dosazuji z vygenerovaných čísel. Na dalším řádku děláme to, že vezmeme pozici a vložíme ji do Camera.main.ViewpoertToWorldPoint což nám udělá to, že podle zadaných hodnot, najde tu pozici na viditelné scéně a najde ten bod a vezme reálné pozice, které následně vrátí pozici x a y na té mapě. Hodnotu z to nedopočítává, protože by to tak či tak bylo na jednom bodě, proto to jen vrátí hodnoty x a y. Tyto hodnoty následně vrátíme zpět

do proměné pos a následně zavoláme metodu GenerateFlys, kde vložíme i naší připravenou pozici. V metodě si přičtu 1 k poctu již generovaných much, aby se mi nestalo to, že by se mouchy generovali do nekonečna a nezakryli tak kompletně celou obrazovku. Pomocí příkazu Instantiate(GameObject, position, rotation), kde jako první hodnotu vkládáme datový typ GameObject, což je další datový typ, které Unity přidává. Zde jsem si tedy vložil mého nepřítele. Toho jsem tam vložil pomocí příkazu před deklarací proměné a to je příkaz public. Jakmile dáme tento příkaz před deklaraci, `public GameObject fly;` tak se nám v Unity ukáže kolonka u Scriptu, kde můžeme následně přetáhnout náš object, který chceme spawnouvat. Jako druhé se zde vkládá position, což jsou pozice, na kterých se má náš nepřítel spawnout. Jako třetí jsem si vložil rotation což je rotace nepřítele, zde jsem vložil Quaternion.identity, takže rotace zůstane naprosto stejná. Poslední řádek mám pouze pro kontrolu, zdali nepřekračuje počet much aktuální počet na scéně, jak se chová generace a kontrola těchto počtů při různých situacích. Takto jsem vyřešil automatickou generaci nepřátel na viditelné scéně.

2.2 Zabití nepřátel

Poté co nepřátelé jsme schopni Generovat, tak je potřeba je nějak i zabít a dostat z nich něco co budeme schopni později využít v samotné hře.

```
public void OnMouseDown()
{
    if (!Player.sitkaActive)
    {
        Vector2 pos = gameObject.transform.position;
        Destroy(Instantiate(Particles1, pos, Quaternion.identity), 2f);

        Destroy(gameObject, 0.01f);

        Player.Kill++;

        GenerateEnemyScript.pocet--;

        if (Player.loot)
        {
            Loot();
        }
```

Zde využívám již přidanou metodu Unity, která se jmenuje OnMouseDown(). Tato metoda udělá to, že pokud klikneme na nějaký objekt, který má na sobě Collider, tak určí. na jaký objekt to vlastně klikáme a můžeme dělat s ním, co potřebujeme. My si vezmeme tento objekt a vložíme ji do proměné pos. Následně využijeme metody Instantiate(), který je vnořený do metody Destroy(). Stane se zde to, že si spawnu particle smrti, které vlastně po 2 vteřinách smažu, aby mi nezůstali ve scéně i po smrti a tím zamezil popřípadě nějakému sekání-zpomalování hry. Mezitím co se objevili particle a přehrávají se, tak smažu i samotného nepřítele. Program smaže nepřítele a přičte body získané za zabití hráči do centrálního scriptu jménem Player, kde se nahrávají všechna důležitá data, která chceme později ukládat a opět načítat. Poté odečtu již ak-

```
void Loot()
{
    int Wing_drop = Random.Range(0, 2);
    int Leg_drop = Random.Range(0, 4);

    Player.Wing += Wing_drop;
    Player.Leg += Leg_drop;
}
```

tuální počet spawnutých nepřátel a zavolám metodu Loot. V této metodě si opět vygeneruji náhodné číslo. Toto číslo poté přičtu k datům hráče, kolik má křídel a noh, které bude moci využít později ve hře.

2.3. Automatické zabíjení nepřátel

Nyní se vrhnu do zajímavější problematiky a to je automatické hledání a zabíjení nepřátel. Celé by to mělo vypadat tak, že máme například žabku, která má svoji animaci a vždy, když hraje ta animace, tak najde nepřítele, kterého zabije a posune se na dalšího.

Celé to začíná programem, který spawne naši žabku ve scéně na správných pozicích.

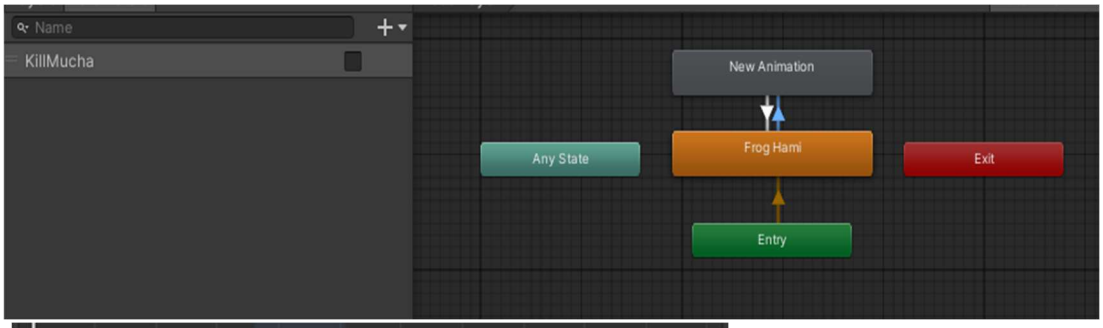
Poté se zapne časovač, to je díky metodě StartCoroutine(), který zavolá metodu, která by měla vypadat takto:

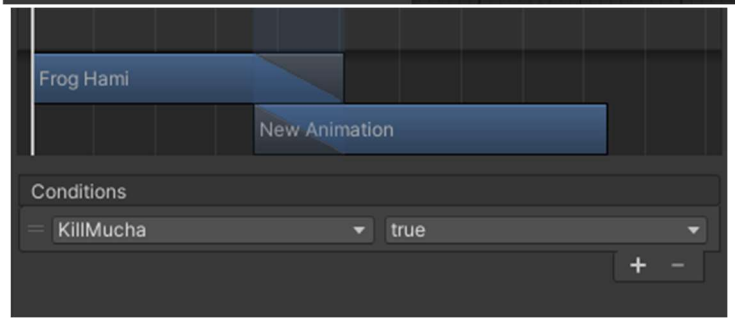
IEnumerator JménoMetody(){

yield return new příkaz;

}

Já jsem využil metodu WaitForSeconds(), kde pak vkládám, jak dlouho chci čekat.





Jakmile skončí časovač, tak si zavolám animátor, komponent toho objektu, a nastavím v něm hodnotu, kterou jsem si přednastavil, na true.

```
public void Start()
{
    transform.position = new Vector3(-1.68f, -3.3f, 0.5f);
    StartCoroutine(WaitBeforekill());
}

Počet odkazů: 1
IEnumerator WaitBeforekill()
{
    yield return new WaitForSeconds(Player.FrogCoolDown);
    GetComponent<Animator>().SetBool("KillMucha", true);
}
```

```
public static GameObject mucha = null;
public static GameObject svetluska = null;
public GameObject Parts;
Vector2 pos;
```

Tím započnu Animaci a zároveň volám script, který probíhá během této animace. Animaci jsem si nastavil tak, aby když se zapne hra, tak automaticky jde do políčka Frog Ham. To je animace, kdy je žába v klidovém stavu a jen tak si odpočívá. Poté, co

```
override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    mucha = GameObject.Find("mucha 1(Clone)");
    svetluska = GameObject.Find("svetluska 0(Clone)");
}
```

ze scriptu nastavím KillMucha na true, tak se z políčka Frog Ham přeskočí na New Animation, kde žába útočí a buď zraní, nebo rovnou zabije nepřítele. Po přehrání animace nic nebrání tomu, aby se animace vrátila zpátky do stavu Frog Ham, kde zas bude čekat, než se znova spustí odpočet.

```
if (mucha != null)
{
    pos = mucha.transform.position;
    mucha.GetComponent<LakeHmyzScript>().TakeDamage();
```

V kódu jsem si jako první nadeklaroval proměnné. První dva GameObjecty jsem rovnou inicializoval na null, to je z toho důvodu, že až budu zjišťovat jestli mám nalezený nějaký objekt, tak jestli ne - budu hledat dál, pokud ano- budu s ní dál pracovat.

Metoda OnStateEnter se volá vždy, když se animace začíná přehrávat. Zde poté hledám objekty se stejným názvem, jako jsou v uvozovkách. Na konci je (Clone) z toho důvodu, že pokud pomocí metody Instantiate() něco vytvoříme, tak se automaticky přidá toto na konec jména. Díky tomuto jsem schopen dobře ve scéně rozeznat, co je vytvořeno mnou a co je vytvořeno kódem.

Jakmile animace končí, spustí se metoda OnStateExit ve které bude zbytek kódu. Porovnám si pomocí if () jestli je GameObject mucha prázdný, jestli ne, tak vezmu pozici GameObjectu mucha a uložím ji do proměné pos, kterou jsem deklaroval na začátku skriptu. Poté si zavolám metodu z jiného skriptu. Zde využívám toho, že jsou dva skripty v jednom objektu, takže pomocí metodu GetComponent<>() si vytáhnu

ten komponent, který potřebuji, a zavolám si metodu TakeDamage(), která vypadá takto.

```
public void TakeDamage()
{
    GOhealthbar.gameObject.SetActive(true);
    currentHealth -= Player.FrogPower;
    healthBar.Sethealth(currentHealth);
}
```

Zde zapnu viditelnost healthbaru, odečtu damage od nynějšího zdraví a nastavím nynější zdraví pro healthbar, ve kterém se následně graficky zobrazí.

```
if (mucha.GetComponent<LakeHmyzScript>().currentHealth <= 0)
{
    Player.Kill += 1 * Player.FrogCoinMultiple;

    GenerateEnemyScript.pocet--;

    Destroy(Instantiate(Parts, pos, Quaternion.identity), 2f);
    Destroy(mucha, 0.01f);

    animator.SetBool("KillMucha", false);
    animator.GetComponent<Frog>().Start();
    mucha = null;
}
```

Pomocí dalšího příkazu if () zkontroluji aktuální zdraví nepřítele a pokud je menší, tak přičtu hráči peníze, které vynásobím číslem, které se mění podle toho, jak mám vylepšený tento atribut u žabky. Následně odečítám počet nepřátel na scéně. Znova spawnuji particle jako když zabiju nepřítele ručně a zničím objekt nepřítele. Nastavím vnitřní hodnotu animatoru KillMucha na false, čímž zastavím zacyklení animace a znova spouštím metodu start, která znova začne odpočítávat odpočet. Nakonec nastavím hodnotu na null. Tím uvolním místo pro dalšího nepřítele.

```
animator.SetBool("KillMucha", false);
animator.GetComponent<Frog>().Start();
```

Pokud zdraví nepřítele není menší nebo rovno nule, tak jen nastavím hodnotu Killmucha na false a opět si zavolám metodu start, abych znovu mohl spustit odpočet.

```
else
{
    animator.GetComponent<Frog>().Start();
}
```

Pokud to mouchu nenašlo, tak se znova spustí odpočítávání, které dá popřípadě čas na to, aby se nepřátelé zas mohli spawnout a mohl bych je najít.

2.4. Swipování a zabíjení pomocí ježdění po displaji

Jelikož vytvářím hru na mobil, bylo by určitě dobré využít funkci swipu, například na přecházení mezi 2 scénami. Když budeme u práce s displejem, tak bych rovnou mohl zabít nepřítele i tím, že bych na ně jen nemačkal, ale taky ho zabili tím, že bych jezdil po displeji prstem. U této problematiky, jsem si musel pomoci hledáním na internetu, ovšem využil jsem pouze základní princip, další funkce jsem naprogramoval sám.

```
private Vector3 fp; // first touch pos
private Vector3 lp; // last touch pos
private float dragDistance; // minimum distance
```

Jako první jsem si nadeklaroval fp (First Point) což je počáteční pozice kliknutí na displej, lp (Last Point) je poslední pozice na displeji, než zvednu prst a dragDistance je minimální vzdálenost, kterou ty dva musí mít, aby se příkaz vůbec provedl.

```
void Start()
{
    dragDistance = Screen.height * 15 / 100;
}
```

V metodě void Start jsem si vložil výšku displeje * 15 / 100, to umožní responzivní vzdálenost.

V metodě Update pomocí if () kontroluji, zdali je nějaký prst položen na displeji, pokud ano, tak pomocí metody Input.GetTouch(0), kde ta nula znamená 1. dotyk prstu, vložím všechny informace toho dotyku na displeji do proměnné, kterou deklaruji pomocí Unity datového typu touch, který uloží veškeré informace, právě do této proměnné. Dalšími třemi podmínkami if () kontroluji fáze toho dotyku na displeji.

```
void Update()
{
    if (Input.touchCount > 0)
    {

        Touch touch = Input.GetTouch(0);
```

- TouchPhase.Began - nastaví se na true tehdy, pokud se hráč dotkl displeje

- TouchPhase.Moved – nastaví se na true tehdy, pokud hráč nezvedl prst a jezdí prstem po displeji
- TouchPhase.Ended – nastaví se na true tehdy, pokud hráč zvedne prst z displeje

```
if (touch.phase == TouchPhase.Began)
else if(touch.phase == TouchPhase.Moved)
{
    lp = touch.position;

    if (Application.loadedLevel == 0)
    {
        float x = (touch.position.x / (Screen.width / 100)) / 100;
        float y = (touch.position.y / (Screen.height / 100)) / 100;

        Vector3 pos = new Vector3(x, y, 10.4f);
        pos = Camera.main.ViewportToWorldPoint(pos);
        if (spawnuto <= 0)
        {
            Instantiate(SekSek, pos, Quaternion.identity);
            spawnuto++;
        }

        SekSek1 = GameObject.Find("sek sek(Clone)");

        SekSek1.transform.position = pos;
    }
}
```

Pokud je tedy TouchPhase.Began true, tak uložím fp a lp.

Pokud je TouchPhase.Moved true, tak uložím lp a kontroluji, pokud jsem na scéně 0. Pokud ano, tak si spawnu kuličku, která následuje náš pohyb a jsme díky tomu schopni hezky a příjemně manipulovat kuličkou, která při dotyku zabije nepřítele. Dopočítám její pozice a spawnu ji na místě, která odpovídá místě dotyku. Mám zde ještě pomocí jednoho if ošetřeno to, aby se kuličky nespawnovali neustále dokola a nezačali, tak lagovat hru. Poté najdu GameObject jménem “sek sek(Clone)“ což je ta kulička, která se spawnuje. To jsem udělal kvůli tomu, abych následně tomuto objektu, který je již ve scéně, mohl měnit pozice.

```
else if (touch.phase == TouchPhase.Ended)
{
    lp = touch.position;
    if (Application.loadedLevel == 0)
    {
        Destroy(SekSek1, 0.1f);
        spawnuto = 0;
    }

    if (Mathf.Abs(lp.x-fp.x)>dragDistance||Mathf.Abs(lp.y-fp.y)>dragDistance)
    {
        if (Mathf.Abs(lp.x-fp.x)>Mathf.Abs(lp.y - fp.y))
        {
            if(lp.x > fp.x)
```

Pokud TouchPhase.Ended je true, tak naposledy uložím lp a pomocí if () kontroluji, zdali jsem na scéně 0, pokud ano, tak vymažu tu kuličku a zas otevřu if (), který ji spawnuje. Pokud nejsem na scéně 0, tak tento if () přeskočím a jdu porovnávat lp a fp. Nejdříve porovnám to, zdali je bud x a nebo y hodnota lp a fp je větší než dragDistance. Porovnám to tak ,že odečtu tyto dvě hodnoty a výsledek dám do absolutní hodnoty. Dalším krokem. co musím porovnat, je to, zdali se hýbalo, spíše po x a nebo y ose, to opět zjistím tak, že odečtu x hodnoty z fp od lp a to samé i u y a výsledky dám oba dva opět do absolutní hodnoty. Jakmile zjistím, po jaké ose se spíše hýbalo, například po x, tak potřebuji zjistit, zdali to bylo spíše doleva a nebo doprava, to pak následně jistím pouze porovnáním x hodnoty lp a fp.

2.5. Day/Night Cyklus

Další funkcí v mé hře je to, že jsem chtěl, aby se reálně projektoval čas i do hry, takže když je venku den, tak i ve hře je den a to samé i s nocí. Pak dle toho, pokud je noc, tak spawnu jiné nepřátelé než přes den.

```
SunriseSunset = new int[,,]
{
/*leden*/ {/*1.*/{7,58,16,13}, /*2.*/{7,58,16,14}, /*3.*/{7,58
/*9.*/{7,56,16,22}, /*10.*/{7,56,16,23}, /*11.*/{7,55,16,25},
/*17.*/{7,51,16,33}, /*18.*/{7,50,16,35}, /*19.*/{7,49,16,36},
/*25.*/{7,43,16,46}, /*26.*/{7,42,16,47}, /*27.*/{7,41,16,49},
```

Můj hlavní problém byl ten, že jsem nedokázal nějak automaticky a jednoduše dopočítávat východ a západ slunce. Takže jsem musel vytvořit trojdimenzové pole, do kterého jsem vepsal východy a západy slunce. Podle kterého se pak řídí den a noc.

```
if(SunriseSunset[System.DateTime.Now.Month - 1, System.DateTime.Now.Day - 1, 0] * 100 +
   SunriseSunset[System.DateTime.Now.Month - 1, System.DateTime.Now.Day - 1, 1] <
   (System.DateTime.Now.Hour)*100 + System.DateTime.Now.Minute && (System.DateTime.Now.Hour)
   * 100 + System.DateTime.Now.Minute <
   (SunriseSunset[System.DateTime.Now.Month - 1, System.DateTime.Now.Day - 1, 2] *
   100 + SunriseSunset[System.DateTime.Now.Month - 1, System.DateTime.Now.Day - 1, 3]))
```

Jakmile jsem měl trojdimenzové pole dopsané, tak jsem porovnal v poli pouze aktuální čas, kdy jsem hodiny musel vynásobit 100 a tak ošetřil to, aby byla noc pouze tehdy, kdy má být.

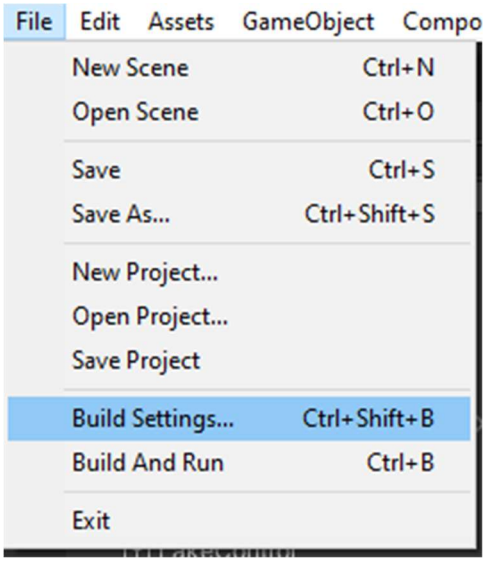
```
Color color = noc.GetComponent<SpriteRenderer>().color;
noc.GetComponent<SpriteRenderer>().color = new Color(color.r, color.g, color.b, 0.01f);
Player.JeDen = true;
```

Pokud je den, vezmu si předpřipravený objekt, kde mám nastavenou tmavě modrou barvu a měním pouze alfu na závislosti toho, zdali je noc a nebo den.

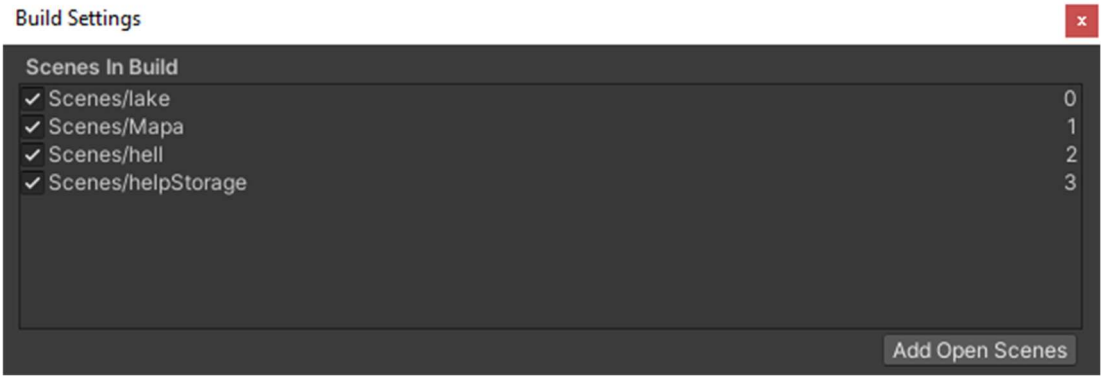
Jakmile jsem schopen zjistit, kdy je noc a kdy den, tak to pouze uložím do centrálního scriptu Player ze kterého to pak následně volám do ostatních scriptů. Vytvořil jsem podmínku if (), kde jen kontroluji, zda-li je noc a pokud ano, spustí se script a místo much se budou spawnovat světlušky.

2.6. Build a následná instalace

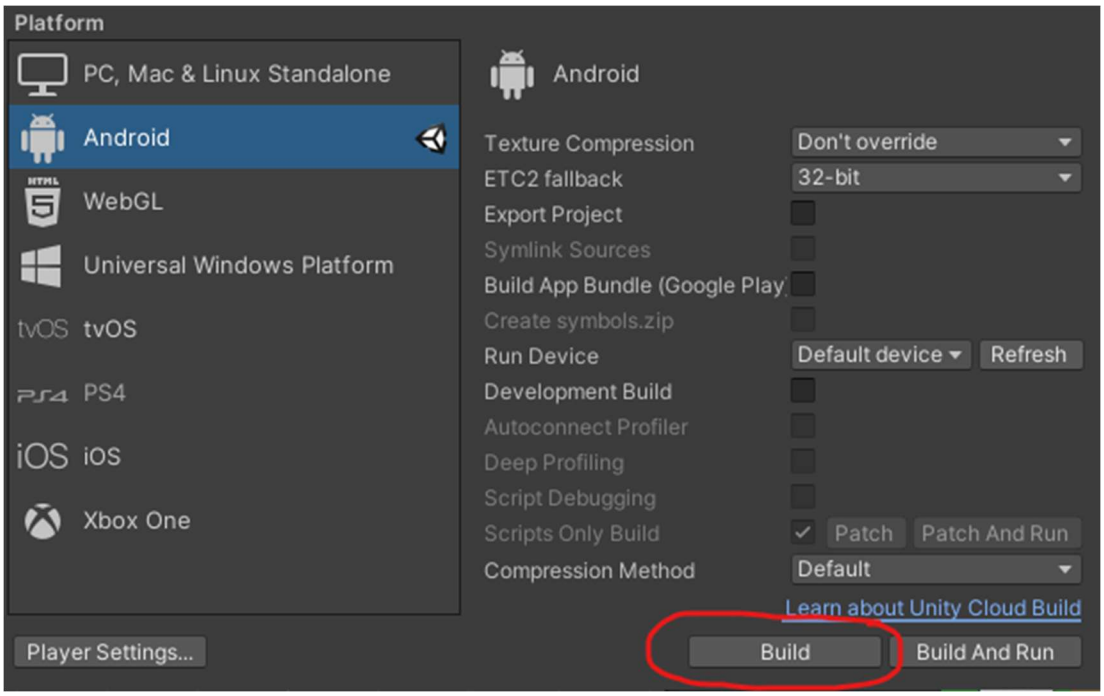
Abych mohl hru otestovat v praxi, tak ji budu muset složit do jednoho souboru, který mi později naistaluje vše potřebné pro to, aby hra fungovala.



Toho dosáhnu tím, že kliknu na File a dám Build Setting… poté se objeví okno, kde nahoře mohu vidět scény, které se buildnou. Pokud tam nejsou všechny moje scény, tak je přidám buď tím, že kliknu na tlačítko Add Open Scenes a nebo tím, že si najdu scénu a pomocí přetáhnutí ji tam vložím a zkontroluji, zda-li je scéna vlevo zaškrtnutá. Pokud by nebyla, nebyla by následně zahrnuta do buildu a ve hře by se pak nenačetla.



Následně pokud mám zkontrolované, že mám všechny scény označené, spustíme Build. Ten vezme veškeré potřebné soubory a vloží je do jednoho balíčku, který bude instalátor hry. Díky tomuto, budu moci snadně a jednoduše nainstalovat aplikaci do zařízení a spustit ji. Po skončení buildu, se ukáže build ve složce, do které byl vložen.



Teď už jen stačí připojit mobil do počítače a povolit přenášení souborů. Přetáhnout instalátor na mobil. Spustit ho a následně otestovat hru přímo v praxi, to jak se chová, jak reaguje, co popřípadě zlepšit a najít nějaké ty bugy.

3 Výhody a nevýhody Vytváření mobilní hry oproti PC hry

Obrovskou nevýhodou je to, že pokud chceme hledat chybu, která se nám ukazuje pouze na mobilu, ale v editoru ne, tak člověk musí vymýšlet různé kreativní cesty, jak to vyřešit. Další nevýhodou je to, že člověk musí řešit rozlišení aktuálního display a dle toho popřípadě přizpůsobovat interface. Jelikož mobily mají různá rozlišení, tak i samotná hra může někdy překvapit tím, že na vašem mobilu to vypadá vše v pořádku, ale na jiném zařízení je vše rozhozené, velikosti nesedí a něco se překrývá. Takže člověk musí myslet zároveň i na ošetření tohoto faktoru. Ovšem výhodou je to, že debugovat a testovat svoji hru, můžete vlastně kdekoliv a kdykoliv, jelikož máte svoji verzi v mobilu, tak kdykoliv se budete nudit, můžete hledat chyby a bugy, které následně opravíte, jakmile se vrátíte domů. Další výhodou je to, že mobilní hru můžeme popřípadě vložit na nějaký App Store a nebo Google play a celkem jednoduše ji rozšířit mezi budoucí hráče.

Závěr

Dle mého se má práce povedla, ač vím, že kdybych měl více času, byl bych schopen do této hry vložit více funkcí, udělat samotnou hru ještě zajímavější a doopravit chybky, které tam jsou. Celkový čas práce je odhadem cca 450h. Jelikož, když jsem začínal, tak jsem s Unity neuměl skoro vůbec nic, musel jsem většinu času hodně experimentovat a hledat způsoby toho, jak vyřešit daný problém a sem tam se podívat i na internet, ač jsem se většinu času snažil vyřešit vše sám a svým způsobem. K tomu grafika, ač není nejlepší, tak i ta zabrala dosti času.

Přínos práce byl pro mě obrovský, jelikož jsem se v Unity strašně moc zlepšil a mé chápání vytváření programu se posunulo o obrovský kus dopředu. Naučil jsem se nové příkazy v C# a o trošičku jsem se zlepšil v Gimpu. Můj záměr do budoucna je takový, že bych se rád věnoval programování této hry a doufám, že bych se jednou mohl dostat i do fáze, kdy by tato práce, ale již kompletní a dle mých představ mohla být hratelná na Google play. S případnýma aktualizacemi a s vloženými reklamami, které by mi v budoucnu mohli pomáhat mě živit, vytvořit menší komunitu a i když je to asi nemožné, tak i menší pasivní příjem. Nechci to rozhodně vzdát, protože dle mého jsem ještě zcela nevyužil celý potenciál této hry a ani svůj.

Zdroje

Základy Unity:

https://www.youtube.com/watch?v=IlKaB1etrik&t=107s&ab_channel=Brackeys

Time Delay:

https://www.youtube.com/watch?v=Qt5KPNmKglM&ab_channel=NoahAnderson

Animátor:

https://www.youtube.com/watch?v=hkaysu1Z-N8&t=684s&ab_channel=Brackeys

Youtube kanál:

https://www.youtube.com/@VratislavMedricky/videos

Čas z PC :

https://forum.unity.com/threads/how-to-get-system-time-in-unity.29667/

Multidimensionální pole:

https://www.youtube.com/watch?v=EVev2NhtVsc&ab_channel=b3agz

Swipe:

https://www.youtube.com/watch?v=haCWs5cg3cY&list=LL&index=1&t=6s&ab_channel=Xnapy

Animation Event:

https://www.youtube.com/watch?v=R51Af_ADgKE&ab_channel=MetalStormGames

Typy herních enginu:

https://appradar.com/blog/mobile-game-engines-development-platforms

Přílohy

Projekt v Unity

Vytvořená grafika

Dokumentace