**ÚVOD DO PROGRAMOVANIA**

Program je sled príkazov. Každý program sa dá rozdeliť na menšie elementárne úlohy, ktoré sú riešiteľné príkazmi konkrétneho jazyka. Príkazy vykonáva procesor, údaje/dáta sú uložené v operačnej pamäti RAM.

Procesor teda vykonáva jednoduché príkazy (inštrukcie – RISC/CISC). Ich zápis sa nazýva strojový kód a je pre ľudí nečitateľný. Čiastočne čitateľnejšia forma je zápis kódu v jazyku assembler, každý procesor má však inú syntax tohto jazyka.

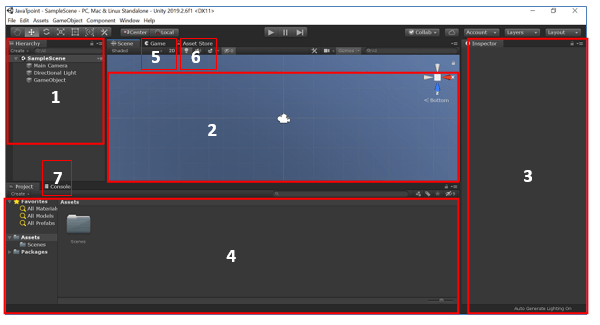
Čitateľnosť zlepšil príchod vyšších programovacích jazykov (napr. C). Tie sú písané príkazmi ako napr. while (pokiaľ), break (preruš) a pod. Takýto jazyk patrí do skupiny tzv. štruktúrovaných. Pre ešte ľahšie/názornejšie programovanie vznikli objektovo orientované jazyky (napr. C++/C#/Java), keďže všetko vôkol nás sa dá vnímať ako objekty s vlastnosťami a činnosťami, ktoré sa s nimi dajú robiť.

Kód programu písaný v prog. jazyku sa volá zdrojový kód. Tento kód spracuje program zvaný kompilátor, ktorý ho pretransformuje do strojového kódu konkrétnej platformy. Zdrojový kód môže byť vstupom aj pre interpréter, čo je program, ktorý priamo spúšťa program v sebe. Niečo ako emulátor. Ponúka multiplatformovosť, no zaťažuje HW oproti natívnemu programu.

Vývojové prostredie je program, ktorý slúži na pohodlné písanie kódu. Okrem editora môže obsahovať kompilátor, simulátor, opravovač/dopisovač kódu, modul na komunikáciu a pod.

**HERNÝ VÝVOJ**

Na jednoduchší a rýchlejší vývoj hier je vhodné použiť herný engine, ktorý obsahuje nástroje na ľahšiu tvorbu hry. Unity engine využíva na skriptovanie jazyk C# a defaultne spúšťa Visual Studio IDE.

1- okno hierarchie obsahuje objekty, ktoré sú v hre.

2- okno scény obsahuje herný svet – rozloženie objektov v scéne

3- okno inšpektora umožňuje editovanie vlastností a komponentov zvoleného objektu

4- okno assetov zobrazuje všetky assety (herné prvky), ktoré sú v projekte (nie hre)

5- okno hry zobrazuje to, čo uvidí hráč z aktívnej kamery (časť scény)

6- okno assettore, kde si užívateľ môže stiahnuť predvytvorené henré prvky

7- okno konzoly na zobrazovanie výpisu pomocou Debug.Log() alebo print()

Ak chceme zvýšiť výkon Unity (renderovania), za názov vo vlastnostiach sa dopíše: *\*\*.exe -force-d3d9*

Pre výpis do konzoly je možné použiť: *print()* alebo *Debug.Log()*

Defaultne zdrojový kód v skripte je:

*using UnityEngine;   
public class demo : MonoBehaviour   
{   
 void Start()  
 {*

*// kód, ktorý sa vykoná pri načítaní objektu – raz na začiatku (inicializácia objektu)  
 }   
void Update()   
{*

*// kód, ktorý sa vykonáva opakovane - cca 60x za sekundu (prepočet objektu)  
}*

*}*

**JAZYK C# - PREMENNÁ**

Základným prvkom každého programu je premenná, keďže program pracuje s dátami. Programátor nevie, aké dáta budú, preto ich univerzálne v kóde reprezentuje premennou (odkaz na časť pamäte, kde sa dáta nachádzajú).

Typy premenných:

*int cislo = 1; // cele cislo*

*float desatinne = 3.14f; // zakladne desatinne cislo*

*double dvojnasobne = 3.14; // presnejsie desatinne cislo*

*char znak = ‘1‘; // znak sa pise do apostrofu*

*string text = “1“; // pouzivaju sa uvodzovky*

*bool bolean = true; // resp. false*

*var univerzalnyTyp; // typ sa prisposoby podla toho, aka hodnota sa don prva zapise*

-deklarácia / inicializácia / definícia

-lokálne / globálne

-existencia v rámci blokov a prekrytie

-pravidlá pre tvorbu názvov (znaky / čísla / ťaví zápis / prvé písmeno )

-modifikátory (unsigned, signed, short, long, const)

Príklad:

*void Start()*

*{*  
 int aktualnyRok = 2024;

int rokNarodenia = 1985;

int vek = aktualnyRok – rokNarodenia;

print(“Mám “ +vek + “ rokov.“); *}*

**JAZYK C# - OPERÁTORY**

Operátory slúžia na prácu s premennými (aritmetické, binárne / logické, relačné):

|  |  |  |
| --- | --- | --- |
| **SYNTAX** | **VÝZNAM** | **POUŽITIE** |
| + | aritmetický súčet (sčítanie) | 5 + 2 (7) |
| - | aritmetický rozdiel (odčítanie) | 5 – 2 (3) |
| \* | aritmetický súčin (násobenie) | 5 \* 2 (10) |
| / | aritmetický podiel (delenie) | 5 / 2 (2) |
| % | zvyšok po celočíselnom delení (modulo) | 5 % 2 (1) |
| ++ | inkremenácia (zvýšenie o 1) | 5++ (6) |
| -- | dekrementácia (zníženie o 1) | 5-- (4) |
| = | priradenie | a = 5 |

|  |  |  |
| --- | --- | --- |
| **SYNTAX** | **VÝZNAM** | **POUŽITIE** |
| & | bitový súčin (AND) | 1001 & 1010 (1000) |
| | | bitový súčet (OR) | 1001 | 1010 (1011) |
| ^ | bitové vylúčenie (XOR) | 1001 ^ 1010 (0011) |
| ~ | bitová negácia (komplement) | ~1001 (0110) |
| << | bitový posun vľavo (shift vľavo) | 01 << 2 (0100) |
| >> | bitový posun vpravo (shift vpravo) | 0110 >> 2 (01) |
| && | logický súčin (AND) | POD1 && POD2 |
| || | logický súčet (OR) | POD1 || POD2 |
| ! | logická negácia (NOT) | !true (false) |
| ? | podmienkový operátor | POD ? PR1 : PR2 |

|  |  |  |
| --- | --- | --- |
| **SYNTAX** | **VÝZNAM** | **POUŽITIE** |
| == | test rovnosti | A == B |
| != | test nerovnosti | A != B |
| < | test menšieho | A < B |
| <= | test menšieho alebo rovného | A <= B |
| > | test väčšieho | A > B |
| >= | test väčšieho alebo rovného | A >= B |

**JAZYK C# - CYKLUS**

Cyklus je príkaz, ktorý sa použije, pokiaľ sa má nejaký kód opakovane vykonávať.

for(;;) – používa sa, keď sa vie, koľkokrát sa cyklus zopakuje

while() – používa sa, keď sa nevie, dokedy sa bude cyklus opakovať

do-while() – používa sa, keď sa má cyklus aspoň raz vykonať (až po vykonaní sa testuje podmienka)

foreach() – používa sa na prehľadanie celej kolekcie (vytvára dočasnú kópiu každého prvku v kolekcii)

Príklad:

int sucet = 0;

for (int i = 0; i < 1000; i++) // cyklus, ktorý sa zopakuje 1000x

{

int nahoda = Random.Range(0, 100); // vygeneruje sa náhodné číslo 0-99

sucet = sucet + nahoda; // alternatívny zápis: sucet += nahoda;

}

int priemer = sucet / 1000; // výpočet priemernej hodnoty

print("Priemerná hodnota je: " + priemer); // spojenie textu a čísla

Príklad demonštruje zápis cyklus for(). Jednotlivé tri časti hlavičky sú nepovinné, najčastejší zápis je však taký, že v prvej časti je definovanie dočasnej premennej pre rátanie iterácii, v druhej časti je podmienka porovnávajúca túto premennú s počtom opakovaní, v tretej časti je inkrementovanie tejto premennej.

*Radom.Range()* – metóda generujúca náhodné číslo v rozsahu min až max-1 (niekedy kompilátor hlási chybu, že nevie, ktoré Random použiť. Vtedy treba dopísať: *using Random = UnityEngine.Random;* )

*Random.value* – atribút (premenná) obsahujúca náhodné desatinné číslo (float) v rozsahu 0 – 1

Zvláštnosťou príkladu je, že i keď ide o 1000 náhodných čísel, priemerná hodnota je vždy rovná približne polovici. Náhodné generovanie asi nie je až tak náhodné :D

**JAZYK C# - VETVENIE**

Vetvenie sa používa vtedy, pokiaľ chceme, aby sa určitá časť kódu vykonala za určitých podmienok a za iných podmienok sa vykonala nevykonalo nič alebo niečo iné.

if() – jednoduché vetvenie, pokiaľ platí podmienka, kód sa vykoná, inak sa nevykoná

if() – kód v bloku za if sa vykoná pokiaľ platí podmienka, inak sa vykoná kód v bloku za else  
else

if() – postupne sa testujú podmienky, pokiaľ niektorá platí, vykoná sa daný blok, zvyšok sa preskočí  
else if()  
else if()  
else

switch() – vetvenie, ktoré testuje rovnosť dvoch hodnôt. Používa sa pri výberoch, stavoch a pod.

Príkazy if a if-else if-else if-else nie sú samostatné príkazy, ide o modifikáciu vetvenia if()-else. Platí totiž, že časť else sa nemusí písať.

Pri vetvení aj cykloch platí, že pokiaľ je v bloku jeden príkaz, zátvorky sa môžu a nemusia písať.

Príklad:

int tretina=0, dvojTretina=0, trojTretina=0; // treba definovať hodnotu =0

for (int i = 0; i < 1000; i++) // cyklus, ktorý sa zopakuje 1000x

{

int nahoda = Random.Range(0, 100); // vygeneruje sa náhodné číslo 0-99

if(nahoda < 33) // pre 0 az 32

{

tretina = tretina + 1;

}

else if(nahoda <= 65) // pre 33 az 65

{

dvojTretina += 1;

}

else // ostatne cisla, cize 66 az 99

{

trojTretina++;

}

}

print("Čísla od 0 do 32: " + tretina); // automatické spojenie string a int

print("Čísla od 33 do 65: " + dvojTretina.ToString()); // konverzia int->string ...

print("Čísla od 66 do 99: " + trojTretina.ToString()); // ... a nasledne spojenie

Príklad:

float priemernaZnamka = 0f;

for (int i = 0; i < 5; i++)

{

string znamka = "";

int cislo = Random.Range(1, 6);

priemernaZnamka += cislo; // automatická konverzia

switch(cislo)

{

case 1: znamka = "výborný";

break;

case 2:

znamka = "chválitebný";

break;

case 3:

znamka = "dobrý";

break;

case 4:

znamka = "dostatočný";

break;

case 5:

znamka = "nedostatočný";

break;

default:

znamka = "ERROR"; // tento pripad nikdy nenastane

break;

}

print("Žiak č." + (i + 1) + " má známku: " + znamka);

}

print("Priemerná známka žiakov je: " + priemernaZnamka/5);

**JAZYK C# - METÓDA (funkcia, procedúra, subrutina)**

Pre zmenšenie veľkosti kódu, znovupoužiteľnosť kódu a pre lepšiu čitateľnosť kódu je možné umiestniť logicky súvisiaci kód do bloku s príslušným názvom. Tento kód sa potom zavolá napísaním príslušného názvu.

Metóda môže len vykonávať kód, alebo vykonávať kód a navraciať hodnotu, alebo prijímať hodnotu/y a vykonávať sa, alebo prijímať, vykonávať a navraciať hodnotu.

void – metóda nič nenavracia (return sa používa pre okamžité/skoršie ukončenie metódy)

Metóda môže navrátiť len jednu vec, tou však môže byť objekt alebo kolekcia s mnohými vecami!

Metóda môže prijímať aj viacero a rôznych parametrov. Rovnako sa volajúcich metód môže byť viacero, musia sa však líšiť typom alebo počtom vstupných parametrov.

Príklad:

void Start()

{

print( SlovnaZnamka(3) ); // zavolanie vlastnej metódy s parametrom 3

}

string SlovnaZnamka(int znamka) // vlastná metóda vracajúca text

{

string slovo = "";

if (znamka == 1) // porovnanie vstupného parametra s číslom 1

slovo = "výborný";

else if (znamka == 2)

slovo = "chválitebný";

else if (znamka == 3)

slovo = "dobrý";

else if (znamka == 4)

slovo = "dostatočný";

else if (znamka == 5)

slovo = "nedostatočný";

else

slovo = "-";

return slovo; // návratová hodnota je string

}

V Unity existuje mnoho metód, ako napr. print(), Random.Range(), toString(). Niektoré očakávajú vstupné parametre, iné aj navracajú hodnotu alebo sa len vykonávajú. Existujú aj metódy, ktoré sa automaticky volajú (sú už predvytvorené a len sa prepisujú), napr. OnCollisionEnter(), Start(), ...

Nasledujúcim kódom sa dá demonštrovať FPS hry a to, čo reprezentuje hodnota Time.deltaTime.

void Awake() // vola sa pred Start()

{

Application.targetFrameRate = 60; // nastavi FPS na 60

}

void FixedUpdate() // vola sa 60x za sekundu

{

Debug.Log(Time.deltaTime); // vypis casu trvania snimku

}

void Update() // vola sa tak casto, ako moze

{

Debug.Log(Time.deltaTime);

}

**JAZYK C# - POLE / LIST (kolekcia)**

Pre definovanie viacerých premenných s rovnakým menom (ale rôznym indexom) sa používa pole. Pokiaľ je v poli ďalšie pole, hovoríme o viacrozmernom (dvoj, troj atď.). Pole musí mať všetky prvky toho istého typu. Pokiaľ je dvojrozmerné, musí mať rovnaký počet prvkov v každom podpoli. Veľkosť poľa sa musí definovať už pri jeho deklarácii.

List je oveľa viac dynamickejší ako pole. Prvky sa doň priebežne pridávajú a odoberajú (Add). Dokáže odstrániť prvky (RemoveAt), zoradiť ich (Sort), spočítať (Count) atď.

Vytvára sa: ***List<string> nazovListu = new List<string>()***

Práca s dvojrozmerným poľom (vytvorenie, naplnenie, načítanie hodnoty z poľa):

int[,] pole2D = new int[4, 2]; //vytvorí sa 2D pole 4x2 (4 polia a každé má: č,č)

pole2D[0, 0] = 10; pole2D[0, 1] = 20; //naplnenie podpoľa 0 (prvky s 0,1 indexom)

pole2D[1, 0] = 30; pole2D[1, 1] = 40;

pole2D[2, 0] = 50; pole2D[2, 1] = 60;

pole2D[3, 0] = 70; pole2D[3, 1] = 80;

if(pole2D[1,1] < 50) print("pod 50"); // porovnanie prvku poľa 1 a prvku 1 s 50

Príklad:

List<int> trpaslici = new List<int>(); // vytvori sa list cisel

for (int i = 0; i < 7; i++) // cyklus, ktory prejde 7x (od 0)

{

trpaslici.Add(Random.Range(50, 100)); // pridanie cisla do listu

}

int najvacsi = trpaslici[0]; // zapamatanie si velkosti prveho T

for (int i = 1; i < 7; i++) // cyklus, ktory prejde 6x (od 1)

{

if (trpaslici[i] > najvacsi) // ak je aktualny T. vacsi, ako NAJ

najvacsi = trpaslici[i]; // ... tak si zapis vysku aktualneho

}

print("Najvacsi trpaslik mal velkost " + najvacsi); // vypis

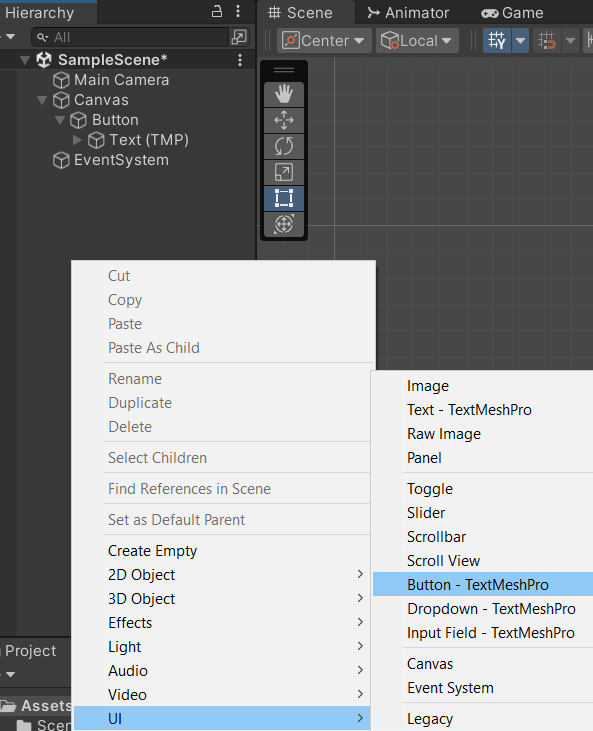
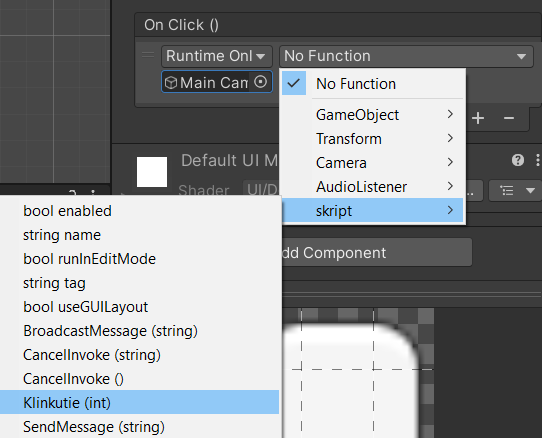
**UNITY ENGINE**

**UI – vstup/výstup**

Klávesy sa dajú zachytiť cez UI-Input\_Field alebo cez skript: **if( Input.GetKey( KeyCode.W ) )**

Kliknutie myši sa dá zachytiť v scéne (cez RayCast) alebo pomocou UI prvkov. Najjednoduchšie je použiť tlačidlo (Button). Tlačidlo sa kreslí do plátna (canva). Canva reprezentuje celú obrazovku, nemá nič spoločné so scénou. UI prvky sa poziciujú vzhľadom na plátno pomocou Anchor. Správne nastavené väzby sú dôležité pre rôzne rozlíšenie cieľového zariadenia.

Na objekte Button sa v časti OnClick() zvolí objekt so skriptom, z ktorého sa vyberie PUBLIC metóda.

Príklad:

List<int> odmena = new List<int>(); // vytvorenie listu

void Start()

{

for (int i = 0; i < 3; i++) // cyklus opakujúci sa 3x

odmena.Add(Random.Range(-1, 2)); // generovanie -1/0/1 do listu

}

public void Klinkutie(int poradie) // metoda ocakava vstupny parameter

{

if (poradie == 1) // ak sa klikol BTN1...

{

print(odmena[0]); // ...vypis LIST 0

}

else if (poradie == 2)

{

print(odmena[1]); // ...vypis LIST 1

}

else if (poradie == 3)

{

print(odmena[2]); // ...vypis LIST 2

}

else

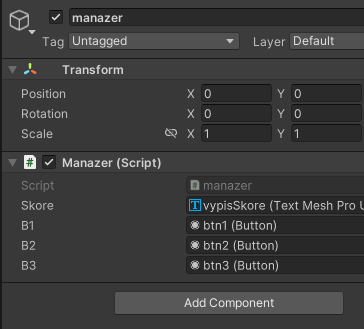
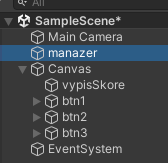
print("ERROR:" + poradie); // nenastane

// ako by bolo cez SWITCH ??

}

Pre zobrazenie textu použijeme okrem print() objekt UI-TMP\_Text. TMP\_Text je vylepšený UI prvok, a preto vyžaduje menný priestor TMPro. Pokiaľ chceme pracovať s objektami v scéne/plátne cez skript, musíme získať na ne odkaz/referenciu (napr. dať ich ako globálne a verejné a dropnúť ich).

Príklad:



using UnityEngine; // vlozenie Unity veci ako Time, Update(), ...

using UnityEngine.UI; // umoznuje pouzit v kode Button

using TMPro; // umoznuje pouzit v kode TMP\_Text

public class manazer : MonoBehaviour

{

public TMP\_Text skore; // referencia na TMP\_Text

public Button b1, b2, b3; // odkaz na tlacidla

//public GameObject univerzalnyObjekt; // mozno dat don hocico

int score = 0; // "globalne" premenne

int volba = 0;

float cas = 0;

void Start()

{

skore.text = "Skóre: 0"; // defaultny text po spusteni

}

void Update()

{

cas = cas + Time.deltaTime; // spocitavanie casov trvania snimok

if (cas >= 1f) // ak je sucet vecsi ako 1, presla sekunda

{

b1.GetComponent<Image>().color = Color.red; // nastavenie farby

b2.GetComponent<Image>().color = Color.red;

b3.GetComponent<Image>().color = Color.red;

volba = Random.Range(1, 4); // vygenerovanie nahodneho cisla

if (volba == 1) // ak sa vygenerovala 1, tak...

b1.GetComponent<Image>().color = Color.green; // ...nastav farbu

else if (volba == 2)

b2.GetComponent<Image>().color = Color.green;

else if (volba == 3)

b3.GetComponent<Image>().color = Color.green;

cas = 0; // na konci vymazeme premennu cas, aby spocitaval od zaciatku

}

}

public void Kliknutie(int ktoreTlacidlo) // metoda pre stlacenie tlacidla

{

if (ktoreTlacidlo == 1 && volba == 1) // ak bolo tlacidlo 1 a aj malo byt...

score++; // ...zvys skore

else if (ktoreTlacidlo == 2 && volba == 2)

score++;

else if (ktoreTlacidlo == 3 && volba == 3)

score++;

else // ak sa nepodarilo spravne stlacit...

score--; // ...zniz skore

skore.text = "Skóre: " + score.ToString(); // vypis do objektu Text

}

}

*Úloha: Príklad modifikujte tak, aby sa ihneď po voľbe vygeneroval nový výsledok; skracoval sa postupne čas generovania; použili sa klávesy namiesto kurzora; nahradili sa obrázky namiesto farieb.*

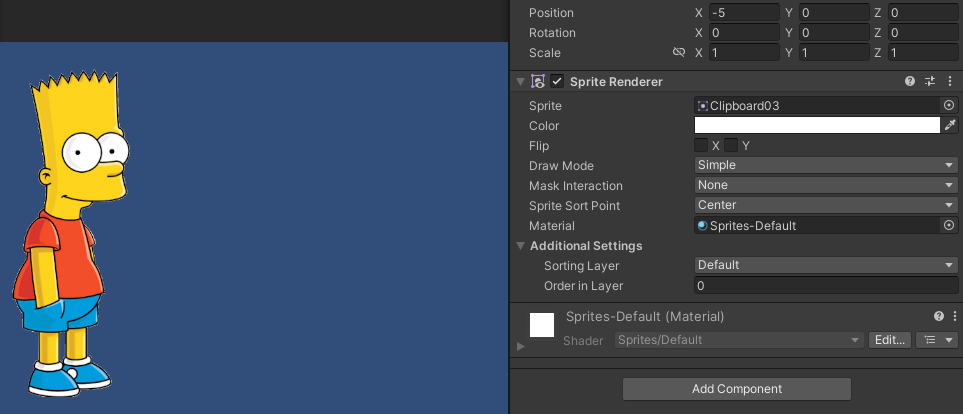
**Time.deltaTime (interval)**

Na vykonávanie kódu v pravidelnom intervale je vhodné použiť: Time.deltaTime . Tento atribút udáva desatinné číslo, ktoré reprezentuje čas trvania jednej snímky. Spočítaním všetkých týchto časov behom jednej sekundy získame číslo +-1. Slúži tiež na korekciu slabého, kolísavého výkonu počítača.

**Transform, GetComponent<>**

Každý objekt má komponent Transform. Cez ten sa dá meniť pozícia, rotácia a mierka. Pozor, position.X sa dá len čítať! Meniť treba celú pozíciu. Tá očakáva tri čísla – Vector3.

Príklad:



public GameObject postava; // referencia na obrazok

float rychlost = 4f; // nasobitel rychlosti pohybu

bool ideVpravo = true; // priznak pre smer pohybu

void Update()

{

if (postava.transform.position.x >= 5f) // ak presiel cez hranicu 5...

{

ideVpravo = false; // zmen smer spat

postava.GetComponent<SpriteRenderer>().flipX = true; // zrkadli obrazok

}

else if (postava.transform.position.x <= -5f) // ak presiel cez hranicu -5...

{

ideVpravo = true; // zmen smer na vpravo

postava.GetComponent<SpriteRenderer>().flipX = false; // neotocena pozicia

}

if (ideVpravo) // ak ide vpravo...

{

postava.transform.position = new Vector3( // zmen poziciu .X + ???

postava.transform.position.x + rychlost \* Time.deltaTime, 0, 0);

}

else // ak ide vlavo...

{

postava.transform.position = new Vector3(

postava.transform.position.x - rychlost \* Time.deltaTime, 0, 0);

}

}

*Úloha: Príklad modifikujte tak, aby sa v náhodný čas menila náhodne rýchlosť.*

Ak by sa nachádzalo v programe viacero takýchto objektov, bolo by nutné spraviť viacero premenných a zopakovať viackrát kód. Čiastočne by sa to dalo riešiť cez listy a cykly. ALE lepšie je kód týkajúci sa konkrétneho objektu dať do samostatného skriptu a ten priradiť objektu. Duplikovaním objektov sa potom duplikujú aj skripty.

*Úloha: Spravte viacero obj. z predošlého projektu a jeden, ktorý budete pohybovať pomocou WSAD.*

**Kolízie**

Na zareagovanie objektov medzi sebou sa používa tzv. „ detekcia kolízie“. Aby sa dala zachytiť, je nutné, aby oba objekty mali na sebe komponent COLLIDER a aspoň jeden z nich RIGIBODY. Pokiaľ nechceme prepočítavať fyziku objektu, môže sa zvoliť možnosť KINEMATIC. Treba rozlišovať 2D/3D objekty. Všetky spomenuté komponenty a metódy udalostí majú verziu 2D aj 3D.

Pokiaľ chceme zachytiť nastatie kolízie, ale vzájomne neovplyvniť objekty (napr. chceme zareagovať na dotyk s mincou, ale nechceme, aby sme sa od nej odrazili), použijeme TRIGGER. Pokiaľ chceme zachytiť dotyk a prepočítať fyziku reakcie (napr. náraz do steny), použijeme COLLISION.

*OnTriggerEnter(Collider other)*

Na objekte, s ktorým chceme interagovať, zapneme vlastnosť IsTrigger v Collidery (nie na oboch). Tým sa stane akoby nehmotný. Môže sa zmeniť v RB2D BODYTYPE na KINEMATIC, čím sa odignoruje pôsobenie fyziky a tým ovplyvňovanie okolím. Constrains na RB sa môže freez\_nuť.

*OnCollisionEnter(Collision other)*

IsTrigger nesmie byť zvolený na objektoch. Aspoň jeden objekt musí byť DYNAMIC.

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag == "enemy")

{

collision.gameObject.SetActive(false); // deakivuje objekt

Destroy(collision.gameObject); // zmaze objekt z Hierarchy

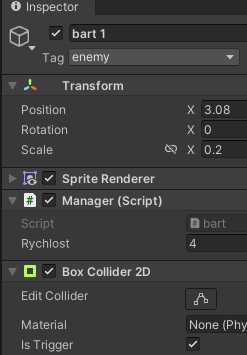
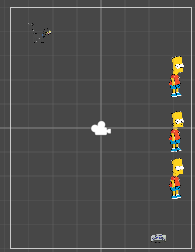
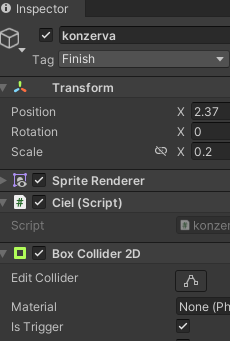
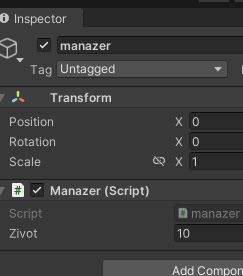
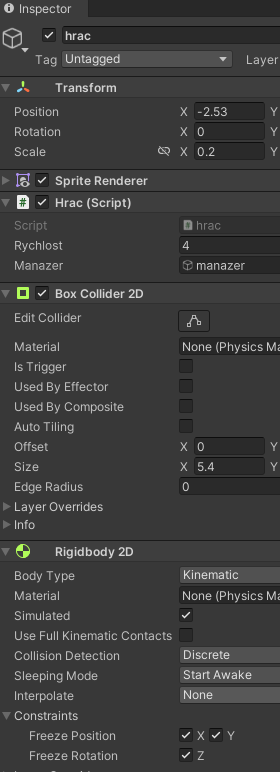
}

}

*DETAILY NA:* [*https://www.youtube.com/watch?v=Bc9lmHjqLZc*](https://www.youtube.com/watch?v=Bc9lmHjqLZc)

Príklad:

Vytvoria sa objekty troch nepriateľov (bart) a pridelí sa im skript BART.CS, BoxCollider2D (na ňom sa zvolí IsTrigger) a Tag sa nastaví na „enemy“. Vytvorí sa objekt konzerva, staví sa na ňom Tag na Finish a pridá sa BoxCollider2D. Vytvorí sa prázdny objekt manazer a priradí sa mu skript MANAZER.CS v ktorom je len verejný INT ZIVOT = 10. Nakoniec sa vytvorí objekt hráča, ktorý má BoxCollider2D, RigiBody2D (môže byť Kinematic, môže mať zablokované Constrains) a priradený skript HRAC.CS

**BART.CS**

public class manager : MonoBehaviour

{

public float rychlost = 4f; // nasobitel rychlosti pohybu

bool ideVpravo = true; // priznak pre smer pohybu

float cas = 0;

float kedy = 1f;

void Update()

{

if (transform.position.x >= 3f) // ak presiel cez hranicu 5...

{

ideVpravo = false; // zmen smer spat

GetComponent<SpriteRenderer>().flipX = true; // otoc/zrkadli obrazok

}

else if (transform.position.x <= -3f) // ak presiel cez hranicu -5...

{

ideVpravo = true; // zmen smer na vpravo

GetComponent<SpriteRenderer>().flipX = false; // neotocena pozicia

}

if (ideVpravo) // ak ide vpravo...

{

transform.position = new Vector3( // zmen poziciu na stara + rychlost\*cas

transform.position.x + rychlost \* Time.deltaTime, transform.position.y, 0);

}

else // ak ide vlavo...

{

transform.position = new Vector3(

transform.position.x - rychlost \* Time.deltaTime, transform.position.y, 0);

}

cas += Time.deltaTime;

if(cas > kedy)

{

cas = 0;

kedy = Random.Range(0.5f, 2);

rychlost = Random.Range(2, 8);

}

}

}

**HRAC.CS**

public class hrac : MonoBehaviour

{

public float rychlost = 4f;

public GameObject manazer;

void Update()

{

if (Input.GetKey(KeyCode.W))

transform.position += new Vector3(0, rychlost \* Time.deltaTime, 0);

if (Input.GetKey(KeyCode.S))

transform.position -= new Vector3(0, rychlost \* Time.deltaTime, 0);

if (Input.GetKey(KeyCode.A))

transform.position -= new Vector3(rychlost \* Time.deltaTime, 0, 0);

if (Input.GetKey(KeyCode.D))

transform.position += new Vector3(rychlost \* Time.deltaTime, 0, 0);

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.tag == "enemy")

{

manazer.GetComponent<manazer>().zivot--;

print(manazer.GetComponent<manazer>().zivot);

}

else if(collision.gameObject.tag == "Finish")

{

print("Wwww Oooo Nnnn !!!");

}

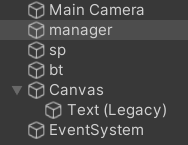
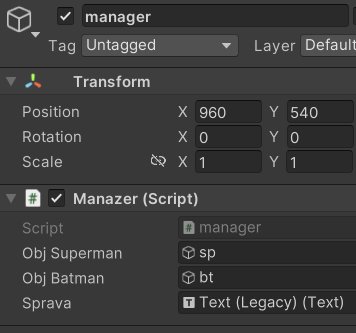
}

}

**ČASOVANIE POMOCOU YIELD**

Príklad:

Vytvoria sa dva objekty (postavy) a jeden UI/Text. Skript sa priradí objektu Manazer. Do skriptu sa prilinkujú oba obrázky i objekt s textom.

using System.Collections; // umoznuje pouzivat IEnumerator

using UnityEngine.UI; // umoznuje pouzivat stary UI/Text

public GameObject objSuperman; // odkaz pre postavu 1

public GameObject objBatman; // odkaz pre postavu 2

public Text sprava; // odkaz pre stary UI/Text

private int HPSuperamn = 100; // pociatocna hodnota je 100

private int HPBatman = 100;

void Start()

{

sprava.gameObject.SetActive(false); // UI/Text sa vypne

StartCoroutine(boj()); // spusti sa asynchronna funkcia boj

}

IEnumerator boj() // tato funkcia bezi popri zvysku hry

{

while (HPSuperamn > 0 && HPBatman > 0) // pokial ma niekto HP

{

HPSuperamn -= Random.Range(0, 11); // vymysli sa uder a odcita HP

HPBatman -= Random.Range(0, 11);

// zapiseme si farby RGBA obrazkov cez vlastnost: material.color

Color cSuperman = objSuperman.GetComponent<SpriteRenderer>().material.color;

Color cBatman = objBatman.GetComponent<SpriteRenderer>().material.color;

cSuperman.a = HPSuperamn / 100f; // upravime priehladnost

cBatman.a = HPBatman / 100f; // je od 0 po 1, preto treba podelit 100

// upravene farby sa spatne zapisu do objektov postav

objSuperman.GetComponent<SpriteRenderer>().material.color = cSuperman;

objBatman.GetComponent<SpriteRenderer>().material.color = cBatman;

yield return new WaitForSeconds(.1f); // pockaj desatinu sekundy

}

if(HPSuperamn > 0) // ak ma Superman zivot, je vytazom

sprava.text = "Víťazom je Superman"; // nastavenie textu

else

sprava.text = "Víťazom je Batman";

sprava.gameObject.SetActive(true); // nakoniec zobraz/aktivuj UI/Text

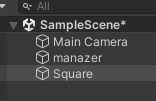
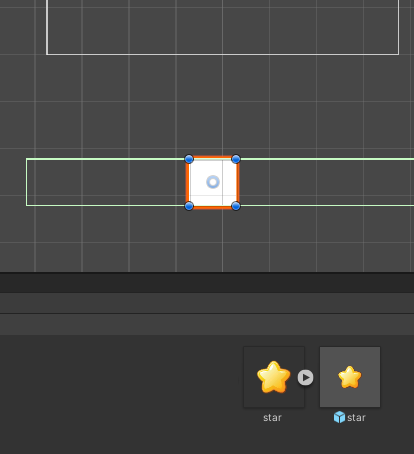
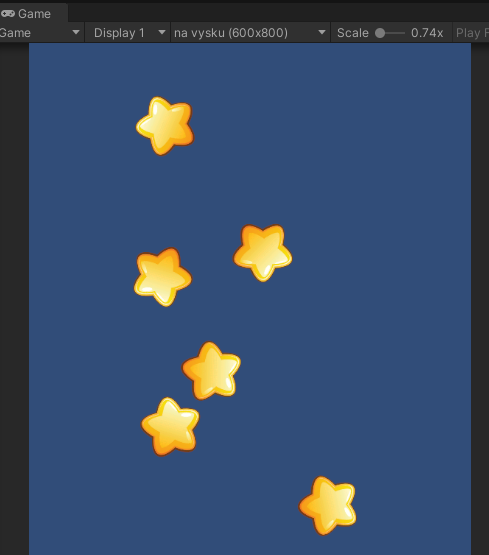
}

**PREFAB A GENEROVANIE OBJEKTOV**

Potiahnutím objektu z Hierarchy do Assets vznikne z objektu prefabrikátor (vzorový objekt). Ten je vhodné používať, pokiaľ sa dané objekty vyskytujú viackrát v scéne (úpravou prefabu sa menia všetky jeho inštancie a zmenu inštancie je možné previesť spätne na prefab, ak treba).

Príklad:

Vytvorí sa objekt hviezdy, ktorý bude obsahovať skript star.cs, tag „star“ a boxCollider2D. Zapne sa vlastnosť Trigger. Aby sa neprehltila pamäť, vytvorí sa objekt pod obrazovku, dá sa mu RB2D na Kinematic a boxCollider cez celú obrazovku. Jeho skript bude detekovať kolíziu a mazať toho, kto sa ho dotkne. Nakoniec prázdny objekt manazer so skriptom bude v náhodnom čase na náhodnej pozícii generovať prefaby. Treba doň priradiť (drag&drop) prefab hviezdy.

 **** 

STAR.CS

float rychlostPadu = 0f;

float rychlostRotacie = 0f;

void Start()

{

rychlostPadu = Random.Range(1, 5);

rychlostRotacie = Random.Range(20, 100);

}

void Update()

{

transform.position += new Vector3(0, -Time.deltaTime \* rychlostPadu, 0);

transform.Rotate(0, 0, Time.deltaTime \* rychlostRotacie);

}

GROUND.CS

private void OnTriggerEnter2D(Collider2D collision)

{

if(collision.gameObject.tag == "star")

{

Destroy(collision.gameObject);

}

}

MANAZER.CS

public GameObject prefab;

float cas = 0f;

float dokedy = 0.1f;

void Update()

{

cas += Time.deltaTime;

if(cas > dokedy)

{

cas = 0;

dokedy = Random.Range(0.4f, 1.5f);

float kde = Random.Range(-3.2f, 3.2f);

Instantiate(prefab, new Vector3(kde,7,0), Quaternion.identity);

}

}

*Úloha: Vytvorte zbraň, z ktorej po stlačení klávesu vystrelí náboj a ten zmizne po 2 sekundách.*

**ANIMÁCIA**

Pokiaľ sa vytvára animácia statického objektu (napr. posun karty, otočenie hviezdy), je možné upraviť parametre objektu (transform, color, ...) v určitom intervale. Interval spravujeme cez Coroutine alebo „Time.deltaTime“. Pokiaľ vytvárame pokročilú animáciu (napr. animovanie postavy, výbuch, oheň), využívame časovú os. V nej sa vytvoria KeyFrame\_y a Unity dopočíta rozdiel medzi nimi podľa FPS.

Animácie môžu byť buď zmenou celých obrázkov – ako má GIF, alebo môže byť objekt tvorený z častí/obrázkov, ktorým sa upraví v čase transform a Unity prepočíta zmeny medzi KF.

Je vhodné ukladať všetky obrázky animácie v jednom súbore/obrázku. Takýto obrázok sa nazýva SpriteSheet. Vo vlastnostiach treba nastaviť mód na MULTIPLE a Apply. (Automatický blur sa dá vypnúť nasavením Compression na none.) Následne cez tlačidlo Sprite Editor sa upravia výrezy. Vľavo hore cez Slice sa určí, či sa to má spraviť automaticky podľa priehľadnosti alebo pomocou veľkosti jedného výrezu, alebo podľa mriežky, alebo voľne rukou. V obrázkoch je možné nastaviť tiež PIVOT a názov obrázkov. Nakoniec sa potvrdí zmena cez Apply.

Výberom podobrázkov v obrázku a posunutím do scény sa vytvorí na disku súbor s animáciou. Zároveň sa vytvorí aj Animator controller (správca prechodu medzi animáciami) a Animation (časová väzba snímok – animácia).

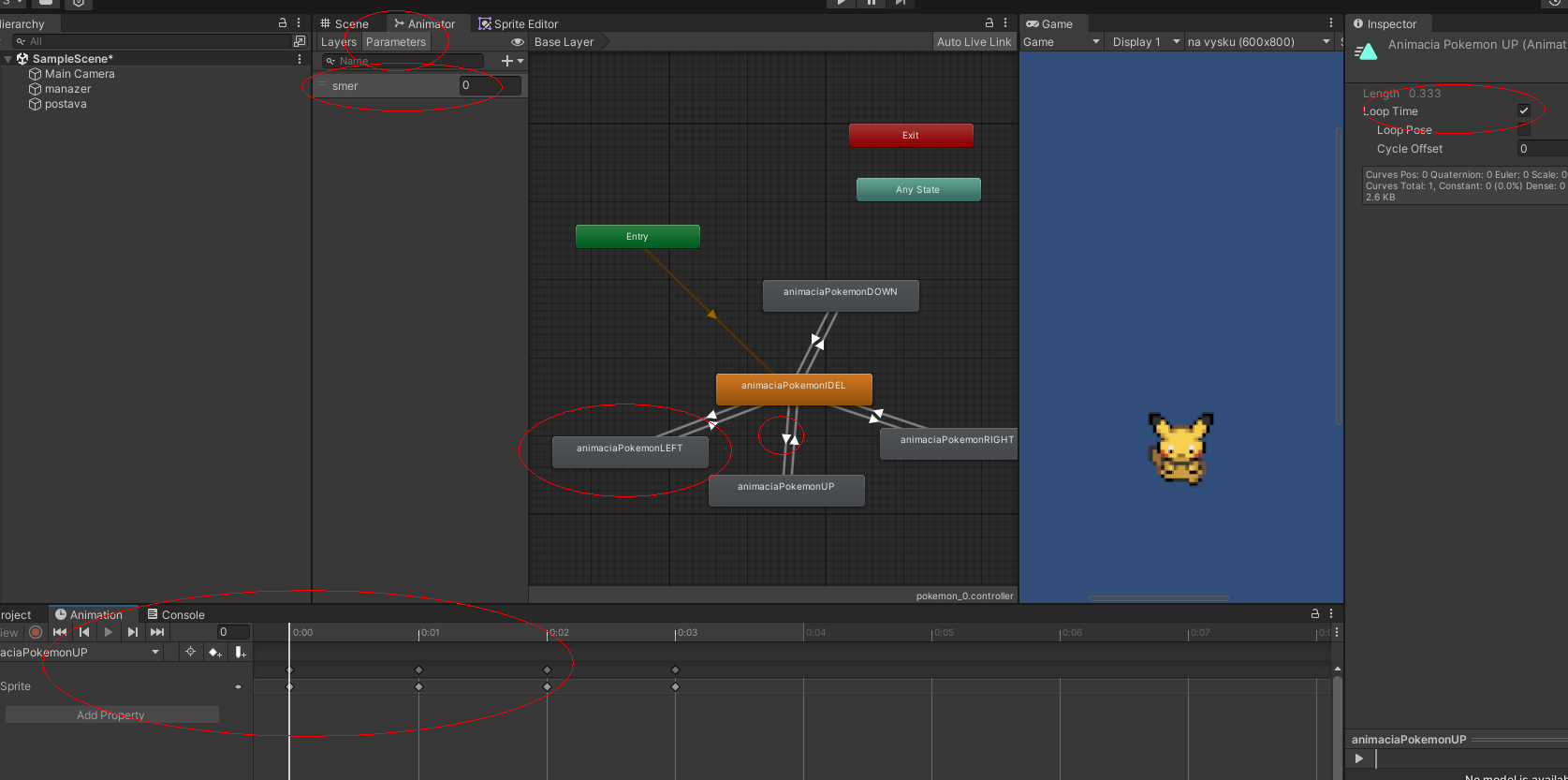
Aby sa animácia prehrala, musí byť v scéne teda objekt, ktorý ma komponent Animator, v ňom je kontrolér animácie, ktorý riadi prehrávanie animácii. V kontroléry môže byť viacero animácii. Animácie sa editujú v okne Window/Animation/Animation.

Cez kód vieme riadiť animáciu pomocou Parametrov. Vytvorí sa podľa typu nejaký parameter, potom sa klikne na väzbu/Transition a v okne Inspectora sa v časti Condition/podmienky určí, na akú hodnotu má väzba zareagovať. Napr. Bool keď bude True, tak sa vykoná prechod medzi animáciami. V skripte sa potom získa komponent Animator a cez metódu SetBool(„NAZOV“, hodnota) sa nastaví daný parameter

Animator animatorCharakteru = postavaGameObject.GetComponent<Animator>();

animatorCharakteru.SetBool("spustenie", true);





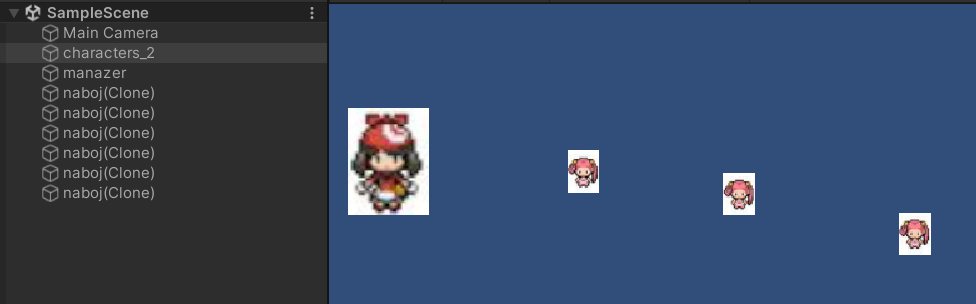
*Úloha: Vytvorte štyri animácie pre pohyb hore/dole/vľavo/vpravo. Animácie spúšťajte až po stlačení.*

**AKCIA/OVLÁDANIE OD HRÁČA**

Okrem UI vie hráč reagovať s hrou aj cez klávesnicu/myš/gamepad. Napr. vie riadiť pohyb objektu. Zachytávanie stlačenia klávesu sa robí v metóde Update(), prepočet pohybu v metóde FixedUpdate(). Input.GetKey("up") identifikuje názov (nastavenie projektu / Input) a vracia True, pokiaľ sa drží priradený kláves. Input.GetKeyDown(KeyCode.M) identifikuje ID klávesu a vracia True len pri prvom stlačení. Input.GetButtonDown ("Fire1")) je vstup, ktorý zachytáva stlačenie nie len na klávesnici, ale aj na myši či gamepade. Na pohyb WSAD sa odporúča použiť metódu Input.GetAxis ("Horizontal"), pretože obsluhuje klávesnicu i gamepad a zároveň spája pohyb vľavo a vpravo do jednej hodnoty. Vľavo\_stred\_vpravo je definované hodnotami -1 \_ 0 \_ 1 .

Príklad:

Vytvorí sa objekt manazer so skriptom, postava s RB2D+skriptom a prefab naboja so skriptom+RB2D. Postave sa da bud Kinematic alebo Dynamic s Gravity=0, freezRotationZ=0.



MANAZER.CS

[SerializeField] // zobrazuje PRIVATE v Inspector okne

GameObject naboj; // objekt pre prefab naboja

[SerializeField]

GameObject charakter; // objekt pre objekt postavy

void Update()

{

if (Input.GetButtonDown("Fire1")) // pokial sa stlaci L\_CTRL...

{

var nieco = Instantiate(naboj, charakter.transform.position, Quaternion.identity);

Destroy(nieco, 2f); // automaticke znicenie objektu po 2sek.

}

}

NABOJ.CS

void Start()

{

this.GetComponent<Rigidbody2D>().AddForce(Vector2.right\*1000);

// pridanie Sily do RB v smere doPrava o velkosti \*1000

}

POSTAVA.CS

Vector2 pos; // "premenna" pre poziciu X/Y

Rigidbody2D rb; // "premenna" pre RigidBody2D

void Start()

{

rb = this.GetComponent<Rigidbody2D>(); // priradenie mojho RB2D

}

void Update()

{

pos.x = Input.GetAxis("Horizontal"); // zapis posunu v smere X

pos.y = Input.GetAxis("Vertical"); // zapis posunu v smere Y

}

private void FixedUpdate()

{

//transform.Translate(pos \* Time.deltaTime \* 20); // pouzit na pohyb, kde nie je RB

rb.velocity = pos \* Time.fixedDeltaTime \* 200; // uprava zrychlenia

//rb.AddForce(pos \* Time.fixedDeltaTime \* 200); // uprava posobenia sily na objekt

//rb.MovePosition(rb.position + pos \* Time.fixedDeltaTime \* 20); // uprava pozicie

}

Na prácu s myšou sa používajú metódy: Input.GetMouseButtonDown(0) – deteguje stlačenie ĽTP; Cursor.visible – nastavuje zobrazenie kurzora; Input.mousePosition – získanie súradnice kurzora.

Na detegovanie toho, na čo sa v scéne kliklo, sa využíva RayCast/hit. Pri detekcii kliknutia na objekt treba brať v úvahu, že objekt sa nachádza v priestore/scéne/svete a kliká sa na obrazovku/kameru. Preto sa využíva metóda, kde sa vygeneruje lúč, ktorý smeruje od pozície kurzora na obrazovke smerom do priestoru/sveta. Následne sa otestuje, či došlo ku kolízii lúča s objektom (ten musí mať priradený collider – všetko je to podmienené zo systému Physics).

Príklad:

Tento kód sa dá do manazera a prida sa mu prefab terca. Terc obsahuje TAG a COLLIDER2D!

[SerializeField] // zobrazuje PRIVATE v Inspector okne

GameObject terc; // sukromny objekt pre prefab tercu

int bodov = 0; // globalna premenna pre body/skore

float cas = 0; // premenne pre pracu s intervalom

void Update()

{

cas += Time.deltaTime; // spocitavanie casu snimkov

if (cas > 0.5f) // ak presiel cas 0,5 sekundy...

{

Instantiate(terc, new Vector2(Random.Range(-9f, 9f), Random.Range(-4f, 4f)), Quaternion.identity);

// vytvori sa objekt na nahodnu suradnicu -9 az 9 v osi X a -4 az 4 v osi Y

cas = 0; // vyresetovanie pocitania casu intervalu

}

if (Input.GetMouseButtonDown(0)) // ak bolo stlacene LTM

{

RaycastHit2D hit = Physics2D.Raycast(Camera.main.ScreenToWorldPoint( Input.mousePosition), Vector2.zero);

// vytvori sa HIT, ktory bude obsahovat udaje o vsetkych koliziach z pozicie kurzora mysi do SVETA

if (hit.collider != null) // ak sa nieco zasiahlo...

{

if (hit.transform.tag == "tag") // ak tag zasiahnutej veci je TAG

{

Destroy(hit.transform.gameObject); // dany objekt znic

bodov++; // pridaj body

print(bodov); // vypis body

}

}

}

}

**SCÉNY A SINGLETON**

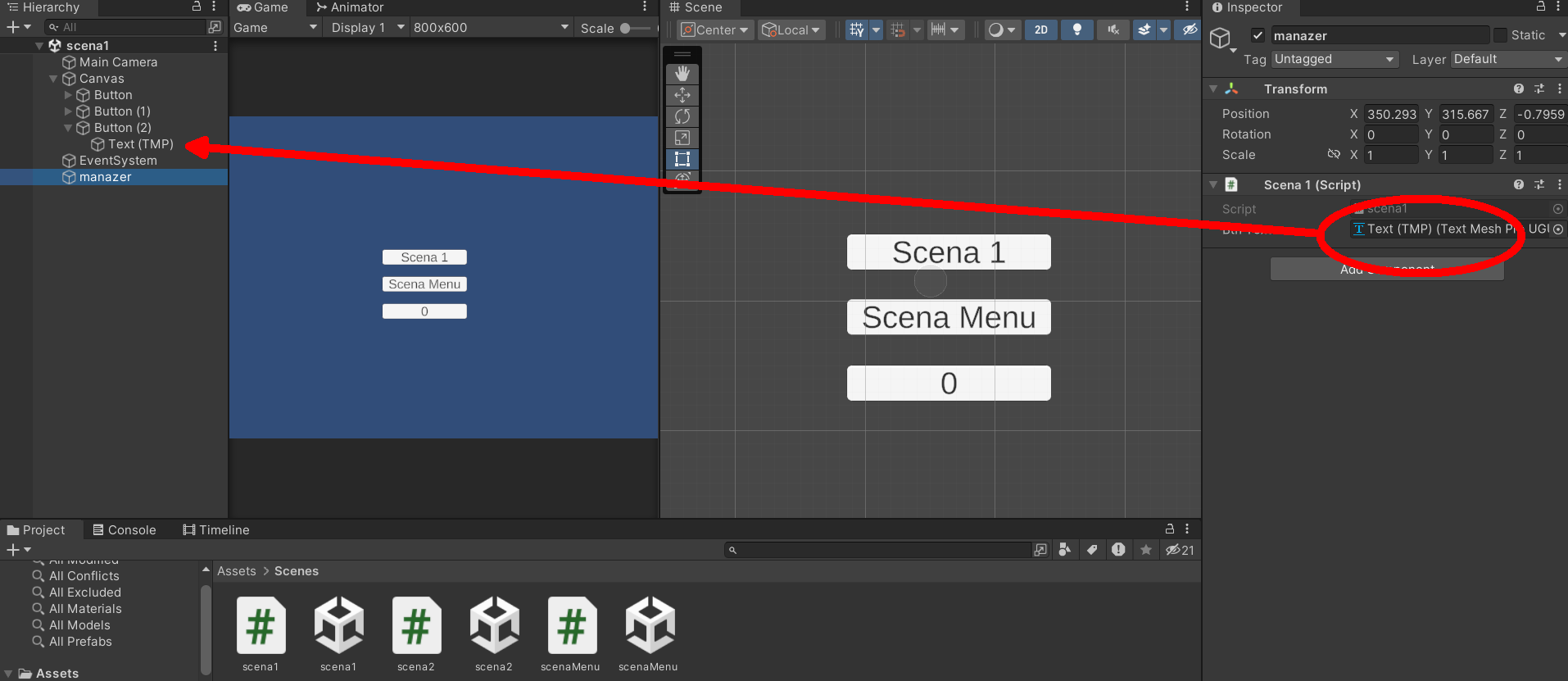
Scéna tvorí svet hry. Je vhodné rozdeľovať hru do viacerých scén kvôli prehľadnosti a tiež kvôli optimalizácii. Napr. ak je každý level v jednej scéne, môže sa po prechode do nového levelu uvoľniť pamäť obsahujúca objekty z prvého levelu. Scéna môže tvoriť aj Menu a môže byť kedykoľvek vyvolaná a navrátená späť do pôvodnej scény.

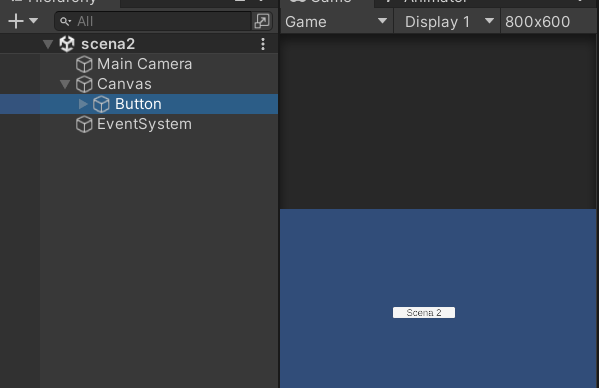
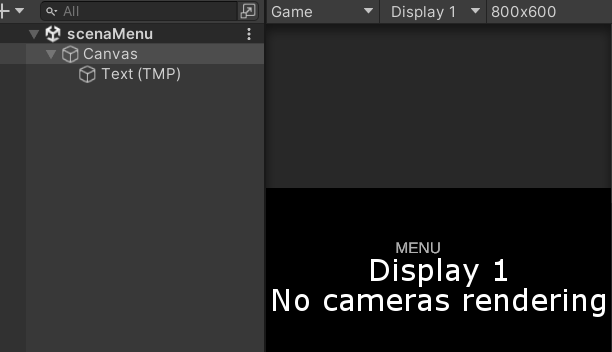
Pre prácu so scénou používame menný priestor UnityEngine.SceneManagement. Novú senu vytvoríme PTM v Assets okne a z voľby sa vyberie Create/Scene. Aby boli scény zahrnuté do projektu je nutné ich v File/BuildSettings pridať (drag&drop) do hornej časti. Scéna s indexom 0 je hlavná.

Príklad:

Nasledujúci príklad obsahuje tri scény (scena1, scena2, scenaMenu). V prvej scéne sú tri tlačidlá a objekt Manazer so skriptom. Tlačidlá majú pridelené metódy z tohto objektu. V scena2 je jedno tlačidlo s kamerou, plátnom a eventSystémom. Tlačidlo volá metódu po kliknutí. V scenaMenu je len plátno s TMP textom. Kamera a EventSystém sú odstránené, aby sa neduplikovali. Na objekte Textu je skript.

Po kliknutí na tlačidlo Scena 1 sa prepne scéna na druhú. Tam po stlačení tlačidla sa prepne scéna na prvú. Vidno, že sa obsah scény vyresetoval. Po kliknutí na tlačidlo Scena Menu sa pridá do scény druhá, ktorá zobrazí text MENU. Tá v sebe spustí skript, ktorý po 3 sekundách zavolá metódu, ktorá zmaže scénu s Menu. V okne Hierarchy vidno, čo sa v danú chvíľu nachádza v pamäti. Tretie tlačidlo po kliknutí zvýši číslo napísane na ňom. Slúži na demonštráciu, že pri prepnutí do scény Menu sa obsah scény 1 zachová, no pri prechode do scény 2 a späť dôjde k vymazaniu a novému načítaniu.



Scena1.CS

using UnityEngine;

using UnityEngine.SceneManagement; // prilinkovanie priestoru pre SCENY

using TMPro; // prilinkovanie priestoru pre UI

public class scena1 : MonoBehaviour

{

public TMP\_Text btnText;

public void klikScena1()

{

SceneManager.LoadScene("scena2");

}

public void klikScenaMenu()

{

SceneManager.LoadScene("scenaMenu",LoadSceneMode.Additive);

}

int kliknuti = 0;

public void kliker()

{

kliknuti++;

print(kliknuti);

btnText.GetComponent<TMP\_Text>().text = kliknuti.ToString();

}

}

scena2.CS

using UnityEngine;

public class scena2 : MonoBehaviour

{

public void klikScena2()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("scena1");

}

}

scenaMenu.CS

using UnityEngine;  
using UnityEngine.SceneManagement;

public class scenaMenu : MonoBehaviour

{

void Start()

{

Invoke("prepnutie", 3f);

}

void prepnutie()

{

SceneManager.UnloadSceneAsync("scenaMenu");

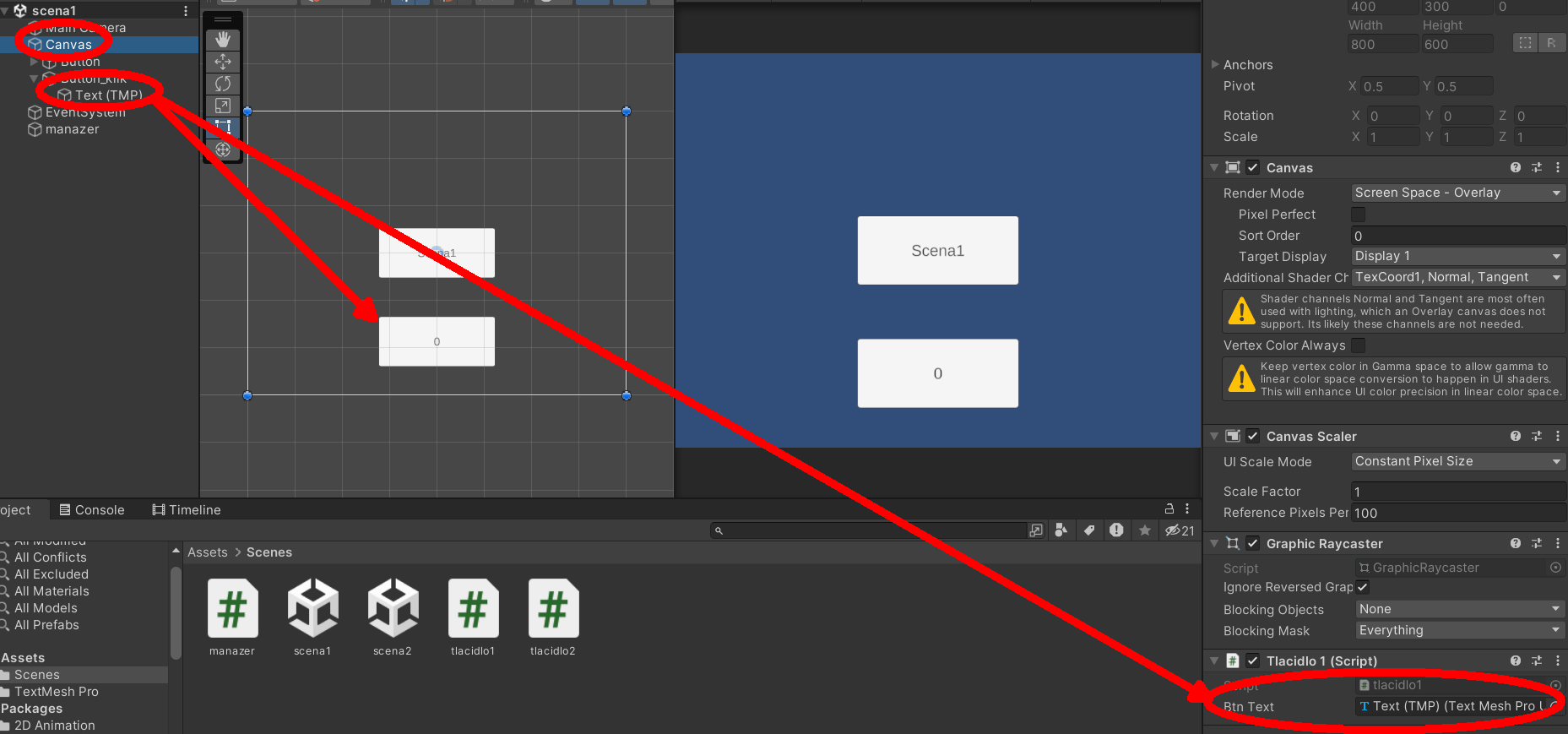
}

}

Ak chceme prenášať údaje z jednej scény do druhej, musíme ich umiestniť na objekt (prenáša sa objekt so svojími komponentami, ako napr. skriptom) a tento objekt vložiť do metódy DontDestroyOnLoad(). Pozor, môže nastať situácia pri prechode medzi scénami, že sa prenesie objekt do scény, kde daný objekt už je a teda dôjde ku duplicite. To môže nie len časom zaplniť pamäť ale aj spôsobiť chybné chovanie sa programu. Odporúča sa preto zmazať kópiu, pokiaľ sa inštancia != null.

Singleton (jedináčik) je objekt, ktorý sa vytvorí v triede a je typu tej istej triedy, čiže niečo také, ako keby som ja sám vytváral sám seba. Tým, že sa mu priradí vlastnosť Static, bude existovať len jeden (nedá sa vytvoriť inštancia) a po celý čas. Prístup k nemu je cez názov triedy. Takýto objekt sa častokrát používa ako manažér a býva presúvaný medzi scénami. Preto je vhodné testovať, či sa znovu nevytvára, ak už existuje, aby nebolo viacero kópii.

Príklad:



manazer.CS

using UnityEngine;

public class manazer : MonoBehaviour

{

public int kliknutia = 0; // premenna, ktora sa bude presuvat scenami

public static manazer jedinacik; // STATICKY objekt typu ROVNAKA TRIEDA==singleton

void Awake() // spusti sa este pred vytvorenim objektu

{

if (jedinacik != null) // ak uz existuje takyto singleton / objekt...

{

Destroy(gameObject); // ...tak tuto kopiu zmaz

}

else // ...inak...

{

jedinacik = this; // ...ak neexistuje, tak singletonu prirad tento objekt

DontDestroyOnLoad(gameObject); // udrzanie objektu pri prechode scenami

}

}

}

tlacidlo2.CS

using UnityEngine;

public class tlacidlo2 : MonoBehaviour

{

private void Start()

{

print("Bolo: " + manazer.jedinacik.kliknutia.ToString());

}

public void prepnutieSceny()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("scena1");

}

}

tlacidlo1.CS

using UnityEngine;

using TMPro; // prilinkovanie priestoru pre UI

using UnityEngine.SceneManagement; // pre Sceny

public class tlacidlo1 : MonoBehaviour

{

public TMP\_Text btnText;

public void prepnutieSceny()

{

SceneManager.LoadScene("scena2");

}

public void klikanie()

{

manazer.jedinacik.kliknutia++; // zo skriptu manazer vezmi singleton a...

btnText.GetComponent<TMP\_Text>().text =manazer.jedinacik.kliknutia.ToString();

}

private void Start()

{

btnText.GetComponent<TMP\_Text>().text =manazer.jedinacik.kliknutia.ToString();

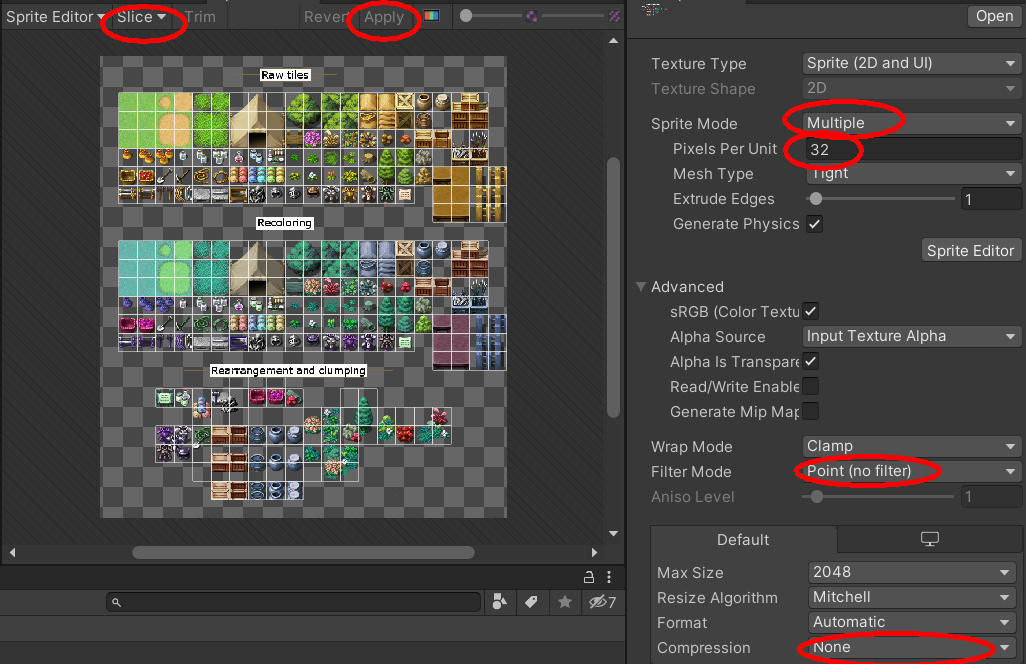
}

}

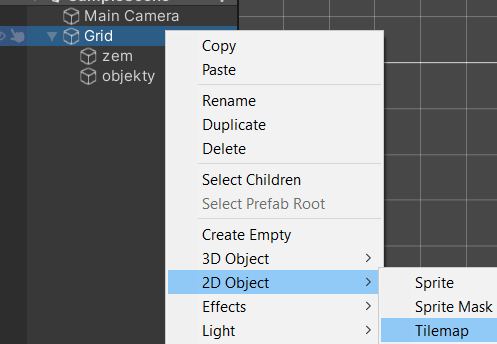
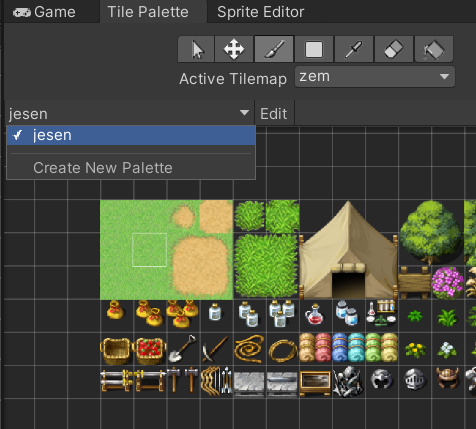
**TILED MAP**

Pri 2D hre je možné vytvárať scénu/mapu z grafických blokov, ktoré sa viackrát opakujú (tiles). Aby sa nemuseli opakovane vkladať, vytvoril sa systém TileMap, ktorý akoby kreslil mapu, pričom namiesto farieb používa obrázky. Takéto mapy sú bežné u top/down hier a plošinoviek.

Príklad:

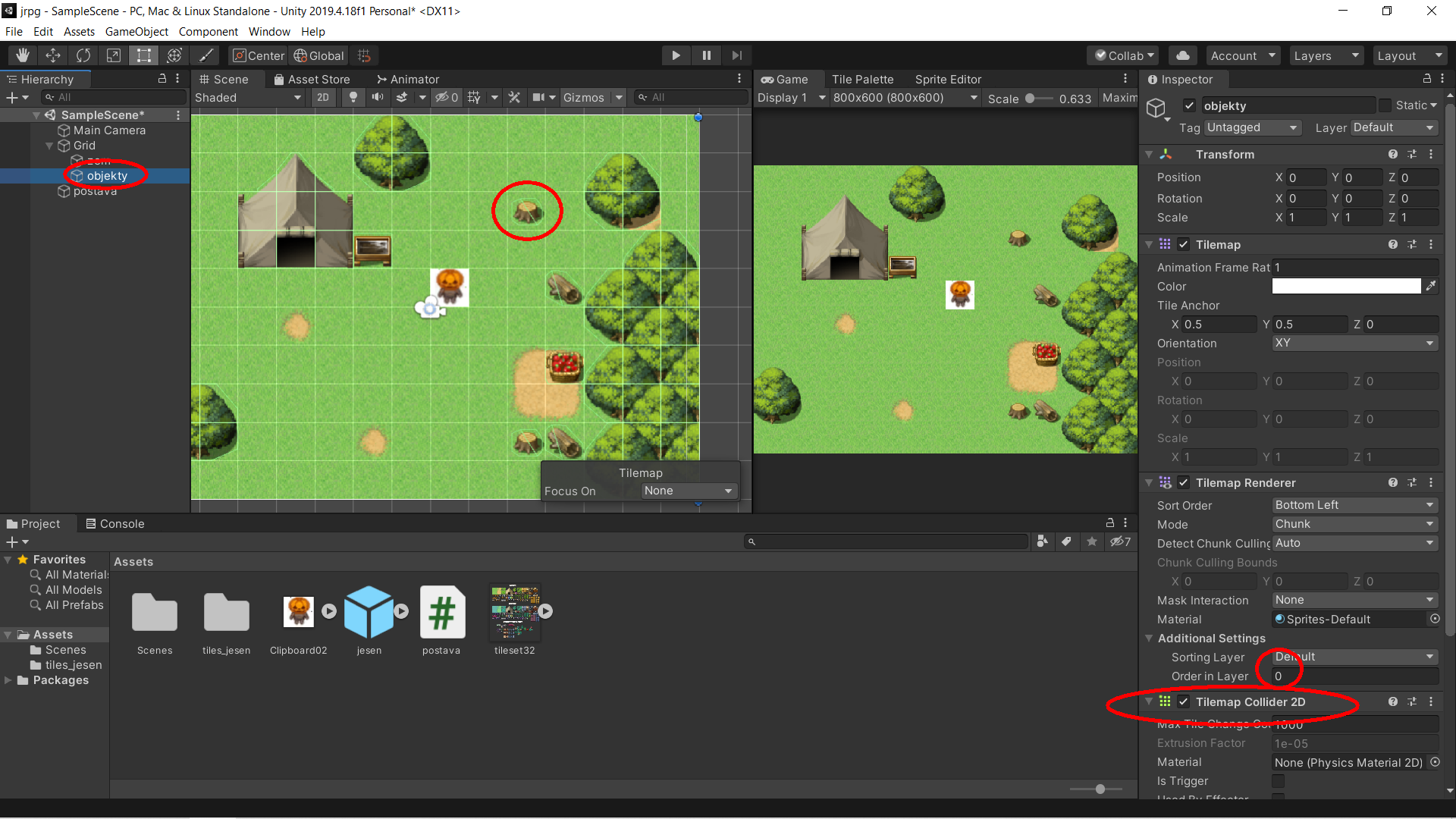


Do projektu sa vloží obrázok s graf.blokmi. Vo vlastnostiach sa nastaví ako Multiple, zvolí sa počet pixelov, ktoré odpovedajú jednému bloku, vyplne sa kompresia. V Sprite Editore sa naseká podľa veľkosti. V Hierarchy sa vloží viacero Tilemap - súvisiace bloky na mapke (napr. zem, tráva, stromy). Podobne ako pri UI sa vytvorí Canva, tak pri Tilemap sa vytvorí Grid.

V okne TilePallete sa vytvorí nová paleta a do nej sa cez drag&drop vloží obrázok s blokmi (môžu sa aj jendotlivo vložiť bloky). Horné menu slúži na umiestňovanie (štetec, guma, posun, ...). Takto sa vytvorí mapa. Pokiaľ nektoré veci majú byť viac vpredu ako iné, treba im nastaviť Order Layer. Taktiež je možné nastaviť Collider. Ak sa použije Tilemap collider, tak sa ohraničia všetky objekty vo zvolenom tilemap objekte.

Pre tento typ hier je tipický posun postavy po blokoch/bunkách. Treba ho teda meniť o veľkosť 1.0, keďže bunky majú veľkosť 1. Vhodné je využiť metódu MoveTowards, alebo Lerp.



postava.CS

bool pohybujeSa = false;

Vector3 kamMaIst;

private void Start()

{

kamMaIst = transform.position;

}

void Update()

{

float hor = Input.GetAxis("Horizontal");

float ver = Input.GetAxis("Vertical");

if(hor > 0 && pohybujeSa == false)

{

pohybujeSa = true;

kamMaIst = transform.position + Vector3.right;

}

else if (hor < 0 && pohybujeSa == false)

{

pohybujeSa = true;

kamMaIst = transform.position + Vector3.left;

}

if (ver > 0 && pohybujeSa == false)

{

pohybujeSa = true;

kamMaIst = transform.position + Vector3.up;

}

else if (ver < 0 && pohybujeSa == false)

{

pohybujeSa = true;

kamMaIst = transform.position + Vector3.down;

}

transform.position = Vector3.MoveTowards(transform.position, kamMaIst, Time.deltaTime\*3);

if(Vector3.Distance(transform.position, kamMaIst) < 0.01f && pohybujeSa==true)

{

transform.position = kamMaIst;

pohybujeSa = false;

}

}

**PARTICLE SYSTEM**

**MATERIAL A SHADER**