**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.

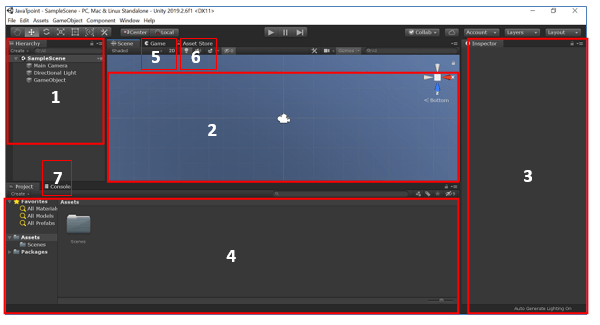
Čitateľnosť zlepšil príchod vyšších programovacích jazykov (napr. C). Tie sú písané príkazmi ako napr. while (pokiaľ), break (preruš) a pod. Takýto jazyk patrí do skupiny tzv. štruktúrovaných. Pre ešte ľahšie/názornejšie programovanie vznikli objektovo orientované jazyky (napr. C++/C#/Java), keďže všetko 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>()***

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

}

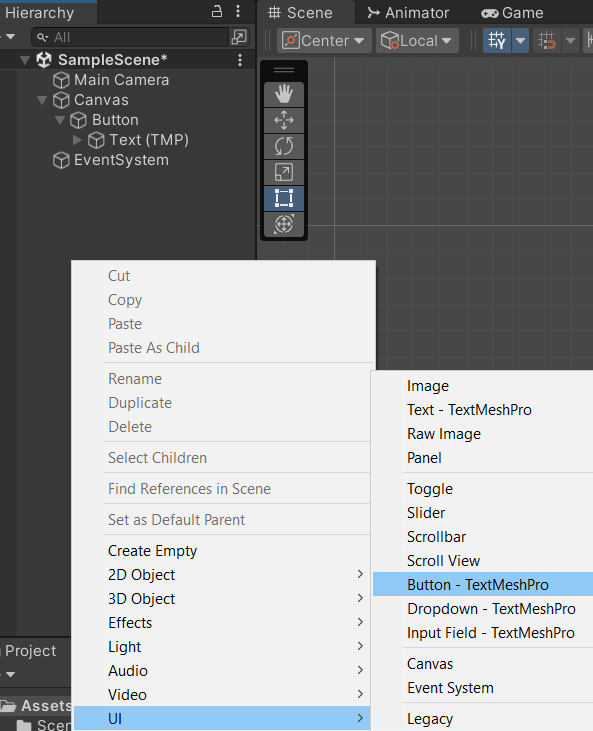
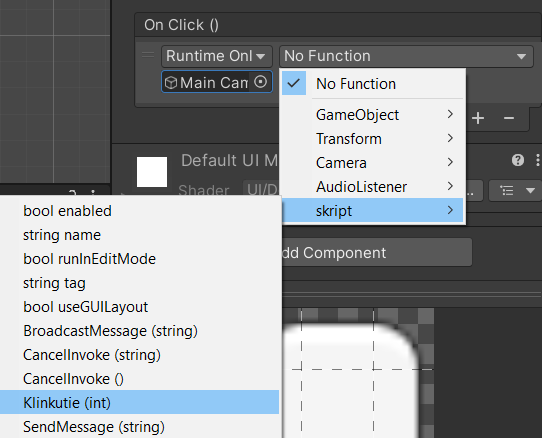
print("Najvacsi trpaslik mal velkost " + najvacsi); // vypis

**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

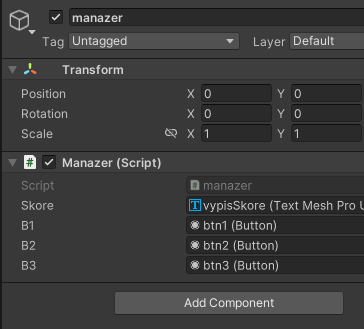
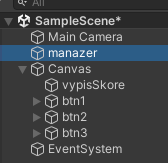
print("ERROR:" + poradie); // nenastane

// 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.*

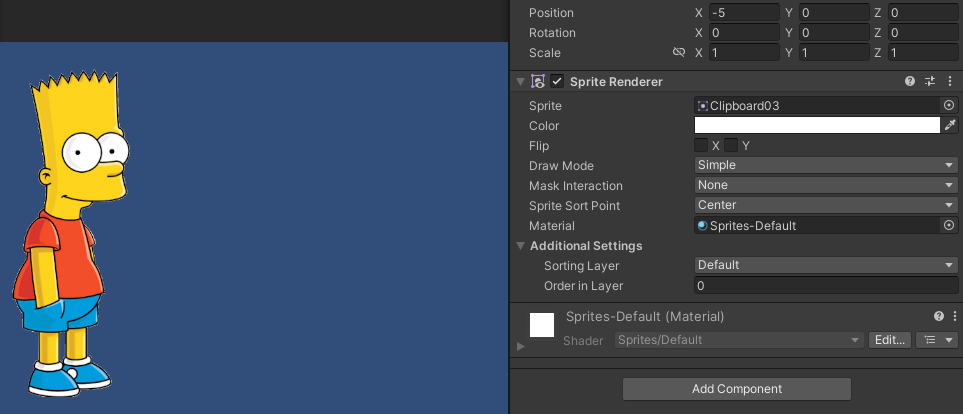
**Time.deltaTime (interval)**

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 objektu na iný sa používa „kolízia“. 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 na RB. Treba rozlišovať 2D/3D objekty. Všetky spomenuté komponenty a metódy udalostí majú verziu 2D aj 3D (napr. RB / RB2D).

Pokiaľ chceme zachytiť nastatie kolízie, ale neovplyvniť „spúšťač“ (napr. chceme zareagovať na dotyk s mincou, cez ktorú skáčeme, 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, ktorý chceme zachytiť, zapneme vlastnosť IsTrigger v Collidery. (nie na oboch). Tým sa stane akoby nehmotný.

*OnCollisionEnter(Collision other)*

IsTrigger nesmie byť zvolený. Môže sa zmeniť v RB2D BODYTYPE na KINEMATIC, čím sa odignoruje pôsobenie fyziky. Constrains sa môže freez\_nuť. V RB2D sa zmení USE\_FULL\_KIN\*\*\* na CHECK.

Príklad:

private void OnCollisionEnter2D(Collision2D col)

{

if (col.gameObject.tag == "minca")

{

skym.gameObject.active = false; // skrytie objektu, nie jeho odstranenie!

}

}

private int body = 0;

private void OnTriggerEnter(Collider col)

{

if(col.tag == "minca")

{

body++;

print(body);

Destroy(col.gameObject);

}

}

// Uloha na vygenerovanie novej mince po zobrati starej

**PREFAB**

// Uloha na vyskusanie prefabu a v nom naprogramovat RANDOM rotaciu mince (rychlost,smer)