Vežba 4

Planiranje projekta i osnovno kretanje aktera u video igri

- Planiranje projekta
- Kreiranje menadžera igre
- Kreiranje logike za pomeranje aktera

Cilj i opis vežbe:

Studenti se u ovoj vežbi uvode u osnove mehanike igre 3D runner. Vežba opisuje način planiranja projekta. Korišćenjem klasa, studenti uče da kreiraju osnovnu logiku za menadžer igre koji kontroliše ostale elemente, kao i za osnovno kretanje glavnog aktera.

4.1 Opis projekta

3D Runner je igra, u kojoj glavni karakter ima za cilj da izbegne prepreke, na beskonačnom putu.

U našem slučaju, igrač će imati mogućnost da se pomera levo i desno, kao i da skoči. Svaka igra igraču daje 3 života. Gubitak poslednjeg života označava kraj igre.

4.2 Definisanje elemenata igre

Pre početka tehničke realizacije jednog projekta, neophodno je da se definišu svi njegovi elementi. Elementi runner-a kojeg ćemo raditi, mogu se podeliti na:

Menadžer

Koristi se za organizaciju svih elemenata igre. Povezuje glavnog aktera, pomeranje terena. Takođe, zadužen je i za praćenje stanja igre, gubljenje života i bodovanje.

Glavni akter

Predstavlja karakter kojim ćemo upravljati. Njegov zadatak će biti da računa pozicije pri pomeranju glavnog karaktera levo i desno. Sam akter neće se kretati unapred, već će se ~~p~~ut kretati ka njemu.

Menadžer puta

Organizuje pomeranje elemenata puta, sa preprekama i bonusima koji se nalaze na njemu.

Menadžer UIa

Služi za kontrolu svih elemenata User interface-a u igri. Pomoću njega će se zadavati zahtev za pokratanje igre, ispisivanje rezultata i života.

Detekcija prepeka

Koristi se detekciju sudaranja prepeke i heroja. U slučaju sudaranja, prepeka će javiti menadžeru da je došlo do sudaranja, što uzrokuje i gubitak života.

Detekcija bonusa

Koristi se detekciju sudaranja bonusa i heroja. U slučaju sudaranja, bonus će javiti menadžeru da je došlo do sudaranja, što uzrokuje i povećavanje bodovanja.

4.3 Kreiranje menadžera

Menadžer predstavlja najbitniji element igre. Od njega počinje da se razvija cela logika ostalih elemenata.

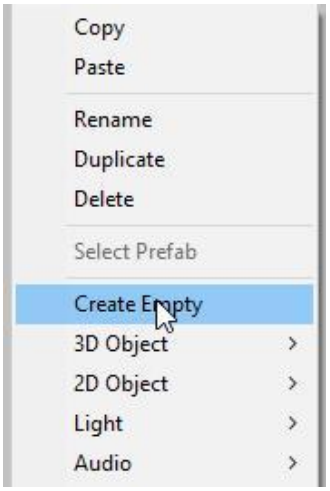
Napravimo klasu „GameManager“ koja će predstavljati glavnog menadžera.

Klasu „GameManager“, sada trebamo dodeliti nekom objektu na sceni.

Do sada smo dodavali u projekat isključivo vizuelne objekte, dok u ovom slučaju trebamo dodati praza objekat, čija će namena samo biti da sadrži klasu za menadžer.

Da bismo dodali objekat koji nije vidljiv na sceni, uradimo sledeće:

Desnim tasterom miša kliknimo na panel **Hierarchy**, a zatim iz padajuće liste izaberimo **Create Empty**.



Slika 4.1 – Kreiranje praznog objekta na sceni

Kada selektujemo ovaj objekat, možemo videti da on na sebi u prozoru Inspector ma samo komponentu Transform.

Promenimo naziv novog objekta u „GameManager“, a zatim, korišćenjem opcije Drag & Drop, dodelimo klasu „GameManager“.

Da bismo pisali logiku u okviru klase, kliknimo duplim tasterom miša na klasu u projektu. Jako bitna stvar na početku, je da uspostavimo u kojim sve stanjima naša igra može da bude. Uzevši u obzir celokupnu problematiku ove igre, možemo izdvojiti četiri različita stanja, i to:

Start

Početno stanje igre. Ono se poziva pri pokretanju, a pre nego što je sam akter krene u akciju trčanja.

Running

Stanje trčanja, odnosno stanje u kome je igrač dok traje sama igra. U ovo stanje je moguće doći iz stanja **Start**, ali i iz stanja kada igrač izgubi jedan život koji nije i poslednji, odnosno iz stanja **Dead**.

Dead

Stanje kada se igrač sudari sa nekom preprekom. U ovom slučaju, dolazi do gubitka života. U ovo stanje je moguće doći iz stanja **Running**.

GameOver

Stanje kada se izgube sva 3 života.

Da bismo definisali najlakše ova stanja igre, koristićemo **enum**.

Definišimo u klasi GameManager nabrojivu listu GameState:

```
public enum GameState
{
    start,
    running,
    dead,
    gameover
}
```

Sada smo definisali sva stanja igre.

Sledeći korak je da definišemo i promenljivu koja će koristiti ova stanja.

S obzirom da promenljiva koja označava stanja treba da bude dostupna svim klasama, kao i da sve instance dele istu promenljivu, napravimo da promenljiva koja će određivati stanja igre, bude statitčka.

To možemo uraditi sledećom naredbom:

```
public static GameState gameState;
```

Ovo je prva faza razvijanja GameManager-a, i za sada nam je ona dovoljna.
Kasnije, u ovoj klasi razvijaćemo dalju logiku.

4.4 Glavni akter

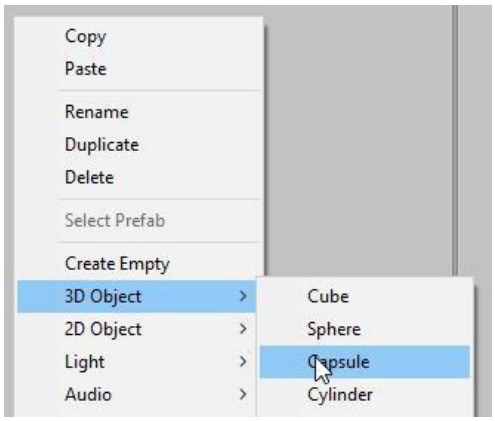
Sledeći korak, je da kreiramo glavnog aktera na sceni.

Na početku, kada smo nabrajali elemente, rekli smo da će logika glavnog aktera biti njegovo pomeranje.

Napravimo klasu **PlayerController**, u kojoj ćemo razvijati logiku za pomeranje.

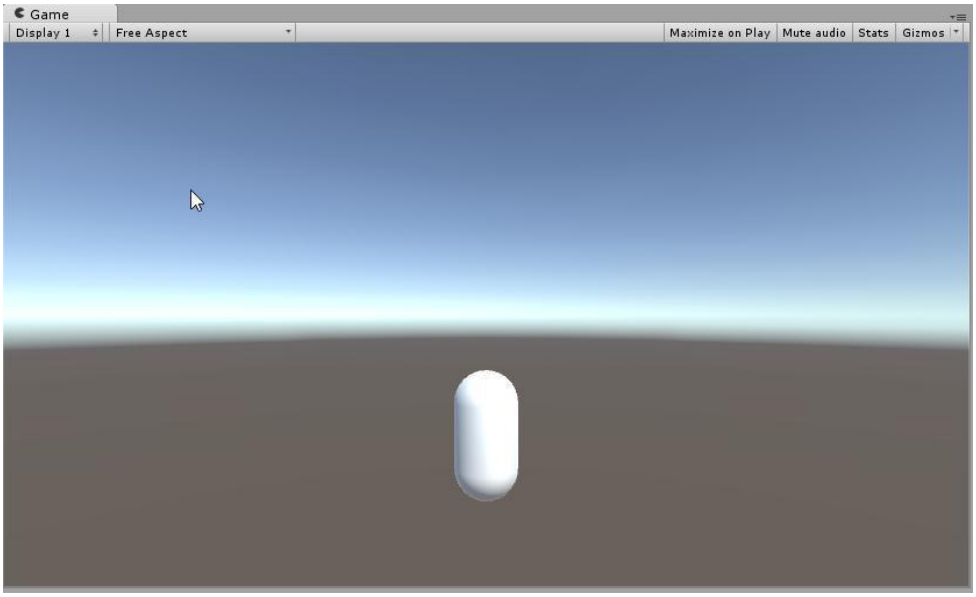
Potom, napravimo i objekat koji ćemo pomerati. Za sada, dok razvijamo logiku, koristićemo neki od primitivnih objekata koje nam nudi Unity.

Napravimo kapsulu kao objekat, koja će nam predstavljati aktera u toku razvoja projekta. Da bismo to uradili, kliknimo desnim tasterom miša na panel **Hierarchy** i iz podmenija 3D Object izaberimo **Capsule**.



Slika 4.2 – Dodavanje 3D objekata

Sada, pozicionirajmo kameru iza objekta kapsule, da izgleda kao na slici, vodeći računa da Z osa kamare i Z osa objekta budu u istom pravcu.

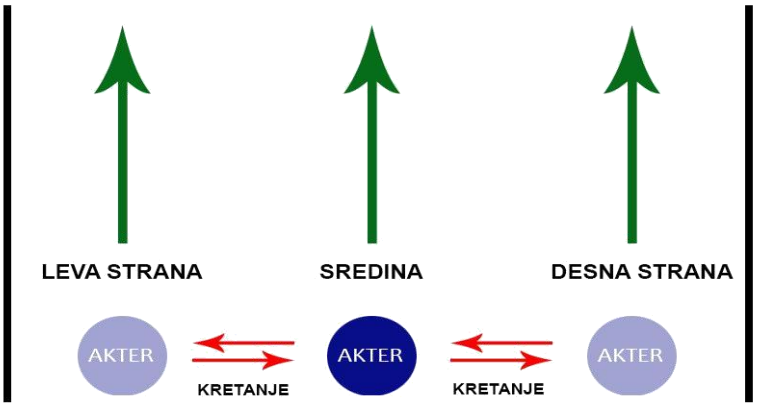


Slika 4.3 – Pozicija kamere i Game panel

Potom, korišćenjem opcije Drag & Drop, prevucimo novokreiranu klasu **PlayerController** na objekat koji smo napravili.

Otvorimo klasu u MonoDeveloper-u (ili Visual Studiju), kako bismo pisali logiku za kretanje. Setimo se, da smo definisali da akter može da se kreće levo i desno. U stvari, on će na svom putu imati 3 predefinisane putanje kretanja: levu stranu, središnju (početnu), desnu stranu.

Početna putanja će se nalaziti na sredini. Iz te putanje će moći da se pomeri u levu ili desnu stranu. Dok će iz bočnih putanja, moći da se vrati jedino u središnju putanju. Za lakše razumevanje, pogledamo sliku:



Slika 4.4 – Definisanje kretanja aktera - skica

Objekat koji ćemo pomerati (kapsula), je postavljen tako da će se njegovo pomeranje na levu i desnu stranu puta, zapravo izvršavati isključivo pomeranjem po X osi.

Takođe, pošto je inicijalno postavljen na nulte koordinate, to će mu predstavljati i centralnu putanju. Odnosno, kretanje levom putanjom će mu pomerati objekat u minus po X osi, odnosno za desnu stranu u plus po X osi.

Kao ulaz sa tastature za pomeranje objekta, koristićemo strelicu na levo i strelicu na desno.

Napišimo sada funkciju koja će određivati smer pomeranja. Funkcija neće imati ulazne parametre, a kao povratnu vrednost ćemo vratiti pozitivnu ili negativnu vrednost, u zavisnosti da li želimo pomeranje na levo, ili pomeranje na desno (odnosno da li je kliknuta strelica na levo ili strelica na desno).

Definišimo funkciju na sledeći način:

```
public int GetInput ()
{

}
```

Zatim napravimo promenljivu **_direction** unutar funkcije, koja će određivati pozitivan ili negativan smer.Podrazumevani smer je 0 . Promenljiva **_direction** napisana unutar funkcije će imati doseg isključivo unutar same funkcije.

```
public int GetInput ()
{
    int direction = 0;
}
```

Dodajmo sada logiku, koja će u slučaju da je pritisnut taster na levo, promenljivoj **_direction** dodeljivati negativan broj, odnosno pozitivan u slučaju pritiska tastera desno. I na kraju, kao povratnu vrednost funkcije, vratimo promenljivu **_direction.**

```
public int GetInput ()
{
    int _direction = 0;

    if (Input.GetKeyDown (KeyCode.LeftArrow))
        _direction = -1;
    else if (Input.GetKeyDown (KeyCode.RightArrow))
        _direction = 1;

    return _direction;
}
```

Sada, ovu funkciju možemo koristiti, kako bismo dobili rezultat smera.

Sledeći korak je da predvidimo koliki će biti pomeraj karaktera po X osi za jedan pritisak dugmeta u stranu i njegov limit pri pomeranju.

Pošto su pojedinačne koordinate objekta tipa float, napravimo da pomeraj po X osi takođe bude tipa float. Promenljivu ćemo nazvati **step** i stavićemo da bude javna, kako bi mogli da joj pristupamo i iz editora. Za početnu vrednost æemo joj dodeliti vrednost 3, na osnovu subjektivne procene:

```
public float step = 3f;
```

Takođe, dodelimo joj i **limit**, koji će isto biti tipa float. Limit ne mora da bude javna promenljiva, obzirom da će po ranije osmišljenoj logici, biti vrednosti jednog koraka pomeraja.

```
float limit;
```

Pošto je limit vrednosti jednog stepena pomeraja, u Start funkciji možemo dodeliti promenljivoj **limit**, vrednost koraka, na sledeæi način:

```
void Start ()
{
    limit = step;
}
```

Napišimo sada i funkciju koja će izvršavati pomeraj objekta. Funkcija će kao vrednost dobiti smer, dok neće imati povratne vrednosti. Ona će biti privatna.

Definišimo funkciju za pomeranje na sledeći način:

```
private void Move (int dir)
{

}
```

Zatim, korišćenjem trenutne vrednosti **X koordinate**, **pravca** i **koraka**, izračunajmo potencijalnu vrednost X koordinate. Za potrebu tog računa, napravićemo novu promenljivu tipa float i nazvati je **posX**, unutar funkcije **Move**:

```
private void Move (int dir)
{
    float posX = transform.position.x + dir * step;
}
```

Ispitajmo sada, da li je potencijalna vrednost pomeranja, odnosno promenljiva posX, u opsegu limita. Limit je sa jedne leve strane negativna vrednost koraka, dok je sa desne strane pozitivna vrednost koraka. Logiku možemo napisati na sledeći način:

```
private void Move (int dir)
{
    float posX = transform.position.x + dir * step;

    if (posX <= limit && posX >= -limit)
    {

    }
}
```

Ukoliko je uslov za limit ispunjen, dodelimo objektu koji pomeramo, vrednost nove izračunate pozicije. Obzirom da objekat pomeramo samo po X osi, onda ostale vrednosti koordinate uzmimo od trenutne vrednosti koordinata objekta:

```
private void Move (int dir)
{
    float posX = transform.position.x + dir * step;

    if (posX <= limit && posX >= -limit)
    {
        transform.position = new Vector3 (posX, transform.position.y, transform.position.z);
    }
}
```

Sada, kada imamo funkcije za računanje smera u odnosu na ulazni parametar i funkciju za pomeranje objekta, potrebno je da sada te funkcije pozivamo svakog frejma, kako bi se željeno pomeranje i izvršilo.

Za izvršavanje funkcija svakog frejma, koristićemo funkciju **Update**, o kojoj je bilo reči u prethodnim vežbama.

Kao prvu operaciju, određivaćemo smer na osnovu funkcije **GetInput**, koju smo ranije napravili. Rezultat te funkcije, smestićemo u promenljivu **direction** tipa int, a koja æe biti deklarisana unutar funkcije Update. Uradimo to na sledeći način:

```
void Update ()
{
    int direction = GetInput ();
}
```

Zatim, ćemo pozvati funkciju za pomeranje Move, koja kao parametar očekuje smer. Obzirom da imamo izračunat smer u promenljivoj **direction**, prosledićemo tu promenljivu kao parametar:

```
void Update ()
{
    int direction = GetInput ();
    Move (direction);
}
```

Klasa **PlayerController** bi sada trebalo da izgleda ovako**:**

PlayerController ▸ No selection

```
1  using UnityEngine;
2
3  public class PlayerController : MonoBehaviour
4  {
5      public float step = 3f;
6      float limit;
7
8
9      void Start ()
10     {
11         limit = step;
12     }
13
14     void Update ()
15     {
16         int direction = GetInput ();
17         Move (direction);
18     }
19
20     public int GetInput ()
21     {
22         int _direction = 0;
23
24         if (Input.GetKeyDown (KeyCode.LeftArrow))
25             _direction = -1;
26         else if (Input.GetKeyDown (KeyCode.RightArrow))
27             _direction = 1;
28
29         return _direction;
30     }
31
32     private void Move (int dir)
33     {
34         float posX = transform.position.x + dir * step;
35
36         if (posX <= limit && posX >= -limit)
37         {
38             transform.position = new Vector3 (posX, transform.position.y, transform.position.z);
39         }
40     }
41
42 }
```

Slika 4.5 – Izgled PlayerController-a sa finalnim kodom

Proverimo još jednom, da li je klasa **PlayerController** dodata na objekat koji želimo da pomeramo, kao i da li je objekat na koordinatama 0,0,0 odnosno na rotaciji 0,0,0. Pokrenimo projekat i testirajmo da li se objekat pomera posle pritisnutih strelica na levo i desno.

Zadatak:

Prateći upustva vežbe 4, napraviti osnovnu logiku za pomeranje aktera na sceni.