**Šolski center Novo mesto**

**Srednja elektro šola in tehniška gimnazija**

**Šegova ulica 112**

**8000 Novo mesto**

**Izdelava računalniške igre- Mope**

(Maturitetna seminarska naloga)

**Predmet:** Računalništvo

**Avtor:** Rok Kebelj

**Razred:** T4A

**Mentor:** Albert Zorko, univ. dipl. inž.

Novo mesto, april 2019

**POVZETEK**

V seminarski nalogi najprej izdelal preprostejšo računalniško igro, v kateri kontroliramo junaka in streljamo na sovražnike in tako pridobivamo točke. Ko sem izdelal igro sem predstavil kako sem uredil okolje v Unityu (igralca, nasprotnika, platforme…) in nato sem še predstavil kodo napisano v jeziku c# v Visual Studiu

***KAZALO VSEBINE***

[1 UVOD 5](#_Toc6252564)

[2 Teoretični del 6](#_Toc6252565)

[2.1 C# 6](#_Toc6252566)

[2.2 Objektno programiranje 6](#_Toc6252567)

[2.2.1 Unity 7](#_Toc6252569)

[2.2.2 Uvod v Unity 7](#_Toc6252570)

[2.2.3 Nadzorovanje dela(console okno) 8](#_Toc6252573)

[2.2.4 Komponente objektov 8](#_Toc6252575)

[2.2.5 Igralec (Player) 9](#_Toc6252580)

[2.2.6 Animacija 10](#_Toc6252581)

[2.2.7 Metode kode 12](#_Toc6252582)

[2.2.8 Trkalniki(Colliders) 12](#_Toc6252589)

[3 Praktični del- izdelava igre v Unity 14](#_Toc6252594)

[3.1 Dodajanje objektov v sceno 14](#_Toc6252595)

[3.2 Animacija objekta junak 15](#_Toc6252596)

[3.3 Gibanje junaka 16](#_Toc6252597)

[3.4 Kamera 22](#_Toc6252598)

[3.5 Sovražniki 24](#_Toc6252599)

[3.6 Izstrelek 27](#_Toc6252600)

[4 Zaključek 28](#_Toc6252601)

[5 Viri in literatura 28](#_Toc6252602)

[6 Citirana dela 28](#_Toc6252603)

***KAZALO SLIK***

[Slika 1 Primer nabora slik(»Sprite sheet«) za animacijo 9](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252677)

[Slika 2 Orodje za ustvarjanje animacije 10](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252678)

[Slika 3 orodje animator, primer prehodov med animaciji (mirovanje ter premikanje-hoja) 11](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252679)

[Slika 4 Škatlasti trkalnik 12](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252680)

[Slika 5 Krožni trkalnik 13](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252681)

[Slika 6 Slika 6 Poleg škatlastega trkalnika, je na objektu, na dnu dodan še trkalnik roba, saj je spodnji del objekta neprestano izpostavljen podlagi 13](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252682)

[Slika 7 Avtomatski trkalnik prepozna robove objekta, če le ta nima ozadja 13](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252683)

[Slika 8 Na levi so nastavitve, ter dodane komponente platforme. Na desno pa je vizualni prikaz objektov platforme in junaka 14](#_Toc6252684)

[Slika 9 V animatorju urejamo prehode med animacijami, ker je funkcija mirovanja privzeta animacija je njen parameter na začetku nastavljen na true. 15](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252685)

[Slika 10 koda razreda PlayerStateListener, nastavimo spremenljivke 16](#_Toc6252686)

[Slika 11 razred PlayerStateController, funkcija onStateCycle 16](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252687)

[Slika 12 razred PlayerStateController, funkcija onStateCycle 17](#_Toc6252688)

[Slika 13 Razred PlayerStateController, funkcija onStateChange 17](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252689)

[Slika 14 Razred PlayerStateController, funkcija onStateChange 18](#_Toc6252690)

[Slika 15 Razred PlayerStateController, funkcija onStateChange 18](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252691)

[Slika 16 Razred PlayerStateController , funkcija checkForValidStatePair 19](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252692)

[Slika 17 Razred PlayerStateController, funkcija checkForValidStatePair 20](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252693)

[Slika 18 orodje za nastavljanje vhodov 20](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252694)

[Slika 19 Razred PlayerStateListener 21](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252695)

[Slika 20 Razred PlayerStateListener, funkcija LateUpdate 21](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252696)

[Slika 21 Razred CameraController 22](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252697)

[Slika 22 Razred CameraController, funkciji LateUpdate in onStateCycle 22](#_Toc6252698)

[Slika 23 Razred CameraController, metoda trackPlayer 23](#_Toc6252699)

[Slika 24 Razred CameraController, funkcija stopTrackingPlayer 23](#_Toc6252700)

[Slika 25 Sovražnik 24](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252701)

[Slika 26 Razred EnemyController 24](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252702)

[Slika 27 Razred EnemyController 25](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252703)

[Slika 28 Razred EnemyController, funkciji updateVisualWalkOrientation in switchDirections 25](#_Toc6252704)

[Slika 29 Razred EnemyController, funkciji hitByPlayerBullet in hitByCrusher 26](#_Toc6252705)

[Slika 30 Razred PlayerColliderListener 26](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252706)

[Slika 31 Razred PlayerBulletController 27](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252707)

[Slika 32 Razred TakeDamageFromPlayerBullet 27](file:///D:\Users\Rok\AppData\Roaming\Skype\My%20Skype%20Received%20Files\SEMINARSKA%20ZA%20MATURO.docx#_Toc6252708)

# **UVOD**

Dandanes imamo na svetovnem spletu več milijonov različnih iger, saj preprostejše igre lahko dandanes naredi že vsakdo, ki ima motivacijo in je dovolj vztrajen. Izdelava preprostejše računalniške je kot nalašč za pridobivanje znanja o osnovah izdelave računalniške igre in s tem ko samo naredimo več preprostejših iger, s tem pridobivamo izkušnje in ustvarjalnost za izdelavo bolj kompleksnejših iger. Igro sem izdelal v programu Unity, ki ima zelo veliko orodij s katerimi si lahko pomagamo pri izdelavi igre, kodo pa sem urejal v Visual Studiu 2017. Unity ja najbolj popularn program za razvijanje iger tretjih oseb med razvijalci po celem svetu. . (1) Vso programsko kodo v Visual Studiu bom podrobneje opisal v seminarski, ter razložil kaj naredi posamezna metoda.

# **Teoretični del**

V teoretičnem delu bom povedal nekaj o jeziku c# v katerem sem pisal kodo, ter o objektnem programiranju. Opisal bom vse stvari, s katerimi sem delal v Unityu in kako sem naredil kakšno stvar, zato bom napisal še nekaj stvari o Unityu.

## C#

C# je razvila ekipa Microsoftovih strokovnjakov pod vodstvom Andersa Hejlsberga leta [2001](http://sl.wikipedia.org/wiki/1983), najprej so ga imenovali Cool (C kot objektno orientiran jezik), a se to ime ni obdržalo. Ime kot pa ga ima sedaj pa je jezik dobil po C# noti, kar naj bi predstavljalo nekaj višjega (kot c# nota), višji programerski jezik .

Programski jezik C# je objektno orientiran jezik, ki so ga v podjetju Microsoft razvili kot osnovni programski jezik, ki podpira vse funkcionalnosti .NET Frameworka. Zadnja verzija programskega jezika C# je 8.0; za svoje delovanje potrebuje .NET Framework 8, z njim pa lahko programiramo v razvojnem okolju od Visual Studio 2010 naprej(novejše verzije). Jezik izvira pretežno iz programskih jezikov C++, Visual Basic in Java. Namenjen je pisanju programov v okolju .NET in je primeren za razvoj najzahtevnejše programske opreme. Poleg tega je zasnovan na tak način, da ga je mogoče preprosto izboljševati in razširjati, brez nevarnosti, da bi s tem izgubili združljivost z obstoječimi programi. (1)

## Objektno programiranje

Poznamo več načinov programiranja in sicer:

* **proceduralni**: seznam navodil, kaj je treba izvesti (npr.: C, Pascal)
* **deklerativni**: opišemo kaj želimo, programski jezik pa izvede (npr., SQL)
* **funkcijski**: programiranje temelji na funkcijah (npr.: Haskell )
* **objektni**: program temelji na objektih, ki imajo lastnosti, funkcije ... (npr.: Java, Smalltalk)

Pri objektno usmerjenem programiranju se ukvarjamo z objekti. Objekt je nekakšna črna škatla, ki dobiva in pošilja sporočila. V tej črni škatli (objektu) se skriva tako koda (torej zaporedje programskih stavkov), kot tudi podatki (informacije nad katerimi se izvaja koda). Pri klasičnem programiranju imamo kodo in podatke ločene. (1) Recimo če imamo opraviti z objekti, ki opisujejo avtomobile, potem bomo v objektu verjetno našli podatke kot so: barva, moč, število vrat, letnik, prevoženi kilometri. Če imamo več avtomobilov, potem lahko opazimo, da potrebujemo objekte v katerih hranimo podobne podatke. Torej so si ti objekti med seboj podobni - imajo enako strukturo. Temu rečemo, da pripadajo istemu razredu. (2)

### Unity

Preden bom začel z opisovati svojo igro, bom za lažje razumevanje razložil Unity, ter orodja s katerimi sem si pomagal pri izdelavi svoje igre.

### Uvod v Unity

Unity poganja igre na različnih platformah z vgrajenimi IDE(Integrated Development Enviroment) orodjem, firme Unity Technologies. Z Unityem lahko razvijamo spletne, mobilne, namizne video igre ter igre na konzolah (3) .

Unity je programsko okolje, ki nam ponuja orodja za preprostejšo izdelavo iger, natančneje grafičnega okolja, programsko kodo pa je bolj praktično da jo urejamo v katerem drugem za to specializiranem programskem okolju, v mojem primeru Visual Studiu. Pri izdelavi iger uporabnika ne obremenjuje s tehnologijami, ampak mu nudi tehnologijo v obliki orodij. (4) Ko se lotimo izdelave igre oz. projekta, se mapa *Assets* (sredstva) samodejno ustvari, v tej mapi pa se nahajajo vsi elementi igre (slike, zvoki, skripte, animacije in ostale strukture, ki jih bomo potrebovali). Zaradi preglednosti si ustvarimo mape, *Sprite* (za objekte s sliko), *sounds* (za zvoke), *animations*(za animacije), *script*( za programsko kodo oz. kode), ter ostale mape, če potrebujemo več elementov. Uporabniški vmesnik okolja unity sestavljajo naslednja okna:

* **Project** - prikaže vsebino mape *Assets*(sredstva)
* **Hierarchy** - prikaže seznam objektov trenutne scene, poleg tega pa tudi starševska (child-parents) razmerja
* **Scene** - v tem oknu je prikazana scena, prikažejo se objekti ki si v *Hierarchy*
* **Console -** izpisujejo se obvestila programa Unity in sporočila, ki jih prikazujemo preko skript (ukaz Debug.log)
* **Inspector –** v tem oknu so nam prikazane komponente objekta, v tem oknu tudi upravljamo s komponentami objekta (npr.: dodajamo zaznavanja trkov, kodo, zaznamke)

### Nadzorovanje dela(*console* okno)

V oknu *console* se ob zagonu projekta izpišejo morebitne napake , sporočila ki so ključna za delovanje igre. Tako lahko s pomočjo tega okna preverjamo, če se kje pojavi napaka, kdaj kaj deluje in kaj ne. Zato je zelo koristna stvar za preverjanje funkcija Debug.Log()

### Komponente objektov

Ena izmed dobrih lastnosti unity je da lahko nastavljamo razne komponenta na objekte(animacije, trkalnike, obnašanja..) in s tem ne vplivamo na ostale stvari v našem projektu. To nam omogoča večjo fleksibilnost, kar nam pride prav tudi pri kasnejših popravkih za nazaj, saj s popravki ne vplivamo na ostale objekte, kodo. (3)

Komponente objekta *Game.Object-a* so vse komponente ki jih ima objekt, najbolj pogoste komponente pri objektih so detektorji trkov(*collider detection)* , skripte oz. koda ki je priložena objektu, animatorji(upravljajo z animacijo objekta). Komponente dodajamo objektom, tako da izberemo objekt(na katerega želimo dodati komponento) in se nam v oknu *inspector* prikažejo lastnosti objekta, kjer pa dodamo komponento z gumbom *Add Component,* to je ročni način nastavljanja komponent na objekt in je bolj praktičen, kadar na objekt ne nameravamo dodajati veliko komponent.

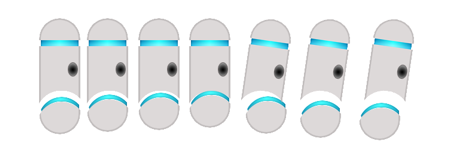
Če pa načrtujemo dodajanje več komponent (več od 10), pa je dodajanje le teh veliko bolj praktično, če jih dodajamo ter z njimi upravljamo preko kode/skripte z ukazom *GetComponent<>()*

Sintaksa : Komponent ime\_komponente = GetComponent<Komponenta>();

### Igralec (*Player*)

Igralec(objekt) je eden izmed najbolj kompleksnih objektov v igri, saj je glavna stvar v igri, ker v igrici le ta predstavlja nas, z njim igramo.

Da nam je igra bolj všečna za igranje, je to da ko upravljamo z igralcem, nam da občutek da je resničen zaradi gibanja, to pa dosežemo z animacijami . Animacija je niz sličic, katere eno za drugo prikazujemo z določeno hitrostjo, to pa nam da občutek gibanja. Vsak objekt ima lahko več različni animacij, te pa se med seboj izmenjujejo glede na gibanje. Izdelavo animacije si lahko pomagamo z nabori slikic(*Sprite Sheet)* , katere lahko uredimo s pomočjo animatorja, v katerem lahko nastavljamo na kolikšnem časovnem intervalu nas se slikice zamenjujejo.



Slika 1 Primer nabora slik(»Sprite sheet«) za animacijo

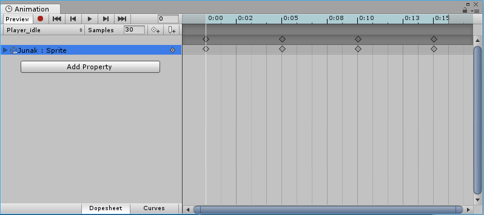
Nabor slik program unity razume kot eno samo sliko, zato je potrebno slike obrezati. Sedaj imamo že veliko orodij za rezanje slik, katera ima unity že vgrajene, lahko pa to opravimo tudi z zunanjem programom. Slike obrežemo v unityu tako, da izberemo nabor slik v mapi (v *hierarchy* zavihku), nato pa se nam v inšpektorju (*inspector)* pojavijo njene lastnosti. Privzete lastnosti (*Sprite mode)* moramo spremeniti iz vrednosti *Single*( predstavlja samo eno sliko) na **Multiple** in nato s pomočjo *Sprite Editorja* lahko uredimo ročno ali pa avtomatsko.

* **Avtomatsko**: Unity ima vgrajena orodja, katera razrežejo slikice(program jih prepozna)
* **Ročno**: Vsako slikico posebej lahko razrežemo slikice kot želimo, sicer je veliko bolj zamudno od avtomatskega rezanja, a nam pride prav če z avtomatskim rezanjem slik nismo zadovoljni.

### Animacija

Untyev sistem animacij je zasnovan na konceptu odsekov animacij, ta pa vsebuje informacije kako bo določen objekt spreminjal svojo smer, rotacijo ali ostale svoje lastnosti. (5)Animacija je sestavljena iz več slik, katere se ena za drugo prikazujejo v določenem časovnem intervalu. Določen objekt ima lahko več animacij(animacija za tek, hojo, skakanje…).

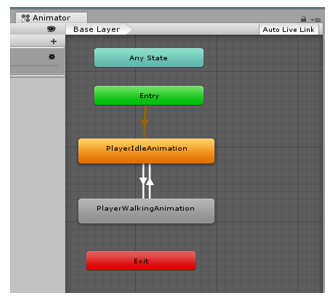
Animacijo naredimo s pomočjo orodja v unity *Animation .* Razrezane slike samo povlečemo v okence na ustrezne časovne intervale(čas med slikami predstavlja, koliko časa je slika prikazana, preden se zamenja) , ter nastavimo hitrost predvajanja slik. Več slik ima animacija, bolj je kvalitetna in se nam zdi bolj realna.



Slika 2 Orodje za ustvarjanje animacije

Ko imamo animacije za različna stanja(mirovanje, premikanje…) narejene, moramo le še urediti prehode med njimi, to pa uredimo z orodjem *Animator.* Ko ga odpremo se nam pokaže graf, ki prikazuje prehode med animacijami. Prvo stanje je zmeraj stanje mirovanja, ta pa je iz stanja *Entry* povezana z enosmerno puščico – to pomeni da se bo ob zagonu programa najprej predvajala ta animacija (mirovanje), to pa zato ker je to privzeto stanje in je obarvano z oranžno barvo, to pa lahko spremenimo (nastavimo drugo animacijo kot privzeto), z gumbom *Set as Layer Default State* , začetno stanje pa je lahko le eno.

Prehodi med animacijami so v animatorju(*Animator*) predstavljeni s puščicami, puščice prikazujejo stanja v katerega se lahko premaknemo iz trenutnega stanja. Puščice (prehode) naredimo tako da označimo animacijo, ter izberemo *Make Transition* in tako potegnemo puščico do animacije, do katere želimo imeti prehod.



Slika 3 orodje animator, primer prehodov med animaciji (mirovanje ter premikanje-hoja)

V animatorju pa moramo tudi nastaviti parametre, z namenom da bo program vedel kdaj se mora predvajati določena animacija. Da to naredimo moramo najprej določiti spremenljivke, njihova vrednost pa bo obveščala program kdaj se bo posamezna animacija predvajala. Program pozna 4 različne tipe spremenljivk : *Integer, Bool, Trigger, Float* . Primer: imamo 2 amimaciji PlayerIdleAnimation(mirovanje na mestu), ter *PlayerWalkingAnimation*(hoja, premikanje). Ustvarimo parameter hitrosti objekta tipa *Integer* , nato v skripti napišemo kodo, da je hitrost objekta 0, predvajamo animacijo mirovanja(*PlayerIdleAnimation*), če pa hitrost ni enaka 0, potem pa se predvaja animacija hoje(*PlayerWalkingAnimation*). V metodi *Update()* se preverja ali je določena tipka spremenila vrednost parametra hitrost, če je potem se animacije zamenjajo do trenutka ko tipko spustimo.

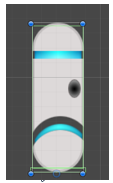
### Metode kode

V unity poznamo 4 glavne metode, katere so ključne za delovanje igre. To so metode *Start(), Update(), FixedUpdate(),* in *LateUpdate().*

* *Start()* je metoda ki se odvije samo enkrat ob zagonu in nikoli več. Izvede pa se pred vsemi metodami *Update()* . Uporablja pa se predvsem za inicializacijo spremenljivk in povezav do drugih stvari (npr. reference do komponente)
* *Update()* metoda se odvije vsako sličico, ko se slika posodobi. Najpogosteje se uporablja implementiranje obnašanje igre(premikanje, preverjanje pogojev)
* *FixedUp()* metoda nam pride prav, kadar upravljamo s komponento *RigidBody2D,* ko se ukvarjamo s fiziko telesa.
* *LateUpdate()* metoda skrbi, da se kamera premika skupaj z igralcem, tako da nam je igralec zmeraj center pozornosti v naši igri

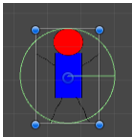
### Trkalniki(*Colliders*)

Če želimo narediti dobro igro, mora naš junak v okolje prepoznavati ostale predmete, saj ni realistično da bi se skozi njih kar sprehajal, da ne pride do tega si pomagamo z trkalniki (6)

Razvoj 2D in 3D igre je možno v Unityu, saj nam omogoča fiziko, s pomočjo Box2D orodja. (7) Zasluge temu pa gredo prav trkalnikom (ang. *Colliders*),te pa uporabimo ko se ukvarjamo s fiziko objektov in sicer zaznavajo dotike med objekti, na katere smo nastavili trkalnike, lahko bi jim rekli senzorji dotika, kajti to je njihova naloga. Ko nastavljamo trkalnike na objekte, so ti predstavljeni z zelenimi črtami(ko jih urejamo) okoli objekta. Eni izmed najbolj pogostih trkalnikov so:

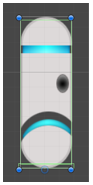
Slika 4 Škatlasti trkalnik

* Škatlasti trkalnik (*BoxCollider2D*) : je pravokotne oblike, ima določeno pozicijo, širino in dolžino, glede na velikost objekta.
* Krožni trkalnik(*CircleCollider2D*): trkalnik zaznava okolico do polmera(radija) objekta

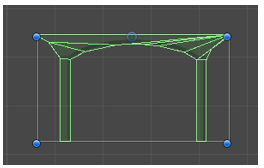


Slika 5 Krožni trkalnik

* **Trkalnik roba(*EdgeCollider2D*) :** Ta trkalnik uporabljamo za mesta na objektu, ki so najbolj izpostavljeni dotikom ostalim predmetom in želimo da so dotiki na teh mestih(točkah, robovih) bolj natančni, kot sicer. Lahko določimo samo točko dotika, ali pa premico.



Slika 6 Slika 6 Poleg škatlastega trkalnika, je na objektu, na dnu dodan še trkalnik roba, saj je spodnji del objekta neprestano izpostavljen podlagi

* **Avtomatski trkalnik(*PolygonCollider2D*)** : ta trkalnik zazna obliko objekta in prilagodi zaznavne črte objektu. Prav nam pride, kadar imamo objekte bolj zapletenih oblik in želimo da so senzorji izključno na njihovih robovih in ne tako kot pri ostalih, ki le okvirno porijejo objekt(primer: škatlasti trkalnik je v obliki pravokotnika in ne pokrije robov objekta, ki ni pravokoten). Za ta način pa mora biti slika brez ozadja, sicer orodje ne bo prepoznalo robov.

Slika 7 Avtomatski trkalnik prepozna robove objekta, če le ta nima ozadja

# **Praktični del- izdelava igre v Unity**

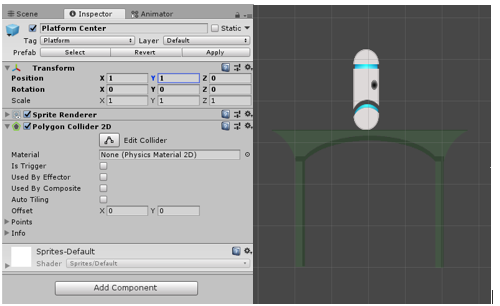
Najprej sem sem moral pridobiti vsa sredstva(*Assets*) oz. slike, katere bodo predstavljale vizualne objekte v igri. Vse to lahko izdelamo samo v drugim programih(kot so npr: *fotoshop* programi) po svojih željah, lahko pa ta sredstva oz. slike za igrico poiščemo na spletu. Tako da preden se lotimo izdelave igre si moramo narediti oz. nabrati vsa sredstva in jih dodati v projekt v unity, da bomo lahko pozneje z njimi upravljali.

Kot sem že povedal ima unity veliko orodij, s katerimi si lahko pomagamo pri izdelavi igre, najboljše pa je to da imamo vizualni pregled nad dogajanjem. Tako da preden sem se lotil logike igre( programiranje, pisanje kode) sem moral v igro(sceno) dodati objekte v igri, za začetek našega glavnega junaka, katerega bomo mi igrali ter platformo, na kateri bo naš junak lahko stal.

## Dodajanje objektov v sceno

V sceno sem najprej ustvaril objekt platforme, na kateri bo naš junak stal. Platformo sem poimenoval *Platform Center* , saj bo to naša centralna platforma(kasneje jih bom dodal še nekaj), njeno pozicijo sem nastavil na koordinate X:0,Y:0 in Z:0, ker je to dve-dimenzionalna (gledamo samo os X in Y) igra, vrednost Z osi ne igra bistvene vloge. Njeno velikost X in Y osi imajo obe vrednosti 1. Pod komponentami sem nastavil na platformo avtomatski trkalnik (*PolygonCollider2D*), kar bo omogočilo da bodo ostali objekti zaznavali platformo in bo lahko za primer naš junak na platformi stal, ker jo bo prepoznal in ne bo mogel iti čez njo.

Ko imam platformo, na kateri lahko junak stoji sem dodal še glavni objekt - junaka.

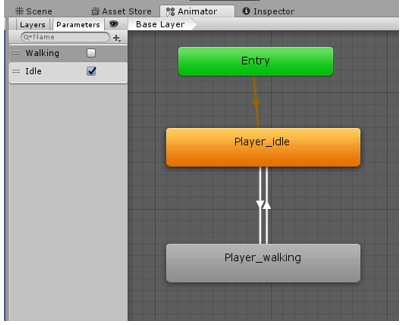


Slika 8 Na levi so nastavitve, ter dodane komponente platforme. Na desno pa je vizualni prikaz objektov platforme in junaka

## Animacija objekta junak

Nabor slik sem zrezal na 7 slik, 4 za mirovanje in 3 za premikanje. Tako sem naredil 2 animaciji s pomočjo orodja *Animation* , to sem naredil tako da sem slike(za mirovanje) povlekel v orodje in določil čas, preden se zamenja slika. Animator deluje tako, da s pomočjo neskončne zanke zamenjuje slike v določenih časovnih intervalih v krogu.

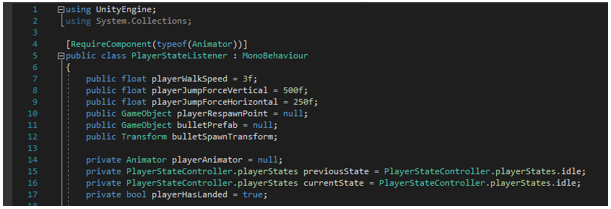
Ko sem ustvaril obe animaciji( za hojo in mirovanje) sem v orodju *Animator* povezal animaciji z dvosmerno puščico, tako sem naredil prehod med animaciji. Da pa bomo lahko program obvestili kdaj naj se predvaja določena animacija, sem dodal spremenljivki tipa **Bool** ( ime : Walking za premikanje in Idle za mirovanje). Torej ko bomo v kodi ovrednotili vrednost spremenljivke na *true* se bo animacija začela izvajati. Animacija mirovanja *(Player\_Idle*) je privzeta animacij, kar pomeni da se ob zagonu metode *Start()* začne prevajati, dokler ne pride do spremembe določenih parametrov.



Slika 9 V animatorju urejamo prehode med animacijami, ker je funkcija mirovanja privzeta animacija je njen parameter na začetku nastavljen na true.

## Gibanje junaka

Najprej sem napisal kodo, ki bo prepoznavala tipke ki jih pritiskamo in bo potem na podlagi tipk spreminjala horizontalno in vertikalno pozicijo objekta, to bom naredil v razredu PlayerStateController , potem bomo pa morali o stanju objekta (na katero stanje mora junak preiti) obvestili program, kar pa bom naredil s kodo PlayerStateListener . Unity shranjuje vsa stanja(skok, pomik v levo,densno…) kot osi, le če imajo vsi vrednost spremenljivke float, to omogoča da program prepoznava kdaj se mora izvesti določeno stanje. (9)

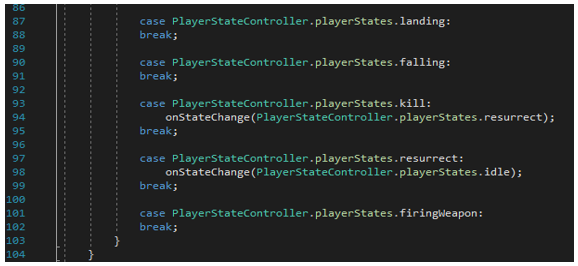


Slika 10 koda razreda PlayerStateListener, nastavimo spremenljivke

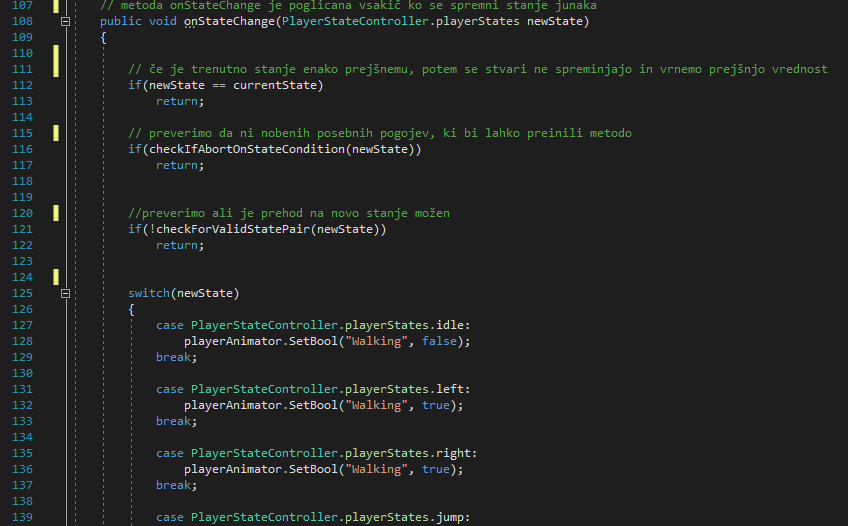
V razredu PlayerStateListener najprej nastavimo vrednosti spremenljivk, najprej se bomo osredotočili na spremenljivke playerWalkSpeed, playerJumpVertical in playerJumpHorizontal. Spremenljivka playerWalkSpeed velja za premikanje v levo in desno, medtem ko pa potrebujemo za skok 2 spremnljivki, saj lahko skočimo samo navzgor ali pa levo in desno. Njihove vrednosti pa nam povedo za koliko se bo objekt premaknil.



Slika 11 razred PlayerStateController, funkcija onStateCycle

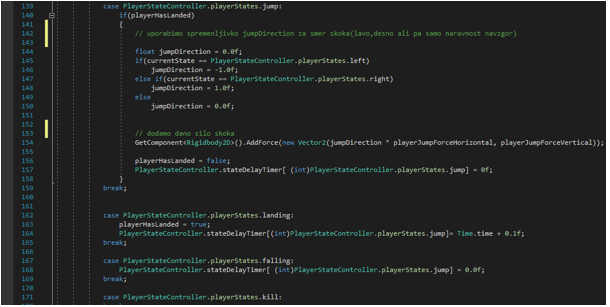


Slika 12 razred PlayerStateController, funkcija onStateCycle

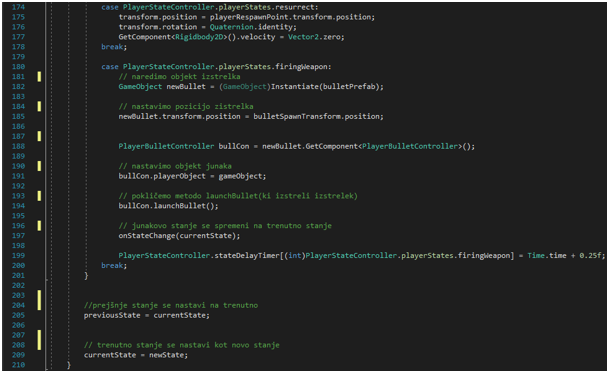
V funkciji **onStateCycle** preverjamo vsak cikel preverjamo stanje našega junaka, da lahko na podlagi njegovega stanja pokličemo primerne metod s pomočjo switch stavka.

Slika 13 Razred PlayerStateController, funkcija onStateChange

**onStateChange** metoda se pokliče vsakič, ko se spremeni junakovo stanje. Če je novo spremenjeno stanje enako prejšnjemu, se ne spremeni nič. Potem pa preverimo, če lahko s trenutnega stanja preidemo na novo(pogoji so zapisani v funkciji checkforValidStatePir, katero bom kasneje opisal), saj junak ne mora iz vsakega stanja preiti na vsako(primer: s skoka ne more preiti na hojo direktno, ampak mora najprej pristati, ali pa umreti). Ko smo prepričani, da je prehod možen in ni posebnih pogojev, ki bi prekinili metodo se izvede switch stavek.

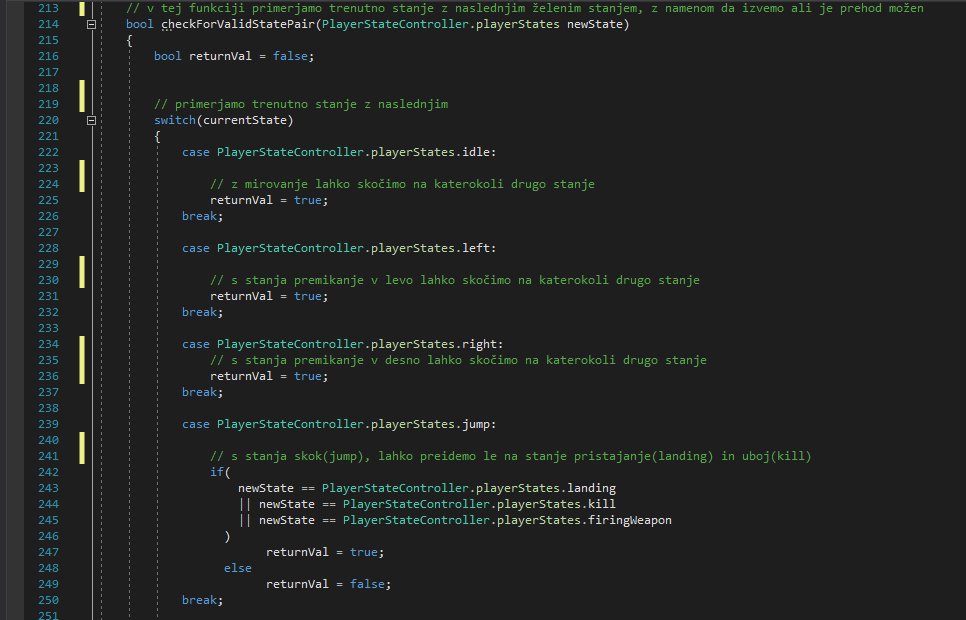


Slika 14 Razred PlayerStateController, funkcija onStateChange

V switch stavku se izvede stanje, katerega spremenljivka(bool) se je ovrednotil na vrednost true. V primeru da pritisnemo tipko, ki predstavlja premik v levo, switch stavek skoči na primer, ki ustreza pogoju novega stanja(levo). V primeru da smo pritisnili tipko ki predstavlja skok, potem preverimo tudi njegovo smer, s pomočjo spremenljivke jumpDirection tipa float(število s plavajočo vejico), da vemo v katero smer skočiti

Slika 15 Razred PlayerStateController, funkcija onStateChange

V primeru da se je spremenljivka resurrect(zgodi se ko junak umre) ovrednotila na true, potem se spremeni njegova pozicija na objekt playerRespawnPoint, ker ponovno »oživi«. V primeru da pritisnemo tipko, ki predstavlja strel pa se ustvari objekt izstrelka, njegova pozicija izhaja iz junakove pozicije(izstrelek je v untyu vezan na junaka, PlayerObject). Pokličemo metodo launchBullet(), ta pa naredi in izstreli izstrelek. Potem ko smo izstrelili izstrelek. Prejšnje stanje se ovrednoti na trenutno, trenutno pa se ovrednoti na novo.



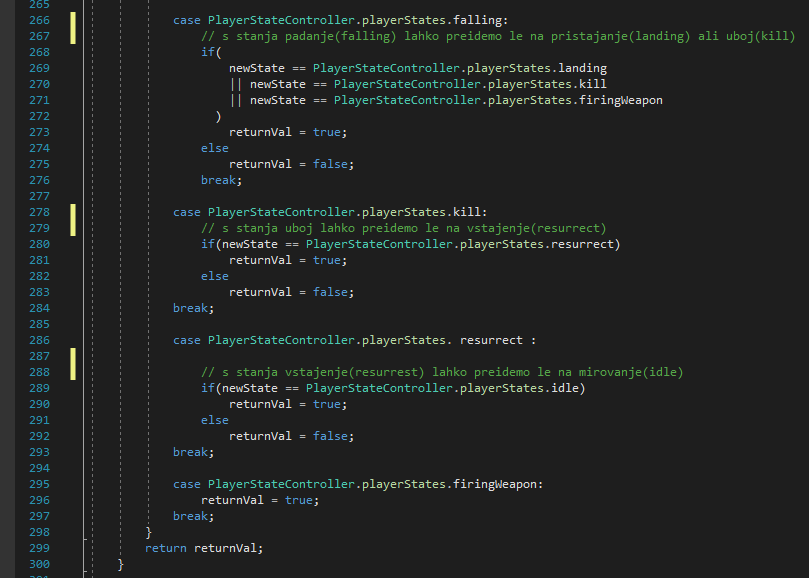
Slika 16 Razred PlayerStateController , funkcija checkForValidStatePair

Ker ne želimo, da bi lahko prehajali med različnimi stanji kakor bi želeli, moramo nastaviti pogoje kateri prehodi so dovoljeni.

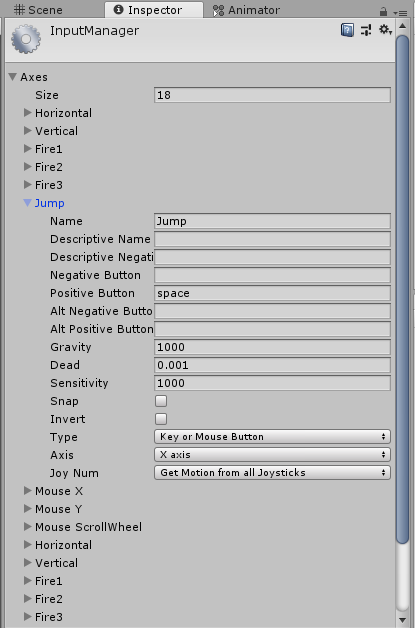
Primer: iz skoka ne moremo preiti direktno na premikanje v levo ali desno smer. Najprej moramo pristati, ali pa da nas ubije sovražnik(katere sem kasneje v igri tudi ustvaril) in potem se lahko šele začnemo premikati v želene smeri.

Torej v funkciji **checkForValidStatePair** preverjamo kateri prehodi med stanji so mogoči. S switch stavkom izvemo kakšno je trenutno stanje in skočimo na primer, kateri ustreza trenutnemu stanju.

Stanja mirovanje(idle) lahko preidemo na katerokoli drugo stanje prav tako velja za pomikanje v obe smeri(desno in levo). Zaplete pa se pri skoku(jump), s katerega lahko preidemo le na pristajanje(landing),strel (firingWeapon) ter uboj(kill). S funkcije padanje(falling) lahko preidemo le na pristajanje(landing), uboj(kill) ali strel(firingWeapon). Če je bil junak ubit, lahko preidemo le na vstajenje(resurrect), s tega stanja naprej pa le na mirovanje(idle). S strela(firingWeapon) pa lahko preidemo na katerokoli stanje.

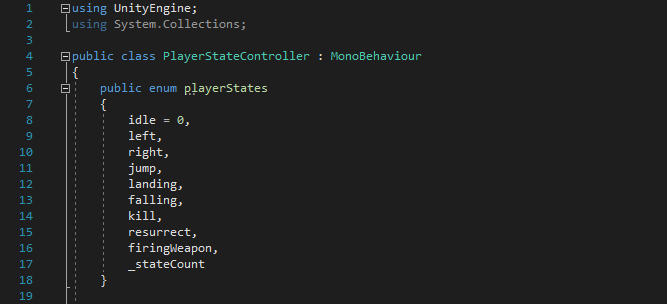


Slika 17 Razred PlayerStateController, funkcija checkForValidStatePair

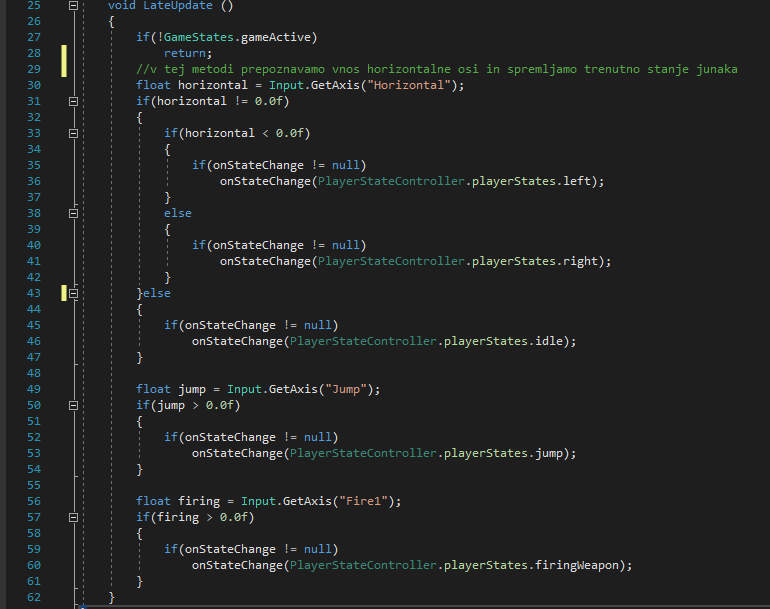
Sedaj ko program ve vse v povezavi s stanji junaka, pa moramo samo še nastaviti kontrole za določena stanja, kot so premikanje, skok in strel. Ker je unity posebej zasnovan za za izdelavo iger, lahko sami nastavljamo(v unity) tipke za določena stanja, tako da v meniju izberemo: Edit🡪Project🡪 Settings🡪Input , kjer nastavimo pozitivne vhode (lahko jih pobiramo s tipkovnice, računalniške miške ali pa igralnega ploščka).

Slika 18 orodje za nastavljanje vhodov

Primer: na sliki desno je za dejanje skok pozitivni vhod s tipkovnice »space«.

Da pa bo program unity vedel katero stanje oz. dejanje se mora izvesti, pa sem napisal v razredu PlayerStateListener.

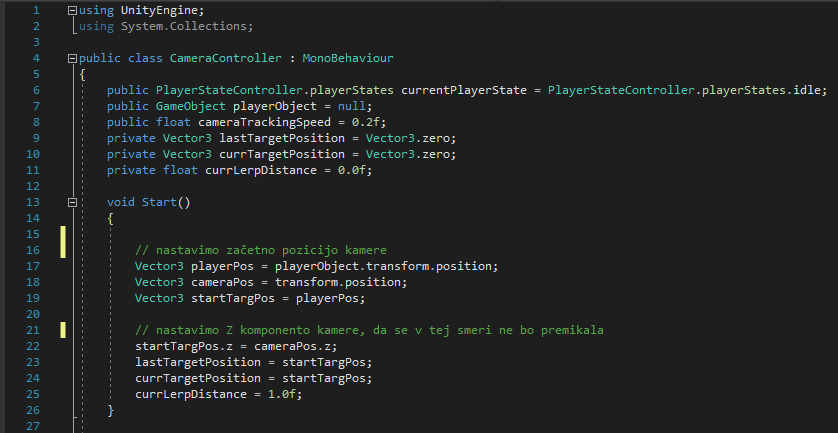
Slika 19 Razred PlayerStateListener



funkcij.Metoda LateUpdate() se le v primeru da smo igro zagnali.Najprej nas zanima vrednost horizontalne spremnljivke, zato ustvarimo spremenljivko tipa float, ter bomo s tipkovnice(v našem primeru) pobirali vrednost horizontalne spremnljivke . Potem preverimo kolikšna je njena vrednost. Če je različna od nič, potem vemo da gre za premikanje. V primeru da je manjša od nič (negativna) obvestimo metodo onStateChange o spremembi stanja v levo smer, če ni negativna pa v desno. Če pa se horizontalna spremenljivka ovrednoti na 0, potem vemo da je junak na mestu, sporočimo metodi onStateChange, da junak miruje(idle).

Slika 20 Razred PlayerStateListener, funkcija LateUpdate

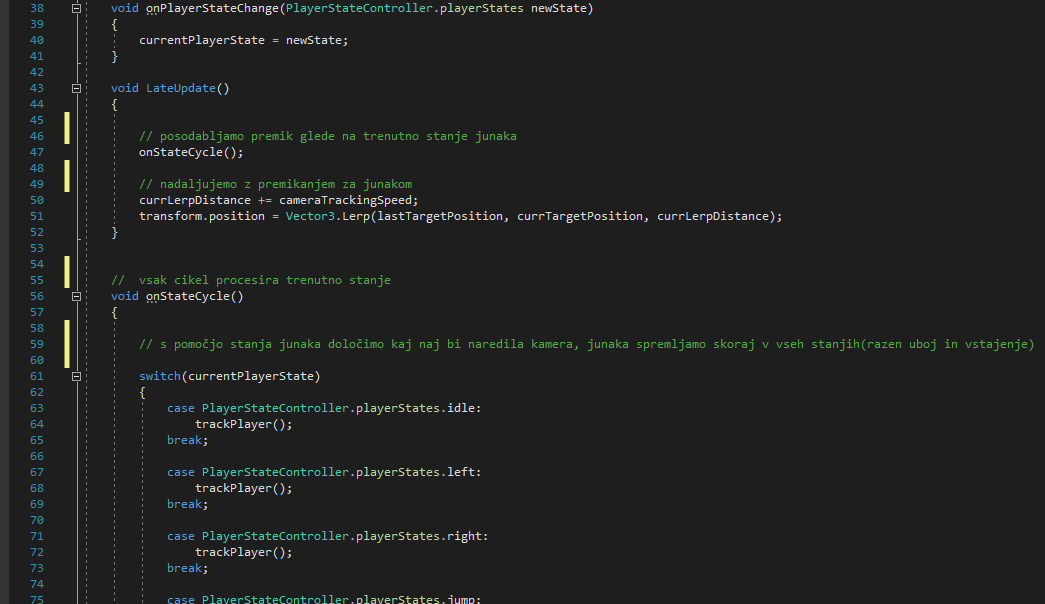
## Kamera

Ker naš junak predstavlja glavni objekt v naši igri, ga želimo imeti neprestano na centru kamere. V razredu CameraController sem napisal kodo, da kamera neprestano sledi igralcu na vsakem koraku.

Slika 21 Razred CameraController

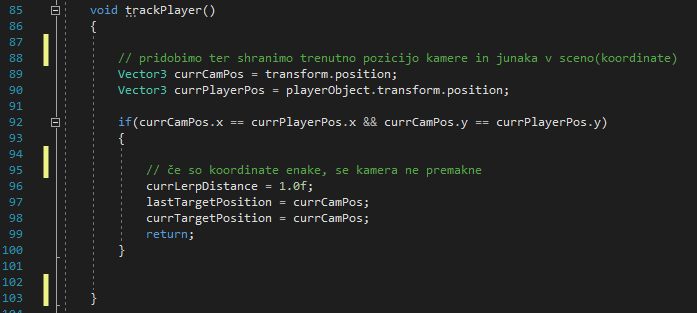
Najprej nastavimo spremenljivke, da bomo vedeli smer in hitrost premika kamere.

V funkciji Start nastavimo začetno pozicijo kamere, ter ciljno pozicijo kateri bomo sledili(junaku). Z komponento moramo nastaviti tako, da se njena vrednost ne bo spreminjala, saj je to dve-dimenzionalna igra, kje so za premik pomembni le X in Y osi.



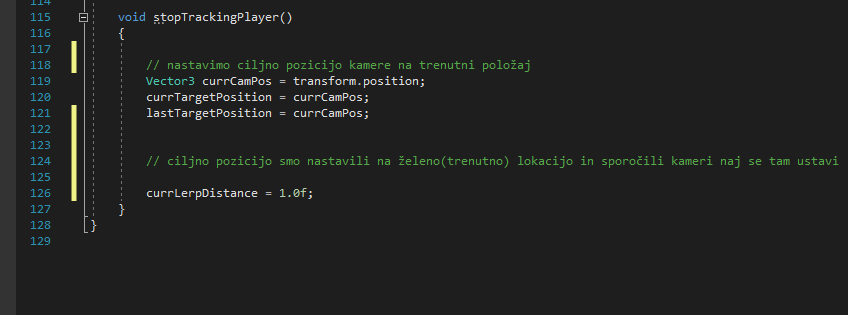
Slika 22 Razred CameraController, funkciji LateUpdate in onStateCycle

V funkciji **LateUpdate** posodabljamo premik glede na stanje junaka, premik pa se premakne za hitrost premikanja kamere(cameraTrackingSped), tako kamera neprestano sledi junaku.

V funkciji onStateCycle moramo vsak cikel preverjati trenutno stanje junaka in glede na na njegovo stanje se tudi kamera premakne. V tej metodi ne rabimo preverjati vseh stanj, saj stanje ne pomeni nujno premika. 

Slika 23 Razred CameraController, metoda trackPlayer

S funkcijo **trackPlayer** pobiramo in shranjujemo pozicijo junaka in kamere (imata isto pozicijo, saj kamera sledi junaku). Če je x in y pozicija kamere enaka x in y pozicijo junaka, potem se kamera ne premakne, miruje.

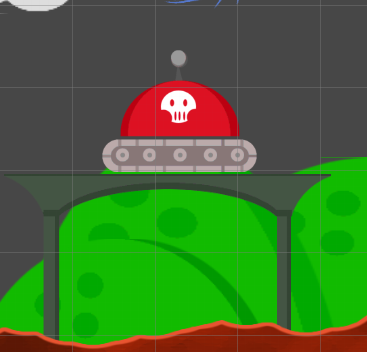


Slika 24 Razred CameraController, funkcija stopTrackingPlayer

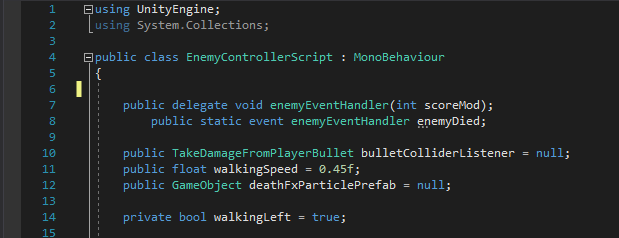
Nastavimo vrednost vektorja trenutne pozicije kamere na trenutni položaj, ter spremenljivki currLerpDistance nastavimo vrednost na 1, saj se želena pozicija ujema s trenutno in se kameri ni treba več premikati.

## Sovražniki

Da bo igra imela smisel, potrebujemo v igri še sovražnike katere moramo premagati. Dodal sem nove objekte v sceno-sovražnike. Prav tako sem za vsakega ustvaril animacijo. Ustvarjanje sovražnika ni nič novega, vsi koraki so skorajda isti kot pri ustvarjanju junaka, le da ima sovražnik par specifičnih stvari, saj ga ne upravljamo mi (9).

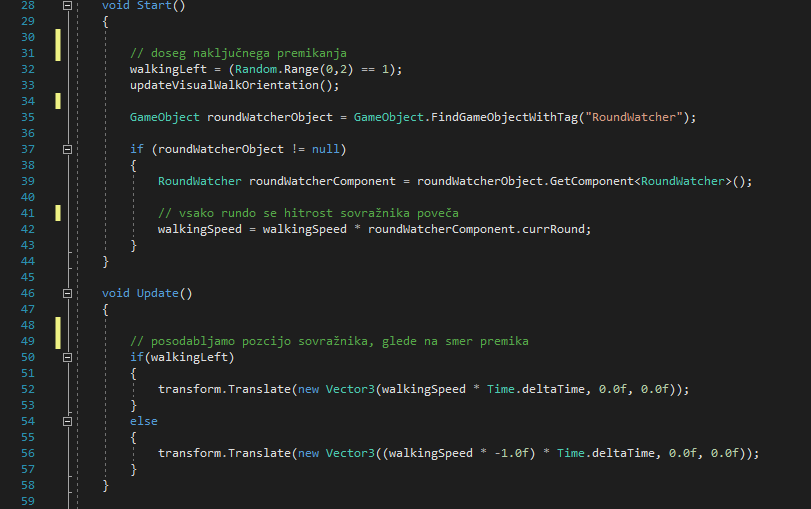


Slika 25 Sovražnik



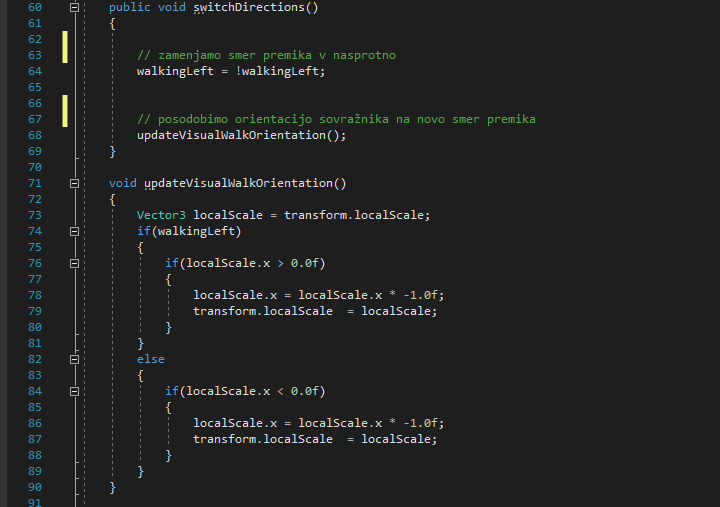
Slika 26 Razred EnemyController

Deklariramo spremenljivke, katere potrebujemo za upravljanje s sovražnikom. V metodi **Start** najprej nastavimo doseg, do kje se bo lahko premikal sovražnik, vsako rundo pa se bo njegova hitrost povečala.



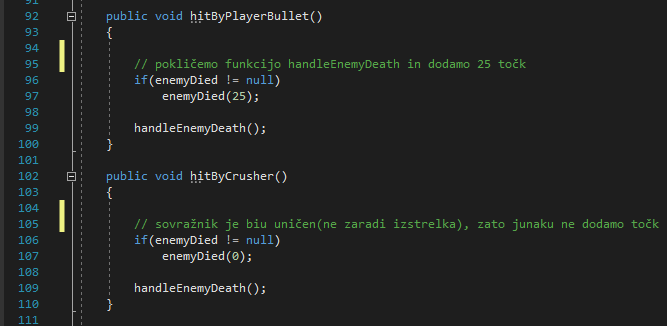
Slika 27 Razred EnemyController

Njegovo pozicijo neprestano posodabljamo v funkciji **Updade** , če je vrednost spremnljivke left true, potem vemo da se premika v levo, v nasprotnem primeru pa gre v desno in mu določimo za koliko naj se premakne s pomočjo njegove hitrosti.



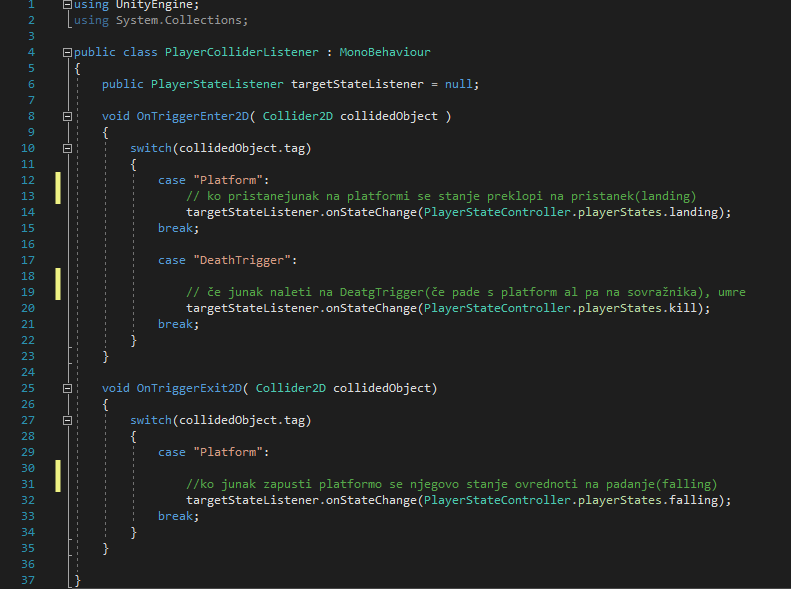
Slika 28 Razred EnemyController, funkciji updateVisualWalkOrientation in switchDirections

Ko sovražnik prepozna da je prišel do roba platforme se pokliče funkcija **switchDirections** , kater pa zamenja smer premikanje in pokliče funkcijo **updateVisualWalkOrientation ,** ki pa preveri v katero smer se je premakni sovražnik in mu posodobi pozicijo.



Slika 29 Razred EnemyController, funkciji hitByPlayerBullet in hitByCrusher

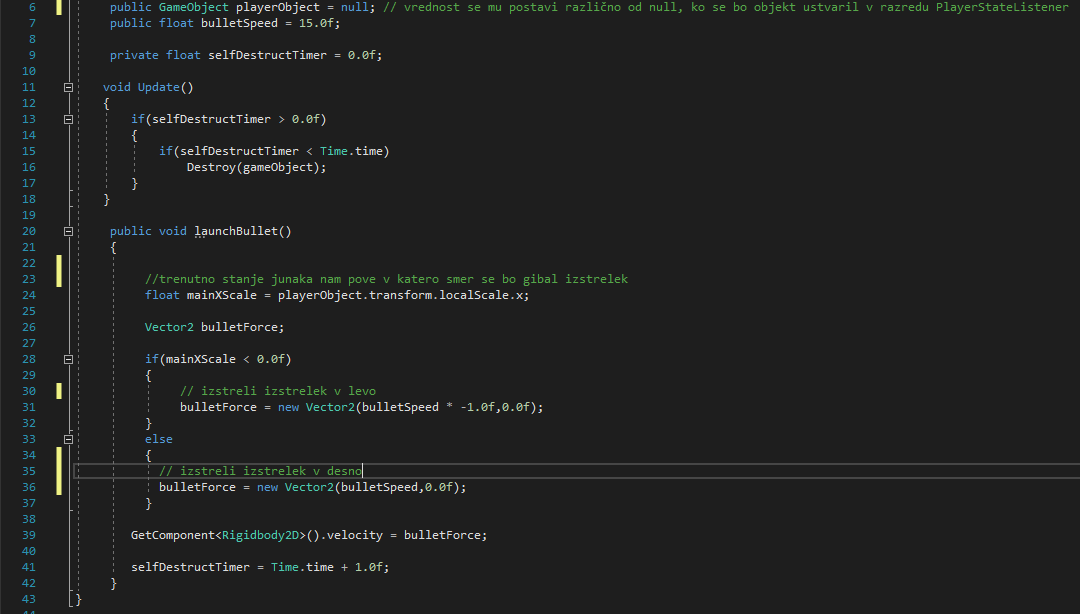
Ker lahko sovražnik umre na 2 možna načina, potrebujemo za vsak način metodo. Če je uničen zaradi izstrelka, se igralcu prišteje 25 točk, v obeh funkcijah pa se pokliče funkcija **handleEnemyDeath** , katera pa ponovno na naključni poziciji ponovno oživi sovražnika.

Naenkrat so v sceni lahko samo štirje sovražniki, vsak je svoj objekt in imajo iste komponente nastavljene. Nastavil sem jih 2 trkalnika, rigidbody2D ter BoxCollider2D, saj mora sovražnik prepoznavati rob platforme(da ne pade z nje) in njihovi robovi so bolj izpostavljeni dotikom, saj če se junak dotakne sovražnika, potem se sproži funkcija onTriggerEnter2D, ki pa ob dotiku z junakom postavi vrednost kill na 1, kar pomeni da junak umre in se ponovno oživi na objektu PlayerRespawnPoint, ki je nastavljen na centralni platformi.

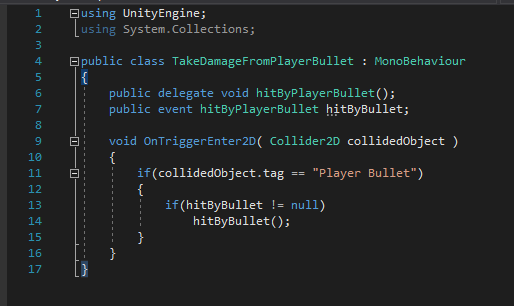
Slika 30 Razred PlayerColliderListener

## Izstrelek

Ko izstrelimo izstrelek, moramo vedeti v katero smer se premika. To pa obvestimo program s funkcijo launchBullet, ki pobere vrednost stanja junaka in nastavi vrednost spremenljivke mainXScale na vrednost premika junaka. Če je vrednost manjša od 0, potem vemo da se junak premika v levo smer(na isti način smo preverjali smer premika junaka v razredu PlayerStateController) in s pomočjo vektorja izstrelimo izstrelek v levo stran, v nasprotnem pa v desno.



Slika 31 Razred PlayerBulletController



Slika 32 Razred TakeDamageFromPlayerBullet

# **Zaključek**

Izdelava igre je bila zapletena, saj se večkrat kakšna stvar zna zaplesti, še posebej pri kodi in popravljanje le te zna biti zelo zapleteno in mučno. Skozi izdelavo igre sem spoznal osnove za delo z programom Unity, ki ima zelo veliko orodij za pomoč pri izdelavi igre. Cilj seminarske igre sem dosegel, a z trdim delom in vztrajnostjo. V prihodnje nameravam še razvijati računalniške igre, a v skupinskem delu, daj je izdelava malo boljše igre prevelik zalogaj za samo enega programerja. Skozi razvijanje igre sem pridobil dosti znanja, kateri mi bo kasneje še prišel prav.

# **Viri in literatura**

Literatura: Unity 2D Game Development, avtor Dave Calabreste, založništvo PACKT, 2014

Viri:

<https://www.udemy.com/2d-game-development-unity/>

<https://answers.unity.com/questions/182391/how-to-make-your-character-move.html>

<https://unity3d.com/learn/tutorials/s/2d-game-creation>

<https://unity3d.com/learn/tutorials/topics/2d-game-creation/2d-game-development-walkthrough>

<https://www.ubuntu.si/vodnik/12.04/fallback-mode.html>

<https://sl.wikipedia.org/wiki/Unity>

<https://ucilnica.fri.uni-lj.si/pluginfile.php/33595/mod_resource/content/0/Unity.pdf>

<http://im.scv.si/wiki/index.php/Unity>

# **Citirana dela**

1. **Unity. *Games made in unity.* [Elektronski] [Navedeno: 10. 4 2019.]**

**2. Srečo Uranič, Matija Lokar. uranic.tsckr. [Elektronski] 4 2009. http://uranic.tsckr.si/VI%C5%A0JA%20%C5%A0OLA/Programiranje%20I/C%23\_objektnoSreco.pdf.**

**3. nauk.si. *PHYTON-objektno programiranje.* [Elektronski] http://www.nauk.si/materials/5758/out/#state=4.**

**4. Polsinelli, Pietro. Design a game. *Unity popular game development.* [Elektronski] 5. 12 2013. https://designagame.eu/2013/12/unity-popular-videogame-development/.**

**5. Učilnica.fri. [Elektronski] https://ucilnica.fri.uni-lj.si/pluginfile.php/33595/mod\_resource/content/0/Unity.pdf.**

**6. Unity Documentation. [Elektronski] https://docs.unity3d.com/Manual/index.html.**

**7. Envato tuts+. *Getting Started With Unity: Colliders & UnityScript.* [Elektronski] 26. 4 2011. [Navedeno: 31. 3 2019.] https://code.tutsplus.com/tutorials/getting-started-with-unity-colliders-unityscript--active-8367.**

**8. Josh Petty. Concept Art Empire. *What is unity.* [Elektronski] https://conceptartempire.com/what-is-unity/.**

**9. Calabreste, Dave. *Unity 2D Game Developmant.* s.l. : PACKT , 2014.**