**DVI Diplomski**

**World Hopper project**

**Dokumentacija**

Detaljna dokumentacija koda, skripti, asseta i rešenja za probleme tokom projekta

# **Table of Contents**

Table of Contents

[**Table of Contents** 1](#_Toc118645053)

[**PLAYER CONTROLLER** 2](#_Toc118645054)

[Player Prefab setup 2](#_Toc118645055)

[Unity Character Controller 2](#_Toc118645056)

[Player Controller Script 3](#_Toc118645057)

[Public/Inspector variable 3](#_Toc118645058)

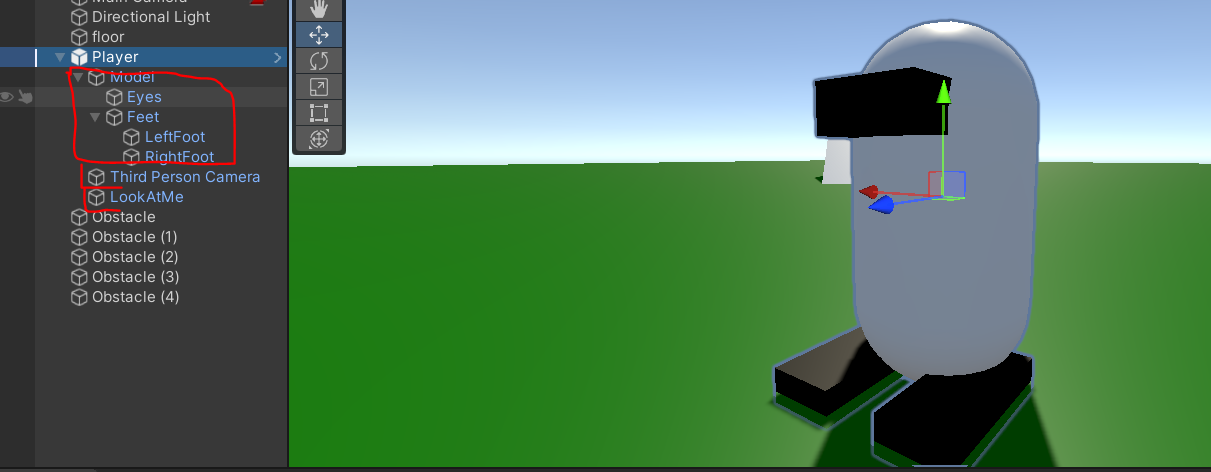
[Privatne/skrivene variable 3](#_Toc118645059)

[Metode 4](#_Toc118645060)

[Cinemachine 6](#_Toc118645061)

# **PLAYER CONTROLLER**

## Player Prefab setup

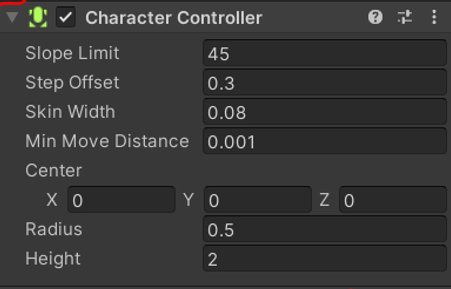


Player prefab je namešten tako da je parent objekat polu-prazan, to jest samo drži skripte i Character controller komponentu. Ispod njega se nalazi model child objekat koji drži krafiku za igrača, kao i Third Person Camera, koja drži Cinemachine kameru, i LookAtMe objekat, koji je prazan objekat u koji kamera gleda, i prati cursor.

U modelu, bitan objekat su stopala, koja proveravaju gde se nalazi zemlja, što je bitno za igračevo skakanje.

Takodje, grafika treba da (zbog nekog čudnog bug-a) bude offsetovana za -0.07999988 na Y osi.

## Unity Character Controller

Unityev Character controller je nešto nalik na rigidbody i capsule collider odjednom, s time da koristi zasebnu fiziku, i izmedju ostalog i gravitaciju, ali automatski detektuje stepenice, slopes, i slične stvari, i u sebi ima ugrađene Move metode, rotacije i slične alate koji olakšavaju posao.

## Player Controller Script

Svaka variabla u inspektoru ima svoj tooltip da, kada se predje preko nje mišem, ispiše šta je ona i šta konkretno radi.

### Public/Inspector variable

* Move Speed: Float koji određuje kojom brzinom se igrač kreće kroz prostor
* Smooth Rotation Time: Float između 0 i 1 koji određuje koliko brzo će igrač da se okreće oko Y ose, sprečava snappy movement
* Jump Force: *trebalo bi možda preimenovati u jump speed,* je brzina na vertikalnoj osi koja se dodaje igraču svaki put kada skoči
* Gravity: Igrač koristi svoju gravitaciju, jer se Character controller ne interactuje sa fizikom na isti način kao i rigidbody. Ovo je variabla koja je, dok je igrač u vazduhu, stalno primenjivana na dole u njegovoj vertikalnoj brzini
* Jumps Available: Koliko puta igrač može da skače pre nego što mora da dotakne zemlju
* Feet: Transform componenta koju koristimo za Physics.CheckSphere() dok proveravamo da li je igrač na zemlji ili nije.
* Ground Check Radijus: poluprečnik sfere koja proverava da li smo na zemlji, a centar te sfere je u Feet transformu
* Ground Layer: Koje layere igre Igrač zapravo smatra kao zemlja, od koju može da odskoči
* Camera Target Divider: Vezan je za položaj, to jest udaljenost kamere od miša
* Dash Distance: Koliko daleko igrač ide napred kada Dash-uje
* Dash Cooldown: Vreme koje mora da prođe da bi igrač mogao da dashuje nakon jednog dash-a

### Privatne/skrivene variable

* Novog Input Sistema
* moveDirection koji određuje pravac kretanja IZ INPUTA
* verticalVelocity koji određuje kretanje gore i dole
* Character Controller referenca
* Transform glavne kamere
* Transform onoga u šta kamera cilja
* Referenca za samu kameru (kamera komponenta tačnije)
* turnSmoothVelocity koji je prazan float namenjen da služi kao referenca za okretanje lika
* originalNumberOfJumpsAvailable je variabla gde čuvamo broj igračevih skokova kako bi mogli da ih resetujemo
* grounded, koji je boolean koji proverava da li smo na zemlji ili nismo
* canDash, boolean koji određuje da li možemo da dash-ujemo
* Lista svih InputActiona koje skripta koristi (move, look, fire i jump)

### Metode

#### OnEnable()

Ovo je metoda koja se poziva kada se objekat sa ovom skriptom upali u sceni. Ovde treba da namestimo Input-e.

Zapisana lista svih Input Akcija kako bi New Input Sistem mogao da radi kako treba. Logika ide tako da:

1. Povežemo akciju sa prikladnom akcijom iz novog input sistema (ime akcije = referenca na input sistem, mapa input sistema, specifična komanda)

Primer: *move = playerInputScript.Player.Move;*

1. Enable-ujemo datu komandu (neophodan korak u novom input sistemu da bi radio).  
   Primer: *move.Enable();*
2. Ako je akcija koja je povezana sa metodom, ili nekom konkretnom akcijom koju igrač radi, kao skakanje, onda želimo da povežemo ovde datu komandu sa njenom metodom, tj onime što želimo da se desi kada igrač pritisne to dugme.  
   Primer: *jump.performed += Jump;*(performed je momenat u input sistemu kada se očita komanda, slično GetKeyDown, i mi na tu metodu dodajemo našu metodu, ali iz nekog razloga ne stavljamo zagrade)

#### OnDisable()

Slično OnEnable(), ovo je metoda koja se zove svaki put kada se objekat ugasi u skripti, a mi takođe koristimo da isključimo dodatak novih inputa, što treba da radimo za svaku input akciju

Primer: *move.Disable();*

#### Awake()

Awake se poziva dok se scena/objekat učitava, i radi optimizacije, sve zahtevnije metode i hvatanje referenca obavljamo u ovoj metodi.

U ovoj metodi iniciramo naš Input Sistem, i hvatamo referencu za karakter kontroler, kameru i njen Transform, kao i šta kamera gleda.

#### Start()

Poziva se u prvom frejmu nakon ucitavanja, ovde nameštamo jednostavnije reference, kao recimo originalan broj skokova, i nebitnije procese, kao recimo gašenje miša

#### Update()

Poziva se svaki frame, i on drži logiku i metode za gledanje kamere, skakanje i gravitaciju, i pomeranje.

Kamera se pomera tako što prvo nađemo poziciju miša u relaciji sa svetom, pa onda namestimo poziciju **gledanja** kamere tako što poziciju miša saberemo sa camera divider-om pomnoženim sa pozicijom samog igrača, i onda to sve opet podelimo sa camera dividerom. Na kraju poziciju gledanja kamere namestimo na tu poziciju.

var cameraPosition = (mousePosition + (cameraTargetDivider - 1) \* transform.position) / cameraTargetDivider;

Skakanje, Gravitacija, i pomeranje su zasebne metode koje pozivamo svaki frame Updatea.

#### Dash(InputAction.CallbackContext context)

Ova metoda se poziva iz Input Sistema, i proverava ako igrač može da dashuje kroz canDash boolean, i ako može, prvo isključi taj boolean (da igrač ne može da spamuje dashovanje), i pokrene korutinu za cooldown, koja odbrojava kada igrač može opet da dashuje. Nakon toga uzimamo **forward vector glavne kamere,** brišemo koju god vrednost da ima na Y osi, i to koristimo kao smer kretanja za dash, to jest jednostavnu Move() metodu kontrolera. Takođe, resetujemo skokove.

#### Jump(InputAction.CallbackContext context)

Jednostavan if, proveravamo da li igrač ima više od 0 skokova da iskoristi, i ako ima, na vertikalnu brzinu mu dodajemo Jump Force, odmah ga pomerimo na gore, i onda oduzmemo jedan skok. Razlog zašto ga odmah pomeramo na gore je zato da ne bi slučajno video zemlju do sledeće provere gravitacije i poremetio metodu.

#### JumpingAndGravity()

Vrlo malo se skače u ovoj metodi koliko se bavi graitacijom, valjalo bi promeniti ime metodi.

Prvo i osnovno, proveravamo da li je igrač na zemlji tako što koristimo:

*Physics.CheckSphere(feet.position, groundCheckRadius, groundLayer);*

Potencijalna mana ovog pristupa je što ako je groundCheckRadius prevelik, igrač će previše rano da detektuje zemlju i da lebdi u vazduhu.

Onda imamo jedan if koji proverava da li je igrač na zemlji.

Ako jeste, stavljamo mu vertikalnu brzinu u neki minus (ne 0, jer će u suprotnom lebdeti iznad zemlje, zavisno od groundCheckRadius-a) i resetujemo mu broj skokova.

Ako nije, na vertikalnu brzinu mu oduzimamo gravitaciju (pomnozenu naravno sa Time.deltaTime jer se ovo ipak poziva u Updateu).

Nakon if-a, pomeramo igrača vertikalno u skladu sa vertikalnom brzinom. *Ovo je dakle jedan od dva Move() poziva za isti karakter kontroler u jednoj skripti.*

#### Moving()

Pozivamo svaki Update(), i prvo gledamo da li imamo movement input (WASD ili strelice) i ako je vrednost inputa veca od 0.1 (to jest ako igrac drži input bilo kako) onda:

* Prvo definišemo targetAngle koji određike za koliko se igrač okreće ako drži ASD ili donje tri strelice, tako što koristimo Mathf.Atan2(x, z) \* Mathf.Rad2Deg i na to dodajemo rotaciju kamere kako bi se okretao u zavisnosti od njenog položaja
* Dalje smoothujemo rotaciju sa angle varijablom koja je u suštini Mathf.SmoothDampAngle() između trenutne pozicije i targetAngle, koristeći smoothRotationTime
* Zadajemo igračevu rotaciju da bude angle na Y osi
* Pravimo novu varijablu koja je moveDir gde u suštini množimo igračevu Y rotaciju sa Vector3.forward da bi dobili finalni smer kretanja
* Na samom kraju normalizovani moveDir množimo sa moveSpeed i Time.deltaTime stavljamo u Move() metodu kontrolera. *Ovo je drugi move koji zovemo u Updateu.*

### Korutine

Korutine su metode nasleđene kroz MonoBehaviour koje koriste neki tip brojača kako bi neke funkcije obavili posle nekog delay-a. Iniciraju se sa IEnumerator *ime\_korutine*(), uzimaju iste parametre kao metode, i pozivaju se kroz StartCoroutine(*ime\_korutine()*).

#### DashCooldown()

Koristimo WaitForSeconds da bi posle dashCooldown vremena (u sekundama) resetovali canDash boolean.

## Cinemachine

Cinemachine je Unity plugin koji se sastoji iz: Praznog objekta koji drži logiku kretanja Cinemachine, kod nas imenovan Third Person Camera, kao i Main Camere koja ima na sebi dodat Cinemachine Brain komponentu, koja sluša komande kako da se pomera. Unutar te Cinemachine trenutno šta sam podesio je da prati Player prefab, da gleda na LookAtMe objekat, kao i Bottom, Mid i Top rig, ali sve promene su podležne promenama od strane Dizajnera.

Note: Istražiti in-depth kako Cinemachine zapravo funkcioniše.