**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](#_Toc121324068)

[**PLAYER CONTROLLER** 2](#_Toc121324069)

[Player Prefab setup 2](#_Toc121324070)

[Unity Character Controller 2](#_Toc121324071)

[Player Controller Script 2](#_Toc121324072)

[Klasa *AnimatorController* 3](#_Toc121324073)

[Public/Inspector variable 3](#_Toc121324074)

[Privatne/skrivene variable 4](#_Toc121324075)

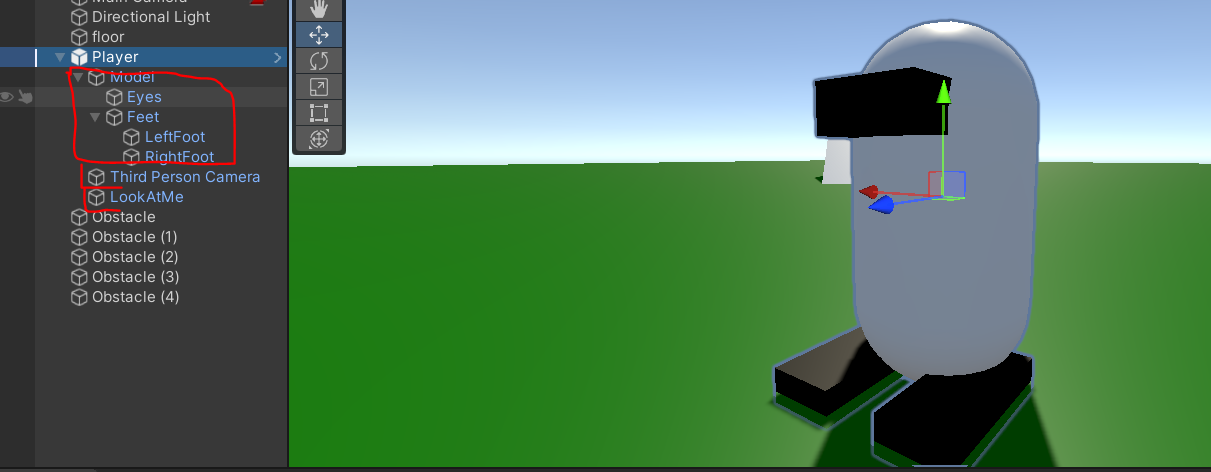
[Metode 5](#_Toc121324076)

[Korutine 7](#_Toc121324077)

[Cinemachine 7](#_Toc121324078)

# **PLAYER CONTROLLER**

## Player Prefab setup

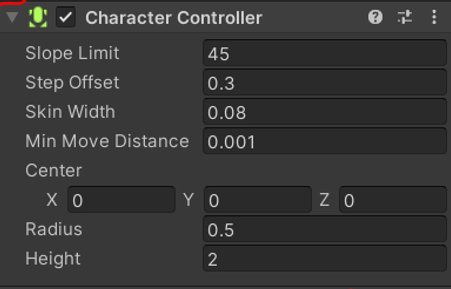


Player prefab je namešten tako da je parent objekat polu-prazan, to jest samo drži skripte i Character controller komponentu. Ispod njega se nalazi model child objekat koji drži krafiku za igrača, kao i Third Person Camera, koja drži Cinemachine kameru, i LookAtMe objekat, koji je prazan objekat u koji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

### Klasa *AnimatorController*

Ova podklasa služi da se Animator i njegovi *state*-ovi lakše i intuitivnije kontrolišu u zavisnosti od player input-a. Sadrži par varijabli, konstruktor, i seriju *public* metoda koje nameštaju vrednosti unutar animatora.

#### Varijable AnimatorController-a

* Transform *modelTransform*: varijabla koja uzima model igrača i prati mu transform, tj poziciju u svetu.
* Animator *modelAnimator*: varijabla koja čuva animator samog modela, koji je najbitniji u kontrolerum očigledno

#### Konstruktor AnimatorController-a

Da bi ova klasa bila pozvana i korišćena, treba da koristimo konstruktore, i trenutno ima samo jedan, *public AnimatorController(Transform transform) { }.* Dakle kada ga pozivamo treba da mu damo neku referencu *transform*a na koju će da se „zakači“ da bi znao na čemu šta treba da radi. U samom kodu on menja svoje varijable tako da *modelTransform* bude onaj transform koji smo mu zadali, a *modelAnimator* nalazi animator komponentu na tom objektu. **To znači da animator komponenta treba da bude na model objektu, i nigde više.**

#### Metode AnimatorController-a

##### *RotateModel(Vector3 rotation)*

Namešta **lokalnu** rotaciju animiranog modela na kojugod vrednost da smo uneli u zagrade metode. Koristi se za slide, privremeno.

##### *SetSpeed(float speed)*

Namešta „Speed“ varijablu animatora na osnovu unešenog broja *speed*. To se kasnije meri kao input za pomeranje da se zna da li da se koristi walk animacija ili run animacija.

##### *SetDirection(float direction)*

Namešta „Direction“ varijablu animatora na osnovu broja *direction*. To određuje na koju se stranu okreće igrač. -1 je levo, 0 je pravo, 1 je desno.

##### *SetJumpTrigger()*

Namešta „Jump“ *trigger* animatora. Koristi se da bi animator znao kada da pređe u animaciju skoka. *Trigger* varijable su slične booleanu, ali ostaju aktivne samo jedan frame (ili dok se ne izvrši tranzicija).

##### *SetJumpHeight(float jumpHeight)*

Namešta „JumpHeight“ varijablu animatora na osnovu broja *jumpHeight*. Ne znam šta treba da radi u trenutnom animatoru.

##### *SetGravityControl(float gravityControl)*

Namešta „GravityControl“ varijablu animatora na osnovu broja *gravityControl*. Ne znam šta treba da radi u trenutnom animatoru.

### 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 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 Slowdown: Koliko brzo se igračev dash usporava, množi se sa *Time.deltaTime*. Što je veća cifra, kraći je dash, to jest brže usporava.
* Dash Speed Limit: Granica, kada se dash uspori do neke tačke, kada se prekida dashovanje. Što je veća cifra, to se brže i more suddenly prekine dash
* Dash Cooldown: Vreme koje mora da prođe da bi igrač mogao da dashuje nakon jednog dash-a
* Sliding Distance: Razdaljina u metrima koju igrač prolazi kada *slide-*uje.
* Sliding Speed: brzina kojom se igrač kreće kada slideuje.
* Maximum Slide Momentum: maksimalni momentum koji igrač može da nakupi kada slideuje. To je vrednost koja se dodaje na skok ili dash igraca nakon što završi *slide*.
* Slide Cooldown: koliko sekundi treba da prođe pre nego što igrač može opet da *slide*uje.

### Privatne/skrivene variable

* Novog Input Sistema
* *moveDirection* koji određuje pravac kretanja IZ INPUTA
* *verticalVelocity* koji određuje kretanje gore i dole
* *dashingVector* koji je privremena variabla koja određuje kuda i kako igrač dash-uje
* *startSlidePos* koji pamti početak *slide*ovanja i od njega meri koliko smo metara prešli.
* Character Controller referenca
* Transform glavne kamere
* Transform onoga u šta kamera cilja
* Referenca za samu kameru (kamera komponenta tačnije)
* Referenca za *AnimatorController*.
* *turnSmoothVelocity* koji je prazan float namenjen da služi kao referenca za okretanje lika
* *momentum* je broj koji raste dok slideujemo sa karakterom.
* *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
* *dashing*, koji proverava da li treba da zovemo Dashing() metodu ili ne
* *canSlide,* boolean koji određuje da li možemo ili ne možemo da *slide*ujemo.
* *sliding,* koji proverava da li zovemo Move() ili Slide() metodu.
* Lista svih InputActiona koje skripta koristi (move, look, fire i jump)
* Varihable za *AnimatorController: moveValueForAnimator, jumpHeightValueForAnimator, jumpBoolForAnimator,* opis njihove upotrebe se može naći u dokumentaciji za *AnimatorController*.

### 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, šta kamera gleda, i nameštamo *AnimatorController* sa *transform*om modela našeg karaktera.

#### 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 *cameraDivider*-om pomnoženim sa pozicijom samog igrača, i onda to sve opet podelimo sa *cameraDividerom*. Na kraju poziciju gledanja kamere namestimo na tu poziciju.

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

Skakanje, Gravitacija, *Dashing*, *Sliding* i *Moving* su zasebne metode koje pozivamo svaki frame Updatea. Treba naglasiti da *if* funkcija ne dozvoljava da *Sliding* i *Moving* budu pozvane u isto vreme, što znači da igrač ne može da se pomera dok kliza.

Privremeno ovde takođe i pozivamo *animatorController* da namestimo brzinu i direkciju kretanja animatora.

#### 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), uključi boolean *dashing* kako bi igrač počeo da poziva istoimenu metodu u Update-u, i pokrene korutinu za cooldown, koja odbrojava kada igrač može opet da dashuje, takođe namesti vertikalnu brzinu na 0. Nakon toga uzimamo **forward vector glavne kamere,** brišemo koju god vrednost da ima na Y osi, i to koristimo kao smer kretanja za dash, tako što normalizujemo vektor, množimo sa *dashDistance*, i opet brišemo vertikalne vrednosti, for safety. Takođe, resetujemo skokove. **Ako igrač klizi,** to jest *slide*uje, onda prekidamo sliding i koji god momentum da je imao igrač ga dodaje na distancu koju prekriva *dash*om, i resetuje rotaciju modela u *animatorController*u.

#### Dashing()

Uzimamo prvo da proverimo *dashingVector,* to jest njegovu veličinu (*magnitude*), da li je veći od limita kojeg smo postavili u *dashSpeedLimit.*

Ako jeste:

Zovemo *controller.Move()* metodu sa *dashingVector*-om, množimo sa *Time.deltaTime,* i onda taj vektor *Lerp*-ujemo do nule kroz *dashSlowdown (pomnožen sa Time.deltaTime, naravno)*.

Ako nije:

Namestimo vrednost *dashingVector*-a da bude 0 i isključimo *dashing* boolean, čisto da ne bi zvali ovu metodu kada ne treba.

U praksi, ovo znači da kada ubacimo *Dash* input, ta metoda namesti vektor koji koristimo za dashovanje, i svaki frame mi taj vektor za malo smanjujemo, dok ne dođemo do neke male granice, kada jednostavno smanjimo na 0 taj vektor i prestajemo sa dashovanjem.

#### Slide(InputAction.CallbackContext context)

Proveravamo prvo da li je igrač na zemlji i da li može da *slide*uje, ili je još na *cooldown*u. Ako može, onda uključimo *sliding* boolean, isključimo *canSlide,* otpočnemo *SlideCooldown* korutinu, i zapamtimo početnu poziciju našeg *slide*ovanja.

#### Sliding()

Čitamo horizontalni *input* igrača da bi mogao da kontroliše smer klizanja, i onda ga pomeramo napred (njegovo napred, ne kamerino napred), i dodajemo momentum, tako što Lerp-ujemo (linearno interpoliramo) *momentum* varijablu, između vrednosti 0 i *maximumSlideMomentum* varijable, na osnovu kog dela puta smo prešli klizeći, što znači da je najveći momentum na samom kraju klizanja. Privremeno krivimo model igrača, da bi znao da trenutno klizi, i onda proveravamo da li smo prešli ograničenu distancu klizanja, i ako jesmo, prestajemo sa klizanjem, i rotiramo model na normalnu rotaciju. Valja naglasiti da pošto pomeramo u *Update* i da to radimo tako što dodajemo vrednost, postoji mala šansa da igrač pređe malo veću distancu, ali za zanemarljivu vrednost.

#### 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 *JumpSpeed*, 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. Takođe namešamo animator varijablu u *true*. **Ako je *sliding* aktivan,** onda takođe prestajemo klizanje i dodajemo vrednost momentuma na sam skok.

Takođe, ako je prvi skok u seriji, onda okinemo *jump trigger* vrednost u animatoru.

#### 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.

#### SlideCooldown()

Koristimo *WaitForSeconds* da bi posle *slideCooldown* vremena (u sekundama) resetovali *canSlide* 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.