REPUBLIKA SLOVENIJA
MINISTRSTVO ZA VZGOJO IN IZOBRAŽEVANJE

teach XR

NAČRT ZA OKREVANJE IN ODPORNOST

Financira Evropska unija
NextGenerationEU

Unity: Ustvarjanje interaktivnih 3D izkušenj

2026

Unity:
Ustvarjanje interaktivnih 3D izkušenj

Marija Absec, Uroš Šmajdek, Klara Žniderščič, Klemen Pečnik, Žana Juvan,
Matija Marolt, Matevž Pesek

Kazalo

Bralcu na pot

Mnogi verjamejo, da je ustvarjanje interaktivnih 3D izkušenj zahtevno opravilo, ki je rezervirano le za strokovnjake s področja računalniške grafike in programiranja. Toda resnica je drugačna. V svet okolja Unity lahko vstopite tudi brez predhodnega znanja o programiranju v jeziku C#, matematičnih osnov grafike ali poznavanja delovanja igralnih pogonov. Ker v slovenskem prostoru primanjkuje strokovne literature s tega področja, ki bi praktično delo povezovala s poglobljeno razlago teoretičnega ozadja, smo avtorji pripravili gradivo, s katerim želimo to področje približati popolnim začetnikom.

Sprehodili se bomo skozi ključne korake razvoja 3D izkušenj s pomočjo igralnega pogona Unity: od namestitve urejevalnika, do ustvarjanja s kompleksnimi objekti in scenariji. Poglavja se začnejo z razlago teoretičnega ozadja in so nato podkrepljena s konkretnimi primeri in izzivi. V zaključku se dotaknemo tudi področja tehnologij razširjene resničnosti (XR), ki danes narekujejo smer razvoja digitalnih vsebin.

Vsebina temelji na izkušnjah iz izobraževanj v sklopu projekta *TeachXR* ter uradnih Unity vodičev [5]. Vsi primeri so bili pripravljeni v različici *Unity 6 (6000.2.9f1)*.

Navodila za sledenje primerom

Za lažje razumevanje ukazov in možnosti v vmesniku smo v besedilu uvedli naslednje oznake:

`Gumb` označuje elemente vmesnika, na katere je treba klikniti (npr. potrditveni gumbi, ikone).

≡ *Spustni seznam* ▶ *Možnost* označuje ukaze v spustnih seznamih (npr. v zgornji orodni vrstici ali v kontekstnem meniju ob desnem kliku). Zaporedje besed nakazuje pot skozi podsezname.

Parameter/polje označuje nastavitve znotraj različnih oken urejevalnika. Navadno gre za polja, ki lahko predvidevajo vnos besedilnih ali številskih vrednosti ali pa izbiro s pomočjo drsnika ali potrditvenega polja.

Objekt označuje besede, ki se nanašajo na objekte v sceni. Takšna oznaka pripomore k razločevanju med splošnim samostalnikom in konkretnim elementom v sceni.

Avtorstvo slik: Vse slike in posnetki zaslona v učbeniku so, razen kjer je navedeno drugače, lastni avtorski material.

POGLAVJE 1

Kaj je Unity?

Unity je igralni pogon, namenjen ustvarjanju video iger, animacij, vsebin za navidezno in obogateno resničnost (NR in OR, angl. VR in AR)[1], simulacij ter drugih 2D ali 3D interaktivnih izkušenj. Igralni pogon (*angl. game engine*) je celovita programska platforma, ki razvijalcem nudi vsa potrebna orodja za razvoj interaktivnih vsebin na enem mestu. Večinoma so ti pogoni del integriranih razvojnih orodij (*angl. integrated development environment – IDE*), kar omogoča učinkovit potek dela.

Namesto da bi razvijalci za vsak projekt znova pisali kodo za osnovne sisteme, igralni pogon običajno že vključuje podporo za:

- **upravljanje scene:** organizacijo objektov v navideznem prostoru;
- **grafični pogon:** upodabljanje 2D in 3D vizualnih elementov;
- **fizikalni pogon:** simulacijo gravitacije, trkov in drugih fizikalnih zakonov;
- **sistem za animacije:** upravljanje gibanja likov in predmetov;
- **zvočni sistem:** upravljanje prostorskega zvoka in mešanja zvočnih virov.

Unity je le eden izmed številnih igralnih pogonov, ki so danes na voljo na trgu. Ustvarila ga je skupina razvijalcev z Danske z vizijo, da »demokratizirajo razvoj iger« – torej ustvarijo pogon, ki podpira več platform, je finančno dostopnejši in enostaven za uporabo. Pred pojavom Unityja so bili zmogljivi pogoni praviloma izjemno dragi in dostopni le velikim korporacijam. Prva različica programa je bila predstavljena leta 2005, nato pa je razvoj potekal izjemno hitro in do leta 2015 je bila izdana že peta različica, ki je prinesla pomembne izboljšave predvsem na področju grafične kakovosti. V nadaljevanju so razvijalci prešli na sistem poimenovanja različic po letnicah izida, kar je uporabnikom olajšalo orientacijo med posameznimi verzijami. Leto 2024 predstavlja mejnik z izidom različice Unity 6.0, ki nadaljuje razvoj platforme z izboljšavami na področju upodabljanja, zmogljivosti in podpore sodobnim napravam. Danes Unity poganja več kot polovico vseh mobilnih iger na svetu, kar jasno kaže na njegovo razširjenost, prilagodljivost ter širok spekter možnosti uporabe.

[1] V nadaljevanju bomo uporabljali angleški kratici VR in AR.

1.1 Unity v praksi: več kot le igre

Glavna prednost pogona Unity je enostavna prenosljivost med različnimi platformami (*angl. cross-platform development*). Razvijalci lahko svojo vsebino brez večjih sprememb objavijo na več kot 25 različnih platformah, vključno z mobilnimi napravami, igralnimi konzolami, osebnimi računalniki in napravami za navidezno resničnost. Trenutno sta med najbolj priljubljenimi igralnimi pogoni Unity[2] in Unreal Engine[3], Unity pa je postal razširjen predvsem zaradi svoje dostopnosti, fleksibilnosti in ugodne licenčne politike za manjše razvijalce.

Danes Unity ni več omejen le na industrijo video iger, temveč se intenzivno uporablja tudi v:

- **arhitekturnih vizualizacijah** (*angl. ArchViz*): ustvarjanje navideznih sprehodov po zgradbah, ki še niso zgrajene;
- **avtomobilski industriji**: simulacije vožnje in interaktivni prikazi novih modelov vozil;
- **izobraževanju in usposabljanju**: realistične simulacije za kirurgijo, vojaško usposabljanje ali varno usposabljanje za delo z nevarno opremo;
- **filmih in animacijah**: hitro upodabljanje v realnem času (*angl. real-time rendering*), ki spremlja filmsko produkcijo.

1.2 Osnovna terminologija razvoja s pogonom Unity

V svetu Unityja končni izdelek (igro, aplikacijo ali simulacijo) imenujemo interaktivna izkušnja. Ta predstavlja celoto, ki jo uporabnik izvaja na ciljni napravi. Vsaka izkušnja je strukturno sestavljena iz ene ali več scen, ki delujejo kot samostojni segmenti ali nivoji navideznega sveta – na primer glavni meni, posamezne stopnje ali nastavitve. Scena predstavlja podatkovni vsebnik, v katerem so shranjeni vsi elementi, ki so v določenem trenutku naloženi v pomnilnik in aktivni v okolju. Te elemente imenujemo igralni objekti (*angl. game objects*) in so prostorsko definirani v 2D ali 3D koordinatnem sistemu scene.

Igralni objekt je osnovni gradnik in temeljna entiteta vsake scene. Njegova primarna vloga je nosilec identifikacije, hierarhije in prostorske postavitve. Objekti se glede na svojo vlogo v sceni delijo v dve skupini:

- **vsebinski (vidni) objekti**: tisti, ki imajo vizualno reprezentacijo, kot so igralni liki, okolica, vegetacija ali elementi uporabniškega vmesnika;
- **funkcionalni (nevidni) objekti**: entitete, ki skrbijo za tehnično izvedbo izkušnje, npr. kamere, ki določajo zorni kot izrisa, viri svetlobe, prostorski sprožilci (*angl. triggers*) ali kontrolni objekti, ki vsebujejo zgolj programsko logiko za upravljanje stanja aplikacije.

[2] `https://unity.com/`
[3] `https://www.unrealengine.com/`

Igralni objekt sam po sebi nima vnaprej določene funkcionalnosti, temveč je v jedru sistema definiran zgolj z umestitvijo v koordinatni sistem. Vse ostale lastnosti objekt pridobi šele preko komponent.

Komponenta je modularna in specializirana enota, ki igralnemu objektu določi specifičen videz, fizikalne lastnosti ali vzorce obnašanja. Unity temelji na modularnem sestavljanju objektov, kar pomeni, da objektov ne gradimo s kompleksnim dedovanjem razredov, temveč s sestavljanjem oz. pripenjanjem različnih komponent. S komponentami določamo lastnosti, vezane na upodabljanje (npr. določanje materialov), fizikalne lastnosti (npr. simulacija togega objekta, zaznavanje trkov) in programsko logiko, ki določa dodatna logična pravila za obnašanje objektov.

Vse te elemente urejamo v urejevalniku Unity (*angl. Unity Editor*). Ta predstavlja osrednje razvojno okolje, ki razvijalcu omogoča preplet vizualnega oblikovanja in naprednega urejanja. Urejevalnik ponuja pester nabor specializiranih orodij za:

- **upravljanje z viri**: uvažanje, optimizacijo in organizacijo digitalnih virov, kot so 3D modeli, zvočni zapisi in teksture;
- **vizualno manipulacijo**: urejanje scene v realnem času, kjer lahko razvijalec neposredno vpliva na prostorske odnose med objekti;
- **napredno avtomatizacijo**: uporaba orodij, kot sta Shader Graph za vizualno programiranje senčilnih algoritmov in ProBuilder za neposredno geometrijsko modeliranje znotraj razvojnega okolja.

Ta sinergija med modularno zasnovo komponent in obsežno podporo integriranih orodij omogoča, da je Unity primeren tako za hitro prototipiranje idej kot za razvoj tehnološko zahtevnejših industrijskih rešitev.

POGLAVJE 2

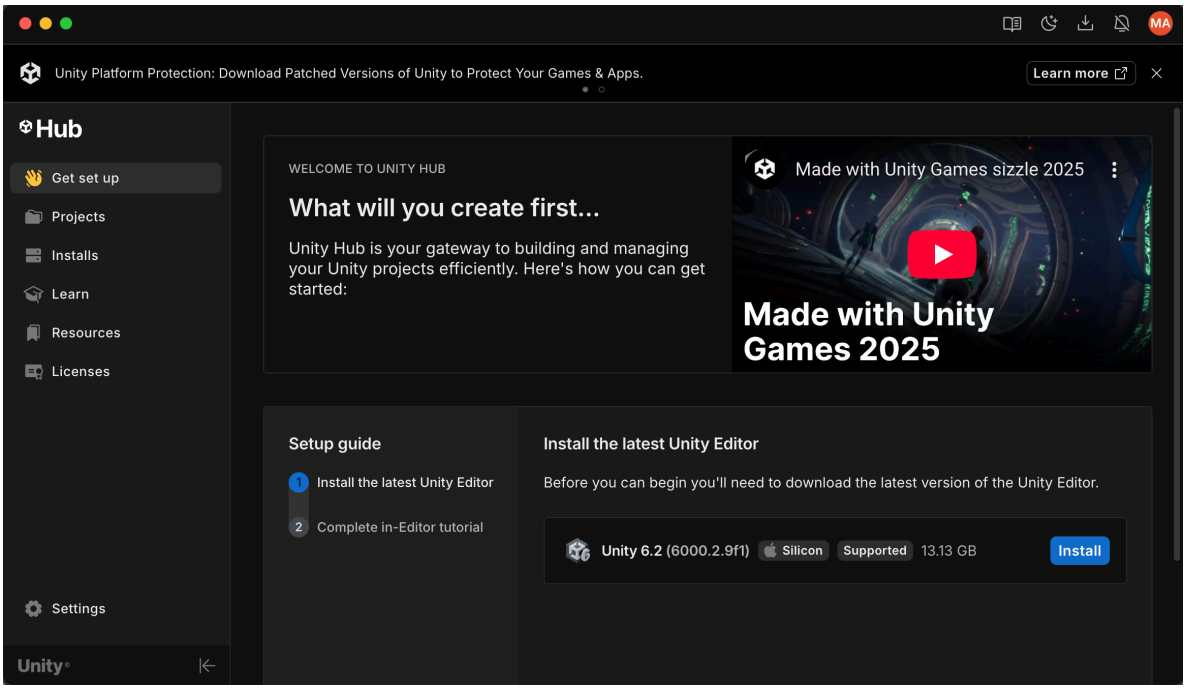
Ustvarjanje interaktivnih izkušenj v pogonu Unity

Unity ponuja obsežen nabor možnosti za razvoj interaktivnih izkušenj. Medtem ko večino neposrednega razvoja, kot sta snovanje scen in konfiguracija objektov, opravimo v urejevalniku Unity, je za stabilen razvojni proces ključna aplikacija Unity Hub. Ta služi kot nadzorna plošča za vse razvojne dejavnosti. Njena vloga ni zgolj zagon urejevalnika, temveč tudi zagotavljanje konsistence med različnimi generacijami pogona, ki se skozi čas nenehno posodabljajo. Unity Hub omogoča učinkovito upravljanje projektov na enem mestu, nudi pregled nad nameščenimi različicami urejevalnikov ter služi kot stična točka za dostop do dokumentacije, licenc in uporabniškega profila. Poleg tega skrbi za organizacijo projektov glede na lokacijo na disku ter hitro preklapljanje med njimi, kar je še posebej pomembno pri delu na več projektih hkrati ali v sodelovalnih okoljih.

2.1 Unity Hub

Unity Hub je uradna namizna aplikacija, ki je namenjena upravljanju Unity projektov in različic urejevalnika. Aplikacija je postala nepogrešljiv del delovnega toka zaradi kompleksnih odvisnosti med zunanjimi knjižnicami in različicami pogona. Projekte v Unityju je namreč pogosto mogoče uspešno urejati le znotraj tiste različice urejevalnika, v kateri so bili ustvarjeni. Prilagajanje starejšega projekta novejšim različicam je zahteven proces, ki lahko povzroči napake v delovanju.

Zato Unity Hub vsakemu projektu pripiše specifično različico urejevalnika, hkrati pa razvijalcu omogoča, da ima na svojem računalniku nameščenih več različic urejevalnikov hkrati. Prav tako omogoča namestitev dodatnih razvojnih modulov, kot so podpora za platforme Android, iOS ali WebGL, sistem za pripravo VR aplikacij ter integrirano razvojno okolje za pisanje programske kode *Visual Studio*. S tem Unity Hub ne deluje zgolj kot upravljalnik različic, temveč kot centralno okolje za pripravo razvojnega sistema, ki ga lahko prilagodimo specifičnim zahtevam posameznega projekta ali ciljne platforme.



Slika 2.1: Uporabniški vmesnik Unity Hub po uspešni prijavi.

2.1.1 Namestitev in prijava v Unity Hub

Unity Hub je namizna aplikacija za računalnik, ki za uporabo zahteva prijavo z uporabniškimi podatki. Za namestitev in prijavo v aplikacijo sledimo spodnjim korakom:

1. **Namestitev aplikacije:** Unity Hub namestimo tako, da sledimo navodilom na uradni spletni strani: `https://unity.com/download`. Ob tem moramo izbrati različico, primerno za naš operacijski sistem.
2. **Ustvarjanje računa:** Aplikacijo zaženemo. Če še nimamo Unity računa, ga na tej točki ustvarimo.
3. **Prijava:** V aplikacijo se prijavimo s svojimi uporabniškimi podatki.

Na sliki 2.1 je prikazan pogled aplikacije, ki se odpre ob prijavi. Aplikacija je sestavljena iz treh glavnih delov: orodnega menija na levi, osrednjega dela za prikaz vsebine in orodne vrstice na vrhu. Vmesnik je trenutno na voljo le v angleškem jeziku. Orodna vrstica na vrhu ponuja hitre bližnjice do uradne dokumentacije, nastavitve barvne teme, spremljanje aktivnih prenosov, dostop do obvestil in informacij o uporabniškem računu.

Navigacijski meni na levi strani je razdeljen na več namensko ločenih zavihkov. Ko prvič odpiramo aplikacijo, se na začetku prikaže zavihek Začetek (*angl. Get set up*), ki nas vodi skozi postopek prenosa Unity urejevalnika in osnovno konfiguracijo. Sledi zavihek Projekti (*angl. Projects*), ki prikazuje seznam vseh naših Unity projektov. Če še nismo ustvarili nobenega projekta, je ta seznam prazen. Tukaj lahko ustvarjamo, odpiramo in upravljamo svoje projekte.

Zavihek Nameščeni urejevalniki in orodja (*angl. Installs*) prikazuje seznam vseh nameščenih različic urejevalnika Unity s pripadajočimi nameščenimi orodji. Omogoča enostavno namestitev poljubnih različic urejevalnika in drugih orodij. Zavihka Učenje (*angl. Learn*) in Viri (*angl. Resources*) ponujata neposreden dostop do različnih podpornih vsebin, kot so interaktivni vodiči, dokumentacija, uporabniški forumi, skupnosti razvijalcev, in trgovina Unity Asset Store. Ti viri predstavljajo pomembno podporno okolje za razvijalce, saj omogočajo hitrejše spoznavanje funkcionalnosti pogona, lažje reševanje tehničnih izzivov ter dostop do že pripravljenih vsebin, ki lahko bistveno pospešijo razvoj. Zadnji zavihek, Licence (*angl. Licenses*), vsebuje seznam aktivnih Unity licenc, vezanih na uporabniški račun. Pri prvem odprtju aplikacije je vsakemu uporabniku dodeljena brezplačna osebna licenca, možen pa je nakup plačljivih licenc[1], ki omogočajo uporabo naprednejših funkcionalnosti.

2.2 Ustvarjanje projekta

Vsaka interaktivna izkušnja v okolju Unity se začne z ustvarjanjem novega projekta. Projekt ni le mapa na disku, temveč zaokrožena celota, ki vključuje vse vire (*angl. assets*) , nastavitve, 3D ali 2D scene, digitalne objekte ter celotno programsko logiko. Strukturno je projekt definiran kot nabor povezanih datotek v skupni mapi, kar zagotavlja visoko stopnjo organiziranosti podatkov in omogoča preprosto prenašanje celotnega razvojnega okolja.

Postopek priprave delovnega okolja poteka centralizirano prek aplikacije **Unity Hub**. Preden ustvarimo svojo prvo sceno, moramo izvesti tri ključne korake: namestitev ustrezne različice urejevalnika, izbiro projektne predloge in končno konfiguracijo projekta.

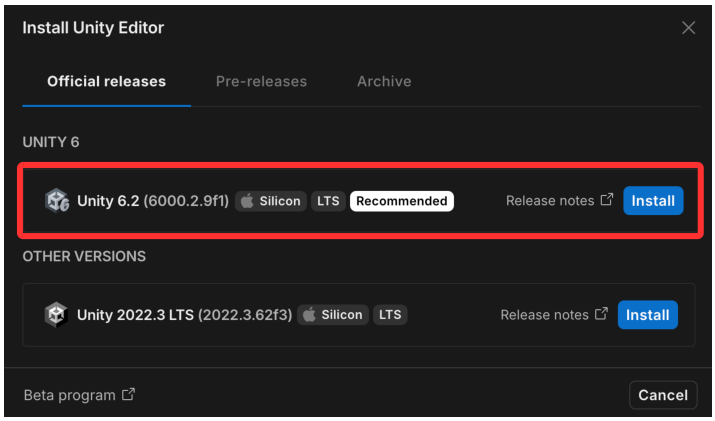
Primer 2.1: Namestitev urejevalnika

Za razvijanje in urejanje projektov potrebujemo nameščen urejevalnik Unity. Urejavalnik namestimo v zavihku Nameščeni urejevalniki in orodja (*angl. Installs*) s klikom na gumb Install Editor *(sl. namesti urejevalnik)*.

Odpre se pojavno okno, podobno prikazu na sliki 2.2. Izberemo različico, ki jo želimo namestiti. Priporočljivo je izbrati najnovejšo podprto različico, ki je običajno označena kot priporočena (*angl. Recommended*).

Nato se odpre okno za izbiro dodatnih modulov. Za nemoteno delo in pisanje programske kode je ključno, da označimo **Visual Studio** (ali drugo želeno razvojno okolje) ter izbiro potrdimo z gumbom Install *(sl. namesti)*. Stanje prenosa lahko spremljamo v zgornji orodni vrstici. Pred nadaljevanjem počakamo, da se namestitev zaključi.

[1] `https://unity.com/products`



Slika 2.2: Pojavno okno za namestitev primerne različice urejevalnika Unity.

2.2.1 Upravljanje s projekti

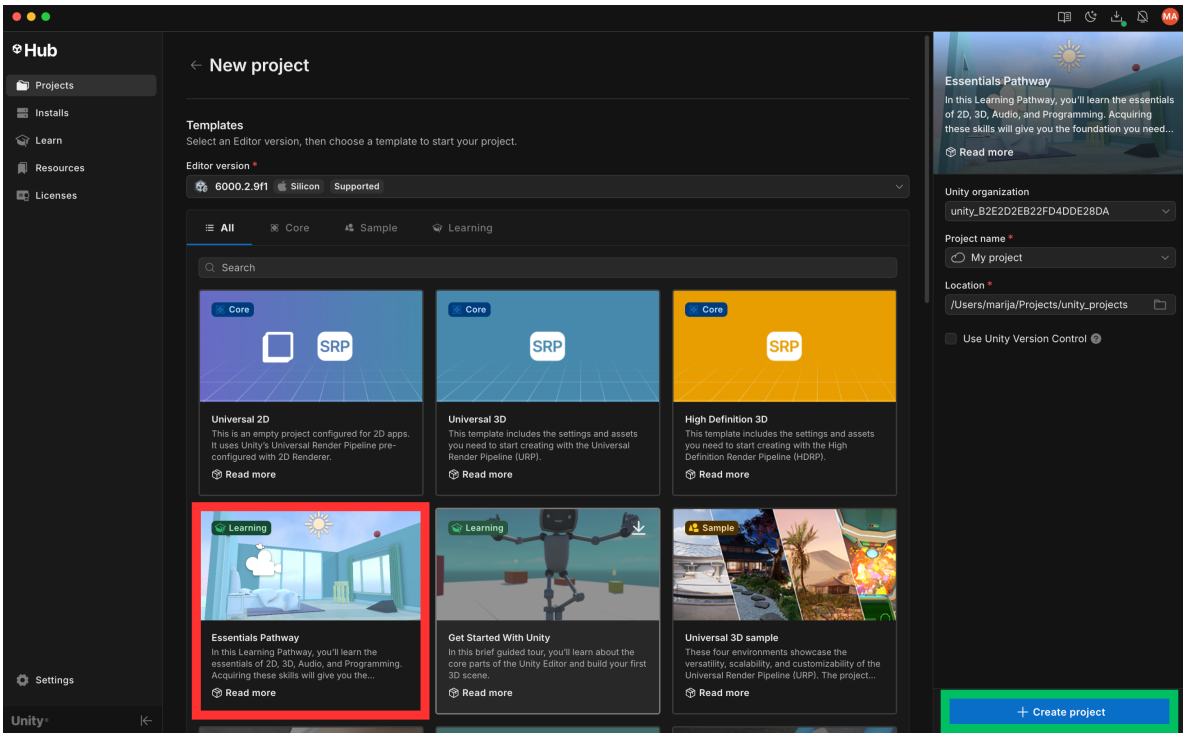
S svojimi projekti upravljamo v zavihku Projekti. Če želite uvoziti obstoječ projekt (npr. prenesen s spleta ali shranjen na zunanjem disku), ga na seznam dodate z gumbom Add *(sl. dodaj)*. Unity Hub nato vzpostavi povezavo in samodejno zazna zahtevano različico urejevalnika.

Primer 2.2: Ustvarjanje novega projekta s predlogo

Za ustvarjanje novega projekta izberemo gumb New project *(sl. nov projekt)*. Odpre se konfiguracijsko okno za izbiro predloge, kot prikazuje slika 2.3. V zgornjem delu lahko s spustnim seznamom določimo privzeto različico urejevalnika.

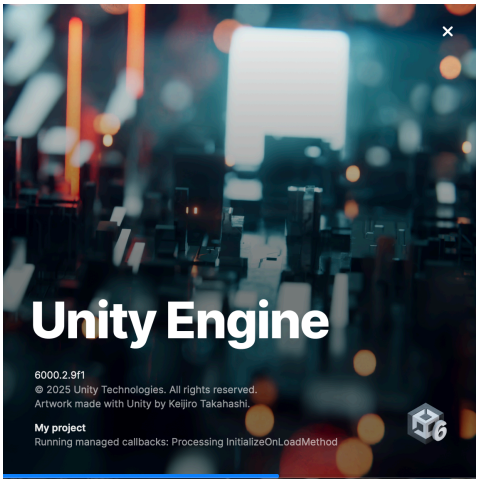
Nato izberemo eno izmed predlog za projekt, ki določajo nabor privzetih nastavitev in vsebin. Za spremljanje primerov v tem učbeniku izberemo predlogo Essentials Pathway [5]. Ta predloga vključuje uradne vodiče in vnaprej pripravljene učne materiale. Podrobnosti o izbrani predlogi so prikazane na desni strani okna. Če predloga še ni nameščena na napravi, jo je treba najprej prenesti z gumbom Download template *(sl. prenesi predlogo)*.

Po prenosu v desnem stolpcu določimo unikatno ime projekta in ciljno lokacijo na disku. Postopek zaključimo z izbiro gumba Create project *(sl. ustvari projekt)* (na sliki 2.3 označen z zelenim pravokotnikom). Ob tem se samodejno sproži proces ustvarjanja datotečne strukture in zagon urejevalnika.



Slika 2.3: Pogled pri ustvarjanju novega projekta, ki omogoča izbiro različice urejevalnika ter začetno predlogo.

Ko je projekt uvrščen na seznam v zavihku Projekti, ga lahko kadarkoli odpremo z dvoklikom na pripadajočo vrstico. Prvi zagon novega projekta običajno traja nekaj minut, saj mora pogon generirati potrebne knjižnice in indeksirati vire. Napredek procesa lahko spremljate v zagonskem oknu (slika 2.4). Ko je okolje pripravljeno, se odpre glavno okno urejevalnika.



Slika 2.4: Zagonsko okno urejevalnika Unity, ki prikazuje napredek odpiranja projekta.

POGLAVJE 3

Osnove urejevalnika Unity

Urejevalnik Unity je glavno razvojno okolje za ustvarjanje in urejanje interaktivnih vsebin. Njegova zasnova temelji na prilagodljivem vmesniku, ki razvijalcem omogoča dostop do različnih orodij za delo z grafiko, zvokom, fiziko in programsko kodo na enem mestu. V tem poglavju bomo spoznali sestavo vmesnika ter vlogo posameznih oken, ki so ključna za razvojni proces. Poudarek bo na strukturi in organizaciji projekta ter razumevanju hierarhije objektov znotraj scene. To znanje je nujno za kasnejšo implementacijo kompleksnejših funkcionalnosti interaktivne izkušnje.

3.1 Okna urejevalnika

Uporabniški vmesnik urejevalnika Unity je razdeljen na več funkcionalnih razdelkov, ki jih imenujemo okna (*angl. windows*). Osnovna postavitev, prikazana na sliki 3.1, vključuje najpogosteje uporabljena okna.

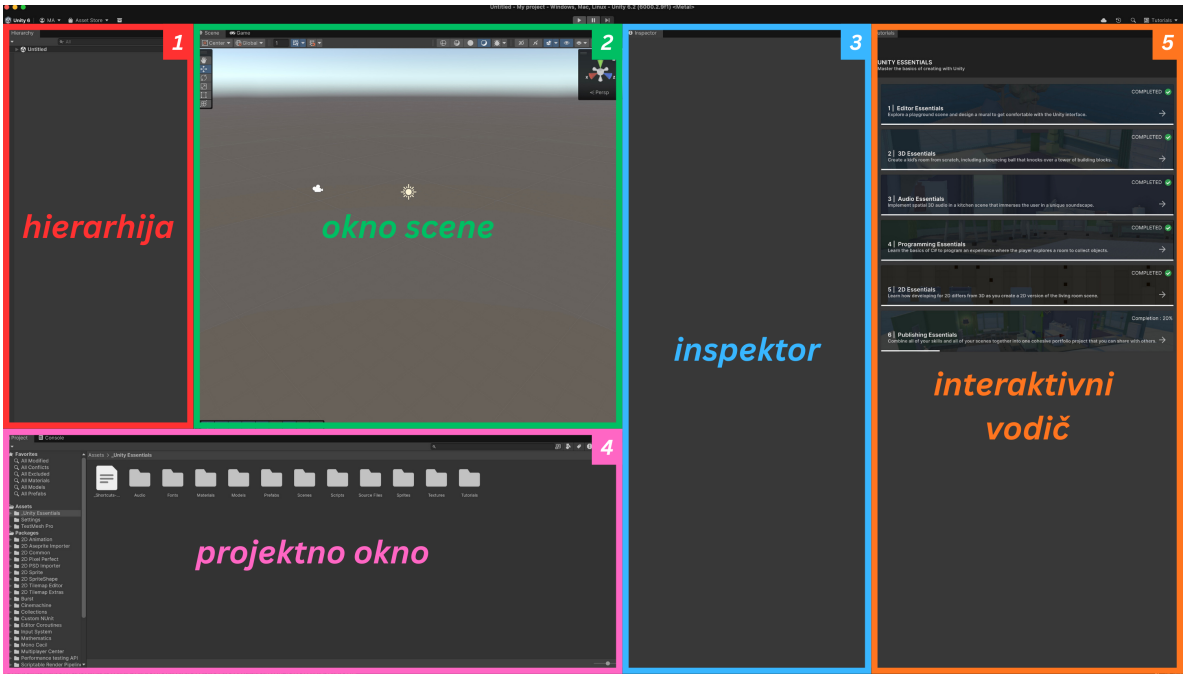
Na levi strani se nahaja okno **hierarhija** (*angl. hierarchy*) (1), v katerem so navedeni vsi objekti trenutne scene, organizirani v hierarhično drevesno strukturo (več o hierarhiji objektov v poglavju 3.2.1). Osrednji del zavzema **okno scene** (*angl. scene*) (2), ki služi kot vizualni prostor za neposredno umeščanje in premikanje objektov v tridimenzionalnem svetu. To okno omogoča prostorsko orientacijo in natančno postavljanje elementov igre.

Lastnosti izbranega objekta v sceni so prikazane v oknu **inspektor** (*angl. inspector*) (3) na desni strani vmesnika. Na spodnjem delu se nahaja **projektno okno** (*angl. project*) (4), ki prikazuje datotečno strukturo projekta. Poleg iskalne vrstice ponuja dvojni pogled: levo stran z datotečnim drevesom za hitro navigacijo med mapami ter osrednji del, ki prikazuje vsebino izbrane mape. V našem primeru imamo na začetku na skrajni desni odprto še okno **interaktivni vodič** (*angl. tutorial*) (5), ki vsebuje navodila za vodeno učenje.

Vmesnik urejevalnika je popolnoma prilagodljiv, kar pomeni, da lahko okna poljubno prerazporejamo, dodajamo ali zapiramo. Vsako okno urejevalnika lahko odstranimo z gumbom s tremi pikami v njegovem zgornjem desnem kotu, kjer izberemo možnost Close Tab *(sl. zapri zavihek)*. Če želimo vmesniku dodati novo okno (ali povrniti tisto, ki smo ga pomotoma zaprli), uporabimo orodno vrstico na vrhu urejevalnika in izberemo

≡ *Window* ▶ *General*. Za dodajanje okna z interaktivnim vodičem v orodni vrstici izberemo ≡ *Tutorials* ▶ *Show Tutorials*. Nova okna se pogosto odprejo v samostojnem (pojavnem) oknu, ki ga v urejevalnik preprosto vpnemo tako, da z miško povlečemo njegov zavihek na želeno mesto znotraj vmesnika.

Za večjo preglednost so posamezni deli urejevalnika organizirani v skupine zavihkov. Tako se ob oknu scene običajno nahaja še okno igre (*angl. game*), v katerem bomo opazovali potek igre z igralčevega zornega kota. Podobno je projektnemu oknu pogosto pridružena konzola (*angl. console*), ki je ključna za razvojni proces, saj sistem vanjo izpisuje vsa obvestila, opozorila in morebitne napake, kar omogoča sprotno razhroščevanje projekta.



Slika 3.1: Unity urejevalnik je razdeljen na okna. Na sliki so oštevilčena glavna okna urejevalnika: (1) hierarhija, (2) okno scene, (3) inspektor, (4) projektno okno in (5) interaktivni vodič. Postavitev oken v urejevalniku je spremenljiva, okna lahko poljubno prerazporedimo, zapremo, odpremo dodatna okna ali pa jih premaknemo v povsem ločen zavihek.

3.2 Scena

V Unityju termin scena označuje dva vsebinsko povezana elementa. Prvi je osnovni gradnik projekta (vsebinska datoteka), ki smo ga omenili v poglavju 1.2, drugi pa je okno scene, kjer vsebino vizualno urejamo. V izogib dvoumnosti, bo v nadaljevanju za orodje urejevalnika dosledno uporabljen izraz okno scene, medtem ko uporabniški vmesnik aplikacije ter vodiči v angleščini pogosto ne razlikujejo med tema pojmoma.

Scena je del ustvarjenega 2D- ali 3D-sveta, s katerim uporabnik interaktira, in je sestavljena iz igralnih objektov. Projekt lahko vsebuje poljubno število scen, vendar je igralec med igranjem (praviloma) lahko prisoten le v eni. Enako velja za proces izdelave, kjer lahko hkrati urejamo le eno, *t. i.* aktivno sceno.

Aktivna scena je prikazana v oknu scene, ki omogoča navigacijo ter izbiranje in urejanje objektov. Po njej se lahko premikamo na več načinov. S kolescem na miški se približujemo ali oddaljujemo od središča pogleda, s pritiskom na kolesce pa se premikamo vstran. Z desnim gumbom lahko obračamo pogled. Med držanjem gumba pa lahko istočasno uporabljamo tipke `W`, `A`, `S` in `D` za premikanje naprej, levo, desno in nazaj. Opisani način gibanja po sceni se na prvi pogled zdi zapleten, vendar je najučinkovitejši in najpreprostejši, ko ga usvojimo. Predmete v sceni izbiramo z levim klikom ali z izbiro vrstice objekta v hierarhiji.

Ob ustvarjanju novega projekta Unity samodejno odpre začetno sceno. Ta je pogosto skoraj prazna, razen pri uporabi specifičnih predlog, ki že vsebujejo nekatere vnaprej nastavljene objekte. Pomembno je vedeti, da začetna scena še ni shranjena na disk, kar pove oznaka na vrhu hierarhije – `*Untitled`. Oznaka predstavlja ime scene, zvezdica na začetku imena pa nam pove, da scena vsebuje neshranjene spremembe. *Untitled* je posebno ime za sceno, ki še ni bila shranjena v pomnilnik. Sceno lahko shranimo s pritiskom bližnjice `CTRL+S` (ali `CMD+S` na macOS) ali z izbiro možnosti ≡ *File* ▶ *Save* v orodni vrstici. Pri prvem shranjevanju se prikaže pojavno okno, v katerem določimo lokacijo scene v pomnilniku in njeno ime. V nadaljevanju bomo za našo sceno uporabljali ime »Demo«. Po uspešnem shranjevanju se shranjeno ime prikaže v zgornjem delu hierarhije. Redno shranjevanje sprememb je ključnega pomena pri delu v Unityju, saj so vse neshranjene spremembe ob zaprtju urejevalnika izgubljene.

Do shranjenih scen dostopamo prek projektnega okna. Aktivno sceno zamenjamo z dvoklikom na njeno ikono ali z desnim klikom in izbiro možnosti ≡ *Open*. Novo sceno ustvarimo z ukazom ≡ *File* ▶ *New Scene*, kjer v pojavnem oknu izberemo želeno predlogo. Nova scena ob ustvarjanju takoj postane aktivna.

3.2.1 Igralni objekt

Preden začnemo ustvarjati kompleksno igro, moramo podrobneje poznati osnovne gradnike vsake scene v Unityju – to so igralni objekti (*angl. game objects*). Če Unityjevo sceno primerjamo z gledališko igro, so igralni objekti vse, kar lahko vidimo na odru, od igralcev in rekvizitov do luči in zvočnikov, ki so skriti nekje v ozadju.

Igralni objekt v Unityju je prazen vmesnik, ki sam po sebi nima določene oblike ali funkcije. Njegov izgled in funkcionalnost urejamo tako, da mu dodajamo poljubne lastnosti. Igralni objekti v sceni so lahko vidni (2D ali 3D), z določeno obliko, izgledom in načinom interakcije z ostalimi objekti. Lahko pa so tudi nevidni objekti, ki v sceni delujejo zgolj s svojimi funkcionalnostmi, na primer luči, zvočni objekti ali učinki. Objekti so lahko tudi povsem brez lastnosti in služijo zgolj organizaciji ostalih objektov v sceni.

Vse igralne objekte, ki so prisotni v sceni, najdemo v oknu hierarhija. Tam so objekti prikazani v seznamu, v tem oknu pa lahko opazujemo tudi razmerja med njimi (razmerja med objekti so podrobneje opisana v poglavju 3.2.2). Kadar je scena prazna, tudi v hierarhiji ne vidimo objektov.

Primer 3.1: Ustvarjanje novega objekta

Nov igralni objekt ustvarimo tako, da v orodni vrstici okna urejevalnika izberemo zavihek ≡ *GameObject* ▶ *3D object* ▶ *Cube (sl. kocka)* (glej sliko 3.2 levo). Vidimo, da se je v hierarhiji pojavila vrstica *Cube*, v oknu scene pa lahko vidimo kocko. Če iskanega objekta ne vidimo v sceni, lahko pogled vedno premaknemo z dvoklikom na njegovo vrstico v hierarhiji.

Urejevalnik omogoča, da vsakemu ustvarjenemu objektu dodelimo poljubno ime. To storimo tako, da izberemo njegovo vrstico v hierarhiji in ob desnem kliku izberemo možnost ≡ *Rename*, nato pa v označeno besedilno okno vpišemo poljubno ime. V našem primeru novo ustvarjeni objekt poimenujemo »Kocka«.

Objekt lahko izberemo s klikom na njegovo vrstico v hierarhiji ali z levim klikom nanj v sceni. Kadar je objekt izbran, so njegovi robovi označeni, inspektor pa prikazuje njegove komponente.

Lastnosti igralnih objektov so določene s tako imenovanimi komponentami, ki so prikazane v inspektorju. Vsak objekt ima komponento **transformacije** (*angl. Transform*), ki mu določa translacijo, rotacijo in velikost. Vsakemu objektu lahko dodamo poljubno mnogo drugih komponent, a nobena od teh ni obvezna. Najprej bomo spoznali komponento transformacije, ostale pa bomo podrobneje opisali v nadaljnjih poglavjih.

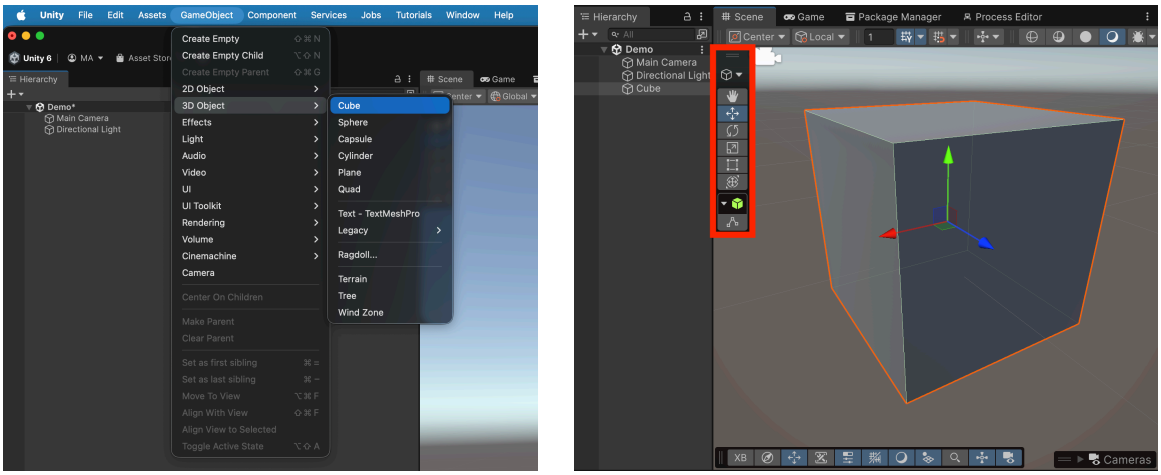
S komponento transformacije objektu določamo translacijo, rotacijo in velikost s spreminjanjem vrednosti parametrov v oknu inspektor. Enak rezultat lahko dosežemo z uporabo vizualnih orodij znotraj okna scene. V oknu scene je več pomožnih orodij, ki omogočajo lažje urejanje objektov in scene (označena so z rdečo na sliki 3.2 desno). Glavna orodja, s katerimi lahko objekte premikamo, rotiramo ali jim spreminjamo velikost, so v oblačku na levi strani okna scene. Ta vsebuje več gumbov, ki določajo izbiro enega od načinov urejanja objektov. Vedno je označen natanko en gumb. Izbiro lahko spreminjamo s klikom na gumbe ali s tipkami `Q`, `W`, `E`, `R`, `T` in `Z`.

Prvo orodje je view tool *(sl. orodje pogleda)*, s katerim z levim klikom sceno premikamo vstran (podobno kot premikanje z uporabo pritiska kolesca miške). Drugo je move tool *(sl. orodje za premikanje)*, ki omogoča premikanje objekta po sceni glede na dane koordinatne osi. Kadar je izbrano to orodje, se v središču izbranega objekta v oknu scene pojavijo barvne puščice, s katerimi objekt lahko premikamo. Poleg puščic so še barvni kvadrati, ki omogočajo premikanje objekta po izbrani ravnini.

Rotate tool *(sl. orodje za rotiranje)* omogoča rotiranje objekta po različnih oseh s pomočjo vrtenja ene izmed obarvanih krožnic, izrisanih v središču izbranega objekta v oknu scene. Z orodjem scale tool *(sl. orodje za spreminjanje velikosti)* lahko spreminjamo velikost objekta v posamezni dimenziji. V središču označenega objekta v oknu

scene se pri izbiri tega orodja pokažejo tri ročice, s katerimi predmet raztegnemo ali skrčimo v izbrani smeri.

Orodje rect tool *(sl. orodje za spreminjanje očrtanega kvadra)* omogoča spreminjanje očrtanega kvadra izbranega objekta, kar hkrati spremeni tudi velikost objekta. Vse opisane funkcionalnosti je mogoče uporabiti hkrati z orodjem transform tool *(sl. orodje za transformacije)*.



Slika 3.2: Levo: Ustvarjanje novega objekta *Kocka*. Desno: Pogled označenega igralnega objekta v oknu scene. Z rdečo je označen seznam s pomožnimi orodji za urejanje.

3.2.2 Kompleksni igralni objekti v hierarhični ureditvi

Pri razvoju igre je pogosto smiselno, da vse igralne objekte, ki skupaj sestavljajo večji objekt, združujemo hierarhično. Pravimo, da so objekti združeni v hierarhično drevo, kjer lahko opazujemo razmerje starš-otrok (*angl. parent-child relationship*). Objekt, ki je višje v hierarhičnem drevesu, predstavlja starševski objekt, medtem ko objekti en nivo nižje predstavljajo njegove otroke.

Najpomembnejša posledica razmerja starš-otrok je dedovanje transformacije. Natančneje, položaj otroka je določen s seštevkom njegove komponente transformacije in komponent transformacije vseh njegovih prednikov. Na primer, če določimo translacijo starša kot $(1, 0, 0)$ in translacijo otroka kot $(0, 0, 1)$, bo dejanska translacija otroka v sceni $(1, 0, 1)$. Dober primer uporabe hierarhične ureditve je modeliranje udov. Če vzamemo enostaven primer dlani, jo lahko definiramo kot starševski objekt, ki ima otroške objekte prstov. To omogoča, da se ob premiku dlani samodejno premaknejo tudi prsti.

Primer 3.2: Hierarhično urejen objekt

Poglejmo si čim bolj enostaven primer z dvema objektoma. Kadar v hierarhiji ustvarimo nov objekt, je ta samodejno na najvišjem nivoju (tj. nima prednikov). V prejšnjem poglavju smo ustvarili objekt *Kocka*, zdaj ustvarimo še enega, ki ga poimenujemo »Druga kocka«. S pomočjo orodij za urejanje objektov jo premaknemo

poleg *Kocke* in jo pomanjšamo na približno polovico začetne velikosti.

Če pogledamo hierarhijo, opazimo, da sta obe kocki trenutno na najvišjem nivoju. Če premaknemo eno (pomagamo si lahko z orodjem move tool), to ne vpliva na drugo. Da med njima ustvarimo razmerje starš-otrok, moramo eno spremeniti v starševski objekt.

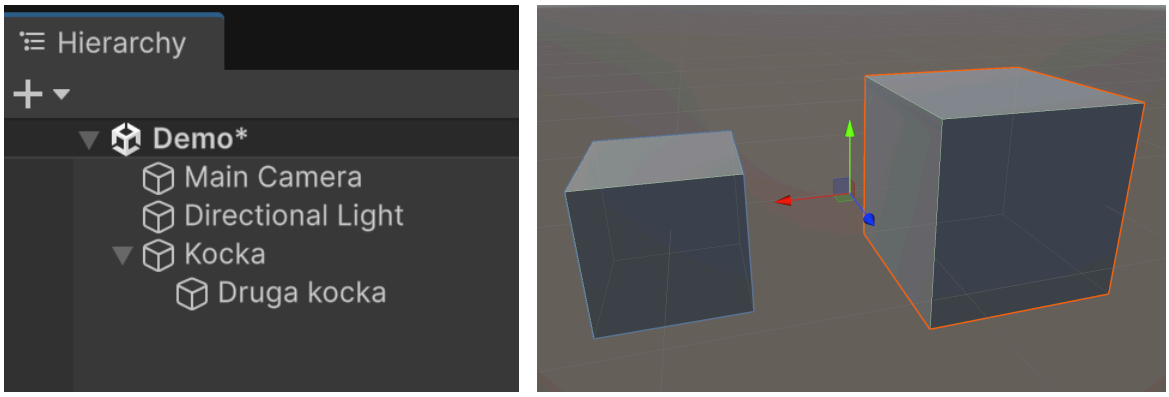
Za starševski objekt izberemo *Kocko*, nato pa z levim klikom držimo vrstico *Druge kocke* in jo povlečemo na vrstico *Kocke*. Ko spustimo klik, se, podobno kot na sliki 3.3, v vrstici *Kocke* na levi pojavi trikotnik, ki nakazuje, da je to sestavljen objekt. Vrstica *Druge kocke* je zdaj zamaknjena v desno. *Druga kocka* je otrok *Kocke*.

Če zdaj kliknemo vrstico *Kocka*, se v oknu scene označita obe kocki, orodje za premikanje pa je prikazano v sredini med kockama in ne več na sredini *Kocke*.

Kaj pa se zgodi, če označimo zgolj vrstico *Druge kocke*? Vidimo, da se označi le njen objekt, v inspektorju pa vidimo, da komponenta transformacije prikazuje relativno pozicijo glede na *Kocko*. V komponenti transformacije lahko s spreminjanjem koordinat preverimo, da gre res za relativne koordinate glede na starševski objekt (tj. *Kocko*). Če v koordinate za pozicijo vpišemo vrednosti $X = 0$, $Y = 0$ in $Z = 0$, se Druga kocka postavi na sredino *Kocke*.

Izbranemu objektu lahko dodajamo potomce tudi tako, da na njegovi vrstici v hierarhiji z desnim klikom v meniju izberemo možnost za ustvarjanje novega objekta. Tako ustvarjen objekt je samodejno ustvarjen na nižjem nivoju kot otrok izbranega objekta.

S tem načinom bomo zdaj ustvarili kroglo z imenom »Krogla«, ki naj bo potomec *Kocke*. Ko jo ustvarimo, jo postavimo na poljubno lokacijo poleg *Kocke*. Zdaj sta *Druga kocka* in *Krogla* na istem nivoju, oba sta otroka Kocke. Ko smo s spremembami zadovoljni, sceno ponovno shranimo.



Slika 3.3: *Kocka* in *Druga kocka* v razmerju starš-otrok.

3.2.3 Podvajanje in brisanje objektov

Pogledali smo že, kako dodajamo nove objekte. Včasih pa bi v sceni želeli podvojiti določen objekt. To lahko storimo na več načinov. Izbrani objekt označimo v sceni in z desnim klikom izberemo možnost ≡ *Duplicate (sl. podvoji)*. Podobno lahko izberemo vrstico objekta v hierarhiji in tam na enak način z desnim klikom podvojimo objekt. V obeh primerih bi namesto možnosti ≡ *podvoji* lahko izbrali možnost ≡ *kopiraj* (angl. copy) in nato ponovili desni klik ter izbrali možnost ≡ *prilepi* (angl. paste). Pozorni moramo biti le na to, da pri podvajanju objekta podvajamo tudi vse njegove komponente. To pomeni, da se ohranijo enake vrednosti za komponento transformacije, zato morda na prvi pogled v oknu scene izgleda, kot da se objekt ni podvojil. Ko nov objekt premaknemo, bo razvidno, da imamo dve instanci enakega objekta.

Objekte lahko brišemo z izbiro objekta in možnostjo ≡ *Delete (sl. izbriši)* ob desnem kliku. Bližnjica za brisanje je tudi tipka `delete` (`CMD+del` na macOS).

3.3 Datotečni sistem projekta in projektno okno

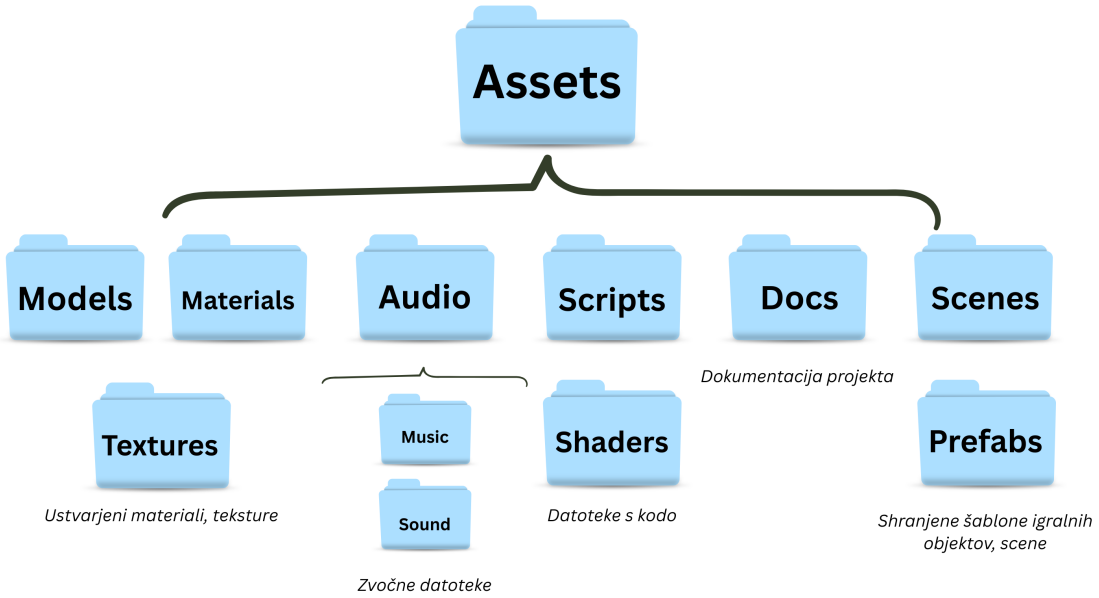
Ker projekti v Unityju običajno vsebujejo veliko količino datotek, je pomembna dobra organizacija. Osrednje orodje za nadzor nad datotekami je projektno okno, ki odraža dejansko stanje map na disku.

V tem poglavju bomo na kratko povzeli uradna priporočila za organizacijo datotek v Unity projektih[1]. Dobro strukturiran projekt ne olajša le dela posamezniku, temveč je nujen za vzdrževanje in sodelovanje pri kompleksnejših projektih.

V projektnem oknu na najvišjem nivoju najdemo mapo `Assets`. To je krovna mapa za vse komponente, objekte, scene in skripte. Zlato pravilo Unityja je, da vse nove elemente shranjujemo izključno v podmape znotraj mape `Assets`. Poleg nje je še mapa `Packages`, ki vsebuje privzete sistemske pakete in običajno ne zahteva ročnega posega-nja razvijalca.

Mapa `Assets` je nadalje razdeljena na podmape, ki vsebine ločujejo glede na njihov tip. Datoteke, namenjene grafiki in upodabljanju objektov, kot so modeli, materiali in teksture, običajno shranjujemo v mape `Models`, `Materials` in `Textures`. Zvočne datoteke uvrščamo v mapo `Audio`, datoteke, povezane s programiranjem, pa razvrščamo v mape, kot sta `Scripts` in `Shaders`. Scene hranimo v mapi `Scenes`, šablone objektov (prefabi) pa v mapi `Prefabs`. V kompleksnejših projektih, pri katerih sodeluje več razvijalcev, postane struktura map še pomembnejša. Poleg osnovnih vsebin pogosto vključuje tudi dokumentacijo projekta, ki določa smernice za programiranje, opisuje organizacijo datotečnega sistema ter lahko vsebuje dodatna gradiva, na primer materiale za predstavitev ali marketing. Takšna organizacija omogoča boljše sodelovanje med člani ekipe, večjo preglednost nad projektom in lažje vzdrževanje skozi čas. Slika 3.4 prikazuje primer dobro organiziranega projekta.

[1] `https://unity.com/how-to/organizing-your-project`



Slika 3.4: Priporočena struktura datotečnega sistema projekta znotraj mape `Assets`, kjer so datoteke organizirane v podmape glede na njihov tip.

Primer 3.3: Organizacija datotek v datotečnem sistemu projekta

V projektu mapa `Assets` že vsebuje mape `_Unity Essentials`, `Settings` in `TextMesh Pro`, ki so del uporabljene predloge. Naše vsebine bomo shranjevali ločeno od teh map, da ohranimo čisto strukturo. Mapa `_Unity Essentials` vsebuje večino map, ki jih prepoznamo iz slike 3.4.

V poglavju 3.2 smo shranili svojo sceno, a pri tem nismo namenili posebne pozornosti temu, kam natanko se je shranila. Premakniti jo moramo v mapo `Scenes` v mapi `Assets`. Morda smo že opazili, da zaenkrat v mapi `Assets` še nimamo mape `Scenes`, zato jo moramo najprej ustvariti. To storimo tako, da v projektnem oknu izberemo mapo `Assets` in v desnem razdelku (tistem, ki prikazuje vsebino izbrane mape) na praznem polju z desnim klikom izberemo možnost ≡ *Create* ▶ *Folder* (oziroma enako možnost z uporabo zgornje orodne vrstice urejevalnika pod zavihkom ≡ *Assets*).

V projektnem oknu se pojavi mapa, ki ji takoj spremenimo ime v `Scenes`. Sedaj moramo poiskati še mesto, kamor smo shranili svojo sceno *Demo*. Če smo jo shranili neposredno v mapo `Assets`, jo enostavno prestavimo v mapo `Scenes` z uporabo *t. i.* klika in potega, sicer pa lahko uporabimo iskalno vrstico projektnega okna.

Ker poznamo ime svoje scene, ga lahko vnesemo v iskalno vrstico, desni raz-

delek projektnega okna pa bo prikazal vse datoteke, ki ustrezajo iskanju. Sceno lahko s klikom označimo in jo izrežemo[a], nato pa pogled v desnem razdelku okna prestavimo na mapo `Scenes` in izrezek prilepimo.

[a] Datoteko ali objekt lahko izrežemo in pozneje prilepimo na izbrano mesto s kombinacijo bližnjic na tipkovnici `CTRL+X` in `CTRL+V`.

3.4 Okno igre in igralni način

Spoznali smo že nekaj osnovnih igralnih objektov in kako lahko z njimi napolnimo sceno. Vendar smo v oknu scene do sedaj urejali zgolj statično postavitev objektov, kar pa ni edino, kar Unity ponuja. Pri igrah je morda celo pomembneje, kako se objekti v sceni premikajo, kot pa, kako so postavljeni na začetku. Pri razvoju je pomembno, da lahko igro in njen potek simuliramo ter s tem prilagajamo lastnosti posameznih objektov.

Unity urejevalnik to omogoča v oknu igre z vklopom igralnega načina (*angl. game mode*). Slednjega bomo podrobneje spoznali v tem poglavju s pomočjo ene izmed scen iz predloge projekta.

Primer 3.4: Odpiranje shranjene scene

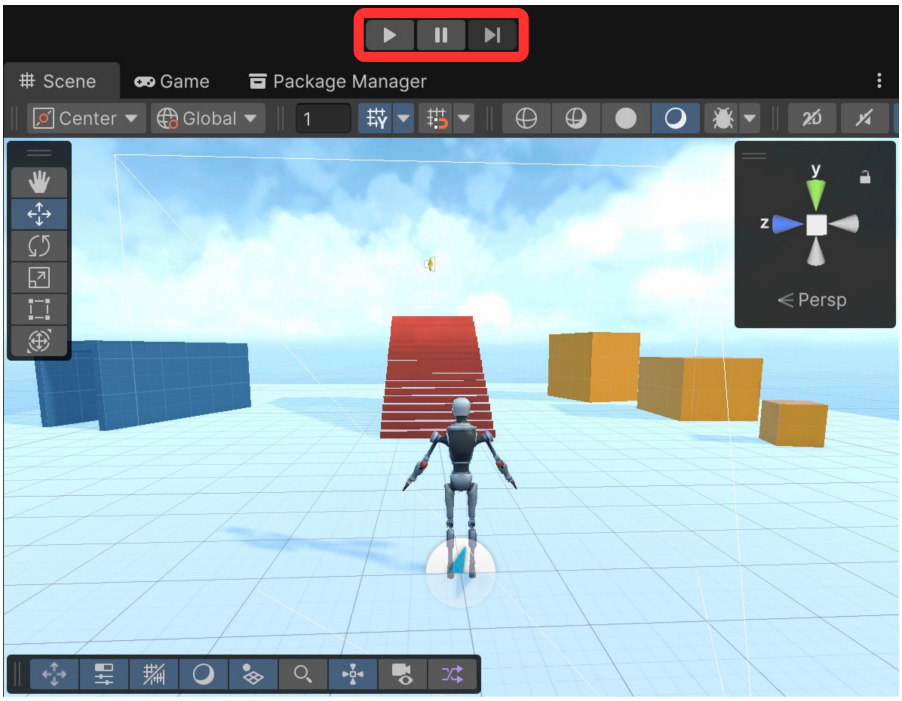
V nadaljevanju bomo potrebovali sceno `1_Starter_Scene`, ki jo najdemo med scenami iz predloge. Najprej jo poiščemo v projektnem oknu na lokaciji `_Unity Essentials/Scenes`.

Nanjo dvokliknemo in v oknu scene vidimo, da se je vsebina zamenjala. Tudi hierarhija zdaj prikazuje drug nabor objektov. Nekateri med njimi so hierarhično urejeni, drugi pa enostavni. Osrednji objekt je *Igralec (angl. Player)*, ki predstavlja igralca, ko je igra pognana.

3.4.1 Vklop igralnega načina

Igralni način je pogled, v katerem urejevalnik v oknu igre prikazuje pognano igro. Omogoča spremljanje premikov objektov in njihovih interakcij, poleg tega pa lahko med igralnim pogledom spreminjamo parametre in lastnosti objektov.

Na vrhu urejevalnika (glej sliko 3.5) so gumbi Play *(sl. Začni)*, Pause *(sl. Prekini)* in Step *(sl. Naslednji korak)*. S pritiskom na gumb Play sprožimo igralni način, pri čemer urejevalnik samodejno iz okna scene preklopi v okno igre. Iz igralnega načina izstopimo z gumbom Stop na vrhu urejevalnika (na mestu, kjer je bil prej gumb Play). Urejevalnik takrat samodejno vrne v ospredje okno scene.



Slika 3.5: Z rdečim kvadratom so označeni gumbi za vklapljanje in izklapljanje igralnega načina. Gumbi so (od leve proti desni): Play *(sl. Začni)*, Pause *(sl. Prekini)* in Step *(sl. Naslednji korak)*. Kadar je igralni način vklopljen, se gumb Play nadomesti z gumbom Stop.

Primer 3.5: Vklop igralnega načina in sprehajanje igralca po sceni

Igralni način vklopimo z gumbom Play na vrhu urejevalnika. Takrat se v ospredje postavi okno igre in v sredini vidimo *igralca*.

Ker je igra v teku, lahko *igralca* premikamo. Z miško kliknemo znotraj okna igre, nato pa ga premikamo s puščicami ↑, ↓, ←, → ali s tipkami `W`, `A`, `S`, `D`. Tako *igralec* hodi po prostoru, če pa držimo še tipko `SHIFT`, lahko tudi teče. S tipko `SPACE` *igralec* skače, smer njegovega pogleda uravnavamo z miško. Pogled v oknu igre sledi *igralcu*, medtem ko ga premikamo.

Med igralnim načinom lahko opazujemo, kako *igralec* reagira, ko se dotika določenih igralnih objektov. Po stopnicah se lahko vzpenja, v škatle se zaleti in ne more skoznje, na podlagi stoji, če pa se premakne s podlage, pada v globino. Če miške na zaslonu ne vidimo, se ta s klikom na tipko `ESC` ponovno prikaže.

Dokler je igralni način aktiven, lahko sami preklopimo iz okna igre v okno scene, kjer na običajen način opazujemo igralne objekte, jih premikamo in spreminjamo. Pomembno se je zavedati, da se spremembe, narejene v igralnem načinu, **ne shranijo**. Če torej pri aktivnem igralnem načinu spremenimo lokacijo enega izmed objektov, se ta sprememba ne bo ohranila, ko iz igralnega načina izstopimo. To vedenje omogoča, da razvijalci znotraj igralnega načina preizkušajo različne parametre, ki se ne shranjujejo. Če želimo spremembe shraniti, jih moramo nastaviti, ko igralni način ni aktiven.

Pozor! Pogosto se zgodi, da pozabimo, da je igralni način aktiven. Zato se lahko zmedemo in ne najdemo razloga, zakaj se spremembe ne shranjujejo, kot bi želeli.

Urejevalnik Unity zato pri igralnem pogledu nekoliko spremeni barvo vseh oken (ta sprememba je s prednastavljenimi nastavitvami lahko le malo opazna). Po želji lahko v nastavitvah urejevalnika v zavihku ≡ *Colors* nastavimo **Playmode tint**, ki določa, kako bodo obarvana okna urejevalnika, ko bo igralni način aktiven.

Izziv 1: Osvoji zvezdo

V sceni `1_Starter_Scene` se pri vrhu stopnic v zraku nahaja *zvezda (angl. star)*. V igralnem načinu poskusite osvojiti zvezdo. Ker je izhodiščno mesto zvezde visoko, poskusite v igralnem načinu popraviti njeno pozicijo v prostoru (s spreminjanjem vrednosti v komponenti transformacije).

Kaj je potrebno storiti, da je zvezda na želenem mestu že takoj, ko zaženemo igralni pogled?

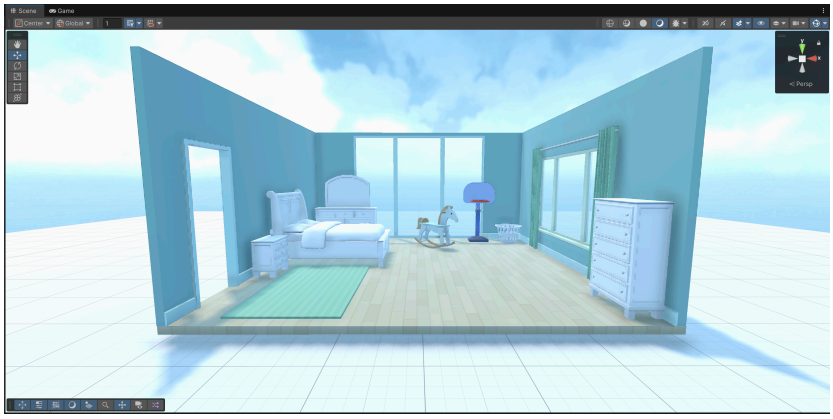
POGLAVJE 4

Ustvarjanje v sceni

Unity omogoča snovanje dinamičnih in interaktivnih scen, v katerih objekti niso le statični elementi, temveč se aktivno odzivajo na okolico. Sistem omogoča simulacijo realističnih fizikalnih sil, določanje specifičnih lastnosti materialov (kot je npr. prožnost) ter napredne nastavitve osvetlitve in prostorskega zvoka. V kombinaciji s programsko logiko te možnosti odpirajo vrata k implementaciji še tako drznih in kreativnih scenarijev. V tem poglavju se bomo osredotočili na uporabo različnih komponent, ki objektom določajo različne lastnosti.

4.1 Ustvarjanje scene z vnaprej pripravljenimi elementi

Pri snovanju scen v Unityju se praviloma izogibamo zamudnemu modeliranju kompleksnih objektov neposredno v urejevalniku. Unity namreč ni primarno orodje za 3D modeliranje; temu so namenjeni specializirani programi, iz katerih nato 3D modele uvozimo v projekt. Številni kompleksni objekti so že vnaprej pripravljeni in dostopni na različnih spletnih platformah, od koder jih lahko prenesemo. Postopek uvažanja zunanjih elementov je podrobneje opisan v poglavju 5, v tem razdelku pa se bomo osredotočili na uporabo objektov, ki so že vključeni v izbrano predlogo projekta.



Slika 4.1: Primer urejene otroške sobe z uporabo elementov iz predloge projekta.

Primer 4.1: Ustvarjanje otroške sobe

V datotečnem sistemu poiščemo sceno `2_KidsRoom_3D_Scene` (za pomoč, kako odpreti vnaprej pripravljeno sceno, glej primer 3.4).

Ko se scena odpre, v oknu scene opazimo osrednjo platformo, ki služi kot podlaga. Drugih igralnih objektov zaenkrat ni in jih moramo dodati sami. Naša naloga je, da s pomočjo vnašanja vnaprej pripravljenih objektov iz datotek predloge projekta ustvarimo otroško sobo.

V projektnem oknu odpremo mapo `_Unity Essentials/Prefabs`, kjer se nahajajo vsi potrebni objekti. Iz mape `Rooms` v sceno dodamo objekt `01_Bedroom`, in sicer tako, da v projektnem oknu nanj kliknemo in ga povlečemo v okno scene.

Pozor! Pri dodajanju objektov se izogibajte dvojnemu kliku na datoteko v projektnem oknu. Dvojni klik odpre *t. i.* »Prefab Mode«, ki služi urejanju same šablone in ne vaše trenutne scene.

Za natančno poravnavo nastavimo vse tri koordinate *spalnice (angl. 01_Bedroom)* v komponenti transformacije na vrednost `0`. Zdaj lahko prostor napolnimo s pohištvom, ki ga najdemo v datotečnem sistemu v mapi `_Unity Essentials/Prefabs/Bedroom`.

Elemente primerno razporedimo po prostoru s pomočjo orodij za premikanje in urejanje igralnih objektov. Primer napolnjene sobe je na sliki 4.1.

Končni izgled opremljene sobe, kot ga prikazuje slika 4.1, dosežemo s kombiniranjem različnih elementov (postelja, miza, omare). Ko smo z razporeditvijo zadovoljni, spremembe shranimo z ukazom ≡ *File* ▶ *Save*.

4.2 Komponente igralnega objekta

Komponente so gradniki, ki določajo lastnosti in obnašanje igralnega objekta. Določajo vizualni izgled (model in materiale), fizikalne lastnosti (kako objekt reagira na trke), zvočne učinke, svetlobne značilnosti in programsko logiko (skripte).

Vse komponente izbranega objekta so vidne v oknu inspektor. Vsak igralni objekt ima privzeto vključeno komponento transformacije, ki določa njegov položaj, orientacijo in velikost v prostoru. Število dodatnih komponent, ki jih lahko pripnemo objektu, je praktično neomejeno.

Okno inspektor je sestavljeno iz več delov. Na vrhu je besedilno polje, v katerem je zapisano ime objekta. Če to ime spremenimo, se spremeni tudi poimenovanje objekta v hierarhiji. Na levi strani je majhno potrditveno polje. Če je označeno s kljukico, pravimo, da je objekt omogočen. Če polje odznačimo, objekta v sceni in v igralnem načinu ne vidimo, vrstica v hierarhiji pa ostane, a se ime objekta obarva sivo. Pravimo, da smo objekt onemogočili. Pod vrstico z imenom je možnost izbire **značke** (*angl. tag*) in **nivoja**

(*angl. layer*) objekta. Če smo objekt ustvarili iz vnaprej pripravljenega objekta (oz. šablone), so tu še informacije o izvornem objektu.

Spodnji del okna zavzema seznam vseh pripetih komponent. Vsaka komponenta ima ob imenu majhen trikotnik, ki omogoča razširitev ali strnitev njenih nastavitev, kar pripomore k boljši preglednosti pri kompleksnejših objektih.

Slika 4.2: Ko zaženemo igralni način, se ne zgodi nič – žoga nepremično lebdi v zraku.

Primer 4.2: Žoga, ki obstane v zraku

V nadaljevanju želimo v sceni ustvariti nov objekt. Ustvarimo kroglo z imenom »Zoga«[a], določimo ji koordinate $X = 0$, $Y = 3$ in $Z = 0$ (*žoga* naj bo nad tlemi sobe, nekoliko v zraku).

Če zdaj zaženemo igralni način, opazimo, da *žoga* nepremično obstane v zraku (glej sliko 4.2). Unity samodejno ne ve, da bi nanjo morala delovati gravitacija. To lastnost ji moramo dodati s posebno komponento, kar si bomo ogledali v naslednjem poglavju.

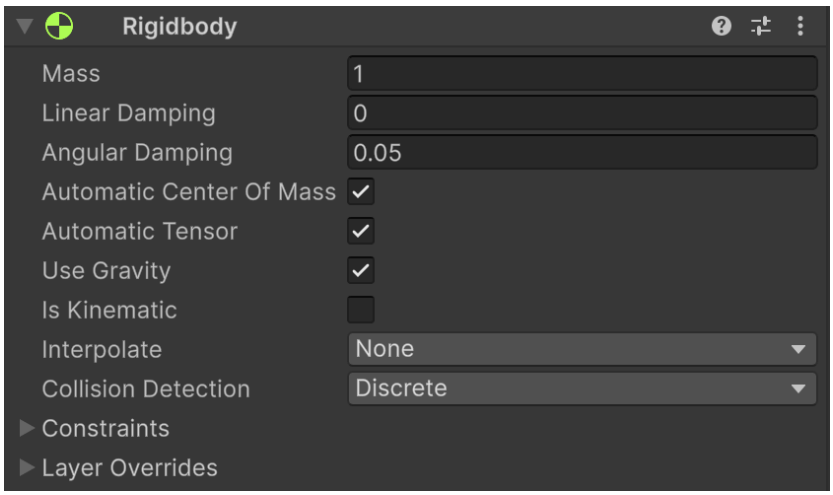
[a] Uporabi slovenskih šumnikov v imenih objektov in datotek se v Unityju zaradi kompatibilnosti izogibamo.

4.2.1 Komponenta Rigidbody – vpliv fizikalnih sil na telo

V primeru 4.2 smo ustvarili žogo in jo postavili v zrak. Pričakovali bi, da bo ob zagonu igralnega načina žoga padla na tla, vendar je le nepremično lebdela v zraku.

Unity vsebuje svoj fizikalni pogon, ki omogoča simulacijo različnih fizikalnih sil, ki delujejo na objekte. Simulira sile, kot so gravitacija, upor, trenje in trki teles. Ker želimo biti pri ustvarjanju interaktivnih izkušenj selektivni pri izbiri objektov, na katere fizikalne

sile delujejo (npr. ne želimo si vpliva gravitacije na objekt tal ali sonca), moramo objekte, na katere naj sile delujejo, ročno izbrati. Dojemljivost za fizikalne sile zato kot lastnost dodamo izbranim objektom v obliki komponente.

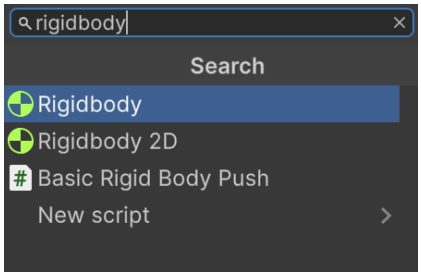


Slika 4.3: Komponenta Rigidbody v pogledu v oknu inspector.

Za simulacijo sil na objektih je odgovorna komponenta Rigidbody. V inspektorju (glej sliko 4.3) lahko vidimo, da vsebuje nekaj nastavljivih parametrov, ki različno vplivajo na obnašanje telesa v prostoru. Parameter **Mass** *(sl. masa)* označuje maso objekta v kilogramih. Parametra **Linear Damping** *(sl. linearen upor)* in **Angular Damping** *(sl. rotacijski upor)* določata koeficienta upora.

Sledijo potrditvena polja: **Automatic Center of Mass** *(sl. avtomatsko računanje centra gravitacije)*, ki določa, ali pogon samodejno izračuna, kje je center gravitacije danega objekta, in **Automatic Tensor** *(sl. avtomatski tenzor)*, ki določa, ali se tenzor vztrajnosti objekta računa avtomatsko. Če je označeno polje **Use Gravity** *(sl. uporabi gravitacijo)*, pogon na objektu simulira delovanje gravitacijske sile. Če je označeno polje **Is Kinematic** *(sl. kinematičnost)*, sile, povzročene s trki ali drugimi interakcijami z ostalimi telesi, ne vplivajo na dani objekt.

Polji **Interpolate** *(sl. interpolacija)* in **Collision Detection** *(sl. detekcija trkov)* uravnavata natančnost izračunov interakcij med objekti. Ker računalnik upodablja sceno v določenem številu sličic na sekundo (npr. 60 FPS), se lahko zgodi, da do trka pride med sličicama. Da lahko zaznavamo tudi takšne trke, je potrebno računanje dogodkov, ki se dogajajo med sličicami. Omenjena parametra določata način tega računanja.



Slika 4.4: Postopek dodajanja komponente Rigidbody objektu z iskalnim poljem.

Primer 4.3: Žogi dodajmo vpliv gravitacije

Nadaljujemo prejšnji primer 4.2. Da bi *žogi* dodali vpliv gravitacije, ji moramo dodati komponento Rigidbody.

Najprej *žogo* označimo, nato pa v oknu inspektor na dnu kliknemo gumb Add Component *(sl. Dodaj komponento)*. Odpre se spustni seznam z iskalnikom. V iskalno vrstico vpišemo *Rigidbody* in izberemo edino vrstico, ki je rezultat iskanja (glej sliko 4.4). Opazimo, da se je v oknu inspektor na dno seznama komponent dodal nov razdelek z imenom Rigidbody. Če je komponenta skrčena, jo s klikom na trikotnik ob imenu razpremo.

Za začetek ohranimo privzete vrednosti vseh parametrov in zaženemo igralni način. *Žoga* takoj po zagonu pade na tla *spalnice*. Vidimo torej, da se simulacije fizikalnih sil izvajajo le v igralnem načinu.

Sedaj shranimo trenutno sceno, v nadaljevanju pa bomo spoznali, zakaj se je žoga ob stiku s tlemi zaustavila in ni padla skoznje.

4.2.2 Komponenta Collider – računska meja telesa

V naravi ima vsako telo svojo fizikalno mejo, ki običajno sovpada z njegovo zunanjo površino. Pravimo, da se dve telesi dotikata, ko se stakneta njuni meji. V digitalnem svetu Unityja pa vidna oblika objekta (tisto, kar vidimo izrisano) in njegova fizikalna meja ne nujno sovpadata.

Za simulacijo trkov in interakcij Unity uporablja komponento Collider. Ta komponenta definira *t. i.* **računsko mejo** objekta, ki fizikalnemu pogonu sporoča, kje in kako se telo nahaja v prostoru. Brez te komponente bi objekte v igri lahko videli, a bi se ti lahko prosto premikali drug skozi drugega, saj fizikalni pogon ne bi imel podatka o tem, kje naj bi se trk zgodil.

Unity ponuja različne vrste colliderjev, prilagojene geometriji objektov:

- Box Collider: Najenostavnejša oblika, primerna za škatle, stene in kvadre.
- Sphere Collider: Uporablja se za krogle in zaobljene predmete.
- Capsule Collider: Ustreza obliki kapsule.
- Mesh Collider: Najnatančnejša oblika, ki se popolnoma prilagaja kompleksni geometriji objekta (takšna geometrija je običajno definirana z mrežo (*angl. mesh*)[1], od koder tudi poimenovanje).

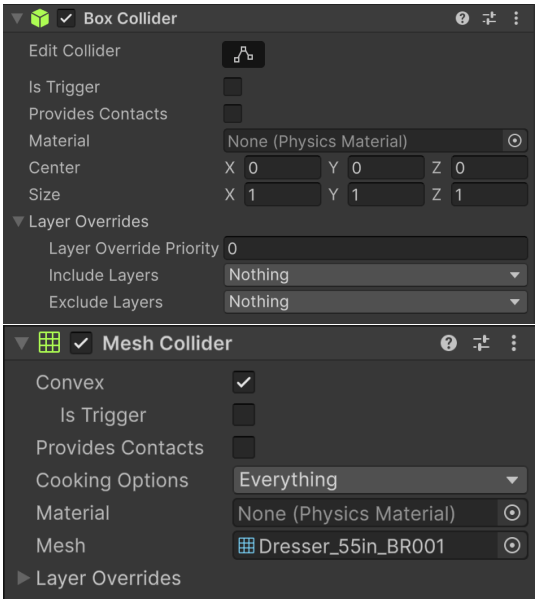
[1] Mreža je način predstavitve 3D modelov, kjer so ti sestavljeni iz oglišč, povezanih v mnogokotnike (navadno trikotnike). V pogonu Unity so mreže glavni način predstavitve 3D modelov.

Zakaj ne bi vedno uporabljali vrste Mesh Collider?

Sicer je Mesh Collider najnatančnejši in najbolje opiše geometrijo objekta, a je hkrati računsko zelo zahteven, saj mora fizikalni pogon pri računanju trkov med objekti preveriti vsak trikotnik mrežne strukture posebej. To lahko upočasni igro, pri ostalih colliderjih pa računanje trkov zahteva veliko manj operacij (npr. pri izračunu trka med dvema kroglama je potrebno izračunati le razdaljo med središčema).

Kadar v interaktivnih izkušnjah razlika med uporabo Mesh Collider-ja ali katerega od enostavnejših ni bistvena, v praksi kompleksnejše objekte aproksimiramo z uporabo preprostejših colliderjev. Na primer, za človeško figuro pogosto uporabimo Capsule Collider, za različne kose pohištva pa Box Collider.

Collider ima v inspektorju polja, podobna tistim na sliki 4.5. Enostavnejše colliderje lahko urejamo s pomočjo gumba Edit Collider *(sl. uredi collider)*, medtem ko to za Mesh Collider ni mogoče. Pri slednjem lahko zahtevamo, da je računska ovojnica konveksna, kar jo naredi enostavnejšo in s tem prispeva k hitrejšemu delovanju igre. Poleg imena komponente najdemo potrditveno polje, ki komponento omogoči oziroma onemogoči[2].



Slika 4.5: Komponenta Box Collider (zgoraj) določa računsko mejo objekta v obliki kvadra. Komponenta Mesh Collider (spodaj) določa računsko mejo objekta, ki je definirana s poligonsko mrežo *(angl. mesh)*.

Potrditveno polje **Is Trigger** *(sl. sprožilec)* se nanaša na posebno vrsto interakcij predmetov; če je ta možnost označena, telo ob trku ne deluje več s silo nazaj na drugo telo, ampak sproži dogodek v ozadju. To je koristno v primerih, kot je bil v prejšnjem poglavju v sceni `1_Starter_Scene`. Da smo *zvezdo* lahko pobrali, je pogon moral zaznati trk. Ta trk pa ni sprožil delovanja sil nazaj na *igralca*, ampak se je v tistem trenutku sprožil zvočni učinek, *zvezda* pa je izginila. Drug primer takšne uporabe je lahko kuharska igra, kjer moramo v lonec dodati vse potrebne sestavine. Na vrh lonca lahko nastavimo prazen objekt, ki ima komponento collider nastavljeno na sprožilec. Z njim lahko zaznavamo, kateri objekti so padli v lonec.

Potrditveno polje **Provides Contact** *(sl. podatki o stikih)* določa, ali pogon računa in hrani natančne podatke o stikih med dvema objektoma. Ti podatki so lahko uporabni za naprednejšo logiko v igri. Tukaj lahko določamo tudi poljuben center objekta (ki v

[2]Ko komponento onemogočimo, je to enako, kot če bi jo izbrisali. Ta način je v določenih primerih zaželen, da ne izgubimo prejšnjih nastavitev, s katerimi smo delali.

kombinaciji z Rigidbody določa gravitacijski center) in velikost računske meje glede na dimenzijo. Pred tem pa imamo še polje **Material**, o katerem bomo več izvedeli v naslednjem poglavju.

Slika 4.6: Če enemu izmed objektov, ki bosta trčila, collider nastavimo tako, da je manjši od vizualne meje (levo), se bo drugo telo pomaknilo do izbranega colliderja. Vizualno to izgleda, kot da je drugi objekt »padel« v prvega (desno).

Primer 4.4: Kako se obnašajo objekti s Colliderjem?

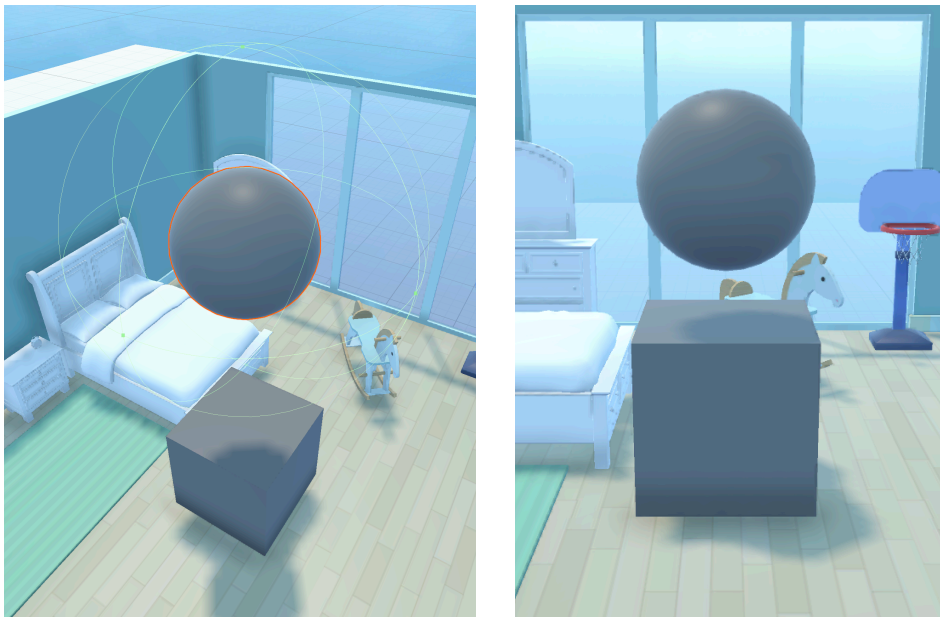
Za demonstracijo najprej ustvarimo nov objekt: kocko z imenom »Kocka«, ki jo postavimo natanko pod *žogo* iz prejšnjega primera (koordinate nastavimo na $X = 0$, $Y = 0{,}7$ in $Z = 0$). Nato zaženemo igralni način in opazimo, da *žoga* pade na *kocko* in se na njej zaustavi.

Zakaj se je *žoga* zaustavila ravno tam? Če odpremo inspektor *kocke*, v njem najdemo komponento Box Collider. Ker tudi *žoga* vsebuje collider (natančneje Sphere Collider), sta znani računski meji obeh objektov, zato je fizikalni pogon lahko izračunal, da je prišlo do trka. Posledično se je *žoga* zaustavila na *kocki*.

Poglejmo, kaj se zgodi, če eden izmed objektov nima komponente collider. Najenostavneje to storimo tako, da odznačimo potrditveno polje poleg imena colliderja (to storimo na *kocki*). Sedaj *žoga* pade skozi *kocko*, kot da je sploh ni. Vidimo torej, da morata imeti oba objekta collider, če želimo, da fizikalni pogon zazna trk. Pred nadaljevanjem ne pozabimo izklopiti igralnega načina.

Obliko colliderja lahko spreminjamo z gumbom Edit Collider. Na objektu v oknu scene se prikaže zelena obroba, ki jo lahko poljubno premikamo. Ta zelena obroba predstavlja računsko mejo. Poglejmo, kaj se zgodi, če jo pomanjšamo (torej je meja znotraj vizualne meje objekta). Ko zaženemo igralni način, se *žoga* zaustavi »v« kocki (kot je prikazano na sliki 4.6). Fizikalni pogon je upošteval nastavitev colliderjev obeh objektov in trk izračunal v notranjosti kocke.

Če spremenimo računsko mejo *žoge* tako, da je ta veliko večja od oblike *žoge*, se bo *žoga* ustavila nekoliko nad *kocko*. Rezultat je podoben tistemu na sliki 4.7.



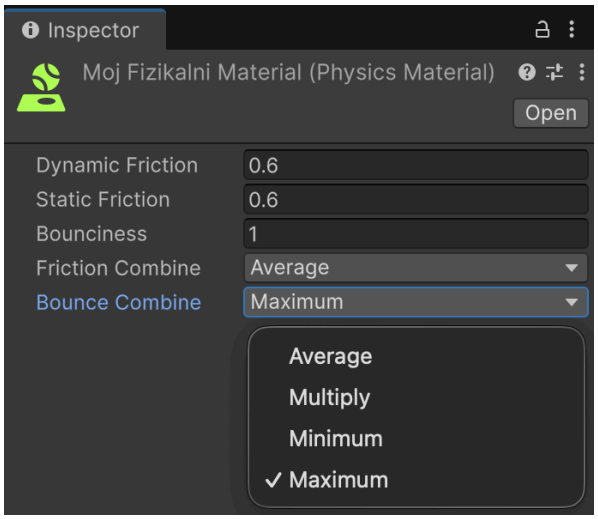
Slika 4.7: Če enemu izmed objektov, ki bosta trčila, collider povečamo tako, da je večji od vizualne meje (levo), se trk izračuna tako, da med objektoma navidezno ostane nekaj prostora. Če na ta način nastavimo večji collider *žogi*, ki pada na *kocko*, se bo zaustavila nekoliko nad *kocko* (desno).

4.2.3 Fizikalni material - določanje odbojnosti predmetov

Predmeti se ob trkih v naravi obnašajo različno: gumijasta žogica se od tal silovito odbije, medtem ko se težka lesena kocka le ustavi. V Unityju lahko odbojnost materialov določamo s pomočjo fizikalnih materialov (*angl. physics material*).

Fizikalni material je posebna vrsta materiala v Unityju, ki ga dodamo colliderju objekta. Določa fizikalne lastnosti površine objekta, kot sta odbojnost in trenje. Določimo mu lahko vrednosti **Dynamic Friction** *(sl. koeficient trenja)*, **Static Friction** *(sl. koeficient lepenja)* in **Bounciness** *(sl. koeficient prožnosti)*, ki vsi sprejemajo le številske vrednosti med 0 in 1. Večja kot je vrednost, večje je trenje, lepenje ali odbojnost.

Kot je razvidno iz slike 4.8, sta med parametri materiala **Friction Combine** *(sl. metoda izračuna skupnega trenja)* in **Bounce Combine** *(sl. metoda izračuna skupne odbojnosti)*. Končno obnašanje objekta v sceni je odvisno ne le od lastnega colliderja in določenega fizikalnega materiala, temveč tudi od fizikalnih lastnosti objekta, s katerim objekt interagira. Ta dva parametra določata, kakšno naj bo obnašanje pri stiku med objektoma. Vrednosti so iz končnega nabora možnosti: **Average** *(sl. povprečje)*, **Minimum** *(sl. minimum)*, **Maximum** *(sl. maksimum)* in **Multiply** *(sl. zmnožek)*.



Slika 4.8: Fizikalni material je posebna vrsta materiala, ki ga dodamo colliderju objekta. Določa fizikalne lastnosti objekta (koeficiente trenja, lepenja in prožnosti).

Primer 4.5: Žoga, ki se odbija od tal

V tem primeru nadaljujemo iz stanja v primeru 4.3 (po potrebi iz scene izbrišemo *kocko* iz primera 4.4 in *žogi* ponovno nastavimo collider tako, da se prilega njeni vizualni obliki). Da bi *žogi* dodali fizikalni material, moramo najprej ustvariti novega.

V projektnem oknu odpremo mapo `Assets` in najprej ustvarimo svojo mapo z imenom `Materials` (kot je priporočljivo glede na smernice iz poglavja 3.3). Nato odpremo to novo mapo in na praznem območju z desnim klikom ustvarimo fizikalni material ter ga poimenujemo »MojFizikalniMaterial« (glej sliko 4.9). S klikom na datoteko materiala se v oknu inspektor odprejo njegove lastnosti. Vrednost **Bounciness** nastavimo na 1, nato pa v hierarhiji poiščemo vrstico *žoge* in jo označimo.

Da bi ustvarjeni fizikalni material dodali *žogi*, njegovo datoteko iz projektnega okna povlečemo na polje **Material** v komponenti Sphere Collider. Enako lahko dosežemo, če v komponenti pri polju **Material** izberemo gumb v obliki pike na desni strani, ki odpre pojavno okno, v katerem nato izberemo med razpoložljivimi fizikalnimi materiali.

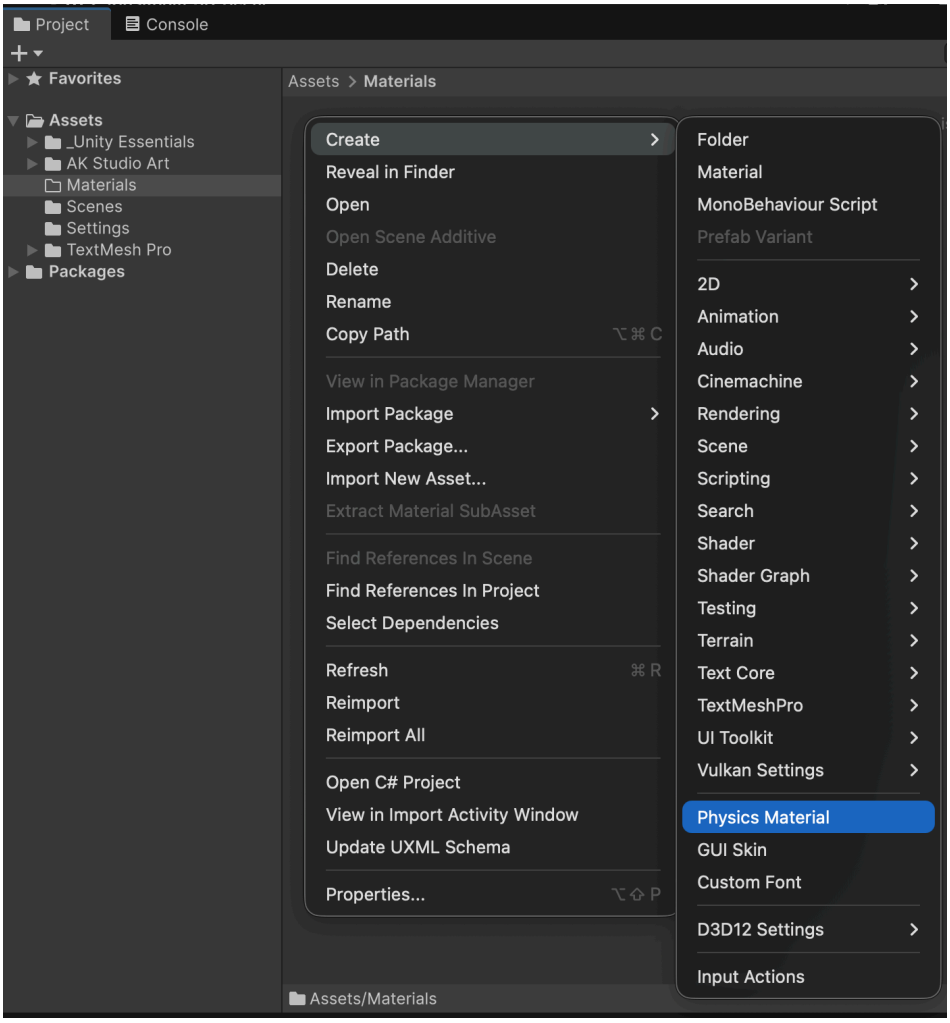
Če zdaj sprožimo igralni način, se *žoga* nekajkrat odbije, preden obstane na tleh. Toda to ni povsem pričakovano, saj smo **Bounciness** fizikalnega materiala nastavili na najvišjo možno vrednost in bi želeli, da se žoga odbija konsistentno brez izgube višine (torej da se pri vsakem odboju odbije nazaj na začetno višino).

To se zgodi zaradi privzete nastavitve **Bounce Combine**, ki ima vrednost **Average**. Ker je *žoga* ob padcu trčila s tlemi, ki nimajo dodanega fizikalnega materiala, se njihova prožnost obravnava kot vrednost 0. Končen odboj *žoge* je odvisen od povprečja teh dveh vrednosti, torej je rezultat $0{,}5$. Odboj žoge lahko povečamo

tako, da v ustvarjenem fizikalnem materialu polje **Bounce Combine** nastavimo na **Maximum**. Sedaj se žoga v igralnem načinu po vsakem odboju vrača na začetno višino.

Zanimivost – numerična napaka v simulaciji

Če simulacijo pustimo delovati dlje časa, opazimo, da žoga začne skakati vedno višje. To je posledica majhnih napak pri zaokroževanju plavajoče vejice v fizikalnem pogonu. Te drobne numerične napake se z vsakim izračunom seštevajo in postajajo vedno bolj opazne.



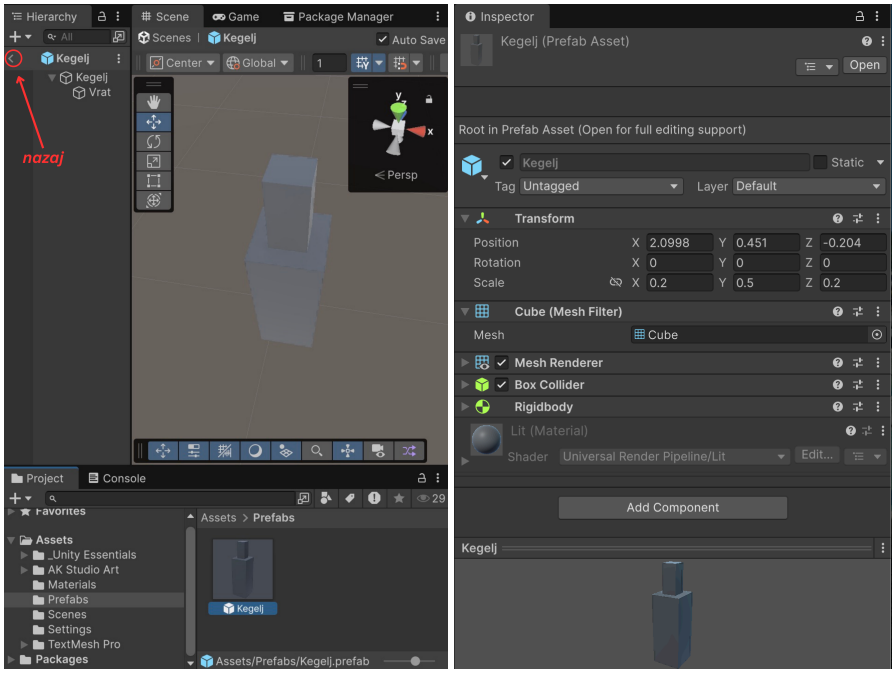
Slika 4.9: Fizikalni material ustvarimo tako, da z desnim klikom v spustnem seznamu izberemo možnost ≡ *Create* ▶ *Physics Material*. To na izbranem mestu ustvari novo datoteko fizikalnega materiala, ki jo lahko poljubno poimenujemo.

4.3 Prefab - šablona igralnega objekta

Pri ustvarjanju kompleksnejših scen z velikim številom objektov se hitro soočimo s težavo usklajevanja. Če imamo v sceni gozd, morajo biti drevesa med seboj podobna po proporcih, fizikalnih lastnostih in vizualnem izgledu. Ročno ustvarjanje in popravljanje stotin dreves posamezno bi bilo izjemno zamudno. Delno lahko to rešimo s kopiranjem objektov, vendar se težava ponovi, ko želimo na vseh kopijah hkrati spremeniti določeno lastnost (npr. barvo listja ali težo debla).

Zato Unity ponuja možnost uporabe Prefabov. To so **šablone** objektov, ki v datoteki na disku določajo vse lastnosti določenega tipa objekta. Takšne vnaprej pripravljene objekte lahko kot instance poljubno dodajamo v različne scene, njihova glavna prednost pa se pokaže pri naknadnem urejanju. Ko želimo posodobiti vse kopije hkrati, spremembo izvedemo le na izvorni datoteki prefaba, ta pa se nato samodejno in takoj uveljavi na vseh pripadajočih instancah v celotnem projektu. Kljub tej povezanosti Unity omogoča tudi specifične prilagoditve na posameznih instancah, ki ostanejo lokalne in ne vplivajo na videz ali obnašanje ostalih enakih objektov v sceni.

Ustvarjanje prefaba je preprosto: igralni objekt, ki smo ga pripravili v sceni, z levim klikom povlečemo iz hierarhije v projektno okno. Ta postopek ustvari novo datoteko s končnico `.prefab`, ki hrani »šablono«, ki smo jo določili. Če na datoteko prefaba v projektnem oknu dvokliknemo, vstopimo v *t. i.* **prefab pogled** (*angl. prefab mode*). V tem načinu preostala scena izgine, v oknu scene pa ostane le izbrani objekt, ki ga lahko podrobno urejamo. Za izhod iz tega pogleda kliknemo puščico nazaj na vrhu hierarhije (slika 4.10 levo).



Slika 4.10: Prefab je shranjen v datotečnem sistemu; lastnosti urejamo v inspektorju (desno), z dvojnim klikom odpremo prefab pogled (levo) in iz njega izstopimo z gumbom nazaj.

Primer 4.6: Podiranje kegljev

Da bomo lahko podirali keglje, jih moramo najprej ustvariti. Najprej ustvarimo kocko, ki jo poimenujemo »Kegelj« in jo preoblikujemo v pokončen kvader. Postavimo ga na tla *spalnice*. Ustvarimo še eno kocko, ki jo poimenujemo »Vrat« in jo preoblikujemo v manjši pokončen kvader, postavimo pa jo na vrh *keglja*. Ko smo zadovoljni z obliko, ustvarimo hierarhičen objekt tako, da *vrat* spremenimo v otroka *keglja* (ustvarjanje hierarhičnih objektov smo opisali v poglavju 3.2.2). Ideja, kako kegelj lahko izgleda, je prikazana na sliki 4.11 levo.

V datotečnem sistemu ustvarimo mapo `Prefabs` in jo odpremo v projektnem oknu. Nato vrstico *keglja* v hierarhiji povlečemo v mapo `Prefabs`. V mapi se ustvari datoteka `Kegelj`, vrstica v hierarhiji pa se obarva modro. Zdaj lahko iz mape v sceno povlečemo poljubno število kegljev in jih razporedimo v trikotno formacijo (slika 4.11 desno).

Keglje bomo z *žogo* podrli tako, da se bo ta odbila od klančine. V sceno pod njo dodamo klančino, ki jo najdemo v mapi `_Unity Essentials/Prefabs/Shapes`, poimenovana je `Ramp`. Ko vklopimo igralni način, *žoga* pade skozi *klančino* in se odbije od tal. Tako kot v primeru 4.4, moramo *klančini* dodati komponento collider in za ta primer izberemo Mesh Collider. Pazimo, da spremembe ustvarjamo, ko igralni način ni vključen. Ko ponovno zaženemo igralni način, se *žoga* odbije od *klančine*. S pomočjo orodij za premikanje in spreminjanje oblike prilagodimo *klančino* tako, da se *žoga* odbije v keglje.

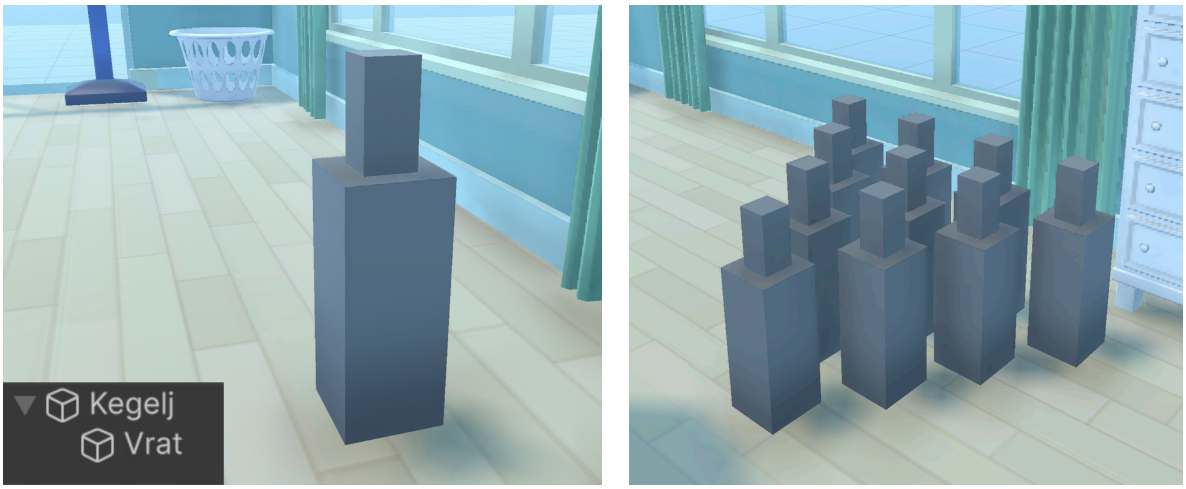
Žoga se sicer odbije v pravilno smer, a se od kegljev odbije, namesto da bi jih podrla. Njena sila nanje ne vpliva in jih ne premika. Da to popravimo, moramo kegljem dodati komponento Rigidbody. Namesto da bi vsakemu posebej dodajali komponente, v mapi `Prefabs` enkrat kliknemo na datoteko `Kegelj` in v inspektorju dodamo komponento Rigidbody[a]. Ta sprememba se bo samodejno uveljavila na vseh kegljih v sceni.

Ko zdaj poženemo igralni način, žoga lahko podre vse keglje.

Namig: Če se žoga zdi prelahka, ji lahko povečamo maso v komponenti Rigidbody.

Namig: Ustvarimo lahko nov prazen objekt, ki ga poimenujemo »Keglji«. Če vse ustvarjene keglje spremenimo v otroke kegljev, lahko enostavno premikamo ustvarjeno konstrukcijo po prostoru.

[a] Komponento Rigidbody dodamo samo starševskemu objektu in ne tudi otroku *keglja*.



Slika 4.11: Kegelj (levo) je sestavljen iz dveh kvadrov, pri čemer je objekt *kegelj* starš objektu *vratu*. Več kegljev lahko razporedimo v postavitev (desno).

Izziv 2: Podri stolp iz kock

Prejšnji primer je pokazal, kako lahko ustvarimo keglje, ki jih podiramo z žogo. Sedaj pa poskusimo narediti stolp iz kock, ki ga žoga podre.

Namig: če želimo postavljene kocke eno na drugi popolnoma poravnati, si lahko pomagamo tako, da pri uporabi orodja za premikanje objekta držimo tipko V. Tako vklopimo način poravnave po ogliščih (angl. vertex snapping). V tem načinu lahko izbrano oglišče objekta premaknemo natanko tako, da se dotika oglišča sosednje izbrane površine.

4.4 Zvok v Unity

V interaktivnih izkušnjah je pogosto zvok tisti, ki jih naredi zanimivejše in prepričljivejše. Brez zvočnih učinkov bi bila scena kuhinje, ne glede na kakovost grafike, videti mrtva. Zvok sporoča, da voda vre, da hladilnik brni ali da zunaj pojejo ptice.

Unityjev zvočni sistem temelji na simulaciji realnega sveta in deluje s pomočjo dveh ključnih komponent:

- Audio Source (*sl. zvočni vir*): Komponenta objekta, ki »oddaja« zvok v sceno (npr. lonec, radio, ptica).
- Audio Listener (*sl. poslušalec*): Komponenta objekta, ki zvok »sprejme«. Običajno imamo v sceni le en tak objekt, ta predstavlja ušesa igralca[3].

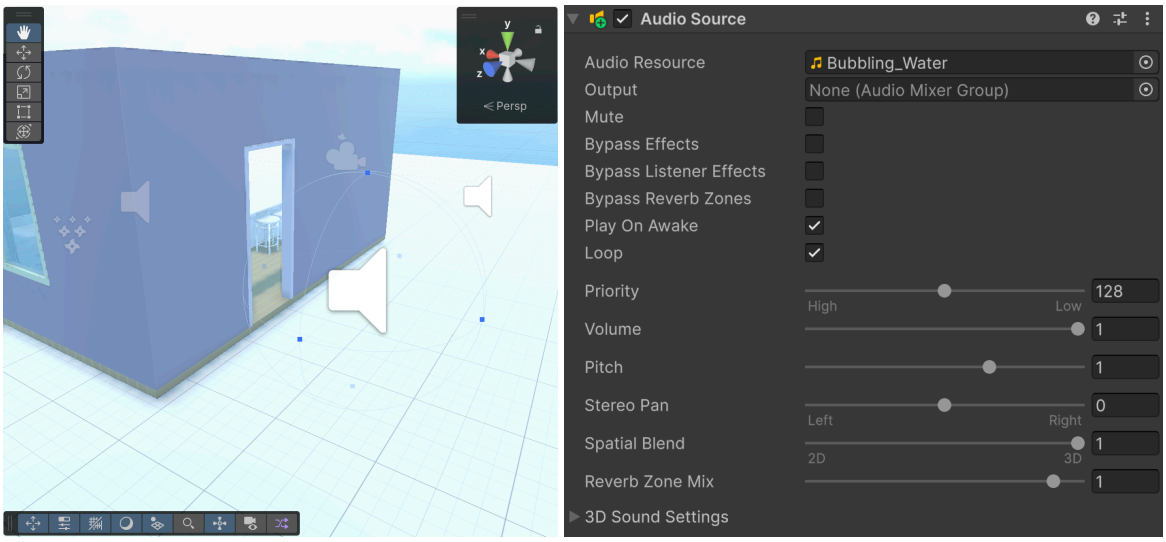
Komponenti Audio Source in Audio Listener lahko dodamo poljubnim objektom. Pogosto imamo v sceni več objektov s komponento Audio Source, komponento Audio Listener pa ima navadno le en objekt.

[3]Ponavadi je komponenta pripeta na objekt *Main Camera (sl. glavna kamera)*, ki uravnava pogled.

4.4.1 Komponenta Audio Source

Vsak objekt, ki naj bi oddajal zvok, potrebuje komponento Audio Source. Ta deluje kot predvajalnik, ki mu določimo zvočni posnetek in pravila predvajanja. Objekti s to komponento so lahko vidni (npr. lonec, radio), lahko pa gre za prazne objekte, ki imajo poleg transformacije le še komponento Audio Source (npr. objekt za glasbo v ozadju). V oknu scene so označeni z belo ikono zvočnika (v igralnem načinu ni vidna), kot levo na sliki 4.12.

Do komponente Audio Source dostopamo prek inspektorja izbranega objekta (glej sliko 4.12 desno). Glavni parameter je **Audio Resource** *(sl. zvočna datoteka)*, določimo pa ga z izbiro zvočne datoteke. Izbiramo lahko med datotekami projekta; če želimo dodati zvočno datoteko, ki še ni del projekta, jo lahko enostavno prilepimo v datotečni sistem v projektnem oknu. Vrednost polja lahko ponovno nastavimo tako, da datoteko iz projektnega okna povlečemo v polje, alternativno pa lahko s pritiskom gumba v obliki pike na desni strani polja odpremo pojavno okno, v katerem izberemo datoteko.



Slika 4.12: Objekti s komponento Audio Source so v oknu scene označeni z belo ikono zvočnika. Na levi sliki so vidni trije takšni objekti. Na desni sliki je prikazano polje komponente v inspektorju. Glavni parameter je **Audio Resource**, ki določa zvočno datoteko, ostali parametri pa se nanašajo na ostale nastavitve predvajanja zvoka.

Potrditveno polje **Play On Awake** *(sl. predvajaj ob začetku)* določa, ali se zvok začne predvajati takoj, ko zaženemo igralni način. Polje **Loop** *(sl. ponavljajoče)* določa, ali se zvočni posnetek predvaja ponavljajoče, drsnik **Volume** *(sl. glasnost)* pa določa jakost zvoka. Poseben učinek ima drsnik **Spatial Blend** *(sl. prostorsko prelivanje zvoka)*, s katerim določimo, ali je glasnost zvoka odvisna od oddaljenosti poslušalca od vira. Privzeto je ta vrednost nastavljena na 0, kar pomeni popolno neodvisnost od oddaljenosti poslušalca (ta nastavitev je primerna za zvoke, ki naj bi bili povsod v sceni enakomerno slišni). Največja možna vrednost tega polja je 1, pri kateri zvok postane del prostorske simulacije – glasnost in smer zvoka sta odvisni od razdalje vira do poslušalca.

Komponenta ponuja še vrsto naprednejših nastavitev, kot sta **Pitch** *(sl. višina tona)*

in **3D Sound Settings** *(sl. nastavitve 3D zvoka s krivuljo)*, vendar so za osnovno uporabo zgoraj našteti parametri povsem zadostni.

Primer 4.7: Lonec z vrelo vodo

Uporabili bomo sceno iz predloge, ki jo najdemo na lokaciji `Assets/_Unity Essentials/Scenes/3_Kitchen_Audio_Scene`.

V tej sceni so že postavljeni osnovni objekti, ki jih bomo potrebovali, vključno s poslušalcem. To je objekt *Main Camera*, ki ima komponento Audio Listener. Zdaj bomo v sceno dodali nekaj svojih zvokov. Najprej bomo dodali objekt lonec z vrelo vodo, ki se nahaja na lokaciji `_Unity Essentials/Prefabs/Kitchen/Boiling_Pot`, in ga postavili na *štedilnik*.

Da bi v igralnem načinu slišali brbotanje vode, moramo *loncu* dodati komponento Audio Source. V komponenti z gumbom na desni strani polja **Audio Resource** odpremo pojavno okno z iskalno vrstico in izberemo datoteko `Bubbling_Water`.

Ko poženemo igralni način, se lahko po sceni premikamo s tipkami `W`, `A`, `S` in `D`. V ozadju lahko slišimo zvok brbotanja vode, ampak le toliko časa, kolikor je dolga njegova zvočna datoteka. Zvok je zaenkrat enak ne glede na našo oddaljenost od *lonca*. Da se bo posnetek predvajal neprekinjeno, moramo označiti polje **Loop**. Poljubno lahko nastavimo še glasnost zvoka z uporabo drsnika **Volume**. Da bo zvok odvisen od pozicije poslušalca, prestavimo še drsnik **Spatial Blend** na vrednost `1`. Te spremembe v igralnem načinu omogočijo, da se zvok brbotanja vode neprekinjeno predvaja in postane glasnejši, ko smo bližje *loncu*.

4.4.2 Igralni objekt Audio Source

Objekt, ki oddaja zvok, lahko vedno enostavno ustvarimo tako, da naredimo prazen objekt in mu dodamo komponento Audio Source. Tak objekt je povsem ekvivalenten že pripravljenemu objektu Audio Source (*angl. zvočni vir*), ki preprosto prihrani nekaj klikov pri ustvarjanju scene. Najenostavneje ga ustvarimo tako, da v orodni vrstici urejevalnika izberemo možnost ≡ *GameObject* ▶ *Audio* ▶ *Audio Source*, ki v sceno doda ta objekt.

S tem objektom v sceno vključujemo zvoke, ki niso neposredno vezani na ostale vidne objekte, na primer glasbo v ozadju ali različne ambientalne zvoke.

Primer 4.8: Glasba v ozadju

Nadaljujemo primer 4.7 tako, da bomo sceni dodali različne ambientne zvoke in glasbo v ozadju. Najprej ustvarimo nov objekt, ki bo zvočni vir (lahko ustvarimo objekt Audio Source), in ga poimenujemo »GlasbaVOzadju«. V njegovi komponenti Audio Source izberemo eno izmed zvočnih datotek z glasbo (v predlogi je že vključenih nekaj različnih zvočnih datotek z imenom »Music_*«). Da bo ta zvočni vir res

služil kot glasba v ozadju, ga premaknemo nekam izven kuhinje in glasnost zvoka nekoliko zmanjšamo, da ostali zvočni učinki pridejo bolj do izraza. Pazimo, da vrednost drsnika **Spatial Blend** ostane na 0, saj bo tako zvok enako slišen povsod v sceni. Označimo lahko tudi polje **Loop**, da se bo glasba neprestano predvajala. Sedaj lahko preizkusimo igralni način.

V primeru bomo dodali še nekaj ambientalnih zvokov in sicer ptičje petje, ki se bo predvajalo naključno. Za ta namen ustvarimo tri objekte tipa zvočni vir in jih poimenujemo »Ptica#«, kjer # označuje zaporedno številko. Razporedimo jih okoli kuhinje poleg vrat in oken. Za zvočne datoteke izberemo nekatere izmed datotek z imeni vzorca »SFX_Bird_#«.

Da se bodo ptičji zvoki začeli predvajati ob naključnih trenutkih in bodo trajali različno dolgo, bomo uporabili že pripravljeno skripto, ki se nahaja med datotekami predloge. Objektom jo dodamo na enak način kot običajne komponente, v iskalno vrstico za dodajanje komponent vpišemo »Play Sound at Random Intervals«. Tako dodana skripta na objekt deluje kot komponenta, ki obnašanje objekta upravlja s pomočjo logike, zapisane v skripti (več o programiranju in skriptah v Unityju je v poglavju 6). V tej komponenti lahko določamo minimalno in maksimalno število sekund trajanja posameznega predvajanja zvoka.

Vsem objektom s ptičjim zvokom nastavimo vrednost **Spatial Blend** na 1 in odznačimo možnost **Play On Awake**, da bo komponenta Play Sound at Random Intervals imela vpliv na obnašanje zvoka. Če sedaj poslušamo zvoke v igralnem načinu, bi v ozadju morali slišati glasbo, od vrat in oken ob naključnih trenutkih ptičje petje ter brbotanje vode v bližini *lonca*.

Napredno upravljanje z mešalnikom zvoka (Audio Mixer)

V kompleksnejših projektih, kjer imamo na desetine različnih zvočnih virov, postane ročno nastavljanje glasnosti za vsak objekt posebej zamudno. Unity za ta namen ponuja objekt Audio Mixer (*sl. mešalnik zvoka*).

Ta deluje kot profesionalna mešalna miza, ki omogoča:

- združevanje zvokov v skupine (npr. skupina »Glasba«, »Učinki«, »Govor«),
- nastavljanje glasnosti za celotne skupine hkrati in
- dodajanje učinkov, kot sta odmev (*angl. reverb*) ali popačenje, na več virov hkrati.

V tem učbeniku se v podrobnosti mešalnika ne bomo spuščali, za nadaljnje raziskovanje pa priporočamo uradni Unity vodič: `https://learn.unity.com/tutorial/audio-setup`.

4.5 Vizualni učinki

Unity omogoča dodajanje različnih vizualnih učinkov, kot so dim, iskre, megla, ogenj in eksplozije. Ti temeljijo na animaciji velike količine majhnih delcev, ki jih sistem nenehno ustvarja, premika po določenih pravilih in nato uniči.

Učinke definiramo z objektom, ki vsebuje komponento Particle System (*sl. sistem delcev*). Ta vsebuje vrsto parametrov, ki so zaradi preglednosti razdeljeni v module. Glavni modul, poimenovan enako kot komponenta, določa splošne lastnosti, kot so trajanje učinka, hitrost, velikost, izgled in število delcev. Ostale module lahko po potrebi vklapljamo ali izklapljamo. Modul **Emission** *(sl. oddajanje)* določa število ustvarjenih delcev v sekundi, modul **Shape** *(sl. oblika)* pa določa obliko območja, iz katerega delci izvirajo (npr. stožec, krogla ali kocka).

Omejitev je le lastna ustvarjalnost

Učinki v Unityju so lahko izjemno napredni in kompleksni, zato je njihova uporaba omejena predvsem z vašo ustvarjalnostjo. Poleg sistema delcev Unity ponuja tudi VFX graf (*angl. VFX graph*), ki temelji na sistemu vozlišč in predstavlja izjemno zmogljiv način ustvarjanja učinkov, vendar zahteva več truda pri učenju. Ker se v tem učbeniku ne bomo podrobno posvečali njihovemu snovanju, prilagamo povezavo do uradnega vodiča, kjer je podrobno razloženo ustvarjanje učinkov tako s sistemom delcev kot z VFX grafom: `https://learn.unity.com/course/creative-core-vfx`.

Primer 4.9: Para iz lijaka

V primeru 4.7 smo loncu dodali komponento za oddajanje zvoka, zaradi katere je v oknu scene nad njim prikazana ikona zvočnika. Poleg nje so vidne še druge bele ikone, ki predstavljajo vizualne učinke pare in mehurčkov v vodi. Gre za učinka, ustvarjena s komponento Particle System, ki ju najdemo pod objektom *lonec* kot otroka z imenoma *Steam* in *Bubbles*.

Zdaj bomo v sceno dodali učinek, ki bo predstavljal paro iz *lijaka*. Ustvarimo ga z izbiro možnosti ≡ *GameObject* ▶ *Effects* ▶ *Particle System* v orodni vrstici urejevalnika in ga poimenujemo »ParaLijak«. V sceni ga prestavimo tako, da delci izhajajo iz *lijaka*. Koordinate bodo približno $X = 3{,}7$, $Y = 1$ in $Z = 1{,}8$. S pomočjo orodja za spreminjanje velikosti zmanjšamo območje, kjer se delci nahajajo. Lahko nastavimo tudi vrednosti v komponenti pod parametrom za velikost na $X = 0{,}03$, $Y = 0{,}05$ in $Z = 0{,}03$. V inspektorju v glavnem modulu lahko vidimo, da je označeno polje **Looping** *(sl. ponavljanje)*, zato se delci nenehno premikajo. Če bi to polje odznačili, bi se učinek izvajal samo določeno količino časa, kar lahko uravnavamo s poljem **Duration** *(sl. trajanje)*. Sedaj lahko preizkušamo še nastavitve drugih parametrov in opazujemo njihov učinek v igralnem načinu. Primer, kako naj bi po postavitvi izgledal objekt v oknu scene, je na sliki 4.13.

Izziv 3: Kuhinjo opremite z zvočnimi in vizualnimi učinki

Podano sceno dopolnite z več zvočnimi viri in vizualnimi učinki po vzoru učnih primerov. Ob vratih in oknih lahko dodate dodatno naključno oglašanje ptic (v projektu lahko najdete še več primernih zvočnih datotek). Dodajte tudi poljubne vizualne učinke. Vnaprej pripravljene vizualne učinke lahko najdete v mapi `_Unity Essentials/Prefabs/VFX`.

Slika 4.13: Postavitev učinka pare v lijaku v oknu scene.

4.6 Upodabljanje v pogonu Unity

Upodabljanje (*angl. rendering*) je eden temeljnih procesov vsakega grafičnega pogona. Gre za postopek pretvorbe navideznega tridimenzionalnega sveta, sestavljenega iz podatkov, kot so geometrija, pozicija luči in opis materialov, v dvodimenzionalno sliko, ki jo končni uporabnik vidi na zaslonu. Z drugimi besedami, prek procesa upodabljanja vsak objekt v sceni pridobi svoj videz. Popolno razumevanje tega procesa zahteva podrobno znanje računalniške grafike, zato se bomo v tem poglavju osredotočili predvsem na praktične osnove.

V praksi je upodabljanje v Unityju odvisno od treh elementov. Vsak viden objekt vsebuje geometrijo (*angl. geometry*), ki določa njegovo obliko. Kot smo spoznali v poglavju 4.3, nad geometrijo v pogonu Unity nimamo večjega vpliva; neposredno imamo prek urejevalnika dostop le do preprostih geometričnih objektov, kot so kocke in krogle, ki smo jih uporabljali pri spoznavanju fizike. Kompleksnejše geometrične objekte moramo uvoziti iz zunanjih virov, zato se geometriji v tem poglavju ne bomo posvečali.

Vsak viden objekt prav tako vsebuje material (*angl. material*), ki določa njegove vizualne lastnosti. Materiali so sestavni del pogona Unity, z njimi pa se bomo podrobneje spoznali v naslednjem podpoglavju. Tretji element, ki določa videz objektov, je luč (*angl. light*). Scena v pogonu Unity lahko vsebuje poljubno število luči, več o njih pa bomo prav tako podrobneje spoznali v nadaljnjem podpoglavju.

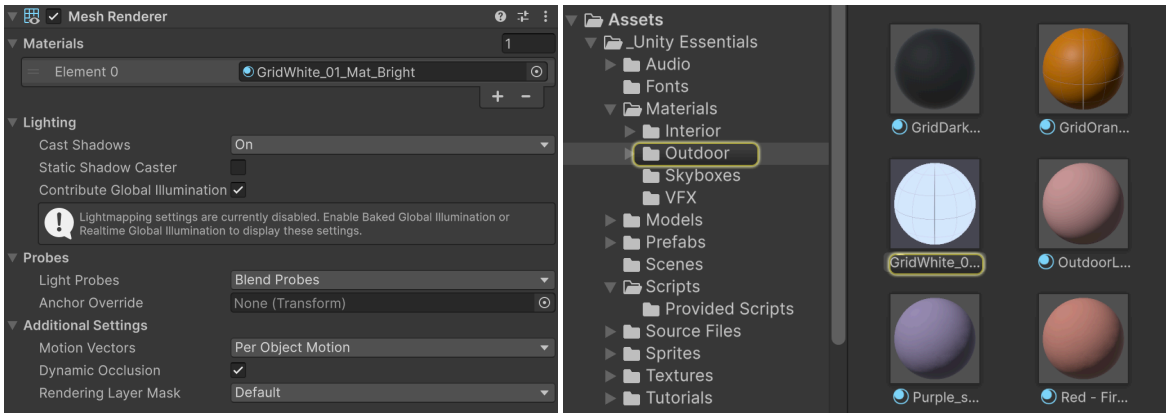
Pozor! Čeprav temu ne bomo namenili večje pozornosti, je na tej točki vredno omeniti, da je upodabljanje v 3D izkušnjah pogosto najzahtevnejši proces ter glavni razlog za njihovo počasno delovanje in *t. i.* zatikanje *(angl. stuttering)*. Brez poglabljanja v podrobnosti so za to največji krivci kompleksnost geometrije, število hkrati vidnih objektov in število luči v sceni. Kot začetniki se temu najlažje izognemo z uporabo preprostejših objektov ter preudarno postavitvijo luči.

4.6.1 Materiali

Vsak viden objekt v pogonu Unity vsebuje enega ali več materialov, ki natančno opisujejo njegov videz. Treba je poudariti, da material določa le videz in ne fizikalnih lastnosti. Medtem ko lahko krogli priredimo videz košarkarske žoge, to še ne pomeni, da bo prevzela tudi njene fizikalne lastnosti.

Materiali se v pogonu Unity ne nahajajo neposredno v sceni, temveč jih najdemo v datotečnem sistemu, preko projektnega okna. Takšna zasnova omogoča, da si enak material deli več objektov, ki se lahko nahajajo v različnih scenah. Material objektu določimo prek lastnosti **Materials** *(sl. material)* v komponenti Mesh Renderer (glej sliko 4.14 levo). Te vidnim objektom ni treba dodajati, saj jo že vsebujejo, kajti brez nje ne bi bili vidni. Alternativno lahko izbran material določimo objektu tako, da ga v projektnem oknu izberemo in z držanjem levega miškinega gumba material prenesemo na želeni objekt v sceni.

Druga pomembna lastnost materialov v pogonu Unity je, da ti niso neposredno povezani z geometrijo objekta. V teoriji to omogoča, da poljuben material dodelimo poljubnemu objektu, ne da bi pogonu Unity povzročali težave. To pa še ne pomeni, da bo vsak material videti smiselno na vsakem objektu, saj so v praksi vsi razen najpreprostejših enobarvnih materialov prilagojeni obliki objektov.



Slika 4.14: V komponenti Mesh Renderer (levo) prek lastnosti **Material** *(sl. material)* določimo videz objekta. Z dvoklikom na parameter **Material** se v projektnem oknu začasno označi izbrani material (desno).

Objekti z več materiali

Pogon Unity podpira tudi upodabljanje kompleksnejših objektov, ki so interno razdeljeni na več delov z različnim videzom. Če objekt ni razdeljen na tak način, dodajanje dodatnih tekstur nima vpliva, na kar nas bo uporabniški vmesnik tudi opozoril. Objektov z več materiali v okviru tega učbenika ne bomo srečali.

Primer 4.10: Ogled materiala objekta

V datotečnem sistemu izberemo sceno `6_Bonus_Custom_Scene`, v katero postavimo vnaprej pripravljen objekt *galerija*, ki ga najdemo na lokaciji `_Unity Essentials/Prefabs/Rooms/04_Gallery`.

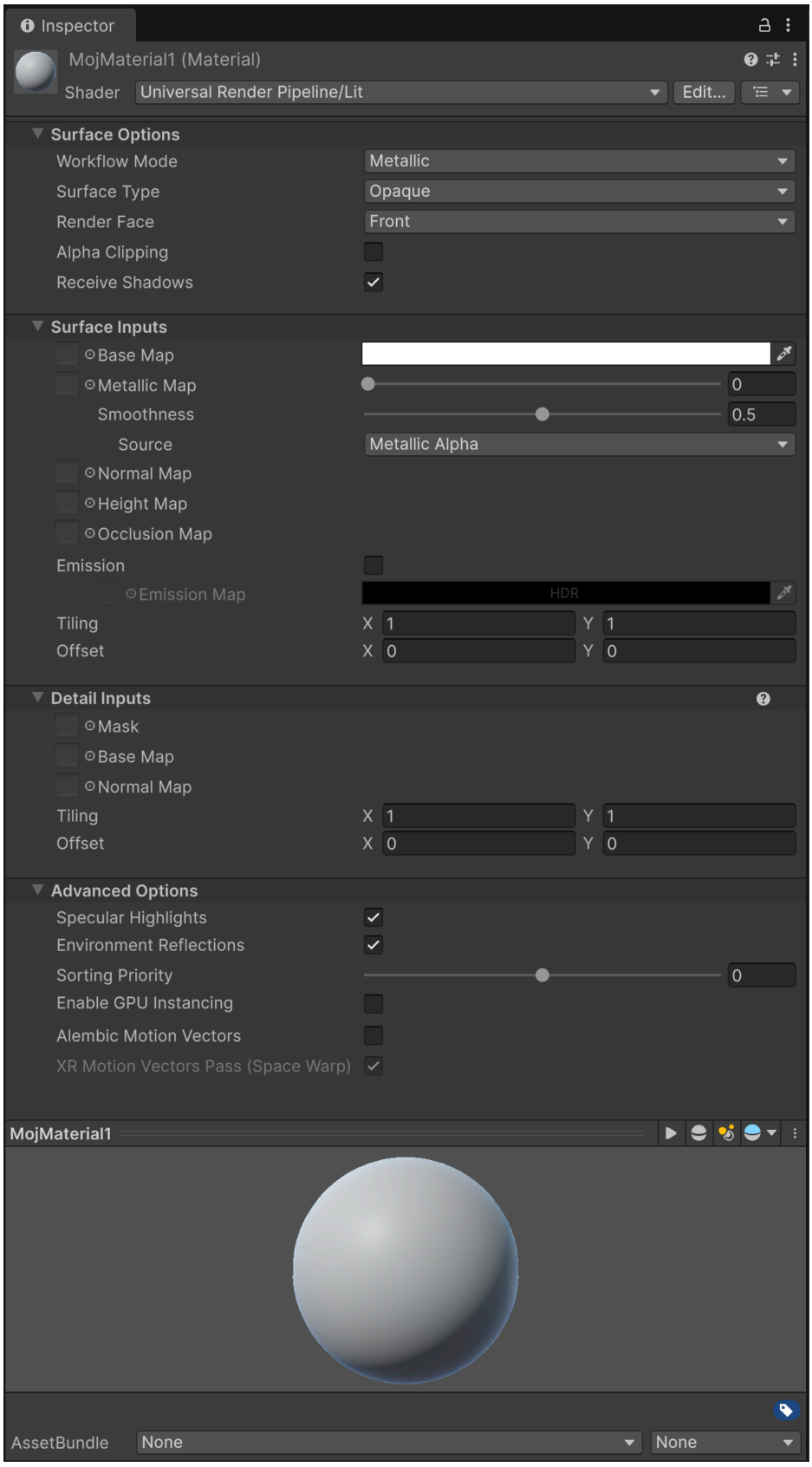
Pri pregledu objekta *galerija* hitro opazimo, da ima potomec, ki predstavlja tla, *mesh_galleryfloor*, drugačen videz kot ostali potomci. Iz tega lahko sklepamo, da vsebuje drugačen material, o tem pa se lahko prepričamo prek komponente Mesh Renderer, oz. natančneje, njene lastnosti Materials (slika 4.14 levo).

Če želimo izvedeti lokacijo tega materiala v datotečnem sistemu, lahko to najlažje storimo z dvoklikom na njegovo ime v komponenti. Pri tem se projektno okno premakne v mapo, kjer se material nahaja, izbrani material pa se tudi začasno obarva (slika 4.14 desno). V tej mapi so tudi drugi vzorčni materiali, ki so del uporabljene učne predloge. Ker materiali niso neposredno vezani na geometrijo, lahko prednastavljeni material na izbranem objektu tal zamenjamo s poljubnim materialom iz te mapi.

4.6.2 Ustvarjanje in lastnosti materialov

Ustvarjanje novega materiala v pogonu Unity je preprost postopek, podoben izdelavi drugih elementov v datotečnem sistemu. Material ustvarimo z desnim klikom v projektnem oknu in izbiro možnosti ≡ *Create* ▶ *Material*. Ustvarjeni material se nahaja v trenutno odprti mapi.

Popolno razumevanje lastnosti in delovanja materialov je za začetnika preobsežno, zato je spodaj predstavljenih nekaj osnovnih lastnosti materialov, do katerih dostopamo preko okna inspektor (slika 4.15). Omogočajo ustvarjanje preprostih enobarvnih materialov z nekaj osnovnimi fizikalnimi lastnostmi, kot sta **gladkost** in **kovinskost**.



Slika 4.15: Lastnosti materiala v oknu inspektor.

Naprednejši materiali

Ustvarjanje naprednejših materialov presega osnovno uporabo barv, tekstur in enostavnih nastavitev, ki smo jih obravnavali doslej. Takšni materiali pogosto vključujejo kompleksnejše senčenje *(angl. shading)*, uporabo večplastnih tekstur, napredne svetlobne učinke ter odzivanje na okoljske vplive, kot so odboji, prosojnost ali emisija svetlobe. Pogosto temeljijo na fizikalno osnovanem upodabljanju *(angl. Physically Based Rendering)*, kjer material posnema lastnosti resničnih površin in se ustrezno odziva na svetlobo v sceni.

Ustvarjanja naprednejših materialov se kot razvijalci začetniki pogosto ne lotevamo sami, saj to zahteva ne le poznavanje konceptov računalniške grafike, temveč tudi uporabo zunanjih namenskih aplikacij, kot sta Blender[a] ali Maya[b]. Namesto tega se uporabimo predpripravljene izdelke, ki jih v okviru uradne trgovine Unity Asset Store ponujajo njihovi avtorji. Več o tem v poglavju 5.

[a] `https://www.blender.org/`
[b] `https://www.autodesk.com/products/maya/overview`

1. **Možnosti površine** (*angl. surface options*) vsebuje parametre, ki določajo način upodabljanja in obnašanja materiala glede na druge objekte v prostoru.
   - **Workflow mode** določa model, ki se uporablja za izračun osvetlitve.
   - **Surface Type** *(sl. vrsta površine)* določa, ali je material prosojen (*angl. transparent*) ali neprosojen (*angl. opaque*).
   - **Receive Shadows** *(sl. sprejme sence)* določa, ali smejo drugi objekti metati sence na površino materiala.
2. **Vhodni parametri površine** (*angl. surface*) so glavni parametri, ki določajo videz materiala.
   - **Base Map** določa osnovni videz materiala, tu lahko nastavimo teksturo ali barvo.
   - **Metallic Map** *(sl. kovinska mapa)* določa, kako kovinska je površina. Za kovinske materiale navadno nastavimo vrednost 1, za nekovine pa 0.
   - **Smoothness** *(sl. gladkost)* določa, kako gladek ali hrapav je material. Višja vrednost pomeni bolj izrazite odseve.
   - **Source** določa, ali gladkost pridobiva podatke iz sivinskega kanala kovinske mape ali pa uporablja ločeno mapo.
   - **Tiling and Offset** *(sl. ponovitev in zamik)* določa, kako se teksture ponavljajo in kako so zamaknjene na površini objekta.

Primer 4.11: Dodajanje novega materiala

V tem primeru izhajamo iz scene, pripravljene v primeru 4.10. V projektnem oknu odpremo mapo `_Unity Essentials/Materials`. Kot smo spoznali v prejšnjem primeru, je tu že nekaj pripravljenih materialov, ki jih lahko uporabimo.

V projektnem oknu se premaknemo v mapo `Assets/Materials`. Z desnim klikom in izbiro možnosti ≡ *Create* ▶ *Material* ustvarimo nov material z imenom MojMaterial1. Ustvarjeni material izberemo v projektnem oknu, kar povzroči, da se v inspektorju prikažejo parametri materiala in predogled upodobitve materiala.

V parametru **Base Map** nastavimo barvo, ki bo v sceni izstopala, npr. živo rdečo. Nato v projektnem oknu ustvarjeni material povlečemo v okno scene na objekt *mesh_galleryfloor*, ki predstavlja tla. Postopek ponovimo in material dodamo še objektu *mesh_bench*, ki predstavlja klop sredi sobe.

Če na tej točki spremenimo katero izmed lastnosti materiala, bomo opazili, da smo spremenili videz vseh objektov, vezanih na ta material. To izhaja iz prej omenjenega dejstva, da so definicije materialov neodvisne od scene, zato je edini način, da dvema objektoma v pogonu Unity določimo drugačen videz, ta, da jima priredimo različna materiala, na kar moramo biti kot začetniki še posebej pozorni.

Izziv 4: Sceno opremite z materiali

S pomočjo lastnih materialov in tistih, ki so že vključeni v projekt, napolnite sceno po svojem okusu. Osredotočite se predvsem na notranjost *galerije*, kjer je veliko klopi, podstavkov, panelov itd. Material lahko nanesete tudi na notranje stene *galerije* in na nebo.

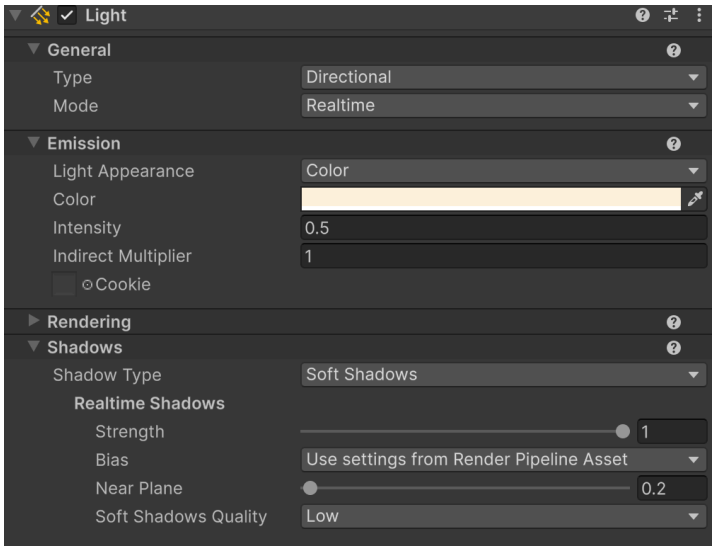
V galerijo lahko dodate tudi katerega od prefabov.

4.6.3 Osvetlitev

Izgled objektov v sceni je močno odvisen od njihove osvetlitve. Svetloba ustvarja kontraste med osvetljenimi in zasenčenimi deli površin, s čimer uporabniku omogoča občutek globine in prostorske orientacije, prav tako pa olajša prepoznavanje njihove medsebojne oblike in lege.

V pogonu Unity osvetlitev scene izhaja iz objektov, ki imajo komponento luč (*angl. Light*). To so lahko vidni objekti ali objekti brez mrežne geometrije in materialov. Pogon Unity iz objekta razbere le položaj luči v prostoru. Podobno kot pri drugih komponentah, lahko parametre komponente luč nastavljamo prek inspektorja (slika 4.16).

Najpomembnejša nastavitev komponente luč je **Type** *(sl. tip)*, ki določa vrsto svetlobnega vira oz. luči. Usmerjena luč (*angl. directional light*) simulira oddaljen vir, kot je sonce, medtem ko točkasta luč (*angl. point light*) svetlobo oddaja enakomerno v vse smeri, podobno kot žarnica. Reflektor (*angl. spot light*) osvetljuje le omejen stožčast



Slika 4.16: Komponenta luč (*angl. Light*) določa, kako objekt v sceni oddaja svetlobo.

prostor in se pogosto uporablja za simulacijo svetilk ali odrskih reflektorjev. Ne glede na izbrano vrsto luči, lahko na komponenti luč svetlobi nastavimo tudi **Intensity** *(sl. jakost)* in **Color** *(sl. barva)*. **Intensity** določa svetlost osvetlitve, barva pa omogoča nastavitev barvnega tona, ki ga oddaja svetlobni vir. Točkastim lučem in reflektorjem lahko nastavimo tudi **Range** *(sl. domet)*, ki omejuje učinek svetlobnega vira na oddaljene objekte, pri čemer objekti oziroma deli objektov zunaj dometa niso osvetljeni.

Izziv 5: Pred galerijo postavite kavarno

Galeriji bi radi dodali majhno kavarno na zunanji strani. Postavite kakšno mizo in stol, posebej pa se potrudite z lepo razsvetljavo. Na mizah naj bo majhna lučka, nad vhodom v galerijo pa naj bodo luči tipa reflektor.

4.6.4 Načini upodabljanja

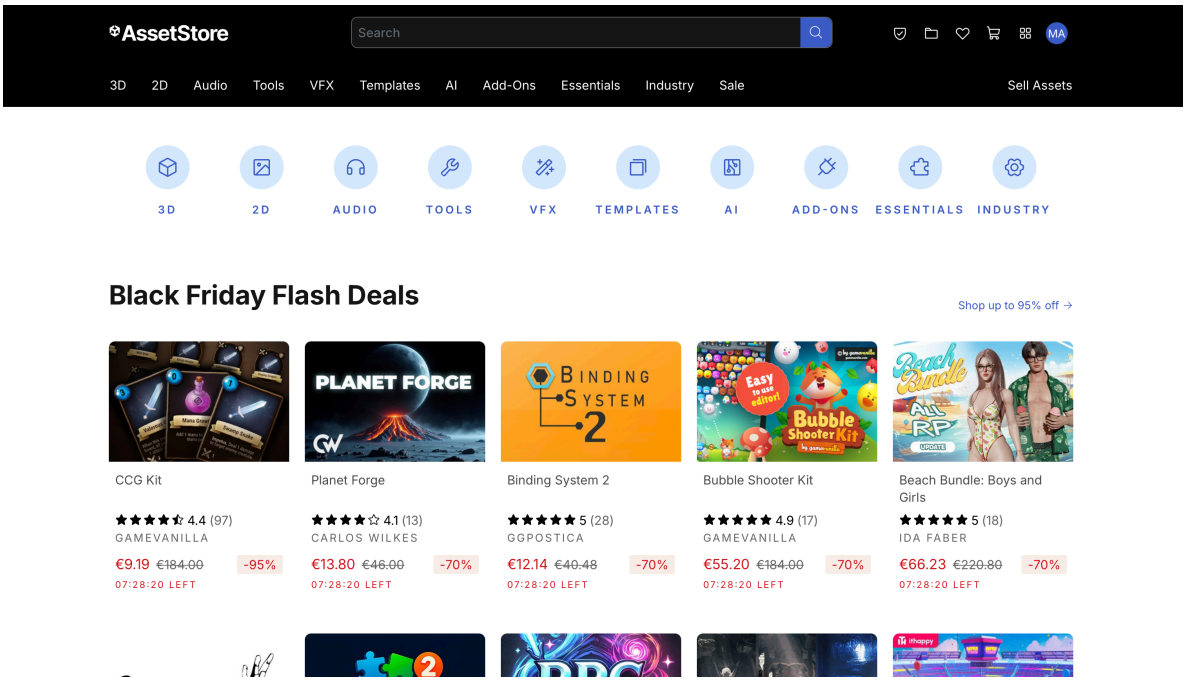
Unity omogoča tri načine upodabljanja, imenovane upodabljalni cevovodi (*angl. render pipelines*). Med seboj se razlikujejo predvsem po pristopu k ustvarjanju naprednih grafičnih učinkov in po učinkovitosti upodabljanja. Najprej omenimo univerzalni upodabljalni cevovod (*angl. Universal Rendering Pipeline*) (URP), ki je zelo prilagodljiv in primeren za uporabo na številnih, tudi manj zmogljivih napravah in sistemih, vključno z napravami razširjene resničnosti in VR očali. Če nimamo posebnih zahtev, je to najprimernejši cevovod za začetno uporabo. Preostala sta še visokoločljivostni upodabljalni cevovod (*angl. High Definition Render Pipeline*) (HDRP) in vgrajeni upodabljalni cevovod (*angl. Built-In Render Pipeline*). Prvi se uporablja za fotorealistično upodabljanje na zmogljivejših napravah, kot so igralne postaje in konzole, drugi pa je bolj splošen in manj prilagodljiv ter namenjen predvsem zagotavljanju združljivosti s starejšimi projekti in zunanjimi knjižnicami. Upodabljalni cevovod izberemo že na začetku, ko ustvarimo nov projekt. Vse nastavitve in začetne datoteke se nato prilagodijo izbranemu cevovodu.

POGLAVJE 5

Unity trgovina z gradivi

Scene smo doslej gradili z osnovnimi geometrijskimi liki ali vnaprej pripravljenimi predlogami. Vendar se v procesu razvoja iger hitro pojavi vprašanje: ali lahko v svojem projektu uporabimo kompleksnejše objekte, ki so jih pripravili profesionalni umetniki in razvijalci?

Odgovor se skriva v **Unity Asset Store**. To je uradna spletna trgovina, kjer razvijalci z vsega sveta delijo in prodajajo svoje izdelke: od 3D modelov, materialov in tekstur do kompleksnih programskih skript in celotnih zvočnih podlag. Trgovina vključuje plačljive in brezplačne vsebine, do nje pa dostopamo s svojim Unity računom na povezavi `assetstore.unity.com`. Unity Asset Store deluje po principu digitalne knjižnice. Ko določen paket *(angl. Asset)* enkrat »kupite« (tudi če je brezplačen), se ta trajno doda v vašo osebno knjižnico (*My Assets*) in ga lahko uporabite v poljubnem številu svojih projektov.



Slika 5.1: Vstopna stran Unity Asset Store.

5.1 Iskanje in filtriranje vsebin

Vstopna stran Unity Asset Store je prikazana na sliki 5.1. Pri brskanju med tisoči paketov je ključno poznati zahteve svojega projekta. Iščemo lahko po kategorijah (npr. 3D, 2D, zvok); vsaka krovna kategorija je nadalje razdeljena v podkategorije.

Pri izbiri moramo biti posebej pozorni na dva tehnična vidika:

- **Glavna različica urejevalnika:** Avtorji pakete pripravljajo za specifične izdaje Unityja. Pri filtrih zato vedno označimo številko **glavne različice**[1] (npr. Unity 6000.x), da se izognemo napakam pri uvozu.
- **Način upodabljanja:** Nekateri vizualni paketi delujejo le v specifičnih načinih (npr. URP ali HDRP). Če uvozimo material, ki ni namenjen izbranemu načinu upodabljanja, bo objekt v sceni videti enobarvno rožnat.

Iskanje olajša uporaba filtrov, s katerimi lahko določimo razpon cene (npr. *Free Assets*), glavno različico urejevalnika, avtorja ali oceno uporabnikov. Pri uporabi iskalne vrstice na vrhu strani velja opozorilo: nekateri izbrani filtri se ob osvežitvi strani ali novem iskanju lahko ponastavijo, zato jih je smiselno pred končno izbiro ponovno preveriti.

5.2 Uvažanje paketa v svoj projekt

Postopek dodajanja vsebin iz trgovine Asset Store v projekt se razlikuje od običajnega prenašanja datotek s spleta. Namesto ročnega prenosa datotek in njihovega uvoza, Unity vse kupljene vsebine hrani v knjižnici uporabniškega računa. Vsak paket, ki ga izberete na spletu, se najprej poveže z vašim Unity računom. Šele nato ga lahko s pomočjo vgrajenega orodja Package Manager (*sl. paketni upravitelj*) prenesete in uvozite v poljuben projekt na svojem računalniku. Ta pristop omogoča lažje posodabljanje vsebin in zagotavlja, da so vaši paketi vedno na voljo v oblaku, ne glede na to, na kateri napravi delate.

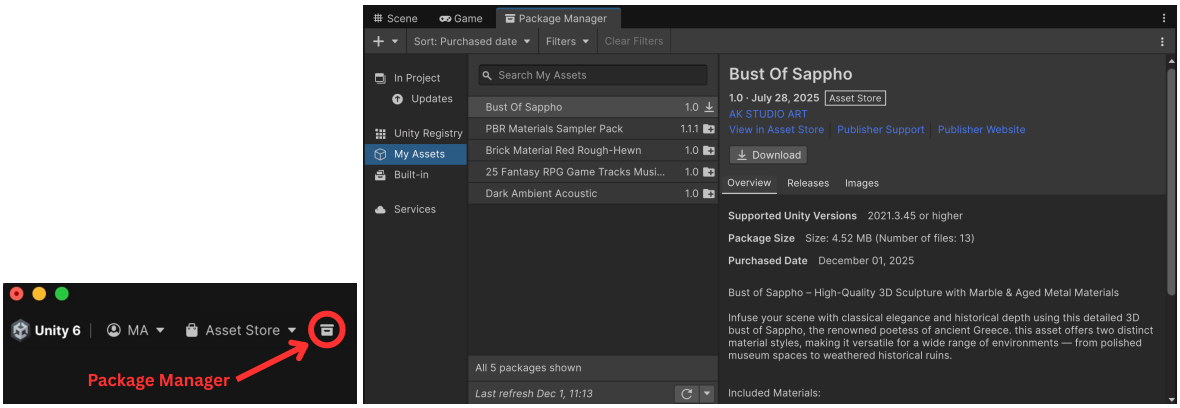
Primer 5.1: Uvoz doprsnega kipa v galerijo

V tem primeru bomo v svojo sceno z galerijo uvozili brezplačen 3D model z imenom »Bust of Sappho«. Sledimo spodnjemu zaporedju korakov:

1. **Izbira v brskalniku:**
   Obiščemo stran Unity Asset Store in z iskalno vrstico poiščemo paket »Bust of Sappho«. Na spletni strani si lahko ogledamo opis vsebine. Podatki razkrivajo, da je bil model ustvarjen v različici *2021.3.45f1* in je združljiv z vsemi tremi glavnimi cevovodi za upodabljanje (*Built-in*, *URP* in *HDRP*).
   Da paket dodamo v svojo digitalno knjižnico, ga moramo »kupiti« (ta paket je brezplačen). To storimo z gumbom `Add to My Assets`.

[1] Različice se v filtrih ločijo le glede na številko glavne izdaje (angl. *major version*, npr. 2022.x ali 6000.x). Podrobnejše številke, ki sledijo glavni izdaji, običajno ne vplivajo na združljivost.

2. **Prenos preko paketnega upravitelja:**
   V Unityju do zavihka s paketnim upraviteljem dostopamo z izbiro možnosti ≡ *Window* ▶ *Package Management* ▶ *Package Manager* v orodni vrstici urejevalnika ali z gumbom na vrhu urejevalnika, ki je označen na sliki 5.2. Zavihek Package Manager se odpre v ločenem oknu, lahko ga z miško primemo za ime zavihka in dodamo med okna v urejevalniku. Najenostavneje ga je dodati na sredino poleg zavihkov okna scene in okna igre.
   V skrajno levem meniju izberemo možnost My Assets, kar na desni prikaže vse pakete, ki jih imamo v svoji digitalni knjižnici. Izberemo Bust of Sappho (pogled je podoben desni sliki na sliki 5.2) in pritisnemo gumb Download *(sl. prenesi)*. Ko je prenos končan, pritisnemo gumb Import *(sl. uvozi)*. Odpre se okno s seznamom datotek. Pomembno je, da si zapomnimo ime mape na najvišji ravni. Nato v spodnjem desnem kotu ponovno kliknemo gumb Import.
3. **Postavitev v sceno:**
   Sedaj moramo preveriti spremembe v datotečnem sistemu. Opazimo lahko, da se je v mapi `Assets` pojavila podmapa `AK Studio Art` (ime mape na najvišjem nivoju iz prejšnjega koraka). Ta vsebuje pravkar uvoženo vsebino, ki je prav tako razdeljena na podmape. Poiščemo objekt doprsnega kipa v podmapi `Prefabs` in ga povlečemo v okno scene.
   Opomba: V oknu scene *je kip najverjetneje rožnate barve. To bomo razrešili v naslednjem poglavju.*



Slika 5.2: Na vrhu urejevalnika na levi strani najdemo gumb, s katerim odpremo okno paketnega upravitelja. Na sliki je označen z rdečo (levo). Na desni sliki je prikazan videz okna paketnega upravitelja, ko je izbran paket, ki ga želimo prenesti in uvoziti.

5.2.1 Pretvarjanje materialov uvoženih paketov

Kot smo opazili v primeru 5.1, se uvoženi objekti v sceni včasih prikažejo v značilni **rožnati barvi**. To ne pomeni, da je z objektom kaj narobe, temveč gre za neskladje med materiali paketa in nastavitvami vašega projekta.

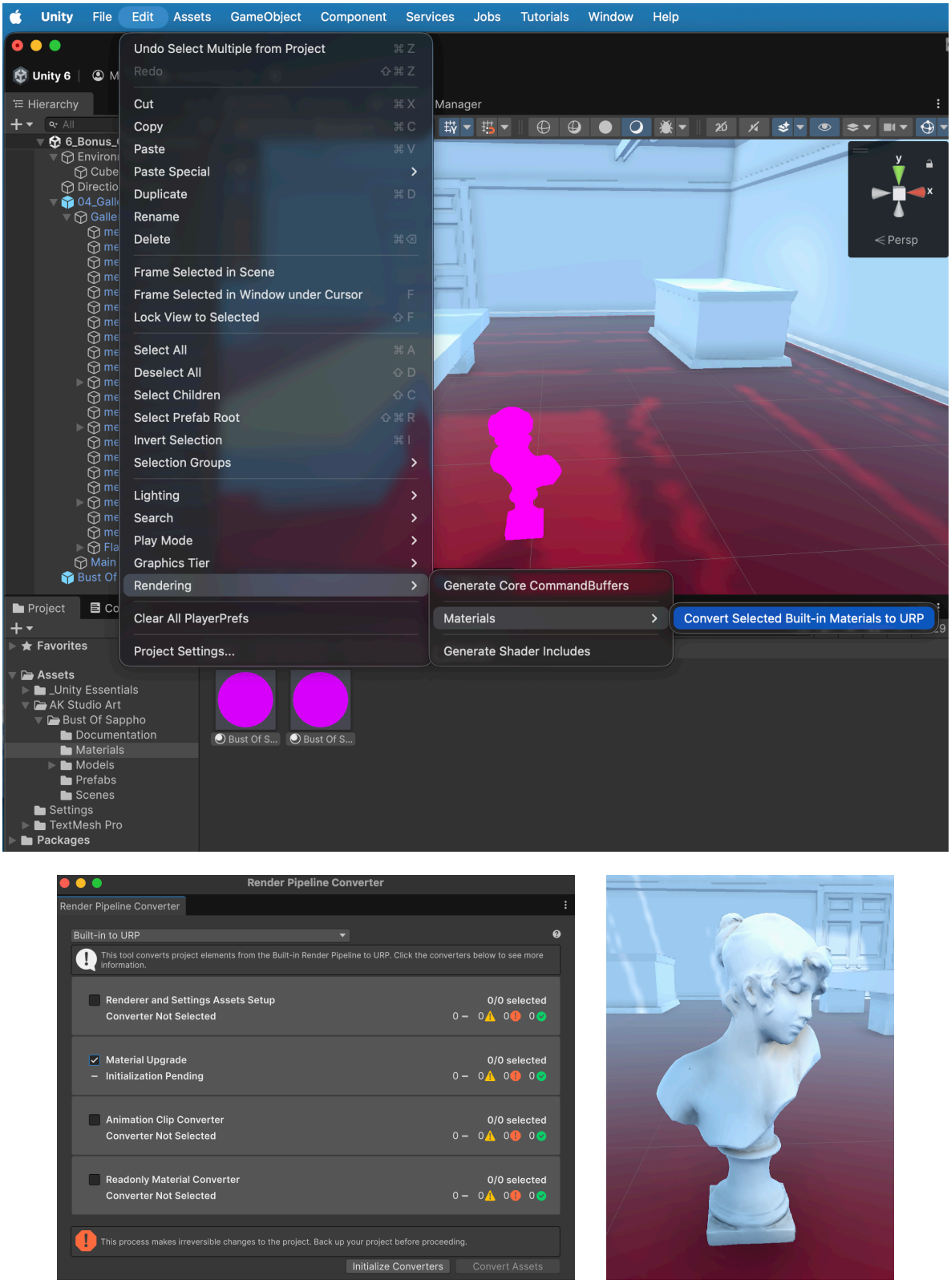
Večina starejših paketov uporablja materiale za standardni cevovod (*Built-in*), medtem ko sodobni projekti (kot je naš) uporabljajo **URP** (*Universal Render Pipeline*). Ker je model »Bust of Sappho« pripravljen na te razlike, lahko materiale pretvorimo le z nekaj kliki.

Pozor! Čeprav je orodje za pretvorbo materialov zelo priročno, pretvarjanje ni vedno uspešno. Uspešnost pretvorbe je odvisna od tega, kako kompleksni so izvorni materiali: pretvorba je skoraj vedno uspešna pri materialih, ki uporabljajo standardne Unityjeve senčilnike (*angl. shaders*), težave se lahko pojavijo pri senčilnikih, ki jih je avtor ustvaril sam. Če pretvorba ni uspešna, objekt ostane rožnat.

Primer 5.2: Odprava rožnate barve na kipu

1. V projektnem oknu poiščite mapo, kjer so shranjeni materiali za uvoženi kip: `Materials`.
2. Označite oba materiala v mapi (kliknite prvega, nato pridržite tipko `SHIFT` in kliknite drugega ali enostavno s kombinacijo `CTRL+A`).
3. **Če uporabljate različico urejevalnika s številko 6.3 ali več:** V zgornji orodni vrstici izberite ≡ *Window* ▶ *Rendering* ▶ *Render Pipeline Converter*. Odpre se pojavno okno, v katerem v zgornjem delu pustimo nastavitev »Built-in to URP« in označimo Material Upgrade. Nato v spodnjem desnem kotu najprej izberemo gumb Initialize Converters *(sl. inicializiraj pretvornike)* in potem Convert Assets *(sl. pretvori elemente)*.
   Sicer (urejevalniki z različico 6.2 in manj): V zgornji orodni vrstici izberite: ≡ *Edit* ▶ *Rendering* ▶ *Materials* ▶ *Convert Selected Built-In Materials to URP* (glej sliko 5.3 zgoraj).
4. V pojavnem oknu potrdite izbiro s klikom na Proceed.

Kip v oknu scene bi moral sedaj izgubiti rožnato barvo in prikazati pravilno teksturo kamna (glej sliko 5.3 desno spodaj). Ker pri nekaterih paketih materiali niso samodejno prilagojeni uporabljenemu uporabljalnemu cevovodu, jih moramo pretvoriti ročno. Postopek pretvorbe je odvisen od različice urejevalnika. Pri novejših različicah (6.3 ali več) uporabimo orodje ≡ *Render Pipeline Converter*, kjer izberemo ustrezen pretvornik in izvedemo pretvorbo materialov. Pri starejših različicah (6.2 in manj) pa je postopek poenostavljen in dostopen neposredno prek menija za pretvorbo izbranih materialov. Če je pretvorba uspešna, se materiali pravilno posodobijo, kar se odrazi tudi v vizualnem prikazu objekta v sceni.



Slika 5.3: Pretvorba materialov v URP in rezultat pravilnega prikaza objekta v sceni.

5.3 Raznolikost vsebin v trgovini Asset Store

Ko boste raziskovali trgovino, boste opazili, da paketi niso omejeni le na posamezne modele. Glede na svoje potrebe lahko iščete:

- **Sestavljene sete (Asset Packs):** Paketi, ki vsebujejo celoten nabor predmetov na določeno temo (npr. komplet pisarniškega pohištva, srednjeveško orožje ali kuhinjski pripomočki).
- **Specializirane vsebine:** Nekateri paketi vsebujejo zgolj knjižnice materialov, tekstur visoke ločljivosti ali zvočnih učinkov.
- **Celovita okolja (Environment Packs):** To so najobsežnejši paketi, ki vključujejo že pripravljene celotne scene, kot so gorate pokrajine, gosti gozdovi ali mestna središča. Takšni paketi prihranijo tedne dela, saj ponujajo že pripravljeno osnovo, ki jo nato le še prilagodite svoji viziji.

Pozor! Uvažanje velikih paketov (predvsem celovitih okolij) lahko zasede veliko prostora na disku in upočasni odpiranje projekta. Priporočljivo je, da v projekt uvažate le tiste vsebine, ki jih dejansko nameravate uporabiti.

Izziv 6: Napolnite muzej

Na Unity Asset Store poiščite objekte, ki bi jih vključili v svoj muzej ali galerijo. Obstaja veliko brezplačnih primernih vsebin, ki jih lahko uporabite. Naj bo galerija zanimiva, polna raznovrstnih eksponatov in slik. S pomočjo poznavanja svetlobnih objektov primerno postavite luči, da bodo eksponati lepo osvetljeni. Sceni dodajte primerno glasbo. Sprehod po galeriji lahko preizkusite v igralnem načinu, saj scena že vsebuje objekt igralca.

POGLAVJE 6

Kamera, igralec in načela programiranja

Doslej smo se naučili graditi svetove, jim dodajati zvočne in vizualne učinke, jih osvetljevati ter vanje uvažati objekte. Te scene pa so trenutno le statičen nabor objektov, ki kvečjemu lahko simulira nekaj fizikalnih sil. Da dosežemo pravo interaktivnost v sceni, potrebujemo tri ključne elemente: Igralca (*angl. player*) (objekt, ki ga uporabnik nadzoruje), Kamero (*angl. camera*) (»oči« uporabnika, ki določajo, kaj in kako vidimo navidezni svet) in logiko (logična pravila, ki sprožijo dogodke v sceni in jih določimo s programskimi skriptami).

Če imamo v sceni objekt igralca, uporabniku omogočimo, da se sprehaja po prostoru in interagira z različnimi deli scene. To je element, s katerim navidezno sceno spremenimo v interaktivno digitalno izkušnjo. V večini primerov, ki smo jih v tem učbeniku predstavili, smo se v igralnem načinu po sceni lahko premikali s tipkami `W`, `A`, `S` in `D`. V primeru 3.5 smo objekt igralca videli pred seboj, v ostalih primerih (npr. primera 4.7 in 4.10) pa igralca sicer nismo videli, a smo se kljub temu lahko premikali.

Kadar imamo v sceni objekt igralca, mu lahko dodamo možnost gibanja s pomočjo enostavne datoteke s kodo. Datoteke s kodo oz. skripte omogočajo, da v sceno dodamo različna logična pravila in pogoje, ki vplivajo na obnašanje objektov. V Unityju so skripte zapisane v programskem jeziku `C#`, kar je za popolne začetnike v programiranju lahko zahtevno. Zato smo v tem učbeniku pripravili nekaj osnovnih vodil za programiranje in urejanje kode.

6.1 Postavitev igralca in prvi stik s kodo

V Unityju lahko objektom dodajamo logična pravila s pomočjo skript, zapisanih v jeziku `C#`. To so besedilne datoteke s končnico `.cs`, ki jih moramo pisati v posebni strukturi oz. moramo upoštevati sintakso programskega jezika. Ker so to besedilne datoteke, jih lahko urejamo v preprosti beležnici, pogosta izbira za urejanje teh datotek pa so naprednejša programska orodja, kot na primer *Notepad++* ali *Visual Studio Code*. Takšni urejevalniki kode pripomorejo pri pisanju tako, da barvno označujejo ključne besede in pomagajo pri zaznavanju napak v sintaksi.

Objekt igralca v sceni najlažje ustvarimo tako, da izbranemu objektu dodamo posebno skripto (dodamo jo na enak način kot komponente), ki vsebuje logiko za premikanje.

Pozor! Pogon Unity je še vedno v razvoju in redno prejema posodobitve. Pred kratkim so razvijalci predstavili nov sistem za zaznavanje vnosov *(angl. input system)* (na primer prepoznavanje pritiska tipk na tipkovnici, klikov miške ali dotikov na zaslonu), ki bo v prihodnjih različicah postal privzeta izbira. Ta sistem ponuja več zmogljivosti, vendar je tudi nekoliko bolj kompleksen, zato ga v tem učbeniku nismo vključili.

Izbira sistema za zaznavanje vnosov je vezana na posamezen projekt. Najdete jo v orodnem meniju ≡ *Edit* ▶ *Project Settings* ▶ *Player* ▶ *Active Input Handling*, kjer lahko izberete med starim sistemom (Input Manager), novim (Input System Package) ali kombinacijo obeh (Both). Za primere v nadaljevanju priporočamo uporabo starega sistema ali kombinacije.

Primer 6.1: Priprava igralca in skripte za premikanje

Uporabili bomo sceno iz predloge `4_LivingRoom_Programming_Scene`. V njej je že pripravljena dnevna soba z nekaj pohištva. Za začetek bomo ustvarili mini igro, s katero bomo spoznali osnove programiranja.

Najprej bomo v sceno dodali objekt igralca. V datotekah predloge v mapi `Prefabs/Characters` izberemo poljuben objekt in ga postavimo na sredino dnevne sobe. V hierarhiji vrstico z njegovim objektom poimenujemo »Igralec«.

Da igralcu omogočimo premikanje, moramo ustvariti skripto. V datotečnem sistemu najprej v mapi `Assets` ustvarimo novo mapo z imenom »Scripts« in jo nato odpremo. V projektnem oknu nato z desnim klikom izberemo možnost ≡ *Create* ▶ *MonoBehaviour Script*. Poimenujemo ga »PlayerController«. Če v projektnem oknu označimo datoteko s kodo, se v inspektorju prikaže njen predogled. Z dvojnim klikom datoteko odpremo v primarnem urejevalniku kode (npr. *Visual Studio Code*, ki je prosto dostopen na `https://code.visualstudio.com/download`).

Datoteka trenutno vsebuje le osnovno predlogo, zato jo dopolnimo tako, da ustreza spodnjemu primeru kode. Več o natančnem pomenu posameznih ukazov si lahko preberete v spletni dokumentaciji Unityjevega aplikacijskega programskega vmesnika[a].

[a] `https://docs.unity3d.com/ScriptReference/index.html`

```
public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f; // Set player's movement speed.
    public float rotationSpeed = 120.0f; // Set player's
       rotation speed.

    private Rigidbody rb; // Reference to player's Rigidbody.

    // Start is called before the first frame update
    private void Start()
    {
        rb = GetComponent<Rigidbody>(); // Access player's
           Rigidbody.
    }

    // Update is called once per frame
    void Update()
    {

    }

    // Handle physics-based movement and rotation.
    private void FixedUpdate()
    {
        // Move player based on vertical input.
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = transform.forward * moveVertical *
           speed * Time.fixedDeltaTime;
        rb.MovePosition(rb.position + movement);

        // Rotate player based on horizontal input.
        float turn = Input.GetAxis("Horizontal") *
           rotationSpeed * Time.fixedDeltaTime;
        Quaternion turnRotation = Quaternion.Euler(0f, turn, 0
           f);
        rb.MoveRotation(rb.rotation * turnRotation);
    }
}
```

Osnovna zgradba skripte `PlayerController.cs`

Vsaka Unity skripta (MonoBehaviour) vsebuje dve ključni metodi:

- `Start()`: Se izvede, ko je objekt prvič vključen v sceno. Tu ponavadi nastavljamo začetne vrednosti parametrov.
- `Update()`: Se izvede pred izrisom sličice (angl. *frame*) na zaslon. Pogostost izvajanja je odvisna od hitrosti izrisa; če vaša igra teče s 60 sličicami na sekundo, se ta metoda izvede 60-krat v eni sekundi.

Pogosto dodamo tudi metodo `FixedUpdate()`, ki deluje podobno kot metoda `Update()`, le da se izvede v enakomernih intervalih ne glede na hitrost izrisa. Najpogosteje se uporablja za izračun fizike in premika igralca.

Primer 6.2: Dodajanje skripte in upravljanje gibanja igralca

Zdaj moramo skripto dodati igralcu. To lahko storimo na dva načina: s potegom skripte iz projektnega okna na *igralca* v oknu scene ali tako, da v inspektorju *igralca* dodamo komponento z imenom Player Controller.

Dodana skripta omogoča, da *igralca* v igralnem načinu premikamo s tipkami `W`, `A`, `S` in `D` ali s puščicami. V komponenti Player Controller najdemo dva nastavljiva parametra: **Speed** *(sl. hitrost)* in **Rotation Speed** *(sl. hitrost obračanja)*, ki nadzirata hitrost premikanja *igralca* in ju lahko poljubno spreminjamo.

6.2 Kamera

Svet okoli nas je tridimenzionalen, a ga naše oči zaznavajo tako, da zajemajo dvodimenzionalne podobe, ki jih možgani nato interpretirajo v globino. Podoben proces se dogaja pri fotografiranju ali snemanju videa, kjer tridimenzionalni prostor »sploščimo« na ravnino slike oziroma zaslona.

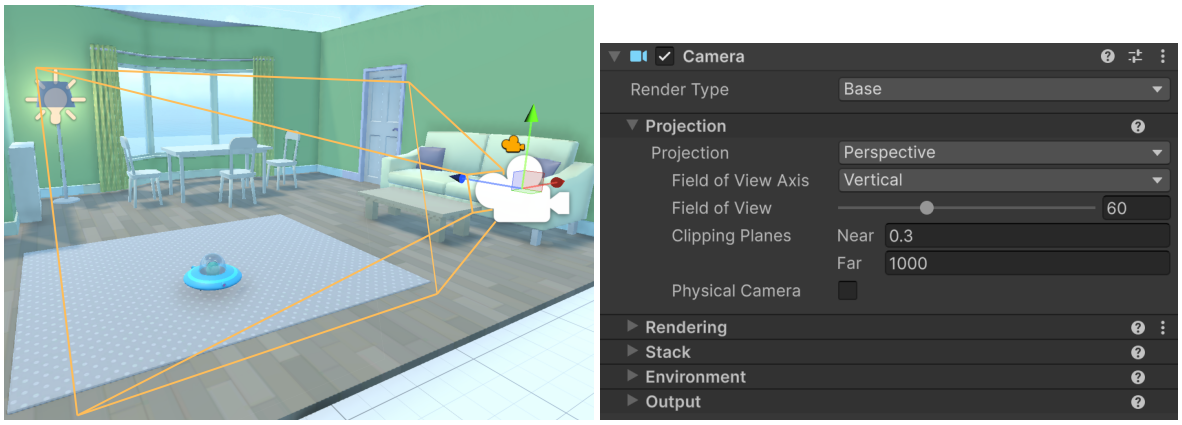
V okolju Unity je postopek enak. Čeprav gradimo kompleksne tridimenzionalne scene, uporabnik končni rezultat vedno vidi skozi lečo kamere, ki prostor pretvori v dvodimenzionalno sliko v oknu igre. Objekt kamere v sceni tako ne predstavlja le opazovalca, temveč dobesedno določa »oči« uporabnika in njegovo perspektivo na dogajanje.

6.2.1 Komponente in delovanje kamere

Objekt *kamera* v Unityju je vsak objekt, ki med svojimi komponentami vsebuje komponento Camera (*sl. kamera*). Ta določa tehnične lastnosti pogleda, kot so način projekcije pogleda, velikost zornega kota in ostale nastavitve. V oknu scene je kamera označena z belo ikono, njeno vidno polje pa s tankimi črtami, ki v piramidni obliki izhajajo iz nje, podobno kot je prikazano na sliki 6.1 levo. To piramidno obliko imenujemo vidni stožec (*angl. View Frustum*) in je določena z dvema mejnima ravninama (*angl. Clipping*

Planes). Kamera, ki uporablja perspektivni pogled, v svojo projekcijo zajame prostor med tema ravninama, ostali del scene pa uporabniku ni viden.

Komponenta Camera vključuje več nastavitev, ki vplivajo na končni videz slike v oknu igre. Primer polja v inspektorju je na sliki 6.1 desno. **Projection** *(sl. projekcija)* določa, kako se tridimenzionalni prostor preslika na zaslon. Najpogosteje je izbrana nastavitev **Perspective** *(sl. perspektiva)*, kjer so oddaljeni objekti videti manjši, kar ustvarja občutek globine. Druga možnost, **Orthographic** *(sl. ortografska projekcija)*, povzroči, da so objekti videti enako veliki ne glede na njihovo oddaljenost od kamere. Ker ni perspektive, se globina prostora izgubi, kar je v nekaterih primerih zaželeno.



Slika 6.1: Levo: Kamera v oknu scene z označenim vidnim stožcem in mejnima ravninama. Desno: Nastavitve lastnosti kamere.

Field of View *(sl. zorni kot)* določa širino pogleda kamere. Večja vrednost pomeni širši pogled na okolico, vendar lahko ob robovih povzroči popačenje slike. Pri ortografski projekciji pa se ta parameter spremeni v **Size** *(sl. velikost)*, ki določa fiksno velikost vidnega območja kamere. Nastavitvi **Near** *(sl. blizu)* in **Far** *(sl. daleč)* pri parametru **Clipping Planes** *(sl. mejni ravnini)* določata največjo in najmanjšo razdaljo od kamere, pri kateri so objekti vidni.

Poleg komponente Camera, kamera ponavadi vsebuje še komponento Audio Listener, ki smo jo omenili že v poglavju 4.4. Ta omogoča, da uporabnik sliši zvoke različnih zvočnih virov v sceni. V celotni sceni je lahko aktivna le ena komponenta poslušalca, ki določa točko, od koder uporabnik sliši zvoke.

V vsaki sceni je običajno ena kamera, ki določa pogled v igralnem načinu. Kot primarna je samodejno označena tista, ki ima dodano značko *(angl. Tag)* **MainCamera**. Če bi v sceno dodali več kamer, bi Unity uporabil le tisto s to oznako, ostale pa bi ignoriral, razen če bi zanje ročno nastavili prioriteto izrisa ali dodali lastno programsko logiko za preklapljanje med pogledi.

Pozor! Ob dodajanju nove kamere v sceno Unity nanjo samodejno doda tudi komponento Audio Listener. Ker je v sceni lahko aktiven le en poslušalec, bo Unity v konzoli javil opozorilo. Če želite v sceni obdržati več kamer, na vseh pomožnih kamerah to komponento obvezno odstranite ali izključite.

6.2.2 Sinhronizacija premikanja igralca in kamere

V raznih interaktivnih izkušnjah je zaželeno, da ob premikanju igralca premika tudi pogled kamere. To dosežemo tako, da njuna objekta povežemo s pomočjo hierarhije objektov. Če bo *kamera* otrok *igralca*, bo njena pozicija avtomatsko odvisna od pozicije *igralca*.

Primer 6.3: Premična kamera

V sceni iz primera 6.2 je že vključena kamera z imenom *Main Camera*. Da med njo in *igralcem* ustvarimo razmerje starš–otrok, njeno vrstico v hierarhiji povlečemo na vrstico *igralca* (podobno kot v primeru 3.2). Zdaj moramo le še izbrati položaj *kamere* glede na *igralca* tako, da bo *igralec* pred njo in bo njeno vidno polje približno zajemalo pogled, ki bi ga videl *igralec*.

Pomagamo si z orodjem za premikanje in s predogledom pogleda kamere, ki ga lahko vidimo v oknu igre ali v oknu Cameras[a]. Podrobnejša navodila za dostop do okna Cameras so na sliki 6.2 levo, desno pa je primer, kako naj bi izgledal končni pogled v igralnem načinu.

[a] Cameras *(sl. kamere)* je pomagalo v oknu scene, do katerega dostopamo preko menija za nastavitev pomagal (*angl. overlay menu*).

Slika 6.2: Levo je označen meni za nastavitev pomagal (*angl. overlay menu*), preko katerega dostopamo do okna Cameras, desno pa primer končnega pogleda kamere za *igralcem*.

6.3 Implementacija sistema točkovanja

Da bo mini igra zabavnejša, bomo v sceno dodali objekte, ki jih lahko *igralec* pobere in npr. s tem pridobi dodatno točko, podobno kot smo to lahko storili v izzivu 1. Za to bomo napisali nekaj lastne kode in bili pozorni na različne koncepte programiranja. V Unityju se koda lahko izvaja nenehno (npr. se izvede pri vsaki sličici, tako kot metoda `Update()`), ali pa se izvede le ob določenem dogodku (npr. ob trku dveh objektov). S pomočjo primerov bomo spoznali oba koncepta.

6.3.1 Vrtenje objekta

Ustvarili bomo novo datoteko s kodo za neprekinjeno vrtenje objekta okoli njegove osi.

Primer 6.4: Objekt, ki se vrti okoli svoje osi

Najprej izberemo objekt, ki ga lahko *igralec* pobere. Izberemo poljuben objekt v mapi `Prefabs/Collectibles`, ga postavimo v sceno ter po potrebi prilagodimo njegovo velikost in rotacijo. Na slikovnem materialu smo za ta primer izbrali kovanec, zato bomo v nadaljevanju uporabljali izraz *kovanec*. Za neprekinjeno vrtenje okoli navpične osi spreminjamo Y koordinato rotacije v komponenti transformacije. To izvedemo z novo skripto, ki jo – podobno kot v primeru 6.2 – ustvarimo in poimenujemo »Collectible«.

Označimo *kovanec* in v inspektorju iz projektnega okna nanj povlečemo datoteko s kodo. Nato jo z dvoklikom odpremo v urejevalniku kode. Prva vrstica določa, katere zunanje definicije program uporablja. Nujno mora navajati `using UnityEngine;`, ki omogoča uporabo pojmov, kot sta `Rigidbody` in `transform`. Sledi definicija razreda `Collectible`, znotraj katerega sta metodi `Start` (izvede se ob zagonu) in `Update` (izvaja se v vsaki sličici), obe zapisani znotraj zavitih oklepajev. V skripti znotraj metode `Update` dodamo vrstico:

```
transform.Rotate(0, 1, 0);
```

To bo povzročilo, da se bodo koordinate rotacije v komponenti transformacije povečale za dane vrednosti. V tem primeru se bo torej spremenila koordinata Y za 1, kar pomeni, da se bo objekt zavrtel za eno stopinjo v vsaki sličici. Zaradi hitrega menjavanja sličic bo videti, kot da se *kovanec* nenehno vrti. Shranimo skripto in si oglejmo igralni pogled.

Vrtenje je zdaj precej hitro. Če želimo hitrost za polovico zmanjšati, lahko v dodano vrstico namesto `transform.Rotate(0, 1, 0);` zapišemo:

```
transform.Rotate(0, 0.5f, 0);
```

Specifika jezika C# je, da moramo pri uporabi decimalnih števil na koncu vedno dopisati znak `f`.

Da bi določili pravo hitrost vrtenja, bi lahko na ta način večkrat poskušali in spreminjali vrednost v podani vrstici kode. Lahko pa se tega lotimo drugače. Nad blokom `Start` (ampak pod zavitim oklepajem) dodamo prazno vrstico in vanjo dopišemo:

```
public float rotationSpeed = 0.5f;
```

To ustvari spremenljivko, ki hrani izbrano hitrost vrtenja objekta, s privzeto vrednostjo 0,5. Če uporabimo ključno besedo `public` na začetku vrstice, lahko to vrednost kot parameter urejamo v oknu inspektor, sicer (če bi uporabili ključno besedo `private`) pa tega ne bi mogli. Sedaj vrstico znotraj bloka `Update` spremenimo v:

```
transform.Rotate(0, rotationSpeed, 0);
```

Ko kodo shranimo, se v inspektorju v komponenti skripte prikaže parameter **rotationSpeed**. Za spreminjanje hitrosti ni več potrebno odpirati datoteke s kodo, dovolj je že sprememba vrednosti parametra v inspektorju. Končna koda mora izgledati takole:

```
using UnityEngine;

public class Collectible : MonoBehaviour
{
    public float rotationSpeed = 0.5f;
    // Start is called once before the first execution of
    // Update after the MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(0, rotationSpeed, 0);
    }
}
```

6.3.2 Poberimo kovance

V prejšnjem poglavju smo dosegli, da se objekt v sceni vrti okoli svoje osi. Če bi v primeru 6.4 na koncu z *igralcem* trčili v *kovanec*, se s *kovancem* ne zgodi nič, *igralec* pa se od njega odbije, ga prevozi, ne more pa ga pobrati ali premakniti. Da bi fizikalni pogon zaznal trk med *kovancem* in *igralcem*, morata imeti oba komponento Collider.

Pri mini igri pa si ne želimo zgolj, da bi *igralec* trčil v *kovanec*, ampak da *kovanec* pri trku izgine (in trk ne vpliva na *igralčevo* premikanje). V poglavju 4.2.2 smo omenili, da Collider vsebuje potrditveno polje **Is Trigger**, ki omogoča, da objekt ob trku sproži določen dogodek, morebitne sile, ki bi bile prisotne pri trku, pa niso simulirane. S tem parametrom in nekaj kode bomo dosegli, da bo *kovanec* ob trku z *igralcem* izginil.

Primer 6.5: Kovanec, ki izginja

Kot smo zapisali v uvodu, moramo *kovancu* najprej vključiti možnost **Is Trigger** v komponenti Collider. Če preizkusimo igralni način, se z *igralcem* lahko premaknemo skozi *kovanec*, ampak se ne zgodi nič drugega. Da bo *kovanec* ob trku izginil, moramo ponovno spremeniti del kode v skripti.

Skripto `Collectible.cs` odpremo v urejevalniku kode in med metodama `Start` in `Update` ustvarimo nekaj praznih vrstic. Nato med njiju dodamo novo metodo s prvo vrstico:

```
private void OnTriggerEnter(Collider other)
```

Ta metoda se izvede vsakič, ko nek objekt s komponento Rigidbody pride v stik s Colliderjem objekta, ki vsebuje komponento s to skripto. Znotraj zavitih oklepajev te novo ustvarjene metode dodamo vrstico:

```
Destroy(gameObject);
```

Tu se `gameObject` nanaša na objekt, na katerega je dodana ta skripta (tj. *kovanec*). Ta del kode bo povzročil, da se objekt uniči in izgine.

Shranimo spremembe in preizkusimo igralni način. Sedaj *kovanec* izgine ob stiku z *igralcem*. Za lepši učinek lahko ob pobiranju dodamo vizualni efekt. V mapi `Prefabs/VFX` poiščemo ustrezen 3D učinek in ga vključimo v sceno. Ker ga bomo sprožili s kodo, ponovno odpremo urejevalnik kode.

Pod vrstico s spremenljivko `rotationSpeed`, dodamo vrstico:

```
public GameObject onCollectEffect;
```

S tem bomo pozneje v urejevalniku Unity lahko izbrali, kateri učinek naj se sproži. Pred tem pa spremenimo še kodo v metodi `OnTriggerEnter`. Po vrstici z `Destroy` dodamo novo vrstico:

```
Instantiate(onCollectEffect, transform.position,
    transform.rotation);
```

Ta pove, da naj se po uničenju sproži izbrani učinek na enaki lokaciji, kot je bila lokacija objekta, torej *kovanca*.

Če zaženemo igralni način, se še ne zgodi nič, saj nismo določili parametra **onCollectEffect**. To storimo tako, da v inspektorju za *kovanec* v komponenti skripte v vrstico za parameter **onCollectEffect**, kjer trenutno piše *None*, povlečemo izbrani učinek iz projektnega okna. V igralnem načinu ponovno preizkusimo in učinek bi se moral izvesti, ko se *igralec* dotakne *kovanca*.

Izziv 7: Pobirajte kovance

V sceno s kopiranjem pripravljenega objekta (npr. kovanca) dodajte še več takšnih objektov. Lahko si pomagate tudi tako, da iz objekta ustvarite Prefab Variant *(sl. varianta prefaba)*, kar storite na enak način, kot če bi ustvarili navaden prefab. Iz kovancev lahko sestavite zanimivo pot, ki jo mora opraviti igralec.

6.3.3 Zadnji popravki

Če smo končno sceno iz primera 6.5 večkrat preizkusili v igralnem načinu, smo lahko opazili, da kovanci izginejo tudi, če se dotaknejo katerega drugega objekta (npr. dela pohištva), ne samo *igralca*. To lahko preverimo, če katerega od kovancev že na začetku postavimo v stik s katerim od delov pohištva.

To lahko popravimo tako, da objekt uničimo samo, kadar pride v stik z *igralcem*, kar bomo storili z manjšim popravkom v kodi.

Primer 6.6: Kovanec naj izgine le, ko se dotakne igralca

Da preprečimo, da kovanci izginejo ob dotiku kateregakoli objekta v sceni, ponovno odpremo skripto `Collectible.cs` v urejevalniku kode. Vsebino metode `OnTriggerEnter` nato ovijemo z `if`-stavkom:

```
if (other.CompareTag("Player"))
```

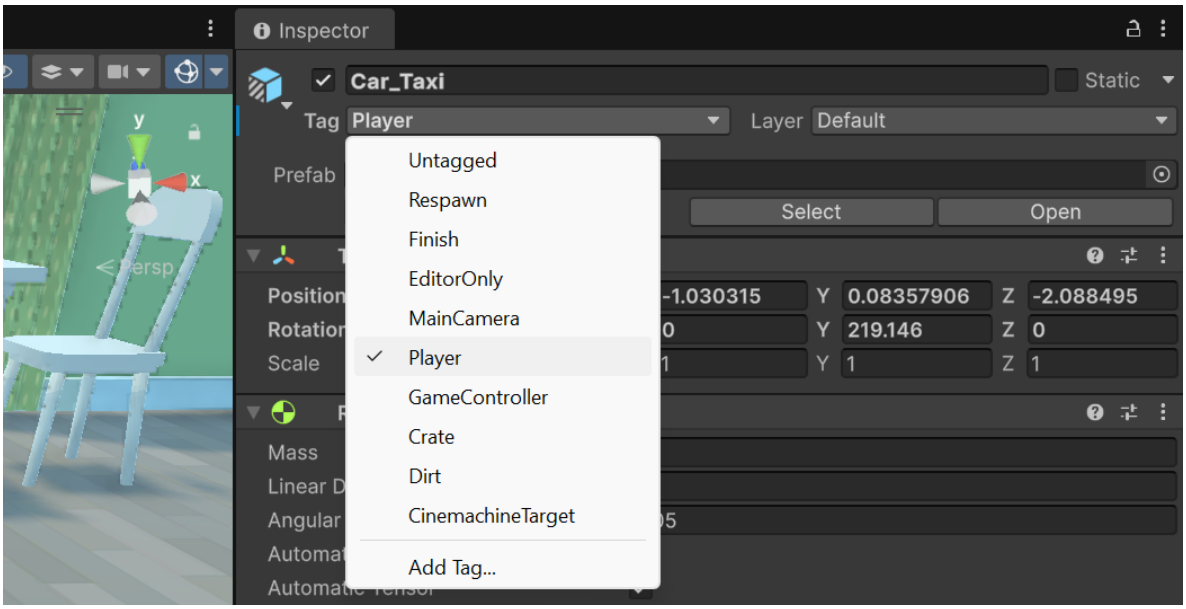
S tem smo določili, da bo *kovanec* izginil le ob trku z objektom, ki ima značko **Player**.

Da bo koda delovala, moramo v inspektorju *igralcu* dodati značko **Player** (na polju **Tag**, glej sliko 6.3). Končna koda mora izgledati tako:

```
using UnityEngine;

public class Collectible : MonoBehaviour
{
    public float rotationSpeed;
    public GameObject onCollectEffect;
    // Start is called once before the first execution of
        Update after the MonoBehaviour is created
    void Start()
    {

    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player")) {
            Destroy(gameObject);
            Instantiate(onCollectEffect, transform.
               position, transform.rotation);
        }
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(0, rotationSpeed, 0);
    }
}
```

Izziv 8: Prikažite število pobranih kovancev

V igrah je običajno zaželeno, da se v ozadju beleži število doseženih točk, naj bo to število pobranih kovancev ali pa katera koli druga vrednost pri različnih igrah. V pripravljeno igro dodajte logiko, ki beleži število pobranih kovancev, nato pa to število prikazujte kot besedilo med igro v igralnem pogledu.



Slika 6.3: *Igralcu* moramo na polju **Tag** določiti značko **Player**, da se lahko nanjo pozneje sklicujemo v kodi.

POGLAVJE 7

XR tehnologije in strojna oprema

Razvoj interaktivnih vsebin v okolju Unity ni omejen le na prikaz 3D scen na klasičnih zaslonih. Ena izmed pomembnejših smeri sodobnega razvoja digitalnih aplikacij je uporaba tehnologij razširjene resničnosti *(angl. Extended Reality, XR)*, ki uporabniku omogočajo neposredno in prostorsko izkušnjo digitalnih okolij.

V prejšnjih poglavjih smo spoznali osnovne principe ustvarjanja 3D scen v okolju Unity, delo z objekti, materiali, osvetlitvijo in kamerami ter načrtovanje osnovnih interakcij. Ti elementi so temelj za razvoj kompleksnejših interaktivnih aplikacij, med katere sodijo tudi rešitve razširjene resničnosti. Pri takšnih aplikacijah digitalna scena ni več namenjena zgolj prikazu na zaslonu, temveč postane del širšega interaktivnega okolja, v katerem uporabnik raziskuje prostor in sodeluje z virtualnimi objekti.

V tem poglavju bomo naredili korak naprej in pogledali, kako se razvoj interaktivnih vsebin v Unityju povezuje s tehnologijami razširjene resničnosti. Najprej bomo predstavili osnovne koncepte XR tehnologij ter njihov razvojni ekosistem, ki vključuje različne oblike povezovanja fizičnega in digitalnega sveta, poseben poudarek pa bomo namenili navidezni resničnosti, ki je ena izmed najpogostejših platform za razvoj potopitvenih *(angl. immersive)* interaktivnih izkušenj.

V nadaljevanju bomo spoznali tudi osnovne značilnosti naprav za prikaz navidezne resničnosti ter način, kako lahko aplikacije, razvite v okolju Unity, prilagodimo za njihovo uporabo. Na praktičnem primeru bomo pokazali, kako lahko obstoječo Unity sceno pripravimo za uporabo na VR očalih Meta Quest 3, ki so ena izmed najbolj razširjenih in dostopnih platform za razvoj in preizkušanje VR aplikacij. Tako bomo prešli iz klasičnega razvoja 3D aplikacij na zaslonu v razvoj interaktivnih okolij, kjer uporabnik postane neposredni udeleženec digitalnega prostora.

7.1 Tehnologije razširjene resničnosti

V sodobnem digitalnem okolju se vse pogosteje srečujemo s tehnologijami, ki na različne načine povezujejo fizični in digitalni svet. Takšne tehnologije združujemo pod skupnim izrazom razširjena resničnost *(angl. Extended Reality, XR)*. Gre za skupino pristopov, ki uporabniku omogočajo zaznavanje in raziskovanje digitalnih vsebin v prostoru

ter interakcijo z njimi na bolj naraven način, kot je to mogoče pri klasičnih računalniških aplikacijah.

XR tehnologije predstavljajo širši tehnološki in razvojni okvir, ki vključuje različne oblike digitalnih okolij ter načine njihovega prikaza in uporabe. Pri tem ne govorimo zgolj o posamezni napravi ali programski rešitvi, temveč o celotnem XR ekosistemu. Ta zajema razvojna orodja, digitalne vsebine, strojno opremo, uporabniške vmesnike ter uporabnike, ki z ustvarjenimi okolji interaktirajo. Pomembno vlogo v tem ekosistemu imajo tudi razvojna okolja, ki omogočajo ustvarjanje interaktivnih scen, definiranje logike interakcij ter prilagajanje aplikacij različnim napravam.

Oblike razširjene resničnosti

Pojem XR združuje več sorodnih tehnologij, ki se razlikujejo predvsem po načinu povezovanja digitalnega in fizičnega sveta. Med najpogosteje uporabljene oblike uvrščamo obogateno resničnost *(angl. Augmented Reality, AR)*, mešano resničnost *(angl. Mixed Reality, MR)* in navidezno resničnost *(angl. Virtual Reality, VR)*.

Obogatena resničnost (AR) digitalne elemente dodaja v realni svet. Uporabnik pri tem še vedno vidi svoje fizično okolje, ki je dopolnjeno z dodatnimi informacijami ali tridimenzionalnimi objekti. Digitalni elementi so prikazani preko slike resničnega sveta in ga tako vizualno obogatijo. Takšne rešitve se pogosto uporabljajo za prikaz dodatnih informacij, vizualizacijo procesov ali vodenje uporabnika skozi določene naloge.

Mešana resničnost (MR) predstavlja nadaljnji razvoj zgoraj opisanega koncepta. Pri MR digitalni objekti niso zgolj prikazani preko slike resničnega sveta, temveč so v prostor umeščeni tako, da upoštevajo obliko in položaj fizičnih predmetov. Naprave za mešano resničnost zato analizirajo okolje okoli uporabnika, prepoznajo površine in objekte ter v prostor postavijo digitalne elemente, ki se z njim smiselno povezujejo. Uporabnik lahko takšne objekte opazuje z različnih zornih kotov ter z njimi tudi neposredno interaktira.

Navidezna resničnost (VR) predstavlja najbolj potopitveno obliko XR tehnologij. Pri njej uporabnik vstopi v popolnoma digitalno ustvarjen tridimenzionalni prostor, ki nadomesti pogled na fizično okolje. Digitalna scena se v realnem času prilagaja uporabnikovim premikom, kar ustvarja občutek prisotnosti v navideznem svetu. Takšna okolja lahko predstavljajo simulacije resničnih prostorov ali pa popolnoma nova digitalna okolja, namenjena učenju, raziskovanju ali interaktivnim izkušnjam.

Pomembno je razumeti, da XR ne pomeni zgolj uporabe posebnih naprav, temveč predvsem drugačen način načrtovanja digitalnih okolij. Pri razvoju XR aplikacij je treba razmišljati o prostorski postavitvi objektov, načinu gibanja uporabnika, načrtovanju interakcij ter odzivanju sistema na uporabnikova dejanja. Igralni pogoni, kot je Unity, omogočajo implementacijo vseh teh elementov in tako predstavljajo eno izmed ključnih razvojnih platform za ustvarjanje XR aplikacij.

7.2 Naprave za prikaz in razvoj XR vsebin

Razvoj in uporaba aplikacij razširjene resničnosti sta tesno povezana z ustrezno strojno opremo. XR tehnologije temeljijo na usklajenem delovanju različnih naprav, ki omogočajo prikaz digitalnih vsebin, zaznavanje uporabnikovega gibanja ter interakcijo z navideznimi objekti. Zmogljivost, natančnost zaznavanja in kakovost prikaza so zato neposredno odvisni od uporabljene opreme ter njene združljivosti z izbranimi razvojnimi orodji. XR vsebine si je mogoče ogledati na več različnih tipih naprav, ki se med seboj razlikujejo predvsem po stopnji potopitve uporabnika v digitalno okolje, načinu interakcije ter tehnični zahtevnosti prikaza. Večina naprav uporablja kombinacijo zaslonov, kamer in različnih senzorjev, s katerimi zaznavajo položaj naprave, gibanje uporabnika ali strukturo prostora okoli njega. Razširjena resničnost ni vezana zgolj na eno vrsto očal ali naprave, temveč gre za širši nabor tehnologij in naprav, ki skupaj omogočajo nove načine prikaza digitalnih informacij, raziskovanja prostora in interakcije z navideznimi objekti.

7.2.1 Očala za razširjeno resničnost

Očala za razširjeno resničnost predstavljajo osrednji vmesnik med uporabnikom in digitalnim okoljem. To so naglavne naprave, ki omogočajo prikaz tridimenzionalnih vsebin neposredno v uporabnikovem vidnem polju ter hkrati zaznavajo gibanje glave in položaj uporabnika v prostoru. Glede na način prikaza digitalnih vsebin in stopnjo potopitve uporabnika jih delimo na tri osnovne skupine.

AR očala so naprave, ki digitalne informacije projicirajo neposredno v uporabnikovo vidno polje, pri čemer uporabnik ves čas vidi tudi realno okolje. Takšne naprave so pogosto namenjene prikazu dodatnih informacij ali navigacijskih elementov, pri čemer digitalni elementi dopolnjujejo resnični prostor.

Slika 7.1: Primeri MR naprav – Rokid Max, Microsoft Hololens, ThirdEye X2

MR očala (slika 7.1) omogočajo prikaz digitalnih objektov v resničnem okolju. Naprava s pomočjo kamer in globinskih senzorjev zaznava prostor okoli uporabnika, prepoznava površine in objekte ter v okolje umešča digitalne elemente. Takšni objekti so prostorsko usklajeni z realnim okoljem in se lahko odzivajo na spremembe v prostoru.

Interakcija z digitalnimi objekti pogosto poteka s pomočjo gest, sledenja pogledu ali glasovnih ukazov.

VR očala (slika 7.2) uporabnika popolnoma potopijo v digitalno ustvarjeno okolje. Naprava prek dveh zaslonov, po enega za vsako oko, prikazuje stereoskopsko sliko, kar ustvarja občutek globine in prisotnosti v navideznem prostoru. Poleg zaslonov VR očala vključujejo tudi senzorje gibanja, kamere ter pogosto ročne krmilnike, s katerimi uporabnik upravlja objekte v navideznem svetu. VR očala lahko glede na način delovanja razdelimo v dve skupini. Samostojna VR očala imajo vgrajen procesor, baterijo in vse potrebne senzorje, zato za delovanje ne potrebujejo zunanje naprave. Takšne naprave omogočajo večjo svobodo gibanja in enostavnejšo uporabo. Drugo skupino predstavljajo VR očala, ki za prikaz vsebin uporabljajo zunanji računalnik ali igralno konzolo. Pri takšnih sistemih izris tridimenzionalne scene poteka na zmogljivejšem računalniku, kar omogoča prikaz grafično zahtevnejših aplikacij.

Slika 7.2: Primeri VR naprav – PICO 4 Ultra, Valve Index, Meta Quest 3

Pri uporabi VR očal je treba upoštevati tudi praktične vidike. V kompletu so običajno priloženi ročni krmilniki, napajalni kabel in osnovna dodatna oprema. Pri uporabi v izobraževalnem ali laboratorijskem okolju pa se pogosto uporabljajo tudi dodatni higienski nastavki, čistila, kovčki za shranjevanje in polnilne postaje, ki omogočajo varnejšo in bolj organizirano uporabo večjega števila naprav.

7.2.2 Mobilne naprave

Mobilni telefoni in tablice so ena najpogosteje uporabljenih platform za prikaz obogatene resničnosti (slika 7.3). Kamera naprave zajema sliko resničnega okolja, na katero aplikacija nato dodaja digitalne elemente. Uporabnik skozi zaslon naprave vidi kombinacijo resničnega prostora in navideznih objektov.

Za pravilno delovanje AR aplikacij mora mobilna naprava omogočati natančno zaznavanje prostora, površin in gibanja naprave v realnem času. To omogočata razvojni platformi ARCore (Android) in ARKit (Apple), ki skrbita za sledenje gibanja, zaznavanje površin ter pravilno umeščanje digitalnih objektov v okolje. Če naprava ene od teh plat-

form ne podpira, AR aplikacij običajno ni mogoče zagnati ali pa delujejo z omejenimi funkcionalnostmi.

Slika 7.3: Primeri AR naprav - mobilni telefon (Samsung S25), tablica (Apple iPad), AR očala (Ray-Ban Meta Smart Glasses)

7.2.3 Razvojna oprema

Pomembno je razumeti razliko med uporabo in razvojem XR vsebin. Medtem ko lahko končni uporabniki uporabljajo aplikacije na samostojnih napravah, razvoj in testiranje XR aplikacij zahtevata zmogljivejšo strojno opremo ter ustrezno razvojno okolje. Pri razvoju aplikacij v programu Unity je pogosto potrebno aplikacijo testirati neposredno na ciljni napravi. Pri nekaterih platformah lahko razvijalec aplikacijo preizkuša v realnem času, pri drugih pa je treba projekt najprej izvoziti in ga ročno namestiti na napravo. Zaradi boljše podpore razvojnim orodjem in neposrednega testiranja je za razvoj XR aplikacij v okolju Unity pogosto priporočljiva uporaba računalnikov z operacijskim sistemom Windows, ki omogočajo hitro in učinkovito testiranje aplikacij na VR napravah.

7.3 Meta Quest 3

Meta Quest 3 spada v kategorijo samostojnih (angl. *standalone*) VR naprav, kar pomeni, da za delovanje ne potrebuje povezave z računalnikom, saj so procesor, baterija in shramba vgrajeni neposredno v naglavni komplet. Meta Quest 3 so napredna očala za navidezno in mešano resničnost, ki jih je razvilo podjetje Meta in so trenutno ena cenovno dostopnejših naprav tega tipa.

Delovanje temelji na operacijskem sistemu Meta Horizon OS, ki je osnovan na Androidu. Vsebujejo dve palačinkasti leči (angl. pancake lens), vsaka z ločljivostjo 2064x2208 slikovnih točk, kar je 30% več v primerjavi s prejšnjo različico, Meta Quest 2. Omogočajo $110°$ horizontalnega in $96°$ vertikalnega zornega kota ter vrsto nastavitev za udobno nošenje: nastavitev medočesne razdalje med lečama, nastavitev globine maske in nastavitev pasov za glavo [2].

K očalom spadata dva krmilnika, ki sta namenjena spremljanju gibanja rok uporabnika. Poleg uporabe s krmilniki očala podpirajo tudi upravljanje z gestami. Na sprednji strani imajo očala več kamer, ki poleg zaznavanja rok in krmilnikov omogočajo tudi pogled skozi (*angl. passthrough*), pri katerem vidimo resnični svet okoli sebe. S tem je podprta možnost mešane resničnosti.

7.3.1 Prilagoditve in urejanje

Naglavna očala so sestavljena iz plastičnega ohišja z naglavnim trakom ter težjim sprednjim delom z lečami in procesorjem. Zaradi teže v sprednjem delu je priporočljivo, da za udobno nošenje naprave pravilno nastavimo naglavni trak. Ta omogoča nastavitev obsega s krakoma na zadnji strani, ki ju lahko povlečemo narazen. Višino, na kateri očala sedijo na obrazu, lahko nastavimo s trakom na vrhu glave.

Obrazna maska naprave je plastična, del, ki se dotika obraza, pa je oblečen v tekstilno prevleko. Pogosto se uporabniki odločijo za dodatne silikonske prevleke, ki omogočajo lažje čiščenje in vzdrževanje naprave. Nastavljiva je tudi globina obrazne maske, ki jo spreminjamo z mehaničnima gumboma na notranji strani, s katerima izberemo eno izmed štirih nastavitev. Ta možnost je še posebej uporabna za uporabnike, ki nosijo korekcijska očala, saj lahko napravo lažje uporabljajo skupaj s korekcijskimi očali. Lečam naprave namreč ni mogoče spreminjati dioptrije, lahko pa uporabniki kupijo leče s svojo dioptrijo in jih zamenjajo.

Za boljšo jasnost slike naprava omogoča tudi nastavitev medočesne razdalje med lečama, ki jo spreminjamo s kolescem na spodnjem levem delu očal. Možne nastavitve zajemajo razdaljo med 53 mm in 75 mm.

7.3.2 Prva uporaba naprave

Za uporabo naprave Meta Quest 3 potrebujemo svoj uporabniški račun in aplikacijo Meta Horizon na pametnem telefonu, ki jo lahko brezplačno prenesemo iz trgovine za mobilne aplikacije (npr. Trgovina Play ali App Store, odvisno od operacijskega sistema).

V aplikacijo se lahko prijavimo s svojim Facebook računom ali pa si ustvarimo račun Meta Horizon. Če izberemo prvo možnost, se bodo aplikacije v VR očalih povezale z osebnim Facebook profilom, prijatelji, objavami, komentarji ipd. Račun Meta Horizon pa je ločen sistem za dostop do storitev podjetja Meta in ni vezan na družbeno omrežje. Za registracijo zadostuje elektronski naslov, s katerim v postopku ustvarjanja računa potrdimo svojo identiteto.

Aplikacija Meta Horizon omogoča pregled nad vsemi napravami, s katerimi smo povezani s svojim računom, poleg tega pa tudi neposredno upravljanje določenih nastavitev v VR napravah. To vključuje izbiro in nameščanje novih aplikacij ter deljenje zaslona VR naprave v mobilno aplikacijo.

Primer 7.1: Povezovanje Meta Quest 3 z mobilno aplikacijo

Ko zaženemo očala, najprej izberemo ustrezno Wi-Fi omrežje, s katerim je povezan tudi naš pametni telefon. Občasno očala v tem koraku začnejo nameščati morebitne posodobitve, kar moramo počakati, nato pa lahko nadaljujemo s povezovanjem.

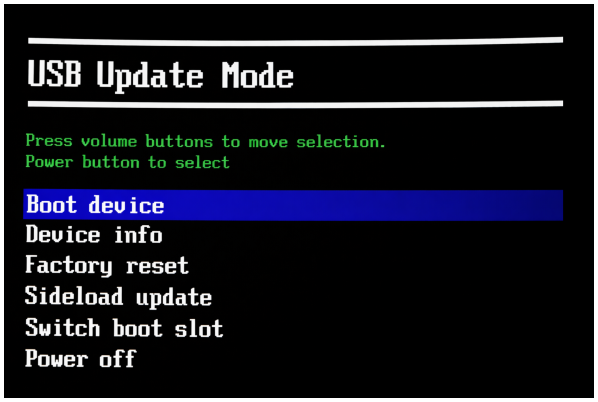
V očalih se prikaže zaslon z identifikacijsko številčno kodo, ki bomo morali vnesti v aplikacijo. Na telefonu v aplikaciji poiščemo zavihek ≡ *Devices (sl. naprave)* in izberemo možnost Pair headset *(sl. poveži se z napravo)*. Odpre se seznam različnih naprav, med katerimi izberemo Meta Quest 3. Aplikacija nas nato vodi skozi postopek povezovanja, pri katerem zazna napravo v bližini in ponudi polje, kamor vpišemo identifikacijsko številčno kodo iz VR očal. Ko se postopek zaključi, je vaš račun povezan z napravo, v kateri si lahko ogledamo uvodne videe in se s pomočjo vodičev naučimo osnovnih načinov uporabe.

7.3.3 Ponastavitev na tovarniške nastavitve

Kadar v napravi pride do večjih napak v delovanju (težave s krmilniki, sliko), je priporočljivo, da očala ponastavimo na tovarniške nastavitve. To lahko storimo na dva načina.

Prvi način je, da napravo ugasnemo, nato pa hkrati držimo gumba za vklop in zmanjšanje glasnosti, dokler se naprava ne vklopi. Pojavi se enostaven meni (podoben kot na sliki 7.4), po katerem se premikamo z gumbom za glasnost, izbiramo pa z gumbom za vklop. Izberemo možnost Factory reset *(sl. ponastavitev na tovarniške nastavitve)*, nato pa ponovno potrdimo odločitev z izbiro Yes. Očala se ponovno zaženejo in lahko ponovimo postopek prijave.

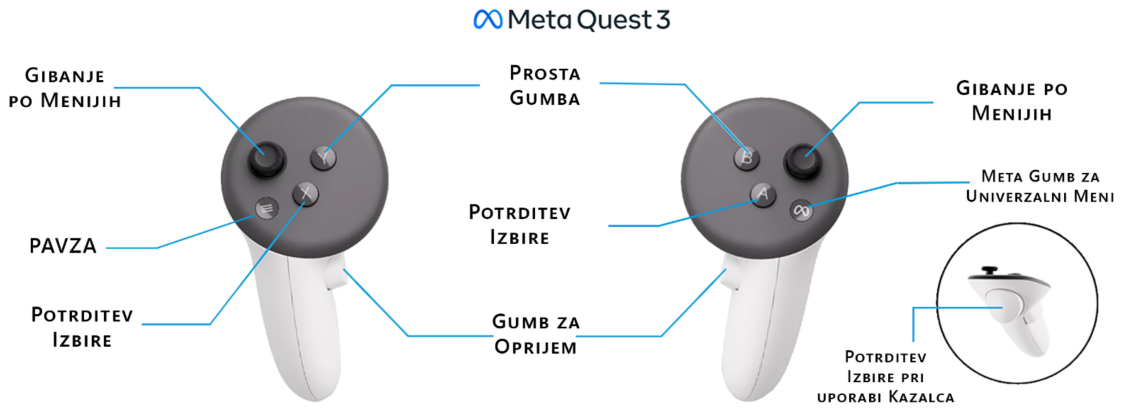
Drugi način je uporaba nastavitev znotraj mobilne aplikacije, kjer v zavihku ≡ *Devices* poiščemo svojo napravo, nato izberemo možnost Headset settings *(sl. nastavitve)* in Factory reset pod zavihkom Advanced Settings *(sl. napredne nastavitve)*.

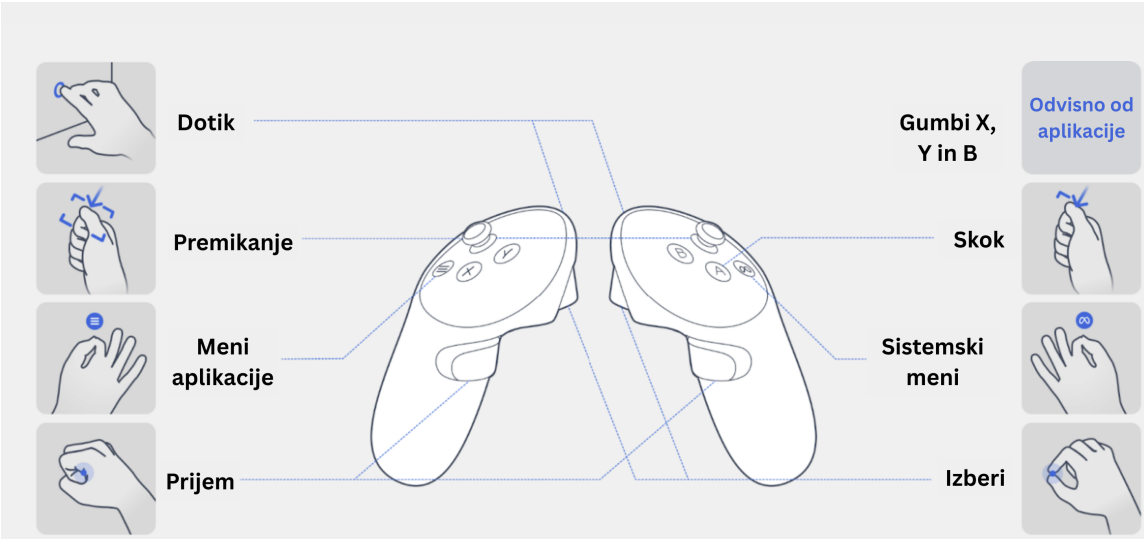


Slika 7.4: Zagonski meni, ki se v VR očalih prikaže, če pri prižiganju držimo gumba za vklop in zmanjšanje glasnosti.

7.3.4 Možnosti uporabe s krmilniki in gestami

V napravi Meta Quest 3 so interakcije možne z dvema krmilnikoma, ki za napajanje uporabljata baterije tipa AA. . Vsakega držimo v eni roki, vključujeta pa tudi paščke, ki jih namestimo okoli zapestij. Na obeh so simetrično razporejeni gumbi, med katerimi imajo nekateri določene funkcije, drugi pa so prosti.





Slika 7.5: Krmilnika Meta Quest 3 z gumbi (zgoraj) ter upravljanje z gestami, ki jih zaznava naprava (spodaj). Vir (zgoraj): prirejeno po [3]. Vir (spodaj): prirejeno po [1].

Kot je označeno na sliki 7.5 zgoraj, imata oba krmilnika na sprednji strani potrditveni gumb. Ta ima glavno vlogo, saj omogoča izbiranje v menijih in deluje podobno kot levi klik računalniške miške. Pritiskamo ga s kazalcem. Posebnost sta gumba na »notranji« strani krmilnikov (pritiskamo ju s sredincem ali prstancem), ki ju uporabljamo, ko želimo

v sceni prijeti kak objekt. Na vsakem krmilniku je tudi krmilni gumb, s katerim v različnih interaktivnih izkušnjah uravnavamo premikanje in obračanje v prostoru.

Gumb z logotipom podjetja Meta oziroma Meta gumb, ki ima vdrto obliko, uporabljamo za odpiranje glavnega menija v napravi in za ponastavitev pogleda. Slednje je pomembno, saj VR očala spremljajo premikanje glave, kar lahko povzroči, da odprta okna in menije v napravi s premikanjem »izgubimo«. Če Meta gumb držimo približno tri sekunde, se pogled ponastavi in okna se skupaj z meniji prestavijo nazaj v sredino našega pogleda.

Osnovne ukaze lahko dosežemo tudi z uporabo gest z rokami, ki jih naprava zaznava s pomočjo kamer in senzorjev na sprednji strani, kar je prikazano na sliki 7.5 spodaj.

7.3.5 Različne uporabne nastavitve

V očalih glavno orodno vrstico odpremo z Meta gumbom. Ta omogoča prikaz in hiter dostop do glavnih funkcionalnosti naprave, kot so prikaz stanja baterije, časa, dostop do hitrih nastavitev in knjižnice aplikacij.

Okno s hitrimi nastavitvami odpremo z izbiro polja, ki prikazuje stanje baterije in trenutni čas. Rezultat je odprto okno, v katerem lahko enostavno nastavljamo glasnost zvoka, svetlost pogleda, omrežne in Bluetooth nastavitve ter nekaj drugih možnosti.

Nastavitev **Boundary** *(sl. varnostna meja)* skrbi za umestitev očal v prostoru, kjer se uporabnik nahaja. Znotraj te nastavitve lahko določimo prostorsko ali stacionarno mejo, ki napravi pomaga določiti natančno oddaljenost od tal in prostor, po katerem se je varno gibati. Ker pri VR pogledu hitro izgubimo občutek, kje v prostoru smo, je **Boundary** varovalo, ki izklopi VR pogled, če se premaknemo izven varnega območja.

Nastavitev **Cast** *(sl. deljenje zaslona)* omogoča deljenje vsebine, ki jo vidimo na očalih, na več različnih načinov. Deljenje preko spletnega brskalnika omogoča, da na napravi, povezani v isto internetno omrežje, ob vnostu povezave `horizon.meta.com/casting` v brskalniku vidimo celoten pogled, ki ga vidi uporabnik očal. Druga možnost je deljenje v mobilno aplikacijo, preko katere smo povezani z napravo.

Med hitrimi nastavitvami najdemo še dostop do aplikacije Link, ki je natančneje opisana v naslednjem poglavju.

7.3.6 Link in Air Link

Aplikacija Meta Horizon Link omogoča povezavo očal Meta Quest 3 z namiznim računalnikom — bodisi prek kabla USB-C (vsaj USB 3.0) bodisi brezžično prek omrežja Wi-Fi, kar aplikacija imenuje Air Link. Povezava je koristna predvsem pri razvoju VR aplikacij, saj omogoča neposredno preizkušanje na napravi.

Za uporabo potrebujemo računalnik z operacijskim sistemom Windows 10 ali Windows 11 in z ustrezno grafično kartico[1] ter nameščeno aplikacijo Meta Horizon Link, v katero se prijavimo s svojim računom Meta.

[1] Sistemske zahteve so dostopne na `https://www.meta.com/help/quest/140991407990979/`

Povezava Link (kabel)

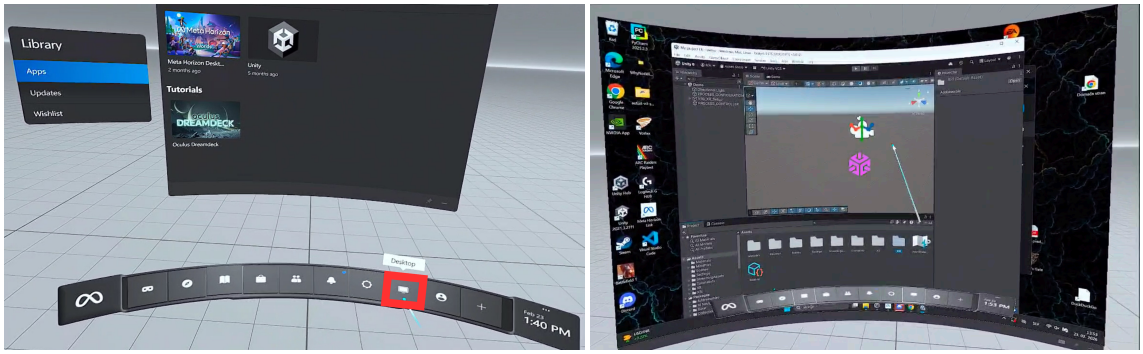
1. Odpremo aplikacijo Meta Horizon Link na računalniku in se prijavimo.
2. Očala povežemo z računalnikom prek kabla USB-C.
3. V očalih pritisnemo gumb Meta in odpremo Hitre nastavitve s pritiskom na ikono ure.
4. Izberemo možnost Link in nato računalnik, s katerim smo povezani.
5. Pritisnemo Launch — v očalih se prikaže okolje Link.

Povezava Air Link (Wi-Fi)

Brezžična povezava zahteva nekaj dodatnih korakov in je načeloma manj stabilna od žične, a odpravi potrebo po uporabi kabla.

1. Odpremo Meta Horizon Link na računalniku, se prijavimo in preverimo, da sta računalnik in očala v istem omrežju Wi-Fi.
2. V očalih pritisnemo gumb Meta in odpremo Hitre nastavitve.
3. Izberemo Link in nato Use Air Link.
4. Izberemo računalnik s seznama in pritisnemo Pair.
5. Preverimo ujemanje prikazane kode v očalih in na računalniku ter potrdimo s Confirm.
6. Pritisnemo Launch — v očalih se prikaže okolje Link.

Ko je povezava vzpostavljena, se v očalih prikaže posebna orodna vrstica (slika 7.6). Z gumbom Desktop preklopimo na pogled namizja računalnika, ki ga lahko upravljamo s krmilniki.



Slika 7.6: S pomočjo aplikacije Meta Horizon Link lahko z VR očali v aplikaciji v orodni vrstici (označeno rdeče na sliki levo) dostopamo do pogleda in upravljanja namizja računalnika (desno).

POGLAVJE 8

Razvoj izkušenj z uporabo VR Builderja

VR Builder je zmogljivo orodje za vizualno ustvarjanje vsebin, ki razvijalcem omogoča gradnjo kompleksnih VR scenarijev brez poglobljenega znanja programiranja [4]. Njegova glavna prednost je intuitiven grafični vmesnik, ki deluje po principu »povleci in spusti«. Z njegovo pomočjo gradimo logično strukturo interaktivne izkušnje, ki temelji na zaporedju različnih stanj. To pomeni, da igro ali simulacijo razdelimo na logično zaporedje dogodkov, kjer se naslednji korak sproži šele, ko uporabnik izpolni določen pogoj.

8.1 Namestitev VR Builderja v projekt

Vtičnik VR Builder ni vključen v osnovno namestitev urejevalnika Unity, zato ga moramo ročno uvoziti v projekte, kjer ga želimo uporabljati. Njegove glavne funkcije so odprtokodne in brezplačne, za več funkcionalnosti pa moramo kupiti različico VR Builder Pro, ki trenutno stane približno 140 EUR[1].

Plačljivo različico lahko pridobimo preko Unity Asset Store, brezplačno osnovno različico pa lahko prenesemo s strani Github na povezavi `https://github.com/MindPort-GmbH/VR-Builder/releases`. Tam prenesemo datoteko s končnico `.unitypackage`, ki jo nato uvozimo v svoj projekt.

Ko je vtičnik vključen v projekt in sceno, se v hierarhiji pojavita objekta *PROCESS CONFIGURATION* in *PROCESS CONTROLLER*, ki skrbita za pravilno upravljanje in izvajanje procesa, določenega z VR Builderjem. Med okna urejevalnika se doda tudi okno urejevalnik procesa (*angl. Process Editor*), v katerem lahko vizualno urejamo zaporedje korakov svojega scenarija.

[1] Natančnejša navodila za namestitev obeh različic najdete na povezavi `https://www.mindport.co/vr-builder/get-vr-builder`

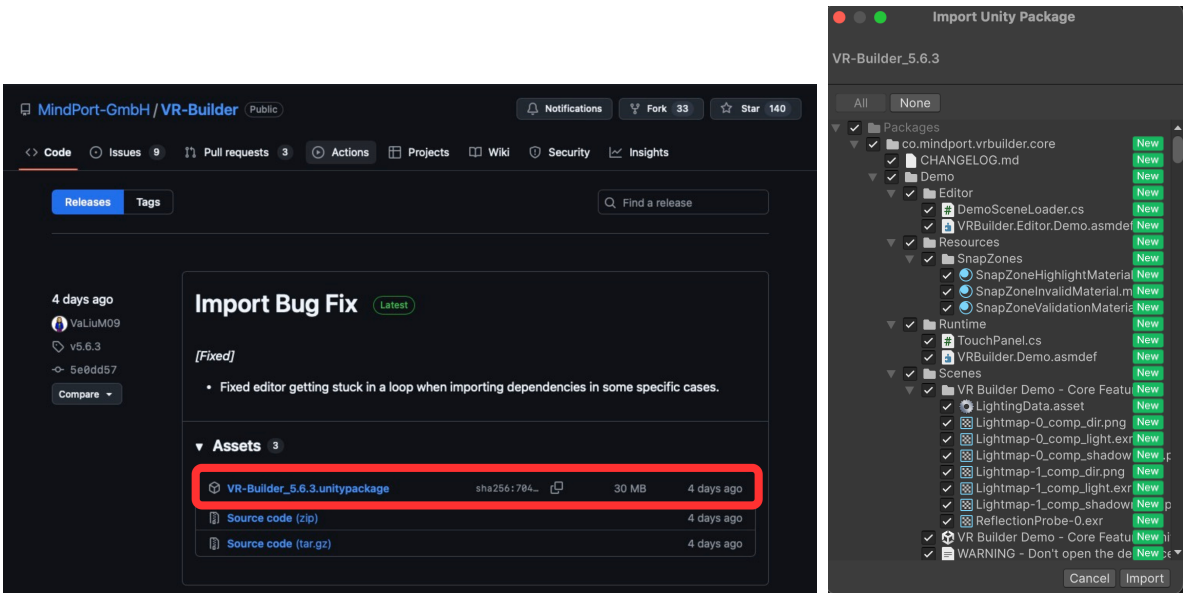
Primer 8.1: Dodajmo VR Builder v svoj projekt

Da lahko VR Builder uvozimo v svoj projekt, mora ta imeti 3D predlogo (primeren je vsak projekt, ki ni eksplicitno ustvarjen za 2D). Uporabimo lahko projekt iz prvega dela učbenika ali pa ustvarimo novega.

Odprtokodne datoteke vtičnika VR Builder najdemo na povezavi `https://github.com/MindPort-GmbH/VR-Builder/releases`, kjer avtorji sproti objavljajo nove različice. Najnovejša različica se vedno nahaja na vrhu strani, podobno kot je prikazano na sliki 8.1 levo. V njenem polju pod razdelkom **Assets** najdemo datoteko s končnico `.unitypackage`, ki jo prenesemo s klikom nanjo.

Vtičnik uvozimo v projekt tako, da v Unity urejevalniku projekta v orodni vrstici urejevalnika izberemo možnost ≡ *Assets* ▶ *Import Package* ▶ *Custom Package*. Odpre se okno za izbiro datoteke, ki jo želimo uvoziti, zato poiščemo ravnokar preneseno datoteko vtičnika. Odpre se okno kot na sliki 8.1 desno, ki prikazuje strukturo datotek vtičnika. Za nadaljevanje izberemo gumb Import *(sl. uvozi)* spodaj desno.

Proces uvažanja lahko traja nekaj minut, na koncu pa se odpre pojavno okno, v katerem izberemo nekatere privzete nastavitve vtičnika. Nato se med okna urejevalnika doda (ali pa se pojavi v ločenem oknu aplikacije) okno urejevalnika procesa (*angl. Process Editor*). Priporočamo, da to okno premaknete v osrednji del urejevalnika, kjer so že okna okno scene, okno igre in paketni upravitelj.



Slika 8.1: Prenos zadnje različice vtičnika VR Builder (levo) ter potrditev uvoza v Unityju z izbiro gumba Import (desno).

8.2 Konfiguracija vtičnika v sceni

Ko vtičnik vključimo v projekt, ta še ni samodejno vključen v posamezno sceno. Za upravljanje in izvajanje procesov VR Builderja sta odgovorna objekta *PROCESS CONFIGURATION* in *PROCESS CONTROLLER*, ki morata biti prisotna v sceni.

Da lahko dogajanje v sceni nadziramo z VR Builderjem, ga moramo dodati s pomočjo čarovnika za konfiguracijo scene. Do njega dostopamo v orodni vrstici prek ≡ *Tools* ▶ *VR Builder* ▶ *Scene Setup Wizard*, kjer lahko izbiramo med dodajanjem vtičnika v obstoječo sceno ali ustvarjanjem nove scene.

Ko vtičnik vključimo v sceno, je samodejno dodan tudi objekt *igralca*, ki že vsebuje vse potrebno za zagon aplikacije na VR očalih in premikanje po sceni s krmilniki ter gibanjem glave. Objekt ima hierarhično sestavo, njegov glavni otrok pa je *XR Rig*, ki nadzoruje gibanje in funkcije krmilnikov. Kot je prikazano na sliki 8.2 desno, gibanje urejamo s pomočjo objektov, ki so del skupine *Locomotion*. Če je posamezen objekt onemogočen, logika premikanja, ki jo omogoča, v sceni ni aktivna.

Primer 8.2: Dodajanje vtičnika v trenutno sceno in poganjanje na očalih

V tem primeru bomo vtičnik dodali svoji trenutni sceni, v kateri morda že imamo nekaj svojih objektov. Sceno odpremo in v orodni vrstici urejevalnika izberemo možnost ≡ *Tools* ▶ *VR Builder* ▶ *Scene Setup Wizard*.

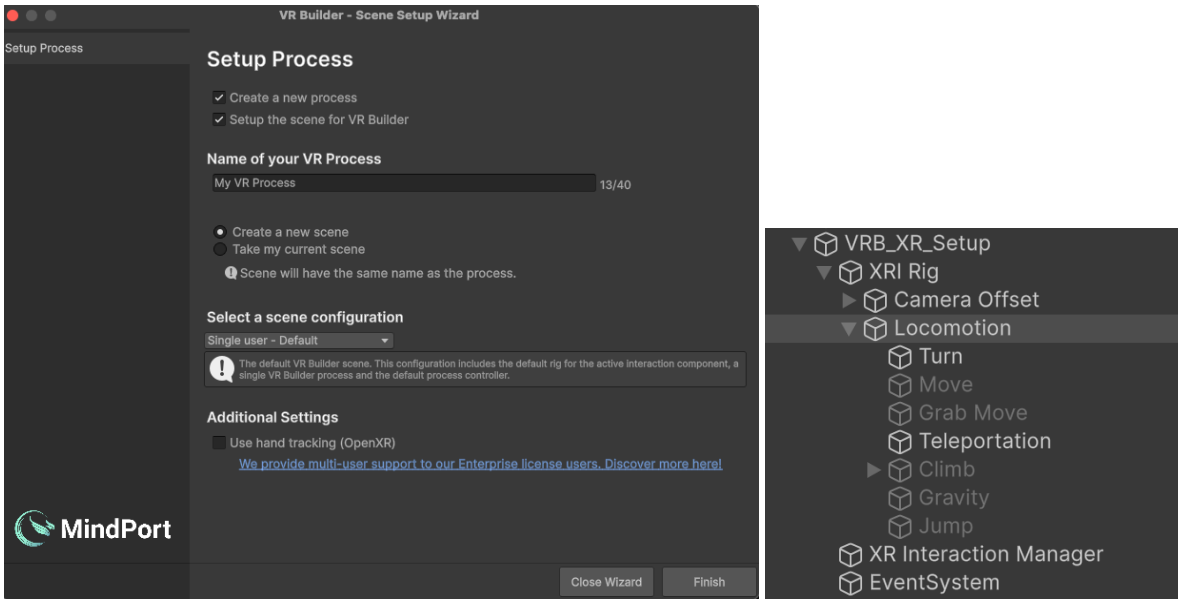
Odpre se pojavno okno, podobno kot na sliki 8.2, v katerem večino nastavitev pustimo prednastavljenih. Po potrebi spremenimo ime svojega VR procesa (angl. Name of your VR Process)[a], nujno pa označimo možnost **Take my current scene** *(sl. uporabi trenutno sceno)*. Na koncu izbiro potrdimo z gumbom Finish *(sl. končaj)* desno spodaj.

Ko je postopek končan, se v hierarhiji pojavijo objekti *PROCESS CONFIGURATION*, *PROCESS CONTROLLER* in *VRB_XR_Setup*[b]. Slednji predstavlja objekt *igralca* v sceni, kar omogoča, da lahko igro poganjamo na VR očalih in s sceno interagiramo s krmilniki.

Da igro poženemo na VR očalih, moramo imeti vzpostavljeno povezavo med očali in namizno aplikacijo Meta Horizon Link. V očalih je takrat prikazana posebna orodna vrstica, na kateri lahko izberemo možnost Desktop *(sl. namizje)*, s čimer v očalih vidimo namizje računalnika in z miško upravljamo preko krmilnikov. Neposredno iz očal (ali preko računalnika) zaženemo igralni način (poskrbimo tudi, da je odprto okno igre) in v očalih se prikaže svet, kot ga vidi igralec v sceni. Če želimo omogočiti več načinov premikanja, moramo v objektu *Locomotion* omogočiti ustrezne objekte.

[a] Vsak ustvarjen proces mora imeti unikatno ime. Na to nas bo opozoril rumen trikotnik poleg polja za vnos imena.

[b] Objekt *VRB_XR_Setup* ima lahko drugačno ime, odvisno od različice vtičnika.



Slika 8.2: Čarovnik za konfiguracijo scene (levo) in dodani objekti v sceni, vključno z igralcem in elementi za gibanje (desno).

8.3 Urejevalnik procesov

Urejevalnik procesov je osrednje delovno okolje vtičnika, kjer snovalec določa potek navidezne izkušnje. Proces razumemo kot usmerjen graf, sestavljen iz posameznih korakov (*angl. steps*), ki so med seboj povezani s prehodi.

Vsak korak v procesu predstavlja specifično stanje v igri (npr. »Vzemi ključ«). Da se izvajanje premakne k naslednjemu koraku, mora biti izpolnjen določen pogoj (npr. ključ mora biti v igralčevi roki). Takšna struktura omogoča ustvarjanje *t. i.* postopkovnih iger, kjer dogajanje sledi determinističnemu zaporedju akcij, kar je izjemno uporabno predvsem pri izobraževalnih aplikacijah in industrijskih simulacijah.

8.3.1 Sestava in urejanje koraka

Osnovni gradnik procesa je korak. Vsak korak je sestavljen iz dveh ključnih delov:

- Vedenja (*angl. behaviors*): To so akcije, ki se zgodijo, ko se korak aktivira (npr. predvaja se zvočni posnetek ali v sceni se prižge luč).
- Prehodi (*angl. transitions*): Določajo nabor pogojev, ki morajo biti izpolnjeni, da proces preide na naslednji korak.

Vsi koraki procesa so prikazani v osrednjem delu urejevalnika procesa, ki deluje kot neskončna površina, na kateri lahko korake poljubno premikamo. Postavitev korakov na tej površini ne vpliva na proces. Nov korak ustvarimo tako, da na praznem območju v urejevalniku procesa z desnim klikom izberemo ≡ *New* ▶ *Step*. Posamezen korak urejamo tako, da nanj kliknemo, kar odpre ločeno aplikacijsko okno inspektor korakov (*angl. Step Inspector*), ki ga lahko s potegom dodamo med okna urejevalnika. Priporo-

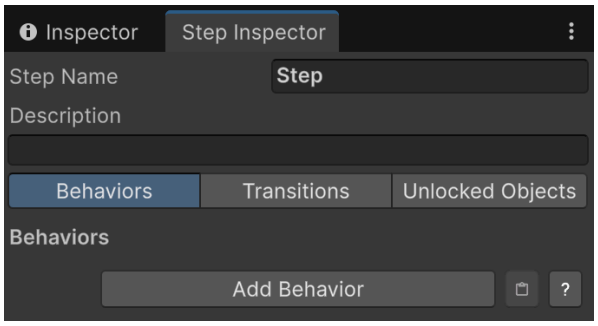
čamo, da dodamo v isti predel, kjer je tudi inspektor objektov.

V inspektorju korakov sta na vrhu polji **Step Name** *(sl. ime koraka)* in **Description** *(sl. opis)*, pod njima pa je vsebina razdeljena na tri glavne zavihke: **Behaviors** *(sl. vedenja)*, **Transitions** *(sl. prehodi)* in **Unlocked Objects** *(sl. omogočeni objekti)*.

Primer 8.3: Ustvarjanje koraka

V urejevalniku procesa je trenutno le korak z imenom »Start« (gre za poseben tip koraka, več o tem sledi v naslednjem poglavju). Najprej bomo v proces dodali lasten korak.

Na praznem območju urejevalnika procesa z desnim klikom izberemo ≡ *New* ▶ *Step*. Pojavi se pravokotnik, ki ima trenutno ime »Step«. Če ga označimo, se odpre inspektor korakov, ki trenutno prikazuje vsebino tega koraka, podobno kot na sliki 8.3. Ime koraka spremenimo v »Poberi ključ« (vsebino koraka bomo dopolnili v nadaljevanju).



Slika 8.3: Inspektor korakov je okno za urejanje posameznega koraka v procesu VR Builderja. Poleg imena in opisa vključuje tri razdelke: **Behaviors** *(sl. vedenja)*, **Transitions** *(sl. prehodi)* in **Unlocked Objects** *(sl. omogočeni objekti)*.

8.4 Pregled vedenj in prehodov

Vsak korak v procesu VR Builderja je določen z naborom različnih vedenj in prehodov. Vedenja so akcije, ki se zgodijo, ko se korak začne. Nato korak ostane aktiven, dokler niso izpolnjeni pogoji prehodov za napredovanje v naslednji korak.

8.4.1 Prehodi

Na podlagi prehodov v procesu VR Builder objektom samodejno doda posebne komponente. Te omogočajo, da lahko objekt v igralnem načinu s krmilnikom primemo, ga prestavimo na določeno vnaprej določeno mesto ali na drug način interagiramo z njim.

Vsak korak ima lahko več različnih prehodov, vsak prehod pa je sestavljen iz nabora pogojev. Da se nek prehod zgodi, morajo biti izpolnjeni vsi njegovi pogoji. Iz koraka izstopimo takoj, ko se zgodi eden izmed njegovih prehodov. To omogoča, da določimo

več izključujočih se potekov igre.

Pogoji, ki jih lahko uporabimo v prehodih, so razdeljeni v štiri skupine: **Environment** *(sl. okolje)*, **Interaction** *(sl. interakcije)*, **Locomotion** *(sl. premikanje)* in **Utility** *(sl. pripomočki)*. Ko pogoje dodamo koraku, jim moramo določiti več različnih parametrov, ki povedo, na katere objekte se pogoj nanaša.

8.4.1.1 Pogoj Grab Object

Pogoj Grab Object (*sl. poberi objekt*) iz skupine **Interaction** določa, da moramo za izpolnitev s krmilnikom pobrati izbran objekt v sceni. Ko v proces dodamo ta pogoj in mu določimo objekt, se objektu doda komponenta Grabbable Property (sl. lastnost, da objekt lahko primemo)[2]. Ta komponenta omogoči, da lahko s tem objektom v sceni interagiramo tako, da ga primemo s krmilniki. Ta lastnost je omogočena šele, ko se korak s tem pogojem aktivira.

Pogoj Grab Object (glej sliko 8.4 levo) potrebuje določen en parameter. To mora biti objekt, ki ga želimo prijeti, določimo pa ga tako, da ga s potegom spustimo na polje parametra.

Primer 8.4: Soba pobega: primimo ključ

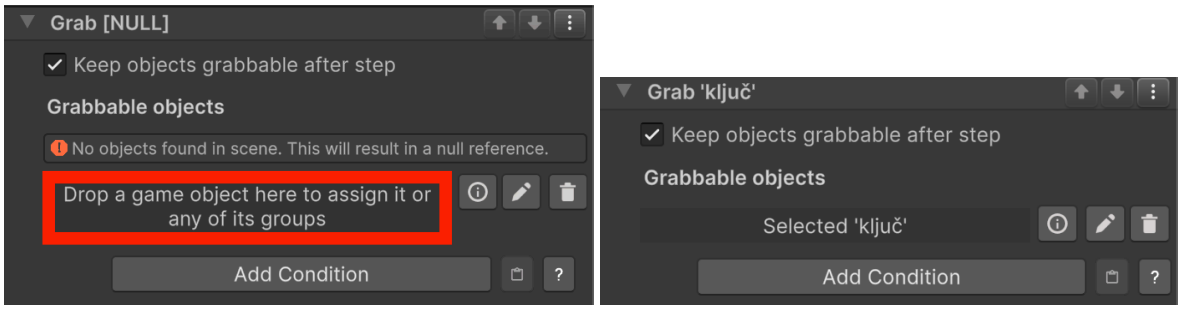
V tem in naslednjih primerih bomo na primeru sobe pobega prikazali vse glavne funkcionalnosti VR Builderja, različne korake in njihovo vsebino. Sobo pobega bomo sestavili tako, da bomo v treh sobah reševali izzive. V prvi sobi bomo morali s ključem odpreti vrata.

Najprej moramo pripraviti sobo pobega, za kar v sceni potrebujemo objekt prostora (sobe), vrata in ključ. Priporočamo, da te elemente poiščete v trgovini Unity Asset Store (natančnejša navodila za uvažanje najdete v poglavju 5.1). Za sestavo prostorov lahko uporabite paket »Interiors A«, brezplačen paket z vrati je »Free Wood Door Pack«, za ključ pa lahko uporabite objekt iz paketa »Rust Key«.

Sceno pripravimo tako, da je *igralec* na začetku v sobi, ključ naj se nahaja na tleh ali na neki podlagi (slika 8.5 levo). Nato v urejevalniku procesov dodamo nov korak z imenom »Poberi ključ« in mu dodamo prehod s pogojem Grab Object. Iz hierarhije v polje za pogoj povlečemo vrstico *ključa*. Ne pozabimo povezati tega koraka s korakom »Start«, nato pa preizkusimo igralni način.

Če *ključ* poberemo in ga nato spustimo, pade na tla. To je v igrah pogosto nezaželeno, saj se moramo v tem primeru vsakič skloniti do tal, da *ključ* ponovno poberemo. To rešimo tako, da v komponenti Rigidbody *ključa* označimo polje **Is Kinematic**, polje **Use Gravity** pa odznačimo. Sedaj bomo v igralnem načinu lahko ključ prijeli, če ga spustimo, pa ostane na istem mestu in ne pade na tla.

[2]Vse komponente podobnega tipa lahko tudi sami ročno dodamo objektom, vendar je priporočljivo, da to prepustimo VR Builderju, saj se s tem izognemo mnogim morebitnim napakam.



Slika 8.4: Pogoj Grab Object zahteva en objekt, ki ga želimo pobrati. Dodamo ga s potegom v označeno polje (levo), kjer se nato izpiše njegovo ime (desno).

8.4.1.2 Pogoj Snap Object

Pogoj Snap Object (*sl. pripni objekt*) iz skupine **Interaction** omogoča, da izbranemu objektu določimo lokacijo, kamor želimo, da ga igralec prestavi. Ta lokacija je določena z objektom **Snap Zone** *(sl. območje, kamor se objekt pripne)*, ki ga ustvarimo s pomočjo VR Builderja. Podobno kot pri pogoju Grab Object moramo v polje pogoja dodati objekt, za katerega želimo, da ga uporabnik prestavi na končno mesto. VR Builder mu nato doda komponento Snappable Property (*sl. lastnost, da objekt lahko pripnemo na končno lokacijo*), s katero lahko nato ustvarimo *Snap Zone*. Ustvarjeni objekt moramo nato prav tako dodati v polje pogoja.

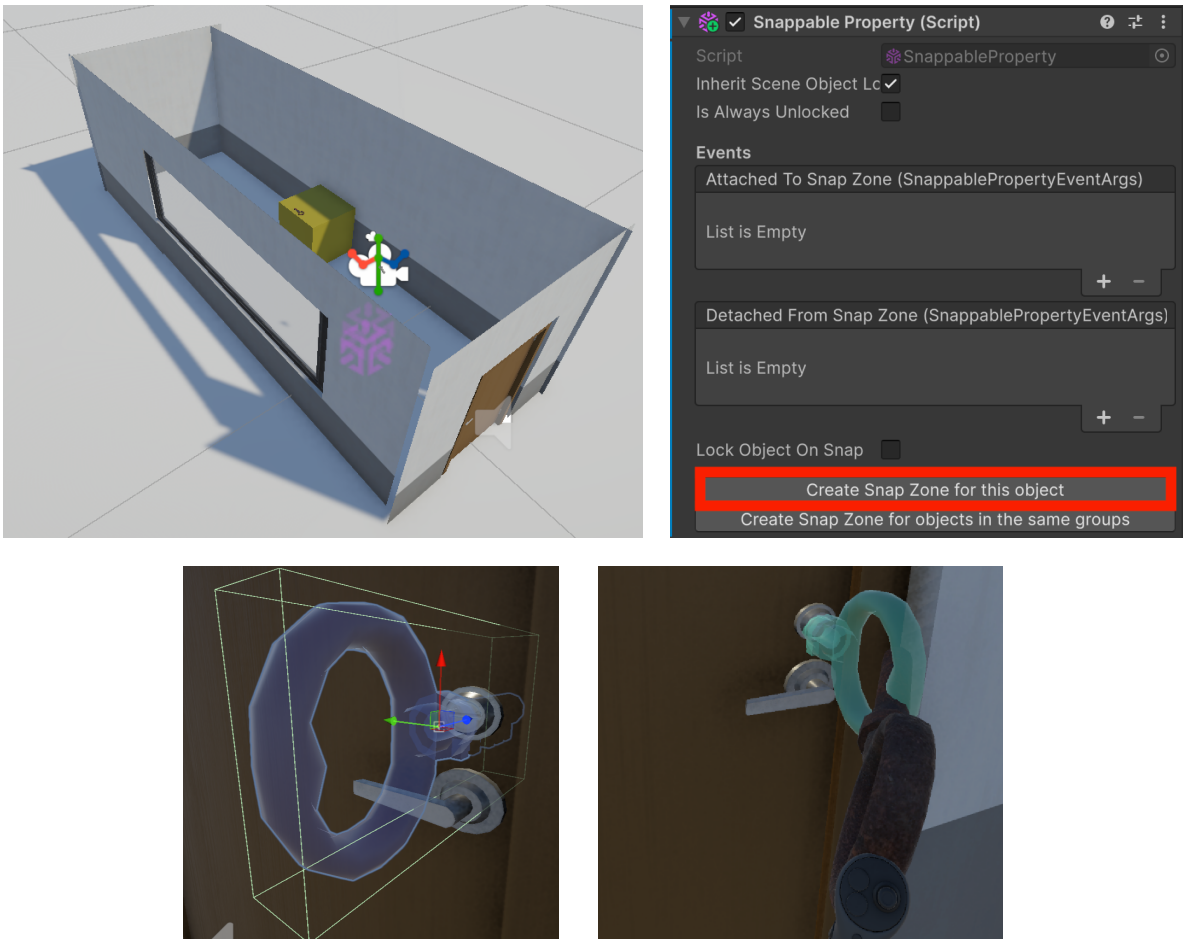
Primer 8.5: Soba pobega: ključ damo v ključavnico

Nadaljujemo primer 8.4 tako, da v urejevalniku procesov ustvarimo nov korak z imenom »Ključ v ključavnico« in mu dodamo prehod s pogojem Snap Object. V tem scenariju *igralec* začne v sobi, kjer se nahaja tudi *ključ* (slika 8.5 levo zgoraj).

V inspektorju korakov v polje objekta dodamo *ključ*, nato pa si ogledamo komponento Snappable Property, ki je vidna v inspektorju *ključa*. Na dnu komponente je gumb `Create Snap Zone for this object` *(sl. ustvari Snap Zone za ta objekt)*, ki ga pritisnemo (slika 8.5 desno zgoraj).

V hierarhiji se pojavi objekt *ključ_SnapZone*. Ta ima enako obliko kot *ključ*, vendar je prikazan kot prosojen moder objekt in določa končno lokacijo, kamor se bo *ključ* postavil (slika 8.5 levo spodaj). Ker želimo *ključ* odnesti do ključavnice v vratih, objekt *SnapZone* premaknemo na ustrezno mesto.

Ko preizkusimo igralni način, se *SnapZone* ob približevanju *ključa* obarva svetlo zeleno (slika 8.5 desno spodaj), kar pomeni, da je objekt v pravilnem položaju. Če *ključ* takrat izpustimo, se samodejno poravna na pozicijo, določeno z objektom *SnapZone*.



Slika 8.5: Soba pobega z igralcem in ključem (levo zgoraj) ter nastavitev in delovanje *SnapZone* za postavitev objekta (ostalo).

8.4.2 Vedenja

Z vedenji v VR Builderju določamo akcije, ki se izvedejo ob aktivaciji posameznega koraka. To pomeni, da se ob aktivaciji koraka najprej izvedejo vsa vedenja, šele nato pa se lahko izpolnijo pogoji za prehod v naslednji korak. Podobno kot prehodi so tudi vedenja razdeljena v štiri večje skupine, mi pa si bomo podrobneje ogledali le nekatere izmed njih.

8.4.2.1 Vedenje Move Object

Vedenje Move Object (*sl. premakni objekt*) iz skupine **Animation** omogoča, da ob aktivaciji koraka sprožimo animiran premik izbranega objekta iz njegove trenutne pozicije na drugo lokacijo. Objekt izberemo tako, da ga povlečemo v ustrezno polje v inspektorju korakov, za to vedenje pa moramo določiti še *t. i. Position Provider* (sl. objekt, ki določa lokacijo). *Position Provider* je lahko katerikoli objekt, saj je za to vedenje pomembna le njegova komponenta transformacije. Ta natančno določa, kje in kako bo animacija premika izbranega objekta zaključena. Dodatno lahko pri tem vedenju izberemo tudi trajanje (v sekundah) in krivuljo animacije.

Primer 8.6: Soba pobega: odpiranje vrat

V primeru 8.5 smo dosegli, da *ključ* postavimo v ključavnico. Sedaj pa bomo poskrbeli še, da se vrata nato odprejo.

> **Pozor!** Če smo uporabili vrata iz paketa »Free Wood Door Pack«, moramo najprej urediti nekaj tehničnih podrobnosti. Vrata iz tega paketa imajo že dodane nekatere komponente, ki omogočajo odpiranje in predvajajo določene zvoke. Zato moramo iz vrat, ki smo jih vnesli v sceno, odstraniti vse komponente, ki bi lahko v kombinaciji z VR Builderjem povzročale nepričakovane težave (predvsem komponento Door (Script) potomca *Door*). Splošno pravilo je, da lahko pri uvoženih objektih določene komponente povzročajo težave, če niso skladne z našim okoljem.

Vrata iz paketa »Free Wood Door Pack« so sestavljena tako, da je okvir vrat ločen od glavne ploskve, ki se odpira. Da bi se vrata odprla, se mora spremeniti vrednost rotacije potomca *Door* na koordinati Y na $Y = 90$ (če ste uporabili druga vrata, poskusite poiskati vrednosti transformacij, ko so vrata odprta). Najdene vrednosti, ki opisujejo transformacije odprtih vrat, si bomo zapomnili tako, da vrednosti komponente kopiramo (desno zgoraj v komponenti najdemo gumb s tremi pikami, kjer izberemo možnost ≡ *Copy* ▶ *Component*). Nato ustvarimo nov prazen objekt, ga poimenujemo v »animacijaVrata« in ga spremenimo v potomca *vrat* (na enak nivo kot je potomec *Door*). V inspektorju v komponenti transformacije z gumbom s tremi pikami zgoraj desno izberemo možnost ≡ *Paste* ▶ *Component Values*. Tako smo ustvarili objekt, ki natančno vsebuje transformacije odprtih vrat in ga lahko uporabimo kot *Position Provider* v vedenju Move Object. Nato spremenimo vrednosti transformacij originalnih vrat nazaj, da so zaprta.

Sedaj lahko v urejevalniku procesov dodamo korak, ki ga poimenujemo »Odpiranje vrat«. Dodamo mu vedenje Move Object in v ustrezna polja vnesemo pripravljene objekte (*Door* in *animacijaVrat*) ter nastavimo trajanje animacije na nekaj sekund. Korak povežemo s prejšnjim, nato pa preizkusimo igralni način. Sedaj se *vrata*, ko *ključ* vstavimo v ključavnico, animirano odprejo.

8.4.2.2 Vedenji Behavior Sequence in Delay

Vedenji **Behavior Sequence** (*sl. zaporedje vedenj*) in **Delay** (*sl. zamik*) sta pripomočka, s katerima lahko potek korakov dodatno urejamo po svojih potrebah.

Če imamo v nekem koraku določenih več vedenj, se ta pričnejo izvajati hkrati. Vedenje Behavior Sequence pa je vsebnik, v katerega navedemo vsa vedenja, za katera želimo, da se izvajajo v določenem vrstnem redu. Tako lahko poskrbimo, da se eno vedenje zaključi, preden se prične naslednje. Vedenje Delay deluje tako, da mora miniti izbrano število sekund, preden lahko iz koraka izstopimo.

8.5 Širša sestava procesa VR Builderja

Kot smo spoznali v prejšnjem poglavju, je proces VR Builderja sestavljen iz posameznih korakov. Obstaja poseben korak Start, ki nima vsebine, a določa, kje se proces začenja, v celotni sliki pa je proces razdeljen na več nivojev.

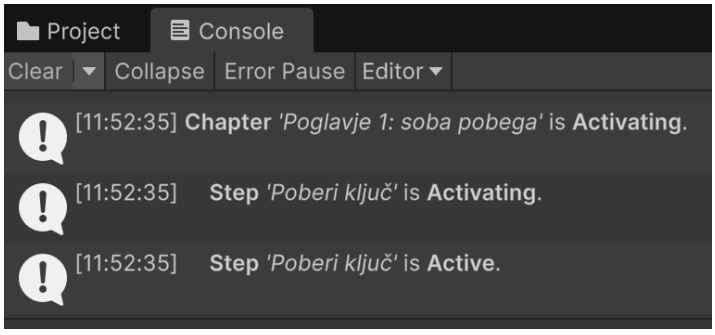
Proces VR Builderja je razdeljen na poglavja (*angl. chapters*). Vsako poglavje lahko v urejevalniku ločeno urejamo in se začne s korakom Start. Zadnji korak v poglavju je samodejno povezan z začetkom naslednjega poglavja (naslednje poglavje je določeno v posebnem koraku ali pa je to naslednje zaporedno poglavje). To omogoča, da igro razdelimo na zaokrožene enote in tako ohranjamo večjo preglednost procesa (več manjših datotek je lažje berljivih kot ena velika).

S pomočjo poglavij razumemo, da korak Start ne služi zgolj kot začetek procesa, temveč določa tudi začetek znotraj posameznih poglavij (in določenih manjših delov, kot jih imamo pri ostalih posebnih tipih korakov).

8.5.1 Spremljanje poteka procesa v konzoli

Vsako poglavje in korak imata svoje določeno ime. Proces razvoja takšnih iger, ki se zanašajo na prehode med različnimi stanji, je lahko dolgotrajen in zapleten, pojavi se lahko veliko napak. Zato je v pomoč, če spremljamo potek procesa v konzoli. Tu se namreč samodejno zabeležijo vsi dogodki, vezani na proces VR Builderja: vsak začetek, aktivacija, prehod in deaktivacija poglavij ter korakov.

Kot je prikazano na sliki 8.6, je izpis v konzoli vezan na imena korakov in poglavij. Zato je priporočljivo, da vsa poglavja in korake smiselno poimenujemo in si tako olajšamo iskanje morebitnih napak.



Slika 8.6: Konzola prikazuje zapise med igralnim načinom, vključno s stanjem procesa ter aktivacijo in deaktivacijo korakov.

Sklepna beseda

Prišli ste do konca vodnika po svetu igralnega pogona Unity. Če ste vestno sledili poglavjem, zdaj razumete, da razvoj 3D izkušenj in XR vsebin ni le domena programerjev, temveč kreativno polje, kjer se prepletajo tehnika, umetnost in načrtovanje uporabniške izkušnje.

Postavili ste trdne temelje: od razumevanja hierarhije objektov in komponent do pisanja prvih skript v jeziku `C#` ter optimizacije vsebin za sodobne naprave. Toda je razvoj iger je področje, ki se nenehno spreminja in različica Unity, ki smo jo uporabili, bo kmalu dobila naslednico, tehnologije XR pa bodo postale še bolj dostopne.

Naj vas morebitne napake v kodi ali zapleteni logični izzivi ne ustavijo. Skupnost razvijalcev Unity je velika in rešitev za skoraj vsak problem je običajno le nekaj klikov stran v uradni dokumentaciji, na forumih ali s pomočjo orodij umetne inteligence.

Vaša pot se tukaj ne konča, temveč se šele zares začenja. Zdaj je čas, da odprete prazen projekt in začnete graditi svojo prvo samostojno idejo. Srečno na poti ustvarjanja!

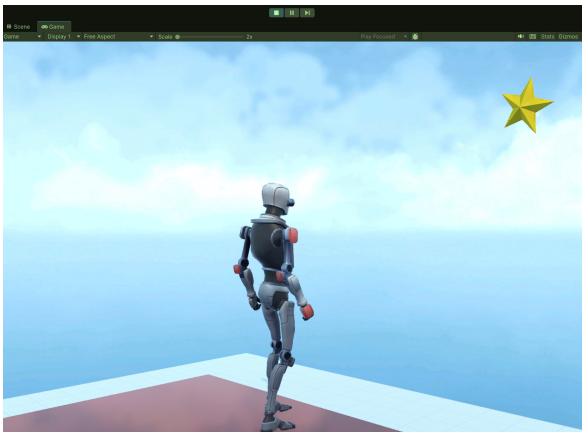
DODATEK A

Rešitve izzivov

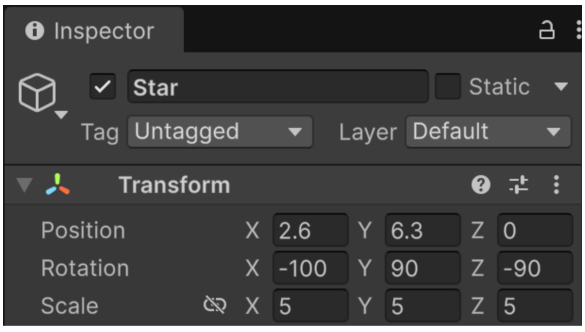
Rešitev izziva 1

Najprej zaženemo igralni pogled, nato pa se z igralcem postavimo na vrh stopnic. Kot je razvidno tudi na sliki A.1, je *zvezda* nekoliko višje in pred *igralcem*. Če poskušamo skočiti, je ne moremo osvojiti. Njeno lokacijo lahko popravimo tako, da v hierarhiji izberemo njen objekt *Star* (sl. zvezda). V oknu inspektor se pojavijo njene komponente, osredotočimo se na transformacije. S spreminjanjem vrednosti koordinate X lahko *zvezdo* prestavimo natanko nad igralca (za pomoč pri nastavljanju vrednosti glej sliko A.2). Sedaj jo lahko s skokom osvojimo in nato njen objekt izgine tako iz scene kot iz hierarhije.

Če igralni način zaustavimo in ga znova zaženemo, je *zvezda* še vedno postavljena tako, da je *igralec* ne doseže (spremembe, ki smo jih naredili prej, se niso shranile). Da bi se *zvezda* vedno pojavila na želeni poziciji, moramo zaustaviti igralni način in ponovno spremeniti koordinate transformacije. Sedaj se pri ponovnem zagonu igralnega načina narejene spremembe ohranijo.



Slika A.1: Z *igralcem* se vzpnemo po stopnicah, *zvezda* je na začetku precej oddaljena od *igralca* in je ne moremo enostavno pobrati.

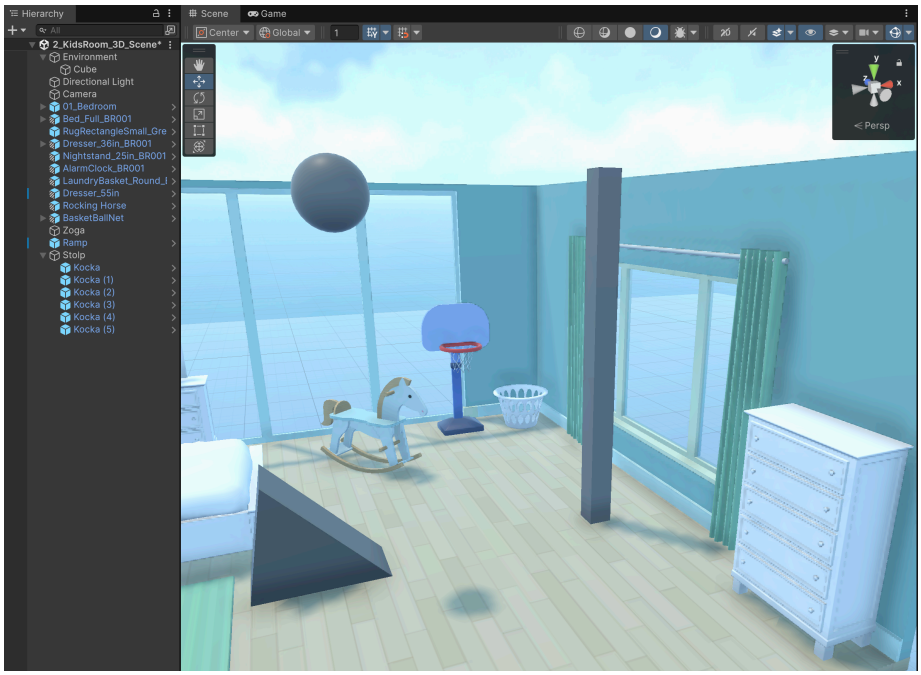


Slika A.2: Koordinate komponente transformacije, ki zadostujejo, da lahko *zvezdo* poberemo s skokom.

Rešitev izziva 2

Najenostavneje je iz scene pobrisati ustvarjene keglje in na istem mestu ustvariti en pokončen kvader z imenom Kocka, ki ga takoj spremenimo v prefab. Nato v sceno dodamo toliko *kock*, kot želimo, da je stolp visok. Pri poravnavanju kock si pomagamo z načinom poravnave oglišč.

V inspektorju prefaba preverimo, da ima *kocka* komponenti Rigidbody in collider. Ustvarjene kocke lahko prestavimo kot sinove v prazen objekt »Stolp«, da ga je enostavneje premikati po sceni. Ostale korake za podiranje stolpa z *žogo* ponovimo po zgledu primera 4.6. Rezultat, kako naj bi stolp izgledal, je prikazan na slikah A.3 in A.4.



Slika A.3: Stolp je sestavljen iz posameznih blokov, ki so postavljeni eden na drugega.

Slika A.4: Rezultat podiranja stolpa v igralnem načinu.

Rešitev izziva 8

Število pobranih kovancev lahko beležimo na objektu igralca s pomočjo že pripravljene skripte. V datoteki `PlayerController.cs` dodamo parameter `score`, ki bo hranil število pobranih kovancev. To storimo tako, da v glavi razreda `PlayerController` pod vrstico s spremenljivko `rotationSpeed` dodamo vrstico

```
private int score = 0;
```

Tukaj izberemo ključno besedo `private` namesto `public`, ker ne želimo, da bi bilo to vrednost mogoče urejati v inspektorju. V razred dodamo še novo metodo, s katero bomo povečali število točk. Metoda lahko izgleda tako:

```
public void AddScore(int points)
{
    score += points;
}
```

Potrebno je tudi zagotoviti, da se metoda izvede, kar naj se zgodi, ko poberemo kovanec. V datoteki `Collectibe.cs` lahko v metodi `OnTriggerEnter` poskrbimo, da se istočasno poveča število točk igralca. To storimo s kodo:

```
PlayerController playerController = other.GetComponent<
   PlayerController>();
playerController?.AddScore(1);
```

V metodi spremenljivka `other` predstavlja komponento Collider danega objekta, mi pa potrebujemo sam objekt igralca. To dosežemo s klicem metode `GetComponent` in rezultat shranimo v spremenljivko `playerController`. Nato na tem objektu pokličemo metodo `AddScore`.

Na tej točki si igra beleži število pobranih *kovancev*, igralcu pa ta informacija še ni prikazana. Slednje dosežemo s posebnim objektom, ki ga v hierarhiji ustvarimo z izbiro ≡ *UI* ▶ *Text – TextMeshPro*. Ustvaril se bo objekt *Canvas*, ki vsebuje otroka *Text (TMP)*. To sta posebna objekta, katerih namen je izris uporabniškega vmesnika. Posebnost takega tipa objektov je, da je njihov prikaz pravilen le v oknu igre. Priporočljivo je, da je to okno odprto, saj bomo tako sproti videli posledice sprememb parametrov izrisa uporabniškega vmesnika.

Privzeto se bo izpis točk nahajal sredi zaslona, kar večinoma ni zaželeno. Objektom tipa položaj urejamo prek komponente Rect Transform ter položaj prek komponente , sam proces pa spominja na urejanje *Word* dokumenta. Da izpis premaknemo na zgornji del zaslona, na komponenti Rect Transform v zgornjem levem kotu izberemo Sidro

(angl. Anchor) in mu določimo vrednost zgoraj sredina *(angl. top-center)*. Nastaviti moramo tudi vrtišče *(angl. pivot)*, ki določa poravnavo besedila glede na sidro, v našem primeru mu pripišemo vrednosti $X = 0.5, Y = 1$. Nazadnje popravimo še natančen položaj glede na sidro (Pos X, Pos Y in Pos Z), ki ga v vseh dimenzijah nastavimo na 0.

Poravnavo besedila in začetno vrednost lahko uredimo v komponenti TextMeshPro – Text(UI). V polje **Text input** zapišemo začetno besedilo »Točke: 0«, pri nastavitvi poravnave *(angl. Alignment)* pa izberemo sredinsko poravnavo.

Zadnji korak je posodabljanje vsebine besedila, kadar se spremeni število točk. To storimo tako, da v skripti `PlayerController.cs` na vrhu dodamo vrstico

```
using TMPro;
```

in spremenljivkam v glavi razreda dodamo vrstico

```
public TextMeshProUGUI scoreText;
```

Na koncu v metodi `AddScore` dodamo vrstico

```
scoreText.text = "Tocke: " + score.ToString();
```

V inspektorju *igralca* sedaj dodamo v komponenti za skripto objekt *Text (TMP)* v parameter **scoreText**. Po tem postopku bi moralo biti v igralnem načinu v pogledu igralca vidno besedilo, ki beleži število točk.

Končna koda bi morala izgledati približno tako:

DATOTEKA: **PlayerController.cs**

```
using UnityEngine;
using TMPro;

// Controls player movement and rotation.
public class PlayerController : MonoBehaviour
{
    public float speed = 5.0f; // Set player's movement speed.
    public float rotationSpeed = 120.0f; // Set player's
       rotation speed.
    private int score = 0; // Set score
    public TextMeshProUGUI scoreText;
    private Rigidbody rb; // Reference to player's Rigidbody.
```

```
    // Start is called before the first frame update
    private void Start()
    {
        rb = GetComponent<Rigidbody>(); // Access player's
           Rigidbody.
        if (scoreText != null)
        {
             scoreText.text = "Tocke: 0";
        }
    }

    // Update is called once per frame
    void Update()
    {

    }

    // Handle physics-based movement and rotation.
    private void FixedUpdate()
    {
        // Move player based on vertical input.
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = transform.forward * moveVertical *
           speed * Time.fixedDeltaTime;
        rb.MovePosition(rb.position + movement);

        // Rotate player based on horizontal input.
        float turn = Input.GetAxis("Horizontal") *
           rotationSpeed * Time.fixedDeltaTime;
        Quaternion turnRotation = Quaternion.Euler(0f, turn, 0
           f);
        rb.MoveRotation(rb.rotation * turnRotation);
    }

    public void AddScore(int points)
    {
        score += points;
        scoreText.text = "Tocke: " + score.ToString();
    }
}
```

DATOTEKA: **Collectible.cs**

```
using UnityEngine;

public class Collectible : MonoBehaviour
{
    public float rotationSpeed = 0.5f;
    public GameObject onCollectEffect;
    // Start is called once before the first execution of
       Update after the MonoBehaviour is created
    void Start()
    {

    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player")) {
            Destroy(gameObject);
            Instantiate(onCollectEffect, transform.position,
               transform.rotation);
            PlayerController playerController = other.
               GetComponent<PlayerController>(); // get the
               playerController component
            playerController?.AddScore(1); // increment the
               score of the player
        }
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(0, rotationSpeed, 0);
    }
}
```

DODATEK B

Slovar

Architectural Visualization
arhitekturne vizualizacije

Asset Packs
sestavljeni seti

Assets
splošnejši izraz za elemente v Unity projektu, kot na primer komponente, objekti, scene, skripte, paketi ...

Audio Listener
poslušalec (komponenta)

Audio Mixer
mešalnik zvoka

Audio Source
zvočni vir (komponenta ali igralni objekt)

Behavior Sequence
zaporedje vedenj

Behaviors
vedenja

Boundary
varnostna meja

Built-In Render Pipeline
vgrajeni upodabljalni cevovod

Camera
kamera (igralni objekt)

Cast
deljenje zaslona

Chapter
poglavje

Clipping Planes
mejni ravnini

Collider
komponenta Collider, ki objektu doda računsko mejo

Component
komponenta

Console
konzola

Cross-Platform Development
razvoj programske opreme, ki je prenosljiva med različnimi platformami

Delay
zamik

Directional Light
usmerjena luč

Environment Packs
celovita okolja

Extended Reality
razširjena resničnost

Frame
sličica

Game
okno igre

Game Engine
igralni pogon

Game Mode
igralni način

Game Object
igralni objekt

Geometry
geometrija

Get Set Up
zavihek Začetek

Grab Object
poberi objekt

Grabbable Property
lastnost, da objekt lahko primemo

Hierarchy
okno hierarhija

High Definition Render Pipeline
visokoločljivostni upodabljalni cevovod

Immersive
potopitveno

Inspector
okno inspektor

Installs
zavihek Nameščeni urejevalniki in orodja

Integrated Development Environment
integrirano razvojno orodje

Learn
zavihek Učenje

Licenses
zavihek Licence

Light
luč

Locomotion
premikanje

Mesh
poligonska mreža

Mixed Reality
mešana resničnost

Move Object
premakni objekt

Move Tool
orodje za premikanje

Package Manager
paketni upravitelj

Pancake Lens
palačinkasti leči

Parent-Child Relationship
razmerje starš-otrok

Particle System
sistem delcev

Passthrough
pogled skozi

Physically Based Rendering
fizikalno osnovano upodabljanje

Physics Engine
fizikalni pogon

Physics Material
fizikalni material

Player
igralec

Point Light
točkasta luč

Position Provider
objekt, ki določa lokacijo

Prefab
šablona objekta

Prefab Mode
prefab pogled

Prefab Variant
varianta prefaba

Process Editor
urejevalnik procesa

Project
projektno okno

Projects
zavihek Projekti

Real-Time Rendering
upodabljanje v realnem času

Rect Tool
orodje za spreminjanje očrtanega kvadra

Render Pipeline
upodabljalni cevovod

Rendering
upodabljanje

Resources
zavihek Viri

Rigidbody
komponenta Rigidbody, ki objektu doda simulacijo fizikalnih sil

Rotate Tool
orodje za rotiranje

Scale Tool
orodje za spreminjanje velikosti

Scene
okno scene

Shading
senčenje

Snap Object
pripni objekt

Snappable Property
lastnost, da objekt lahko pripnemo na končno lokacijo

Spot Light
reflektor

Standalone
samostojna (naprava)

Step
korak

Step Inspector
inspektor korakov

Stuttering
zatikanje

Transform
(komponenta) transformacije

Transform Tool
orodje za transformacije

Transitions
prehodi

Trigger
sprožilec

Tutorial
interaktivni vodič

Unity Editor
urejevalnik Unity

Universal Rendering Pipeline
univerzalni upodabljalni cevovod

Vertex Snapping
poravnava po ogliščih

VFX Graph
VFX graf

View Frustum
vidni stožec

View Tool
orodje pogleda

Windows
okna urejevalnika

DODATEK C

Slovar gumbov in polj v vmesniku

3D Sound Settings
nastavitve 3D zvoka s krivuljo

Add
dodaj

Add Component
dodaj komponento

Add to My Assets
dodaj paket v mojo knjižnico

Advanced Settings
napredne nastavitve

Angular Damping
rotacijski upor

Audio Resource
zvočna datoteka

Automatic Center of Mass
avtomatsko računanje centra gravitacije

Automatic Tensor
avtomatski tenzor

Average
povprečje

Base Map
osnovni videz materiala

Bounce Combine
metoda izračuna skupne odbojnosti

Bounciness
koeficient prožnosti

Box
škatlasta oblika oz. kvader

Capsule
oblika kapsule

Clipping Planes
mejni ravnini

Clipping Planes
mejni ravnini

Close Tab
zapri zavihek

Collision Detection
detekcija trkov

Color
barva

Convert Assets
pretvori elemente

Copy
kopiraj

Create Project
ustvari projekt

Create Snap Zone for this object
ustvari SnapZone za ta objekt

Cube
kocka

Delete
izbriši

Description
opis

Desktop
namizje

Devices
naprave

Download
prenesi

Download Template
prenesi predlogo

Duplicate
podvoji

Duration
trajanje

Dynamic Friction
koeficient trenja

Edit Collider
uredi collider

Emission
oddajanje

Environment
okolje

Factory Reset
ponastavitev na tovarniške nastavitve

Far
daleč

Field of View
zorni kot

Finish
končaj

Friction Combine
metoda izračuna skupnega trenja

Headset Settings
nastavitve naprave

Import
uvozi

Initialize Converters
inicializiraj pretvornike

Install
namesti

Install Editor
namesti urejevalnik

Intensity
jakost

Interaction
interakcije

Interpolate
interpolacija

Is Kinematic
kinematičnost

Is Trigger
sprožilec

Launch
zagon

Layer
nivo

Linear Damping
linearen upor

Locomotion
premikanje

Loop
predvajaj ponavljajoče

Looping
ponavljanje

Mass
masa

Material Upgrade
nadgradi material

Maximum
maksimum

Mesh Collider
collider, ki se natančno prilagaja kompleksni geometriji objekta, definirani s poligonsko mrežo

Metallic Map
kovinska mapa

Minimum
minimum

Multiply
zmnožek

Near
blizu

New Project
nov projekt

New Scene
ustvari novo sceno

Open
odpri

Ortographic
ortografska projekcija

Pair Headset
poveži se z napravo

Paste
prilepi

Pause
prekini

Perspective
perspektiva

Pitch
višina tona

Play
začni

Play on Awake
predvajaj ob začetku

Playmode Tint
barva igralnega načina

Proceed
nadaljuj

Projection
projekcija

Provides Contact
podatki o stikih

Range
domet

Receive Shadows
sprejme sence

Recommended
priporočena različica

Rename
preimenuj

Reverb
odmev

Rotation Speed
hitrost obračanja

Save
shrani

Shape
oblika

Show Tutorials
prikaži interaktivne vodiče

Size
velikost

Smoothnes
gladkost

SnapZone
igralni objekt, ki določa, kam lahko pripnemo objekt s komponento Snappable Property

Source
vir pridobivanja informacij o gladkosti

Spatial Blend
prostorsko prelivanje

Speed
hitrost

Sphere
sfera

Star
zvezda

Static Friction
koeficient lepenja

Step
naslednji korak

Step Name
ime koraka

Surface
vhodni parametri površine

Surface Options
možnosti površine

Surface Type
vrsta površine

Tag
značka

Take My Current Scene
uporabi trenutno sceno

Tiling and Offset
ponovitev in zamik

Type
tip (luči)

Unlocked Objects
omogočeni objekti

Untitled
nepoimenovano; navadno poimenovanje za sceno, ki še ni bila shranjena v pomnilnik

Use Air Link
uporabi Air Link

Use Gravity
uporabi gravitacijo

Utility
pripomočki

Volume
glasnost

Workflow Mode
model, ki se uporablja za izračun osvetlitve

DODATEK D

Kratice

AR
obogatena resničnost, OR

ArchViz
arhitekturne vizualizacije

HDRP
visokoločljivostni upodabljalni cevovod

IDE
integrirano razvojno orodje

MR
mešana resničnost

URP
univerzalni upodabljalni cevovod

VR
navidezna resničnost, NR

XR
razširjena resničnost

Stvarno kazalo

Slike

Literatura

[1] Meta Platforms, Inc. *Hands — Mapping Controller Interactions to Hands*. 2026. URL: https://developers.meta.com/horizon/design/hands/ (pridobljeno 3. 4. 2026).

[2] Meta Platforms, Inc. *Meta Quest 3: The ultimate mixed reality headset*. Meta. 2026. URL: https://www.meta.com/quest/quest-3 (pridobljeno 20. 2. 2026).

[3] Meta Platforms, Inc. *Meta Quest Touch Plus Controller*. 2026. URL: https://www.meta.com/quest/accessories/quest-touch-plus-controller/ (pridobljeno 4. 3. 2026).

[4] MindPort GmbH. *VR Builder: Free Unity plug-in for VR development*. MindPort. 2026. URL: https://www.mindport.co/vr-builder (pridobljeno 21. 2. 2026).

[5] Unity Technologies. *Unity Essentials*. Unity Technologies. 2026. URL: https://learn.unity.com/pathway/unity-essentials (pridobljeno 2. 4. 2026).

TeachXR

Projekt TeachXR je usmerjen v raziskovanje in uvajanje tehnologij navidezne, obogatene in razširjene resničnosti (XR) v osnovno- in srednješolsko izobraževanje. V ospredje postavlja učitelje kot ključne nosilce sprememb, ki lahko s premišljeno uporabo tehnologije pomembno obogatijo učni proces.

Osrednji rezultat projekta predstavlja razvoj digitalnih učnih gradiv, vključno z interaktivnimi učbeniki, strokovnimi monografijami in XR vsebinami, ki nadgrajujejo obstoječe učne vire. Gradiva omogočajo bolj vizualno, prostorsko in interaktivno obravnavo učnih vsebin ter podpirajo bolj poglobljeno razumevanje zahtevnejših konceptov.

V okviru projekta so bila izvedena strokovna usposabljanja in praktične delavnice, v katerih so učitelji pridobili temeljna znanja za uporabo XR orodij ter razvoj lastnih učnih pristopov. Ta znanja predstavljajo osnovo za nadaljnje vključevanje XR tehnologij v vsakodnevno pedagoško prakso. Vzpostavljeni učni laboratoriji po Sloveniji dodatno omogočajo dostop do tehnologije, preizkušanje vsebin in nadaljnji razvoj kompetenc, hkrati pa spodbujajo sodelovanje in izmenjavo dobrih praks med učitelji.

TeachXR tako ne prinaša zgolj novih tehnologij, temveč vzpostavlja temelje za njihovo smiselno, trajnostno in dolgoročno vključevanje v izobraževalni proces.

Avtorji

Marija Absec
Uroš Šmajdek
Klara Žniderščič
Klemen Pečnik
Žana Juvan
Matija Marolt
Matevž Pesek