

**IZDELAVA LASTNEGA IGRALNEGA POGONA (TJ GAME  
ENGINE)  
raziskovalna naloga**

Avtorji: Simon Gajšek, Urh Jelen in David Motoh

Mentor: mag. Boštjan Resinovič

Mestna občina Celje, Mladi za Celje

Celje, 2025

## IZJAVA\*

Mentor/-ica Boštjan Resinovič v skladu z 20. členom Pravilnika o organizaciji mladinske raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljam, da je v raziskovalni nalogi z naslovom IZDELAVA LASTNEGA IGRALNEGA POGONA (TI GAME ENGINE), katere avtorji so Simon Gajšek, Urh Jelen in David Motoh:

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje.

Celje, 4. 4. 2025

žig šole



Podpis mentorja

Podpis odgovorne osebe

\*

### POJASNILO

V skladu z 20. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja (-ice) in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja (-ice) fotografskega gradiva, katerega ni avtor (-ica) raziskovalne naloge, pa hrani šola v svojem arhivu.

## **ZAHVALA**

Iskreno se zahvaljujemo vsem, ki so nam med izdelavo raziskovalne naloge pomagali s svojimi nasveti in strokovnim znanjem.

Posebna zahvala gre našemu mentorju, mag. Boštjanu Resinoviču za nujno podporo, usmerjanje in dragocene nasvete, ki so nam pomagali pri iskanju pravih rešitev.

Hvaležni smo tudi vsem sodelujočim, ki so si vzeli čas za našo anketo in tako pripomogli k celovitosti naše raziskave.

Posebno zahvalo namenjamo tudi našim staršem, ki so nas ves čas spodbujali, nam stali ob strani ter nam omogočili pogoje za nemoteno delo.

Hvaležni smo vsem, ki so kakorkoli prispevali k nastanku te naloge.

## Kazalo vsebine

1	Uvod.....	6
1.1	Hipoteze.....	6
1.2	Cilj .....	7
1.3	Metode dela .....	7
2	Uporabljena orodja .....	8
2.1	Programski jezik C#.....	8
2.2	Visual Studio .....	9
2.3	Windows Forms.....	10
2.3.1	Splošno .....	10
2.3.2	Kako deluje.....	10
2.3.3	Uporabniški vmesnik in delovanje.....	10
2.3.4	Razvojne prakse.....	11
3	Videoigre .....	11
4	Trenutne značilnosti igralnih pogonov .....	13
5	Kultura in etična vprašanja v igrah.....	13
6	Izdelava našega igralnega pogona.....	13
6.1	Prva verzija – upravljanje samo prek kode .....	14
6.1.1.	Povzetek prve verzije igralnega pogona .....	14
6.2	Druga verzija – dodan grafični uporabniški vmesnik (GUI) .....	15
7	Grafični vmesnik .....	16
8	Fizika .....	18
9	Scene .....	18
10	Animacije .....	19
11	Izdelava igre v našem igralnem pogonu.....	19
11.1	Pong.....	19
11.2	Flappy bird .....	20
12	Pregled že obstoječih igralnih pogonov .....	20
12.1	Unreal Engine .....	20
12.2	Unity .....	21
12.3	Godot.....	22
12.4	Zakaj izdelati lasten igralni pogon namesto uporabe obstoječega? .....	22
12.5	Primerjava funkcionalnosti igralnih pogonov .....	23
12.5.1.	Enostavnost uporabe .....	23
12.5.2.	Zmogljivost .....	23

12.5.3.	Lastnosti.....	24
12.5.4.	Skupnost in podpora .....	25
12.5.5.	Primeri uporabe .....	25
13	Anketa o igralnem pogonu.....	26
14	Potrjevanje hipotez .....	30
15	Zaključek in smernice za nadaljnjo delo.....	31

## Kazalo slik

Slika 1:	Logotip jezika c#.....	8
Slika 2:	Logotip Visual Studia .....	9
Slika 3:	Prikaz 2d igre.....	12
Slika 4:	Upravljanje prek kode .....	15
Slika 5:	Grafični vmesnik.....	16
Slika 6:	lastnosti objektov .....	17
Slika 7:	Izrez igre pong v našem pogonu .....	19
Slika 8:	Izrez igre Flappy bird v našem pogonu.....	20
Slika 9:	Izgled Unreal Engin z različnimi okni .....	21
Slika 10:	Logotip Untity.....	21
Slika 11:	Izgled Godota .....	22

## Kazalo grafov

graf 1:	Prikaz izkušenj anketirancev .....	26
graf 2:	Prikaz pomembnosti enostavnega pogona .....	27
graf 3:	Prikaz najpomembnejših elementov v pogonu.....	27
graf 4:	Prikaz ustrežnejšega vnosa .....	28
graf 5:	Prikaz najbolj uporabljenih lastnosti fizik.....	28
graf 6:	Prikaz potrebnega časa .....	29
graf 7:	Prikaz mnenja o nadaljnji uporabi našega pogona .....	29

## **POVZETEK**

Raziskovalna naloga se osredotoča na razvoj lastnega igralnega pogona, namenjenega izdelavi 2D platformskih iger. Cilj je ustvariti uporabniku prijazen pogon, ki bi bil dostopen tudi tistim brez naprednega programerskega znanja. Razvili smo dve različici pogona:

Prva verzija – v celoti odvisna od kode, brez grafičnega vmesnika, kar omogoča večji nadzor nad delovanjem, a zahteva programersko znanje.

Druga verzija – dodan grafični uporabniški vmesnik (GUI), ki omogoča lažjo uporabo in hitrejši razvoj iger, hkrati pa ohranja možnost programiranja za naprednejše funkcionalnosti.

Pogon omogoča osnovne funkcije, kot so upodabljanje grafike, fizikalni sistem in zaznavanje trkov. Testiran je bil z izdelavo enostavnih iger, kot sta Pong in Flappy Bird.

Raziskava vključuje tudi primerjavo z obstoječimi pogoni, kot so Unity, Unreal Engine in Godot, ter izpostavlja prednosti in slabosti razvoja lastnega pogona. Ključne ugotovitve kažejo, da je mogoče ustvariti uporaben igralni pogon brez finančnih stroškov in z znanjem na srednješolski ravni, čeprav ima tak pogon omejene zmogljivosti v primerjavi z uveljavljenimi rešitvami, grafično pa še ni dovolj prijazen uporabniku.

**Ključne besede:** igralni pogon, platformske igre, C#, razvijanje iger

## **ABSTRACT**

The research focuses on the development of a custom game engine designed for creating 2D platform games. The goal is to develop a user-friendly engine that is accessible even to those without advanced programming knowledge. We developed two versions of the engine:

First version – entirely code-dependent, without a graphical user interface, providing greater control over functionality but requiring programming skills.

Second version – includes a graphical user interface (GUI), making it easier to use and accelerating game development while still allowing programming for advanced features.

The engine provides essential functions such as graphics rendering, a physics system, and collision detection. It was tested by creating simple games like Pong and Flappy Bird.

The research also includes a comparison with existing engines like Unity, Unreal Engine, and Godot, highlighting the advantages and disadvantages of developing a custom game engine. Key findings indicate that it is possible to create a functional game engine without financial costs and with high school-level knowledge, although such an engine has limited capabilities compared to established solutions and its graphical interface is not yet fully user-friendly.

**Keywords:** game engine, platform games, C#, game development

# 1 Uvod

Igralni pogon (ang. *game engine*) je programsko okolje namenjeno izdelovanju video iger. Po navadi vsebuje različna orodja in funkcije za razvoj aplikacij. Nekatere izmed funkcionalnosti sodobnih igralnih pogonov so upodabljanje, animacija, podpora okolju, fizika, zvočna in omrežna podpora, podpora posebnim efektom, uporabniški vmesnik ter podpora za vhodne naprave. Najbolj znana igralna pogona v sodobnem času sta Unity in Unreal Engine.

Platformske igre so podzvrst akcijskih video iger v katerih je glavni cilj premik lika med dvema točkama v danem okolju. Take igre večinoma vsebujejo različne nivoje, spreminjajoča se okolja, s katerimi lahko ima lik različne interakcije in opravlja različne manevre. Platformske igre so večinoma 2D, poznamo pa tudi 3D različice, kjer je kamera postavljena za našim likom in je okolje trodimenzionalno. Najbolj znane platformske igre so Space panic, Donkey kong, Mario, Sonic the hedgehog...

V našem projektne delu bomo ustvarili igralni pogon, specifično oblikovan za razvoj 2D platformskih iger. Primerjali jih bomo z drugimi pogoni in preverili, kakšna bo uporabniška izkušnja in kvaliteta narejenih iger. Primerjave bomo naredili s pomočjo različnih anket in testiranjem dejanske uporabniške izkušnje učencev računalniške šole.

## 1.1 Hipoteze

Namen raziskovalne naloge je predstaviti, kako narediti lasten igralni pogon in ga primerjati z ostalimi pogoni. Pri tem smo si zastavili naslednje hipoteze.

1. Naš igralni pogon bo lažji za uporabo in bolj prijazen uporabnikom pri izdelavi platformskih iger kot uveljavljeni pogoni.
2. Že obstoječe popularne platformske igre bodo lahko narejene v našem igralnem pogonu.
3. V našem igralnem pogonu bodo lahko ustvarjali ljudje z amaterskim znanjem ustvarjanja video iger.
4. Igralni pogon je možno narediti brez stroškov in s trenutnim srednješolskim znanjem.

## **1.2 Cilj**

Cilj raziskovalne naloge je izdelati uporaben in kvaliteten igralni pogon, s katerim bodo lahko uporabniki na enostaven način ustvarili 2D platforme igre. S tem ciljamo predvsem na to, da bodo lahko naš pogon uporabljali ustvarjalci, ki za to ne bi potrebovali profesionalnega znanja na tem področju. Nameravamo vključiti naslednje glavne elemente sodobnih igralnih pogonov: upodabljanje, fizika, dodajanje entitet.

## **1.3 Metode dela**

Za pripravo raziskovalne naloge smo uporabljali naslednje metode:

- Metoda ankete (anketirali smo dijake računalniške šole glede na uporabnost in prijaznost našega pogona)
- Metoda raziskave (raziskali smo kako narediti igralni pogon)
- Metoda študij literature
- Metoda primerov
- Metoda eksperimentiranja (izdelek – igralni pogon)
- Metoda prototipov (prototip pogona in njegova testiranja, testirali smo jih tako mi kot drugi)

## 2 Uporabljeni orodja

### 2.1 Programski jezik C#

C# je vsestranski programski jezik na visoki ravni. Podpira objektno orientirano programiranje. Jezik so razvili Anders Hejlsberg, Scott Wiltamuth in Peter Golde pri Microsoftu, prvič pa je bil predstavljen julija 2000. Kasneje sta ga kot mednarodni standard potrdili organizaciji Ecma (ECMA-334) leta 2002 in ISO/IEC leta 2003. Microsoft je C# predstavil skupaj z .NET Framework in Microsoft Visual Studio, ki sta bila sprva zaprta kodna, saj podjetje takrat še ni ponujalo odprtokodnih rešitev.<sup>1</sup>



Slika 1: Logotip jezika c#

Leta 2004 je bil predstavljen odprtokodni projekt Microsoft Mono, ki je omogočal prevajalnik in izvajalno okolje za C# na več platformah. Desetletje kasneje je Microsoft odprl kodo za več svojih ključnih orodij, vključno z urejevalnikom Visual Studio Code, prevajalnikom Roslyn in enotnim ogrodjem .NET. Vsa ta orodja zdaj podpirajo C#, so brezplačna, odprtokodna in delujejo na več platformah. Mono se je sčasoma pridružil Microsoftu, vendar ni bil združen z .NET.

---

<sup>1</sup> (C# language documentation, 2023)

## 2.2 Visual Studio

Visual Studio je integrirano razvojno okolje (IDE), ki ga je razvil Microsoft za ustvarjanje različnih vrst programov, vključno s spletnimi mesti, spletnimi aplikacijami, spletnimi storitvami in mobilnimi aplikacijami. Podpira razvoj na Microsoftovih platformah, kot so Windows API, Windows Forms, Windows Presentation Foundation (WPF), Microsoft Store in Microsoft Silverlight. Omogoča pisanje kode za izobraževalne namene, kot tudi za profesionalno uporabo. V Visual Studio je vgrajen urejevalnik kode s podporo za IntelliSense, ki omogoča samodejno dopolnjevanje kode, ter funkcijami za preoblikovanje kode. Integrirani razhroščevalnik lahko deluje tako na izvorni ravni kot na ravni strojne kode. Poleg tega vključuje številna druga orodja, grafični urejevalnik za izdelavo GUI aplikacij, spletni oblikovalec, oblikovalec razredov in oblikovalec sheme baze podatkov. Funkcionalnost Visual Studia je mogoče razširiti z vtičniki, ki omogočajo dodatne možnosti, kot so podpora za sisteme za nadzor različic, novi urejevalniki in vizualni oblikovalci za specifične programske jezike ali orodja za upravljanje življenjskega cikla razvoja programske opreme, kot je Azure DevOps: Team Explorer.<sup>2</sup>



Slika 2: Logotip Visual Studia

---

<sup>2</sup> (Visual Studio, 2025)

## 2.3 Windows Forms

### 2.3.1 Splošno

Windows Forms (ali krajše WinForms) je bilo med prvimi orodji, ki jih je Microsoft ponudil za razvoj namiznih aplikacij v okolju .NET. Zasnovano je na Windows API (Win32) in je zato tesno povezano z operacijskim sistemom Windows. Čeprav je Microsoft v zadnjih letih odprl velik del okolja .NET, WinForms kot tak ostaja omejen zgolj na Windows in se ne izvaja večplatformno. Kljub temu je zaradi svoje preprostosti in stabilnosti še vedno priljubljena izbira za številne poslovne ali interaktivne projekte, kjer so glavna zahteva klasični uporabniški obrazci in enostavno upravljanje z dogodki.<sup>3</sup>

### 2.3.2 Kako deluje

Jedro Windows Forms je zasnovano na dogodkovno-krmiljenem modelu. To pomeni, da se delovanje aplikacije sproži in usmerja prek dogodkov, kot so kliki, vnosi besedila in druge uporabniške interakcije. Celotno dogajanje se odvija v enem ali več »obrazcih« (Forms), znotraj katerih so nameščeni kontrolniki (gumbi, besedilna polja, sezname ipd.). Vsaka uporabniška akcija na kontrolniku ustvari dogodek, ki ga lahko v programskem jeziku (najpogosteje C#) prestrežemo in obdelamo. Ta zasnova omogoča preprost način dela, saj se logika aplikacije v glavnem razvija okoli dogodkov in metod, ki se izvedejo, ko uporabnik klikne, vpiše besedilo ali sproži druge dogodke.

### 2.3.3 Uporabniški vmesnik in delovanje

Uporabniški vmesnik v Windows Forms se razvija večinoma s pomočjo Visual Studio Designerja. Razvijalec ali oblikovalec lahko z metodo »povleci in spusti« (drag and drop) vstavi želene kontrolnike na obrazec, prilagaja njihove lastnosti, spreminja velikosti ter jih poljubno razporeja. Visual Studio pri tem v ozadju samodejno generira posebno datoteko, ki skrbi za inicializacijo in nastavitve vseh kontrolnikov. Tako je oblikovanje vmesnika ločeno od glavnega programskega dela, čeprav se vse skupaj na koncu prevede v enoten .NET razred. Dodatno se

---

<sup>3</sup> (Windows Forms documentation, 2023)

pri večjih projektih pogosto uporablja tudi datoteka z viri (resx), v kateri se shranjujejo napisi, slike ali druge lokalne informacije.

#### **2.3.4 Razvojne prakse**

Ker Windows Forms ne temelji na deklarativnem označevalnem jeziku, kot ga ponuja WPF s svojim XAML, se lahko logika in koda za oblikovanje hitro prepleteta, če razvijalci niso pozorni na vzdrževanje jasne strukture. Kljub temu obstajajo dobro uveljavljene prakse za ločevanje poslovne logike od logike vmesnika, na primer z uporabo arhitekturnih vzorcev, kot so MVP (Model-View-Presenter) ali MVVM (Model-View-ViewModel). Čeprav so takšni pristopi bolj pogosto povezani z WPF, se jih da z nekaj truda uvesti tudi v WinForms, s čimer zagotovimo boljše preglednost in vzdržljivost kode. V podjetjih, kjer se WinForms uporablja že dolgo, obstaja vrsta internih in knjižničnih razširitev, ki poenostavijo ali standardizirajo razvoj uporabniškega vmesnika.

### **3 Videoigre**

Videoigre so danes pomemben del računalniške industrije, saj vplivajo na številne tehnološke in kulturne vidike našega vsakdana. Od svojega nastanka so videoigre prešle dolgo pot, od preprostih 2D iger na začetkih računalništva do kompleksnih tridimenzionalnih svetov, ki jih danes poznamo. Pomembnost videoiger v računalniški industriji se ne kaže le v njihovem komercialnem uspehu, temveč tudi v tehnološkem napredku, ki ga prinašajo, ter v vplivu, ki ga imajo na druge industrije, kot so umetnost, zabava in izobraževanje.

Ena izmed ključnih značilnosti videoiger je njihova sposobnost spodbujanja razvoja novih tehnologij in inovacij. Na področju grafike, na primer, so videoigre ena izmed glavnih spodbujajočih sil za razvoj naprednih grafičnih kartic in procesorjev, saj zahteva natančno in hitro obdelavo vizualnih informacij. Sodobne igre se naslanjajo na tehnologije, kot so ray tracing, real-time rendering in AI-driven gameplay, ki so izredno zahtevne, a omogočajo izjemno realistične in dinamične izkušnje. Te napredne tehnologije se nato pogosto prenašajo v druge industrije, kot so filmska produkcija, arhitektura, medicina in avtomobilska industrija, kjer se uporabljajo za simulacije, vizualizacije in modeliranje. Poleg tehnološkega napredka so videoigre pomembne tudi kot pomemben kulturni fenomen. Zaradi

svojega medijskega dosega in vpliva na milijone uporabnikov po vsem svetu so videoigre postale način izražanja umetnosti, zgodb in čustev, ki sega čez zgolj zabavo. Igralci lahko postanejo del globoko razvitih zgodb, ki vključujejo morala vprašanja, politične teme in etične dileme, kar omogoča ustvarjanje vsebin, ki se ukvarjajo z resnimi in pomembnimi temami. Ta preplet umetnosti in tehnologije omogoča videoigram, da se razvijejo v pomemben kulturni medij.

Računalniška industrija je tako v veliki meri povezana z razvojem videoiger, saj te sprožajo potrebo po naprednih algoritmih, bolj zmogljivih računalnikih, večji pasovni širini in naprednih platformah za distribucijo. Hkrati pa videoigre služijo kot testno okolje za nove tehnologije, ki kasneje najdejo uporabnost tudi v drugih panogah. Razvoj videoiger torej ni le zabaven in kreativen proces, temveč pomemben del tehnološkega napredka in gospodarskega razvoja v računalniški industriji.

Videoigre so prav tako postale ključni del digitalne zabavne industrije, kjer spodbujajo različne poslovne modele, kot so nadstandardne igre in igranje v oblaku. Te novosti so omogočile širšo dostopnost iger, povečanje uporabniške baze ter razvoj novih poslovnih priložnosti. Z rastjo mobilnih iger, iger v oblaku in e-športa je videoigra postala globalno prevladujoč del zabavne industrije, kar vpliva na oblikovanje prihodnosti digitalnih medijev.



Slika 3: Prikaz 2d igre

## **4 Trenutne značilnosti igralnih pogonov**

Razvoj igralnih pogonov se je začel v 90-ih letih prejšnjega stoletja, ko so igre postale vse bolj kompleksne in so zahtevale napredne sisteme za obdelavo grafike, fizike, zvoka in umetne inteligence. Ena izmed ključnih prelomnic je bil id Tech 1, ustvarjen za igro Doom leta 1993, ki je omogočil naprednejšo obdelavo grafike in ustvarjanje 3D okolij. Kasneje sta pogona, kot sta Unreal Engine in Unity, prinesli pomembne izboljšave na področju zmogljivosti, dostopnosti in fleksibilnosti, kar je omogočilo širšo uporabo teh platform za razvoj iger vseh vrst. Z napredkom računalniške strojne opreme, zlasti grafičnih procesorjev (GPE) in več jedrnih procesorjev (CPE), so igralni pogoni postali ključna orodja za ustvarjanje realističnih svetov in naprednih igralnih izkušenj. Danes so ti pogoni povezani z umetno inteligenco in algoritmi, ki omogočajo bolj realistično obnašanje NPC "lik, ki ni igralec", dinamično prilagajanje igre in boljšo optimizacijo. Razvoj tako še naprej napreduje in se prilagaja hitremu tehnološkemu napredku, kar omogoča ustvarjanje vedno bolj kompleksnih in interaktivnih iger.

## **5 Kultura in etična vprašanja v igrah**

Poleg tehnoloških vidikov razvoja iger pa je pomembno poudariti tudi kulture in etična vprašanja, ki jih igre sprožajo. Videoigre niso le zabava, ampak postajajo tudi močan medij za izražanje idej, zgodb in tem, ki vključujejo resne moralne in politične dileme. Zgodbe, ki se ukvarjajo z vojno, etiko in okolijsko odgovornostjo, omogočajo igralcem, da se soočajo z realnimi vprašanji skozi interaktivne in včasih čustveno intenzivne izkušnje. Vendar pa igre prinašajo tudi etična vprašanja v povezavi z zasvojenostjo, nasiljem in vplivom, ki ga imajo na mlade igralce. Zato je v industriji vedno večji poudarek na odgovornem razvoju in vključevanju mehanizmov, ki omogočajo zaščito igralcev pred potencialnimi nevarnostmi.

## **6 Izdelava našega igralnega pogona**

Pri razvoju našega 2D-igralnega pogona smo ustvarili dve različici, ki sta se razvijali glede na naše potrebe in željo po izboljšanju uporabniške izkušnje. Prva različica je bila brez grafičnega vmesnika medtem ko smo ga v drugi različici dodali in nam je olajšal delo in omogočil hitrejši razvoj iger.

## 6.1 Prva verzija – upravljanje samo prek kode

Prva verzija našega pogona je bila zasnovana tako, da je bilo celotno upravljanje igre mogoče le prek kode. To pomeni, da ni bilo nobenega vizualnega urejevalnika, zato smo morali vse elemente igre ročno definirati v programski kodi. Vsak objekt, ki smo ga želeli dodati na sceno, je bilo treba ustvariti z ukazi in mu določiti lastnosti, kot so položaj, velikost in obnašanje.

Takšen pristop je imel določene prednosti, saj smo imeli popoln nadzor nad delovanjem pogona in ni bilo dodatnega grafičnega sloja, ki bi lahko vplival na zmogljivost. Kljub temu pa se je kmalu pokazalo, da je takšen način dela manj intuitiven in da zahteva veliko časa za testiranje in prilagajanje. Vsaka sprememba je namreč zahtevala ponovni zagon programa, kar je upočasnjevalo razvoj.

V prvi verziji smo razvili osnovne funkcionalnosti, kot so premikanje objektov, risanje na zaslon in osnovna fizika. Pogon je že omogočal upravljanje igre, vendar je bilo vse delo odvisno od programiranja, kar pomeni, da je bil dostopen predvsem tistim, ki so imeli znanje programiranja. Čeprav je bil učinkovit in zmogljiv, smo hitro ugotovili, da potrebujemo bolj intuitiven način urejanja igre, zato smo začeli delati na drugi verziji.

### 6.1.1. Povzetek prve verzije igralnega pogona

V prvi različici našega igralnega pogona je celotno upravljanje igre potekalo izključno prek kode, brez grafičnega uporabniškega vmesnika. Slika prikazuje delovanje preprostega 2D-igralnega okolja, kjer so prikazani naslednji elementi:

- **Upravljanje objektov prek kode** – Vsi elementi igre, vključno z igralcem in drugimi objekti, so definirani in manipulirani neposredno prek programske kode.
- **Risanje na zaslon** – Pogon omogoča risanje osnovnih grafičnih elementov, kot so liki, ozadja in druge komponente igre.
- **Fizika in premikanje** – Objekti v igri se premikajo na podlagi določenih pravil, pri čemer je vidna osnovna fizika, kot so gravitacija in kolizije.
- **Testno okolje** – Prikazan je preprost primer igre, kjer se igralec premika po 2D-sceni, kar dokazuje, da pogon omogoča osnovno interakcijo in igranje.

Ta različica je bila sicer funkcionalna, vendar je zahtevala popolno ročno konfiguracijo prek kode, kar pomeni, da je bil razvoj iger nekoliko bolj zahteven in manj intuitiven. To je bil glavni razlog, zakaj smo se odločili za nadgradnjo in dodajanje grafičnega uporabniškega vmesnika v drugi verziji pogona, saj naša hipoteza predvideva, da bo naš pogon enostavnejši od drugih, kar pa zahteva uporabniški vmesnik.

```
var window = new Window("Flappy // TJ", 1280, 720, false, false);
var keyboard = new Keyboard();
var graphics = new Graphics();
var camera = new Camera(window);
var renderer = new Renderer(camera, window);
var world = new World(window, keyboard, graphics, camera, renderer)
    .Spawn<Events>(out var events)
    .Spawn<Player>(out var player)
    .Spawn<PipeSpawner, Player>(player, out var spawner)
    .Spawn<Prop, PropConfig>(new("Assets/bg_sky.png".Find(), new(1280, 720), new(1f, -5))
    .Spawn<Transition>(out var transition)
    .OnStart(world =>
    {
        BG.Create(window, world, "bottom", -2, new(1280, 277), 1f, out var bg1);
        BG.Create(window, world, "middle", -3, new(1280, 434), 0.5f, out var bg2);
        BG.Create(window, world, "top", -4, new(1280, 425), 0.2f, out var bg3);
        bg1.Concat(bg2).Concat(bg3).ToList().ForEach(bg =>
        {
            events.OnStart += bg.Start;
            transition.OnCover += bg.Reset;
        });
        events.OnReset += transition.Do;
        events.OnStart += player.Start;
        events.OnStart += spawner.Start;
        events.OnReset += player.Stop;
        events.OnReset += spawner.Stop;
        transition.OnCover += player.Reset;
        transition.OnCover += spawner.Reset;
        transition.OnFinish += events.AllowStart;
        player.OnDie += events.Reset;
    });
    new Game(world, window, keyboard, graphics, renderer).Start();
```

Slika 4: Upravljanje prek kode

## 6.2 Druga verzija – dodan grafični uporabniški vmesnik (GUI)

Z namenom izboljšanja uporabniške izkušnje in pospešitve razvoja smo se odločili nadgraditi pogon z grafičnim uporabniškim vmesnikom. Dodali smo urejevalnik kode v grafičnem vmesniku, nekaj primerov generiranja kode s klikom na gumb, možnost nalaganja datotek, shranjevanja datotek preko vmesnika in odpiranje že narejenih datotek. Ta pristop je nekakšna mešanica obeh svetov – grafičnega in programerskega, saj imamo v grafičnem vmesniku še vedno sistem pisanja koda, ki pa je nekoliko lažji in imamo orodja, ki nam pri tem pomagajo.

Z novo verzijo pogona lahko uporabniki enostavno spreminjajo lastnosti objektov, kot so položaj, velikost in rotacija, brez potrebe po ponovnem zaganjanju programa. Še vedno nam dovoli precej svobode, zaradi dostopa do dejanske kode. Hitreje je tudi nalaganje datotek.

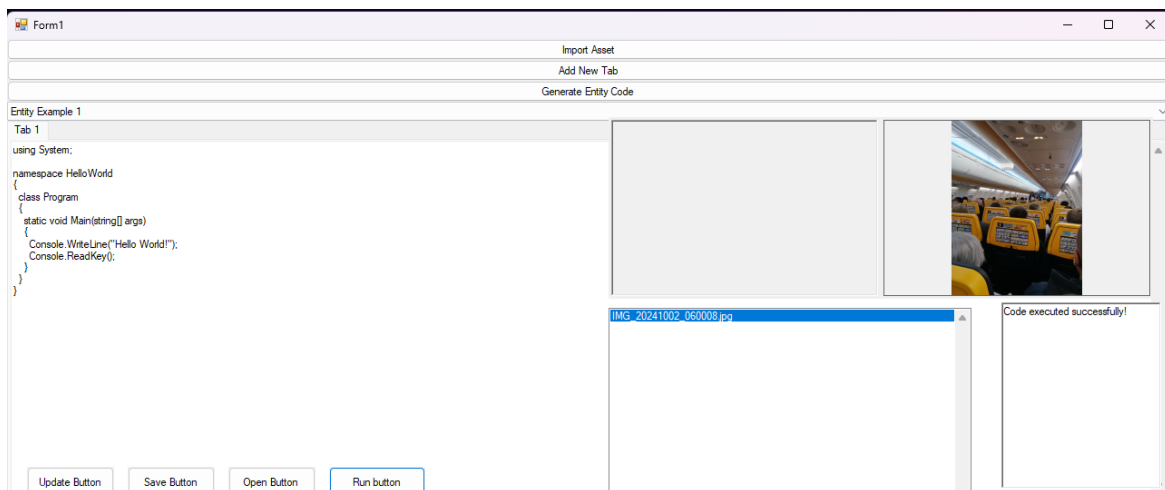
Prehod iz popolnoma kodnega pristopa na vizualno urejanje je bil velik korak naprej, saj smo s tem omogočili bolj dostopno in hitrejšo izkušnjo pri delu z našim

pogonom, hkrati pa ohranili nekaj funkcionalnosti programiranja s kodo. Programerji še vedno lahko uporabljajo kodo za naprednejše funkcionalnosti, vendar je zdaj mogoče veliko stvari urejati tudi brez potrebe po neposrednem pisanju kode. S tem je pogon postal bolj uporaben tudi za tiste, ki niso izkušeni programerji, vendar želijo ustvarjati igre.

Razvoj pogona se nadaljuje, saj želimo dodati še več funkcionalnosti in izboljšav, ki bodo omogočile še enostavnejše ustvarjanje 2D-iger. Naš cilj je, da postane pogon čim bolj intuitiven in učinkovit, hkrati pa še vedno dovolj zmogljiv, da omogoča ustvarjanje kompleksnejših projektov.

## 7 Grafični vmesnik

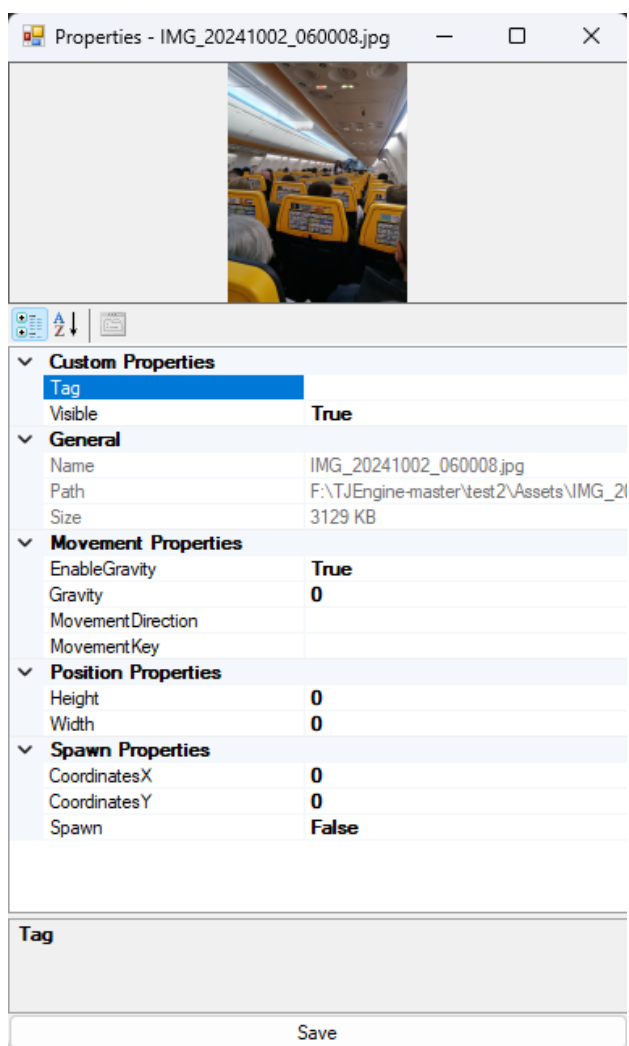
Po dodajanju osnovnega grafičnega vmesnika (GUI) želimo v nadaljevanju nadgraditi urejevalnik scene, da bo še bolj interaktiven in uporabniku prijazen. Načrtujemo izboljšave, kot so možnost direktnega spreminjanja lastnosti objektov, uporaba slojev za organizacijo scene in boljša vizualna predstavitev entitet. Poleg tega želimo izboljšati intuitivnost urejevalnika, da bo primeren tudi za tiste, ki nimajo veliko izkušenj s programiranjem.



Slika 5: Grafični vmesnik

V zgornjem delu so trije gumbi: "Import Asset", "Add New Tab" in "Generate Entity Code". Ti omogočajo uvoz sredstev (npr. slik in drugih datotek), dodajanje novih zavihkov in generiranje kode za entitete. Pod gumbi se nahajajo zavihki, za urejanje kode. Zavihki se lahko preimenujejo z dvoklikom in zaprejo z desnim

klikom na miški. Pod urejevalnikom so štiri gumbi: "Update Button", "Save Button", "Open Button" in "Run Button", ki omogočajo posodabljanje, shranjevanje, odpiranje in izvajanje kode. Vključen je tudi seznam dodanih datotek, kjer lahko vidimo njihov predogled. Desno spodaj je izpis, ki prikaže ali se je koda pravilno zagnala. Celoten vmesnik deluje kot razvojno okolje, kjer lahko uporabnik piše in izvaja kodo, uvaža datoteke in dela s sredstvi za igro.



Slika 6: lastnosti objektov

Okno lastnosti vsebuje več razdelkov, ki prikazujejo informacije o objektu. Med njimi so splošni podatki, kot so ime datoteke, njena pot in velikost. Poleg tega so na voljo različne lastnosti, povezane z gibanjem, položajem in ustvarjanjem objekta v prostoru. Razdelek "Movement Properties" vsebuje nastavitve, povezane z gravitacijo in premikanjem objekta. "Position Properties" prikazuje dimenzije, medtem ko "Spawn Properties" določa začetne koordinate in ali naj se objekt

ustvari v sceni. V zgornjem delu okna je prikazana slika objekta, ki je del projekta. Spodaj je vnosno polje za oznake ter gumb za shranjevanje sprememb.

## 8 Fizika

Fizikalni sistem je v našem igralnem pogonu že dobro razvit in zagotavlja vse bistvene elemente za izdelavo iger. Trenutno so že implementirane funkcionalnosti, kot so osnovno zaznavanje trkov, gravitacija, posodabljanje položajev objektov skozi čas ter osnovna simulacija gibanja. Sistem podpira delo s fizikalnimi lastnostmi, kot so hitrost, pospešek in masa, kar razvijalcem omogoča realistično simulacijo gibanja in interakcij med objekti.

Obstoječi sistem že omogoča zaznavanje trkov med objekti, kar je dovolj za enostavnejše igre in prototipe. Prav tako je vgrajena osnovna logika za obravnavo odzivov na trke – na primer sprememba smeri gibanja ob stiku z drugim objektom. Osnovna gravitacija deluje tako na igralne like kot na druge objekte, kar že ustvari občutek naravnega gibanja.

Za še večjo natančnost in realističnost fizikalnega sistema načrtujemo nadgradnje, ki bodo izboljšale interakcije med objekti. Med njimi je uvedba naprednejšega zaznavanja trkov, vključno z ločenima fazama širokega in ozkega zaznavanja, kar bo zmanjšalo število nepotrebnih izračunov in povečalo zmogljivost pri kompleksnejših scenah.

## 9 Scene

V razvoju iger scena predstavlja osnovno enoto igre, ki vsebuje vse potrebne objekte, kot so liki, okolje, elementi in skripte za upravljanje interakcij. Vsaka scena lahko predstavlja določen nivo, meni ali katerikoli drug del igre, ki ga je mogoče ločeno upravljati in naložiti.

Naš cilj je omogočiti boljše upravljanje s prizorišči (scenami) znotraj igre. Želimo dodati možnost dodajanja scen in preprostega prehajanja med njimi, nalaganje različnih segmentov igre ter podporo za shranjevanje in nalaganje stanja scene. S

tem bo mogoče ustvarjati kompleksnejše igre z več nivoji in interaktivnimi elementi, kar bo povečalo uporabnost pogona.

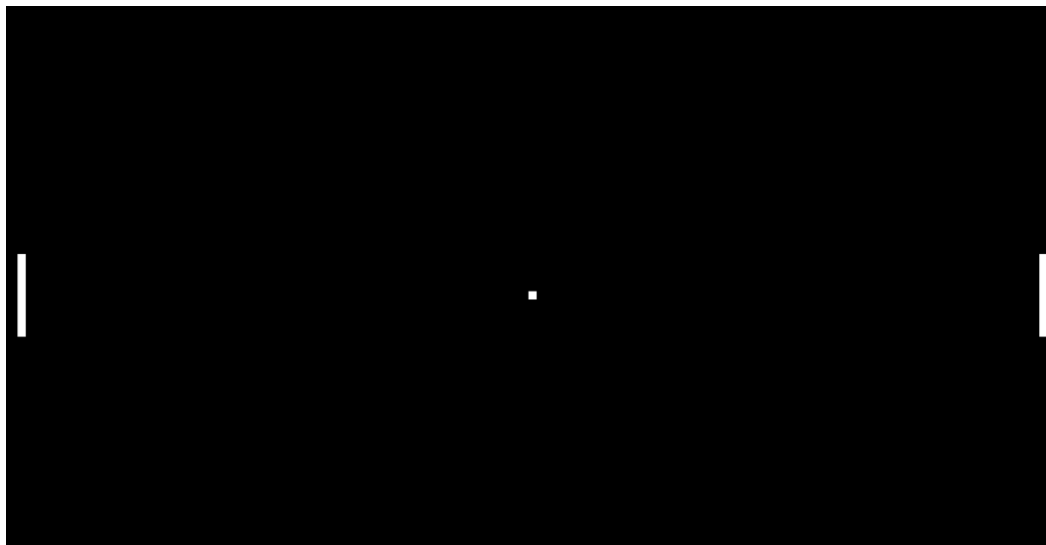
## 10 Animacije

Trenutno pogon omogoča osnovno risanje objektov, vendar želimo dodati podporo za animacije, ki bodo igralcem omogočile bolj dinamične in vizualno privlačne igre. Načrtujemo sistem za preprosto dodajanje animacijskih sličic, možnost predogleda animacij v grafičnem vmesniku in nadzor nad hitrostjo ter prehodi med animacijskimi stanji. S tem bo mogoče ustvarjati bolj tekoče in živahne like, ki bodo povečali kakovost igre.

## 11 Izdelava igre v našem igralnem pogonu

### 11.1 Pong

Ko smo dokončali naš igralni pogon, smo se usmerili v preprosto igro, s katero bi ga preizkusili. Prva misel je bila pong – igra, ki jo vsi dobro poznamo in je ena najstarejših iger sploh, s tem pa tudi ena najenostavnejših. Lotili smo se z ustvarjanja entitet dveh igralcev, nato pa še posebne entitete za žogo, ki ima spremenljivko za hitrost, ki se povečuje. Nato smo ločili entitete za dva igralca, za eno smo implementirali vhode tipkovnice in jih povezali s premikanjem, za drugega pa smo naredili samodejno premikanje vzporedno z žogo.



Slika 7: Izrez igre pong v našem pogonu

## 11.2 Flappy bird

Po uspešno ustvarjenem Pongu, smo iskali nekaj malo bolj zahtevnega, a še vedno nekaj poznanejšega, zato smo se odločili za Flappy bird. Tu smo se malo več osredotočili na slike(sprite), ki smo jih dodali v projekt, nato pa jih povezali z entitetami. Eno z glavno entiteto – igralcem in ostale z ovirami na poti, nazadnje pa še za ozadje. Vključili smo spremenljivko za gravitacijo in simulirali premike tako, da smo spreminjali pozicijo slik na ozadju. Trke smo preverjali s prekrivanjem slik, kar je že vgrajena funkcija v našem pogonu. Nato smo poskrbeli za ustvarjanje in uničevanje ovir, da program ni porabil preveč pomnilnika in imeli smo svojo igro mladosti.



Slika 8: Izrez igre Flappy bird v našem pogonu

## 12 Pregled že obstoječih igralnih pogonov

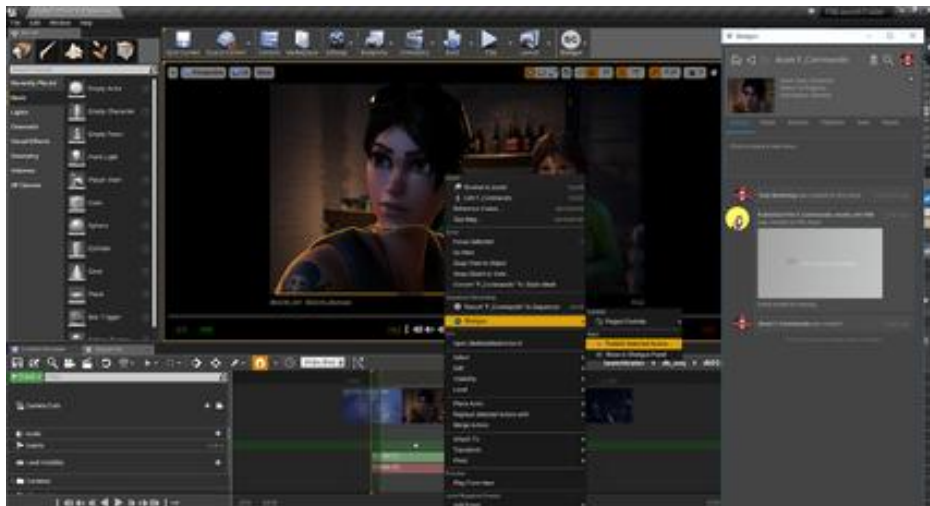
Danes obstaja več zmogljivih in priljubljenih igralnih pogonov, ki omogočajo razvoj iger za različne platforme. Med najbolj uveljavljenimi so Unity, Unreal Engine in Godot, ki jih uporabljajo tako neodvisni razvijalci kot tudi velika podjetja v industriji videoiger.

### 12.1 Unreal Engine

Unreal Engine je programsko orodje za ustvarjanje videoiger, ki ga je razvilo podjetje Epic Games. Prvič je bil predstavljen leta 1998 v prvoosebni strelski igri *Unreal*. Čeprav je bil sprva zasnovan za strelske igre, se je sčasoma razširil na

različne žanre, vključno z borilnimi igrami, prikritimi igrami, MMORPG-ji in igrami vlog.

Unreal Engine je napisan v programskem jeziku C++, kar omogoča njegovo prilagodljivost in prenosljivost. Zaradi svoje zmogljivosti in dostopnosti izvorne kode ga uporablja veliko razvijalcev po vsem svetu. Zadnja izdana različica je Unreal Engine 5, ki je izšla leta 2025.<sup>4</sup>



Slika 9: Izgled Unreal Engin z različnimi okni

## 12.2 Unity

Unity je programska oprema za razvoj iger na več platformah, ki jo je razvilo podjetje Unity Technologies. Prvič je bila predstavljena junija 2005 na svetovni konferenci razvijalcev podjetja Apple kot orodje za razvoj iger na operacijskem



Slika 10: Logotip Unity

sistemu OS X. Sčasoma se je razširila in do leta 2018 podpirala kar 27 različnih platform. Z Unityjem je mogoče ustvarjati tako 2D kot 3D igre ter različne

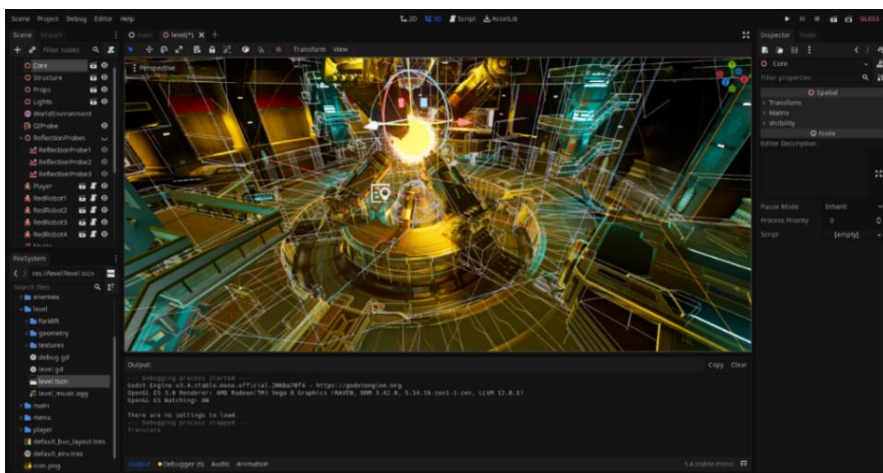
---

<sup>4</sup> (Unreal Engine 5, 2023)

simulacije. Od njegovega izida so razvijalci izdali več pomembnih posodobitev, pri čemer je zadnja stabilna različica Unity 6, ki je izšla 17. oktobra 2024.<sup>5</sup>

## 12.3 Godot

Godot je večplatformski, brezplačen in odprtokodni pogon za razvoj iger, ki je na voljo pod licenco MIT. Razvila sta ga argentinska razvijalca Juan Linietsky in Ariel Manzur v Buenos Airesu, sprva za različna podjetja v Latinski Ameriki, nato pa je bil leta 2014 javno objavljen. Zadnja različica, GoDot 4.4, pa je bila objavljena marca 2025. Razvojno okolje Godot deluje na različnih operacijskih sistemih in omogoča izvoz iger na številne platforme.<sup>6</sup>



Slika 11: Izgled Godota

## 12.4 Zakaj izdelati lasten igralni pogon namesto uporabe obstoječega?

Čeprav obstoječi igralni pogoni ponujajo izjemno zmogljivost in široko podporo, obstajajo razlogi, zakaj bi nekdo želel razviti svoj lasten igralnega pogona

- Popoln nadzor nad kodo – Nobenih omejitev in nepotrebnih funkcij, ki bi lahko vplivale na zmogljivost igre.
- Učenje in razumevanje tehnologije – Razvoj lastnega igralnega pogona je odličen način za globlje razumevanje upodabljanje, fizike in arhitekture programske opreme.
- Optimizacija za specifične potrebe – Lastni pogon omogoča prilagoditev za specifične projekte, kjer obstoječi pogoni vsebujejo preveč nepotrebnih funkcij.

<sup>5</sup> (Unity (game engine), 2025)

<sup>6</sup> (Godot (game engine), 2025)

- Brez licenčnih stroškov in provizij – Uporaba komercialnih pogonov lahko zahteva plačila licence ali del dobička.

Seveda je izdelava lastnega igralnega pogona zelo zahteven proces, ki terja ogromno časa in znanja, zato večina razvijalcev raje uporablja obstoječe pogone. Kljub temu pa je razvoj lastnega pogona lahko izjemno koristna izkušnja za programerje in razvijalce iger.

## **12.5 Primerjava funkcionalnosti igralnih pogonov**

### **12.5.1. Enostavnost uporabe**

Unity je intuitiven urejevalnik, ki je primeren tako za začetnike kot profesionalce. Podpira programski jezik C#, vendar zahteva dobro poznavanje objektno usmerjenega programiranja (OOP). Vsebuje vizualna orodja za animacijo in fiziko, kar olajša delo razvijalcem, poleg tega pa omogoča uporabo številnih vtičnikov in orodij za avtomatizacijo razvoja.

Unreal Engine je bolj kompleksen pogon, ki zahteva več tehničnega znanja. Uporablja C++ in Blueprints, kar omogoča hitro izdelavo prototipov brez programiranja. Primeren je predvsem za velike ekipe in projekte s kompleksno grafiko, saj vsebuje napredna orodja za simulacijo realističnih fizikalnih učinkov in osvetlitve.

Godot ponuja preprost vmesnik, ki je lažji za začetnike. Podpira več programskih jezikov, vključno z GDScriptom, ki je podoben Pythonu, C# in C++. Zaradi modularne arhitekture omogoča prilagoditev delovnega okolja glede na potrebe razvijalca, kar ga naredi idealnega za hitro izdelavo prototipov in enostavnih iger.

TJ ima urejevalnik z možnostjo pisanja kode v C# programskem jeziku, ki pa še ni na standardu ostalih. Želimo si ga precej izboljšati, smo ga pa razvili do točke uporabnosti in je usmerjen v intuitivno uporabo.

### **12.5.2. Zmogljivost**

Unity ponuja dobro optimizacijo tako za 2D kot 3D igre, saj ima prilagodljive grafične nastavitve, vključno z URP in HDRP. Deluje na različnih platformah,

vključno z mobilnimi napravami, konzolami in osebni računalniki, ter nudi močno podporo za VR in AR aplikacije.

Unreal Engine je odličen za visoko zmogljive 3D igre, vendar je preveč zahteven za enostavne 2D igre. Zahteva močnejšo strojno opremo, a hkrati ponuja napredno optimizacijo s tehnologijama Nanite in Lumen, kar omogoča fotorealistično vizualizacijo in simulacije.

Godot je optimiziran za 2D igre in deluje zelo hitro tudi na šibkejših napravah. Ima manj zahtevne sistemske zahteve v primerjavi z Unityjem in Unreal Engineom, kar ga naredi idealnega za indie igre z manjšimi zahtevami. Njegova odprtokodna narava omogoča visoko prilagodljivost.

TJ je dobro optimiziran za 2D igre saj je namenjen le za njih in je s tem manj nepotrebne opreme, ki jo imajo drugi pogoni. Takoj deluje tudi na slabših napravah vendar je tudi manj zmogljiv z manj funkcijami.

### **12.5.3. Lastnosti**

Unity je močan 2D in 3D pogon, ki podpira tilemaps, sprite atlase in UI sisteme. Ima napredno fiziko s podporo za Box2D in PhysX, ponuja veliko razširitev ter dostop do Asset Store-a. Prav tako podpira večigralske igre in povezovanje s strežniškimi sistemi.

Unreal Engine izstopa po vrhunskih 3D grafičnih zmogljivostih, kot sta ray tracing in real-time global illumination. Blueprint sistem omogoča hitro vizualno programiranje, močan AI sistem pa omogoča napredno obnašanje NPC-jev. Poleg tega nudi napredno upravljanje animacij in materialov.

Godot je najboljša izbira za 2D igre, saj nativno podpira tilemaps, pathfinding in sprite atlase. Ima lahek vgrajen fizikalni pogon, je popolnoma brezplačen in odprtokoden, kar omogoča razvijalcem, da prilagodijo pogon svojim potrebam ter prispevajo k njegovemu razvoju.

TJ je osredotočen na 2D igre in omogoča delo s spriti in ima implementirano zaznavanje trkov, nima pa nepotrebnih kompleksnosti, kar ga naredi privlačnega za novince, ki jih preveč informacij ali kompleksnost velikokrat odžene.

#### **12.5.4. Skupnost in podpora**

Unity ima ogromno skupnost razvijalcev, obsežno dokumentacijo ter veliko video vodičev. Aktivni forumi in Discord skupnosti zagotavljajo odlično podporo, redni razvoj in nadgradnje pa zagotavljajo dolgoročno vzdrževanje pogona.

Unreal Engine ima močno podporo, predvsem v AAA industriji. Uradni vodiči in vadnice s strani Epic Games so izčrpani, vendar se pogon manj osredotoča na 2D razvoj. Njegov sistem za verzioniranje projektov je prilagojen velikim ekipam.

Godot ima manjšo, a zelo aktivno indie skupnost. Njegova odlična dokumentacija in številni odprtokodni prispevki omogočajo hitro učenje. Ker je odprtokoden, se hitro razvija in izboljšuje.

TJ še nima skupnosti saj smo za njim le trije razvijalci, vendar se trudimo, da bi izdali čim boljše dokumentacijo in da bi razširilo našo skupnost, ko bo pogon nared za večjo publiko.

#### **12.5.5. Primeri uporabe**

Unity je najboljša izbira za mobilne, indie in srednje velike igre. Pogosto se uporablja za 2D platformerje, puzzle igre in strategije, pri čemer so znani primeri Hollow Knight, Cuphead in Ori and the Blind Forest. Prav tako je priljubljen pri razvoju VR in AR projektov.

Unreal Engine je primarno namenjen 3D AAA igram in je manj primeren za 2D razvoj. Veliko se uporablja pri FPS, RPG in akcijskih igrah, kjer so znani primeri Fortnite, Gears of War in Street Fighter V. Poleg tega se pogosto uporablja v filmski industriji za vizualizacijo.

Godot je odlična izbira za 2D igre in indie razvoj. Hitrejši razvoj in enostavnejše orodje omogočata izdelavo manjših iger, pri čemer so znani primeri Pixelorama,

Brotato in Dome Keeper. Poleg tega se pogosto uporablja pri izobraževalnih in odprtokodnih projektih.

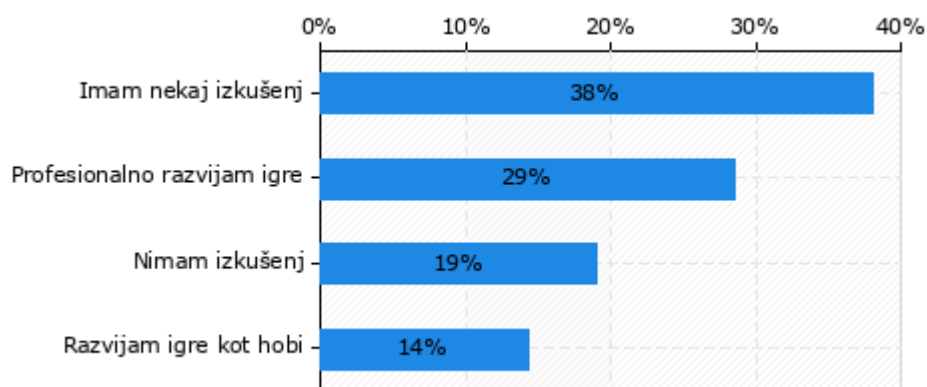
TJ je prilagojen razvoju 2D iger, omogoča hitro izdelavo ter je idealen za neodvisne razvijalce, ki želijo eksperimentirati s svojimi igrami. Je dobra izbira pri izdelavi manjših projektov, ki niso preveč zahtevni in je usmerjen v hitro izdelavo.

## 13 Anketa o igralnem pogonu

Anketa je bila izvedena z namenom pridobitve povratnih informacij o uporabnosti in enostavnosti uporabe našega pogona. Želeli smo preveriti, ali grafični uporabniški vmesnik prispeva k boljši uporabniški izkušnji ter ali naša rešitev dejansko izpolnjuje pričakovanja glede enostavnosti uporabe. V raziskavi je sodelovalo 21 anketirancev, ki so vsi imeli izkušnje z računalništvom, kar nam je omogočilo relevantno in strokovno oceno našega sistema.

### Kakšna je vaša izkušnja z razvojem iger?

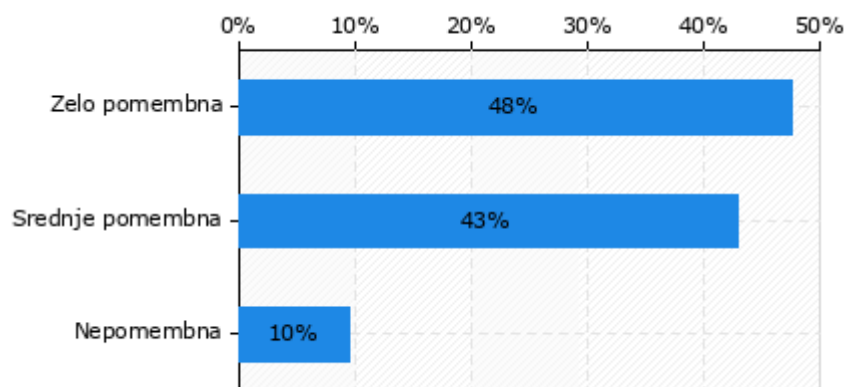
Večina udeležencev ima vsaj nekaj izkušenj z razvojem iger. 8 udeležencev, ima nekaj izkušenj, medtem ko jih 3 razvijajo kot hobi. Le 6 udeležencev igre razvija profesionalno, kar pomeni, da je prehod iz hobija v profesionalno okolje zahteven. 4 udeleženci pa nimajo nobenih izkušenj z razvojem iger.



graf 1:Prikaz izkušenj anketirancev

## Kako pomembna vam je enostavnost pri uporabi igralnih pogonov

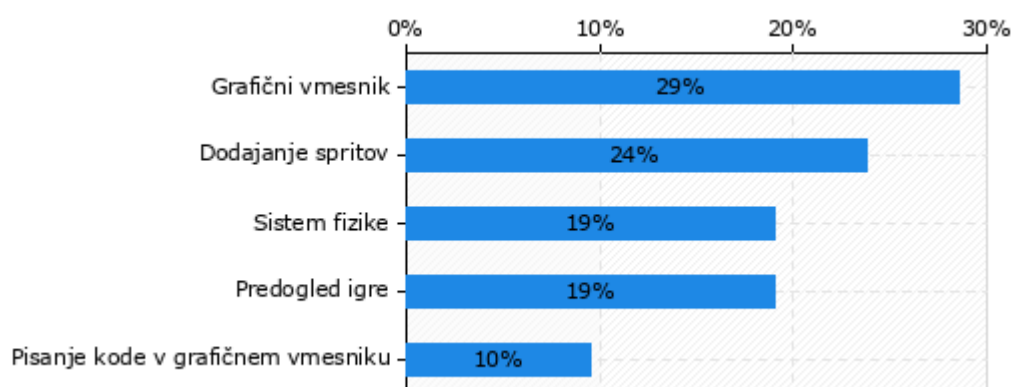
Večina udeležencev meni, da je enostavnost zelo pomembna. Medtem ko jo 9 udeležencev vidi kot srednje pomembno. Le 2 osebi menita, da tema ni pomembna. To pomeni, da večina udeležencev enostavnost vidi kot ključen dejavnik.



graf 2: Prikaz pomembnosti enostavnega pogona

## Kaj v našem igralnem pogonu vam je najpomembnejše

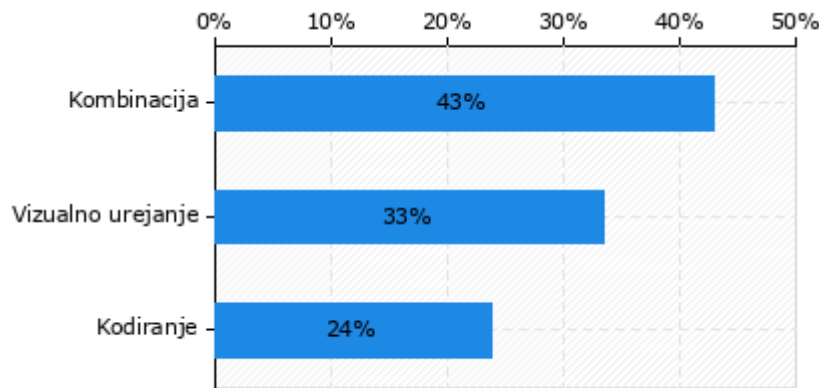
Glede na odgovore je najpomembnejša lastnost igralnega pogona grafični vmesnik, saj ga je izbralo 6 udeležencev. 5 udeležencev meni, da je najpomembnejše dodajanje sličic (spritov). Sistem fizike in predogled igre sta enako pomembna, saj so ju zbrali 4 udeleženci. Najmanj pomembno je pisanje kode v grafičnem vmesniku, kar sta izbrala 2 udeleženca. Večina udeležencev daje prednost vizualnim in interaktivnim funkcijam igralnega pogona.



graf 3: Prikaz najpomembnejših elementov v pogonu

### Kateri način uporabe pogona vam najbolj ustreza

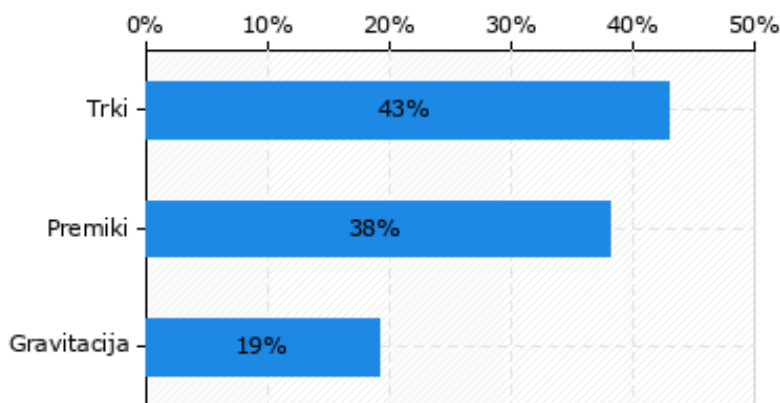
9 udeležencev ima najraje kombinacijo različnih načinov uporabe pogona. 7 udeležencev se najbolj nagiba k vizualnemu urejanju, medtem ko 5 udeležencev raje uporablja kodiranje. To pomeni, da večina udeležencev želi prilagodljivost pri uporabi pogona.



graf 4: Prikaz ustrežnejšega vnosa

### Katere lastnosti fizike ste najbolj uporabljali

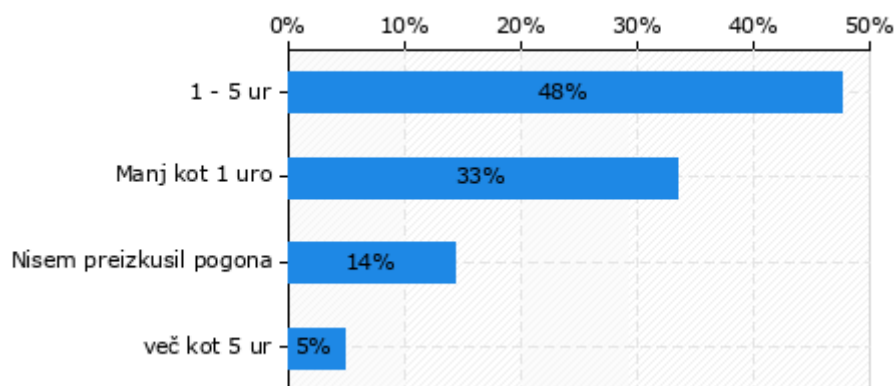
Največ udeležencev je pri uporabi pogona največkrat uporabljalo trke. 8 udeležencev je pogosto uporabljalo premike, medtem ko so 4 udeleženci največkrat uporabljali gravitacijo.



graf 5: Prikaz najbolj uporabljenih lastnosti fizik

### Koliko časa ste potrebovali, da ste se naučili uporabljati naš pogon

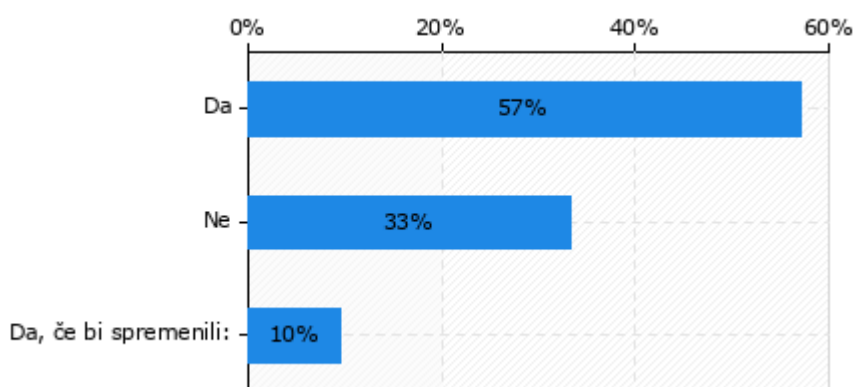
Večina udeležencev je za učenje uporabe pogona potrebovala 1 do 5 ur, kar 10 ljudi. 7 udeležencev je osvojilo osnove v manj kot eni uri, to pomeni, da je pogon enostaven za hiter začetek. Le 1 udeleženec je vložil več kot 5 ur, saj je za poglobljeno znanje potrebna dodatna praksa.



graf 6: Prikaz potrebnega časa

### Bi še naprej uporabljali naš igralni pogon za izdelavo 2D iger

12 udeležencev bi še naprej uporabljala naš igralni pogon za izdelavo 2D iger. 7 udeležencev se je odločilo, da ga ne bi več uporabljalo. 2 udeleženca bi ga sicer uporabljala, vendar le ob določenih spremembah, izpostavila sta grafični vmesnik in lažjo implementacijo metod.



graf 7: Prikaz mnenja o nadaljnji uporabi našega pogona

## 14 Potrjevanje hipotez

1. Naš igralni pogon bo lažji za uporabo in bolj prijazen uporabnikom pri izdelavi platformskih iger kot uveljavljeni pogoni.
  - Rezultat ankete: Večina uporabnikov je izpostavila pomembnost enostavnosti in uporabnosti GUI-ja v našem pogonu. Večina jih je tudi hitro osvojila uporabo pogona (1-5 ur) in dobra polovica bi jih še naprej uporabljala naš pogon.
  - Zaključek: Hipoteza je delno potrjena – pogon je uporabnikom dokaj prijazen, vendar še vedno ne dosega stopnje intuitivnosti večjih pogonov, kot so Unity ali Godot. Vzorec anketirancev je bil narejen na zelo majhnem številu udeležencev, zato si želimo enostavnost še izboljšati.
2. Že obstoječe popularne platformske igre bodo lahko narejene v našem igralnem pogonu.
  - Testi: Uspeli smo ustvariti preproste igre, kot sta Pong in Flappy Bird.
  - Zaključek: Hipoteza je delno potrjena, saj so preproste igre mogoče, a za kompleksnejše bi bile potrebne dodatne funkcionalnosti. Za večjo uporabnost pogona bi potrebovali več naprednih orodij in prilagoditev, ki bi omogočile razvoj zahtevnejših iger.
3. V našem igralnem pogonu bodo lahko ustvarjali ljudje z amaterskim znanjem ustvarjanja video iger.
  - Rezultat ankete: Večina anketiranih je pogon hitro osvojila.
  - Zaključek: Hipoteza je potrjena – pogon je dovolj enostaven za začetnike.
4. Igralni pogon je možno narediti brez stroškov in s trenutnim srednješolskim znanjem.
  - Dejanski razvoj: Pogon je bil razvit brez finančnih stroškov in temelji na znanju srednješolskega nivoja.
  - Zaključek: Hipoteza je potrjena.

## 15 Zaključek in smernice za nadaljnjo delo

V raziskovalni nalogi smo preučili postopek izdelave lastnega igralnega pogona in ga primerjali z obstoječimi rešitvami, kot so Unity, Unreal Engine in Godot. Naš glavni cilj je bil ustvariti uporabniku prijazen 2D igralni pogon, ki bi omogočal izdelavo iger tudi tistim brez naprednega programerskega znanja. Rezultat našega dela sta dve različici pogona – ena temelji izključno na kodi, druga pa vključuje grafični uporabniški vmesnik, ki olajša razvoj iger.

Analiza in testiranje sta pokazala, da je mogoče razviti funkcionalen igralni pogon brez finančnih stroškov in z uporabo srednješolskega znanja programiranja. Kljub temu ima takšen pogon določene omejitve v primerjavi z uveljavljenimi rešitvami, predvsem na področju zmogljivosti in uporabniške izkušnje. Razvoj lastnega pogona se je izkazal kot dragocena izkušnja, saj omogoča poglobljeno razumevanje delovanja igralnih mehanik, fizike in optimizacije.

Na podlagi pridobljenih rezultatov smo ugotovili, da je bil igralni pogon uporabnikom všeč. Kljub temu predlagamo nadaljnji razvoj pogona z izboljšanjem grafičnega vmesnika, optimizacijo fizikalnega sistema in uvedbo podpore za animacije. Poleg tega bi bila smiselna širša uporabniška testiranja, ki bi omogočila boljše prilagajanje potrebam razvijalcev. Možnosti za nadaljnje raziskave vključujejo razširitev pogona na več različnih iger, dodajanje skriptnega jezika za večjo prilagodljivost ter implementacijo naprednejših funkcij, kot je več igralska podpora.

S tem projektom smo dokazali, da je mogoče z zagnanostjo, načrtovanjem in eksperimentiranjem ustvariti lasten igralni pogon, ki lahko služi kot osnova za nadaljnji razvoj in izboljšave v prihodnosti.

V nadaljnjem razvoju pogona želimo izboljšati grafični vmesnik, optimizirati fizikalni sistem za natančnejše simulacije ter dodati podporo za animacije, kar bi omogočilo bolj dinamične igre. Poleg tega bi bilo smiselno izboljšati zmogljivost in uporabniško izkušnjo ter preizkusiti pogon na širšem vzorcu uporabnikov. Dolgoročno bi lahko raziskali tudi možnosti več igralske podpore.

## Bibliografija

- 2D GAME ART STYLES: THE ULTIMATE GUIDE*. (14. Junij 2023). Pridobljeno 10. Oktober 2024 iz 3D-ace: <https://3d-ace.com/blog/2d-game-art-styles-the-ultimate-guide/>
- Baggs, N. (10. Junij 2023). *Ditch Unity, Build A Game Engine In 48 Hours*. Pridobljeno 10. Februar 2025 iz YouTube: <https://www.youtube.com/watch?v=v9x9RgYP8vw>
- C Sharp (programming language)*. (2025. Marec 9). Pridobljeno 17. Februar 2025 iz Wikipedia: [https://en.wikipedia.org/wiki/C\\_Sharp\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))
- C# language documentation*. (2. Avgust 2023). Pridobljeno 19. Januar 2025 iz <https://learn.microsoft.com/en-us/dotnet/csharp/>
- C++*. (19. Avgust 2022). Pridobljeno 18. December 2024 iz W3schools: <https://www.w3schools.com/cpp/default.asp>
- Game engine*. (1. Marec 2025). Pridobljeno 5. Marec 2025 iz Wikipedia: [https://en.wikipedia.org/wiki/Game\\_engine](https://en.wikipedia.org/wiki/Game_engine)
- Game Engine Programming 001 - Introduction | C++ Game Engine*. (27. Junij 2022). Pridobljeno 17. Oktober 2024 iz YouTube: <https://www.youtube.com/watch?v=hRL56gXqj-4>
- Games, G. S. (19. April 2023). *So you want to make a game engine!?(watch this before you start)*. Pridobljeno 10. Oktober 2024 iz YouTube: <https://www.youtube.com/watch?v=3rcka6P2cVI>
- Getting started with OpenGL*. (19. September 2023). Pridobljeno 3. Marec 2025 iz geeksforgeeks: <https://www.geeksforgeeks.org/getting-started-with-opengl/>
- Godot (game engine)*. (11. Marec 2025). Pridobljeno 20. Februar 2025 iz [https://en.wikipedia.org/wiki/Godot\\_\(game\\_engine\)](https://en.wikipedia.org/wiki/Godot_(game_engine))
- Is it better for my games to make my own engine first?* (28. November 2024). Pridobljeno 30. Oktober 2024 iz Quora: <https://www.quora.com/Is-it-better-for-my-games-to-make-my-own-game-engine-first>
- Long, C. (3. Marec 2021). *Text Editor: Saving Files in C#*. Pridobljeno 7. Marec 2025 iz YouTube: <https://www.youtube.com/watch?v=PrGT28eFUfw>
- n8dev. (28. April 2022). *I made my own game engine*. Pridobljeno 20. December 2024 iz YouTube: [https://www.youtube.com/watch?v=hysfq\\_xJAw0&t=155s](https://www.youtube.com/watch?v=hysfq_xJAw0&t=155s)
- OpenGL*. (24. Februar 2025). Pridobljeno 1. Marec 2025 iz Wikipedia: <https://en.wikipedia.org/wiki/OpenGL>
- Pass Textbox Value to another Form - C#*. (3. April 2021). Pridobljeno 7. Marec 2025 iz YouTube: <https://www.youtube.com/watch?v=IN2u3wfbZM8>

*Platformer*. (3. Februar 2025). Pridobljeno 20. Februar 2025 iz Wikipedia:  
<https://en.wikipedia.org/wiki/Platformer>

*Sunny land 2d pixel art pack*. (8. November 2020). Pridobljeno 20. December 2024 iz  
<https://opengameart.org/content/sunny-land-2d-pixel-art-pack>

*Unity (game engine)*. (13. januar 2025). Pridobljeno 25. februar 2025 iz  
[https://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))

*Unreal Engine 5*. (7. Junij 2023). Pridobljeno 20. Februar 2025 iz  
<https://www.unrealengine.com/en-US/unreal-engine-5>

*Visual Studio*. (14. December 2022). Pridobljeno 23. Oktober 2024 iz  
<https://visualstudio.microsoft.com/>

*Visual Studio*. (11. Marec 2025). Pridobljeno 17. Februar 2025 iz Wikipedia:  
[https://en.wikipedia.org/wiki/Visual\\_Studio](https://en.wikipedia.org/wiki/Visual_Studio)

*Windows Forms documentation*. (2. Junij 2023). Pridobljeno 16. Februar 2025 iz Microsoft:  
<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-9.0>