SREDNJA POKLICNA
IN TEHNIŠKA ŠOLA
MURSKA SOBOTA

59. Srečanje mladih raziskovalcev Slovenije

Razvoj VR aplikacije za spodbujanje algoritmičnega razmišljanja s pomočjo blokovnega programiranja

Aplikativni inovacijski predlogi in projekti

Šola: **Srednja poklicna in tehniška šola Murska Sobota**

Avtorji: **Tai Lupscha, Tine Muraus, Jure Zrim**

Mentor: **Dominik Letnar**

Leto izdelave: **2024/2025**

Murska Sobota, 2025

KAZALO VSEBINE

POVZETEK 1
SUMMARY 1
1 UVOD 2
2 OPIS PROBLEMA 3
2.1 Metodologija 4
3 TEORETIČNI DEL 5
3.1 Virtualna resničnost 5
3.1.1 Razvoj VR tehnologije skozi zgodovino 5
3.2 Uporabe VR tehnologij 7
3.3 Zahteve za uspešno VR aplikacijo 8
4 PRIPRAVE NA RAZVOJ 9
4.1 Spoznavanje orodij za razvijanje aplikacije 9
4.2 Razvojno okolje Unity 9
4.2.1 Glavne značilnosti razvojnega okolja Unity 9
4.3 Ideja in koncept 10
4.4 Priprava pred izdelavo projekta 12
5 TEHNIČNI DEL 13
5.1 OKOLJE, MODELI IN FUNKCIONALNOSTI 13
5.2 PROGRAMSKA KODA 19
5.2.1 GameManager 19
5.2.1.1 Uvod v delovanje skripte GameManager 19
5.2.1.2 Funkcija StartProgram() – zbiranje in priprava ukazov 19
5.2.1.3 Funkcija RunCommands() – izvajanje ukazov po vrsti 20
5.2.1.4 Funkcija ResetScene() – ponastavitev poligona 20
5.2.1.5 Zaključek stopnje – ocena uspešnosti in prehod v naslednjo stopnjo 21
5.2.1.6 Funkcija Next() – prehod na naslednjo stopnjo ali vrnitev v meni 21
5.2.2 RobotScript – upravljanje robotskega vedenja v igri 21
5.2.2.1 Premikanje robota za izbrano število polj 22
5.2.2.2 Obračanje robota v igri 23
5.2.2.3 Skok robota v igri 23
5.2.2.4 Preverjanje padanja robota 24
5.2.2.5 Prikaz kape in vrtenje propelerja 24
5.2.3 Posebna funkcija – uporaba trampolina za visoki skok 25
6 TESTIRANJE APLIKACIJA NA PROSTOVOLJCIH 26
7 ANALIZA REZULTATOV IN UGOTOVITVE 28
7.1 Rezultati ankete 28
7.2 Potrditev hipotez 34
7.3 Ugotovitve 36
8 ZAKLJUČEK 37
9 VIRI 38
10 PRILOGA 39
10.1 Programska koda 39
10.2 Anketa 48

KAZALO SLIK

Slika 1: VR očala in krmilniki 5
Slika 2: Prvi VR sistem "Swords of Democles" 6
Slika 3: Uporaba VR v vojski 7
Slika 4: Uporaba VR v medicini 7
Slika 5: Zaslonski posnetek, razvojno okolje Unity 10
Slika 6: Prva konceptualna skica (ideja, »brainstorming«) 11
Slika 7: Druga konceptualna skica (pouk programiranja, dodana vrednost) 11
Slika 8: Zaslonski posnetek, okolje Blender 12
Slika 9: Primeri modelov map 13
Slika 10: Začetna verzija aplikacije 13
Slika 11: Osnovni model polja iz začetne verzije aplikacije 13
Slika 12: Modeliran nov model za polje brez teksture v orodju Blender 13
Slika 13: Modeliran nov model za polje s teksturo v orodju Blender 13
Slika 14: Prva stopnja igre 14
Slika 15: Portal za premikanje 14
Slika 16: Model robotka 14
Slika 17: Prikaz ukazov iz razdelka akcije 15
Slika 18: Prikaz ukazov iz razdelka funkcije 15
Slika 19: Meni igre 15
Slika 20: Prikaz časovnih stopenj v meniju 16
Slika 21: Naključno ustvarjena stopnja 16
Slika 22: Delovna miza v stopnji "Naredi sam" 17
Slika 23: Opis programskega bloka "Obrni se" 17
Slika 24: Opis programskega bloka za stavek "ČE" 17
Slika 25: Uspešno končana stopnja "2 zvezdici" 18
Slika 26: Prikaz klasične rešitve stopnje 18
Slika 27: Prikaz napredne rešitve stopnje 18
Slika 28: Uspešno končana stopnja "3 zvezdice" 18
Slika 29: Prikaz nastavitev v slovenskem jeziku 18
Slika 30: Prikaz nastavitev v angleškem jeziku 18
Slika 31: Vizualni prikaz shranjevanja programskih blokov 19
Slika 32: Zaslonski posnetek igre – vizualni prikaz hoje robota 22
Slika 33: Zaslonski posnetek igre - vizualni prikaz obračanja robota 23
Slika 34: Zaslonski posnetek – prikaz skoka robota v igri 23
Slika 35: Zaslonski posnetek – primer uporabe trampolina za skok 23
Slika 36: Prikaz robota s kapo 25
Slika 37: Prikaz robota brez kape 25
Slika 38: Prikaz robota po odboju trampolina 25
Slika 39: Profesor testira aplikacijo 26
Slika 40: Dijak David testira aplikacijo 27
Slika 41: Dijak Kai testira aplikacijo 27
Slika 42: Razvijalec Tai testira aplikacijo 27
Slika 43: Dijak Tjaš testira aplikacijo 27
Slika 44: Dijaki Aljaž, Jakob in Uroš testirajo aplikacijo 27
Slika 45: Dijak Peter testira aplikacijo 27
Slika 46: Zaslonski posnetek kode – Zagon programa in zbiranje ukazov 39
Slika 47: Zaslonski posnetek kode - Izvajanje ukazov z uporabo funkcije RunCommands 40
Slika 48: Zaslonski posnetek kode - Resetiranje prizorišča (1. del) 41
Slika 49: Zaslonski posnetek kode - Resetiranje prizorišča (2. del) 41

Slika 50: Zaslonski posnetek kode - Zaključek stopnje in izračun rezultatov 42
Slika 51: Zaslonski posnetek kode - Aktivacija zaključka stopnje ob trku z zlatnikom 42
Slika 52: Zaslonski posnetek kode - Prehod v naslednjo stopnjo 43
Slika 53: Zaslonski posnetek komponente – Prikaz skripte RobotScript v Unityju 44
Slika 54: Zaslonski posnetek kode – Premikanje robota po določenem številu polj 45
Slika 55: Zaslonski posnetek kode – Zaznava vstopa robota v novo polje 45
Slika 56: Zaslonski posnetek kode – Preverjanje gibanja robota 45
Slika 57: Zaslonski posnetek kode – Funkcija za obračanje robota 46
Slika 58: Zaslonski posnetek kode – Določanje rotacije robota med obračanjem 46
Slika 59: Zaslonski posnetek kode – Funkcija skoka robota z nastavitvijo višine in hitrosti .. 47
Slika 60: Zaslonski posnetek kode – Dvigovanje robota z nastavitvijo hitrosti leta 47
Slika 61: Zaslonski posnetek kode – Preverjanje padanja robota 47
Slika 62: Zaslonski posnetek kode – Preverjanje prisotnosti leteče kape 48
Slika 63: Zaslonski posnetek kode – Postavitev trampolina 48

KAZALO GRAFOV

Graf 1: Samoocena predznanja računalništva in/ali programiranja med udeleženci 28
Graf 2: Izkušnje anketirancev s programiranjem 28
Graf 3: Razporeditev anketirancev po letniku srednje šole 28
Graf 4: Prva izkušnja udeležencev z virtualno resničnostjo 29
Graf 5: Uporabniška orientacija v VR okolju – gibanje in krmilniki 29
Graf 6: Fizično počutje med uporabo VR 29
Graf 7: Razumevanje posameznih konceptov programiranja 29
Graf 8: Uporabnost aplikacije kot učnega pripomočka za začetnike 30
Graf 9: Ocena težavnosti stopenj v igri 30
Graf 10: Prikaz ocen zanimivosti uporabe aplikacije 30
Graf 11: Kaj ti je bilo najbolj všeč pri aplikaciji? 31
Graf 12: Mnenje o primernosti aplikacije za osnovnošolce (7.–9. razred) 32
Graf 13: Priporočilo za uporabo aplikacije pri pouku v 1. letniku srednje šole 32
Graf 14: Bi priporočil/-a vrstnikom? 32
Graf 15: Kako bi na splošno ocenil/-a aplikacijo »Code in VR« kot celoto? 32
Graf 16: Bi aplikacijo »Code in VR« priporočil/-a kot učni pripomoček za uporabo v šolah? 33

POVZETEK

Tradicionalne metode učenja programiranja so lahko zahtevne, še posebej za začetnike, ki se prvič srečujejo s koncepti, kot so zanke, pogojni stavki in zaporedje ukazov. Zato smo razvili VR aplikacijo, ki uporabnikom omogoča intuitivno učenje algoritmičnega razmišljanja s pomočjo blokovnega programiranja. Namesto osredotočanja na sintakso se uporabniki učijo skozi vizualno sestavljanje ukazov, kar jim omogoča boljše razumevanje temeljnih principov programiranja. Igralec vodi svoj lik skozi različne izzive, kjer mora s sestavljanjem ukazov, kot so »Premakni se naprej«, »Obrni se« in »Ponavljaj«, premagovati ovire in doseči cilj – zlatnik. Postopno uvajanje kompleksnejših nalog uporabniku omogoča razvoj logičnega razmišljanja in razumevanje algoritmov, ne da bi ga pri tem ovirale sintaktične napake, ki so pogosto izziv pri učenju tradicionalnega kodiranja. V igri so različne vrste stopenj, vključno s časovno omejenimi izzivi, naključno generiranimi postavitvami in kreativnimi stopnjami, kjer lahko uporabnik sam oblikuje poligon in preizkusi svoje rešitve. Virtualna resničnost dodatno poglobi učno izkušnjo, saj omogoča prostorsko vizualizacijo algoritmov in neposredno interakcijo s programskimi bloki. S tem pristopom želimo preveriti, ali lahko blokovno programiranje v VR okolju izboljša razumevanje algoritmičnega razmišljanja in poveča motivacijo za učenje programiranja. Cilj je, da aplikacija postane učinkovit učni pripomoček, ki na interaktiven način spodbuja razvoj logičnega razmišljanja in algoritmičnih konceptov ter s pomočjo virtualne resničnosti ustvarja bolj privlačno in zabavno učno izkušnjo za začetnike.

Ključne besede: aplikacija, algoritem, blokovno programiranje, virtualna resničnost.

SUMMARY

Traditional methods of learning programming can be challenging, especially for beginners encountering concepts such as loops, conditional statements, and command sequences for the first time. To address this, we have developed a VR application that enables users to intuitively learn algorithmic thinking through block-based programming. Instead of focusing on syntax, users learn by visually assembling commands, helping them better understand fundamental programming principles. The player guides their character through various challenges, overcoming obstacles and reaching the goal - a gold coin - by assembling commands like "Move forward," "Turn," and "Repeat." The gradual introduction of increasingly complex tasks helps users develop logical thinking and understand algorithms without being hindered by syntax errors, which are often a challenge in traditional coding education. The game includes different types of levels, such as time-limited challenges, randomly generated layouts, and creative levels where users can design their own playground and test their solutions. Virtual reality enhances the learning experience by providing spatial visualization of algorithms and direct interaction with programming blocks. With this approach, we aim to determine whether block-based programming in a VR environment can improve the understanding of algorithmic thinking and increase motivation for learning programming. The goal is for the application to become an effective educational tool that interactively fosters logical reasoning and algorithmic concepts while making learning more engaging and enjoyable for beginners through the use of virtual reality.

Keywords: application, algorithm, block-based programming, virtual reality.

1 UVOD

Tehnologija računalništva in stvari, povezane z računalništvom, se zelo hitro razvijajo in širijo. Brez računalnikov in programov ter procesov, ki se izvajajo na računalnikih, skoraj nič več ne deluje, zato je storitev programiranja izjemno pomembna. Kot dijaki programa tehnik računalništva se zavedamo izzivov, ki jih prinašata programiranje in logično razmišljanje. Zato smo s pomočjo mentorja razvili aplikacijo (igro), ki začetnikom in tistim, ki še niso seznanjeni s programiranjem, omogoča učenje osnov algoritmičnega razmišljanja skozi interaktivno igranje.

Na področju računalništva je v zadnjih letih velik poudarek tudi na tako imenovani navidezni oz. virtualni resničnosti. Ker nam šola omogoča dostop do orodij in naprav za delo z virtualno resničnostjo (za to namenjena očala in krmilniki), smo se odločili, da bomo našo igro uporabnikom podali v virtualni resničnosti, ki še dodatno poudarja interaktivnost in občutek prostora.

Naša igra temelji na večstopenjskem sistemu, kjer igralec upravlja lik robota in z uporabo ukazov premaguje ovire. Namesto klasičnega pisanja kode uporabnik sestavlja programske bloke, ki predstavljajo posamezne ukaze in funkcije. Na ta način se uporabnik postopno uči logičnega razmišljanja in konceptov programiranja, ne da bi se takoj soočil s kompleksnostjo sintakse.

Med razvijanjem aplikacije smo ugotovili, da obstajajo podobni primeri aplikacij, ki so nam še dodatno pomagali pri razvijanju idej. To so zlasti Scratch (vizualni programski jezik, zasnovan za začetnike), Blocky games (vizualni programski jezik, osnova podobna naši aplikaciji, vendar zgolj v spletni različici, 2D ter vizualno neprivlačna) in Programmer VR (podobna aplikacija, ki v VR prikaže programiranje). Pri prvi omenjeni aplikaciji uporabnik podobno kot pri nas s pomočjo blokov sestavlja algoritem, vendar ustvarja igro. Pri drugi aplikaciji gre za to, da uporabnik na različne načine rešuje problem (npr. labirint), vendar je na voljo zgolj v spletni različici, 2D okolju ter je vizualno neprivlačna. Tretja aplikacija je bolj naši rešitvi, a se razlikuje v več stvareh: koncept, vizualnost, težavnost, raznolikost, vodič, gemifikacija. V igro smo vključili 30 različnih stopenj, med katerimi so tudi stopnje z gemifikacijo, sestavljanjem svojega polja igranja ter naključno ustvarjeno mapo. Gemifikacijo spodbujamo s časovnimi omejitvami, kjer je uporabnik pod pritiskom, da ne stori napake in s hitrim razmišljanjem odkrije najoptimalnejšo rešitev za problem, ki mu je zastavljen.

Celotna izkušnja je zelo koristna, saj smo se natančno poglobili v razvoj od ideje do izdelka, in to na področju, ki ga brez pobude profesorja ne bi preveč raziskovali. Aplikacijo še vedno razvijamo, saj skozi razvoj dobivamo nove ideje, odpravljamo napake, je pa že dovolj razvita za uporabo in testiranje z naključnimi prostovoljci. Tako dobimo povratno informacijo o izdelku in ugotovimo, kaj koristnega aplikacija ponuja in kaj je treba še dodelati. Na podlagi tega smo tudi izvedli to raziskovalno nalogo.

2 OPIS PROBLEMA

Ko smo pričeli z razvijanjem aplikacije oz. igre, smo se soočili zlasti z izzivom, kako osnovne koncepte programiranja in logičnega razmišljanja približati popolnim začetnikom. Kot dijaki programa tehnik računalništva smo s temi koncepti že seznanjeni, zato je bil naš cilj razviti rešitev, ki bi začetnikom omogočila praktičen in intuitiven uvod v algoritmično razmišljanje skozi VR izkušnjo.

Začeli smo se pogovarjati s profesorjem in prišli do ugotovitev, na podlagi katerih smo začeli zastavljati naslednja raziskovalna vprašanja:

- Kako učinkovito lahko virtualna resničnost simulira programiranje in pripomore k razvoju algoritmičnega razmišljanja?
- Ali lahko uporaba VR aplikacije poveča motivacijo za učenje programiranja v primerjavi s klasičnimi učnimi metodami?
- Kako igrifikacija vpliva na učni proces pri začetnikih in ali lahko igra služi tudi naprednejšim uporabnikom?
- Ali lahko virtualna resničnost v celoti nadomesti klasično programiranje ali pa deluje le kot dopolnilo pri učenju osnov?
- Na kakšen način bi lahko aplikacijo uporabili kot del učnega procesa pri programiranju in kakšne prednosti bi to prineslo?
- Kako bodo uporabniki dojemali težavnost VR aplikacije in ali je primerna za različne stopnje znanja?
- Ali igrifikacija v VR pomaga pri dolgoročnem pomnjenju programskih konceptov?
- Katere omejitve VR aplikacije bi bilo treba premagati, da bi bila enakovredna klasičnemu programiranju?

Glede na zastavljena vprašanja smo postavili naslednje hipoteze:

H1: Virtualna resničnost lahko učinkovito simulira proces programiranja in pomaga razvijati algoritmično razmišljanje.

H2: Uporaba VR aplikacije kot učnega pripomočka povečuje motivacijo za učenje programiranja in spodbuja razumevanje osnov algoritmičnega razmišljanja.

H3: Možno je združiti igrifikacijo in izobraževanje na način, ki olajša učenje začetnikom ter omogoča postopno povečevanje težavnosti tudi za naprednejše uporabnike.

H4: Virtualna resničnost lahko v celoti nadomesti klasično programiranje in služi kot učinkovit uvod v razumevanje programskih konceptov.

H5: Aplikacijo bi bilo možno uporabiti kot dopolnilo pri pouku programiranja, saj omogoča praktično učenje v varnem okolju brez strahu pred napakami.

2.1 Metodologija

Najprej smo pripravili načrt razvoja aplikacije, da smo na koncu vsake stopnje razvoja imeli jasen pregled nad napredkom in smerjo, po kateri smo se ravnali. Ta način načrtovanja nam je omogočil boljšo organizacijo dela in hkrati dvigal motivacijo, saj smo ob vsakem doseženem mejniku videli, kako aplikacija postaja vedno bolj dovršena.

Pri zasnovi koncepta smo uporabili metodo možganske nevihte in pristop, ki smo ga po svoje poimenovali »Vsak prispeva svojo idejo«. Tako smo se osredotočili na funkcionalnost in izgled aplikacije, izmenjevali ideje ter skupaj sprejemali ključne odločitve o tem, katere funkcionalnosti so bistvene in katere bi lahko dodali v prihodnjih različicah. Nekaj predlogov za izboljšave so nam podali tudi prostovoljci – dijaki šole, ki so sodelovali pri testiranju aplikacije v razvoju.

Skozi celoten razvoj smo prakticirali metodo preizkusa. Po vsaki novi verziji je bilo treba aplikacijo natančno testirati – ne le preverjati, ali deluje, kot smo pričakovali, ampak tudi kritično preizkušati njene meje in iskati morebitne hrošče. Pri tem nam je bil v veliko pomoč profesor, ki je testiral delovanje aplikacije in nas usmerjal na pravo pot.

Za boljšo organizacijo ekipnega dela in nadzor nad različicami aplikacije smo vse posodobitve skrbno shranjevali v skupne mape. Tako smo zagotovili boljši pregled nad spremembami in olajšali ekipno sodelovanje. Uporabljali smo tudi programsko rešitev Trello za beleženje napredka programske kode, izdelave modelov ter evidentiranje idej in programskih hroščev, kar nam je omogočilo lažji nadzor nad celotnim razvojem.

Oprli smo se tudi na metodo anketiranja in izpraševanja. Iskali smo prostovoljce, saj smo potrebovali povratne informacije o uporabniški izkušnji. Anketo so izpolnili dijaki različnih letnikov in z različnim predznanjem, kar nam je omogočilo širok spekter mnenj in idej za izboljšave. Aplikacijo smo premierno predstavili na informativnih dnevih, kjer so jo preizkusili osnovnošolci in nam podali koristne povratne informacije za prihodnje izboljšave.

3 TEORETIČNI DEL

3.1 Virtualna resničnost

Virtualna resničnost (VR) je tehnologija, ki uporabnikom omogoča, da se potopijo v računalniško ustvarjene, interaktivne tridimenzionalne svetove. Z uporabo posebnih VR očal (kot so Meta Quest, HTC Vive, PlayStation VR itd.) in dodatkov, kot so ročni krmilniki ali haptične rokavice, lahko uporabniki doživijo občutek prisotnosti v navideznem okolju. S pomočjo VR očal in namenskih VR krmilnikov lahko uporabniku pričaramo, da se navidezni svetovi oz. simulacije zdijo skoraj popolnoma resnični (Slika 1) (Vir [1][4]).



Slika 1: VR očala in krmilniki
(VIR: The Leaked Quest 3 Headset Video Teases Meta's VR Ambitions [7])

3.1.1 Razvoj VR tehnologije skozi zgodovino

VR ima kratko, vendar zanimivo zgodovino, ki sega v drugo polovico 20. stoletja. Koncept virtualne resničnosti se je sprva pojavil kot ideja znanstvene fantastike, vendar se je kasneje razvil v tehnologijo, ki se danes uporablja v številnih panogah (Vir [2][3][4]).

Leta 1956 je Ivan Sutherland, ameriški računalniški znanstvenik, razvil prvi sistem VR, ki ga je imenoval "Sword of Damocles" (Slika 2). Ta sistem je vključeval veliko napravo, ki je bila nameščena na stropu in je uporabniku omogočala, da vidi 3D slike v realnem času (Vir [2][3][4]).

V 60. in 70. letih 19. stoletja so znanstveniki začeli razvijati simulacijske sisteme, ki so uporabnikom omogočali, da se počutijo, kot da so v drugem okolju. Prvi tak velik izum v razvoju VR je bil izum Head-Mounted Display (HMD). Ta tehnologija je prvotno bila razvita za vojaške namene, kjer je bila najpogosteje uporabljena za simulacijo letenja (Vir [2][3][4]).

Leta 1968 je skupina znanstvenikov iz Massachusetts Institute of Technology (MIT) razvila sistem, ki je uporabljal kamere in senzorje, da je uporabnikom omogočal, da se premikajo po virtualnem okolju. Ta sistem je bil znan kot "The Ultimate Display" in je bil namenjen raziskavam na področju VR (Vir [2][3][4]).

V 80. letih so se začeli pojavljati prvi komercialni sistemi VR. Podjetje VPL Research je leta 1985 izdalo sistem DataGlove, ki je uporabnikom omogočal, da nadzirajo virtualne objekte z uporabo rok. Leta 1989 je bila ustanovljena tudi prva VR tematska parkovna atrakcija – Virtuality (Vir [2][3][4]).

V 90. letih se je razvoj VR nadaljeval, saj so se pojavile nove tehnologije, kot so pospeševalniki gibanja, 3D grafika in bolj napredne HMD naprave. VR se je začel uporabljati tudi v medicini, kjer so tehnologijo uporabljali za simulacije kirurških posegov in za lajšanje bolečin pri bolnikih (Vir [2][3][4]).

V začetku 21. stoletja se je razvoj VR znova pospešil, saj so se pojavile nove platforme, kot sta Oculus Rift in HTC Vive, ki so uporabnikom omogočale boljše in bolj realistične VR izkušnje. Z razvojem pametnih telefonov so se pojavile tudi VR mobilne aplikacije. Vendar kljub napredku VR tehnologija še vedno predstavlja nekaj izzivov. Eden izmed glavnih izzivov je visoka cena in zahtevnost uporabe za običajne uporabnike, saj zahteva zmogljive računalnike in drage naprave. Poleg tega se pojavljajo tudi pomisleki glede potencialnih negativnih učinkov na zdravje uporabnikov, kot so slabost, omotica in glavoboli (Vir [2][3][4]).

Prihodnost virtualne resničnosti se vse bolj nagiba k multisenzornim izkušnjam. Ne gre le za to, kaj lahko uporabniki vidijo. Ampak gre za to, česa se lahko dotaknejo, povohajo in celo okusijo. Bolj ko je virtualni svet realističen, bolj poglobljena in privlačna je izkušnja za uporabnika. Seveda pa si prihodnosti ne moremo predstavljati, saj se VR tehnologija razvija iz dneva v dan in je vedno bolj napredna (Vir [2][3][4]).



Slika 2: Prvi VR sistem "Swords of Democles"
(VIR: Ivan Sutherland's first VR Head Mounted Display, The Sword of Damocles [8])

3.2 Uporabe VR tehnologij

VR tehnologija je zdaj že zelo razširjena, zaradi svoje prilagodljivosti pa ima velik potencial na številnih področjih. Uporablja se v zdravstvu, vojski, zabavi in tudi izobraževanju (Vir [4]).

V vojski se VR uporablja (Slika 3) za izboljšanje učinkovitosti usposabljanja, hkrati pa zmanjšuje tveganja in stroške, povezane s tradicionalnimi metodami. S pomočjo prilagojenih sistemov za usposabljanje VR se lahko servisno osebje vključi v široko paleto scenarijev, od arktične vojne do pristankov letal, in to v nadzorovanih in varnih okoljih (Vir [4]).

V medicini se VR uporablja (Slika 4) kot orodje, s katerim si kirurgi lažje predstavljajo in vizualizirajo del telesa, ki ga morajo operirati. Z pomočjo virtualne resničnosti pa se usposabljajo tudi pripravniki v vseh vejah zdravstva. Moderna tehnologija jim omogoča boljšo vizualizacijo in s tem si pripravniki lažje zapomnijo pomembne stvari (Vir [4]).

Eno od najpomembnejših področij uporabe VR tehnologije je izobraževanje. Kot se morajo vojaki in zdravniki izobraževati v svoji stroki, lahko virtualna resničnost omogoča tudi izobraževanje ljudi na različnih področjih in to želimo doseči mi z našim izdelkom. S pomočjo VR se lahko učenci, dijaki, študentje podajo v zgodovinske svetove, lahko potujejo po vesolju, upravljajo z raznimi komponentami pri sestavljanju računalnikov in mnogo več. Z uporabo VR tehnologije pri uporabnikih raste zanimanje za določene stroke in posledično to tehnologijo uporablja vse večji spekter ljudi v različnih poklicih (Vir [4]).

VR tehnologija se uporablja tudi v zabavni industriji, predvsem pri igranju računalniških iger. Naša igra ni samo zabavna, je tudi izobraževalna, se pa pojavljajo tudi igre za čisto zabavo. Na tem področju se VR zelo razvija in je iz leta v leto bolj podoben resničnosti (Vir [4]).



Slika 3: Uporaba VR v vojski
(VIR: Ortmore agency, Use of VR in military [5])



Slika 4: Uporaba VR v medicini
(VIR: SmartTek, Virtual reality training for healthcare [6])

3.3 Zahteve za uspešno VR aplikacijo

Čeprav zahteva visoke finančne vložke, je VR tehnologija v izobraževanju v velikem vzponu. Šole in univerze po vsem svetu jo že uporabljajo za poučevanje tako naravoslovnih kot družboslovnih predmetov. Če se bo razvoj tehnologije nadaljeval, lahko pričakujemo, da bo VR postala še bolj dostopna in priljubljena metoda poučevanja.

Vendar uvedba VR v izobraževalne programe ni preprost proces, saj zahteva izpolnitev določenih pogojev. Prvi in najpomembnejši je primerna oprema. Za uporabo VR aplikacij potrebujemo VR očala, krmilnike in zmogljive računalnike, ki morajo biti redno vzdrževani in posodobljeni. Določeni VR kompleti, kot je Meta Quest, sicer omogočajo delovanje brez zunanjega računalnika, vendar so zaradi tega omejeni s strojno močjo in življenjsko dobo baterije.

Poleg tega je potreben ustrezen prostor, saj VR aplikacije omogočajo veliko več interakcije kot klasični računalniški programi. Uporabniki se morajo prosto premikati, ne da bi pri tem ogrožali sebe ali druge. To lahko predstavlja organizacijski in finančni izziv za izobraževalne ustanove, saj je treba zagotoviti varno okolje za uporabo tehnologije.

Eden ključnih dejavnikov pri uvajanju VR v izobraževanje so tudi finančna sredstva. VR tehnologija je še vedno relativno draga, zato si številne šole ne morejo privoščiti celotne opreme za vse učence. Nekatere ustanove zato uvajajo postopne rešitve, kjer kupijo le nekaj VR sistemov za testiranje in prilagajanje učnega procesa.

Nenazadnje je strokovna usposobljenost osebja bistvenega pomena za učinkovito uporabo VR pri poučevanju. Profesorji in učitelji morajo biti usposobljeni za pravilno uporabo in vzdrževanje tehnologije, saj gre za drugačen in še relativno nov način poučevanja. Le tako lahko izkoristimo vse prednosti, ki jih VR prinaša v izobraževalni proces.

4 PRIPRAVE NA RAZVOJ

Ko smo se odločili za izdelavo te igre oz. aplikacije, je bilo vse, kar smo imeli, le ideja. Sprva nismo bili prepričani, ali bomo lahko tako zahteven projekt izpeljali brez večjih zapletov, vendar nas je profesor ves čas spodbujal, zato smo z veliko volje začeli z delom. Imeli smo osnovno znanje iz izdelovanja 3D modelov in tudi nekaj osnovnega znanja o izdelovanju iger. Skozi razvijanje aplikacije pa smo zelo dobro usvojili tudi veščini sodelovanja in dela v timu. Med razvijanjem smo podpirali drug drugega in tako prišli do končnega izdelka.

4.1 Spoznavanje orodij za razvijanje aplikacije

Pred začetkom razvoja smo morali poiskati orodja, s pomočjo katerih bomo razvijali aplikacijo. Za okolje, v katerem smo kreirali in oblikovali 3D modele, smo izbrali Blender, saj smo s tem že imeli izkušnje.

Programsko okolje pa smo izbrali po naslednjih zahtevah:

- omogoča delo z že ustvarjenimi 3D modeli,
- možnost izdelave aplikacij za VR okolje,
- omogoča testiranje na računalniku ter VR napravi in
- obsežna uporabniška skupnost.

Tako smo se odločili za okolje Unity, s katerim smo se vsi strinjali, saj smo ga vsi že uporabljali. Tudi sicer ima to okolje veliko uporabnikov, kar posledično pomeni, da je na internetu veliko virov o uporabi, nasvetov in rešitev, če bi morda naleteli na težavo, ki je ne bi znali rešiti. Ker je bil projekt ekipno delo, smo za posredovanje datotek in olajšanje ekipnega dela uporabili orodje Trello.

4.2 Razvojno okolje Unity

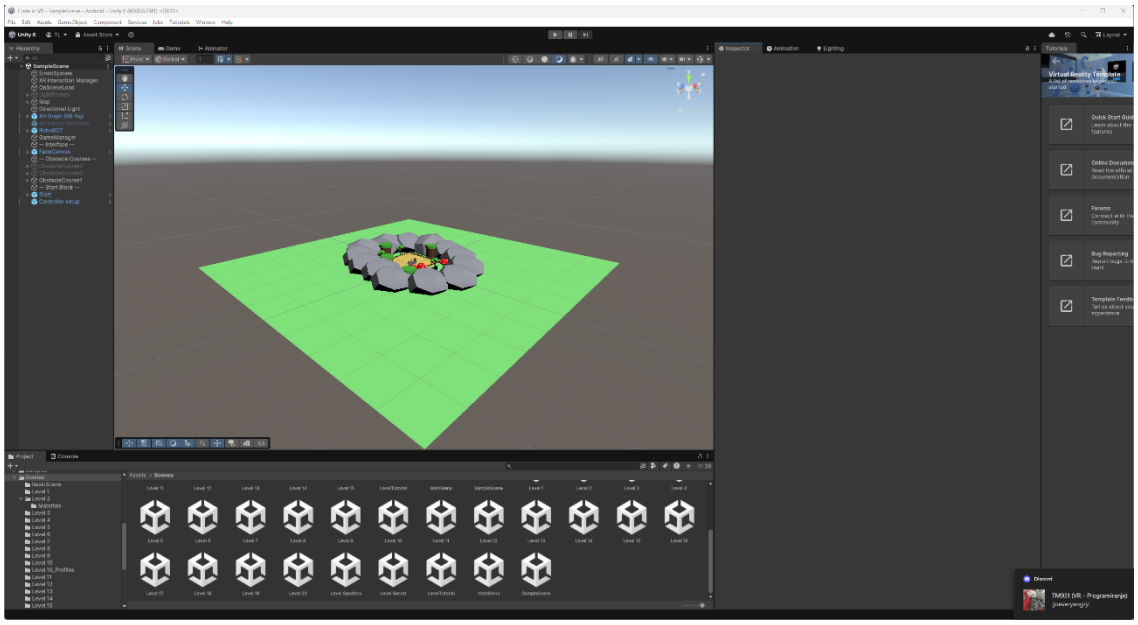
Unity je eno izmed najbolj priljubljenih okolij za razvoj iger in interaktivnih aplikacij. Omogoča ustvarjanje 2D in 3D iger, virtualne resničnosti (VR), obogatene resničnosti (AR) in drugih izkušenj. Razvijalci ga uporabljajo za izdelavo iger, simulacij, animacij in arhitekturnih vizualizacij.

4.2.1 Glavne značilnosti razvojnega okolja Unity

- Večplatformska podpora: Unity omogoča razvoj za PC, konzole (PlayStation, Xbox, Nintendo Switch), mobilne naprave (Android, iOS), splet in VR/AR naprave.
- C# kot osrednji programski jezik: Scripting v Unityju temelji na jeziku C#, kar omogoča enostavno učenje in močno podporo skupnosti.
- Močna skupnost in obsežna dokumentacija: Ker je Unity zelo razširjen, je na forumih veliko informacij, dokumentacije za pomoč pri učenju in video gradiv.

Poleg navedenih stvari Unity nudi še tako imenovan Asset store, ki vsebuje modele, ki jih lahko uporabljamo v svojih projektih. Vendar mi tega nismo rabili, saj smo svoje modele izdelali sami. (Vir [4]).

Vsebuje tudi veliko lastnih sistemov in tehnologij, ki se razvijajo že vrsto let, vsebuje pa tudi dostop do zunanjih aplikacij, kar omogoča tesno povezavo med temi aplikacijami in Unity okoljem. S tem je Unity (Slika 5) zelo prijazen novim uporabnikom, ki jim primanjkujejo izkušnje, saj za veliko manjših stvari poskrbi kar okolje samo in nam s tem omogoča, da se popolnoma osredotočimo na razvoj aplikacije.

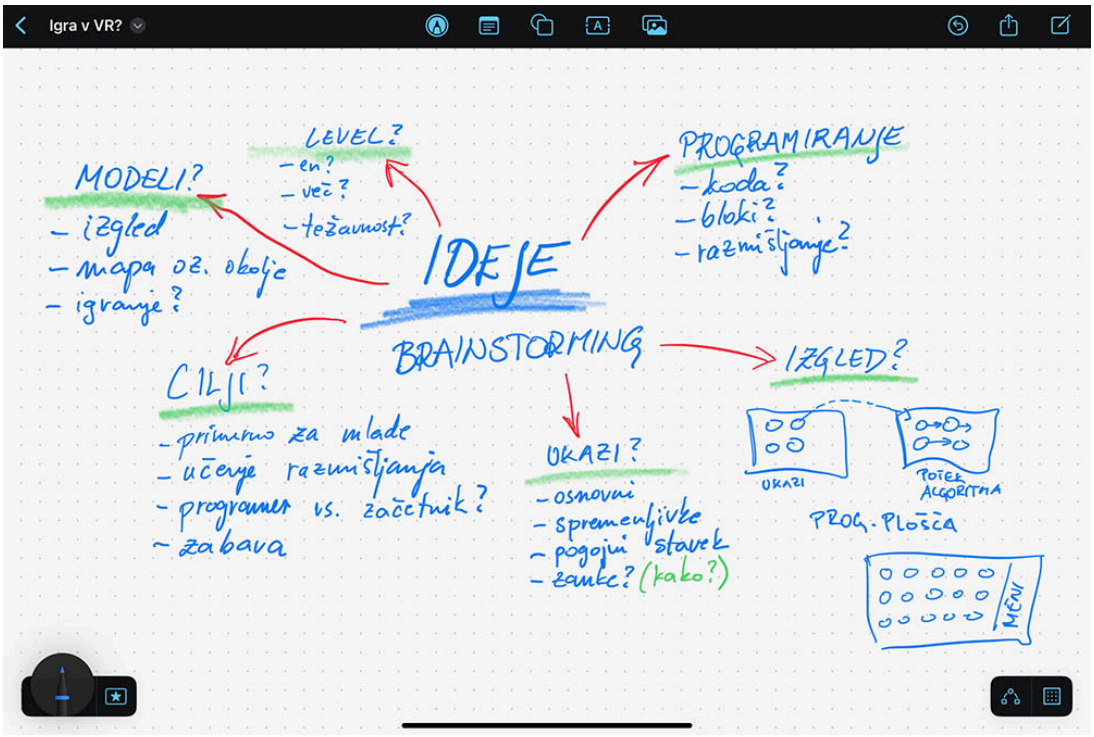


Slika 5: Zaslonski posnetek, razvojno okolje Unity

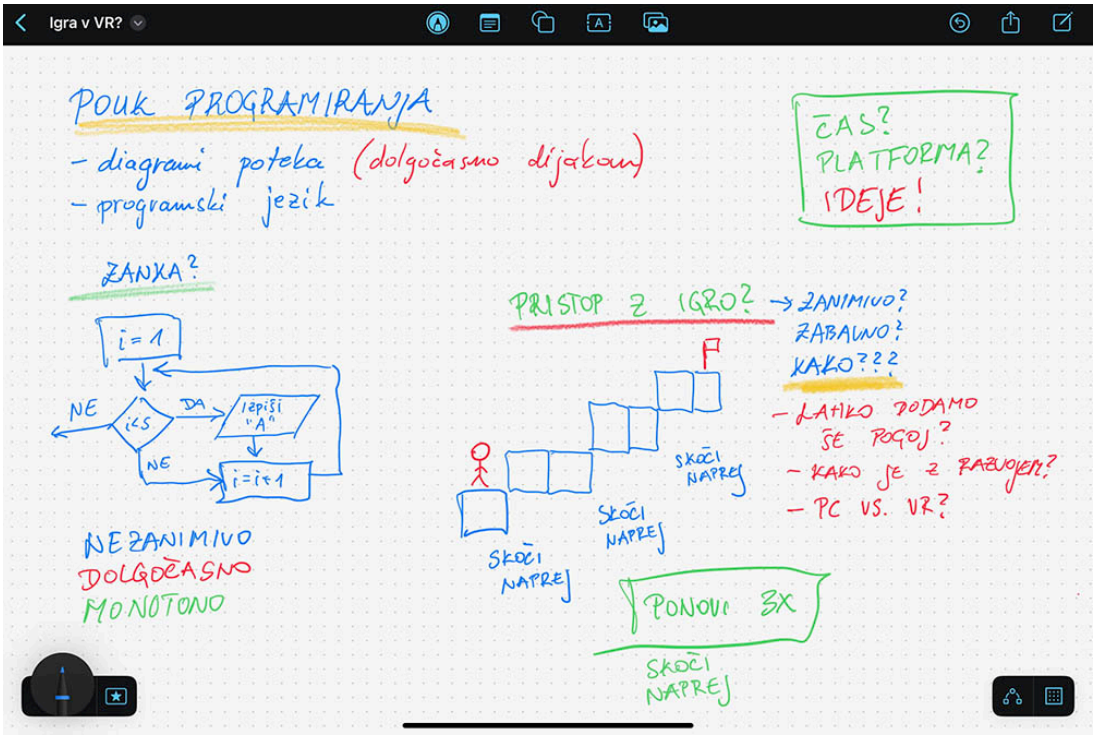
4.3 Ideja in koncept

Vsak dober projekt najprej potrebuje dobro idejo in koncept. Tudi mi smo začeli na ta način, in sicer smo s pomočjo metode možganska nevihta oz. »brainstorming« izdelali miselni vzorec okvirnih stvari, ki jih želimo implementirati v našo aplikacijo, in tudi hiter zapis zadanih ciljev. Miselni vzorec je zajemal okvirne stvari, ki smo jih želeli imeti v aplikaciji (Slika 6).

Zanimalo nas je, kako približno bo delovalo premikanje robotka, kaj vse želimo z grafičnim vmesnikom prikazati uporabnikom, kakšni naj bodo 3D modeli, primerjali smo našo idejo učenja programiranja z drugimi (diagrami poteka, klasično programiranje). Na hitro smo tudi skicirali okviren in grob prikaz aktivnega nivoja in kako bo moral uporabnik razmišljati, če bo želel optimalno rešiti stopnjo (Slika 7).



Slika 6: Prva konceptualna skica (ideja, »brainstorming«)



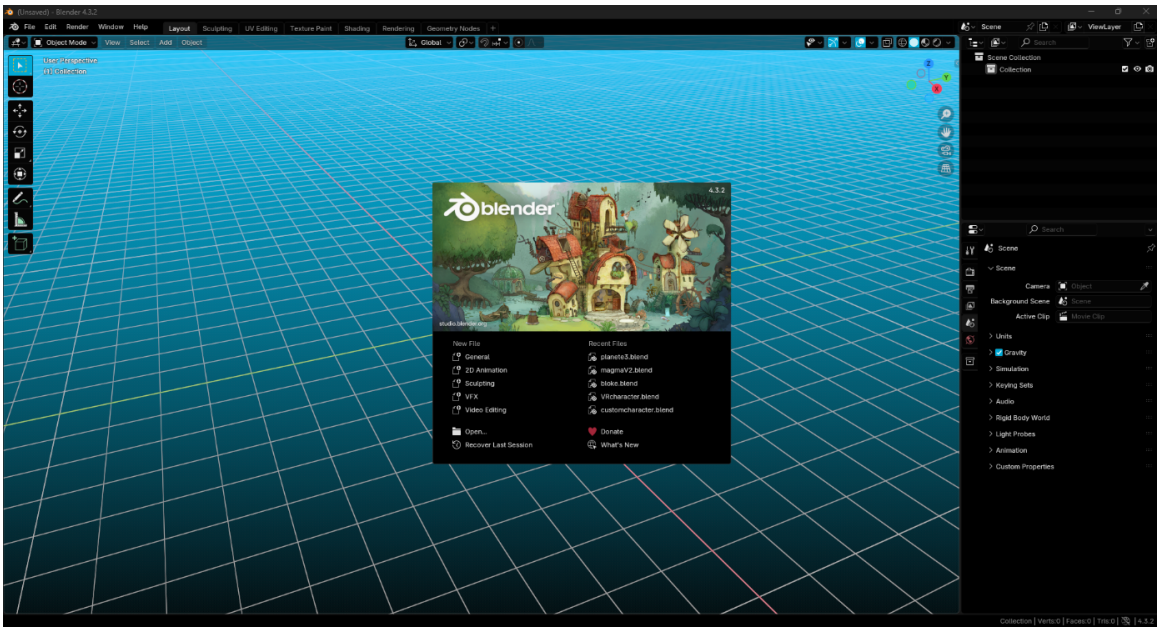
Slika 7: Druga konceptualna skica (pouk programiranja, dodana vrednost)

V osnovni ideji smo vedeli, da želimo lik robota premikati po igralnem polju, kjer bi moral preskakovati, se obračati v desno ali levo ter pobrati zlatnik. Debatirali smo tudi o tem, kako pametno je vključiti naprednejše ukaze. Ker naj bi naša aplikacija delovala

v smislu igre, smo prav tako ugotovili, da moramo vanjo dodati nekaj stvari, ki uporabniško izkušnjo naredijo bolj pestro in privlačnejšo. Tako smo tudi dobili idejo za bloke s posebnimi močmi, ki jih bo lik lahko pobral in jih uporabil. Ugotavljali smo tudi, kako bi v aplikacijo vključili uporabo logičnega razmišljanja, da bi bila uporaba naše aplikacije koristna pri uvodu v programiranje, saj bi od uporabnika zahtevala podobno razmišljanje, kot ga potrebujemo pri diagramih poteka.

4.4 Priprava pred izdelavo projekta

Preden smo začeli z delom, je bilo potrebno pripraviti nekaj stvari. Ena izmed njih je bila namestitev programskega okolja Unity na naše naprave. Nato sta nastopila težja dela, saj smo se morali odločiti, ali bomo ustvarili svoje 3D modele ali uporabili že ponujene modele iz Unityjeve knjižnice Asset store, prav tako smo si morali organizirati posredovanje posodobitev in nadgradenj na aplikaciji. Odločili smo se, da bomo za posredovanje posodobljenih datotek uporabljali oblačne storitve in si tako delili mape, do katerih smo imeli vsi dostop in smo jih lahko urejali ter v njih shranjevali nove datoteke. Ker je teh datotek bilo veliko, smo si za še boljšo preglednost ustvarili posebne sekcije deljenih map, in sicer za zaslonske posnetke igre, nalaganje oz. posredovanje 3D modelov, rezultate oz. analizo ankete in podobne stvari. Za 3D modele pa smo se odločili, da bomo uporabili svoje, lastno izdelane modele, saj smo imeli nekaj izkušenj z izdelavo le-teh. S tem smo tudi dosegli, da je aplikacija oz. igra popolnoma naš projekt, kar je bila še dodatna motivacija, saj smo aplikacijo sestavljali in razvijali iz praktično ničesar do produkta, ki je nastal na koncu. 3D modele smo ustvarili z orodjem Blender. Prej omenjeno orodje Trello pa smo uporabili za boljšo organizacijo ekipnega dela, saj nam orodje omogoča, da na enostaven način spremljamo napredek aplikacije, sporočamo o obstoječih napakah oz. hroščih v igri in konec koncev tudi beležimo nove ideje in predloge, ki smo si jih izmislili sami, nam jih je predlagal profesor ali pa mogoče kakšen od prostovoljcev, ki so testirali našo aplikacijo.



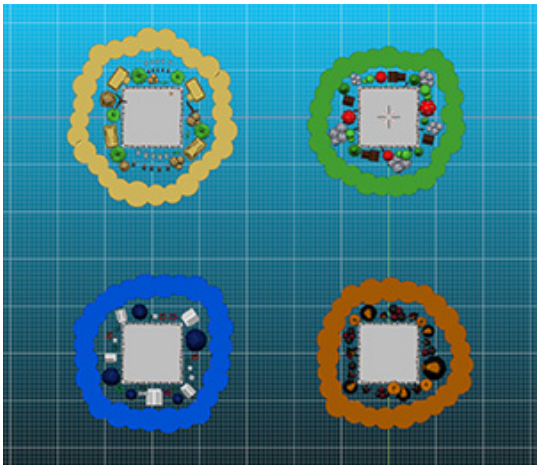
Slika 8: Zaslonski posnetek, okolje Blender

5 TEHNIČNI DEL

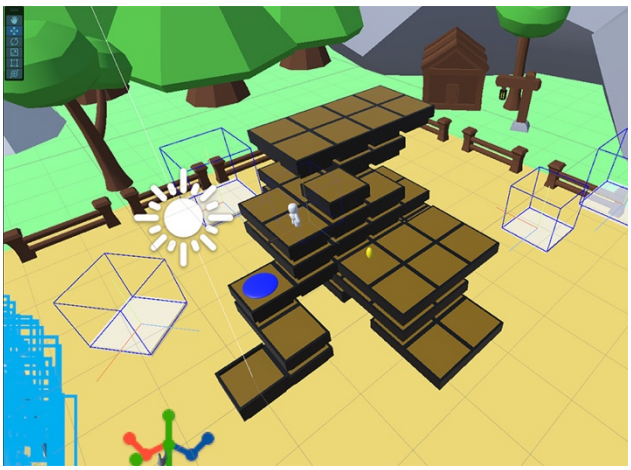
5.1 OKOLJE, MODELI IN FUNKCIONALNOSTI

Pri vsakem projektu je najtežje začeti, zato smo na začetku poskrbeli, da bo delo za postavitev okolja mape bolj zabavno. Na voljo smo imeli različne teme: gozd, puščavo, morje in vulkan (Slika 9).

Pričel se je razvoj in uresničitev načrtovanih idej o tem, kako bi vse skupaj izgledalo in kako bi lahko aplikacijo usmerili v smer, da bi bila zabavna, hkrati še poučna in odličen pripomoček pri izobraževanju začetnikov programiranja oz. algoritmičnega razmišljanja. Za začetek smo pripravili idejno zasnovo za eno stopnjo, da smo lahko testirali različne možnosti, postavitev elementov, premikanje, zagon ukazov, pobiranje zlatnika, trampolina, postavitev svetlobe ipd. (Slika 10).

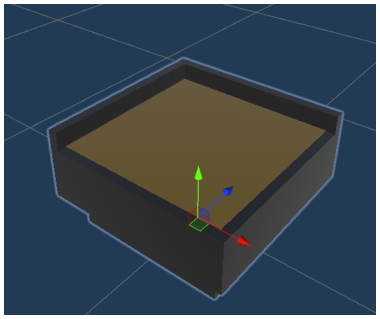


Slika 9: Primeri modelov map

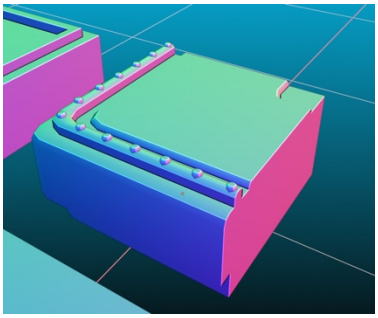


Slika 10: Začetna verzija aplikacije

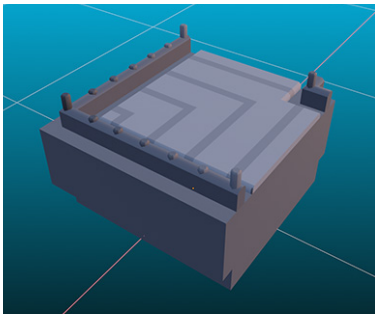
Medtem ko je potekal razvoj aplikacije, dodajanje novih stopenj, razvijanje funkcionalnosti, je hkrati potekalo tudi modeliranje vseh elementov, ki so potrebni za aplikacijo. V začetku smo imeli precej osnovni model (Slika 11), nato pa smo s pomočjo orodja Blender modelirali svoje modele. Primer spodaj prikazuje polje, ki je kotno (Slika 12), z minimalistično ter elegantno izdelano teksturo (Slika 13).



Slika 11: Osnovni model polja iz začetne verzije aplikacije

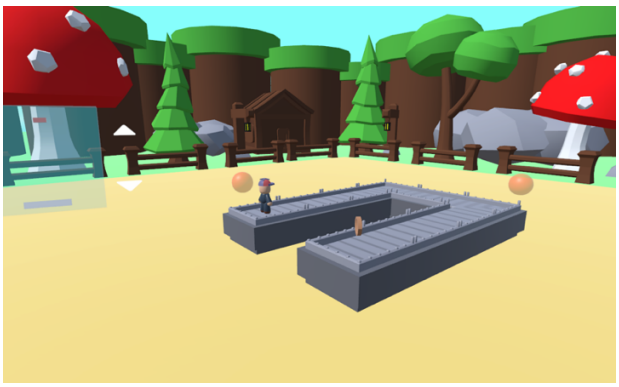


Slika 12: Modeliran nov model za polje brez teksture v orodju Blender

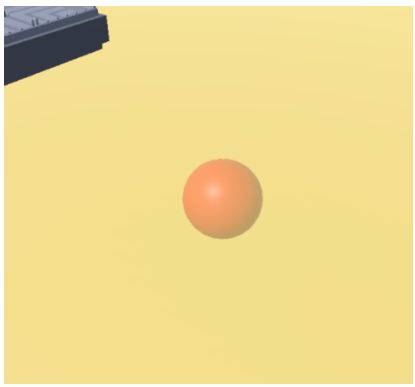


Slika 13: Modeliran nov model za polje s teksturo v orodju Blender

Ustvarili smo namišljeno mapo in prvo stopnjo igre (Slika 14), cilj katere je, da se igralec oz. uporabnik nauči uporabljati igro. Kot je lahko razvidno že iz prej omenjene prve stopnje (Slika 14), lahko opazimo rdeče krogle. Razmišljali smo o tem, kako nadgraditi osnovni pogled igre, da se lahko uporabnik pomika po prostoru ter opazuje stopnjo iz čim več možnih zornih kotov. Pogled na izziv iz drugega zornega kota mu lahko pomaga do lažjega razmišljanja ali načrtovanja, kako se bo lotil reševanja v nadaljevanju. V ta namen smo ustvarili t. i. portale (»teleportna mesta«) (Slika 15), kamor se igralec preprosto premika s pomočjo VR krmilnika. Ker je uporaba le-tega implementirana v aplikacijo, smo pri prvi stopnji priložili navodila, kako uporabljamo premikanje po prostoru in tudi druga navodila za igranje igre.



Slika 14: Prva stopnja igre



Slika 15: Portal za premikanje

Po vzpostavitvi osnove in dodeljeni osnovi delovanja igre smo se odločili, da je to prava smer razvoja aplikacije. Nato smo začeli dodajati druge nujne stvari, ki smo si jih zamislili. Tako smo stvarili lik robota (Slika 16), ki ga upravljamo z našimi vnesenimi bloki z ukazi. Odločili smo se da bo cilj igre, da tega robotka spravimo preko vseh ovir do zlatnika. Ko robot pobere zlatnik, se igra konča. Poskrbeli smo tudi za statistiko za vsako posamezno stopnjo, ki se nam izpiše po končani stopnji.



Slika 16: Model robotka

Za prej omenjene bloke z ukazi smo se odločili pod nekaterimi pogoji. Strinjali smo se, da mora biti programiranje robotka preprostejše kot navadno programiranje in bolj razumljivo splošnim uporabnikom, saj je cilj naše aplikacije predvsem, da uporabnik razvije logično razmišljanje, ki je prisotno pri programiranju. Vsak blok smo razvrstili v

ustrezno področje (akcije, spremenljivke, funkcije in posebne moči). Vsak blok je v programu svoj objekt, ki ima unikatno kodo, kar pomeni, da ima vsak blok tudi svojo vlogo, ki jo opravi, ko zaženemo sestavljen algoritem v igri. Bloki delujejo po principu »run-command«, ki je skripta na objektu in to skripto poišče »game manager« (t. i. skrbnik v posamezni stopnji, ki skrbi za delovanje stopnje) in zažene pravi ukaz. V sklop »akcije« (Slika 17) smo tako uvrstili preproste ukaze za »premik X polj«, kjer uporabnik določi število X in koliko polj se bo robot premaknil naprej. Ta polja določi uporabnik po lastni presoji. Pri tem mu pomagajo prav za to ustvarjeni 3D modeli platform, iz katerih lahko razberemo, kakšna je dolžina, ki jo mora prehoditi naš robotek. Naslednji preprosti ukaz je »skoči«. Ko se ta ukaz zažene, robotek skoči v višini enega bloka navzgor in se pomakne naprej, sledi tudi ukaz »obrni se [levo/desno]«, kjer igralec izbere, ali se bo robot obrnil v levo ali desno ipd. Pri tem ukazu pridejo v poštev pogledi iz več zornih kotov, saj tako lažje razberemo, v katero smer se mora robotek obrniti. V sklop »funkcije« (Slika 18) smo dodali blok za »če«, ki predstavlja pogojni oz. odločitveni stavek, dodali smo tudi blok za »ponavljaj«, ki predstavlja t. i. zanko, torej rutino, ki se ponovi tolikokrat, kot je določeno število X, ki ga tudi določi igralec sam.

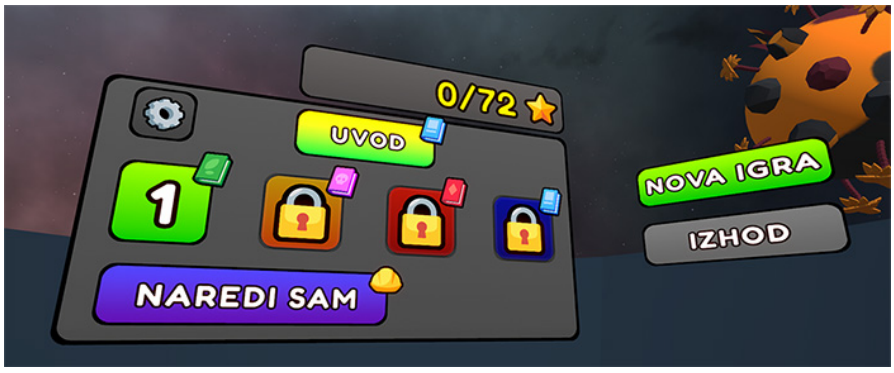


Slika 17: Prikaz ukazov iz razdelka akcije



Slika 18: Prikaz ukazov iz razdelka funkcije

Po dokončanih osnovnih funkcionalnostih igre smo se odločili dodati več različnih stopenj. Dobili smo različne ideje, kako z raznolikimi stopnjami izboljšamo izkušnjo uporabnika. Stopnje so razvrščene po planetih od 1 do 4 (Slika 19).



Slika 19: Meni igre

Prvih 14 stopenj je klasičnih, v posamezni stopnji pa se nahaja več podstopenj, kar pomeni, da če se odločimo za reševanje 5. stopnje, imamo v njej 2 različna izziva,

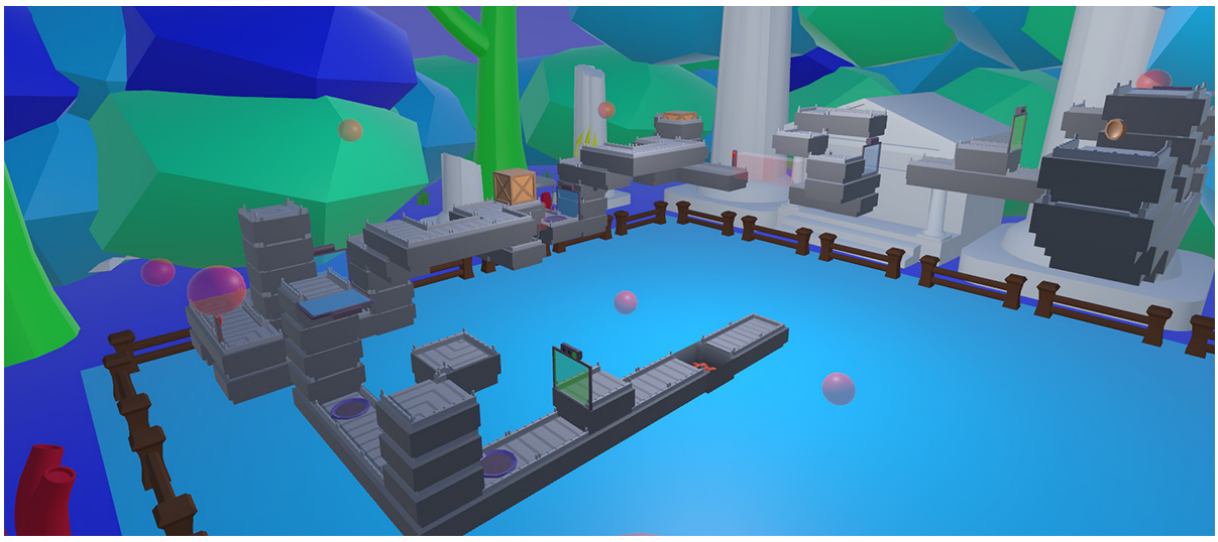
vsakega rešujemo posebej - ko zaključimo prvega, nadaljujemo z drugim. Število izzivov v posameznih stopnjah se prav tako razlikuje, saj se težavnost stopenj stopnjuje glede na to, kako daleč smo se že prebili skozi izzive, ki so nam zastavljeni.

Stopnja 15 je že specifična, saj nas napada tudi šef (t. i. »boss level«), ki nas, poleg tega, da je stopnja že tako težja od prejšnjih, še omejuje in pritiska na uporabnika. Šef nas, če predolgo stojimo na mestu, namreč ustreli s svojo puščico in moramo stopnjo začeti ponovno. K tem stopnjam smo dodali še nekaj drugih specifičnih stopenj. Prve med posebnimi stopnjami so stopnje z omejenim časom (Slika 20). Ob začetku stopnje imamo nekaj časa, da si ogledamo, kako izgleda stopnja in s kakšnim izzivom se bomo morali soočiti. Nato se po nekaj sekundah odklene vmesnik za sestavljanje programa za našega robotka. V tem času je potrebno celotni algoritem sestaviti v enem delu. Če ne uspemo in nam poteče čas, dodeljen za stopnjo, smo igro izgubili. Kot zanimivost je pomembno poudariti, da če ponastavimo stopnjo, ni nujno, da bomo znova reševali enak izziv, saj smo v ozadju pripravili več različnih izzivov za posamezno stopnjo, da še bolj popestrimo igranje. S tem od uporabnika zahtevamo, da mora poleg dobrega algoritmičnega razmišljana tudi hitro sprejeti in sestaviti najboljšo rešitev za stopnjo.



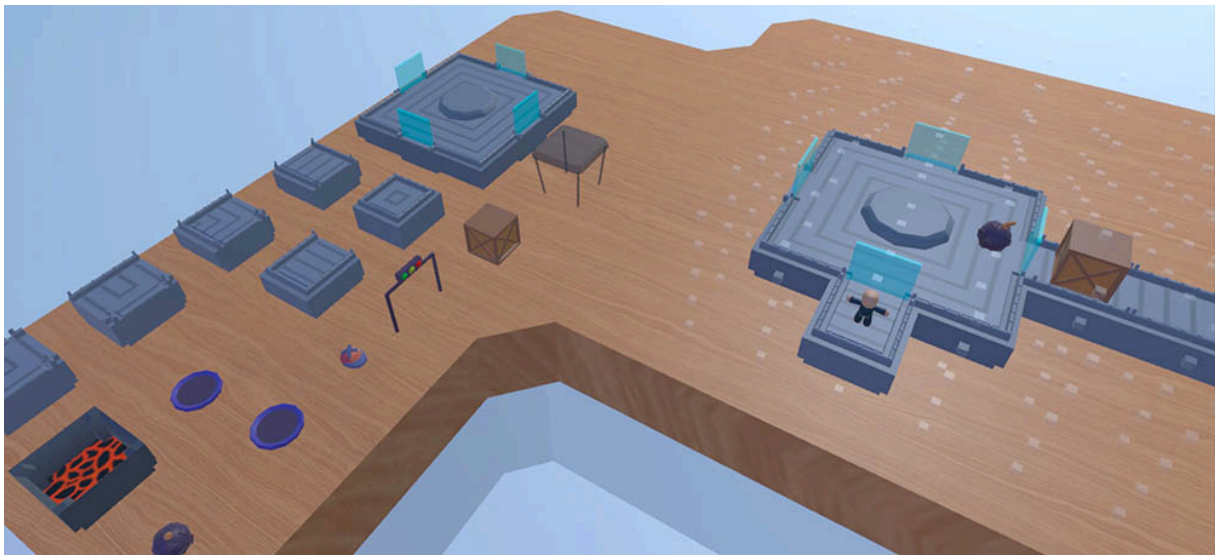
Slika 20: Prikaz časovnih stopenj v meniju

Naslednja posebna stopnja je stopnja z naključno generiranimi ovirami (Slika 21). Pri tem nivoju aplikacija, ko zaženemo izbrano stopnjo, sama generira ovire, ki jih mora robotek premagati s pomočjo ukazov, ki mu jih podamo. Ta nivo doda igri vedno nove možnosti in nove izkušnje, saj je doživetje vedno drugačno. Pomemben del pa je tudi ta, da se uporabnik ne more na pamet naučiti nivoja, saj je pristop reševanja vedno drugačen.



Slika 21: Naključno ustvarjena stopnja

Za vse naprednejše uporabnike smo poskrbeli tako, da smo ustvarili še dodatno vrsto stopnje, kjer uporabnik sam uporabi svojo kreativnost, si sestavi nivo in postavi izzive samemu sebi. Nivo deluje tako, da se pojavimo pred t. i. »delovno mizno« (Slika 22), na kateri so predstavljeni vsi gradniki, ki so lahko uporabljeni v nivoju, ki ga sestavljamo, tako začetek kot konec. Po ustvarjeni lastni stopnji se lahko preizkusimo, kako dober izziv smo si zastavili in ali ga lahko rešimo. Cilj tega nivoja ni samo poučevanje o uporabi logike za programiranje, temveč je poudarek tudi na kreativnosti in unikatnem razmišljanju vsakega posameznika.



Slika 22: Delovna miza v stopnji "Naredi sam"

Ker je bila aplikacija oz. igra razvita v sklopu pouka in za namen izobraževanja, smo želeli uporabnikom, ki se morda prvič srečujejo z blokovnim programiranjem, priskočiti na pomoč. V ta namen smo omogočili, da si lahko uporabnik vključi ali izključi opise in razlage posameznih blokov (Slika 23 in Slika 24), da je bolj razumljivo, kaj je naloga oz. kakšno nalogo opravlja posamezni blok v sestavljenem programu.



Slika 23: Opis programskega bloka "Obrni se"

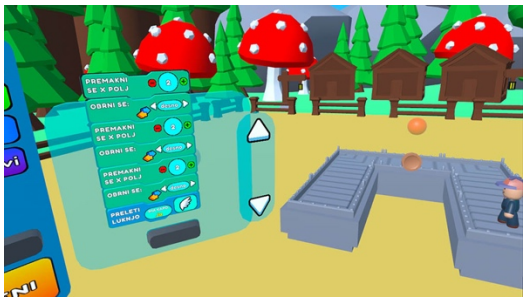


Slika 24: Opis programskega bloka za stavek "ČE"

Poskrbeli pa smo tudi, da samo reševanje izzivov na koncu le ni tako preprosto. Skozi testiranje smo dobili idejo, da lahko nagradimo le najboljše rešitve. V ta namen smo vsako stopnjo skrbno preverili in privzeto nastavili idealno število blokov, potrebnih za rešitev, in dodali še čas. Do optimalnih rešitev za stopnje smo prišli z lastnim testiranjem igre in preizkušanjem več različnih rešitev naloge. Tako igralec lahko stopnjo reši in napreduje po stopnjah, vendar prejme za rešitev sobe le »2 zvezdici« (Slika 25). Takšna klasična rešitev ni nujno najbolj optimalna (Slika 26).



Slika 25: Uspešno končana stopnja "2 zvezdici"



Slika 26: Prikaz klasične rešitve stopnje

V določenih stopnjah pa lahko pripravimo še boljšo rešitev od zgolj klasične, torej naše lastne rešitve te stopnje, in sicer z uporabo zank, npr. bloka »ponavljaj« (Slika 27), kar je boljša rešitev in bo uporabnik za tako rešitev sobe nagrajen s »3 zvezdicami« (Slika 28). S tem tudi spodbudimo uporabnika, da ko bo rešil izziv in bo nagrajen z zgolj »2 zvezdicama«, da bo najverjetneje želel izziv reševati še enkrat in iskal boljše rešitve in možnosti. Z implementacijo zvezdic, ki so dodeljene glede na uspešnost rešene stopnje, želimo poudariti tudi, da je to uporabno v nadaljnjem življenju, saj je v svetu programiranja najboljša rešitev vedno tista, ki je najbolje optimizirana in se izvede v čim krajšem času. Za vse tiste uporabnike, ki so vse stopnje rešili z oceno »3 zvezdice«, pa se odklene zadnja, posebna stopnja. Z odklepanjem te stopnje damo uporabnikom vedeti, da so igro z odliko zaključili in so pripravljeni na svet programiranja.



Slika 27: Prikaz napredne rešitve stopnje



Slika 28: Uspešno končana stopnja "3 zvezdice"

Aplikacija je namenjena širši skupini uporabnikov, zato smo vključili večjezičnost. V ta namen smo v aplikacijo dodali nastavitve, ki omogočajo spremembo jezika, spreminjamo pa lahko tudi zvočne efekte ter opise programskih blokov, ki smo jih že prikazali. Naša aplikacija je tako v slovenskem (Slika 29) in angleškem (Slika 30) jeziku.



Slika 29: Prikaz nastavitev v slovenskem jeziku



Slika 30: Prikaz nastavitev v angleškem jeziku

5.2 PROGRAMSKA KODA

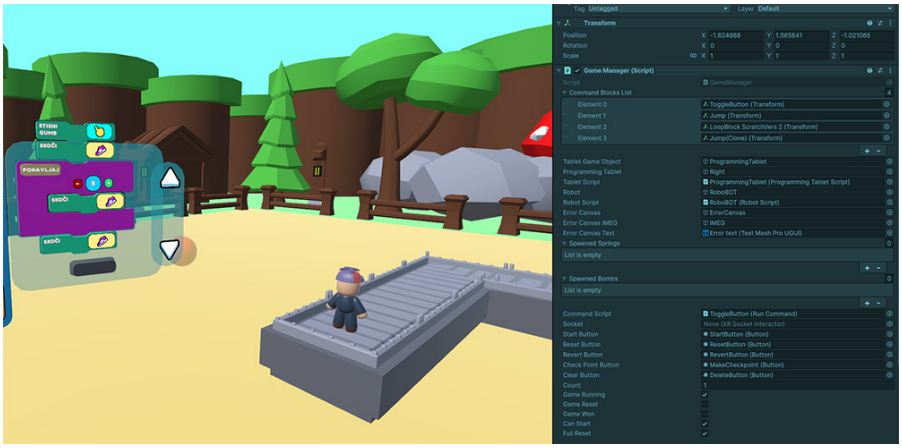
V nadaljevanju bomo predstavili ključne funkcionalnosti, ki omogočajo osnovno delovanje igre. Zaradi kompleksnosti projekta in obsežnega števila skript smo se osredotočili na pregled nadzornega sistema *GameManager* ter skripte *RobotScript*, ki upravlja z gibi in animacijami igralnega lika.

5.2.1 GameManager

5.2.1.1 Uvod v delovanje skripte GameManager

Vloga skripte *GameManager* je osrednjega pomena za pravilno izvajanje programskih ukazov, ki jih uporabnik sestavi s pomočjo blokov v igri. Nadzira celoten potek igre – od začetka izvajanja algoritma, prek upravljanja z napakami in ponastavitvami, pa vse do zaključka posamezne stopnje in prehoda naprej.

S pomočjo orodja Unity smo omogočili dobro vizualizirano programsko kodo (Slika 31). Program se začne z uporabo funkcije *StartProgram()*, ki prebere vse ukaze, ki jih je uporabnik sestavil z blokovnim programiranjem, in jih shrani v seznam (list), ki se po takšnem vrstnem redu tudi izvede, kar pomeni, da naš robotek izvede ukaze, ki so dodeljeni vsakemu od teh blokov. Ko so vsi ukazi zbrani, funkcija *RunCommands()* poskrbi, da se izvedejo po točno določenem vrstnem redu.



Slika 31: Vizualni prikaz shranjevanja programskih blokov

Če med izvajanjem pride do napake (npr. robot pade izven igralnega polja ali naleti na neizvedljivo oviro), se sproži funkcija *ResetScene()*, ki ponastavi prizorišče – bodisi na zadnjo shranjeno točko bodisi na začetek stopnje, če shranjena točka ne obstaja. Ko robot uspešno zaključi izziv in pobere kovanec, se aktivira funkcija *LevelComplete()*, ki oceni rezultat in pripravi prehod. Nazadnje funkcija *Next()* poskrbi za prehod v naslednjo stopnjo ali vrnitev v glavni meni, če stopenj ni več na voljo.

Celoten potek tako ponazarja osnovno strukturo programskega izvajanja: zbiranje ukazov, izvajanje, obravnava napak, preverjanje uspeha, nadaljevanje.

5.2.1.2 Funkcija StartProgram() – zbiranje in priprava ukazov

Funkcija *StartProgram()* poskrbi za zbiranje ukazov, ki jih je uporabnik sestavil iz programskih blokov. Ti ukazi se shranijo v seznam in pripravijo za izvajanje. Sistem

nato robota postavi na začetno pozicijo in pripravi prizorišče za začetek izvajanja algoritma.

Programska koda, prikazana na sliki (Priloga 10.1, Slika 49), omogoča, da se z uporabo funkcije *StartProgram()* zažene proces zbiranja programskih blokov z igralne površine. S pomočjo metode *StartGameCoroutine()* sistem poišče vse vstavljene ukaze in jih po vrstnem redu doda v seznam za izvajanje. Ko so ukazi zbrani, se robot poravna na trenutno polje glede na to, ali gre za ponovni zagon (*reset*) ali prvi začetek stopnje. Vključeni so tudi časovni zamiki, ki zagotovijo tekoče izvajanje animacij, nato pa se izvedejo ukazi s funkcijo *RunCommands()*.

5.2.1.3 Funkcija RunCommands() – izvajanje ukazov po vrsti

Funkcija *RunCommands()* skrbi za postopno izvajanje ukazov, ki jih je uporabnik predhodno sestavil z bloki. Ko robot zaključi en ukaz, se ta funkcija pokliče znova in začne izvajati naslednjega. Postopek se ponavlja, dokler niso izpolnjeni vsi ukazi ali dokler ne pride do prekinitve (npr. napake v igri ali konca algoritma).

Programska koda funkcije *RunCommands(),* prikazana na sliki (Priloga 10.1, Slika 50), preverja, ali obstajajo še neizvedeni ukazi v seznamu in ali igra še poteka. Če pogoji ustrezajo, se iz trenutnega bloka pridobi ustrezen skript z ukazom, ki ga je treba izvesti. Če je vse pripravljeno, se zabeleži, da je trenutno ukaz v izvajanju. V primeru, da ukaza ni, se izvajanje ustavi. Če je ukaz veljaven, se ta izvede z *RunBlockCommand()*.

5.2.1.4 Funkcija ResetScene() – ponastavitev poligona

Funkcija *ResetScene()* poskrbi za ponastavitev poligona, kadar pride do napake – na primer če robot pade s ploščadi ali stori napako v algoritmu. Najprej se ustavijo vsi procesi, animacije in izvajanje ukazov. Nato se preveri, ali obstaja shranjena kontrolna točka (*checkpoint*). Če obstaja, se igra vrne nanjo, sicer pa se ponastavi na začetno stanje. Ta postopek omogoča hitro vračanje na mesto napake brez potrebe po ponovnem začetku.

Programska koda na sliki (Priloga 10.1, Slika 51) prikazuje funkcijo *ResetCoRutine()*, ki najprej prikaže sporočilo o napaki prek *errorCanvas*, ki opozori uporabnika, zakaj je prišlo do prekinitve. Nato ustavi izvajanje igre, pobriše seznam ukazov in ponastavi stanje robota (npr. animacije, gibanje, moč, položaj, gravitacijo, uporabljene blokade in podobno). Vsi aktivni bloki se vrnejo v privzeto stanje – tudi vizualno, če so bili pred tem posebej označeni. Poleg tega se ponastavijo tudi bombe, hitrost gibanja in drugi začasni učinki. Koda v tej fazi ne glede na stanje vedno poskrbi, da se vse animacije ustavijo in se igralna logika pripravi na nadaljnje delovanje.

Programska koda na sliki (Priloga 10.1, Slika 52) je nadaljevanje zapisa kode za funkcijo *ResetCoRutine()* in v tem delu preverja, ali obstaja shranjena kontrolna točka (*lastCheckPoint*). Če shranjena kontrolna točka obstaja, se robot vrne na tisto točko skupaj z vsemi svojimi lastnostmi, ki so bile tam aktivne (npr. zbrane bombe, hitrost, dodatki, uporaba magneta ...). Vse spremembe, ki so se zgodile po tej točki, se izbrišejo (z uporabo metode *ClearObjectsSinceCP()*), s čimer se vzpostavi točno takšno stanje, kot je bilo ob shranitvi. Če kontrolna točka ne obstaja, se robot vrne na začetni položaj, ponastavi se usmerjenost in vsi učinki (npr. trampolin, magneti ...) se

nastavijo na začetne vrednosti. V tem primeru se izvede popolna ponastavitev (*fullReset* = *true*) in pripravi se novo začetno stanje za igro.

5.2.1.5 Zaključek stopnje – ocena uspešnosti in prehod v naslednjo stopnjo

Programska koda funkcije *LevelCompleted()*, ki je prikazana na sliki (Priloga 10.1, Slika 53), se aktivira, ko igralec uspešno dokonča stopnjo. V tem trenutku se izračuna skupni rezultat, ki temelji na dveh dejavnikih – času in številu uporabljenih blokov. Če igralec dokonča izziv znotraj predvidenega časa, prejme maksimalno časovno oceno, v nasprotnem primeru se točkovanje zmanjša. Poleg tega se upošteva še učinkovitost rešitve – manj kot je bilo uporabljenih blokov glede na optimalno število, več točk igralec prejme. Funkcija nato ustavi interakcijo z gumbi in izpiše poročilo o uspešnosti. Prikazano besedilo je prilagojeno jeziku, ki ga ima uporabnik nastavljenega (slovenščina ali angleščina).

Programska koda (Priloga 10.1, Slika 54) prikazuje preverjanje, če robot doseže cilj in se dotakne zlatnika. V tem primeru se preveri, ali gre za konec stopnje (s pomočjo funkcije *CompareTag("End")*). Če da, se predvaja zvok uspeha, prikaže se zaključek izziva in zlatnik se odstrani iz prizorišča. Hkrati se sproži funkcija *LevelCompleted()*, ki zaključi igro in omogoči prehod naprej.

5.2.1.6 Funkcija Next() – prehod na naslednjo stopnjo ali vrnitev v meni

Programska koda funkcije *Next()*, prikazana na sliki (Priloga 10.1, Slika 55), poskrbi za prehod med stopnjami ali vrnitev v meni. Ko uporabnik zaključi izziv in izbere možnost za nadaljevanje, funkcija preveri, ali obstaja naslednji izziv v tej stopnji. Če obstaja, jo naloži; če ne, igralca preusmeri v glavni meni. Obenem shrani doseženi rezultat in poskrbi za prehod robota ter vmesnika v novo stanje.

Na začetku se z ukazom *RobotToStart()* pripravi igralno okolje za novo stopnjo. Nato se na podlagi jezika izpiše sporočilo o naloženem nadaljevanju. Funkcija deaktivira uporabniški vmesnik za potrjevanje (*continueCanvas*), onemogoči trenutno mapo in poveča števec trenutne stopnje. Če je uporabnik dosegel zadnjo stopnjo (*currentLevel* > *obstacleCourses.Length - 1*), se preveri, ali je bil rezultat boljši od prejšnjega, in se shrani. Nato se uporabnik vrne v glavni meni. V nasprotnem primeru pa se aktivira naslednja stopnja in omogoči nadaljnje igranje z ustrezno nastavitvijo tipk za interakcijo (*SetButtonsInterractable()*).

5.2.2 RobotScript – upravljanje robotskega vedenja v igri

RobotScript je osrednja skripta, ki določa vedenje in delovanje robota znotraj igre. Skrbi za gibanje, interakcijo z ovirami in predmeti, uporabo moči (power-upov) in odzivanje na različne situacije, kot so padci, skoki, zaznave trkov in animacije.

Skripta omogoča nadzor nad ključnimi parametri robota, kot so:

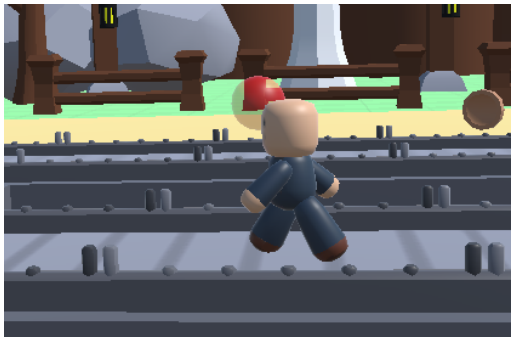
- premikanje (npr. število korakov, hitrost, smer),
- sposobnosti (npr. skok, trampolin, hoja po gredi z magnetom),
- stanja (npr. letenje, rotacija, stik z oviro),
- avdio-vizualni elementi (npr. zvoki in animacije),
- interakcija z elementi (npr. bomba, kapa, zlatnik).

Slika (Priloga 10.1, Slika 56) prikazuje nastavitve skripte *RobotScript* v Unityju. Na levi strani se nahajajo različna logična stanja (*boolean states*), ki odražajo trenutno situacijo robota (ali je na pritiskalni plošči, ali se lahko premika naprej ipd.), na desni pa so povezani objekti in viri (*GameObjecti*, zvoki, animatorji), ki jih skripta uporablja. *RobotScript* je središče celotne logike za robota in ključno prispeva k temu, da robot pravilno izvaja ukaze, se pravilno odziva na okolje ter nudi uporabniku tekočo, odzivno in poučno izkušnjo.

5.2.2.1 Premikanje robota za izbrano število polj

Programska koda funkcije *MoveXTiles()*, prikazana na sliki (Priloga 10.1, Slika 57), omogoča, da se robot premakne za točno določeno število polj naprej. Ko je aktivirana, sproži gibanje robota in nadzoruje izvajanje animacije premikanja. Gibanje se zaključi, ko robot prehodi vse določene korake, nato se izvede naslednji ukaz iz algoritma.

Ko se funkcija *MoveXTiles(int x)*, prav tako prikazana na sliki (Priloga 10.1, Slika 57), kliče, prejme parameter x, ki določa število polj, ki jih mora robot prehoditi. Funkcija aktivira gibanje (*movingOnGround* = *true*) in nastavi zvočni posnetek hoje. Nato pokliče funkcijo *MoveTiles()*, ki preverja, ali je število korakov še večje od 0 – v tem primeru robot začne hoditi in animacija se aktivira. Primer prikaza hoje v aplikaciji prikazuje spodnja slika (Slika 32).



Slika 32: Zaslonski posnetek igre – vizualni prikaz hoje robota

Ko število korakov doseže 0, se animacija hoje ustavi, premikanje se onemogoči in robot se natančno postavi v središče ciljne ploščice (*tileCenter*). Nato funkcija preveri, ali mora sistem nadaljevati z izvajanjem naslednjega ukaza – bodisi z *RunCommands()* iz *GameManagerja* ali znotraj povezanega ukaznega priklopa (*commandSocketScript*), če obstaja.

Funkcija *OnTriggerEnter()*, ki je prikazana na sliki (Priloga 10.1, Slika 58), se sproži ob trku z določenim objektom. V tem primeru zazna, ali je robot dosegel objekt z oznako CenterPoint. Če se zazna trk, se zmanjša vrednost tiles, kar pomeni, da je robot prešel eno polje. Nato se posodobi pozicija centra (tileCenter), hitrost hoje se povrne na osnovno vrednost in ponovno se pokliče MoveTiles().

Funkcija *MoveCheck()*, prikazana na sliki (Priloga 10.1, Slika 59), nadzira gibanje robota glede na stanje premikanja. Če je premik aktiven, se nastavi hitrost naprej. Če ni, se hitrost premikanja ustavi. Funkcija preverja stanje spremenljivke *move*. Če je vrednost *true*, robotu določimo premik v smeri naprej z nastavljeno hitrostjo, v

nasprotnem primeru se hitrost gibanja ustavi tako, da ostane samo vertikalna komponenta hitrosti (npr. pri padanju), horizontalno pa robot miruje. Ta pristop omogoča natančno in tekoče gibanje robota v igri.

5.2.2.2 Obračanje robota v igri

Ko želi robot spremeniti smer, se sproži funkcija za obračanje. Ob ukazu se shrani nova usmerjenost, robot pa začne z animacijo obračanja (Slika 33). Po zaključku rotacije nadaljuje z izvajanjem naslednjih ukazov v algoritmu.



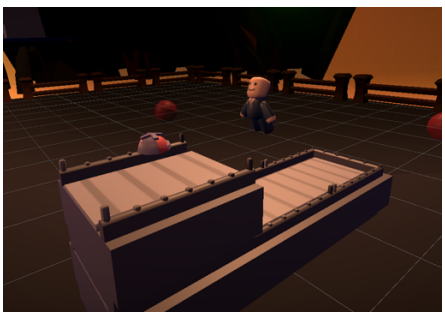
Slika 33: Zaslonski posnetek igre - vizualni prikaz obračanja robota

Obračanje robota poteka v dveh korakih. Programska koda na sliki (Priloga 10.1, Slika 60) prikazuje funkcijo *Turn(),* ki prejme kot parameter smer gibanja (*levo ali desno*) in na podlagi tega nastavi vrednosti za rotacijo (*rotateDir*) ter izvede animacijo. Nato se zažene funkcija *FinishRotating()*, ki skrbi za časovno izvedbo obračanja. Po 1 sekundi se robot poravna v novo smer (s pomočjo *Quaternion.Euler*) in nastavi zastavico *rotate* na *false*, kar pomeni, da je obračanje zaključeno.

Funkcija *RotateCheck(,)* prikazana na sliki (Priloga 10.1, Slika 61), medtem preverja, ali je rotacija v teku. Če je, rotira robota postopoma glede na hitrost in smer obračanja.

5.2.2.3 Skok robota v igri

Funkcija *Jump()* omogoča, da robot skoči navpično navzgor na določeno višino. Pri tem upošteva več parametrov, kot so višina skoka, hitrost, aktivacija gibanja na tleh ter posebne pogoje (kot je skok s trampolina). Uporablja se predvsem za prečkanje višinskih ovir (Slika 34) ali ob uporabi trampolina (Slika 35).



Slika 34: Zaslonski posnetek – prikaz skoka robota v igri



Slika 35: Zaslonski posnetek – primer uporabe trampolina za skok

Programska koda, prikazana na sliki (Priloga 10.1, Slika 62), vključuje funkcijo *Jump()* in funkcijo, posebno korutino, ki omogoča časovno upravljanje skokov, animacij in drugih dejanj v igri. Funkcija *Jump()* sprejme več parametrov, s katerimi določa, kako se bo robot dvignil v zrak. Če je robot trenutno prilepljen na kovinsko gred z magnetom, se višina obrne (negativna smer), saj v tem primeru deluje drugače. Funkcija nato zažene korutino *JumpAnimator()*, ki dejansko upravlja s trajanjem skoka in omogoča animacijo ter izračun hitrosti.

V funkciji *JumpAnimator()*:

- se aktivirata letenje (*fly* = *true*) in animacija skoka;
- glede na parameter *useSpeed* se določi hitrost skoka;
- če je število prehojenih polj manjše ali enako 0, se avtomatsko nastavi na 1 – to je ključno za pravilno delovanje trampolina.

Po čakanju, ki ustreza višini in hitrosti skoka, se robot ustavi in ponovno aktivira gibanje na tleh (velja le ob predhodni uporabi trampolina).

Funkcija *FlyCheck()*, ki je prikazana na sliki (Priloga 10.1, Slika 63), preveri, ali je robot v stanju letenja. Če je, mu dodeli navpično hitrost (*transform.up * flySpeed*), kar omogoča nadzorovan premik navzgor.

5.2.2.4 Preverjanje padanja robota

Funkcija *FallCheck()* sproti preverja, ali robot pada – bodisi zaradi izgube tal pod nogami ali napake v algoritmu. Če je robot pritrjen z magnetom na kovinsko površino, se ob določenih pogojih spremeni njegova hitrost. V primeru prostega padca pa se hitrost ustavi in sproži se animacija padanja.

Programska koda funkcije, ki je prikazana na sliki (Priloga 10.1, Slika 64), poskrbi za preverjanje vertikalne hitrosti robota (*rb.linearVelocity.y*), kar omogoča zaznavanje gibanja navzgor ali navzdol. Če je robot pritrjen z magnetom, se ob večji hitrosti navzgor ustavi (*speed* = *0f*), če pa ni pritrjen z magnetom, se ob večji hitrosti navzdol ustavi, kar preprečuje nenadzorovano premikanje, ko je v magnetnem polju.

V običajnem stanju, če robot pada s hitrostjo manjšo od -1 in je še vedno v stanju hoje (*movingOnGround*), se:

- ustavi njegova hitrost,
- vključi status padanja (*falling* = *true*),
- zažene animacija padanja z oznako *falling_b*.

To preprečuje nenaravne premike robota in omogoča, da se vizualno pravilno odzove na nenadne padce, kot so napake v programu ali nepravilno postavljene ploščadi.

5.2.2.5 Prikaz kape in vrtenje propelerja

Funkcija *CapCheck()* preveri, ali je robot pridobil kapo za letenje. Če jo ima, se kapa prikaže na modelu robota (Slika 36), če je nima, ostane skrita (Slika 37). V primeru letenja pa ta funkcija aktivira tudi vrtenje propelerja, ki je pritrjen na kapo.



Slika 36: Prikaz robota s kapo



Slika 37: Prikaz robota brez kape

Funkcija *CapCheck()*, prikazana na sliki (Priloga 10.1, Slika 65), najprej preveri, ali ima robot aktivirano lastnost *hasFlyCap*, kar pomeni, da je nekje v igri pobral kapo za letenje. Glede na rezultat se ustrezno nastavi vidnost kape s *SetActive(true)* ali *SetActive(false)*.

V zadnjem delu funkcije se nahaja pogoj za *rotateCap*. Če je ta aktiviran, to pomeni, da robot trenutno leti in je treba zavrteti propeler, ki je del vizualnega učinka letenja. Vrtenje se izvaja s pomočjo funkcije *Rotate()*, ki stalno obrača propeler okoli svoje osi (os Z) s hitrostjo, prilagojeno času (*Time.deltaTime*), da gibanje ostane tekoče in neodvisno od hitrosti sličic.

5.2.3 Posebna funkcija – uporaba trampolina za visoki skok

Funkcija *PlaceSpring()* poskrbi, da robot uporabi trampolin, če ga ima v svojem inventarju. Ko se trampolin postavi pod robota, ta nemudoma odskoči (Slika 38) – do štiri polja navzgor in eno polje naprej. Trampolin ni na voljo, če je robot pritrjen na kovinsko gred z magnetom.



Slika 38: Prikaz robota po odboju trampolina

Programska koda na sliki (Priloga 10.1, Slika 66) prikazuje funkcijo *PlaceSpring()*, ki najprej preveri, ali ima igralec na voljo trampolin (*gameManager.springs > 0*) in ali trenutno ni aktiviran magnet. Če so pogoji ustrezni, se ustvari nov objekt trampolina s pomočjo *Instantiate()* in ga postavi na določeno točko (*springSpawnPoint*). Ta nov trampolin se doda v seznam vseh ustvarjenih trampolinov (*spawnedSprings*), ki jih aplikacija spremlja tudi za kasnejše ponastavitve. Ob uspešni postavitvi se število razpoložljivih trampolinov zmanjša.

Če pogoji za postavitev niso izpolnjeni (npr. ni več trampolinov ali je robot na magnetu), se preveri stanje ukazov in po potrebi nadaljuje izvajanje naslednjega ukaza prek *RunCommands()*.

6 TESTIRANJE APLIKACIJA NA PROSTOVOLJCIH

Aplikacijo smo seveda testirali mi sami, vendar samokritika ni vedno najbolj iskrena, zato je bil glavni vir povratnih informacij profesor (Slika 34), saj nam je podal iskren odziv na izkušnjo in predlagal nove posodobitve oz. nove dodane funkcionalnosti.

Ker smo projekt razvijali v sklopu šole in so nam pri razvijanju bili zelo v pomoč medsebojni nasveti, smo želeli, da aplikacijo testirajo tudi drugi dijaki, kar bi nam dalo še najbolj realen odziv na naš izdelek in tudi najbolj realno mnenje o potencialnih posodobitvah in nadgradnjah, ki bi jih dodali uporabniki sami.



Slika 39: Profesor testira aplikacijo

Na šoli smo med dijaki izvedli množično testiranje aplikacije, po zaključku testiranja pa so sodelujoči izpolnili še anketo. Anketa je bila izvrsten vir pridobivanja povratnih informacij od že izkušenih ali pa tudi manj izkušenih posameznikov s področja računalništva.

Anketiranje je potekalo po letnikih, od prvega do četrtega letnika. Vsi so preizkušali enako verzijo aplikacije, in to na isti dan, predvsem zaradi enakovredne izkušnje (Slika 35, Slika 36, Slika 37, Slika 38, Slika 39 in Slika 40). Vsakemu izmed sodelujočih smo ponudili možnost, da lahko po zaključku stopnje odigra še eno poljubno, saj so tako lahko dijaki pridobili več informacij o sami ideji ter aplikaciji in v anketi ponudili več mnenj in predlogov. Reševanje anket je potekalo anonimno, odgovarjali pa so na nekaj osnovnih vprašanj, lahko pa so posredovali tudi kakšno dodatno idejo za nadgradnjo aplikacije. Ob koncu razvijanja aplikacije so nam taki nasveti prišli še kako prav, saj se pogosto zgodi, da sami ne opazimo možnosti za potencialne spremembe, ki jih splošni uporabnik opazi takoj. Na osnovi teh odzivov smo aplikacijo lahko še izboljšali.



Slika 40: Dijak David testira aplikacijo



Slika 41: Dijak Kai testira aplikacijo



Slika 42: Razvijalec Tai testira aplikacijo



Slika 43: Dijak Tjaš testira aplikacijo



Slika 44: Dijaki Aljaž, Jakob in Uroš testirajo aplikacijo



Slika 45: Dijak Peter testira aplikacijo

7 ANALIZA REZULTATOV IN UGOTOVITVE

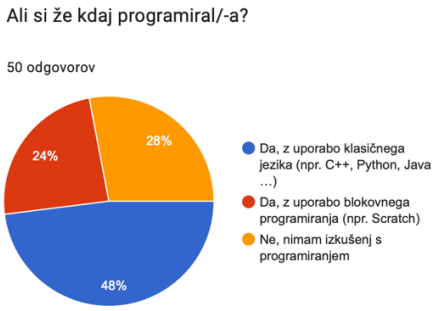
V tem poglavju bomo povzeli vse naše ugotovitve in tako odgovorili na zastavljena vprašanja ter potrdili ali ovrgli zastavljene hipoteze.

7.1 Rezultati ankete

V anketi je sodelovalo skupno 50 dijakov iz različnih srednjih strokovnih izobraževalnih programov, ki jih izvajamo na Srednji poklicni in tehniški šoli Murska Sobota. Najprej smo želeli o anketirancih izvedeti nekaj splošnih podatkov, kar smo združili v rubriko »O udeležencu«. Zanimalo nas je, kakšno je njihovo predznanje računalništva oz. programiranja (Graf 1), ali so že kdaj programirali (Graf 2) in kateri letnik obiskujejo (Graf 3).



Graf 1: Samoocena predznanja računalništva in/ali programiranja med udeleženci



Graf 2: Izkušnje anketirancev s programiranjem



Graf 3: Razporeditev anketirancev po letniku srednje šole

Večina anketirancev je imela predhodno znanje s področja računalništva – 38 % jih je ocenilo svoje znanje kot dobro, 34 % kot osnovno, 28 % pa kot zelo dobro (Graf 1). Skoraj polovica sodelujočih v anketi (48 %) je že programirala v klasičnih jezikih (npr. C++, Python, Java ...), 24 % jih je uporabljalo blokovno programiranje (npr. Scratch), 28 % pa ni imelo predhodnih izkušenj (Graf 2). Največ anketirancev prihaja iz 1. letnika (34 %), preostali pa so enakomerno razporejeni med višje letnike (Graf 3). Podatki kažejo, da je bila skupina raznolika tako po znanju kot po letniku, ki ga obiskujejo, kar omogoča celostno oceno uporabnosti aplikacije za različne ravni predznanja.

V naslednji rubriki »Izkušnja z VR okoljem« nas je zanimalo, ali je to bila za anketirance prva izkušnja z virtualno resničnostjo (Graf 4), kako hitro so se znašli pri uporabi VR okolja (Graf 5) ter ali so občutili kakšne neprijetne težave med uporabo VR (Graf 6).

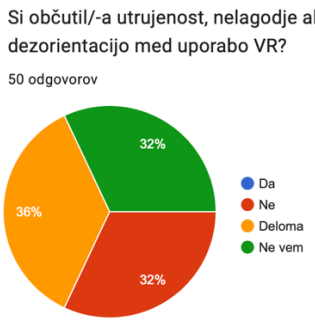
Več kot polovica dijakov (56 %) je že imela izkušnje z virtualno resničnostjo, za tretjino (32 %) pa je bila to prva izkušnja (Graf 4). Kljub temu se je večina (64 %) v VR okolju dobro znašla (Graf 5), kar potrjuje intuitivno zasnovo aplikacije. Nekateri so poročali o začetni zmedenosti, kar bi lahko pripisali dejstvu, da v trenutni različici igre vodič sicer obstaja, a ga uporabnik lahko preskoči. Zato bi bilo smiselno razmisliti o sistemski rešitvi, ki bi ob prvi uporabi zahtevala obvezno dokončanje stopnje vodiča, preden uporabnik nadaljuje z drugimi stopnjami.



Graf 4: Prva izkušnja udeležencev z virtualno resničnostjo



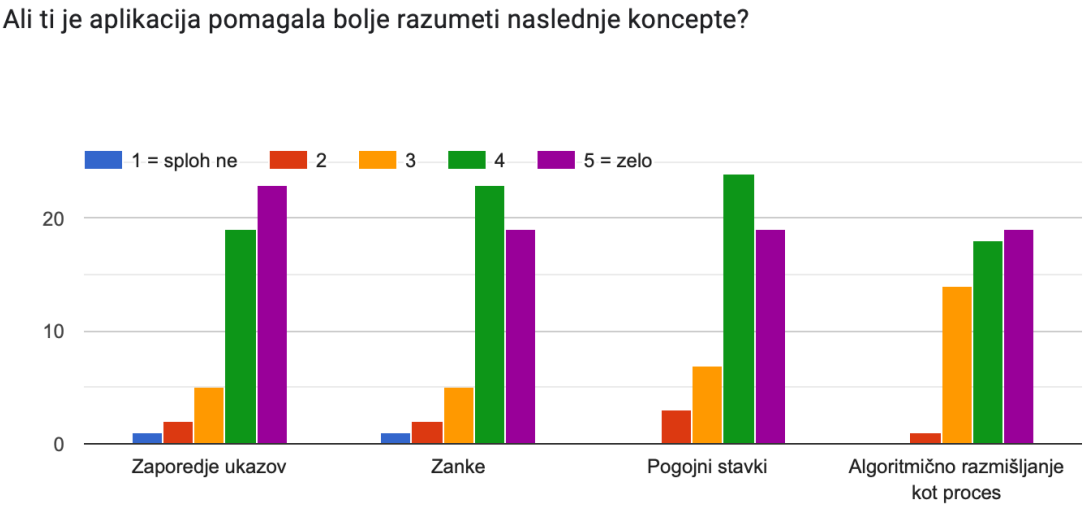
Graf 5: Uporabniška orientacija v VR okolju – gibanje in krmilniki



Graf 6: Fizično počutje med uporabo VR

Glede fizičnega počutja (Graf 6) so bili odgovori razdeljeni – tretjina ni občutila težav, tretjina je zaznala nelagodje, preostali delno utrujenost. To je značilen pojav pri uporabi VR tehnologij in nas opominja na pomen dobrega tempa, omejitve trajanja in možnosti odmora. V nadaljnjem razvoju aplikacije bi bilo smiselno razmisliti o vizualnih opozorilih glede trajanja igranja ali priporočilu za odmor. Čeprav je bila uporaba VR za nekatere prva izkušnja, se je večina uporabnikov uspešno prilagodila. Aplikacija se je izkazala kot zanimiva, dostopna in učinkovita izhodiščna točka za spoznavanje VR okolja.

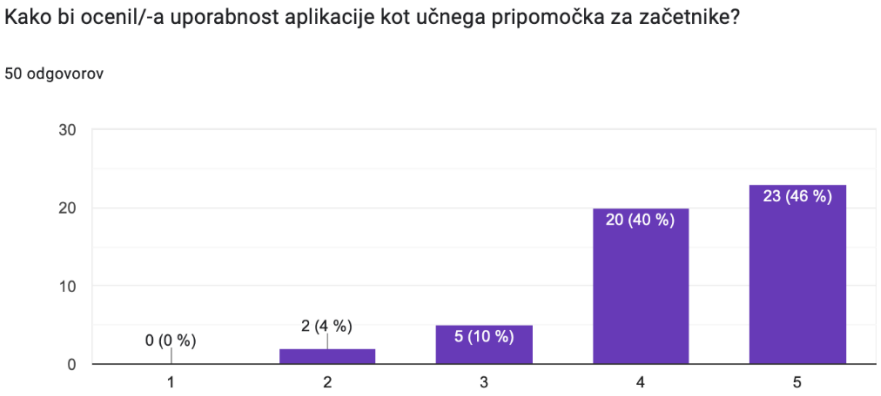
V nadaljevanju sledi rubrika »Razumevanje konceptov in učni učinek«, ki je za nas še toliko bolj pomembna, saj lahko ugotovimo, ali ima naša aplikacija tudi pozitiven učni učinek na uporabnika. V tem delu nas je zanimalo, ali so anketiranci s pomočjo aplikacije bolje razumeli koncepte (Graf 7), kako bi ocenili uporabnost aplikacije kot učnega pripomočka za začetnike (Graf 8) ter kakšna je težavnost stopenj v igri (Graf 9).



Graf 7: Razumevanje posameznih konceptov programiranja

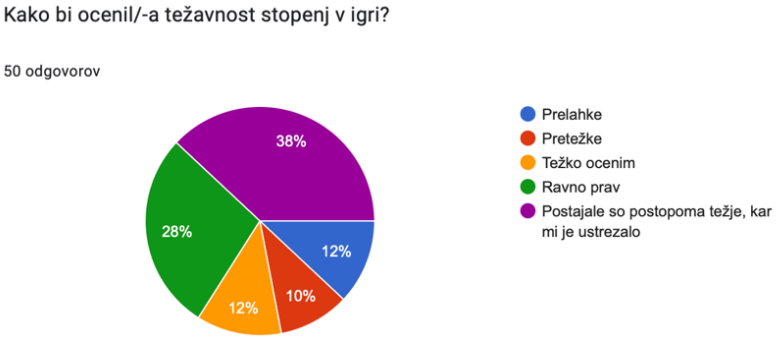
Rezultati v grafu (Graf 7) kažejo, da je velika večina dijakov (več kot 80 %) dosegla boljše razumevanje ključnih konceptov, kot so zaporedje ukazov, zanke, pogojni stavki in celo abstraktnejše algoritmično razmišljanje. To potrjuje učinkovitost blokovnega sestavljanja v VR okolju kot metode za vizualno učenje logike in algoritmičnega načina razmišljanja.

Tudi splošna uporabnost aplikacije kot učnega pripomočka za začetnike je bila ocenjena zelo visoko – 86 % dijakov je podalo oceno 4 ali 5 (Graf 8), kar potrjuje, da aplikacija ni zgolj zanimiva, ampak tudi didaktično učinkovita in primerna za uvajanje v programiranje.



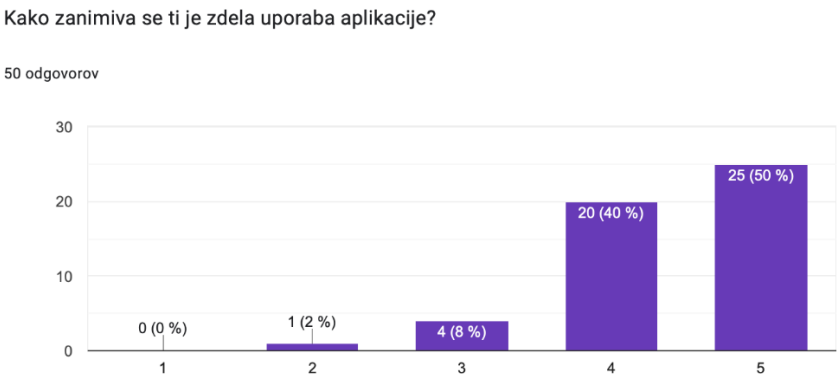
Graf 8: Uporabnost aplikacije kot učnega pripomočka za začetnike

Večina dijakov je ocenila, da je bila težavnost stopenj primerno uravnotežena (Graf 9). Največ (38 %) jih je menilo, da so stopnje postopoma postajale težje, kar jim je ustrezalo, dodatnih 28 % pa jih je ocenilo kot ravno prav zahtevne. Manjši delež dijakov je stopnje doživljal kot prelahke (12 %) ali pretežke (10 %), kar je pričakovana posledica individualnih razlik. Skupno lahko zaključimo, da aplikacija uspešno vzpostavlja postopno naraščajočo kompleksnost, ki večino uporabnikov motivira.



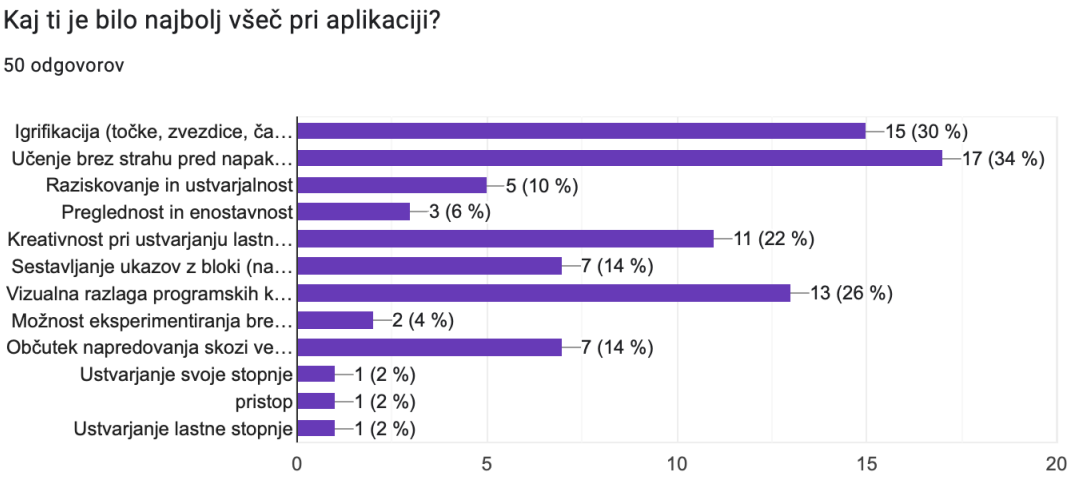
Graf 9: Ocena težavnosti stopenj v igri

V anketo smo vključili tudi rubriko »Motivacija in izkušnja«, kjer nas je zanimalo, kako uporabna se jim zdi aplikacija (Graf 10) in kaj jim je bilo najbolj všeč (Graf 11).



Graf 10: Prikaz ocen zanimivosti uporabe aplikacije

Vprašanje o zanimivosti uporabe aplikacije (Graf 10) je pokazalo visoko motivacijsko vrednost: kar 90 % dijakov je ocenilo izkušnjo z ocenama 4 ali 5, polovica (50 %) celo z najvišjo možno. Le peščica je podala nižje ocene. Tak rezultat kaže, da aplikacija ni le učinkovit učni pripomoček, temveč tudi privlačno in angažirano okolje za raziskovanje osnov programiranja. Združevanje učenja in igrifikacije se je izkazalo kot ključni element motivacije.

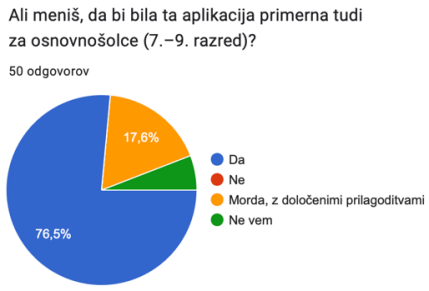


Graf 11: Kaj ti je bilo najbolj všeč pri aplikaciji?

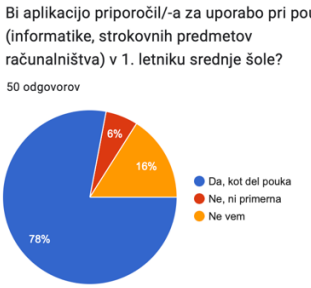
Rezultati (Graf 11) kažejo, da je uporabnikom najbolj ustrezala kombinacija varnega učnega okolja, igrifikacije in vizualnega pristopa. Največ dijakov (34 %) je izpostavilo učenje brez strahu pred napakami, kar potrjuje, da aplikacija omogoča sproščeno eksperimentiranje brez občutka kaznovanja. Igrifikacija (30 %) – zvezdice, točke, časovni izzivi – je bila pomemben motivacijski dejavnik, ki je povečal angažiranost. Visoko se je uvrstila tudi vizualna razlaga programskih konceptov (26 %), ki je uporabnikom pomagala bolje razumeti temeljne pojme, kot so zanke, pogoji in zaporedja. Kreativnost pri ustvarjanju stopenj (22 %) je omogočila izražanje idej in razvijanje algoritmičnega mišljenja. Podobno pomembni so bili še: sestavljanje ukazov z bloki in občutek napredovanja (oba po 14 %), ki krepita uporabniško izkušnjo in občutek napredka. Skupno rezultati kažejo, da aplikacija uspešno združuje učenje, ustvarjalnost in motivacijo v vizualno privlačnem okolju.

Tovrstna kombinacija elementov nakazuje, da je aplikacija premišljeno zasnovana ne le kot učni pripomoček, ampak tudi kot spodbujevalec zanimanja za algoritmično razmišljanje, ki predstavlja temeljni korak k razumevanju programiranja.

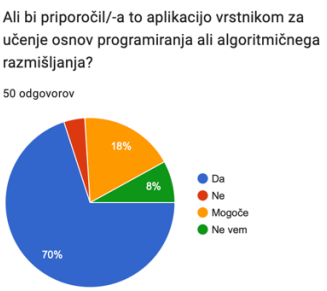
Pri pripravi in testiranju aplikacije smo imeli ves čas v mislih njen učni namen, zato smo v anketo vključili tudi rubriko »Pedagoška uporabnost in potencial«. Naš cilj je, da bi se aplikacija uporabljala tudi kot del pouka – najprej pri uvajanju v programiranje, pozneje pa širše, kot orodje za razvijanje algoritmičnega razmišljanja, ki presega zgolj področje programiranja in je uporabno na številnih področjih učenja in reševanja problemov.



Graf 12: Mnenje o primernosti aplikacije za osnovnošolce (7.–9. razred)



Graf 13: Priporočilo za uporabo aplikacije pri pouku v 1. letniku srednje šole



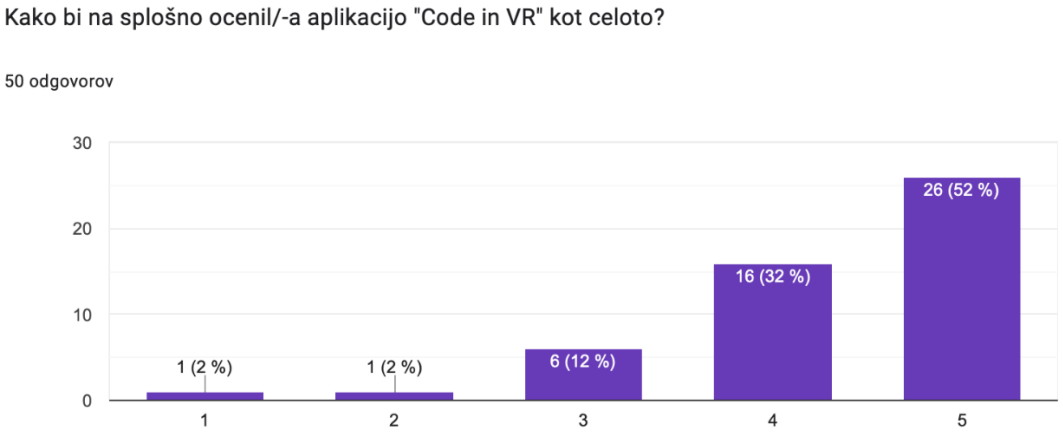
Graf 14: Bi priporočil/-a vrstnikom?

Večina anketirancev meni, da je aplikacija primerna tudi za osnovnošolce. Kar 76,5 % vprašanih meni, da bi jo brez večjih težav lahko uporabljali učenci 7.–9. razreda. Dodatnih 17,6 % bi aplikacijo priporočilo z določenimi prilagoditvami, kar pomeni, da skoraj vsak anketiranec prepoznava potencial za uporabo tudi pri mlajših učencih. Le peščica (5,9 %) ni bila povsem prepričana (Graf 12). To potrjuje, da je aplikacija že dovolj dostopna, hkrati pa pušča odprt prostor za nadaljnje izboljšave uvodnih vsebin in prilagojenih izzivov za najmlajše.

Še jasnejša pa je podpora uvedbi aplikacije v srednješolski pouk. Kar 78 % dijakov bi jo priporočilo kot del učnega procesa v 1. letniku srednje šole, kar nakazuje na visoko pedagoško vrednost aplikacije tudi z vidika samih uporabnikov. Le 6 % jih meni, da za pouk ni primerna, kar je zanemarljivo malo (Graf 13).

Aplikacijo bi svojim vrstnikom priporočila velika večina dijakov. Kar 70 % bi jo brez zadržkov priporočilo za učenje osnov programiranja ali algoritmičnega razmišljanja, dodatnih 18 % pa meni, da bi to morda bilo smiselno. Le majhen del dijakov (4 %) je odgovoril nikalno. To potrjuje, da je aplikacija prepoznana kot učinkovito orodje za začetek učenja računalniškega razmišljanja (Graf 14).

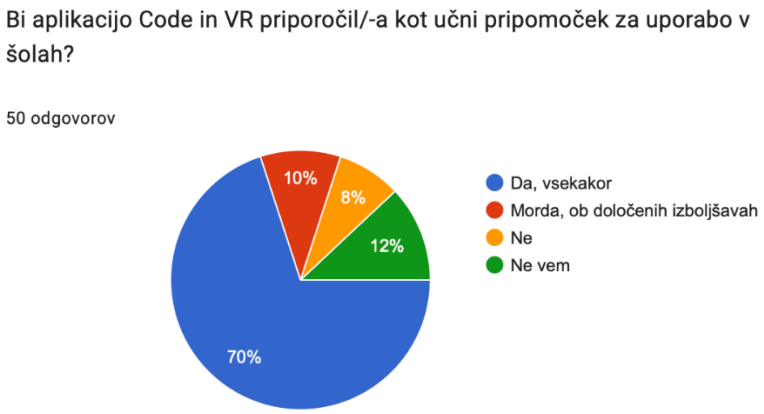
V zaključku ankete smo vključili še rubriko »Sklepna ocena in mnenje«, kjer nas je zanimala splošna ocena aplikacije kot celote (Graf 15) in ali bi aplikacijo priporočili kot učni pripomoček za uporabo v šolah (Graf 16).



Graf 15: Kako bi na splošno ocenil/-a aplikacijo »Code in VR« kot celoto?

Zaključna ocena aplikacije »Code in VR« kaže visoko raven zadovoljstva – kar 84 % anketirancev ji je dodelilo oceno 4 ali 5, od tega več kot polovica najvišjo možno (Graf 15). Negativne ocene so bile redke, kar potrjuje, da je večina uporabnikov aplikacijo doživela kot uspešno in uporabno.

Tudi vprašanje o priporočilu za uporabo v šolah potrjuje njen potencial: 70 % vprašanih bi jo z veseljem priporočilo kot učni pripomoček, dodatnih 8 % pa ob določenih izboljšavah (Graf 16). Skupno kar 78 % uporabnikov vidi aplikacijo kot primeren didaktični pripomoček, kar poudarja njeno pedagoško vrednost in splošno sprejemljivost med dijaki.



Graf 16: Bi aplikacijo »Code in VR« priporočil/-a kot učni pripomoček za uporabo v šolah?

Ker smo izvajali anketo, se nam je zdelo zelo smiselno vključiti tudi rubriko »Predlogi in izboljšave«, saj nam lahko prav anketiranci dajo največ predlogov in idej. Zanimalo nas je, katere dodatne funkcionalnosti bi si želeli v prihodnosti, ali so med igranjem naleteli na težave, nejasnosti ali zmedo ter če so na voljo neomejene možnosti, kaj bi dodali, da bi bila aplikacija še boljša. V nadaljevanju na kratko povzemamo rezultate.

Udeleženci so v tej rubriki ankete podali več konstruktivnih predlogov, ki bi lahko pripomogli k še boljši uporabniški izkušnji z aplikacijo. Med najbolj izpostavljenimi je bila želja po namigih ali analizi rešitve po končani stopnji, predvsem v primerih, ko uporabnik ne doseže maksimalnega števila točk ali obtiči pri nalogi. Takšna povratna informacija bi dodatno okrepila učni vidik in spodbujala refleksijo.

Nekateri dijaki so izrazili tudi željo po večji interaktivnosti z elementi v okolju, čeprav večino teh elementov aplikacija že vključuje. Zanimiv je bil predlog, da bi lahko objekte, kot so škatle, tudi fizično premikali, kar bi odprlo nove možnosti za reševanje izzivov in še dodatno spodbudilo prostorsko in logično razmišljanje. V povezavi z nadgradnjo programerskih znanj so nekateri udeleženci predlagali uvajanje kompleksnejših blokov, kot so funkcije ali vnaprej pripravljeni sklopi ukazov, ki bi omogočili sestavljanje bolj strukturiranih rešitev in razumevanje koncepta ponovne uporabe kode. Zanimiva zamisel je bila tudi uvajanje sistema namigov ob zastanku – npr. če uporabnik določen čas ne napreduje – ter tekmovalni način v realnem času, ki bi dodal element dinamike in primerjanja napredka z drugimi igralci. Nekateri udeleženci so izrazili tudi željo po krajši programski kodi kot alternativi blokovnemu pristopu, kar bi lahko služilo kot naravni prehod za bolj napredne uporabnike. Skupna ugotovitev teh predlogov je, da

aplikacija že zdaj ponuja širok in učinkovit nabor funkcionalnosti, obenem pa odpira prostor za kreativno nadgradnjo in individualno prilagajanje učne poti.

S tem se je zaključilo testiranje aplikacije oz. igre s strani dijakov. Anketiranje je bilo uspešno, pridobili smo veliko koristnih povratnih informacij ter smernic, kako razmišljati v prihodnje in nadaljevati razvoj. Dobili smo vpogled v to, kaj dijaki menijo o taki aplikaciji, kako jim je všeč in kaj jim predstavlja v smislu pouka. Zagotovo bi v prihodnje bilo zanimivo izvesti še dodatno testiranje na mlajših populaciji, ki morda še ne razmišlja o programiranju in programskih jezikih. V letošnjem šolskem letu žal nismo imeli priložnosti, da bi dijakom v 1. letniku pred koraki v programski jezik omogočili preizkus aplikacije, saj smo z razvojem aplikacije začeli pozneje. Med mnenji v anketi smo lahko tudi zasledili, da so nekateri dijaki 1. letnika zapisali, da bi tovrstni pristop bil še posebej zanimiv in zaželen, ne pa, da so v sklopu pouka začeli z diagrami poteka kot prvimi koraki v svet programiranja in razvijanja algoritmičnega oz. logičnega razmišljanja.

7.2 Potrditev hipotez

V tem poglavju povzemamo rezultate in ugotovitve, do katerih smo prišli med razvojem in testiranjem aplikacije. Na podlagi teh rezultatov bomo odgovorili na zastavljena raziskovalna vprašanja ter potrdili ali ovrgli hipoteze.

H1: Virtualna resničnost lahko učinkovito simulira proces programiranja in pomaga razvijati algoritmično razmišljanje.

Po zaključku projekta ugotavljamo, da se z razvitim VR okoljem močno približamo osnovnim principom programiranja, predvsem z vidika razvoja algoritmičnega razmišljanja. Aplikacija uporabnikom omogoča, da skozi interaktivne izzive razvijajo logično mišljenje, reševanje problemov ter sposobnost postopnega načrtovanja rešitev, kar so ključni elementi, ki tvorijo temelj programiranja. Poudarek je na razumevanju zaporedja, zank in pogojev skozi vizualno sestavljanje ukazov, kar omogoča simulacijo programerske logike brez potrebe po dejanskem pisanju kode. Na podlagi naših opažanj in povratnih informacij uporabnikov lahko hipotezo **potrdimo**, saj aplikacija učinkovito spodbuja razvoj algoritmičnega razmišljanja in uporabnikom približa strukturo programiranja na dostopen način.

H2: Uporaba VR aplikacije kot učnega pripomočka povečuje motivacijo za učenje programiranja in spodbuja razumevanje osnov algoritmičnega razmišljanja.

Eden izmed ključnih ciljev razvite aplikacije je bil spodbujanje motivacije za učenje programiranja ter pomoč uporabnikom pri razumevanju osnov algoritmičnega razmišljanja, kot so zaporedje ukazov, ponavljanje (zanke) in pogoji. Anketni rezultati in praktični preizkusi kažejo, da interaktivna narava VR okolja pozitivno vpliva na uporabnikovo vključenost, saj omogoča eksperimentiranje, vizualizacijo in takojšnjo povratno informacijo. Uporabniki so izrazili, da so skozi igro lažje razumeli logiko postopkovnega razmišljanja, kar jim je pomagalo pri dojemanju temeljnih konceptov,

ki so osnova programiranja. Na podlagi teh ugotovitev lahko hipotezo **potrdimo**, saj je aplikacija pomembno prispevala k večji motivaciji in razumevanju algoritmične strukture problema.

H3: Možno je združiti igrifikacijo in izobraževanje na način, ki olajša učenje začetnikom ter omogoča postopno povečevanje težavnosti tudi za naprednejše uporabnike.

Aplikacija ponuja dostopno, vizualno podprto učno okolje, ki začetnikom omogoča razumevanje osnov algoritmičnega razmišljanja, hkrati pa naprednejšim uporabnikom ponuja izzive višje zahtevnosti, kjer lahko poglobijo svoje logično sklepanje in optimizacijo postopkov. Z vključitvijo igrifikacijskih elementov, kot so različne vrste nalog, vizualna privlačnost, časovni izzivi ter možnost ustvarjanja lastnih stopenj, smo dosegli, da aplikacija uporabnikom ostaja zanimiva, spodbuja angažiranost ter ustvarja občutek napredovanja. Igrifikacija se je tako izkazala kot učinkovit pristop za povečanje vključenosti uporabnikov in krepitev dolgoročne motivacije za učenje. Na podlagi opažanj in povratnih informacij lahko hipotezo **potrdimo**, saj aplikacija uspešno združuje izobraževalne cilje in elemente igre, kar pozitivno vpliva na učni proces tako pri začetnikih kot pri naprednejših uporabnikih.

H4: Virtualna resničnost lahko v celoti nadomesti klasično programiranje in služi kot učinkovit uvod v razumevanje programskih konceptov.

Na podlagi analize ugotavljamo, da virtualna resničnost lahko služi kot učinkovit uvod v razumevanje osnovnih programskih konceptov, vendar v trenutni obliki ne more v celoti nadomestiti klasičnega programiranja. Aplikacija omogoča vizualno predstavitev in praktično preizkušanje konceptov, kot so zaporedje ukazov, zanke in pogoji, kar uporabnikom pomaga razvijati algoritmično razmišljanje. Kljub temu aplikacija ne vključuje pisanja dejanske programske kode, kar je ključni element pri učenju programiranja v tradicionalnem pomenu. Manjka torej stik s sintakso in z vsemi napakami, ki jih mora uporabnik obvladati pri realnem kodiranju. Zato hipotezo **delno potrjujemo** – VR aplikacija se je izkazala kot učinkovit pripomoček za uvod v programersko razmišljanje in razumevanje logike programov, vendar v svoji trenutni obliki ne more v celoti nadomestiti klasičnih pristopov k učenju programiranja.

H5: Aplikacijo bi bilo možno uporabiti kot dopolnilo pri pouku programiranja, saj omogoča praktično učenje v varnem okolju brez strahu pred napakami.

Povratne informacije anketirancev potrjujejo, da aplikacija prispeva k boljšemu razumevanju osnovnih programskih konceptov, saj omogoča njihovo vizualizacijo in takojšnjo povratno informacijo o delovanju. Uporabniki lahko na enostaven način eksperimentirajo z različnimi rešitvami, preizkušajo različne pristope in brez negativnih posledic popravljajo napake, kar ustvarja sproščeno in spodbudno učno okolje. Takšen pristop zmanjšuje strah pred napakami, ki je pogosto prisoten pri začetnikih pri učenju programiranja in lahko zavira motivacijo. Zaradi interaktivnosti, vizualne podpore in možnosti ponavljanja v varnem okolju aplikacija deluje kot učinkovito dopolnilo klasičnemu pouku. Na podlagi ugotovitev lahko hipotezo **potrdimo**.

7.3 Ugotovitve

Na podlagi zbranih podatkov in opravljene analize ugotavljamo, da ima razvita VR aplikacija dobro zasnovo, saj uspešno združuje interaktivnost, izobraževalno vrednost in elemente igrifikacije. Rezultati raziskave kažejo, da tak pristop olajša razumevanje osnovnih konceptov algoritmičnega razmišljanja ter spodbuja večje zanimanje in motivacijo za učenje programiranja, zlasti pri začetnikih. Aplikacija uporabnikom omogoča intuitivno sestavljanje programskih ukazov, postopno uvajanje zahtevnejših konceptov ter eksperimentiranje brez strahu pred napakami, kar pomembno prispeva k bolj sproščenemu in učinkovitemu učenju.

Zavedamo se, da ima aplikacija tudi določene omejitve. Tako naša aplikacija v trenutni obliki ne more v celoti nadomestiti klasičnega programiranja, saj ne vključuje pisanja dejanske kode ali dela s sintakso, kar je ključnega pomena v nadaljnjih fazah učenja. Poleg tega obstajajo priložnosti za nadgradnjo, kot so vključitev naprednejših programerskih izzivov, širši nabor stopenj ter možnost primerjave rešitev med uporabniki. Eden izmed predlogov za nadgradnjo je uvedba sistema zvezdic, kjer bi moral uporabnik doseči določeno število točk za napredovanje na zahtevnejše stopnje. Tako bi spodbujali uporabnike, da vsako nalogo temeljito preučijo, poiščejo optimizirane rešitve in razvijajo občutek za kakovostno ter berljivo rešitev.

Pri razvoju smo se osredotočili predvsem na uporabniško izkušnjo in motivacijo začetnikov. V ta namen smo v aplikacijo vključili različne tipe izzivov, časovno omejene naloge, naključno generirane stopnje in kreativne elemente, ki povečujejo raznolikost in ponovljivost igranja. Pomemben vidik je tudi vizualizacija programskih konceptov, ki omogoča uporabnikom razumevanje logike in strukture algoritmov, ne da bi se hkrati ukvarjali z zahtevno sintakso.

Ugotavljamo, da aplikacija ni zgolj orodje za linearno reševanje nalog, temveč platforma, ki spodbuja raziskovanje, eksperimentiranje in vizualno učenje. To je tudi eden izmed ključnih ciljev aplikacije – omogočiti začetnikom motivacijsko bogat in dostopen vstop v svet programerskega razmišljanja. Povratne informacije uporabnikov potrjujejo, da je tak način poučevanja učinkovit in da bi lahko dopolnjeval klasične metode, sprva na osnovni ravni, kasneje pa – z nadgradnjo aplikacije – tudi pri naprednejšem učenju programiranja.

Aplikacija ima velik potencial kot izobraževalno orodje, zlasti v začetnih fazah učenja. Čeprav ne nadomešča učenja pisanja kode, nudi alternativen in spodbuden način za razumevanje osnovnih konceptov. Na tej podlagi lahko sklenemo, da VR tehnologija lahko pomembno prispeva k izboljšanju dostopnosti in razumljivosti programerskih vsebin, saj uporabnikom omogoča vizualizacijo problemov in razmislek o lastnih rešitvah. Posebej primerna se izkaže za začetnike, ki se prvič srečujejo s programskimi izzivi.

8 ZAKLJUČEK

Čeprav smo uspeli potrditi velik del zastavljenih hipotez, se zavedamo, da izkušnja programiranja v igri ne more v celoti nadomestiti tradicionalnega programiranja. Kljub temu pa lahko virtualna resničnost učinkovito uprizori logično razmišljanje in pomaga pri razumevanju osnovnih programskih konceptov, obenem pa nudi močno motivacijo za začetnike, ki se prvič srečujejo s programerskimi izzivi.

Zavedamo se, da ima aplikacija velik potencial na področju razvijanja in nadgrajevanja pristopa k učenju programiranja. Verjamemo, da lahko z nadaljnjim razvojem dosežemo, da aplikacija ne le spodbuja programsko razmišljanje in osnovne veščine, temveč tudi krepi željo po učenju in nadaljnjem raziskovanju področja programiranja.

Z izdelkom smo zadovoljni, saj smo v razvojnem procesu presegli naša začetna pričakovanja in vključili številne funkcionalnosti, ki v začetni fazi niso bile predvidene. Poleg tehničnih znanj smo pridobili dragocene izkušnje tudi na področju timske dinamike, projektnega vodenja in reševanja izzivov, ki so del razvoja VR aplikacij.

Aplikacijo nameravamo v prihodnje še razvijati in dopolnjevati, saj smo tudi sami ob delu na projektu pridobili dodatno motivacijo, ko smo prepoznali njen izobraževalni potencial. V zadnjih tednih smo dodali več kot 20 novih stopenj, še nekaj jih imamo pripravljenih, v prihodnosti pa jih bo na voljo kar 60. Poleg tega smo dodali dva nova svetova ter omogočili, da lahko uporabnik spremlja razvoj algoritma, ki ga je sestavil. Pripravljamo tudi možnost, da uporabnik namesto sestavljanja programskih blokov uporablja poenostavljeno programsko kodo ter tako preklaplja med vizualnim in kodnim vmesnikom. Ker Srednja poklicna in tehniška šola Murska Sobota dijakom omogoča uporabo VR očal, imajo ti priložnost aplikacijo redno preizkušati ter podajati povratne informacije, ki pogosto vodijo do novih idej in predlogov za nadgradnjo.

Naš cilj ostaja jasen: z nadaljnjim razvojem in izboljšavami želimo potrditi vse hipoteze v še večji meri, kar bi pomenilo, da smo aplikacijo izpopolnili do točke, ko lahko učinkovito dopolnjuje tradicionalne učne metode in predstavlja pomembno podporo pri učenju programiranja, zlasti za začetnike.

9 VIRI

[1] »Navidezna resničnost«, Wikipedia. [Elektronski]. Povezava: https://sl.wikipedia.org/wiki/Navidezna_resni%C4%8Dnost [Dostopno: 8. februar 2025].

[2] »History of VR – Timeline of Events and Tech Development«, VirtualSpeech. [Elektronski]. Povezava: https://virtualspeech.com/blog/history-of-vr [Dostopno: 8. februar 2025].

[3] »Virtual reality«, Wikipedia. [Elektronski]. Povezava: https://en.wikipedia.org/wiki/Virtual_reality#History [Dostopno: 8. februar 2025].

[4] A. Cvetko in V. Smodiš, Aplikacija za sestavljanje računalnika v virtualni resničnosti, raziskovalna naloga ZOTKS, Srednja poklicna in tehniška šola Murska Sobota, Murska Sobota, 2023. [Elektronski]. Povezava: https://zbirke.zotks.si/2023/resources/SS_aplik_3431.pdf [Dostopno: 8. februar 2025].

[5] »How can virtual reality be used in military training«, drawandcode. [Elektronski]. Povezava: https://drawandcode.com/learning-zone/how-can-virtual-reality-be-used-in-military-training/ [Dostopno: 10. februar 2025].

[6] »Applications of Virtual Reality in Medicine, Medical life sciences. [Elektronski]. Povezava: https://www.news-medical.net/health/Applications-of-Virtual-Reality-in-Medicine.aspx [Dostopno: 10. februar 2025].

[7] »The Leaked Quest 3 Headset Video Teases Meta's VR Ambitions«, wired. [Elektronski]. Povezava: https://www.wired.com/story/meta-quest-3-headset-video-leak/ [Dostopno: 15. februar 2025].

[8] »The Sword of Damocles - Head Mounted Display«, D'source. [Elektronski]. Povezava: https://www.dsource.in/course/virtual-reality-introduction/evolution-vr/sword-damocles-head-mounted-display [Dostopno: 22. marec 2025)

10 PRILOGA

10.1 Programska koda

```
void StartProgram()
{
    socket = programmingTablet.GetComponent<XRSocketInteractor>();
    count = 0;
    StartGame();
}

void StartGame() { StartCoroutine(StartGameCoRutine()); }

IEnumerator StartGameCoRutine()
{
    commandBlocksList.Clear();
    minus = 1;

    while (socket != null)
    {
        commandBlock = socket.GetOldestInteractableSelected();

        if (commandBlock != null)
        {
            commandBlocksList.Add(commandBlock.transform);
            socket = commandBlock.transform.GetComponent<XRSocketInteractor>();
        }
        else
            socket = null;
    }

    usedBlocks += commandBlocksList.Count;
    gameRunning = true;

    if (!gameReset)
        robot.transform.position = robotScript.tileCenter;
    else
    {
        robot.transform.position = startTile.transform.position + Vector3.up * 0.32f;
        Quaternion robotRotation = startTile.transform.rotation * new Quaternion(0, 1, 0, 0);
    }

    gameReset = false;
    yield return new WaitForSeconds(0.05f);
    RunCommands();
    yield return new WaitForSeconds(0.1f);

    if (isTimerLevel)
        timer.didGameRun = true;
}
```

Slika 46: Zaslonski posnetek kode – Zagon programa in zbiranje ukazov

```
public void RunCommands()
{
    if (!commandExecuting && count < commandBlocksList.Count && gameRunning)
    {
        if (commandBlocksList[count] != null)
        {
            commandScript = commandBlocksList[count].GetComponent<RunCommand>();

            if (commandBlocksList[count].GetComponent<ChangeMaterialWLIBlock>() != null)
            {
                if (changeBlockMat != null)
                {
                    changeBlockMat.SetDefaultMaterial();
                    changeBlockMat = null;
                }
                changeBlockMat = commandBlocksList[count].GetComponent<ChangeMaterialWLIBlock>();
                changeBlockMat.SetCurrentMaterial();
            }

            commandExecuting = true;
        }
        else
        {
            commandScript = null;
            gameRunning = false;
            if (changeBlockMat != null)
            {
                changeBlockMat.SetDefaultMaterial();
                changeBlockMat = null;
            }
        }

        count++;
    }
    else
    {
        commandScript = null;
        gameRunning = false;
        if (changeBlockMat != null)
        {
            changeBlockMat.SetDefaultMaterial();
            changeBlockMat = null;
        }
    }

    if (commandScript != null)
        commandScript.RunBlockCommand();
}
```

Slika 47: Zaslonski posnetek kode - Izvajanje ukazov z uporabo funkcije RunCommands

```
public void ResetScene(string errorText) { StartCoroutine(ResetCoRutine(errorText)); }

IEnumerator ResetCoRutine(string errorText)
{
    errorCanvas.SetActive(true);
    errorCanvasText.text = errorText;
    errorCanvasIMEG.transform.localScale = new Vector3(1, .8f, 1);
    errorCanvasIMEG.transform.localPosition = new Vector3(errorCanvasIMEG.transform.localPosition.x, -2, errorCanvasIMEG.gameObject.transform.localPosition.z);

    reachedEnd = false;
    commandScript = null;
    usingMagnets = false;
    robotScript.ResetAnimations();
    usedBlocks = 0;
    gameRunning = false;
    count = 0;

    foreach (var block in commandBlocksList)
    {
        if (block == null) continue;

        if (block.GetComponent<ChangeMaterialWLIBlock>() != null)
        {
            changeBlockMat = block.GetComponent<ChangeMaterialWLIBlock>();
            changeBlockMat.SetDefaultMaterial();
        }
    }

    commandBlocksList.Clear();
    commandExecuting = false;

    robotScript.move = false;
    robotScript.fly = false;
    robotScript.tiles = 0;
    robotScript.speed = 1;
    robotScript.rotate = false;
    robotScript.isInMagnetField = false;

    robotScript.StopAllCoroutines();

    ClearBombs();

    yield return new WaitForSeconds(0.05f);

    robotScript.rotateCap = false;
    robotScript.flying = false;

    StartCoroutine(SetBlocksActive());
```

Slika 48: Zaslonski posnetek kode - Resetiranje prizorišča (1. del)

```
    yield return new WaitForSeconds(0.05f);

    if (robotScript.lastChekPoint != null)
    {
        robot.transform.position = robotScript.lastChekPoint.transform.position;
        robot.transform.rotation = robotScript.lastChekPoint.transform.rotation;
        robotScript.currentRotateY = robotScript.lastChekPoint.transform.eulerAngles.y;
        bombs = bombSave;
        hasFlyCap = flySave;
        springs = springsSave;
        hasMagnets = magnetsSave;
        usingMagnets = usingMagnetsSave;
        usedBlocks = saveUsedBlocks;

        yield return new WaitForSeconds(0.1f);
        robotScript.useGravity = true;

        StartCoroutine(ResetObjectsSinceCP());
        ClearObjectsSinceCP();
    }
    else
    {
        gameReset = true;
        RobotToStart();
        bombs = startBombs;
        hasFlyCap = startFly;
        springs = startSprings;
        hasMagnets = startMagnets;
        usingMagnets = false;
        robotScript.currentRotateY = 0;
        fullReset = true;
        ClearSprings();

        yield return new WaitForSeconds(0.1f);
        robotScript.useGravity = true;

        StartCoroutine(ResetPowerUpsNdObstacles());
    }
}
```

Slika 49: Zaslonski posnetek kode - Resetiranje prizorišča (2. del)

```
public void LevelCompleted (string victoryText)
{
    float timeScore = 0;
    float blockScore = 0;

    star1.SetActive(false);
    star2.SetActive(false);
    star3.SetActive(false);

    SetButtonsNotInterractable();

    if (!isTimerLevel)
    {
        if (usedTime <= TimeToSolve)
            timeScore = 100f;
        else if (usedTime > TimeToSolve)
            timeScore = 100 - (usedTime - TimeToSolve);
        if (minBlocksToSolve[currentLevel] != 0)
        {
            if (usedBlocks < minBlocksToSolve[currentLevel])
                blockScore = 100;
            else
                blockScore = minBlocksToSolve[currentLevel] / usedBlocks * 100f;
        }

        score += timeScore + blockScore;

        if (blockEndText)
        {
            if (lang == 0)
                timeText.text = "Time used: " + usedTime.ToString("F2") + "/" + TimeToSolve + "s";
            else if (lang == 1)
                timeText.text = "Porabljen čas: " + usedTime.ToString("F2") + "/" + TimeToSolve + "s";


            if (TimeToSolve == float.PositiveInfinity)
```

Slika 50: Zaslonski posnetek kode - Zaključek stopnje in izračun rezultatov

```
if (other.gameObject.CompareTag("End"))
{
    if (mainCol.bounds.Intersects(other.bounds))
    {
        if (lang == 0)
            gameManager.LevelCompleted("Congratulations! You successfully completed the challenge!");
        else if (lang == 1)
            gameManager.LevelCompleted("Čestitke! Uspešno ste zaključili izziv!");

        audioSource.PlayOneShot(coinPickUp);
        Instantiate(areYaWinninSan, transform.position, transform.rotation);
        Destroy(other.gameObject);
    }
}
```

Slika 51: Zaslonski posnetek kode - Aktivacija zaključka stopnje ob trku z zlatnikom

```
public void Next()
{
    //Debug.Log("NEXT");
    RobotToStart();

    if (lang == 0)
        ResetCompletely("Loaded Next Level");
    else if (lang == 1)
        ResetCompletely("Naložena naslednja stopnja");

    continueCanvas.SetActive(false);
    reachedEnd = false;
    obstacleCourses[currentLevel].SetActive(false);
    currentLevel++;

    if (currentLevel > obstacleCourses.Length - 1)
    {
        if (oldScore < score)
            PlayerPrefs.SetFloat(LevelScoreKey, score);

        if (levelInt > PlayerPrefs.GetInt("LevelsCompleted"))
            PlayerPrefs.SetInt("LevelsCompleted", levelInt);

        tabletScript.AllOff();
        MainMenu();
    }
    else
    {
        obstacleCourses[currentLevel].SetActive(true);
        SetButtonsInterractable();
    }
}
```

Slika 52: Zaslonski posnetek kode - Prehod v naslednjo stopnjo



Slika 53: Zaslonski posnetek komponente – Prikaz skripte RobotScript v Unityju

```
public void MoveXTiles(int x)
{
    tiles = x;
    movingOnGround = true;
    audioSource.clip = walk;
    audioSource.loop = true;
    MoveTiles();
}

void MoveTiles ()
{
    if (tiles > 0)
    {
        move = true;
        robotAnimator.SetBool("walking_b", true);
    }

    if (tiles == 0)
    {
        robotAnimator.SetBool("walking_b", false);
        move = false;
        movingOnGround = false;
        transform.position = tileCenter;
        if (reset)
        {
            gameManager.commandExecuting = false;
            gameManager.RunCommands();
        }
        else if (commandSocketScript != null)
            commandSocketScript.RunCommands();

        gameManager.returnErrorTime = 5f;
    }
}
```

Slika 54: Zaslonski posnetek kode – Premikanje robota po določenem številu polj

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("CenterPoint"))
    {
        if (mainCol.bounds.Intersects(other.bounds))
        {
            tiles--;
            speed = normalSpeed;
            flySpeed = normalFlySpeed;
            tileCenter = new Vector3(other.transform.position.x, transform.position.y, other.transform.position.z);
            MoveTiles();
            gameManager.returnErrorTime = 5f;
            blockCenterTransform = other.transform;
        }
    }
```

Slika 55: Zaslonski posnetek kode – Zaznava vstopa robota v novo polje

```
void MoveCheck()
{
    if (move)
        rb.linearVelocity = transform.forward * speed + new Vector3(0, rb.linearVelocity.y);
    else
        rb.linearVelocity = new Vector3(0, rb.linearVelocity.y, 0);
}
```

Slika 56: Zaslonski posnetek kode – Preverjanje gibanja robota

```
public void Turn(string direction)
{
    if (direction == "right" || direction == "desno")
    {
        rotate = true;
        rotateDir = true;
        currentRotateY = transform.eulerAngles.y + 90;
        robotAnimator.SetInteger("rotate_i", 2);
    }

    if (direction == "left" || direction == "levo")
    {
        rotate = true;
        rotateDir = false;
        currentRotateY = transform.eulerAngles.y - 90;
        robotAnimator.SetInteger("rotate_i", 1);
    }

    StartCoroutine(FinishRotating());
}

IEnumerator FinishRotating()
{
    yield return new WaitForSeconds(1);

    if (rotateDir && rotate)
        transform.rotation = Quaternion.Euler(0, currentRotateY, 0);
    else if (rotate)
        transform.rotation = Quaternion.Euler(0, currentRotateY, 0);

    rotate = false;
    robotAnimator.SetInteger("rotate_i", 0);

    yield return new WaitForSeconds(0.2f);

    if (reset)
    {
        gameManager.commandExecuting = false;
        gameManager.RunCommands();
    }
    else
        commandSocketScript.RunCommands();

    gameManager.returnErrorTime = 5f;
}
```

Slika 57: Zaslonski posnetek kode – Funkcija za obračanje robota

```
void RotateCheck()
{
    if (rotate)
    {
        if (rotateDir)
            transform.Rotate(Vector3.up * rotateSpeed * Time.deltaTime);
        else
            transform.Rotate(Vector3.up * -rotateSpeed * Time.deltaTime);
    }
}
```

Slika 58: Zaslonski posnetek kode – Določanje rotacije robota med obračanjem

```
public void Jump(float y, float flySpeedVar, bool useSpeed, bool useMoveOnGround, bool spring)
{
    if (gameManager.usingMagnets && isInMagnetField && !spring)
        StartCoroutine(JumpAnimator(y, -flySpeedVar, useSpeed, useMoveOnGround));
    else
        StartCoroutine(JumpAnimator(y, flySpeedVar, useSpeed, useMoveOnGround));
}

IEnumerator JumpAnimator(float multiplier, float flySpeedVar, bool useSpeed, bool x)
{
    fly = true;
    audioSource.PlayOneShot(jump);
    robotAnimator.SetBool("jumping_b", true);
    flySpeed = flySpeedVar;

    if (!useSpeed)
        flySpeedVar = 1;

    move = true;
    if (tiles <= 0)
        tiles = 1;

    MoveTiles();
    yield return new WaitForSeconds(0.185f * multiplier / flySpeedVar);

    if (x)
        movingOnGround = true;

    fly = false;
    robotAnimator.SetBool("jumping_b", false);
    speed = normalSpeed;
    flySpeed = normalFlySpeed;
}
```

Slika 59: Zaslonski posnetek kode – Funkcija skoka robota z nastavitvijo višine in hitrosti

```
void FlyCheck()
{
    if (fly)
        rb.linearVelocity = transform.up * flySpeed;
}
```

Slika 60: Zaslonski posnetek kode – Dvigovanje robota z nastavitvijo hitrosti leta

```
void FallCheck()
{
    if (gameManager.usingMagnets && isInMagnetField)
    {
        if (rb.linearVelocity.y > 1 && movingOnGround)
            speed = 0f;
    }
    else
    {
        if (rb.linearVelocity.y < -1 && movingOnGround)
        {
            speed = 0f;
            falling = true;
            robotAnimator.SetBool("falling_b", true);
        }
    }
}
```

Slika 61: Zaslonski posnetek kode – Preverjanje padanja robota

```
void CapCheck()
{
    if (gameManager.hasFlyCap)
        flyCap.SetActive(true);
    else
        flyCap.SetActive(false);

    if (rotateCap)
        propeller.transform.Rotate(Vector3.forward * 1000 * Time.deltaTime);
}
```

Slika 62: Zaslonski posnetek kode – Preverjanje prisotnosti leteče kape

```
public void PlaceSpring()
{

    if (gameManager.springs > 0 && !(gameManager.usingMagnets && isInMagnetField))
    {
        GameObject springTemp = Instantiate(spring, springSpawnPoint.transform.position, spring.transform.rotation);
        springTemp.transform.SetParent(gameManager.obstacleCourses[gameManager.currentLevel].transform);
        gameManager.spawnedSprings.Add(springTemp);
        gameManager.spawnedSpringsSinceCP.Add(springTemp);
        gameManager.springs--;
    }
    else
    {
        if (reset)
        {
            gameManager.commandExecuting = false;
            gameManager.RunCommands();
        }
        else
            commandSocketScript.RunCommands();
    }

    gameManager.returnErrorTime = 5f;
}
```

Slika 63: Zaslonski posnetek kode – Postavitev trampolina

10.2 Anketa

Povratna informacija o aplikaciji Code in VR – izkušnja, uporabnost in mnenje

Pozdravljeni!

Hvala, ker sodelujete pri testiranju naše aplikacije **Code in VR**, ki smo jo razvili dijaki SPTŠ MS – izobraževalne VR igre, namenjene spodbujanju algoritmičnega razmišljanja in osnovnemu razumevanju programskih konceptov.

Vaše mnenje nam je zelo pomembno, saj nam bo pomagalo izboljšati uporabniško izkušnjo, vsebino in pedagoško vrednost aplikacije. Anketo izpolnite iskreno – ne ocenjujemo vas, temveč želimo razumeti, kako uporabna in dostopnaje aplikacija za vas in druge učence.

Izpolnjevanje ankete vam bo vzelo približno 5 minut. Vsi odgovori so anonimni, rezultati pa bodo uporabljeni izključno za namen raziskave in nadaljnji razvoj aplikacije.

Tai L., Tine M. in Jure Z. (mentor: Dominik L.)

O UDELEŽENCU

Ali si že kdaj programiral/-a? *

- ◯ Da, z uporabo klasičnega jezika (npr. C++, Python, Java ...)
- ◯ Da, z uporabo blokovnega programiranja (npr. Scratch)
- ◯ Ne, nimam izkušenj s programiranjem

Kako bi ocenil/-a svoje predznanje računalništva/programiranja? *

- ◯ Zelo dobro
- ◯ Dobro
- ◯ Osnovno
- ◯ Nimam znanja
- ◯ Ne vem

Kateri letnik obiskuješ? *

- ◯ 1. letnik
- ◯ 2. letnik
- ◯ 3. letnik
- ◯ 4. letnik

IZKUŠNJA Z VR OKOLJEM

Je bila to tvoja prva izkušnja z virtualno resničnostjo? *

- ◯ Da
- ◯ Ne
- ◯ Ne spomnim se

Kako enostavno si se znašel/-a pri uporabi VR okolja (premikanje, uporaba krmilnikov)? *

- ◯ Zelo enostavno
- ◯ Brez večjih težav
- ◯ Malo zmedeno
- ◯ Težko

Si občutil/-a utrujenost, nelagodje ali dezorientacijo med uporabo VR? *

- ◯ Da
- ◯ Ne
- ◯ Deloma
- ◯ Ne vem

RAZUMEVANJE KONCEPTOV IN UČNI UČINEK

Ali ti je aplikacija pomagala bolje razumeti naslednje koncepte? *

	1 = sploh ne	2	3	4	5 = zelo
Zaporedje ukazov	☐	☐	☐	☐	☐
Zanke	☐	☐	☐	☐	☐
Pogojni stavki	☐	☐	☐	☐	☐
Algoritmično razmišljanje kot proces	☐	☐	☐	☐	☐

Kako bi ocenil/-a uporabnost aplikacije kot učnega pripomočka za začetnike? *

	1	2	3	4	5	
sploh ni uporabna	○	○	○	○	○	zelo uporabna

Kako bi ocenil/-a težavnost stopenj v igri? *

- ○ Prelahke
- ○ Pretežke
- ○ Težko ocenim
- ○ Ravno prav
- ○ Postajale so postopoma težje, kar mi je ustrezalo

MOTIVACIJA IN IZKUŠNJA

Kako zanimiva se ti je zdela uporaba aplikacije? *

	1	2	3	4	5	
sploh ne	○	○	○	○	○	zelo zanimiva

Kaj ti je bilo najbolj všeč pri aplikaciji? *

- [ ] Igrifikacija (točke, zvezdice, časovni izzivi)
- [ ] Učenje brez strahu pred napakami
- [ ] Raziskovanje in ustvarjalnost
- [ ] Preglednost in enostavnost
- [ ] Kreativnost pri ustvarjanju lastne stopnje
- [ ] Sestavljanje ukazov z bloki (namesto pisanja kode)
- [ ] Vizualna razlaga programskih konceptov (npr. zanke, pogoji)
- [ ] Možnost eksperimentiranja brez posledic (napake niso kaznovane)
- [ ] Občutek napredovanja skozi vedno težje stopnje
- [ ] Drugo: _____

PREDLOGI IN IZBOLJŠAVE

Katere dodatne funkcionalnosti bi si želel/-a v prihodnosti? *

- [ ] Bolj raznolike izzive (npr. logični labirinti, več korakov, pasti)
- [ ] Več povratnih informacij po končani stopnji (ocena, namig, analiza)
- [ ] Možnost primerjave rešitve z drugimi (npr. prikaz drugačnih rešitev)
- [ ] Več interakcije z elementi okolja (npr. sprožilci, objekti - škatla, vrata)
- [ ] Dodatno vodenje ali pomoč pri težjih stopnjah
- [ ] Drugo: _____

Si med igranjem naletel/-a na kaj, kar ti ni bilo jasno, nerazumljivo ali zmedeno?

Vaš odgovor

Če bi imel/-a neomejene možnosti – kaj bi sam/-a dodal/-a v aplikacijo, da bi bila še boljša?

Vaš odgovor

PEDAGOŠKA UPORABNOST IN POTENCIAL

Ali meniš, da bi bila ta aplikacija primerna tudi za osnovnošolce (7.–9. razred)? *

- ◯ Da
- ◯ Ne
- ◯ Morda, z določenimi prilagoditvami
- ◯ Ne vem

Bi aplikacijo priporočil/-a za uporabo pri pouku (informatike, strokovnih predmetov računalništva) v 1. letniku srednje šole? *

- ◯ Da, kot del pouka
- ◯ Ne, ni primerna
- ◯ Ne vem

Ali bi priporočil/-a to aplikacijo vrstnikom za učenje osnov programiranja ali algoritmičnega razmišljanja? *

- ◯ Da
- ◯ Ne
- ◯ Mogoče
- ◯ Ne vem

ODPRTA VPRAŠANJA

Kaj ti je bilo pri aplikaciji najbolj všeč?

Vaš odgovor

Kaj bi izboljšal/-a ali dodal/-a?

Vaš odgovor

Imaš še kakšno drugo mnenje, izkušnjo ali predlog glede uporabe aplikacije pri učenju?

Vaš odgovor

SKLEPNA OCENA IN MNENJE O APLIKACIJI

Kako bi na splošno ocenil/-a aplikacijo "Code in VR" kot celoto? *

	1	2	3	4	5	
Zelo slaba	○	○	○	○	○	Odlična

Bi aplikacijo Code in VR priporočil/-a kot učni pripomoček za uporabo v šolah? *

- ○ Da, vsekakor
- ○ Morda, ob določenih izboljšavah
- ○ Ne
- ○ Ne vem

Kaj bi povedal/-a o aplikaciji, če bi jo moral/-a predstaviti prijatelju, učitelju ali novemu uporabniku? *

Vaš odgovor