UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

IGRA ZA PODPORO RAZVOJU MOŽGANSKO-RAČUNALNIŠKIH VMESNIKOV

JAN TOMŠIČ PIVK

2021
ZAKLJUČNA NALOGA
TOMŠIČ PIVK

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Igra za podporo razvoju možgansko-računalniških vmesnikov

(A game to support brain-computer interface evolution)

Ime in priimek: Jan Tomšič Pivk
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, avgust 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Jan Tomšič Pivk

Naslov zaključne naloge: Igra za podporo razvoju možgansko-računalniških vmesnikov

Kraj: Koper

Leto: 2021

Število listov: 31 Število slik: 12 Število tabel: 4

Število referenc: 4

Mentor: doc. dr. Peter Rogelj

Ključne besede: možgansko-računalniški vmesnik, računalniška igra, programsko inženirstvo

Izvleček:

V diplomski nalogi je z vidika programskega inženirstva opisan postopek načrtovanja, razvoja in testiranja igre za pomoč pri razvoju možgansko-računalniških vmesnikov. Igra je v obliki dirkalne igre iz pogleda tretje osebe, kjer se igralec izogiba prihajajočim oviram. Igro je mogoče prek nastavitev konfigurirati, da je združljiva z različnimi paradigmami BCI. Nastavitve omogočajo tudi spreminjanje težavnosti. Za povezavo s sistemom BCI je definiran komunikacijski kanal prek protokola UDP. Uporabnik je za povezavo dolžan sam ustvariti preprost komunikacijski vmesnik. Za primer uporabe in testiranje je bil ustvarjen kontrolni modul, ki za vhod namesto komponente BCI uporablja tipkovnico.

Key document information

Name and SURNAME: Jan Tomšič Pivk

Title of the final project paper: A game to support brain-computer interface evolution

Place: Koper

Year: 2021

Number of pages: 31 Number of figures: 12 Number of tables: 4

Number of references: 4

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: brain-computer-interface, computer game, software engineering

Abstract:

The final project paper describes the process of planning, development and testing a game intended to aid the evolution of Brain-Computer Interfaces, from the perspective of software engineering. The game takes the form of a third person racing game, where the player avoids incoming obstacles. The game can be configured to better fit various BCI paradigms via its settings. The settings also allow for the adjustment of the game's difficulty. For connecting the game to a BCI system, a UDP based communications channel is defined. The user of the software can use the game through a custom interface module, which is out of the scope of this thesis. The game was tested using a simple keyboard interface module that demonstrates the control interface principles.

ZAHVALA

Hvala ...

... mentorju doc. dr. Petru Roglju za vse strokovne usmeritve, ideje in popravke.

... družini, ki mi je študij omogočila in me skozi vsa leta potrpežljivo spodbujala.

... prijateljem, ki so me spodbujali vsak na svoj način, z vsem od toplih besed do ostre kritike.

Brez vas bi bilo vse to nemogoče.
Hvala!

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO SLIK IN GRAFIKONOV

SEZNAM KRATIC

BCI – Možgansko-računalniški vmesnik (ang. Brain-Computer Interface)
EEG – Elektroencefalografija (ang. Electroencephalography)
UDP – Nepovezovalni protokol za prenašanje paketov (ang. User Datagram Protocol)
SDL – ang. Simple DirectMedia Layer
SSVEP – ang. Steady-State Visually Evoked Potentials

1 UVOD

Možgansko-računalniški vmesnik (ang. Brain-Computer Interface) je direktna komunikacijska povezava med možgani in računalnikom. Možgansko-računalniški vmesniki se pogosto uporabljajo za pomoč, povečanje ali izboljšanje človeških kognitivnih ali senzomotoričnih funkcij, kot so vid, sluh, gibanje in komuniciranje [1].

1.1 Paradigme možgansko-računalniških vmesnikov

Razvoj možgansko-računalniških vmesnikov vključuje razvoj algoritmov analize elektroencefalografskih (EEG) signalov, kot tudi s tem povezanimi načini upravljanja. V tem trenutku so najbolj uporabljane tri paradigme. Prva je P300, kjer značilen EEG-odziv na dogodek omogoči razpoznavo elementa pozornosti, kjer se je dogodek zgodil. Druga je SSVEP (ang. steady-state visually evoked potentials), kjer pozornost uporabnika na utripajoči element ojača prisotnost frekvence utripanja tudi v EEG-signalih, s tem pa razpoznavanje elementa pozornosti. Tretja paradigma je namišljeno gibanje (ang. motor imagery), kjer je ključ identifikacija EEG-odziva na gibanje, oziroma namišljeno gibanje, katerega odziv je primerljiv s pravim gibanjem [2].

1.2 Definicija problema

Za razvoj možgansko-računalniških vmesnikov potrebujemo okolja, kjer lahko preizkušamo različne načine upravljanja, četudi končne aplikacije delujejo v drugačnem okolju. V tej zaključni nalogi želimo razviti okolje za razvoj BCI v obliki računalniške igre. Da bi bila igra primerna za to, bi morala izpolnjevati nekaj pogojev: težavnost bi morala biti nastavljiva, mogoče bi moralo biti prilagajanje različnim paradigmam BCI, vključno z možnostjo postavitve animiranih slik na zaslon zaradi paradigme SSVEP, in idealno bi igra delovala v okolju Linux.

1.2.1 Težavnost

Na možgansko-računalniške vmesnike se je treba privaditi oziroma se jih je treba naučiti uporabljati. Da bi lahko uporabnik preko procesa učenja postopno pridobival veščine učinkovite uporabe vmesnika, bi moralo biti težavnost igre mogoče spreminjati. V osnovi bi bila preprosta in počasna, z največ dvema različnima ukazoma (na primer levo/desno). Z višanjem težavnosti bi lahko povečevali tudi hitrost in dodajali mogoče dodatne komande (gor/dol ali še kaj drugega). Lepo bi bilo, če bi bila (pri višjih zahtevnostih) igra atraktivna tudi za običajne igralce.

1.2.2 Prilagajanje paradigmam BCI

Ker je cilj igre, da bi se jo lahko uporabljalo z več različnimi sistemi BCI, bi deloma moralo biti mogoče tudi prilagajanje, ki bi omogočalo izvedbo različnih paradigem BCI. Ti sistemi bi delovali kot zunanji moduli, ki bi lahko igro upravljali prek vmesnika. Kritično je, da bi bil razvoj vmesnika čim preprostejši. Namen igre bi bil namreč, da raziskovalcem olajšamo delo.

Zunanji modul bi lahko bil klasičen (na osnovi upravljanja z miško ali tipkovnico), omogočeni pa bi bili tudi kompleksnejši kontrolerji, ki bi lahko temeljili na analizi EEG in delovali kot BCI, lahko pa tudi kakšni drugi tehnologiji, na primer spodbujevalno učenje (ang. reinforcement learning). Zmožnost ustvarjanja vmesnika bi bilo uporabno demonstrirati, na primer z enostavnejšim vmesnikom na osnovi tipkovnice.

Komunikacija z zunanjim kontrolerjem bi morala biti obojesmerna, tako da ne bi šlo le za posredovanje ukazov, pač tudi za obveščanje kontrolerja o stanju igre. To bi omogočalo beleženje iger in učenje (ali postopkov BCI ali avtomatskega igralca z AI).

Ker se nekatere paradigme BCI zanašajo na prepoznavanje različnih vidnih vzorcev, bi bilo treba omogočiti postavitev poljubnih animiranih slik GIF ali drugih oblik animacij na zaslon (prikazano igralno površino).

1.2.3 Podpora za Linux

Linux je brezplačen operacijski sistem, podoben Unixu, s prosto dostopno izvorno kodo. Ker razvoj možgansko-računalniških vmesnikov pogosto poteka v njem, je prednost, če igra deluje v tem okolju.

1.3 Študija izvedljivosti

Za predlogo oblike igre izbiramo med tremi vzpostavljenimi žanri, ki se na prvi pogled zdijo primerni. Pri vseh primerjamo, kakšen minimalen vhod potrebujejo, do kakšne mere je mogoče prilagajati njihovo težavnost in kako se skladajo s tremi omenjenimi paradigmami BCI.

1.3.1 Top-down racer

Prvi je dirkalna igra iz ptičje perspektive (ang. top-down racer). En ali več igralcev upravljajo vozila, s katerimi dirkajo v krogih po eni ali več pistah. Za žanr so značilni, ampak ne

obvezni, pojavi velikega števila računalniško vodenih nasprotnikov in več možnih poti do cilja. Značilen je tudi koncept različnih »priboljškov« (ang. pick-ups), kot na primer začasna povečana hitrost, ki se jih pobira na določenih mestih na pisti.

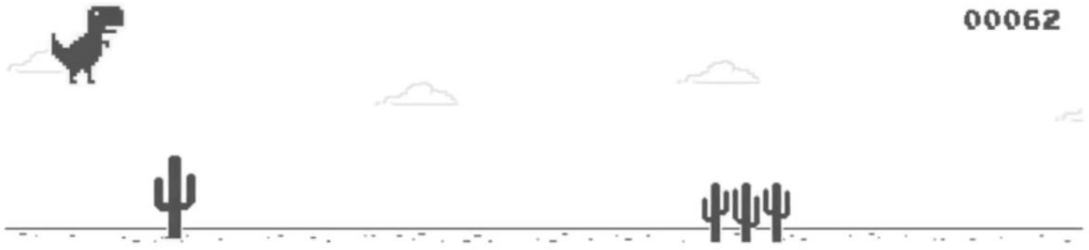
Slika 1: Pogled iz ptičje perspektive v igri Race Arcade (2017) [3]

Vhod
Žanr je primeren za BCI, ker je preprost in ker je igra v osnovi zelo samoumevna. Ob stalni hitrosti bi lahko upravljanje omejili na dva vhoda: zavoj levo in desno. Težava je v tem, da je vhod neprekinjen – se ga ne da na preprost način razdeliti na zaporedje ukazov s časovnim zamikom. To bi lahko otežilo implementacijo BCI po paradigmah P300 in SSVEP.

Težavnost
Ob trku z mejo piste bi se lahko avtomobil ob njej začel drsati ali se odbil nazaj. Tako bi igralec ob napaki vedno imel priložnost za popravek. To bi skupaj z omejitvijo hitrosti igro zelo poenostavilo. Za povečanje težavnosti bi lahko povečali hitrost, igralca huje kaznovali za trk z mejo piste, dodali hitre računalniško vodene nasprotnike, čeprav bi lahko bila implementacija računalniško vodenih nasprotnikov zelo zamudna, in zapletli vozni model (na primer dodali zdrs ob nenadnem zavijanju).

1.3.2 Endless runner

V t. i. endless runner igralec upravlja figuro, ki stalno teče od leve proti desni. Na voljo ima nekaj ukazov, kot na primer skok in sklanjanje, s katerimi se izogiba približujočim oviram.



Slika 2: Igralec skače čez oviro v preprosti igri Chrome Dino

Vhod
Upravljanje je lahko v najpreprostejši različici omejeno le na en vhod, na primer skok. V tem primeru bi igralec moral skakati ob pravem času. Če so časovne omejitve neprimerne, bi se lahko upravljanje omejilo na dva vhoda, s katerima igralec izbira pravilen način za izogib oviri. Ker so vsi vhodi diskretni, ta žanr ni zelo primeren za paradigmo BCI motor imagery.

Težavnost
Kot je bilo že omenjeno, bi bila igra najlažja z majhnim številom vhodov. Dodatno se jo lahko še upočasni oziroma ustavi pred vsako oviro. Ustavljanje pred vsako oviro s samo enim vhodom bi bilo neprimerno, saj igralec ne bi imel več nikakršnega smiselnega nadzora nad igro. Igro se lahko otežuje z večanjem hitrosti in vrst ovir. Ovire se lahko tudi zaplete tako, da njihova vrsta ni popolnoma jasna, dokler se jim igralec ne približa. To se lahko izrazi kot na primer zid, ki ga je sprva mogoče preskočiti, nato pa se dvigne v zrak, da se mora igralec posledično pod njim skloniti.

1.3.3 Staromodna dirkalna igra

Ko govorimo o staromodnih dirkalnih igrah (ang. old school racing games), imamo v mislih tipičen pogled od zadaj in iluzijo tridimenzionalne perspektive. Igralec se v nadzoru dirkalnega avtomobila izogiba drugim vozilom na cesti. Žanr je za BCI primeren, ker je potek igre zelo intuitiven.

Slika 3: Igralec se izogiba oviram v dirkalni igri Outrun (1986) [4]

Vhod

Igro se tipično upravlja z dvema ukazoma za zavoj levo in desno. Doda se lahko še ukaze za pospešek in zaviranje, če je igra snovana tako, da se je z raznimi ovirami optimalno soočiti ob različnih hitrostih. Namesto neprekinjenega zavijanja (v praksi premikanja levo in desno po zaslonu) se lahko položaj avtomobila omeji na cestne pasove in tako ukaze spremeni v diskretne, kar je primerneje za paradigme BCI P300 in SSVEP. Različni ukazi lahko tudi določajo, na kateri pas se premaknemo (na primer ukaz 1 nas postavi na prvi pas od leve proti desni itd.).

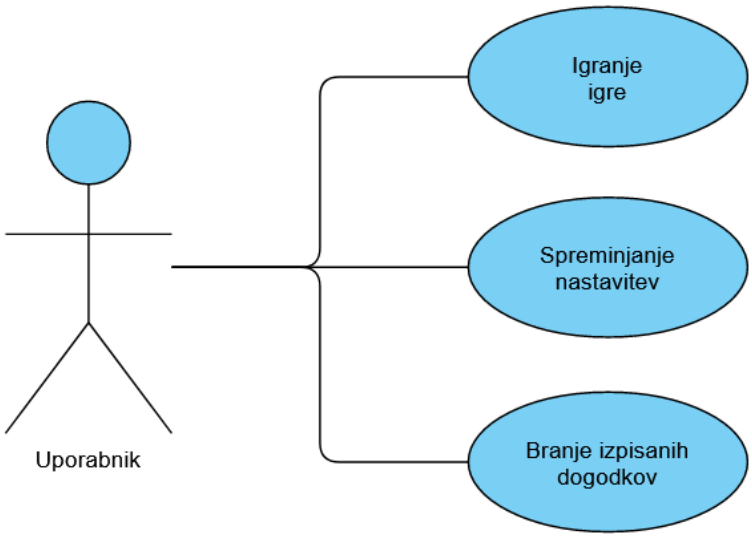
Težavnost

Težavnost se lahko omeji z zmanjšano hitrostjo ali v ekstremu ustavitvijo igre, ko se igralec znajde pred oviro. Za večjo težavnost se lahko poveča hitrost, doda na cesto ovire, ki se nam približujejo z različnimi hitrostmi, in implementira ovinke, kjer je treba skladno z njimi zavijati.

Ker izmed vseh treh najbolje izpolnjuje zadane pogoje, je bil izbran žanr staromodne dirkalne igre.

2 SPECIFIKACIJE IGRE

Za igro so bile določene naslednje funkcijske zahteve. Ker mora igra nuditi okolje za preizkušanje sistemov BCI, jo mora biti mogoče igrati. Da je igro možno igrati, jo mora biti mogoče upravljati in igralcu mora biti na neki način vidno njeno stanje. Igra mora torej sprejemati ukaze in svoje stanje izrisovati na zaslon. Ker mora biti igra nastavljiva za prilagajanje različnim paradigmam BCI, morajo obstajati nastavitve, ki lahko igro tem paradigmam prilagajajo, in te nastavitve mora biti mogoče spreminjati. Podobno mora biti omogočeno tudi spreminjati nastavitve, povezane s težavnostjo. Da je potek igre mogoče strojno analizirati v realnem času, mora kontrolnemu modulu med izvajanjem sporočati informacije o njenem stanju. Da je igro mogoče analizirati po njenem izvajanju, mora te informacije tudi beležiti. Igra te zahteve izpolnjuje v treh primerih uporabe: spreminjanje nastavitve igre, igranje igre in branje izpisanih dogodkov.



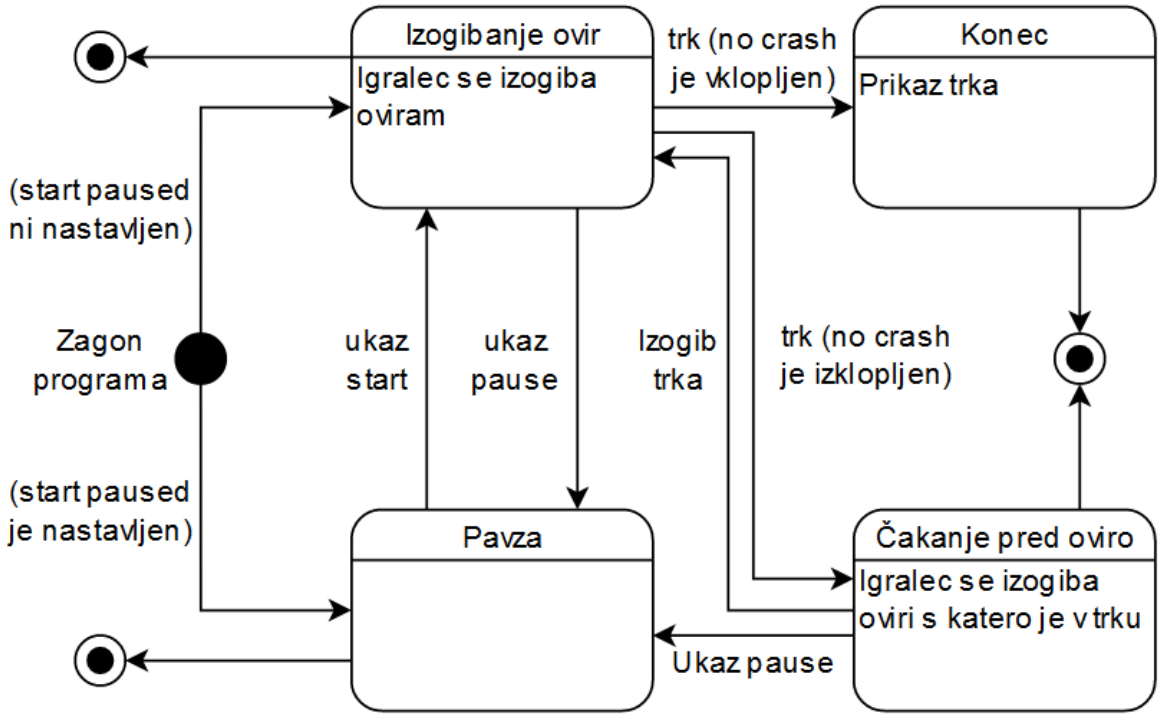
Slika 4: Primeri uporabe

Nefunkcijske zahteve
Kot je bilo že omenjeno, morajo biti na voljo nastavitve, s katerimi se lahko nastavlja težavnost in se igro prilagaja različnim paradigmam BCI. Ker se je na sisteme BCI treba privaditi, mora obstajati možnost, da se težavnost zniža na skoraj trivialno. To se lahko doseže s tem, da zmanjša hitrost ali da se omeji število vhodov, ki so potrebni za igranje igre. Med nastavitvami za prilagajanje paradigmam mora biti možnost, da na poljubna mesta na ekranu lahko postavimo animacije. Prednost bi bila, če bi bila najvišja težavnost atraktivna tudi za običajne igralce.

Ker je namen igre, da raziskovalcem olajšamo delo, mora biti komunikacijski kanal med igro in kontrolnim modulom oblikovan tako, da razvoj kontrolnega modula ni otežen brez razloga.

2.1 Igranje igre

Igranje igre se začne, ko uporabnik zažene njen program. Uporabnik je lahko nekdo, ki igro igra kot del študije, raziskovalec, ki testira nastavitve igre, razvijalec, ki testira igro za vzdrževanje, ali nekdo, ki igro igra za zabavo. Za vse te namene se v besedilu uporabljata sopomenki uporabnik in igralec. Med igranjem igra na zaslon računalnika izrisuje sliko, ki igralcu predstavlja stanje igre. Da jo je mogoče upravljati, ves čas sprejema ukaze iz kontrolnega modula (in delno iz računalnika, kjer se izvaja, kot na primer klik na gumb za zaprtje okna). Ko se zgodi kakršen koli omembe vreden dogodek (kot na primer konec igre), igra informacijo o tem dogodku pošlje kontrolnem modulu in jih hkrati na neki način lokalno zabeleži. Igra je lahko med igranjem v štirih različnih stanjih: izogibanje ovir, pavza, čakanje pred oviro in konec igre. V katerem koli stanju lahko igralec zaključi izvajanje igre, takrat pa se igranje igre konča. Po zaključku igre je uporabnik pridobil ali izpopolnil svoje kompetence za uporabo sistema BCI in igranje igre ali izboljšal razumevanje o tem, kako igra deluje, glede na namen, s katerim je igro igral. Po končani igri tudi postane na voljo datoteka *log.txt*, v katero so bile zapisane morebitne napake in omembe vredni dogodki. Ta datoteka je lahko na voljo tudi pred izvajanjem igre, če se je igra že prej izvajala. V tem primeru nova različica datoteke prepiše staro.



Slika 5: Stanja igre

Izogibanje ovir

Glavni del igre je izogibanje ovir. Če nastavitev `start paused` ni vključena, se igra začne v tem stanju. Igralec premika avtomobil levo in desno po cesti ter se s tem izogiba prihajajočim oviram. Če je igra v poravnanem načinu (je nastavitev aligned vključena), se z vsakim ukazom (levo ali desno) premakne za en pas. Sicer se premika v določeno smer, dokler igra ne prejme ukaza stop ali avtomobil ne doseže roba cestišča.

Ovire se z obzorja bližajo proti igralčevem avtomobilu z vnaprej določeno hitrostjo, pomnoženo z vrednostjo nastavitve `speed factor`. Hitrost bližanja ovir je stalna, oziroma z drugimi besedami, avtomobil se po cesti vedno vozi z enako hitrostjo. Ovire se prikazujejo na naključnih položajih na cesti, razen če je izbrano vnaprej določeno zaporedje. V tem primeru se najprej pojavijo ovire, kot so navedene v zaporedju, nato pa na naključnih položajih. Če je igra v poravnanem načinu, se ovire lahko prikazujejo le točno na sredini vsakega pasu. Cesta je razdeljena na štiri pasove, ki so v neporavnanem načinu le za okras. Časovni zamiki med pojavi naključnih ovir se čez potek igre manjšajo po vnaprej določeni formuli.

Ovire in igralčev avtomobil imajo določeno širino in dolžino, ki definirata pravokotnik. Ti pravokotniki predstavljajo, kje v prostoru se nahajajo. V kontekstu razvoja iger se za ta fenomen uporablja angleški izraz »hitbox«. Z njim lahko izračunamo, kdaj pride do trka, tako da preverimo, ali se igralčev pravokotnik prekriva s pravokotnikom katere koli ovire. V tem primeru se igra premakne v stanje konca ali stanje čakanja pred oviro, če je vključena nastavitev `no crash`. Širina igralčevega avtomobila je namenoma nastavljena tako, da je nekoliko manjša od širine slike, ki ga predstavlja. To igralcu omogoča, da se lahko oviram približa malo več, kot je mogoče videti na prvi pogled. Namen tega je, da igralca ne kaznujemo za zelo majhne napake, kar bi ga lahko od igre odvrnilo.

Če igralec pošlje ukaz za pavzo, se igra premakne v stanje pavze.

Pavza

Pavza je stanje čakanja, kjer se nič ne dogaja. Ta funkcionalnost je pomembna, ker omogoča igralcu, da od igre varno odvrne pozornost, ne da bi s tem tvegal poraz. Med pavzo se ovire igralčevemu avtomobilu ne bližajo in igralec ga ni zmožen premikati. Če je vključena nastavitev `start paused`, se igra začne v tem stanju. Ko igralec pošlje ukaz za nadaljevanje, se igra premakne v stanje izogibanja ovir.

Čakanje pred oviro

Čakanje pred oviro je podobno pavzi, le da je igralec zmožen premikati svoj avtomobil. Igra se vrne v stanje izogibanja ovir, ko se igralčev avtomobil ne prekriva več z oviro. Kot v izogibanju ovir je mogoč prehod v stanje pavze.

Konec
Ko se zgodi konec igre, se prikažeta igralčev avtomobil in ovira, v katero je trčil, na rdečem ozadju. To igralcu jasno sporoči, kako je prišlo do trka. Iz tega stanja je mogoč le izhod iz igre.

2.1.1 Komunikacija z igro

Uporabnik igro upravlja tako, da ji z uporabo kontrolnega modula pošilja ukaze. Ukazi so poslani po povezavi UDP na komunikacijska vrata, ki so nastavljena v nastavitvah. Vsak ukaz je svoj paket v obliki besedila v kodiranju UTF-8.

Tabela 1: Ukazi, ki jih igra sprejema

Ukaz	Učinek
`none`	Če se avtomobil trenutno premika, se ustavi.
`left`	Če je igra v poravnanem načinu, se avtomobil premakne en pas levo. Drugače se začne premikati levo.
`right`	Če je igra v poravnanem načinu, se avtomobil premakne en pas desno. Drugače se začne premikati desno.
`moveto-`*n*	Če je igra v poravnanem načinu, se avtomobil premakne na *n*-ti pas. Drugače se avtomobil takoj premakne na *n*-ti odstotek širine ceste.
`end`	Igra se konča in program se zaključi.
`pause`	Če igra ni v stanju konec, se premakne v stanje pavza.
`start`	Če je igra v stanju pavza, se premakne v stanje izogibanje ovir.
`hello`	Igra kontrolnemu modulu odgovori s sporočilom »`hi`« (za namen preverjanja komunikacije).

Igra kontrolnemu modulu tudi pošilja informativna sporočila o stanju igre.

Tabela 2: Sporočila, ki jih igra pošilja

Koda	Pomen
`hi`	Odgovor na kodo `hello.`
`started`	Igra se je začela.
`paused`	Igra je prešla v stanje pavze.
`atlane-`*n*	Avtomobil se nahaja na ali premika proti pasu *n*.
`spawnatlane-`*n*	Ovira se je pojavila na pasu *n*.
`spawnat-`*n*	Ovira se je pojavila na poziciji *n*. Pozicija je v tem primeru definirana kot odmik od leve strani ceste v pikslih.
`crash`	Zgodil se je trk.
`gameover`	Igra je prešla v stanje konec.
`end`	Igre je konec in program se zaključuje.

2.2 Spreminjanje nastavitev

Pred začetkom igre ima uporabnik na voljo veliko različnih nastavitev, s katerimi lahko igro oblikuje tako, da je čim primernejša njegovim potrebam. Nastavitve spreminjajo raziskovalci za nastavljanje igre in razvijalci za testiranje. Uporabnik začne spreminjati nastavitve, ko mu trenutne ne ustrezajo. Nastavitve so v datotekah *settings.ini*, *texturePaths.txt* in *gifSettings.csv*. Spreminja se jih lahko z vsakim urejevalnikom besedila, ki jih je zmožen shraniti v isti format. Vse spremenjene nastavitve je treba shraniti, da vplivajo na izvajanje igre. Ko uporabnik zaključi spreminjanje nastavitev, je igra primernejša njegovim potrebam.

V datoteki *settings.ini* so splošne nastavitve. Nastavitve so navedene v obliki:

```
ime nastavitve=vrednost nastavitve
```

Tabela 3: Učinki nastavitev

Naziv nastavitve	Učinek nastavitve
`start paused`	Če je vrednost različna od 0, se bo igra zagnala v stanju pavze.
`aligned`	Če je vrednost različna od 0, bo igra potekala v poravnanem načinu.
`start lane`	Igralčev avtomobil bo na začetku igre postavljen na tisti zaporedni pas (od leve

	proti desni), ki je naveden v vrednosti. Prvi pas ima vrednost 1.
`random seed`	Generator naključnih števil bo inicializiran z dano vrednostjo.
`no crash`	Če je vrednost različna od 0, trk igralčevega avtomobila z oviro ne bo povzročil konca igre.
`framerate`	Igra bo vsako sekundo prikazala največ toliko sličic, kot je navedeno v vrednosti.
`port`	Komunikacijska vrata, na katerih igra posluša za prihajajoče povezave, bodo nastavljena na dano vrednost.
`speed factor`	Hitrost poteka igre bo pomnožena z dano vrednostjo. To ne vpliva na število prikazanih sličic na sekundo.
`use custom car order`	Če je vrednost različna od 0, bo uporabljeno vnaprej določeno zaporedje ovir.
`custom order file`	Igra bo vnaprej določeno zaporedje ovir brala iz datoteke, ki ima isto ime kot dana vrednost.

V datoteki *texturePaths.txt* so poti do slik, ki jih igra uporablja za izris različnih objektov. Poti so navedene vsaka v svoji vrstici, v določenem vrstnem redu.

Tabela 4: Zaporedje poti slik

Mesto v zaporedju	Sorodni predmet
1	Leva prekinjena črta
2	Srednja prekinjena črta
3	Desna prekinjena črta
4	Ozadje
5	Avtomobil

V datoteki *gifSettings.csv* so navedene datoteke GIF, za katere uporabnik želi, da so med igro prikazane na zaslonu (na prikazani igralni površini) skupaj z njihovimi zaželenimi položaji. Vsaka datoteka je navedena v svoji vrstici. Z vejicami so ločene tri vrednosti v točno določenem zaporedju, in sicer pot do datoteke, horizontalna komponenta položaja in vertikalna komponenta položaja. Horizontalna komponenta določa odmik središča prikazanega GIF od leve meje igralne površine, medtem ko vertikalna komponenta določa

odmik od vrha. Odmik je merjen v pikslih, razen če za številko takoj sledi znak »%«, takrat pa je merjen v odstotkih širine oziroma višine igralne površine. Če takoj pred številko stoji velika črka »R«, je odmik merjen od položaja dna igralčevega avtomobila in je znak »%« zanemarjen. Zanemarjeni so tudi vsi presledki in vrstice, ki se začnejo z znakom »#«.
Kot je bilo že omenjeno, se v splošnih nastavitvah lahko definira še pot do dodatne datoteke, ki hrani vnaprej določeno zaporedje ovir. Te so navedene vsaka v svoji vrstici, z dvema vrednostma, ločenima z vejico. Prva navaja, kje na širini ceste se bo pojavila, druga pa časovni zamik do pojavitve naslednje ovire v milisekundah. Če je igra zagnana v poravnanem načinu, prva vrednost določa pas ceste, drugače pa odmik od leve strani ceste v odstotkih širine ceste. Vrstice, ki se začnejo z znakom »#«, so zanemarjene.

2.3 Branje dogodkov

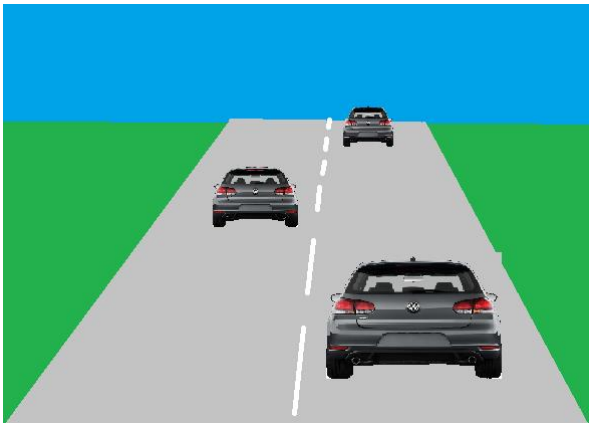
Tretja interakcija, ki jo ima lahko uporabnik z igro, je branje dogodkov, ki so bili beleženi med igranjem igre. Dogodke bere raziskovalec, ki jih želi analizirati v namene študije, ali razvijalec, ki igro testira. Branje se začne po izvajanju igre, če potek igre ni bil zadostno razločen med izvajanjem (na primer želi raziskovalec meriti reakcijske čase uporabnika, ipd.). Dogodke se lahko bere ročno ali strojno, do naslednjega izvajanja igre (ko se dogodki prepišejo z novimi). Uporabnik lahko dogodke po želji ureja, saj to ne vpliva na izvajanje igre. Ko jih uspešno prebere, so mu na voljo za analizo in priporočeno je, da si jih shrani drugam.
Dogodki so beleženi v datoteko *log.txt*. Vsak dogodek je zapisan v svojo vrstico. Najprej se zapišejo koraki inicializacije in kakršne koli napake, ki se ob tem zgodijo. Za njimi se zapisujejo dogodki igre v istem formatu kot sporočila, ki se pošiljajo kontrolnemu modulu, le da je vsakemu dodan tudi čas, kdaj se je zgodil, v milisekundah od zagona programa.
Primer:

```
SDL initialized
Window initialized
Renderer initialized
Fonts loaded
5 textures loaded
Can't open example.gif
0 gifs loaded
Sockets bound
00001109:started
00001110:spawnatlane-3
00003122:spawnatlane-2
00003367:atlane-2
00003623:spawnatlane-4
```

```
00008743:crash
00009832:end
```

2.4 Izris igre

Grafična predstavitev stanja igre se večkrat na sekundo izriše v okno, temu namenjeno, ki se ustvari ob zagonu programa. Zaradi kompatibilnosti je bila določena ločljivost okna 1280 × 720 pikslov. V tem tekstu pravimo tej predstavitvi »prikazana igralna površina«.

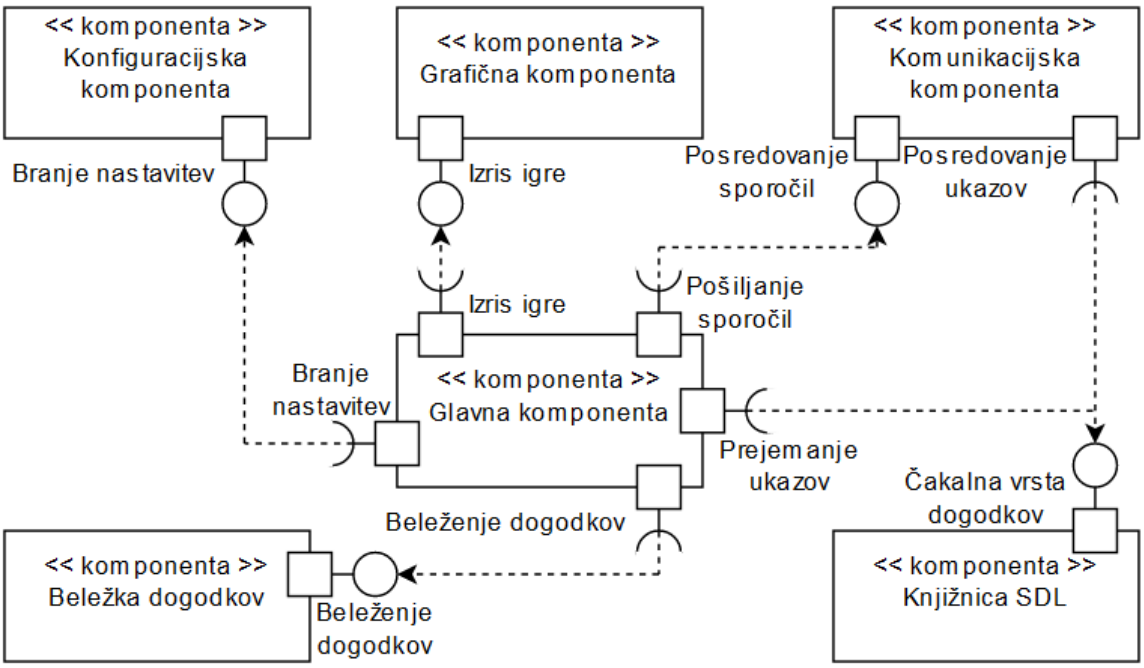


Slika 6: Predviden videz igre

Na površino se izrišejo ozadje, cesta s prekinjenimi črtami, ki označujejo pasove, ovire, igralčev avtomobil in po potrebi vse določene animacije GIF v tem vrstnem redu (vrstni red izrisa določa, kako se elementi prekrivajo). Igralčev avtomobil se vedno izriše na isti višini: 100 pikslov od spodnjega roba igralne površine. Ovire in črte na cesti se prikazujejo glede na oddaljenost od igralčevega avtomobila, premaknjene in pomanjšane na tak način, da tvorijo iluzijo tridimenzionalne perspektive. Ker se ovire pojavljajo na cesti, je logično, da bi ovire predstavljali drugi avtomobili. Zato je bila zanje določena ista slika kot za igralčev avtomobil.

3 ARHITEKTURNI NAČRT

Igra je sestavljena iz glavne in pomožnih komponent. Delitev na komponente izhaja iz delitve programske izvorne kode v različne datoteke. Notranje pomožne komponente so grafična komponenta, komunikacijska komponenta, beležka dogodkov in konfiguracijska komponenta. Zunanje pomožne komponente so knjižnice SDL2, SDL2_ttf, SDL2_image in SDL2_giflib_sa.



Slika 7: Povezave med moduli

3.1 Glavna komponenta

Glavna komponenta je odgovorna za logiko igre in klicanje pomožnih komponent. Ob zagonu inicializira vse druge komponente, inicializira podatkovne strukture za delo z logiko igre, z uporabo konfiguracijske komponente naloži nastavitve in nato začne igro. Med igro v zanki iz komunikacijske komponente sprejema ukaze s pomočjo knjižnice SDL2, glede na ukaze premika igralčev avtomobil, postavlja ovire, premika ovire, preverja za trk, odstranjuje ovire, ki niso več v igri, in z uporabo grafične komponente izrisuje igralno površino. Medtem z uporabo beležke dogodkov beleži vse pomembne dogodke. Po koncu igre v primeru izklopljene nastavitve no crash pokaže končni trk s pomočjo grafične komponente.

Izpostavljene podatkovne strukture, ki jih igra uporablja za logiko, so pozicija igralčevega avtomobila, vektor pozicij vseh ovir in čakalna vrsta vnaprej določenega zaporedja ovir.

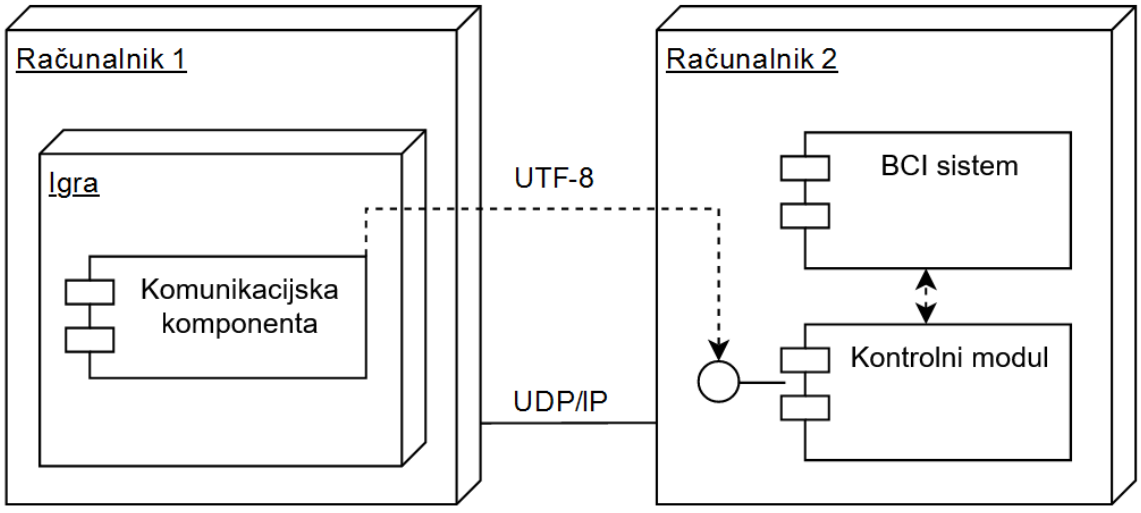
Ukaze iz komunikacijske komponente sprejema prek čakalne vrste dogodkov v knjižnici SDL2. Za pošiljanje sporočil jo kliče direktno.

Za izris igralne površine grafični komponenti sporoča hitrost igre, pozicijo igralčevega avtomobila in pozicije vseh ovir.
Vsi pomembni dogodki se istočasno beležijo z uporabo beležke dogodkov in pošiljajo kontrolnemu modulu z uporabo komunikacijske komponente.
Za izris končnega trka grafični komponenti sporoči poziciji igralčevega avtomobila in ovire, v katero je trčil.

3.2 Grafična komponenta

Grafična komponenta je odgovorna za izris igre. Izris opravlja s pomočjo knjižnic SDL2, SD2L_tff, SD2L_image in SDL2_giflib_sa. SDL2 je podlaga za druge knjižnice, ki omogočajo večino grafične funkcionalnosti, SDL2L_ttf omogoča izris teksta, SDL2L_image omogoča nalaganje dodatnih formatov slik in SDL2_giflib omogoča nalaganje in predvajanje animacij GIF. Grafična komponenta je odgovorna tudi za nalaganje in hranjenje vseh podatkov, relevantnih za izris, kot so na primer teksture in animacije GIF. Ko v njeni inicializaciji ulovi napako jo z uporabo beležke dogodkov zabeleži in nadaljuje izvajanje, če napaka ni kritična. Drugače prekine izvajanje programa. Primer nekritične napake je nepravilna navedba poti do animacije GIF. Primer kritične napake je neuspela inicializacija ene izmed zunanjih knjižnic.

3.3 Komunikacijska komponenta



Slika 8: Primer postavitve na dveh računalnikih

Komunikacijska komponenta deluje kot vmesnik med igro in kontrolnim modulom. Sporočila si z modulom izmenjuje prek protokola UDP. Ker je zmožna komunicirati tudi prek lokalnega omrežja, ni nujno, da se kontrolni modul izvaja na istem računalniku. Med inicializacijo se z pomočjo konfiguracijske komponente iz nastavitev prebere številka

komunikacijskih vrat. Kasneje se na novi niti ustvari poslušalec, ki na teh vratih čaka na sporočila. Poslušalec se mora izvajati na svoji niti, ker dejanje čakanja na sporočila blokira izvajanje niti. Ko poslušalec sprejme sporočilo, ga interpretira kot enega izmed predpisanih ukazov in s pomočjo knjižnice SDL2 vstavi v čakalno vrsto dogodkov. Ko sprejme prvo sporočilo, shrani naslov, od koder je bilo poslano. Na ta naslov se nato pošiljajo informativna sporočila o stanju igre.

3.4 Beležka dogodkov

Beležka dogodkov je odgovorna za hranjenje vseh dogodkov, ki so ji poslani iz drugih modulov. Te shranjuje v tekstovno datoteko *log.txt*. Vsak dogodek zapiše v svojo vrstico. Ko se inicializira, tekstovno datoteko odpre, in ko se izvajanje programa zaključi, jo zapre.

3.5 Konfiguracijska komponenta

Konfiguracijska komponenta je odgovorna za branje nastavitev iz argumentov programa in datoteke *settings.ini* ter nalaganje vnaprej določene vrstne rede ovir. To stori, ko to druge komponente od nje zahtevajo. Če zahtevane nastavitve ni v datoteki *settings.ini* ali če je kakšno polje v datoteki z vnaprej določenim vrstnim redom v neveljavnem formatu, to zabeleži s pomočjo beležke dogodkov. Datoteko, ki vsebuje vrstni red ovir, interpretira kot format CSV. Konfiguracijska komponenta ni odgovorna za delo z datoteko *gifSettings.csv*. To interpretira grafična komponenta, ko nalaga animacije GIF.

3.6 Kontrolni modul

Čeprav kontrolni modul ni del igre, je nujno potreben za njeno uporabo. Služi kot vmesnik med igro in sistemom BCI. Njegova funkcija je, da interpretiran namen igralca pretvori v ustrezne ukaze in jih pošlje igri. Neobvezno lahko tudi sprejema sporočila o stanju igre.

4 IZVEDBA

Za programiranje igre je bil izbran jezik C++, ker lahko v njem delamo z zelo močno knjižnico SDL2. Ta nam omogoča izris grafike, delo z okni, lovljenje sistemskih dogodkov, merjenje časa itd. Velik del kode, povezane z grafiko, je bil povzet iz vadnice avtorja Lazy Foo [5].

Program je bil ustvarjen v in za okolje Microsoft Windows. Za razširitev v okolje Linux bi bilo treba nadomestiti kodo za komunikacijo po spletu, upravljanje niti in branje datotek formata INI, ker knjižnice, ki so bile za to uporabljene, ne delujejo v okolju Linux (sicer pa obstajajo alternative).

Sprva je bil program razvijan z orodjem QT Creator, ker je ta nudil zelo obširno funkcionalnost. Ker je QT Creator plačljiv program, je bil kasneje nadomeščen z mnogo preprostejšim in brezplačnim Visual Studio Code in osnovno prevajalniško verigo (GNU make, g++) za večjo prenosljivost.

Za verzioniranje programa je bila uporabljena storitev GitHub.

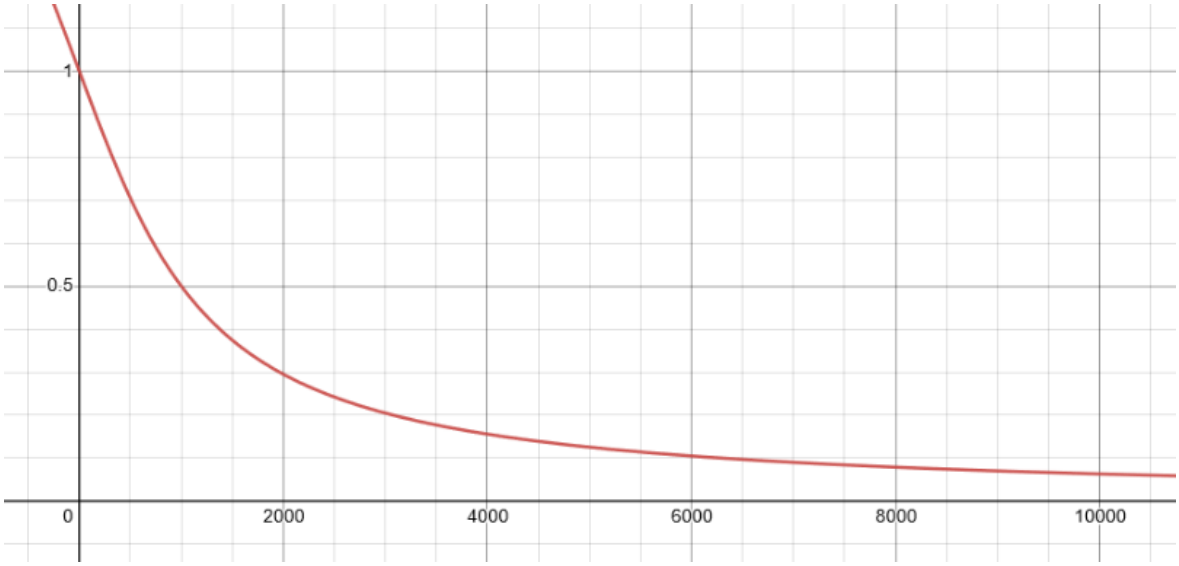
4.1 Iluzija perspektive

Za učinek iluzije je bilo treba najti matematično formulo, ki izračuna, kje in kakšne velikosti morajo biti ovire, ko so na določeni razdalji od igralca. Položaji ovir so bili hranjeni v obliki dveh koordinat. Prva je definirala, kje na širini ceste je bila ovira (od 0 do 1280), druga pa razdaljo od igralca (do 10.000 enot). Treba je bilo izračunati faktor *S*, ki je določil relativno velikost ovire in oddaljenost od sredine ceste, ter na kateri višini[1] med igralcem in obzorjem naj se jo izriše. Za faktor *S* se je uporabila kotna funkcija arctan, in sicer v obliki:

$$S = 1 - \arctan\left(\frac{y}{1000}\right) \cdot \frac{2}{\pi}$$

kjer je *y* razdalja od ovire do igralca.

[1]Opomba: Višina se v tem kontekstu meri od vrha okna navzdol, torej na vrhu okna je višina 0 in na dnu je višina 720.

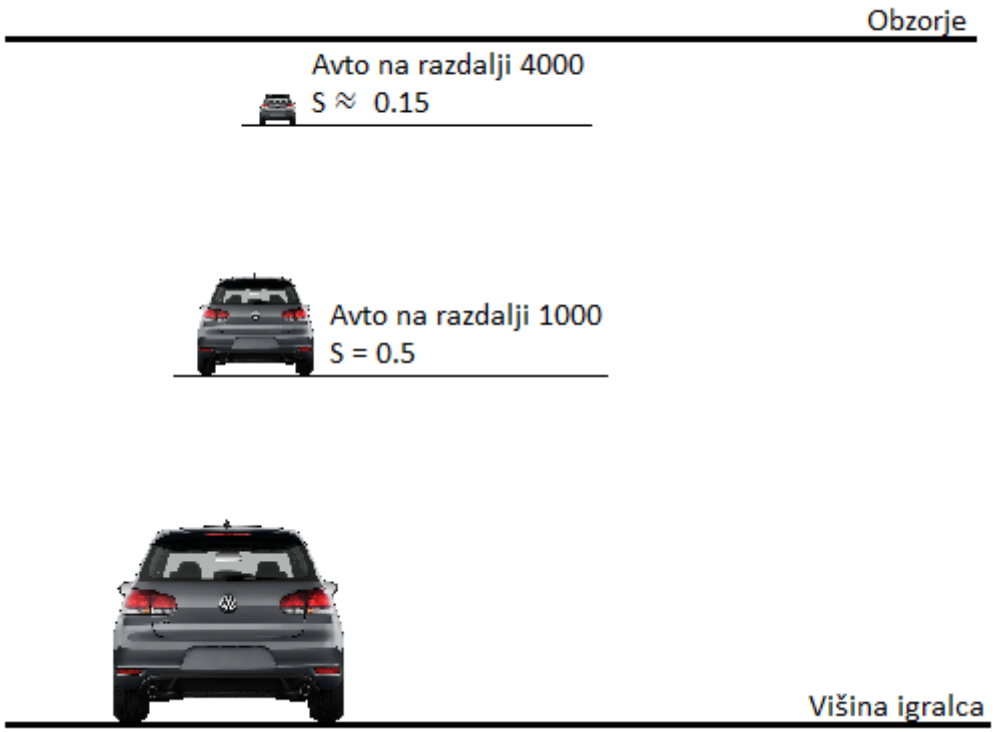


Slika 9: Graf faktorja S, ki določa, kako se ovire oddaljujejo na ekranu v odvisnosti od njihove fizične razdalje

Za izračun višine, na kateri naj se ovira izriše, je bila uporabljena formula:

$$p - S \cdot (p - h)$$

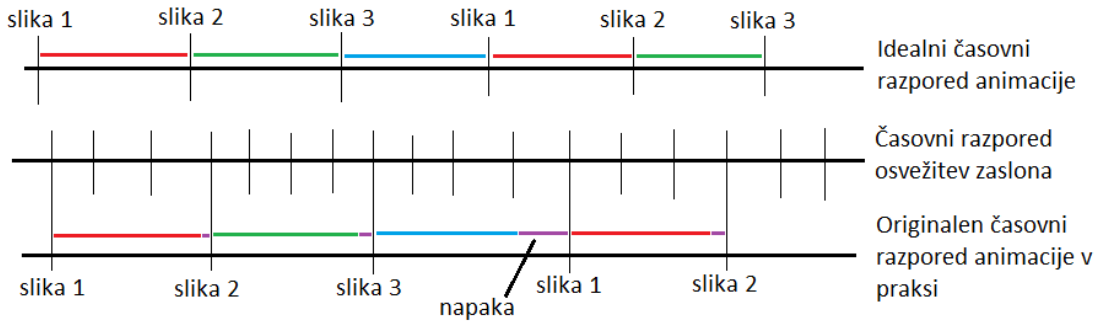
kjer sta p višina igralca in h višina obzorja.



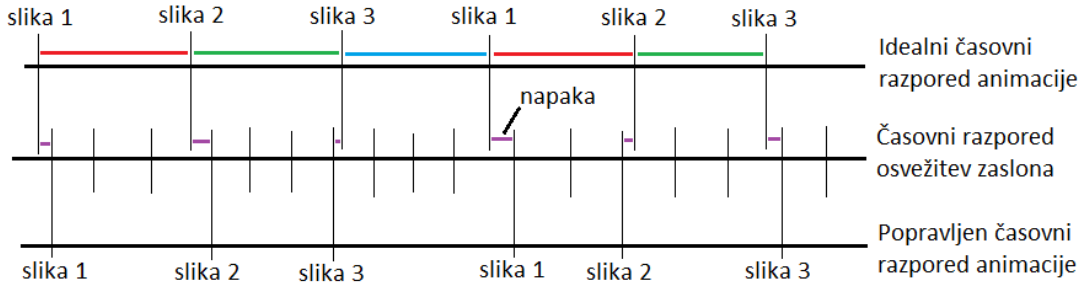
Slika 10: Primer velikosti avtomobila na različnih razdaljah

4.2 Napaka v knjižnici SDL2_giflib_sa

V knjižnici SDL2_giflib_sa je bila odkrita semantična napaka, ki je povzročala naraščajoč časovni zamik pri izrisu animacij GIF. Napaka je izhajala iz časovne razlike med tem, kdaj se je osvežila površina igre in kdaj naj bi se izrisala naslednja sličica v animaciji. Naslednja sličica se je vedno izrisala ob prvi osvežitvi po poteklem trajanju sličice, to trajanje pa se je merilo od zadnjega izrisa namesto od poteka trajanja prejšnje sličice. Ta zamik se je seštevал in sčasoma so se sličice izrisovale vedno kasneje od predvidenega časa, kar se je v praksi izrazilo kot rahlo počasnejše predvajanje animacije. Ustvarjen je bil nov »fork« knjižnice in napaka je bila odpravljena.



Slika 11: Izvirna časovnica animacije



Slika 12: Popravljena časovnica animacije

Ob testiranju se je izkazalo tudi, da so vse časovne netočnosti manjše ob večji hitrosti osveževanja, zaradi česar je bila privzeta vrednost povečana s 60 na 100 sličic (osvežitev) na sekundo.

4.3 Primer kontrolnega modula

Za lažjo predstavo in testiranje je bil ustvarjen primer kontrolnega modula na osnovi vhoda s tipkovnico. Da delo z njim ne bi bilo preveč zamudno in da bi ga bilo čim preprosteje razumeti, je bil napisan v jeziku Python. Za pisanje sta bili uporabljeni orodji IDLE in

preprost urejevalnik besedila. IDLE je uporaben za testiranje, ampak ima slab urejevalnik besedila, zaradi česar sta se uporabljala izmenično, glede na potrebe. Za vhod je bila uporabljena knjižnica keyboard. Ob testiranju modula je bila najdena napaka, in sicer zrušitev programa ob izhodu. Rešitve ni bilo moč najti, vendar to ni predstavljalo težave, ker napaka ni ovirala izvajanja programa [6].

4.4 Testiranje

Čeprav sta recenzija in testiranje ključnega pomena za uspeh programske opreme, celovito testiranje zaradi časovnih omejitev ni bilo opravljeno. Testiranje se je izvajalo vzporedno z razvojem. Po implementaciji vsake nove funkcionalnosti se je funkcionalnost testirala tako, da se je izvedla na vse predvidene načine uporabe in z najverjetnejšo predvideno napako. Na primer: ko se je uvedla funkcionalnost postavitve animacij na zaslon, se je testirala igra s pravilno nastavljeno animacijo, z nepravilno nastavljeno animacijo in brez nastavljene animacije. Testiran bi lahko bil še primer, ko je animacija v neveljavnem formatu.

5 ZAKLJUČEK

Razvoj igre je bil uspešen. Končni produkt deluje po načrtu, brez znanih napak, čeprav bi ga bilo najverjetneje mogoče podrobneje testirati. Doseženi so bili vsi cilji, razen podpore za okolje Linux. Zaradi pomanjkanja časa ni uspela nikakršna pretvorba ali prilagoditev, ki bi omogočala delovanje v okolju Linux, in igri so manjkale okrasne grafike in dopolnjena »težavnostna krivulja«, ki bi več časa zabavala vsakdanje igralce.

Igra bi lahko bila dodatno dopolnjena z možnostjo pospeševanja in zaviranja. Ob ovirah, ki se igralcu bližajo z različnimi hitrostmi, bi to lahko dodalo dodaten nivo težavnosti. Spreminjalo bi se lahko tudi število pasov, kar bi v poravnanem načinu omogočalo pojavljanje ovir v bolj zapletenih vzorcih. Implementirani bi lahko bili tudi ovinki, ki bi v neporavnanem načinu igralčev avtomobil efektivno pomikali v eno smer (proti zunanji strani ovinka).

Igre nismo uspeli testirati s konkretnim sistemom BCI. Težav z integracijo sicer ne pričakujemo zaradi nizke kompleksnosti vmesnika in ker smo igro preizkusili z alternativnim nadzornim modulom, ki omogoča enostavno upravljanje s tipkovnico. Ker se področje še vedno razvija in se skladno s tem spreminjajo tudi zahteve vmesnikov BCI, je pričakovano, da bo v prihodnosti zaželena kakšna sprememba ali nadgradnja. To najverjetneje ne bo težava, ker je zaradi modularnosti produkt mogoče enostavno vzdrževati.

Igra je izdana z odprtokodno licenco GNU General Public Licence.

Izvorna koda igre je na voljo na naslovu:
https://github.com/Pivkst/BCIRacer/
Izvorna koda dopolnjene knjižnice za prikazovanje animacij je na voljo na naslovu:
https://github.com/Pivkst/SDL2_giflib_sa

6 LITERATURA IN VIRI

[1] "Možgansko-računalniški vmesnik", Januar 2017. [Na spletu]. Na voljo: https://sl.wikipedia.org/wiki/Mo%C5%BEgansko-ra%C4%8Dunalni%C5%A1ki_vmesnik. [Dostop 4. 8. 2021].

[2] J. Lindgren, "Tutorial – Level 1 – Choosing the BCI paradigm", Julij 2018. [Na spletu]. Na voljo: http://openvibe.inria.fr/tutorial-level-1-choosing-the-bci-paradigm/ [Dostop 6. 8. 2021]

[3] "Race Arcade - Probably the BEST Classic Top Down Racer on the PS4", youtube.com, Julij 2018. [Na Spletu]. Na voljo: https://i.ytimg.com/vi/WDh_bUiBXwQ/maxresdefault.jpg, [Dostop 9. 8. 2021].

[4] J. Gordon, "How to build a racing game", Junij 2012. [Na spletu]. Na voljo: https://codeincomplete.com/articles/javascript-racer/outrun.jpg, [Dostop 9. 8. 2021]

[5] L. Foo, "Beginning Game Programming v2.0," Januar 2021. [Na spletu]. Na voljo: https://lazyfoo.net/tutorials/SDL/ [Dostop 4. 8. 2021].

[6] direprobs, M. Pieters, "Fatal Python error and `BufferedWriter`" Julij 2017. [Na spletu]. Na voljo: https://stackoverflow.com/questions/45267439/fatal-python-error-and-bufferedwriter. [Dostop 4. 8. 2021].