Izrada igre simulacije kućnog ljubimca u programskom alatu Godot

Brežnjak, Nino

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:980372

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-04-20**

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Nino Brežnjak

Izrada igre simulacije kućnog ljubimca u programskom alatu Godot

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Nino Brežnjak

Matični broj: 46203/17–R

Studij: Informacijski sustavi

Izrada igre simulacije kućnog ljubimca u programskom alatu Godot

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Radošević Danijel

Varaždin, kolovoz 2021.

Nino Brežnjak

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj završni rad, teme izrade igre simulacije kućnog ljubimca u programskom alatu Godot, će obraditi mogućnosti alata Godot u izradi različitih tipova igara s naglaskom na simulacije kućnih ljubimaca poput Tamagotchi, Digimon, mobilne igre Pou i sl., te će se izraditi odgovarajući programski primjer.

Rad će također proći povijest razvoja video igara, posebice žanra simulacije kućnog ljubimca, te iznijeti metodologiju dizajna i razvoja računalnih igara, koristeći alate otvorenog izvora, posebice Godot-a na primjeru igre simulacije kućnog ljubimca.

Svi elementi korišteni u razvoju igre će biti izrađeni u programima otvorenog izvora.

Ključne riječi: Godot; igra; simulacija; kućni ljubimac; otvoreni izvor; dizajn; implementacija;

Sadržaj

1. Uvod

Video igre su masivno rastuća globalna industrija vrijedna više od 100 bilijuna dolara, simulacije virtualnih ljubimaca je slabo predstavljen žanr u toj industriji, no kada je predstavljen, proširi se svijetom poput oluje.

Tamagotchi, jedna od prvih video igra u žanru simulacije kućnih ljubimaca, se proširio svijetom kasnih 1990-ih i 2000-ih. Danas igre tog žanra nisu toliko proširene niti popularne, no žanr je i dalje živ, u različitim oblicima.

Ovaj rad počinje opisom povijesti video igara, od pojave prvih video igara do prelaska istih od arkadnih ormara do osobnih računala i kućnih konzola, te specifično o povijesti samog žanra simulacije kućnih ljubimaca. Nakon opisa povijesti u radu su analizirane odlike simulacija kućnih ljubimaca, na način da se prvo analiziraju odlike začetnika žanra, Dogz-a, te odlike koje su dodali nasljednici žanra, da bi se došlo do same srži žanra i saznalo što sve igra u tom žanru treba sadržavati. Nakon uspješne analize je dizajnirana igra simulacije kućnog ljubimca, te su za istu izrađena sredstva, te je na kraju igra implementirana.

2. Korišteni alati

U ovom radu je korišteno četvero alata otvorenog izvora. Ti alati su:

- Godot, za razvoj igre;
- Krita, za razvoj 2D grafike;
- MuseScore, za razvoj glazbe;
- Draw.io, za dizajniranje i kreiranje žičanih modela

2.1. Godot

Stranica Godot Engine definira Godot kao potpuno besplatan, među-platformski pokretač video igara otvorenog izvora za razvoj 2D i 3D igara pod MIT licencom [1].

Ta definicija označava nekoliko ključnih svojstva Godot-a, to su:

I. Potpuno besplatan, svojstvo koje označava da nije potrebno platiti za korištenje Godot-a u bilo kojem slučaju. To se razlikuje od drugih pokretača video igara koji često zahtijevaju kupljenje komercijalne dozvole ako korisnik ima primanja viša od $20,000 godišnje koristeći te alate. Godot je potpuno besplatan za korištenje u sve svrhe.

II. Među-platformski (eng. cross-platform) označava da se Godot može koristiti za razvoj igra na više različitih platformi. Primjerice, moguće je razvijati istu igru za osobno računalo, neku konzolu poput playstation-a i android u potpunosti na Godot-u. Nema potrebe koristiti druge alate za prebacivanje igre razvijene za osobno računalo na android uređaje, Godot pruža funkcionalnost automatskog izdavanja na različite platforme, uz ručno uređivanje same igre da njezino upravljanje i izgled ima smisla na drugim uređajima.

III. Pokretač video igara (eng. game engine) definira Godot kao razvojno okruženje specijalizirano za razvoj video igara. Iako ništa korisnika ne sprječava da Godot koristi za razvoj drugih vrsta softvera, Godot dolazi sa alatima i dodacima koji ponajprije služe razvoju igara.

IV. Za razvoj 2D i 3D igara nam govori da Godot nije ograničen na određenu dimenziju, već je u potpunosti moguće razvijati bilo kakve igre u Godot-u.

V. Pod MIT licencom nam daje do znanja da je Godot dan na korištenje preko MIT licence. Open Source Initiative (bez dat.) objašnjava da MIT licenca krajnjem korisniku daje potpuna prava korištenja, kopiranja, prepravljanja, spajanja, publiciranja, distribucije, pod licenciranja, i/ili prodaje kopije softvera, uz uvjet

da je softver dan „kakav je“ (eng. „as is“), te Godot Engine ne odgovara za bilo kakvu štetu nastalu korištenjem njihovog softvera. [2]

Rad sa Godot-om je omogućen na Windowsu, Mac OS-u, Linux distribucijama i BDS-u, te je omogućeno izdavanje razvijene igre na [3]:

- platforme osobnih računala: Windows, Mac OS, Linux, BSD i UWP;
- mobilne platforme: iOS i Android;
- igrače konzole: Nintendo Switch, Playstation 4 i Xbox One koristeći pružatelja usluga treće strane (eng. third-party provider);
- web koristeći HTML5 i WebAssembly („Godot Engine“, bez dat.)

Od programskih jezika, Godot nudi izbor između C# 8.0, C++, VisualScript, koji je Godot-ov jezik za vizualno programiranje, GDScript, koji je python-liki jezik i nativan jezik Godot-a. te mnoštvo drugih jezika čiju je potporu razvila i još uvijek razvija Godot zajednica, među kojima su uključeni Rust, Nim, D i drugi [3]

2.1.1. GDscript

GDscript je dinamičko tipizirani, skriptni jezik visoke razine koji se koristi u Godot-u za izradu sadržaja. Koristi sintaksu sličnu pythonu, te teži biti optimiziran za i usko integriran sa Godot-om [4].

Identifikatori u GDscript, tj. imena varijabli, konstanti i funkcija, se sastoje od malih slova engleske abecede a-z, velikih slova engleske abecede A-Z, brojeva 0-9 i znaka „_“. Dodatno, identifikator ne smije započeti brojem.

Iznimke tome su predefinirane ključne riječi, koje se ne mogu koristiti kao identifikatori. To su riječi poput „if“, „else“, „elif“, „for“, „while“, „PI“, „TAU“ i slično.

Blokovi koda u GDscript su odvojeni uvlačenjem koda, a njihov početak je definiran znakom „:“. Zbog toga je bitno paziti na stavljanje uvlaka, ne samo zbog čitljivosti već funkcionalnosti koda. Primjer bloka koda (funkcije) u GDscript-u je:

```
func funkcija():
   var tekst1 = „Ovo je string u funkciji“
var tekst2 = „Ovo je string izvan funkcije“
```

Selekcija u GDscript-u se vrši pomoću ključnih riječi „if“, „else“, „elif“ i „match“.

If, else i elif se koriste u kombinaciji. Ako upit kod if nije istinit, provest će se upit kod prvog elif ispod if, ako ni on nije istinit, provest će se upit kod sljedećeg elif i tako nastavit do kad ne dođe do else, koji će se provest ako su svi prijašnji neistiniti. Primjer korištenja ove selekcije je:

```
if a<2:
   print(„a je manji od dva“)
elif a<4:
   print(„a je  veći ili jednak 2, ali manji od 4“)
else:
   print(„a je veći ili jednak 2 i veći ili jednak 4“)
```

Ternarni operator nije prisutan u GDscript, no njegova funkcionalnost se može dobiti korištenjem if i else. Primjer korištenja te funkcionalnosti je postavljanje druge početne vrijednosti ako je ispunjen neki uvjet:

```
var uvjetnaVarijabla = 1 if (A<5) else 2
```

U tom primjeru će uvjetnaVarijabla poprimit vrijednost 1 ako je varijabla A manja od 5, a vrijednost 2 ako je varijabla A veća ili jednaka 5.

Match je selekcija koja odgovara switch-u u drugim jezicima, uz neke promjene. Match može uspoređivati bilo kakve vrijednosti, od direktne vrijednosti, tipa podataka, enumeracije itd. Također nije potrebno poredati slučajeve u bilo kakvom redoslijedu, ili koristiti break da osiguramo da program neće skočiti u sljedeći slučaj, već je to zadano, a programer mora odrediti kada da izvršavanje koda prijeđe u sljedeći slučaj koristeći komandu „continue“. Primjer sintakse match-a je:

```
match a:
      5:
            print(„a je jednak 5“)
      „Lijepi tekst“:
            print(„a je string koji sadrži 'Lijepi tekst'“)
            continue
      3:
            print(„a je jednak 3 ili je string koji sadrži 'Lijepi tekst'“)
      1,4,6:
            print(„a je jednak 1, 4 ili 6“)
      _:
            print( „a je jednak bilo čemu, a nije definiran u prijašnjim
slučajevima“)
      var novaVarijabla:
```

```
            print(„a je jednak bilo čemu, nije definiran u prijašnjim
slučajevima, ali sad ga je moguće pozvati sa identifikatorom
novaVarijabla“)
```

Za iteraciju GDscript koristi „while“ i „for“.

While se ponaša kao i u drugim programskim jezicima, izvodi programski blok dokle god je zadani upit istinit. Primjer toga je:

```
while(true):
      print(„Ovo je beskonačna petlja, ali iz nje se može izaći koristeći
break“)
      break
```

For će izvršiti programski blok onoliko puta koliko elemenata traženi objekt ima. To može biti string, polje, rječnik, funkcija range(broj), range(početak, kraj), range(početak, kraj, korak), ili obični broj, u kojem će slučaju pretpostavit da se misli na range(broj), tj, „for a in 3“ i „for a in range(3)“ su funkcijski jednaki.

Primjer for petlje je:

```
for broj in range(2,5):
      print(broj)
```

Ovaj primjer će ispisati brojeve 2, 3, i 4. To jest, range(prvi, drugi) vrati polje brojeva od prvog, inkluzivno, do drugog, ekslkuzivno.

Funkcije u GDscript su uvijek metode, dijelovi klase, budući da su same skripte u kojima se piše kod klase. Funkcije se definiraju kao:

```
func funkcija(a, b = „zadana vrijednost“, c: String, d := „zadana
vrijednost sa zadanim tipom“):
      var a = „Nema zadanu vrijednost.“
      var b = „Ima zadanu vrijednost, no može se proslijediti bilo koja
druga vrijednost, čak i drugog tipa.“
      var c = „Nema zadanu vrijednost, no mora se proslijediti vrijednost
tipa String.“
      var d = „Ima zadanu vrijednost, te bilo koja proslijeđena vrijednost
mora biti istog tipa.“
      return a + b
```

Sve skripte napisane u GDscriptu su neimenovane klase, kojima se može pristupiti preko njihove lokacije. Za lakši pristup im je moguće dodijeliti ime koristeći ključnu riječ class_name. Budući da su skripte u GDscriptu ponajviše pisane za izvođenje direktno nad elementima u Godot-u, gotove sve nasljeđuju već postojeću klasu. Nasljeđivanje je određeno ključnom riječi extends. Primjer imenovane klase je:

```
extends Button
class_name mojGumb
var tekstGumba = „Klikni me“
```

```
func pritisnut():
      print(„Gumb je pritisnut“)
```

2.2. Krita

Krita Foundation definira Kritu kao profesionalan, besplatan alat za slikanje, otvorenog izvora. Krita je pokrivena GNU općom javnom licencom verzija 3 (eng. GNU General Public License version 3, skraćeno GNU GPLv3 ili GNU), što znači da je krajnji korisnik slobodan koristiti Kritu u bilo kakve svrhe, uključujući izradu komercijalnih radova, instalacije u školama ili poduzećima [5] [6].

Kritu je u potpunosti moguće koristiti pomoću običnog računalnog miša, no namijenjena je ponajprije za korištenje pomoću grafičkog tableta. Za izradu ovog rada softver Krita je korišten pomoću grafičkog tableta „One by Wacom – Medium“.

2.3. MuseScore

„MuseScore je softver za notaciju glazbe otvorenog izvora koji radi na Windows-u, MacOS-u i Linux-u, te je dostupan u 40 različitih jezika“ [7]

Pokriven je GNU općom javnom licencom verzija 3, poput Krite, što znači da krajnji korisnik ima sva jednaka prava korištenja MuseScore-a kao i Krite [8].

On dozvoljava stvaranje notacije za nekoliko stotina instrumenata, te reprodukciju i ispis stvorene notacije.

Ponajprije se koristi u profesionalne svrhe za stvaranje orkestralnih kompozicija ili stvaranja notacije za glazbenike, no u ovom radu će naglasak biti na njegovoj mogućnosti reprodukcije zvuka dane notacije, koji će se koristiti kao pozadinska glazba u igri

2.4. Draw.io

„Draw.io je potpuno besplatan online uređivač dijagram izgrađen oko Google Drive-a, koji omogućuje stvaranja dijagrama toka, UML-a, dijagrama relacija entiteta, mrežnih dijagrama, maketa i više.“ [9]

Draw.io je pokriven Apache 2.0 licencom, što znači da je korisnik slobodan koristiti draw.io u komercijalne svrhe, modificirati, distribuirati, koristiti kao dio patenta, i koristiti osobno. Draw.io nije odgovoran za nikakvu štetu nastalu korištenjem softvera. [10]

U ovom radu će draw.io biti korišten za dizajniranje igre prije implementacije u Godot-u. Pod tim većinski spada stvaranje žičanog modela.

3. Povijest video igara

Ovo poglavlje će objasniti povijest video igara, od njihovog nastanka do prelaska iz specijaliziranih uređaja u osobna računala i kućne konzole. Također će objasniti podjelu igri u žanrove, kako bi se postavila podlogu za odvajanje simulacija kućnih ljubimaca od ostalih igara, te objasnilo po kojim odlikama se žanrovi dijele.

Poglavlje također sadrži povijest simulacija kućnog ljubimca kroz najpopularnije i najznačajnije predstavnike žanra. Ovo će služiti kao podloga sljedećem poglavlju, koje će iz tih predstavnika izvući bazne odlike žanra, kako bi definirali minimalne uvjete da igra bude simulacija kućnog ljubimca.

3.1. Prve video igre

„Mada se video igre danas nalaze u domovima diljem svijeta, početci im se ustvari nalaze u laboratorijima znanstvenika“ [11].

Prvi prepoznati uređaj za igranje igri je predstavio dr. Edward Uhler Condon na Njujorškom Sajmu 1940. To je bio elektromehanički stroj zvan Nimatron, koji je služio za igranje drevne igre Nim [11] [12].

Na sajmu na kojem je predstavljen, Nimatron je korišten oko 100,000 puta. Funkcionirao je kao protivnik ljudskom igraču, te je svako korištenje bila jedna cijela igra Nim-a. Od tih oko 100,000 igri, Nimatron je pobijedio više od 90% [13].

Nakon Nimatron-a je bilo još nekolicina razvijenih igri, no većinski su bile razvijene u svrhe doktorata, poput A.S. Douglas-ovog OXO-a, ili za sustave u to doba nađene većinski u sveučilištima, poput Steve Russell-ovog „Spacewar!“ [11].

Mnogi „Spacewar!“, razvijen 1958., smatraju prvom pravom video igrom. „Spacewar!“ je bila vrlo jednostavna igra tenisa razvijen za PDP-1 (Programmed Data Processor-1), u to vrijeme napredno računalo. To je bila prva video igra koju je bilo moguće igrati na više računala, za razliku od prijašnjih koje su funkcionirale samo na računalu na kojem su razvijene. [11][14].

3.2. Prelazak u javno korištenje

Jedno desetljeće nakon što se pojavio „Spacewar!“, programeri u Sanders Associate Inc., vođeni Ralphom Baerom su 1967. godine razvili prototip za više programski sustav za video igre, sa podrškom za više igrača, zvani „TV Game Unit #7“, bolje poznat po nadimku Smeđa kutija (eng. The Brown Box). Smeđa kutija je bila sklopovlje vakuumskih cijevi, koja se mogla spojiti na televizor te je dozvolila dvojici igrača da igraju nekoliko različitih igri, koje su se mijenjale pomoću prekidača na prednjoj strani kutije. Kako bi se znalo koja kombinacija prekidača daje koju igru, koristile su se programske kartice, vidljivo na slici 1 [11][12][15].

Slika 1: Smeđa kutija s programskim karticama [16]

3.2.1. Magnavox i Atari

Smeđu kutiju je licencirao Magnavox, te je 1972. godine istu pustio u prodaju pod nazivom Magnavox Odyssey, par mjeseci prije Atari-a. Odyssey je originalno u Sjedinjenim Američkim Državama kod prodaje došao sa dvanaest igara, jedna od kojih je bio stolni tenis, po uzoru na kojeg je Atari kasnije stvorio arkadnu igru, Pong [11][12][17].

Odyssey nije funkcionirao kao suvremena konzola za igre. Prikazivao je samo par kockica pod kontrolom igrača, i vertikalnu crtu u sredini ekrana. Zbog tog ograničenja igre nisu imale automatsko prepoznavanje pobjede ili brojanje bodova, već je to palo na igrače. Odyssey je također došao sa ne elektronskom dodatnom opremom poput špila karti, kockama, žetonima i slično. Ta dodatna oprema je služila za kompliciranije dijelove igri, te je u isto vrijeme približila

u to vrijeme strani koncept konzole za igre kopiranjem poznatih elemenata popularnih fizičkih igara. Nažalost, unatoč tome, Odyssey se pokazao kao komercijalan neuspjeh [11][18].

Kao što je prije rečeno, 1972. godine je Atari razvio Pong. Pong je bio prva komercijalno uspješna arkadna igra, te se igrala na arkadnom kabinetu. Zahvaljujući izvrsnoj recepciji Pong-a, atari je imao sredstva za razvoj kućne verzije Pong-a, Home Pong. Home Pong je bio točno što mu ime nalaže, konzole dedicirana za igranje Pong-a u kući. Odmah nakon dolaska na tržište, Home Pong je postao hit te također postao Sears-ov do tada najprodavaniji predmet. [19]

Nakon uspjeha Pong-a, Magnavox i Sanders Associates su tužili Atari za kršenje autorskih prava. To je prisili Atari da licencira prava od Magnavoxa, koji je od istih zaradio više od sto milijuna dolara u narednih 20 godina. [11]

1977. godine je Atari na tržište stavio Atari 2600, naprednu konzolu s joystickom i igrama u više boja. To je učvrstilo mjesto konzola u kućama i pokrenulo drugu generaciju konzola za video igre. [11]

3.3. Žanrovi

Napretkom video igara je došlo do potrebe kategorizacije, te su se s vremenom javili žanrovi. Žanrovi grupiraju video igre ne po priči kao u drugim medijima, već po načinu igranja igre. Primjerice, žanr RTS (eng. Real Time Strategy) tj. strategija u stvarnom vremenu grupira igre u kojima igrač mora u stvarnom vremenu osmisliti i izvesti strateške poteze kako bi pobijedio. Žanr koji je suprotan od RTS-a, TBS (eng. Turn Based Strategy) tj. strategija bazirana na potezima grupira igre strategije u kojima igrač i protivnik izmjenjuju poteze. [20]

U osnovi, video igre možemo podijeliti u 9 glavnih žanrova koji se tada dijele na danje pod žanrove, to su [20]:

- Akcijske igre
- Akcijsko-avanturističke igre
- Avanturističke igre
- Igre igranja uloga
- Igre simulacije
- Strateške igre
- Športske igre
- Igre slagalice
- Besposlene igre

Simulacija kućnog ljubimca, tema ovog rada, je pod žanr žanra simulacija. Specifičnije pod žanr simulacija života.

3.4. Povijest simulacija kućnog ljubimca

14.12.1995. godine, PF.Magic je na tržite stavio računalnu igru Dogz: Your Computer Pet. Dogz, kao što i samo ime insinuira, je bila igra u kojoj se igrač igra sa virtualnim psima. To je bila prva interaktivna simulacija kućnog ljubimca, i pokazala da je žanr iznimno popularan. Nedugo nakon uspjeha Dogz-a, PF.Magic je objavio sljedeću igru u serijalu, Catz, igra u kojoj se igrač igra s mačkama. Nakon popularnosti oba naslova, dodana je funkcionalnost u Dogz 2 i Catz 2 koja, kada su oba naslova instalirana na isto računalo, je spojila obje igre u jednu igru, Petz. Tako je nastao Petz serijal, začetnik simulacije kućnog ljubimca. [21][22]

3.4.1.Tamagotchi

1996. godine, kada osobna računala još nisu imala mjesto u svakom domu, i kad su pametni mobiteli bili udaljeni nekoliko desetljeća, Akihiro Yokoi, predsjednik WiZ-a u Japanu, i Aki Maita, član prodaje i marketinga u Bandai-u, tvrtke odgovorne za popularan serijal Mighty Morphin Power Rangers, su imali revolucionarnu ideju. Donijeti dijeci diljem svijeta ljubimca kojeg mogu nosit sa sobom, gdje god išli. Rezultat te ideje, je bio Tamagotchi. [22]

U doba kad su razvili tu ideju, u Japanu je već postojalo dosta simulacija kućnog ljubimca u prenosivom formatu, no nedostajalo im je nešto kritično. Sve igre s ljubimca dotad, uključujući i Petz, su fokusirane samo na igru i zabavu s ljubimcima. Yokoi je smatrao da je to krivo, jer je znao koliko posla ljubimci mogu biti, i kolika su ustvari odgovornost. To je dovelo do velike razlike u Tamagotchi-u od ostalih simulacija kućnog ljubimca, naime, ako se korisnik ne brine dovoljno o svom ljubimcu, on će umrijet. [22]

Ova promjena u načinu igranja je potpuno promijenila pristup ljubimcu. Više nije bio nešto zabavno što igrač pogleda kad mu je dosadno, već je bio obaveza kojeg je trebalo hraniti kroz dan, paziti da je čist, i staviti spavat po noći.

Budući da je ta potreba za pažnjom značila da Tamagotchi mora biti nešto što igrač zaista može nositi bilo gdje, dizajn je promijenjen u jajolik oblik sa tri gumba (slika 2). Taj jajolik oblik je također i izvor imena Tamagotchi, Tamago je japanska riječ za jaje, a -tchi dolazi iz japanske riječi za ručni sat, uotchi. [22]

Slika 2: Tamagotchi [23]

Tamagotchi je bio izuzetno popularan čim je izašao na tržište u Japanu. U manje od godine dana je postigao deset milijuna prodanih komada, no, Bandai je znao da tu nije kraj. U svibnju 1997. godine Tamagotchi je predstavljen američkom tržištu. Do sredine siječnja je prodano više od tri i pol milijuna komada Tamagotchi-a. (Rossen J., 5.7.2021.)

3.4.2. Digimon

U tijeku izuzetno popularnog Tamagotchi-a, Bandai je odlučio izdati novu verziju za drugo tržište. Naime, Tamagotchi je bio namijenjen za djevojčice u osnovnoj školi, mada je postao popularan kod osoba svake dobi, pa je Bandai odlučio stvoriti novu igru inspiriranu Tamagotchi-em, no koja bi bila privlačnija dječacima u osnovnoj školi. Tako je stvoren takozvani V-Pets uređaj (slika 3) i Digimon serijal [22][24].

Slika 3: Digimon V-Pets uređaji [25]

Digimon je od svog osnutka izrastao u golemi serijal, sa nekoliko crtanih serija, filmova, stripova, igrački i računalnih igri. [24]

3.4.3. Chao Garden

„Chao Garden je sasvim moguće najkompleksniji simulator odgoja djece ikad napravljen“ [26]

Chao Garden je serijal takozvanih mini igra, igra u igri, u kojoj igrač odgaja stvorenja zvana „Chao“ (slika 4) koristeći likove i materijale iz glavne igre u kojoj se nalazi.

Slika 4: Slika ekrana Chao Garden-a u igri Sonic Adventure DX: Director's Cut

Prva igra koja je sadržavala Chao Garden je bio Sonic Adventure, koji je izašao 23.12.1998. u Japanu za Sega Dreamcast, te 1999. godine na međunarodno tržište. [27]

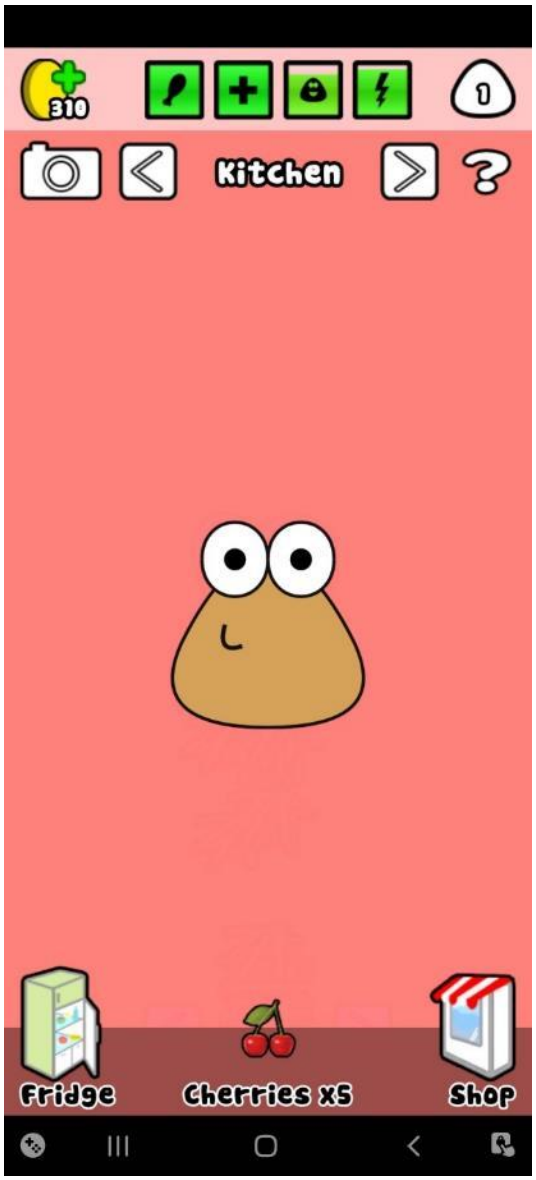
Sonic Adventure je prva igra u istoimenom serijalu, koji se sastoji od Sonic Adventure, Sonic Adventure 2, izašao 23.6.2021. za Sega Dreamcast , te ažuriranih verzija istih prenesenih na različite konzole. [28]

Kako su prolazile godine i Sonic Adventure serijal postajao sve ažuriraniji, tako je i Chao Garden dobio mala ažuriranja koja su ga dovela do trenutnog stanja.

Chao Garden se također pojavio u drugim serijalima, kao Tiny Chao Garden. Tiny Chao Garden je jednostavnija igra od Chao Gardena, budući da je Tiny Chao Garden izašao na ručnoj konzoli Game Boy Advance, no poseban je po tome što sadrži nekoliko Chao-a ekskluzivnih boja, te dozvoljava spajanje sa Nintendo Gamecube verzijama igri Sonic Adventure i Sonic Adventure 2, što omogućava prijenos tih ekskluzivnih Chao-u u njih, ili prijenos drugih Chao-a između Adventure igara. [29]

3.4.4. Pou

Pou, pušten u prodaju 5.8.2012., je mobilna igra simulacije kućnog ljubimca za Android i iOS, u kojoj igrač odgaja Pou-a (slika 5), malo trokutasto stvorenje oblih rubova bez ruku i nogu. Pou-ov izgled se može u potpunosti prilagoditi, te je moguće kupiti odjeću za njega ili nju koristeći novčiće koje igrač zasluži brinući se o Pou-u ili igrajući mini igre. Sa više od petsto milijuna preuzimanja na Trgovini Play, Pou je jedna od najpopularnijih simulacija kućnog ljubimca ikad. [30] [31]



Slika 5: Slika ekrana Pou, snimljeno na Samsung Galaxy A51

4. Odlike simulacije kućnog ljubimca

Kako bi razvili kvalitetnu igru simulacije kućnog ljubimca, potrebno je prvo znati što uopće klasificira igru kao takvu. U tu svrhu će u ovom poglavlju biti analizirane igre simulacije kućnog ljubimca koje su spomenute u prijašnjem poglavlju, te iz toga izvučene osnovne karakteristika žanra.

4.1. Karakteristike začetnika žanra

Budući da začetnik žanra, Dogz, više nije u prodaji, za analizu igranja te igre će se koristiti video igranja Youtube korisnika „68k Mentat“ [32]

Igra započinje opcionalnim kvizom, koji igraču da preporuku koji od mogućih pet pasa bi njemu najviše odgovarao. Igrač nije primoran odabrati preporučenog psa, već je slobodan iskušati njih svih pet, i odabrati kojeg god želi. Svih pet mogućih psa imaju drugačiji izgled i karakteristike.

Nakon što igrač odabere željenog psa, mora mu dati ime, te se on pojavi u zasebnog prozoru gdje se igrač s njim može igrati, maziti ga, hraniti ga, prati, itd. Klikom na posebnu ikonu taj prozor nestaje, a pas prelazi u hodanje po cijelom zaslonu. Igrač psa također može obojati u različite boje.

Iz ovog video možemo uočiti nekoliko karakteristika, najbitnija od kojih je fokus na ljubimca, u ovom slučaju psa. Naime, sve radnje koje korisnik radi direktno ili indirektno utječu na ljubimca, koji je u centru igre.

Druga karakteristika koja iskače je količina prilagodljivosti ljubimca. Igrač ima na izbor 5 pasmina, koje naknadno može obojati u mnoštvo boja. Ovo pruža igraču mogućnost da ljubimca učini potpuno svojim, što potiče bolju povezanost sa virtualnim ljubimcem.

Nažalost, ovdje mogućnosti prestanu. Dok igra pruža dovoljno zabave sa ljubimcem s kojim se stalno možeš igrati i kojeg možeš prilagoditi svojim željama, igra ne pruža nikakav poticaj za igranjem. Ljubimac neće nestat ako se igrač ne brine o njemu, ne igra s njim, ili potpuno zaboravi na njega.

4.2. Dodaci nasljednika

Kao što je prije spomenuto u radu, Tamagotchi je dodao veliki poticaj igraču da se marljivo brine o svom ljubimcu.

Naime, ako se igrač dobro brinuo o svom ljubimcu i redovito se s njim igrao, on bi izrastao iz loše definirane mrlje u slatko stvorenje. No, ako se igrač nije brinuo o ljubimcu, on bi izrastao u virtualnog delikventa, ili još gore, bi jednostavno umro. [22]

Dakle, Tamagotchi je dodao poticaj za konstantno igranje s ljubimcem, i veći osjećaj povezanosti s jednostavnim dodatkom okidača kraja igre, te promjenom slobodnog prilagođavanja ljubimčevog izgleda u izgled ovisan o kvaliteti brige igrača.

Digimon je tu formulu proširio s dodatkom borbe između ljubimca vlasnika i drugih igrača s kojima vlasnik poveže uređaj. To je dalo poticaj za kvalitetnu brigu ne samo kako bi ljubimac bolje izgledao i dulje živio, već i kako bi se vlasnik mogao natjecati s prijateljima. [22]

Chao Garden je imao mnoštvo dodataka, no najinteresantniji dodatak je vjerojatno reinkarnacija i „Chaos Chao“. Naime, ako je Chao dovoljno sretan kada umre, što se desi poslije oko 3 stvarna sata koje igrač provede s njim nakon rođenja, Chao će se reinkarnirati u jaje, ali će zadržati ime i 10% statistika. Ovo je dodatni poticaj da se igrač dobro brine o ljubimcu, jer ako se brine dovoljno dobro, njegova smrt nije nužno kraj. [33]

Chaos Chao je posebna vrsta Chao-a, u koju se može pretvoriti bilo koji Chao ako je ispunio nekoliko uvjeta. Chaos Chao su posebni jer oni ne stare, nikad neće umrijeti niti se reinkarnirati. Ako igrač uspije pretvoriti svog Chao-a u Chaos Chao, može biti siguran u činjenicu da će biti s njim dokle god ima pristup igri. [34]

Dakle, Chao Garden je proširio element smrti u nešto što se može privremeno izbjeći s dobrim odgajanjem, te trajno izbjeći uz puno truda. Time daje još više poticaja igraču da bude čim bolji u igri.

Pou nije puno inovirao u žanru, ali je vrlo dobar primjer bitnosti platforme na kojoj se igra nalazi. Naime, Pou je mobilna igra, te se kao takva igra na pametnom uređaju, kojeg velika većina osoba danas nosi sa posvuda sa sobom. To je velik korak od Tamagotchi-a, koji je imao poseban uređaj koji je igrač morao nositi sa sobom, i pogotovo velik korak od Dogz i Chao Garden-a, koji se nalaze na kućnim konzolama i osobnim računalima, koje korisnik nikako ne može ponijeti sa sobom.

4.3. Definiranje baznih karakteristika žanra

Uzeći karakteristike začetnika i dodatke nasljednika u obzir, možemo zaključiti koje karakteristike su najbitnije za izradu dobre igre simulacije kućnog ljubimca.

Po mom mišljenju, najvažnija karakteristika, mada ju začetnik žanra nema, je smrt ljubimca.

Ako ljubimac može nestati ako igrač nije dovoljno pažljiv, to daje velik poticaj igraču da bude čim bolji, te daje smisao hranjenju i čišćenju ljubimca, te brige o njegovom rasporedu spavanja. To također daje razlog da se korisnik vraća u igru, umjesto da samo odigra 5 minuta kad mu je dosadno i onda potpuno zaboravi na nju.

Sljedeća karakteristika, koja se javlja u svim predstavnicima žanra, je prilagodljivost ljubimca. Ako je ljubimac kojeg igrač ima jedinstven u njegovom društvu, lakše će se vezati za njega nego ako svi u društvu imaju jednakog ljubimca s istim imenom.

Posljednja karakteristika kojoj bih dao pažnje kao ključna u izradi kvalitetne igre simulacije kućnog ljubimca je upravo društveni faktor. Ako igrač može uspoređivati svog ljubimca s ljubimcima svog društva, ili bolje još povezati svog ljubimca s drugima direktno, igra će se lakše stopiti u igračev život, te će igrači sami tražiti priliku da uvedu svoje prijatelje u nju.

Uzevši to se u obzir, najbitnije karakteristike igra simulacija kućnog ljubimca su:

- Mogućnost smrti ljubimca
- Prilagodljivost ljubimca
- Društveni faktor ljubimca

Naravno, nedostatak bilo koje od tih ne znači da igra ne pripada žanru, već znači da vjerojatno neće biti uspješna koliko ostale igre u žanru.

4.4. Primarna petlja, sekundarne, i tercijarne petlje

Kako bi si olakšali posao u dizajniranju igre, korisno je odrediti petlje igranja.

Stephen Duetzmann iz Engaged Family Gaming petlju igranja definira kao „…pojam dizajna igre koji se koristi za opisivanje ponavljajućih aktivnosti koje će igrač poduzimati tijekom igranja igre.“ Laički rečeno, petlje igranja su aktivnosti čije ponavljanje nam dozvoljava napredak u igri, ili čije ponavljanje nas zadržava u igri. [35]

Kad pričamo o simulacijama kućnog ljubimca, primarna petlja igranja, ona koju ponavljamo kao glavnu aktivnost igranja igre, je briga o kućnom ljubimcu. Ona se sastoji od hranjenja ljubimca, kupanja ljubimca te igranja s ljubimcem kako bi mu potrebe bile ispunjene.

Sekundarna petlja, čija je svrha zadržati igrače, bi u simulaciji kućnog ljubimca bila odrastanje ljubimca od jajeta do smrti. Igrač počinje sa jajetom, brine se o ljubimcu do kad ne odraste, te kada ljubimac umre dobije novo jaje. Dužina ljubimčevog života i rijetkost jajeta kojeg dobije kad on umre mogu ovisiti o kvalitete brige iz primarne petlje.

Tercijarnu petlju, čija je svrha osigurati dugotrajnost igre, je najlakše osigurati korištenjem postignuća. Ako igrač dobije postignuće za 5 savršeno odgojenih ljubimaca, ima poticaj da radi upravo to, umjesto da prestane igrati kad izgubi prvog ljubimca.

5. Razvoj simulacije kućnog ljubimca

Sa postavljenom teorijskom podlogom, možemo početi sa izradom video igre.

Prvi korak će biti odrediti koje mogućnosti će naša igrati imati, kako će biti prezentirana igraču, te kako ćemo implementirati određene petlje igranja. Kraće rečeno, prvi korak je dizajniranje same igre.

5.1. Dizajniranje igre

Počevši sa ljubimčevim potrebama, ljubimac će imati tri statistike koje prate njegove potrebe. To su:

- Sreća, koja se prazni brže čim je niža, koristeći formulu $sreća = trenutna\ sreća - \left(5 * \frac{maksimalna\ sreća}{trenutna\ sreća}\right)$ svaki sat
- Hrana, koje se potpuno isprazni kroz 24 sata, gubeći 3.75% po satu ako je ljubimac budan, i 5% po satu ako ljubimac spava
- Energija, koja se prazni 6.25% po satu, ako je ljubimac budan

Igrač može raditi 4 akcije kako bi djelovao na ljubimca:

- Klikom na gumb s ikonom glazbene note, počinje svirati kratka glazba koja poveća ljubimčevu sreću za 20%.
- Klikom na gumb s ikonom hrane, ljubimac dobije keks koji pojede, te mu se napuni Hrana za 30%.
- Klikom na ikonu žarulje, gase se svijetla i ljubimac počinje spavati. Prilikom spavanja ljubimac dobiva 12.5% energije po satu.

Kako bi igrač imao poticaja za igranje, niske statistike vode do sljedećih loših događaja:

- Ako je sreća na 0% 72 sata, ljubimac umire.
- Ako je hrana na 0% 48 sati, ljubimac umire.
- Ako je energija na 0% 24 sati, ljubimac umire.

Sve ove mogućnosti će biti prikazane u jednom prikazu. Žičani model tog prikaza je vidljiv u slici 6.

Slika 6: Žičani prikaz glavnog ekrana igre izrađen u Draw.io

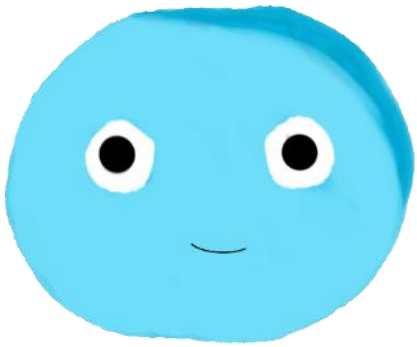
Kako bi igrač mogao postaviti ljubimčevo ime, postojat će još jedan prikaz, koji se pojavi kada igrač prvi puta pokrene igru, te kada igračev ljubimac umre. On je vidljiv na slici 7.

Slika 7: Žičani prikaz imenovanja ljubimca izrađen u Draw.io

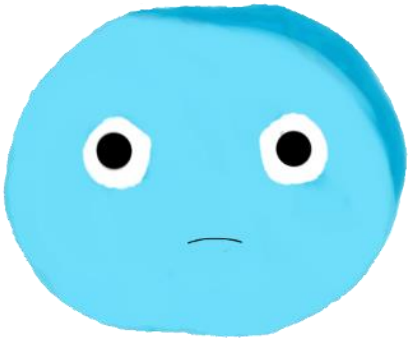
5.2. Razvoj grafike

Grafika razvijena za ovaj završni rad je:

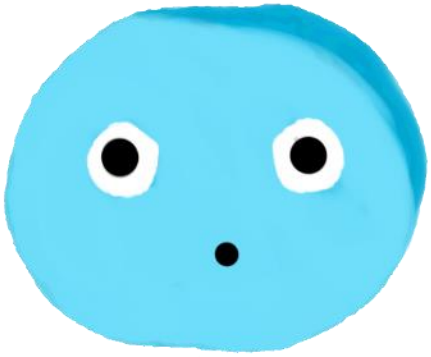
- Ljubimac, u četiri stanja: sretan (slika 8), nesretan (slika 9), s otvorenim ustima za jedenje (slika 10) i u stanju spavanja (slika 11)
- Keks, koji će ljubimac jesti sa tri razine pojedenosti te za korištenje u statusu ljubimca (slika 12, 13, 14)
- Nota, za gumb za pokretanje muzike (slika 15)
- Žarulja, za gumb za gašenje svijetla i stanje ljubimčeve energije (slika 16)



Slika 8: Sretan ljubimac, izrađeno u Krita (autorski rad)



Slika 9: Nesretan ljubimac, izrađeno u Krita (autorski rad)



Slika 10: Ljubimac s otvorenim ustima, izrađeno u Krita (autorski rad)

Slika 11: Ljubimac koji spava, izrađeno u Krita (autorski rad)

Slika 12: Keks, izrađeno u Krita (autorski rad)

Slika 13: Djeloično pojedeni keks, izrađeno u Krita (autorski rad)

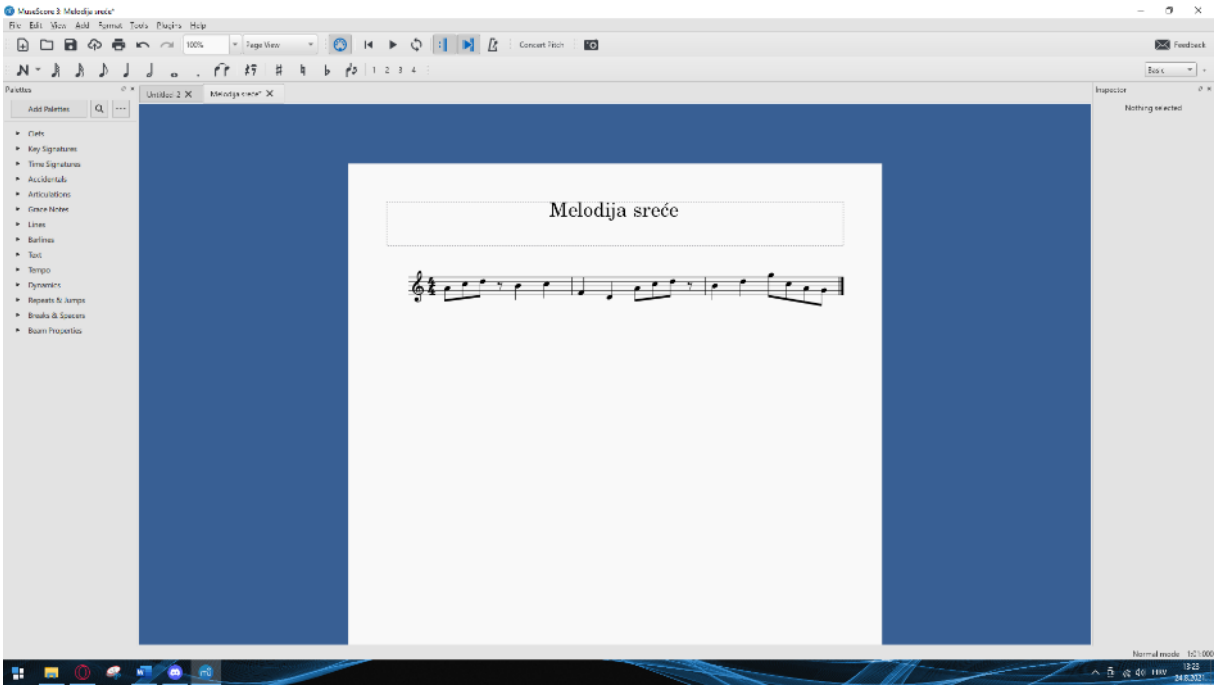
Slika 14: Većinski pojeden keks, izrađeno u Krita (autorski rad)

Slika 15: Nota, izrađeno u Krita (autorski rad)

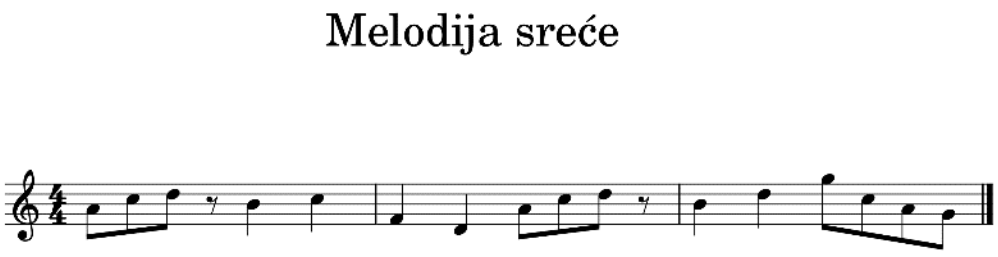
Slika 16: Žarulja, izrađeno u Krita (autorski rad)

5.3. Razvoj glazbe

Za ovaj rad je razvijena jedna melodija, koju ljubimac sluša kako bi si obnovio sreću. Melodija traje otprilike pet sekundi, napisana je za ksilofon u MuseScore-u, te preko istog izvezena u gog formatu. Slika ekrana rada u MuseScore-u je vidljiva na slici 17, a notacija pjesme u slici 18.



Slika 17: Slika ekrana rada u MuseScore



Slika 18: Notacija melodije, izrađeno u MuseScore (autorski rad)

5.4. Implementacija igre

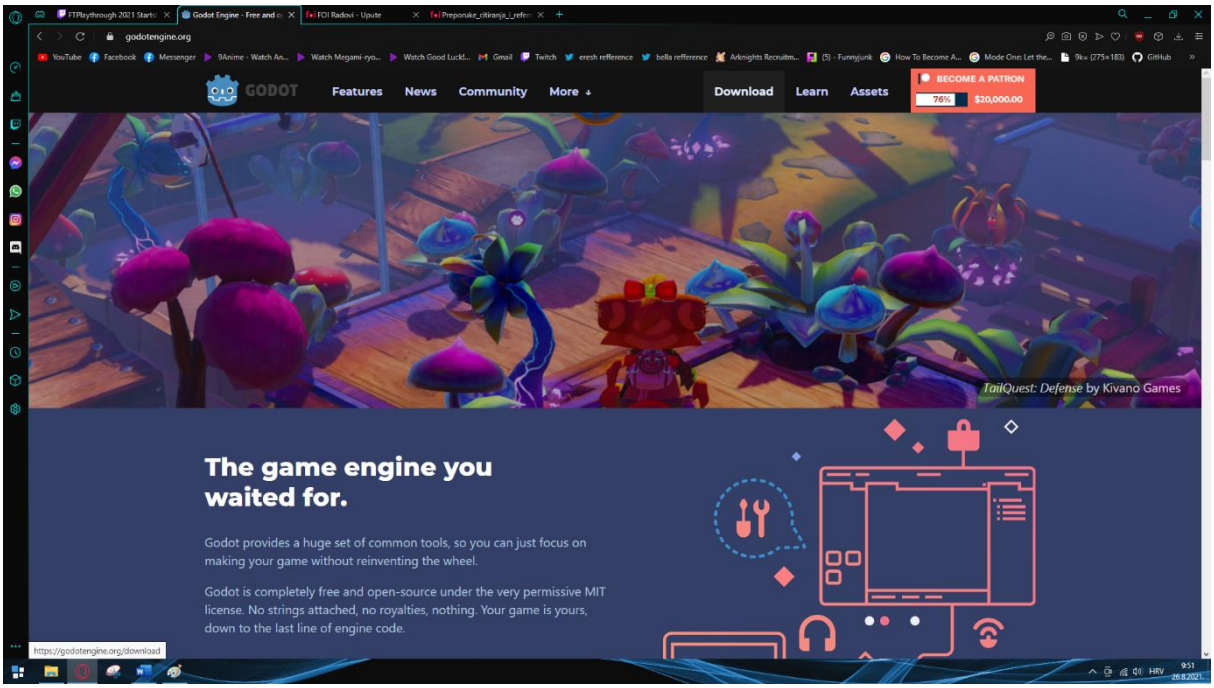
Kao što je ranije u radu spomenuto, igra će biti realizirana pomoću pokretača video igara Godot. No prije njegovog korištenja, potrebno ga je preuzeti i instalirati.

5.4.1. Instalacija Godot-a

Instalacija Godot-a je vrlo jednostavna, te zahtijeva samo 2 koraka:

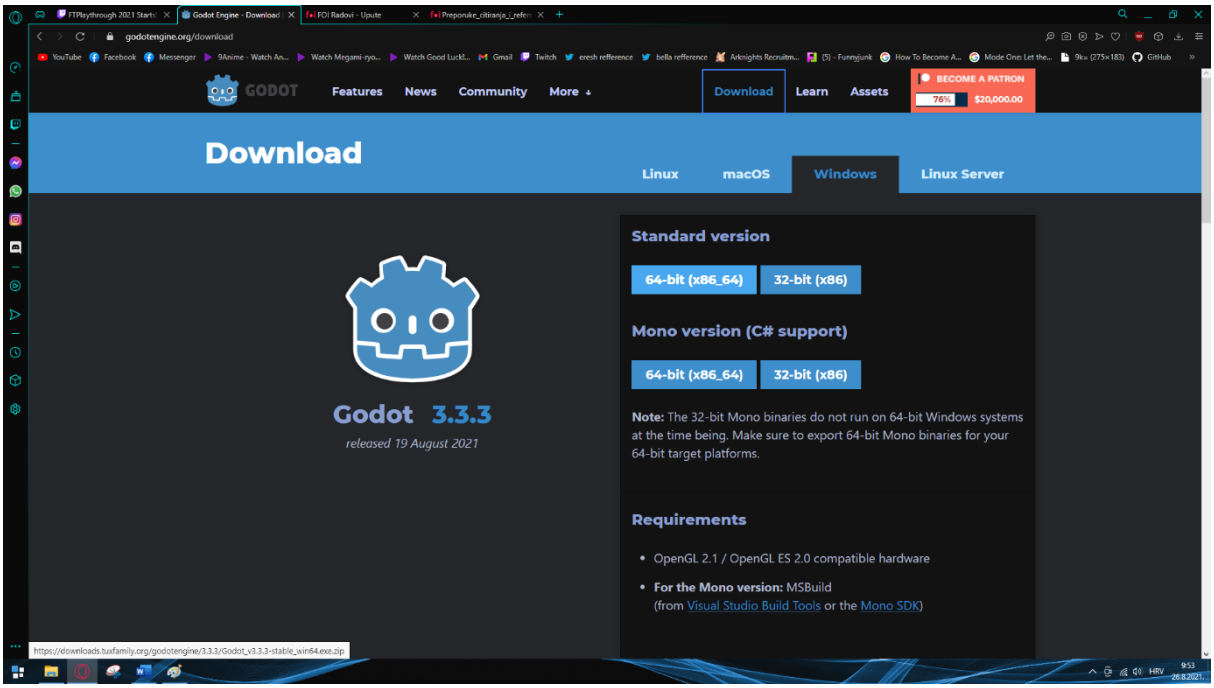
1. Preuzimanje Godot-a sa GodotEngine stranice
2. Raspakiranje preuzete zip datoteke

Za preuzimanje Godot-a, potrebno je posjetiti stranicu godotengine.org, te na gornjoj navigacijskoj traci kliknuti gumb „Download“ (eng. download = preuzmi). Slika zaslona GodotEngine stranice je vidljiva na slici 19.



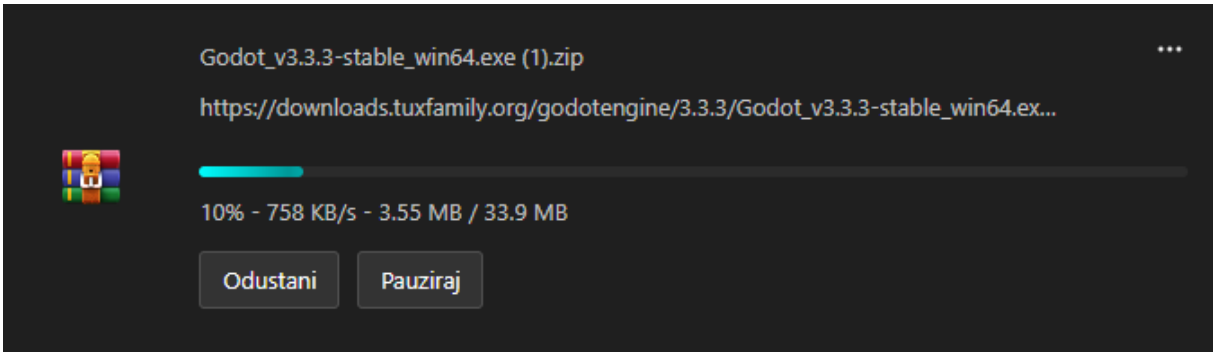
Slika 19: Izgled početne stranice GodotEngine (godotengine.org)

Nakon klika na gumb „Download“ otvara se stranica za preuzimanje. Ovdje korisnik odabire koju verziju Godot-a želi preuzeti, za svrhe ovog završnog rada korištena je verzija 3.3.3 Standard Edition 64+bit. Slika stranice za preuzimanje vidljiva je na slici 20.



Slika 20: Izgled stranice za preuzimanje Godot-a (godotengine.org/download)

Klikom na željenu verziju počet će preuzimanje zip datoteke (slika 12).

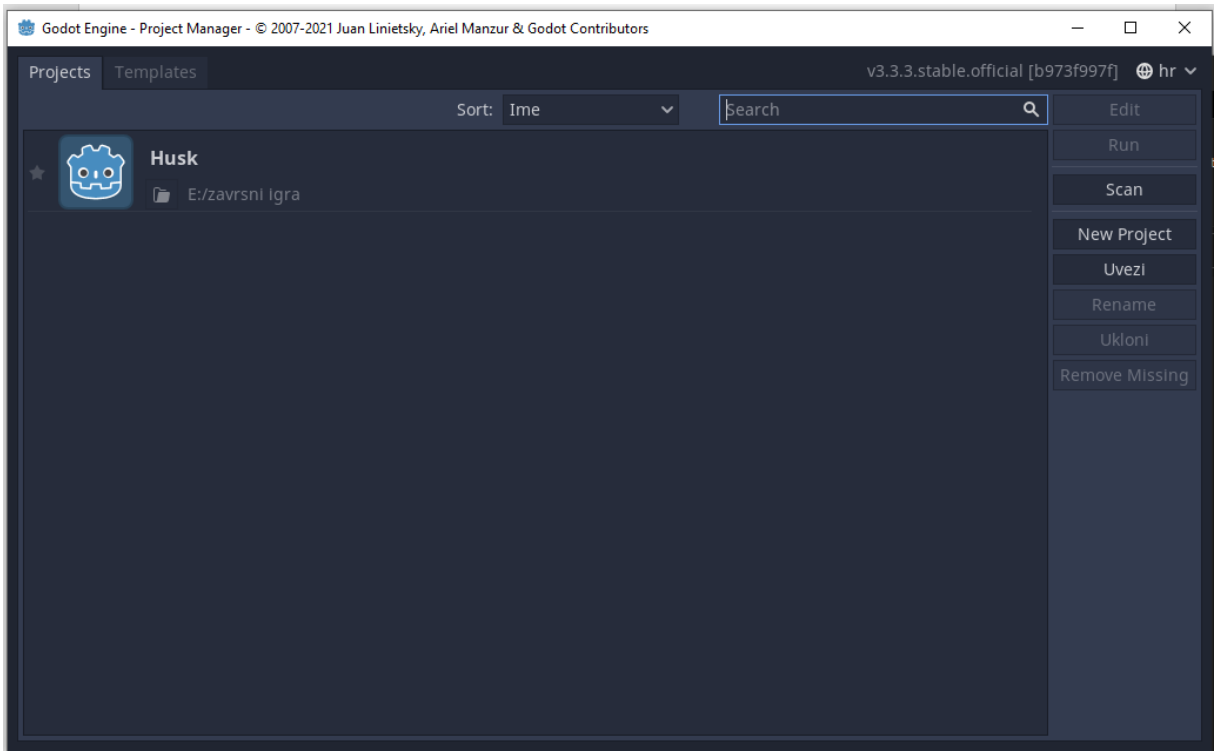


Slika 21: Preuzimanje zip datoteke Godot-a

Kada preuzimanje završi, potrebno je raspakirati .zip, te samo otvoriti danu .exe datoteku (slika 22).

Slika 22: Godot.exe

Otvaranjem dane .exe datoteke se otvara Godot (slika 23). Nije potrebna nikakva daljnja instalacija.

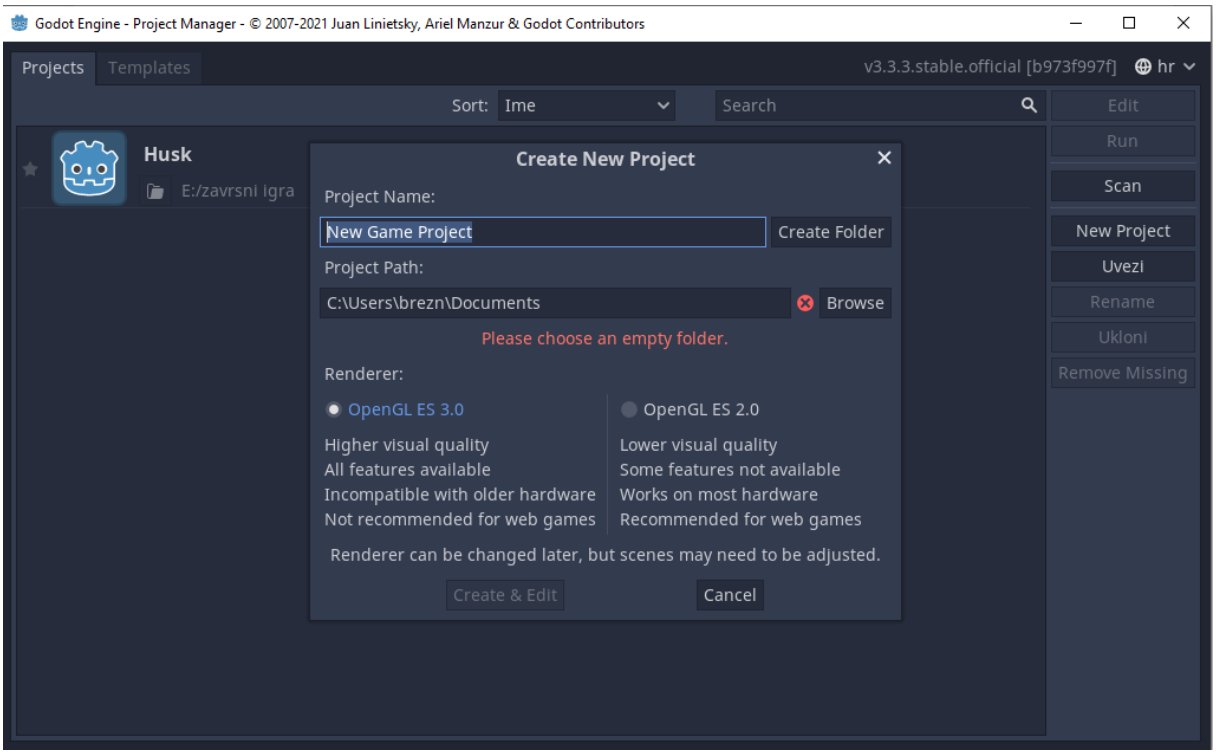


Slika 23: Slika zaslona alata Godot

5.4.2.Kreiranje projekta

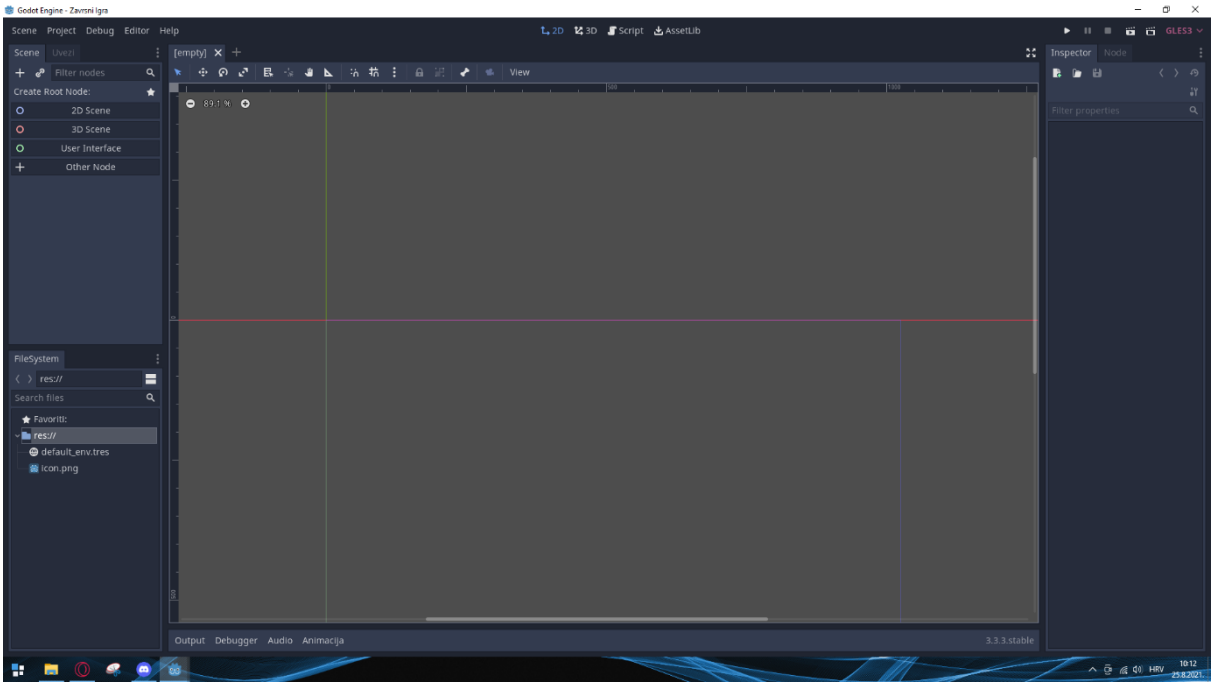
Kada je Godot preuzet i otvoren, potrebno je stvoriti projekt igre. Projekt će odrediti mjesto gdje će se spremati svi resursi vezani uz igru, te sam naziv igre (može se i naknadno promijeniti).

Za stvaranje projekta potrebno je kliknuti na gumb „New Project“. Pritiskom na taj gumb se otvara novi prozor „Create New Project“ (slika 24) u kojem možemo odrediti lokaciju projekta (mora biti praza mapa), naziv projekta te grafički renderer. Za ovaj rad je korišten OpenGL ES 3.0.



Slika 24: prozor „Create New Project“

Nakon odabira svih potrebnih opcija, klikom na „Create & Edit“ se otvara novi prazni projekt, izgled istog je vidljiv na slici 25.



Slika 25: Prazni projekt u alatu Godot

5.4.3.Čvorovi i scene

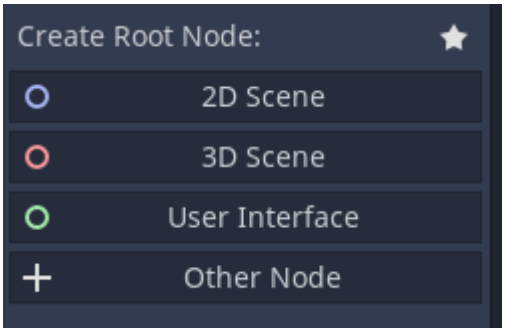
Sa otvorenim praznim projektom, moguće je krenuti radit na igri. No, prije toga, potrebno je razumjeti par pojmova, specifično pojmove čvora, scene i skripte.

Čvor je osnovna građevna jedinica u Godot-u. Neki čvorovi služe za prikaz slika, neki za sviranje zvučnih efekata, neki za registriranje sudara i slično, ali svaki čvor ima nekoliko osnovnih svojstava [36]:

- Ima ime
- Ima svojstva koja se mogu urediti
- Može primiti povratni poziv za obradu svaku sličicu
- Može biti proširen
- Može biti dodan drugom čvoru kao njegovo dijete, time stvaraju stablo

Scena je hijerarhična grupa čvorova, koja uvijek ima točno jedan korijenski čvor, može viti spremljena na disk te se može instancirati kao čvor druge scene.

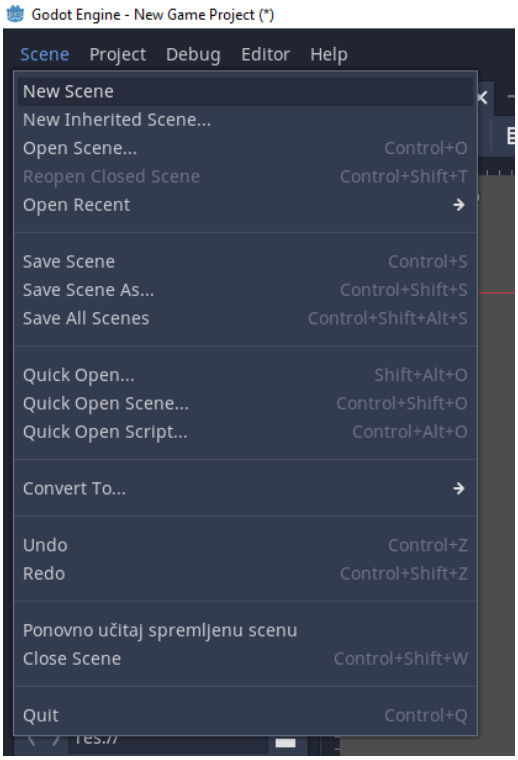
Projekt započinjemo tako da početnoj sceni dodamo korijenski čvor te istu spremimo na disk. To radimo klikom na jednu od opcija za dodavanja čvorova na gornjoj lijevoj strani Godot-a (slika 26).



Slika 26: Opcije dodavanja čvorova

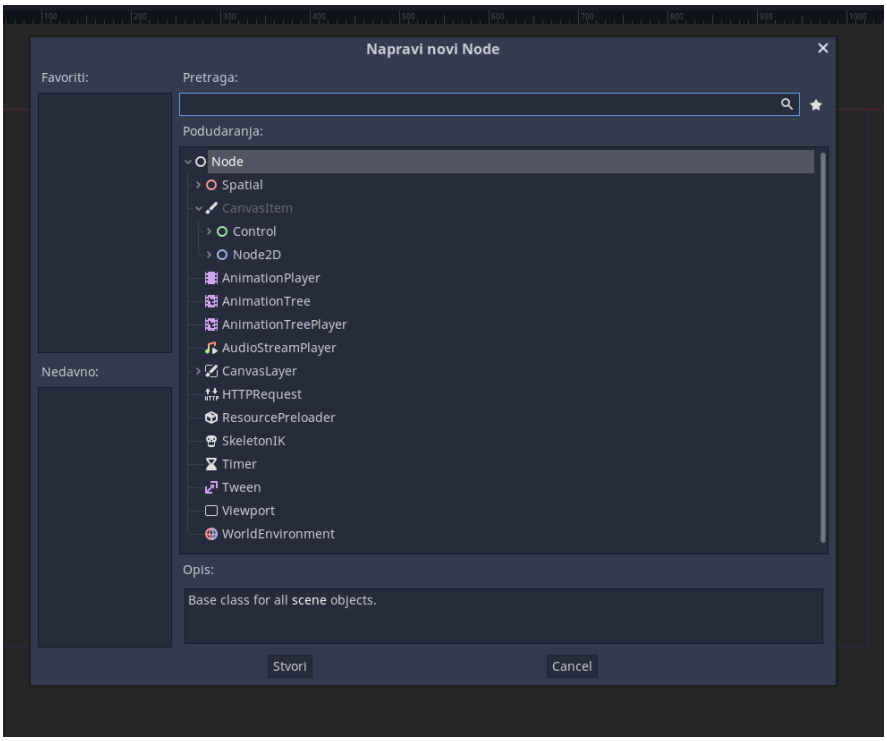
Koji čvor postavimo kao korijen određuje kakvu scenu, te ujedno i kakvu igru imamo. Postavimo li kao korijen čvor „2D scene“, poznat i kao „node2D“, dvodimenzionalan čvor, odredili smo da će naša igra (ili bar ta scena) sadržavati dvije dimenzije.

Ako želimo dodati drugu scenu, to radimo klikom na „Scene“ u alatnoj traki, te klikom na „New Scene“ (slika 27). Toj novoj sceni također moramo odrediti korijenski čvor.



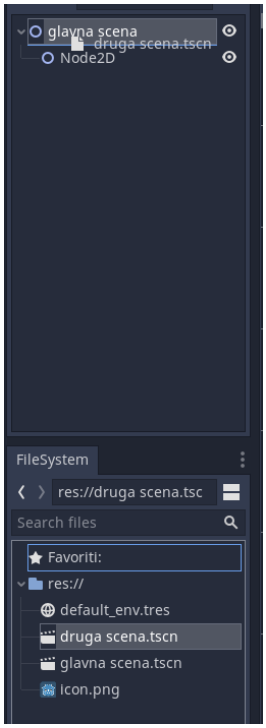
Slika 27: Dodavanje nove scene

Sceni možemo dodati nove čvorove kao djecu pritiskom na kraticu CTRL+a, ili desnim klikom na scenu u pogledu „Scene“, gdje smo prije odredili korijen, te klikom na „Add Child Node“. To nam otvara novi prozor gdje možemo odabrati tip čvora koji želimo dodati (Slika 28). Isti se otvara ako prilikom odabira korijena odaberemo „Other Node“.



Slika 28: „Napravi novi Node“

Ako želimo jednu scenu instancirati u drugoj kao čvor, to radimo potezanjem scene koja će biti instancirana iz pogleda „FyleSystem“ u pogled „Scene“, na scenu roditelj. To je prikazano u slici 29.

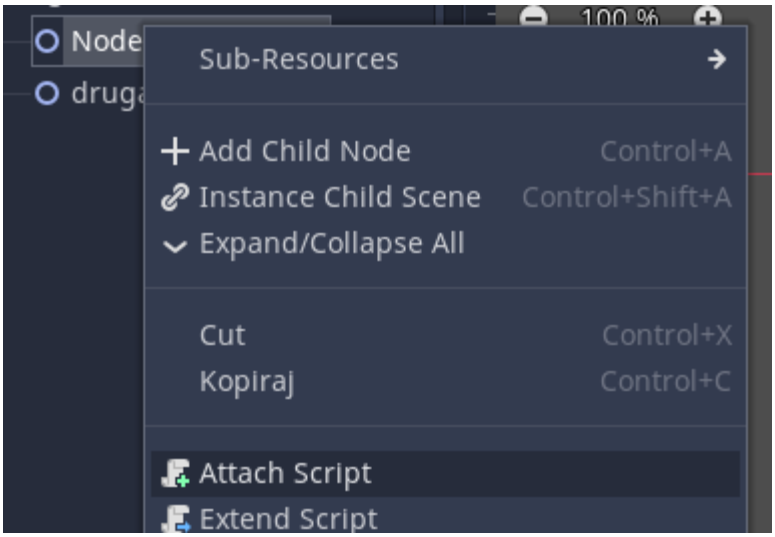


Slika 29: Instanciranje

5.4.4.Skripte

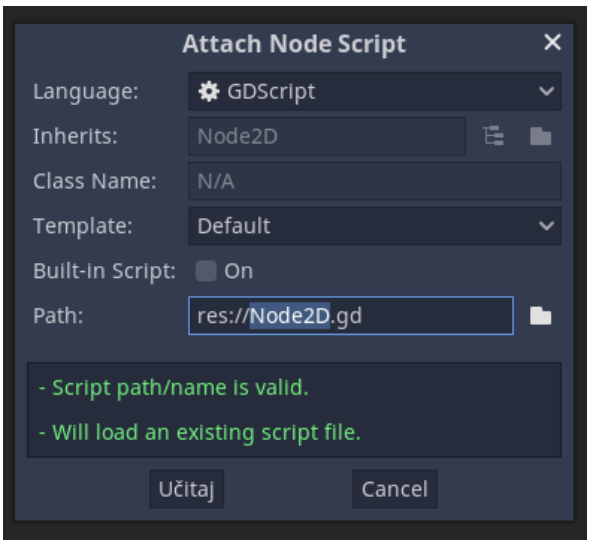
Dok čvorovi i scene služe kao građevni blokovi, skripte služe za upravljanje čvorovima i scenama tijekom rada igre. To su programski blokovi koji se izvršavaju nad, te su povezani sa čvorovima. Svaki čvor može imati pridruženu jednu skriptu.

Skripta se dodaje desnim klikom na željeni čvor, te odabirom opcije „Attach Script“ (slika 30).

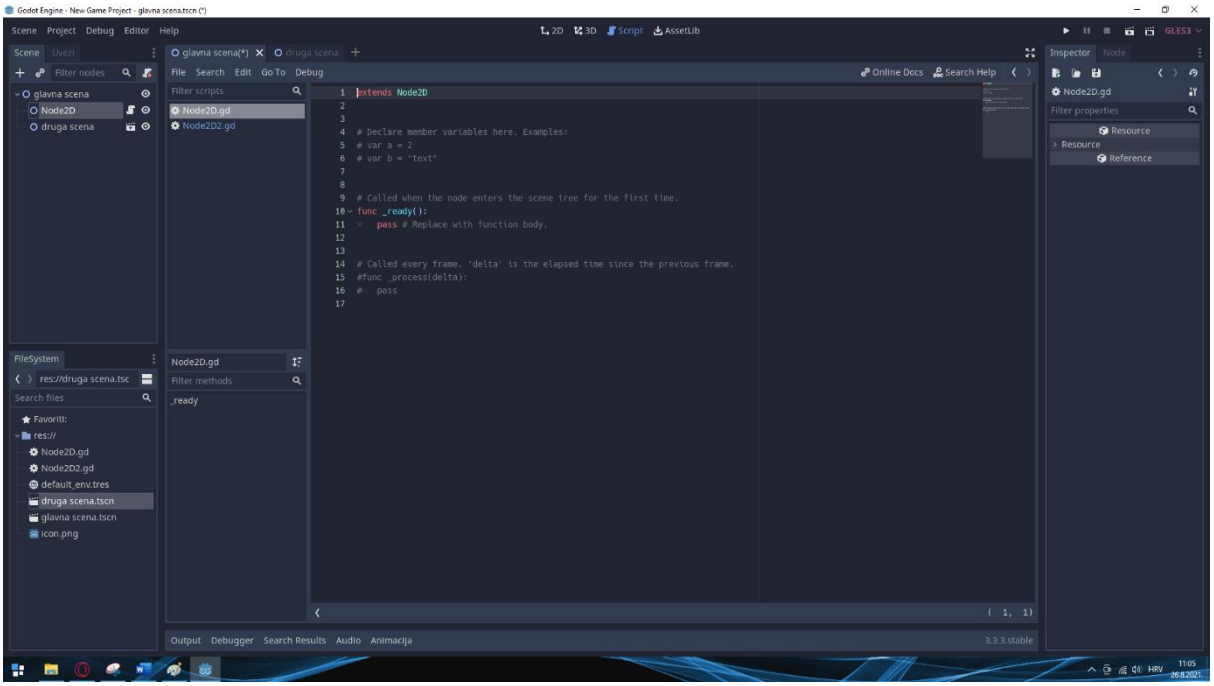


Slika 30: Dodavanje skripte

Klikom na Attach Script se otvara novi prozor sa opcijama za željenu skriptu (slika 31), klikom na „Učitaj“ se stvara nova skripta i otvara pogled uređivanja skripti (slika 32).



Slika 31: „Attach Node Script“

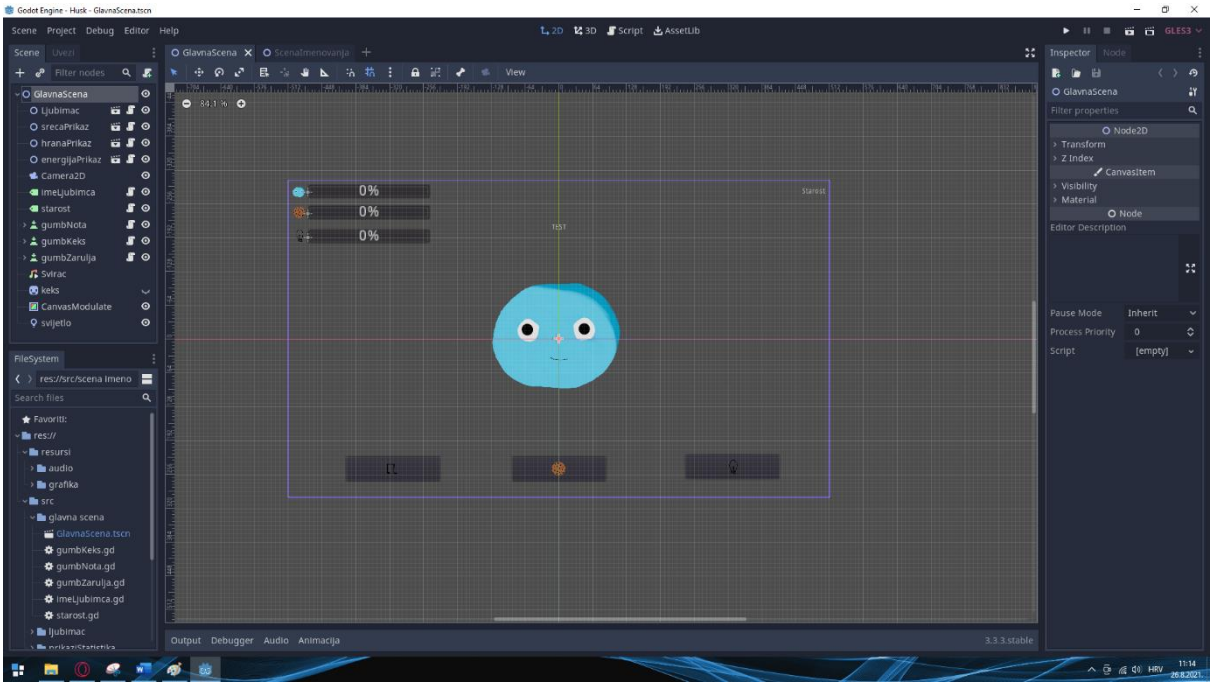


Slika 32: Pogled uređivanja skripti

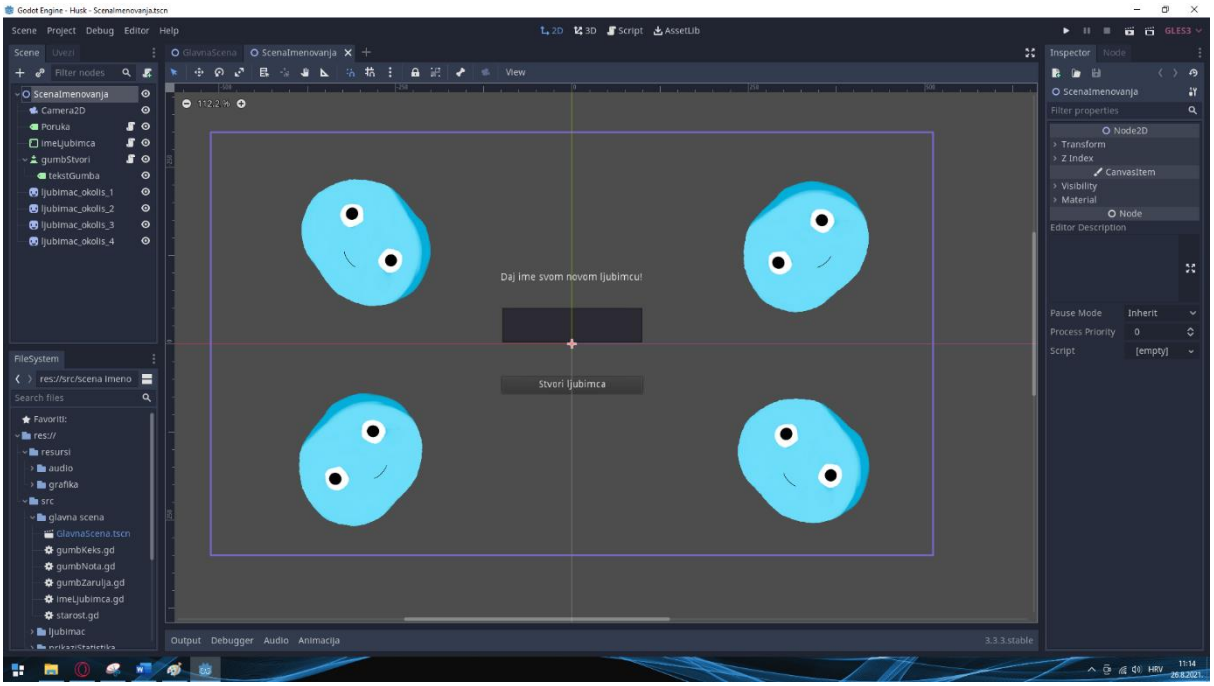
5.4.5.Završena igra

Sa pokrivenim teoretskim znanjem, možemo prikazat strukturu završene igre.

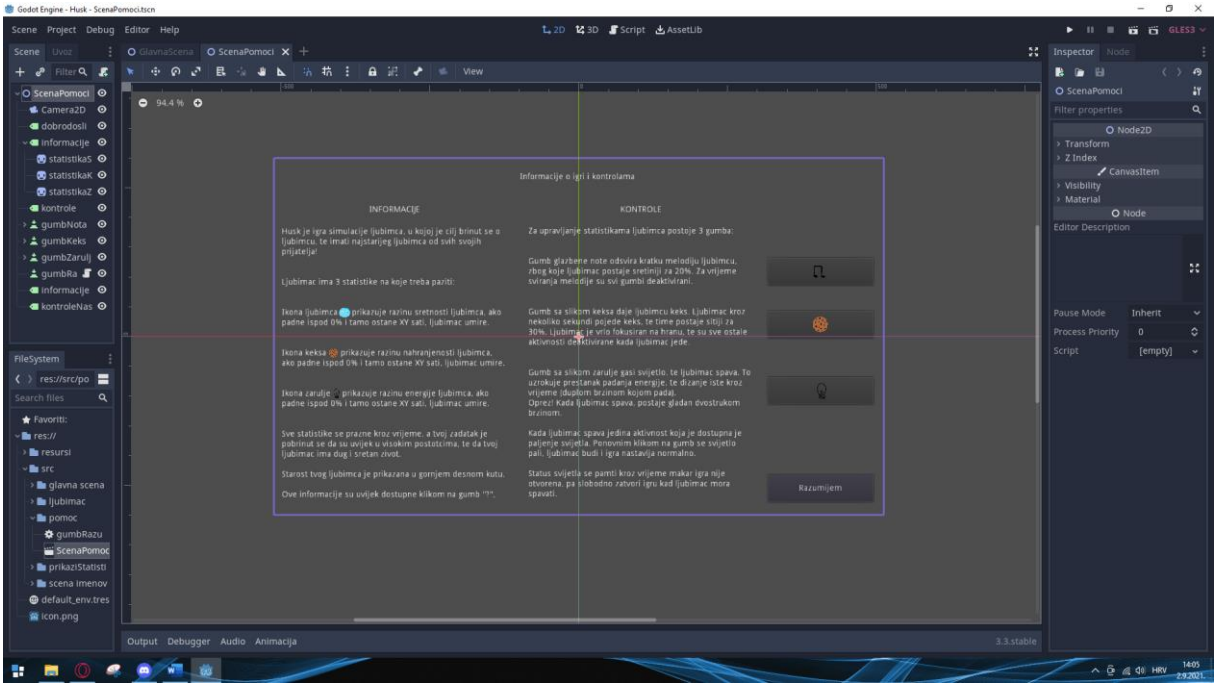
Izrađena igra se sastoji od tri glavne scene, „GlavnaScena“ (Slika 33). koja ima četiri instancirane scene, „ScenaImenovanja“ (Slika 34), i „ScenaPomoci“ (Slika 35).



Slika 33: Glavna scena



Slika 34: Scena Imenovanja



Slika 35: Scena pomoći

Glavna scena sadrži četiri instancirane scene:

- „Ljubimac“, koja sadrži grafiku ljubimca te je na nju vezan glavna skripta igre;
- „srecaPrikaz“, koja sadrži grafiku sretnog ljubimca i traku za napredak, koja prati stanje sreže ljubimca;

- „hranaPrikaz“, koja je identična prijašnjoj no sadrži grafiku keksa te prati stanje hrane;
- „energijaPrikaz“, koja prati stanje energije i sadrži grafiku žarulje,

Te 11 čvorova različitiv uloga:

- „Camera2D“ je čvor istoimenog tipa koji je odgovoran za pozicioniranje „kamere“ kod pokretanja igre, tj. on je odgovoran za određivanje što će biti u vidnom polju prozora igre.
- „imeLjubimca“ i „starost“ su čvorovi tipa „Label“, te prikazuju tekst o imenu i starosti ljubimca
- „gumbNota“, „gumbKeks“ i „gumbZarulja“ su čvorovi tipa „Button“, te svaki ima dijete tipa „Sprite“ sa prikladnom grafikom. Oni su odgovorni za akcije koje povisuju statistike ljubimca
- „Svirac“ je čvor tipa „AudioStreamPlayer“, te je odgovoran za sviranje melodije kada je pritisnut „gumbNota“
- „keks“ je čvor tipa „Sprite“, koji je odgovoran za prikaz keksa kojeg ljubimac jede pritiskom na gumb „gumbKeks“. On je zadano skriven.
- „CanvasModulate“ je čvor istoimenog tipa, koji je zadužen za zamračivanje scene, kada je svijetlo ugašeno.
- „svijetlo“ je čvor tipa „Light2D“, koji je odgovoran za držanje pogleda u normalnoj svijetlosti kada je svijetlo upaljeno.
- „gumbPomoci“ je čvor tipa „Button“, odgovoran za promjenu scene u scenu pomoći ako je igraču potrebna pomoć.

Scena imenovanja sadrži 8 čvorova:

- „Camera2D“, čvor istoimenog tipa
- „Poruka“, čvor tipa „Label“ koji sadrži poruku „Daj ime svom novom ljubimcu!“ ako je igrač novi, ili „Ljubimac ti je umro :/ \nStvori novog i daj mu ime!“ ako je igrač već izgubio jednog ljubimca
- „imeLjubimca“, čvor tipa „TextEdit“, koji je odgovoran za upis imena novog ljubimca
- „gumbStvori“, čvor tipa „Button“ sa dijetetom tipa „Label“, koji je odgovoran za stvaranje novog ljubimca. On je deaktiviran ako je „imeLjubimca“ prazno.
- „ljubimac_okolis_1“, „ljubimac_okolis_2“, „ljubimac_okolis_3“ i „ljubimac_okolis_4“, čvorovi tipa „Sprite“, koji služe kao dekoracija. Oni sadrže grafiku sretnog ljubimca ako je igrač novi, te grafiku tužnog ljubimca ako je igraču umro ljubimac.

Scena pomoći sadrži 10 čvorova:

- „Camera2D“, čvor istoimenog tipa
- „naslov“, čvor tipa „Label“, koji prikazuje naslov scene
- „informacije“, čvor tipa „Label“, sa troje djece tipa „Sprite“, koji prikazuje informacije o igri, a čija djeca prikazuju prikladne sličice odgovarajućih statusa
- „kontrole“, čvor tipa „Label“, koji prikazuje obrazloženje načina igranja igre
- Tri deaktivirana čvora tipa „Button“, kopije čvorova „gumbNota“, „gumbKeks“ i „gumbZarulja“ sa glavne scene, koji se nalaze pored obrazloženja, kako bi igrač vidio na koji se gumb odnosi koje obrazloženje
- „gumbRazumijem“, čvor tipa „Button“ koji mijenja scenu u glavnu scenu
- Čvorovi „informacijeNaslov“ i „kontroleNaslov“ koji prikazuju naslov sekcije informacije i kontrole, centrirane iznad teksta

Igrino funkcioniranje omogućuje četrnaest skripti, svaka vezana za jedan čvor.
Prva, i centralna skripta je „Ljubimac.gd“. Njezin kod je:

```
extends Node2D

var sretan = true;
var ziv = true
var zadnjeVrijeme = null;

var ime = "John Doe"
var sreca = 50.00
var hrana = 50.00
var energija = 50.0
var spava = true
var rodendan = OS.get_unix_time()

var timer = null
var saveTimer = null

func _ready():
   _ucitajPodatke()

   timer = Timer.new()
   add_child(timer)
   timer.connect("timeout", self, "_promijeniStatus")
   timer.set_wait_time(1.0)
   timer.set_one_shot(false)
```

```
      timer.start()


      saveTimer = Timer.new()
      add_child(saveTimer)
      saveTimer.connect("timeout", self, "_spremiPodatke")
      saveTimer.set_wait_time(5.0)
      saveTimer.set_one_shot(false)
      saveTimer.start()


   func _promijeniStatus(var protekloSekundi = 1):
      if(!ziv):
            get_tree().change_scene("res://src/scena
imenovanja/ScenaImenovanja.tscn")
      sreca = sreca - protekloSekundi * ((5 * (100/(sreca if (sreca>1) else
1)))/3600)
      hrana = hrana - protekloSekundi * ((0.00104) if !spava else
(0.00138))
      energija = energija - protekloSekundi * ((0.00174) if !spava else (-
0.00347))


      if(sreca > 100):
            sreca = 100
      if(hrana > 100):
            hrana = 100
      if(energija > 100):
            energija = 100


      if(spava):
            get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_spava.png")
      else:
            if(sreca+hrana+energija<150 || sreca<20 || hrana <20 ||
energija <20):
                  sretan = false;
                  get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")
            else:
                  sretan = true;
                  get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_sretan.png")
      if(sreca < -36000 || hrana < -180 || energija < -150):
            ziv = false
```

```
   func _spremiPodatke():
      var podaci = {
            "ime" : ime,
            "sreca" : sreca,
            "hrana" : hrana,
            "energija" : energija,
            "spava" : spava,
            "rodendan" : rodendan,
            "zadnjeVrijeme" : OS.get_unix_time()
      }
      var spremljeniPodaci = File.new()
      spremljeniPodaci.open("user://ljubimac.json", File.WRITE)
      spremljeniPodaci.store_line(to_json(podaci))
      spremljeniPodaci.close()

   func _ucitajPodatke():
      var spremljeniPodaci = File.new()
      if not spremljeniPodaci.file_exists("user://ljubimac.json"):
            get_tree().change_scene("res://src/scena
imenovanja/ScenaImenovanja.tscn")
      else:
            spremljeniPodaci.open("user://ljubimac.json", File.READ)
            var podaci = parse_json(spremljeniPodaci.get_line())
            spremljeniPodaci.close()
            ime = podaci["ime"]
            sreca = podaci["sreca"]
            hrana = podaci["hrana"]
            energija = podaci["energija"]
            spava = podaci["spava"]
            rodendan = podaci["rodendan"]
            zadnjeVrijeme = podaci["zadnjeVrijeme"]

            var trenutnoVrijeme = OS.get_unix_time()
            var protekloVrijeme = trenutnoVrijeme - zadnjeVrijeme
            _promijeniStatus(protekloVrijeme)
```

Ova skripta ima nekoliko funkcionalnosti. Njezin slijed rada je:

1. Postavi početne varijable
2. Uđu u funkciju _učitajPodatke()
    a. Pokušaj učitati spremljene podatke

i. Ako ne postoje spremljeni podatci, prestani rad i učitaj scenu „Scena imenovanja“
ii. Ako postoje spremljeni podaci, učitaj ih

b. Izračunaj proteklo vrijeme koristeći spremljeno zadnje vrijeme i trenutno vrijeme
c. Simuliraj proteklo vrijeme koristeći funkciju _promijeniStatus(protekloVrijeme)

3. Izađi iz funkcije _učirajPodatke()
4. Kreiraj tajmer koji svaku sekundu pozove funkciju _promijeniStatus()
    a. Ako ljubimac nije živ, prestani rad i učitaj scenu „Scena imenovanja“
    b. Izračunaj promjenu statistika
    c. Ako je bilo koja statistika iznad 100, stavi ju na 100
    d. Ako ljubimac spava, postavi njegovu grafiku u „ljubimac_spava.png“
        i. U suprotnom, ako je sretan postavi njegovu grafiku u „ljubimac_sretan.png“, ako nije, postavi ju u „ljubimac_nesretan.png“
    e. Ako su ispunjeni uvjeti za umrijeti, postati „ziv“ na false.
5. Kreiraj tajmer koji svakih 5 sekundu pozove funkciju _spremiPodatke()
    a. Stvori rječnik sa podacima
    b. Otvori datoteku za spremanje podataka
    c. Zapiši rječnik u datoteku u json formatu
    d. Zatvori datoteku

To je centralna skripta zato jer izvršava najbitnije elemente igre:

- Spremanje i učitavanje podataka o ljubimcu, što omogućuje zatvaranje igre bez gubitka ljubimca
- Simulaciju pada potreba ljubimca
- Određivanje kraja igre, u vidu smrti ljubimca

Skripte „energijaPrikaz.gd“, „hranaPrikaz.gd“ i „srecaPrikaz.gd“ izvršavaju jednaku funkciju ali za različite čvorove. Sve tri imaju isti slijed rada:

- Dohvati prikladnu statistiku iz čvora „Ljubimac“
- Postavi vrijednost trake za napredak u vrijednost dohvaćene statistike
- Stvori tajmer koji ponavlja isto svake sekunde

Kao primjer, programski kod skripte „energijaPrikaz.gd“ je:

```
   extends Node2D


   onready var energijaLjubimac =
get_parent().get_node("Ljubimac").energija


   var _timer = null


   func _ready():
      get_node("Status").set_value(energijaLjubimac)
      _timer = Timer.new()
      add_child(_timer)
      _timer.connect("timeout", self, "updateStatus")
      _timer.set_wait_time(1.0)
      _timer.set_one_shot(false)
      _timer.start()


   func updateStatus():
       energijaLjubimac = get_parent().get_node("Ljubimac").energija
       get_node("Status").set_value(energijaLjubimac)
```

Skripta „glavna scena/imeLjubimca.gd je vrlo jednostavna skripta koja se izvrši samo jednom, sa vrlo kratkim slijedom rada:

- Dohvati ime ljubimca iz čvora „Ljubimac“
- Postavi text u čvoru „imeLjubimca“ u dohvaćeno ime

Njezin kod je:

```
extends Label
onready var ime = get_parent().get_node("Ljubimac").ime
func _ready():
   get_node(".").set_text(ime)
```

Skripta „starost.gd“ je također jednostavnija, no ona se ažurira svake minute kako bi starost bila ispravno prikazana. Njezin kod je:

```
extends Label


onready var rodendan = get_parent().get_node("Ljubimac").rodendan
```

```
var _timer = null

func _ready():
    _postaviStarost();
    _timer = Timer.new()
    add_child(_timer)

    _timer.connect("timeout", self, "_postaviStarost")
    _timer.set_wait_time(60.0)
    _timer.set_one_shot(false)
    _timer.start()

func _postaviStarost():
    var danas = OS.get_unix_time()
    var razlika = danas - rodendan
    var dana = int(razlika) / 60 / 60 / 24
    var sati = int(razlika) / 60 / 60 % 24
    var minuta = int(razlika) / 60 % 60
    var starost = "Starost: %d dana, %d sati, %d minuta" % [dana, sati,
minuta]
    get_node(".").set_text(starost)
```

Slijed rada tog koda je:

1. Dohvati rođendan ljubimca (u unix formatu) iz čvora „Ljubimac“
2. Uđi u funkciju _postaviStarost()
    a. Izračunaj proteklo vrijeme od kreiranja ljubimca
    b. Izračunaj dane, sate i minute iz unix vremena
    c. Postavi vrijednost teksta čvora „starost“prema izračunatim podacima
3. Izađi iz funkcije _postaviStarost()
4. Kreiraj tajmer koji okida funkciju _postaviStarost() svaku minutu

Skripta „gumbNota.gd“ je odgovorna za početak sviranja melodije spremljene u čvoru „Svirac“ pritiskom na gumb „gumbNota“.

Njezin kod je:

```
extends Button
```

```
onready var svirac = get_parent().get_node("Svirac")

func _ready():
   svirac.connect("finished", self, "_zavrsiPjesmu")

func _pressed():
   get_node(".").disabled = true
   get_parent().get_node("gumbKeks").disabled = true
   get_parent().get_node("gumbZarulja").disabled = true
   svirac.play()

func _zavrsiPjesmu():
   get_node(".").disabled = false
   get_parent().get_node("gumbKeks").disabled = false
   get_parent().get_node("gumbZarulja").disabled = false
   if(get_parent().get_node("Ljubimac").sreca<0):
         get_parent().get_node("Ljubimac").sreca = 0
   get_parent().get_node("Ljubimac").sreca += 20
```

Slijed rada tog koda je:

1. Poveži kraj sviranja čvora „Svirac“ sa pokretanjem funkcije _zavrsiPjesmu()
2. Kad je gumb pritisnut:
    a. Deaktiviraj sva tri gumba na ekranu
    b. Počni sviranje melodije
    c. Završetak melodije okine funkciju _zavrsiPjesmu()
        i. Aktiviraj sva tri gumba na ekranu
        ii. Ako je ljubimčeva sreća ispod 0, postavi ju na 0
        iii. Povećaj ljubimčevu sreću za 20
    d. Izađi iz funkcije _zavrsiPjesmu()

Skripta „gumbZarulja.gd“ je odgovorna za gašenje i paljenje svijetla. Tu funkcionalnost realizira aktivacijom i deaktivacijom čvora „svijetlo“.

Kod skripte „gumbZarulja.gd“ je:

```
extends Button

onready var svijetlo = get_parent().get_node("svijetlo")
```

```
func _ready():
   if(get_parent().get_node("Ljubimac").spava):
         svijetlo.enabled = false

func _pressed():
   if(svijetlo.enabled):
         svijetlo.enabled = false
         get_parent().get_node("Ljubimac").spava = true
         get_parent().get_node("gumbKeks").disabled = true
         get_parent().get_node("gumbNota").disabled = true
         if(get_parent().get_node("Ljubimac").energija<0):
               get_parent().get_node("Ljubimac").energija = 0
   else:
         svijetlo.enabled = true
         get_parent().get_node("Ljubimac").spava = false
         get_parent().get_node("gumbKeks").disabled = false
         get_parent().get_node("gumbNota").disabled = false
```

Slijed rada tog koda je:

1. Ako ljubimac spava
    a. Deaktiviraj čvor „svijetlo“
    b. Deaktiviraj druga dva gumba
2. Kada je pritisnut čvor „gumbZarulja“
    a. Ako je čvor „svijetlo“ aktivno
        i. Deaktiviraj svijetlo
        ii. Deaktiviraj druga dva gumba
        iii. Postavi stanje spavanja u čvoru „Ljubimac“ na istinito
        iv. Ako je energija ljubimca ispod nula, postavi ju na nula
    b. U suprotnom
        i. Aktiviraj svijetlo
        ii. Aktiviraj ostale gumbove
        iii. Postavi stanje spavanja u čvoru „Ljubimac“ na lažno

Skripta „gumbKeks.gd“ je najkompliciranija skripta gumba. Ona je zadužena za prikaz keksa, mijenjanja grafike keksa kroz jedenje, te za mijenjanje grafike ljubimca da simulira animaciju jedenja.

Njezin kod je:

```
   extends Button

   onready var keks = get_parent().get_node("keks")
   onready var ljubimac = get_parent().get_node("Ljubimac")

   func _pressed():
      get_node(".").disabled = true;
      get_parent().get_node("gumbNota").disabled = true
      get_parent().get_node("gumbZarulja").disabled = true
      keks.visible = true;
      for i in range(1,4):
            keks.texture = load("res://resursi/grafika/keks_"+ str(i)
+".png")
            _animirajUsta()
            yield(_animirajUsta(), "completed")
      get_node(".").disabled = false;
      get_parent().get_node("gumbNota").disabled = false
      get_parent().get_node("gumbZarulja").disabled = false
      keks.texture = load("res://resursi/grafika/keks_1.png")
      keks.visible = false;
      if(get_parent().get_node("Ljubimac").hrana<0):
            get_parent().get_node("Ljubimac").hrana = 0
      get_parent().get_node("Ljubimac").hrana += 30
      if(ljubimac.sretan):
            ljubimac.get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_sretan.png")
      else:
            ljubimac.get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")


   func _animirajUsta():
      for i in 3:
            if(ljubimac.sretan):
                  ljubimac.get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_sretan.png")
            else:
                  ljubimac.get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")
            yield(get_tree().create_timer(0.3), "timeout")
            ljubimac.get_node("ljubimac_sprite").texture =
load("res://resursi/grafika/ljubimac_otvorena_usta.png")
            yield(get_tree().create_timer(0.3), "timeout")
```

Slijed rada tog koda je:

1. Isključi sva tri gumba
2. Postavi vidljivost čvora „keks“ na vidljivo
3. Ponovi 3 puta:
    a. Učitaj prikladnu grafiku keksa
    b. Pozovi funkciju _animirajUsta()
        i. Ponovi 3 puta:
            1. Postavi grafiku ljubimca na sretan ili nesretan ovisno o sreći
            2. Čekaj 0.3 sekunde
            3. Postavi grafiku ljubimca na ostovrena usta
            4. Čekaj 0.3 sekunde
    c. Izađi iz funkcije _animirajUsta()
4. Aktiviraj sve gumbe
5. Vrati grafiku keksa u početnu
6. Ako je hrana ljubimca ispod nula, postavi ju na nula
7. Povećaj ljubimčevu hranu za 30
8. Vrati ljubimčevu grafiku na početnu

Zadnja skripta glavne scene je „gumbPomoc.gd“. Ona je odgovorna za promjenu scene u scenu „ScenaPomoci“ kada je pritisnut gumb na koji je vezana.

Kod te skripte je:

```
extends Button

func _pressed():
   get_tree().change_scene("res://src/pomoc/ScenaPomoci.tscn")
```

Slijed rada ima samo jedan korak, prebacivanje scene.

Jedina skripta scene „ScenaPomoci“ je „gumbRazumijem.gd“. Ona ima jednaki slijed rada i funkcionalnost kao skripta „gumbPomoc.gd“, no umjesto scene pomoći, ona scenu prebaci u scenu „GlavnaScena“. Njezine kod je:

```
extends Button

func _pressed():
   get_tree().change_scene("res://src/glavna scena/GlavnaScena.tscn")
```

Prva skripta scene „ScenaImenovanja“ je „Poruka.gd“. Ona je zaslužna za postavljanje prikladne poruke i prikladnih ukrasnih ljubimaca ovisno o načinu dolaska na scenu.

Kod skripte „Poruka.gd“ je:

```
extends Label


func _ready():
    var spremljeniPodaci = File.new()
    if spremljeniPodaci.file_exists("user://ljubimac.json"):
        get_node(".").text = "Ljubimac ti je umro :/ \nStvori novog i
daj mu ime!"
        get_parent().get_node("ljubimac_okolis_1").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")
        get_parent().get_node("ljubimac_okolis_2").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")
        get_parent().get_node("ljubimac_okolis_3").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")
        get_parent().get_node("ljubimac_okolis_4").texture =
load("res://resursi/grafika/ljubimac_nesretan.png")
```

Slijed rada te skripte je:

1. Provjeri postoje li spremljeni podaci
2. Ako postoje:
    a. Promijeni tekst u čvoru „Poruka“ u "Ljubimac ti je umro :/ \nStvori novog i daj mu ime!"
    b. Promijeni grafiku ukrasnih ljubimaca u grafiku tužnog ljubimca

Sljedeća skripta ove scene je „scena imenovanja/imeLjubimca.gd“. Ova skripta provjerava prilikom unosa teksta u čvor „imeLjubimca“ da li je tekst prazan. Ako nije prazan, aktivira gumb „gumbStvori“.

Kod te skripte je:

```
extends TextEdit


func _input(event):
    if (!get_node(".").text.empty()):
        get_parent().get_node("gumbStvori").disabled = false
    else:
```

```
        get_parent().get_node("gumbStvori").disabled = true
```

Slijed rada tog koda je:

1. Na unos teksta:
    a. Ako je tekst prazan
        i. Deaktiviraj čvor „gumbStvori“
    b. U suprotnom
        i. Aktiviraj čvor „gumbStvori“

Posljednja skripta ove scene, i same igre je „gumbStvori.gd“, koja je zadužena za stvaranje novog ljubimca. Njezin kod je:

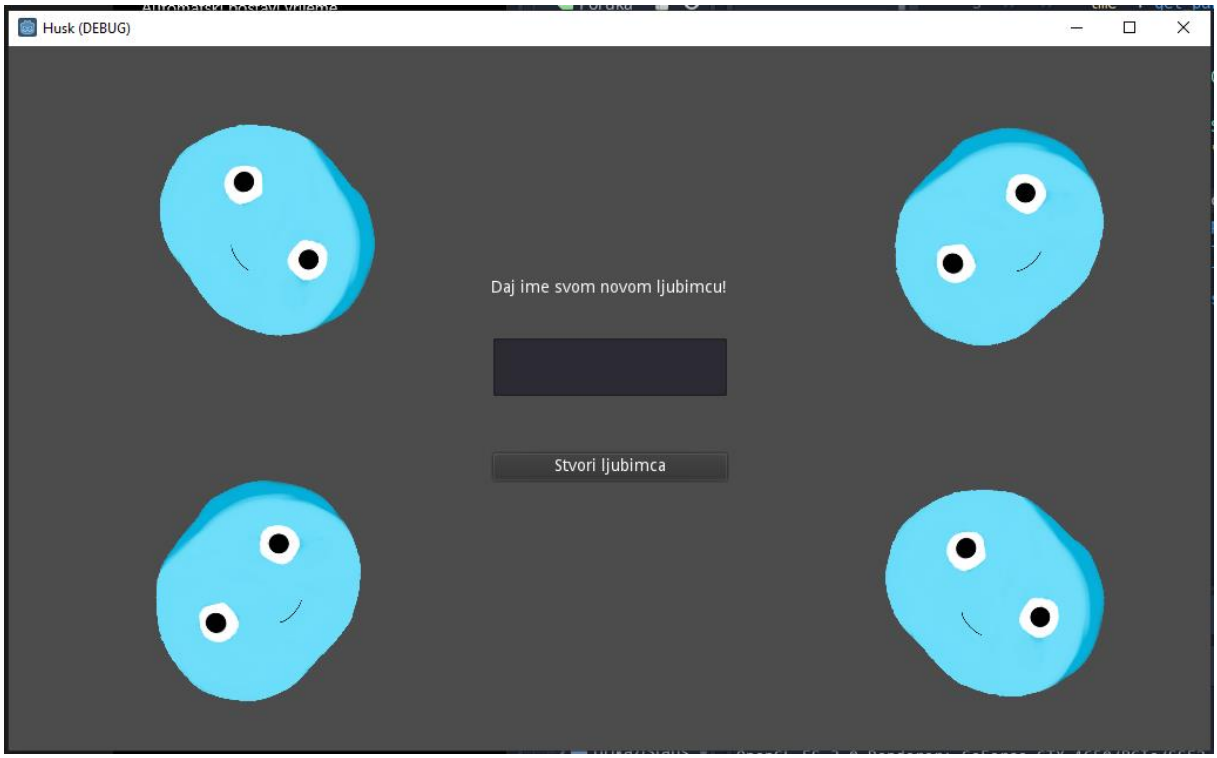
```
extends Button

func _pressed():
   var podaci = {
         "ime" : get_parent().get_node("imeLjubimca").text,
         "sreca" : 50,
         "hrana" : 50,
         "energija" : 50,
         "spava" : true,
         "rodendan" : OS.get_unix_time(),
         "zadnjeVrijeme" : OS.get_unix_time()
   }
   var spremljeniPodaci = File.new()
   spremljeniPodaci.open("user://ljubimac.json", File.WRITE)
   spremljeniPodaci.store_line(to_json(podaci))
   spremljeniPodaci.close()
   get_tree().change_scene("res://src/glavna scena/ ScenaPomoci.tscn")
```

Slijed rada tog koda je:

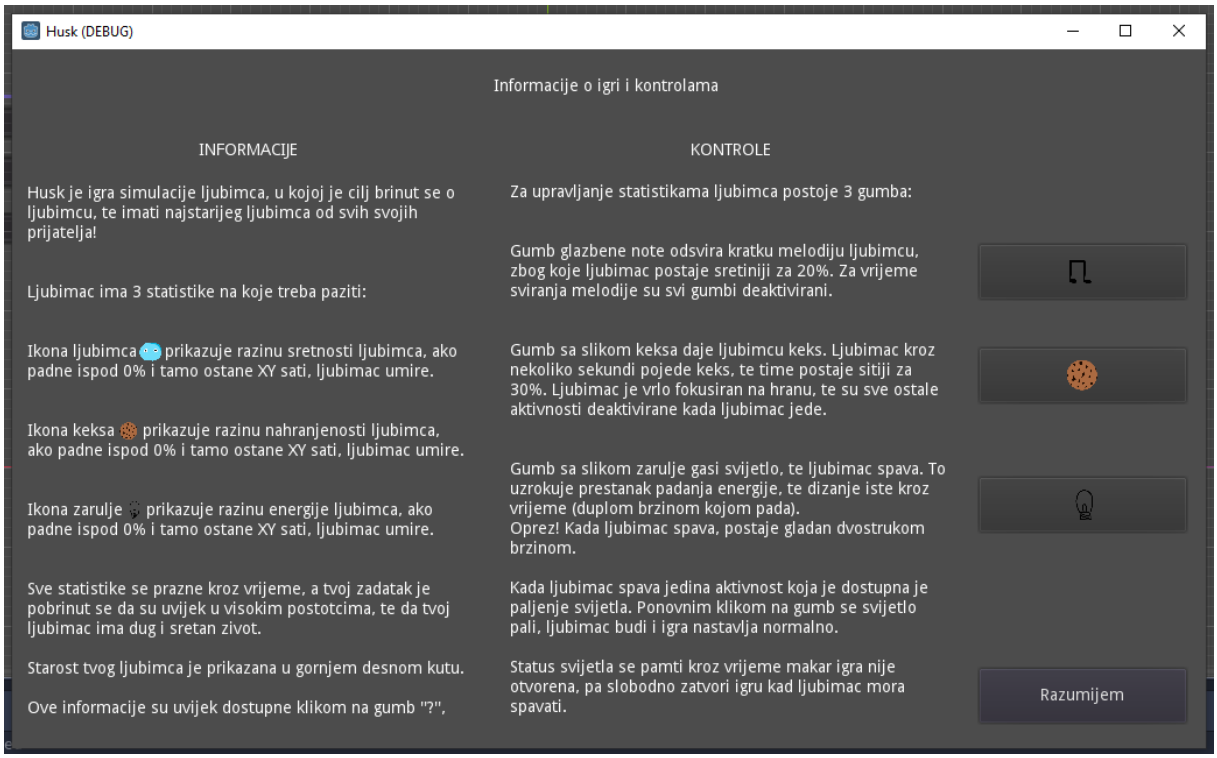
1. Na pritisak čvora „gumbStvori“:
    a. Postavi početne podatke
        i. Iščitaj ime iz čvora „imeLjubimca“
        ii. Dohvati rođendan i zdanje vrijeme iz sistemskog vremena
    b. Otvori datoteku spremljenih podataka za pisanje
    c. Upiši nove podatke
    d. Zatvori datoteku

e. Promijeni scenu u „ScenaPomoci“

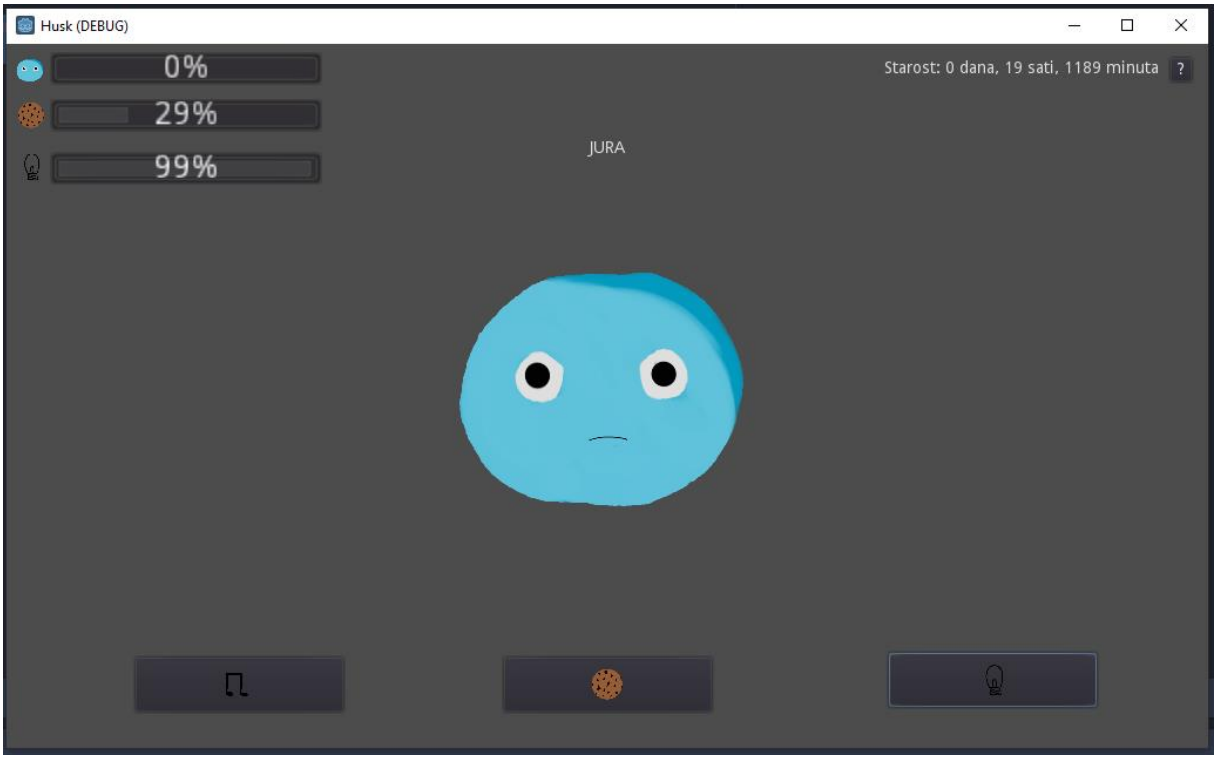
Završni izgledi scene imenovanja novog ljubimca (slika 36), scene pomoći (slika 37), glavne scene (slika 38) i scene imenovanja nakon smrti ljubimca (slika 39) su vidljivi u sljedećim slikama:



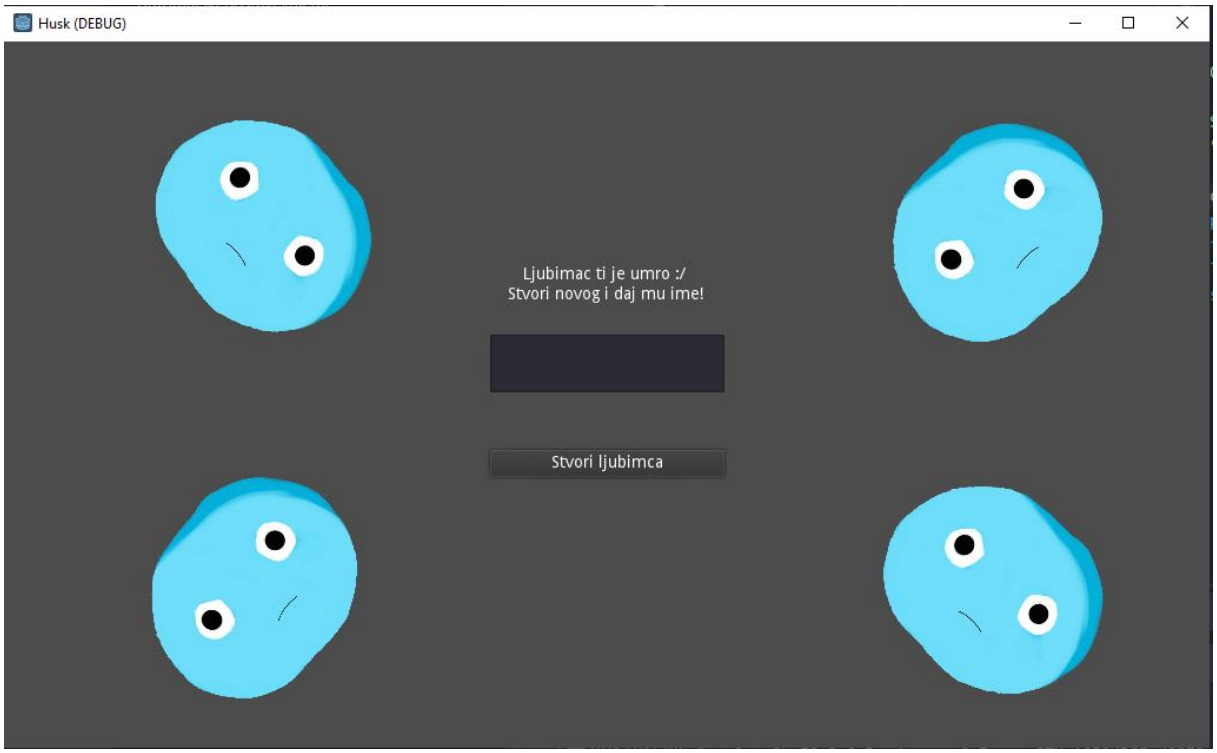
Slika 36: Imenovanje ljubimca kod početka igranja



Slika 37: Pomoć u igri



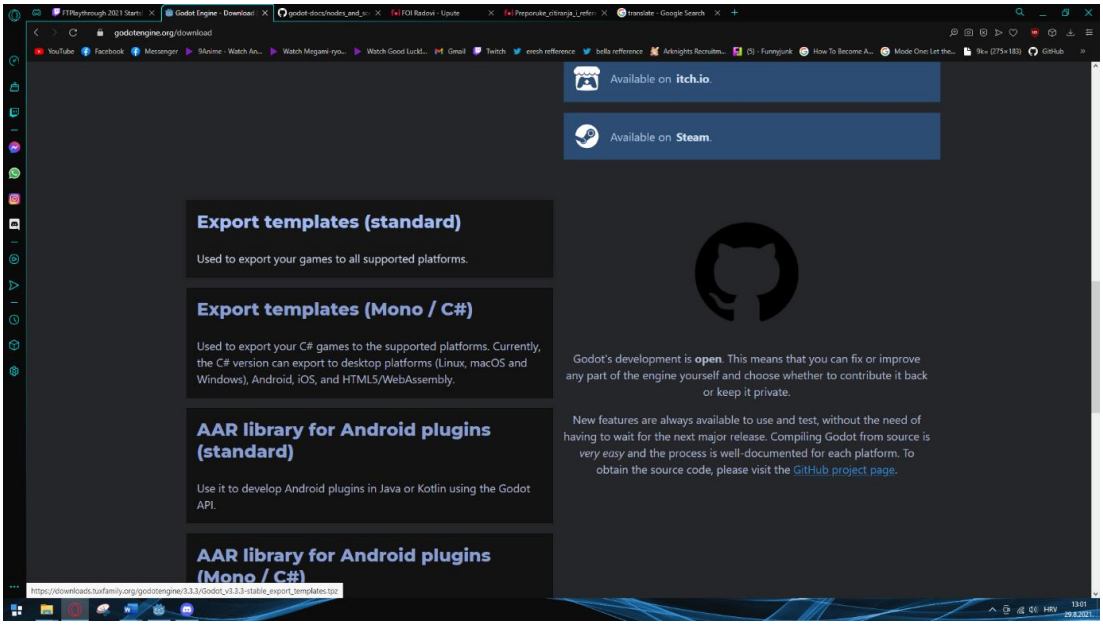
Slika 38: Glavna scena igre



Slika 39: Imenovanje ljubimca nakon smrti ljubimca

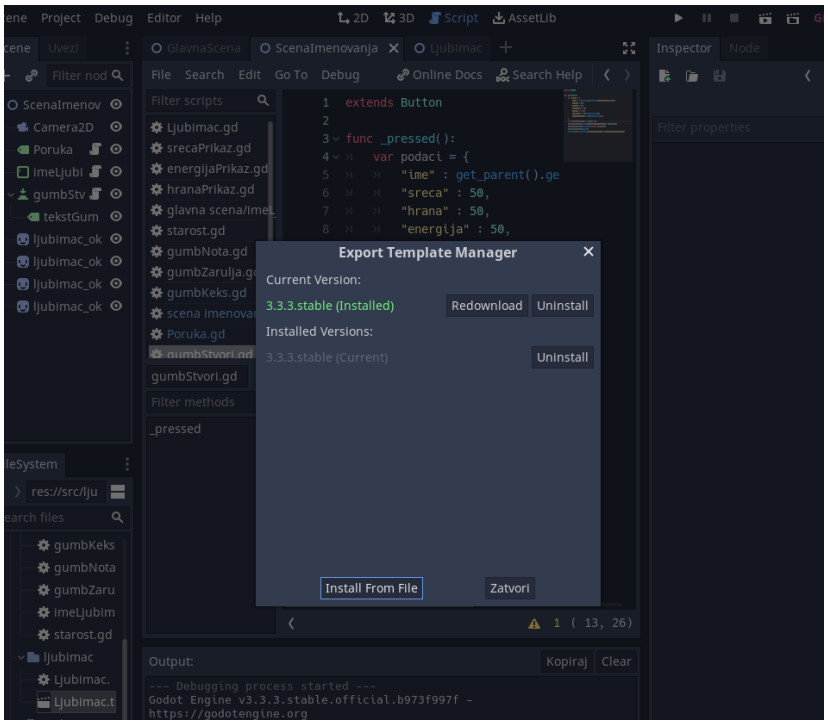
5.4.6. Izvoz igre

Za izvoz igre iz Godot-a u .exe format je potrebno prvo preuzeti predložak za izvoz. Taj predložak se nalazi na istoj stranici kao i sam Godot, godotengine.com/download, no potrebno je kliknuti „Export Templates (Standard)“ za preuzimanje predloška (slika 40).



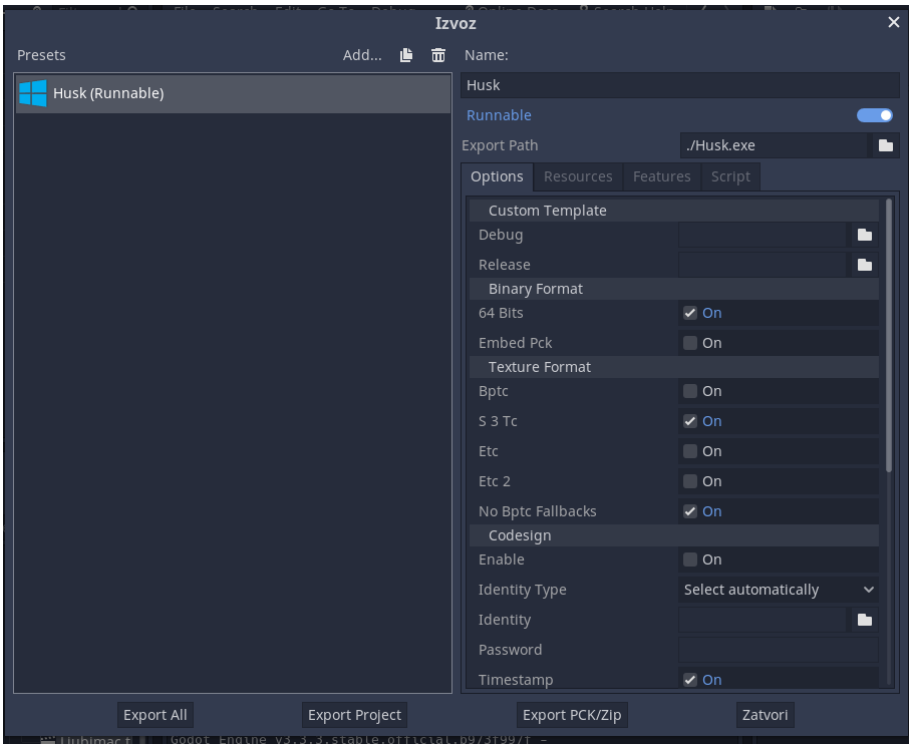
Slika 40: Preuzimanje predloška za izvoz

Nakon preuzimanja predloška, potrebno je u Godot-u ići na „Editor->Manage Export Templates->Install From File“ (Slika 41)



Slika 41: „Install From File“

Po instalaciji predloška, za izvoz igre je potrebno ići na „Project->Export“. To nam otvara novi prozor za izvoz igre (Slika 42).

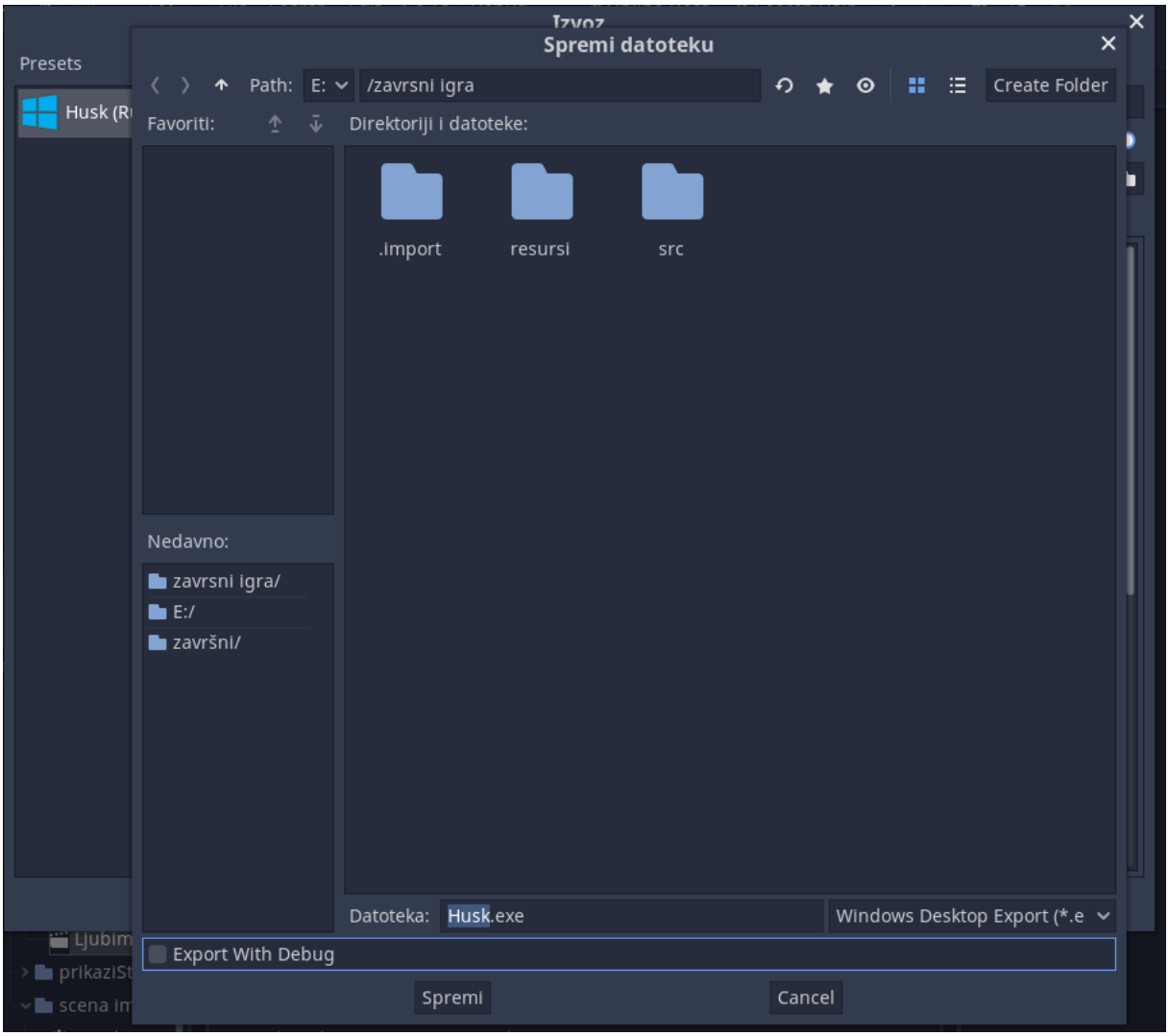


Slika 42: Prozor „Izvoz“

Nakon toga klikom na tipku „Add“ možemo dodati predložak za sustav na koji želimo izvesti. U ovom slučaju to je bio Windows Desktop.

Nakon dodavanja predloška, potrebno je samo kliknuti na predložak u lijevom dijelu, ispuniti željena polja u desnom, te kliknuti „Export Project“. To nam daje .exe i .pck datoteku, koje su obje potrebne za pokretanje igre.

Kako bi osigurali da se pokretanjem igre ne pokrene konzola, potrebno je u pogledu za odabir mape za izvoz, koji se otvara klikom na „Export Project“ odznačiti „Export With Debug“ u donjem lijevom kutu (Slika 43).



Slika 43: Pogled odabira mape sa odznačenim „Export With Debug“

6. Zaključak

Simulacija kućnog ljubimca je u suštini vrlo jednostavan žanr računalnih igara sa nekoliko izuzetno poznatih predstavnika. Izvrsna simulacija kućnog ljubimca bi trebala sadržavati promjenjive ljubimce, koji mogu umrijeti sa nagradom za duže održavanje ljubimca na životu sa socijalnim aspektom koji bi uključio cijelo društvo igrača.

Dok takva simulacija nije postignuta u ovom radu, postignuta je simulacija u kojoj ljubimac može umrijeti, sa socijalnim aspektom prikaza starosti ljubimca.

Ista je postignuta koristeći isključivo aplikacije otvorenog izvora, što pokazuje da cijena ulaska u razvoj računalnih igri nije velika.

Godot posebice je jednostavan no moćan pokretač računalnih igara, sa opsežnom dokumentacijom. Uvelike je ubrzao razvoj igre, te je izvrstan izbor za razvoj svakakvih igara, dokle god korisnik razumije koncepte čvorova, scena i skripti.

Popis literature

[1] Godot Engine, „The game engine you waited for“ [Na internetu]. Dostupno: https://godotengine.org [pristupano 16.8.2021.]

[2] Godot Engine, „Features“ [Na internetu]. Dostupno: https://godotengine.org/features [pristupano 16.8.2021.]

[3] Godot Engine, „GDScript basics“ [Na internetu]. Dostupno: https://docs.godotengine.org/en/stable/getting_started/scripting/gdscript/gdscript_basics.html [pristupano 16.8.2021.]

[4] Open Source Initiative [OSI], „The MIT License“ [Na internetu]. Dostupno: https://opensource.org/licenses/MIT [pristupano 16.8.2021.]

[5] Krita Foundation, „Krita | Digital Painting Creative Freedom“ [Na internetu]. Dostupno: https://krita.org/en/ [pristupano 16.8.2021.]

[6] Krita Foundation, „License | Krita“ [Na internetu]. Dostupno: https://krita.org/en/about/license/ [pristupano 16.8.2021.]

[7] MuseScore, „MuseScore Product Description“, 2020. [Na internetu]. Dostupno: https://musescore.org/en/handbook/musescore-product-description [pristupano 16.8.2021.]

[8] MuseScore „What license is MuseScore released under?“ [Na internetu] https://musescore.org/hr/faq#faq-20203 [pristupano 16.8.2021.]

[9] Draw.io „diagrams.net“ [Na internetu] Dostupno: https://workspace.google.com/marketplace/app/diagramsnet/671128082532 [pristupano 16.8.2021.]

[10] D. Benson, „drawio/LICENSE“, 22.5.2017. [Na internetu]. Dostupno: https://github.com/jgraph/drawio/blob/dev/LICENSE [pristupano 16.8.2021.]

[11] History.com, „Video Game History“, 10.6.2019. [Na internetu]. Dostupno: https://www.history.com/topics/inventions/history-of-video-games [pristupano 16.8.2021.]

[12] R. Chikhani, „The History Of Gaming: An Evolving Community“, 31.10.2015. [Na internetu]. Dostupno: https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/ [pristupano 16.8.2021.]

[13] Jeremy Norman’s HistoryofInformation.com, „NIMATRON: An Early Electromechanical Machine to Play the Game of Nim“, 16.8.2021. [Na internetu]. Dostupno: https://www.historyofinformation.com/detail.php?entryid=4472 [pristupano 18.8.2021.]

[14] A. Chodos, J. Oullette, N. Ramglan i E. Tretkoff, „October 2008 (Volume 17, Number 9) This Month in Physics History October 1958: Physicist Invents First

Video Game“, Listopad 2008. [Na internetu]. Dostupno: https://www.aps.org/publications/apsnews/200810/physicshistory.cfm [pristupano 18.8.2021.]

[15] National Museum of American History, „The Brown Box, 1967-68“ [Na internetu]. Dostupno: https://americanhistory.si.edu/collections/search/object/nmah_1301997 [pristupano 18.8.2021.]

[16] „Smeđa kutija s programskim karticama“ [Slika] Dostupno: https://ids.si.edu/ids/deliveryService?id=NMAH-2006-25049 [pristupano 18.8.2021.]

[17] A. Smith, „1TL200: A MAGNAVOX ODYSSEY“, 16.12.2015. [Na internetu]. Dostupno: https://videogamehistorian.wordpress.com/2015/11/16/1tl200-a-magnavox-odyssey/ [pristupano 19.8.2021.]

[18] National Museum of American History, „Magnavox Odyssey Video Game Unit, 1972“ [Na internetu]. Dostupno: https://americanhistory.si.edu/collections/search/object/nmah_1302004

[19] Centre for Computing History, „Atari PONG“ [Na internetu]. Dostupno: http://www.computinghistory.org.uk/det/4007/Atari-PONG/ [pristupano 19.8.2021.]

[20] Matthews V., „The Many Different Types of Video Games & Their Subgenres“, 12.4.2018. [Na internetu]. Dostupno: https://www.idtech.com/blog/different-types-of-video-game-genres [pristupano 19.8.2021.]

[21] Wayback Petz, „The History of Petz“ [Na internetu]. Dostupno: http://waybackpetz.com/history [pristupano 19.8.2021.]

[22] Rossen J, „A Brief History of the Tamagotchi“, 5.7.2021. [Na internetu]. Dostupno: https://www.mentalfloss.com/article/642373/tamagotchi-history [pristupano 19.8.2021.]

[23] „Tamagotchi“, 23.6.2005. [slika]. Dostupno: https://upload.wikimedia.org/wikipedia/commons/f/f2/Tamagotchi_0124_ubt.jpeg [pristupano 19.8.2021.]

[24] The Digi Archive, „History of Digimon“ [Na internetu]. Dostupno: https://thedigiarchive.com/history-of-digimon/ [pristupano 19.8.2021.]

[25] „Digimon V-Pets uređaji“ [slika]. Dostupno: https://i.postimg.cc/brwgN3Gt/Digimon-v-pet-version-1-guide.jpg [pristupano 19.8.2021.]

[26] Barsby O, „‘Sonic Adventure 2’s’ Chao Garden: the most unnecessarily complex minigame ever made“, 3.11.2020. [Na internetu]. Dostupno: https://theboar.org/2020/11/sonic-adventure-2s-chao-garden-the-most-unnecessarily-complex-minigame-ever-made/ [pristupano 21.8.2021.]

[27] „Sonic Adventure“, u *Sonic News Network*. Dostupno: https://sonic.fandom.com/wiki/Sonic_Adventure [pristupano 21.8.2021.]

[28] „Sonic Adventure 2“, u *Sonic News Network*. Dostupno: https://sonic.fandom.com/wiki/Sonic_Adventure_2 [pristupano 21.8.2021.]

[29] „Tiny Chao Garden“, u *Sonic News Network*. Dostupno: https://sonic.fandom.com/wiki/Tiny_Chao_Garden [pristupano 21.8.2021.]

[30] Zakeh Ltd, „Pou“, 4.2.2021. [Na internetu]. Dostupno: https://play.google.com/store/apps/details?id=me.pou.app&hl=en&gl=US [pristupano 21.8.2021.]

[31] „Pou“, u *Pou Wiki.* Dostupno: https://pou.fandom.com/wiki/Pou [pristupano 21.8.2021.]

[32] 68k Mentat, (16.8.2017.) „Dogz Version 1.1. Playthrough“, *Youtube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=6lKSn_cHw5k [pristupano 21.8.2021.]

[33] „Death“, u *Chao Island*. Dostupno: https://chao-island.com/info-center/life-cycle/death [pristupano 21.8.2021.]

[34] „Chaos Chao“, u Chao Island. Dostupno: https://chao-island.com/info-center/advanced/chaos-chao [pristupano 21.8.2021.]

[35] S. Deutzmann „VIDEO GAME DEFINITION OF THE WEEK: “GAMEPLAY LOOP”“, [Blog post]. 15.10.2020. [Na internetu]. Dostupno: https://engagedfamilygaming.com/videogames/video-game-definition-week-gameplay-loop/ [pristupano 21.8.2021.]

[36] Godot Engine „Scenes and nodes“ [Na internetu]. Dostupno: https://docs.godotengine.org/en/stable/getting_started/step_by_step/scenes_and_nodes.html [pristupano 26.8.2021.]

Popis slika

Prilozi

1. GotodProjekt.zip
2. Husk.zip