LATVIJAS UNIVERSITĀTE

EKSAKTO ZINĀTŅU UN TEHNOLOĢIJU FAKULTĀTE

SPĒLES IZSTRĀDE, IZMANTOJOT BEVY SPĒĻU DZINĒJU

KVALIFIKĀCIJAS DARBS

Darba autors:

Kristiāns Francis Cagulis, kc22015

Darba vadītājs:

prof. Mg. dat. Jānis Iljins

RĪGA 2025

TODOS

ANOTĀCIJA

Kvalifikācijas darbā ir izstrādāta spēle „Maze Ascension“, kas piedāvā spēlētājiem izaicinājumu iziet cauri proceduāli ģenerētiem sešstūrainam labirintiem. Spēle ir veidota, izmantojot Rust programmēšanas valodu un Bevy spēļu dzinēju.

Darba gaitā tika izstrādāta „hexlab“ bibliotēka labirintu ģenerēšanai, kas tika atdalīta no galvenās spēles loģikas. Labirintu ģenerēšanai tiek izmantots rekursīvās atpakaļizsekošanas algoritms, kas nodrošina, ka katrai šūnai var piekļūt no jebkuras citas šūnas.

Spēle ir izstrādāta kā vienspēlētāja režīmā ar progresējošu grūtības pakāpi, kur katrs nākamais līmenis piedāvā lielāku labirintu. Spēle ir pieejama gan kā lejupielādējama versija Windows, Linux un macOS platformām, gan kā tīmekļa versija, izmantojot WebAssembly tehnoloģiju.

Atslēgvārdi:

Labirints datorspēle, sistēmas prasības, programmatūras prasību specifikācija, Bevy, ECS, papilspējas.

ABSTRACT

The qualification work includes the game “Maze Ascension”, which offers players the challenge to pass through procedurally generated hexagons mazes. The game is built using the Rust programming language and Bevy game engine.

The work included the development of a “hexlab” library for maze generation, which was separated from the main game logic. The maze generation is a recursive backtracking algorithm which ensures that each cell can be accessed from any other cell.

The game is designed as a single-player mode with progressive difficulty with each successive level offering a larger maze. The game is available as a downloadable version for Windows, Linux and macOS platforms, and as as a web-based version using WebAssembly technology.

līdz 850 rakstzīmēm

Keywords:

Maze, comtuper game, system requirements, software requirements specification, Bevy, ECS, power-ups.

SATURS

APZĪMĒJUMU SARAKSTS

Audio – Skaņas komponentes, kas ietver gan skaņas efektus, gan fona mūziku.

CI/CD – nepārtraukta integrācija un nepārtraukta izvietošana;

DPD – datu plūsmas diagramma;

ECS – entitāšu komponentu sistēma (angl. Entity-Component-System[1]);

GitHub[2] – izstrādātāju platforma, kas ļauj izstrādātājiem izveidot, glabāt, pārvaldīt un kopīgot savu kodu;

Jaucējtabula[3] – jeb heštabula (angl. hash table[4]) ir datu struktūra, kas saista identificējošās vērtības ar piesaistītajām vērtībām.

Laidiens – Programmatūras versija, kas ir gatava izplatīšanai lietotājiem un satur īpašas funkcijas, uzlabojumus vai labojumus.

PPA – programmatūras projektējuma apraksts;

PPS – programmatūras prasību specifikācija;

Papildspēja – objekts, kas kā spēles mehānika spēlētājam piešķir īslaicīgas priekšrocības vai papildu spējas (angl. power-up[5]);

Pirmkods – Cilvēkam lasāmas programmēšanas instrukcijas, kas nosaka programmatūras darbību.

Renderēšana – Process, kurā tiek ģenerēts vizuāla izvade.

Sēkla – Skaitliska vērtība, ko izmanto nejaušo skaitļu ģeneratora inicializēšanai.

Spēlētājs – lietotāja ieraksts vienas virtuālās istabas kontekstā.

[1] https://en.wikipedia.org/wiki/Entity_component_system
[2] https://en.wikipedia.org/wiki/GitHub
[3] https://lv.wikipedia.org/wiki/Jauc%C4%93jtabula
[4] https://en.wikipedia.org/wiki/Hash_table
[5] https://en.wikipedia.org/wiki/Power-up

IEVADS

Nolūks

Šī dokumenta mērķis ir raksturot sešstūru labirinta spēles „Maze Ascension“ programmatūras prasības un izpētīt Bevy spēļu dzinēja iespējas.

Darbības sfēra

Darba galvenā uzmanība ir vērsta uz būtisku spēles mehāniku ieviešanu, tostarp procedurālu labirintu ģenerēšanu, spēlētāju navigācijas sistēmu, papildspēju integrāciju un vertikālās progresijas mehāniku, vienlaikus ievērojot minimālisma dizaina filozofiju.

Spēles pamatā ir sešstūra formas plāksnes, kas, savukārt, veido sešstūra formas labirintus, kuri rada atšķirīgu vizuālo un navigācijas izaicinājumu. Spēlētāju uzdevums ir pārvietoties pa šiem labirintiem, lai sasniegtu katra līmeņa beigas. Spēlētājiem progresējot, tie sastopas ar arvien sarežģītākiem labirintiem, kuros nepieciešama stratēģiska domāšana, izpēte un papildspēju izmantošana.

Spēlētājam progresējot, tie sastopas ar dažādiem uzlabojumiem un papildspējām, kas stratēģiski izvietoti labirintos. Šī funkcija padziļina spēlēšanas pieredzi, veicinot izpēti un eksperimentēšanu ar dažādām spēju kombinācijām, radot dinamiskākus un aizraujošākus spēles scenārijus.

No tehniskā viedokļa darbā tiek pētīta šo funkciju īstenošana, izmantojot Bevy entitāšu komponentu sistēmas (ECS) arhitektūru. Tas ietver stabilu spēles vides sistēmu izstrādi, stāvokļa pārvaldības mehānismus un efektīvu Bevy iebūvēto funkcionalitāšu izmantošanu.

No darbības sfēras apzināti izslēgta daudzspēlētāju funkcionalitāte un sarežģīti grafiskie efekti, koncentrējoties uz pulētu viena spēlētāja pieredzi ar skaidru, minimālistisku vizuālo noformējumu. Šāda mērķtiecīga pieeja ļauj padziļināti izpētīt spēles pamatmehāniku, vienlaikus nodrošinot projekta vadāmību un tehnisko iespējamību.

Saistība ar citiem dokumentiem

PPS ir izstrādāta, ievērojot LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis“[1] un LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai“[2] standarta prasības.

papildināt dokumentu sarakstu

Pārskats

Dokumenta ievads satur ...

Pirmajā nodaļa tiek aprakstīti ...

Otrajā nodaļā tiek ...

Trešajā nodaļā tiek aprakstīta ...

1. VISPĀRĒJAIS APRAKSTS

1.1. Esošā stāvokļa apraksts

Pašreizējo spēļu izstrādes ainavu raksturo pieaugoša interese pēc neatkarīgajām spēlēm un modernu, efektīvu spēļu dzinēju izmantošana. Izstrādātāji arvien biežāk meklē rīkus, kas piedāvā elastību, veiktspēju un lietošanas ērtumu. Spēļu dzinējs Bevy ar savu moderno arhitektūru un Rust programmēšanas valodas izmantošanu gūst arvien lielāku popularitāti izstrādātāju vidū, pateicoties tā drošām un vienlaicīgām funkcijām.

1.2. Pasūtītājs

Sistēma nav izstrādāta pēc konkrēta pasūtītāja pieprasījuma, tā ir raksturota un projektēta ar iespēju realizēt pēc studenta iniciatīvas kvalifikācijas darba ietvaros.

1.3. Produkta perspektīva

„Maze Ascension“ ir izstrādāta kā daudzplatformu spēle, izmantojot nepārtrauktas integrācijas un nepārtrauktas izvietošanas (CI/CD) darbplūsma[3], lai vienkāršotu izstrādes un izplatīšanas procesu. Šī darbplūsma ir konfigurēts tā, lai kompilētu spēli vairākām platformām, tostarp Linux, macOS, Windows un WebAssembly (WASM). Tas nodrošina, ka spēle ir pieejama plašai auditorijai, nodrošinot konsekventu un saistošu pieredzi dažādās operētājsistēmās un vidēs.

Spēle tiek izplatīta, izmantojot „GitHub releases“[4] un itch.io[6], kas ir populāra neatkarīgo spēļu platforma, kas ļauj viegli piekļūt un izplatīt spēles visā pasaulē. Izmantojot šīs platformas, datorspēle gūst dažādu maksājumu modeļu un kopienas iesasaistes funkcijas, tādējādi palielinot spēles sasniedzamību un atpazīstamību.

1.4. Darījumprasības

Sistēmas izstrādē galvenā uzmanība tiks pievērsta sekojošu darījumprasību īstenošanai, lai nodrošinātu stabilu un saistošu lietotāja pieredzi:

1) Spēles progresēšana un līmeņu pārvaldība: Sistēma automātiski pārvaldīs spēlētāju virzību pa spēles līmeņiem, nodrošinot vienmērīgu pāreju, kad spēlētāji progresē un saskaras ar jauniem izaicinājumiem. Progress tiks saglabāts lokāli spēlētāja ierīcē.

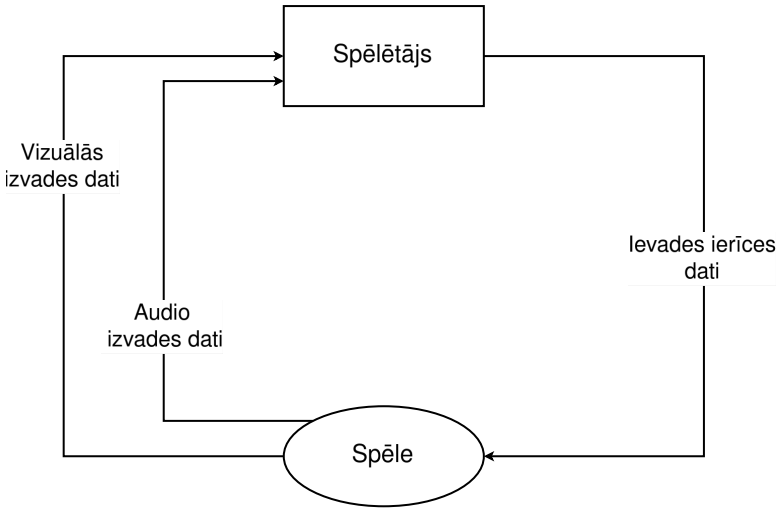
[6] https://itch.io/

2) Nevainojama piekļuve spēlēm: Spēlētāji varēs piekļūt spēlei un spēlēt to bez nepieciešamības izveidot lietotāja kontu vai pieteikties. Tas nodrošina netraucētu piekļuvi spēlei, ļaujot spēlētājiem nekavējoties sākt spēlēt.
3) Savietojamība ar vairākām platformām: sistēma būs pieejama vairākās platformās, tostarp Linux, macOS, Windows un WebAssembly, nodrošinot plašu pieejamību un sasniedzamību.
4) Kopienas iesaiste: Spēle izmantos itch.io[6] kopienas funkcijas, lai sadarbotos ar spēlētājiem, apkopotu atsauksmes un veicinātu atbalstošu spēlētāju kopienu.
5) Regulāri atjauninājumi un uzturēšana: CI/CD cauruļvadu veicinās regulārus atjauninājumus un uzturēšanu, nodrošinot, ka spēle ir atjaunināta ar jaunākajām funkcijām un uzlabojumiem.

1.5. Sistēmas lietotāji

Sistēma ir izstrādāta, ņemot vērā vienu lietotāja tipu – spēlētājs. Spēlētāji ir personas, kas iesaistās spēlē, lai pārvietotos pa tās labirinta struktūrām. Tā kā spēlei nav nepieciešami lietotāja konti vai autentifikācija, visiem spēlētājiem ir vienlīdzīga piekļuve spēles funkcijām un saturam no spēles sākuma brīža.

Ar lietotājiem saistītās datu plūsmas ir attēlotas sistēmas nultā līmeņa DPD (skat. 1.1. att.)



1.1. att. **0. līmeņa DPD**

uzlabot diagrammu

1.6. Vispārējie ierobežojumi

1) Izstrādes vides un tehnoloģijas ierobežojumi:
   a) Programmēšanas valodas un Bevy spēles dzinēja tehniskie ierobežojumi;

b) Responsivitāte;

c) Starpplatformu savietojamība: Linux, macOS, Windows un WebAssembly.

1.7. Pieņēmumi un atkarības

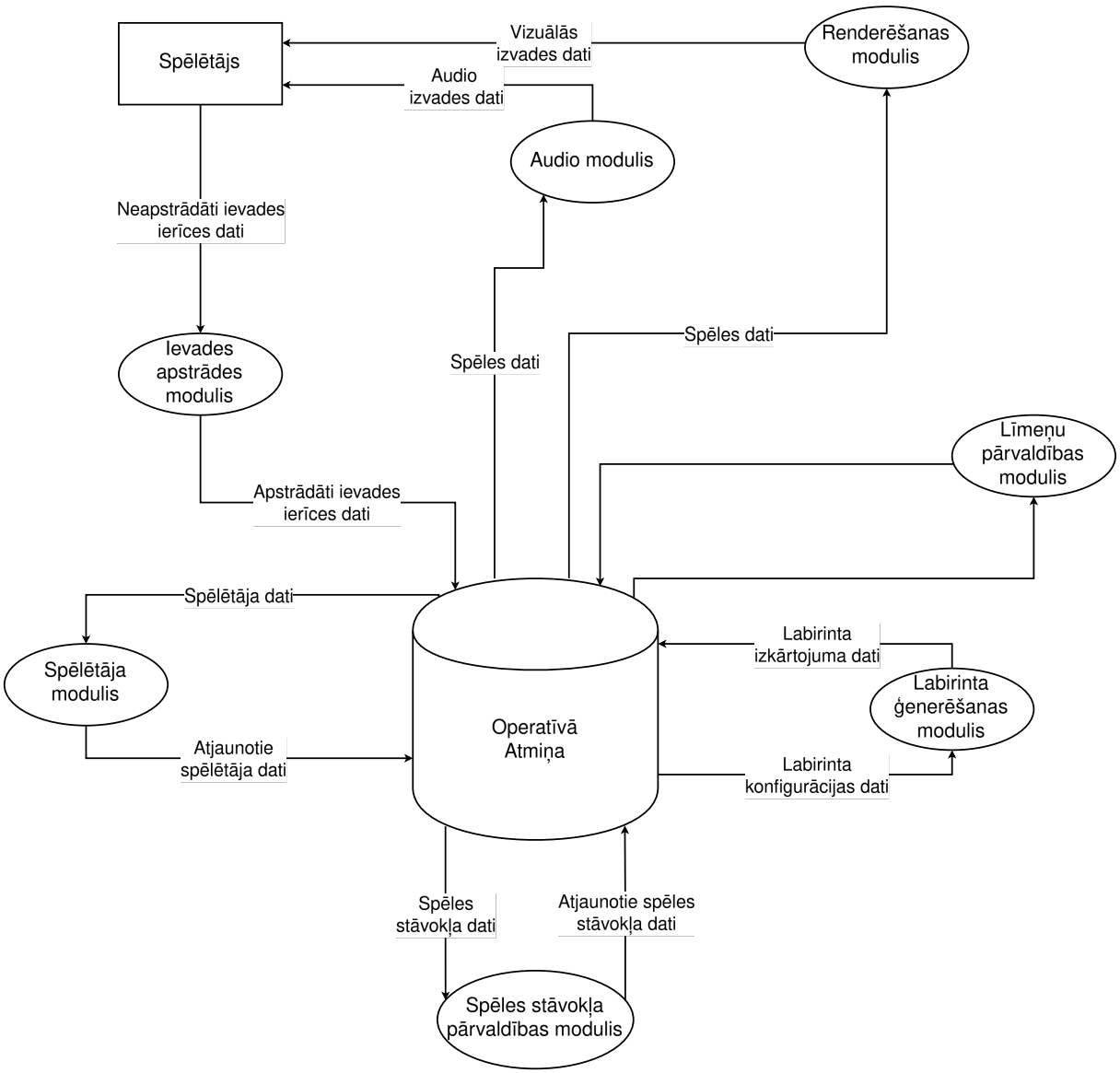
- Tehniskie pieņēmumi:
  - Spēlētāja ierīcei jāatbilst minimālajām aparatūras prasībām, lai varētu palaist uz Bevy spēles dzinēja balstītas spēles.
  - ierīcei jāatbalsta OpenGL 3.3 vai WebGL 2.0, lai nodrošinātu pareizu atveidošanu.
  - tīmekļa spēļu spēlēšanai (WebAssembly versija) pārlūkprogrammai jābūt mūsdienīgai un saderīgai ar WebAssembly.
  - ekrāna izšķirtspējai jābūt vismaz 800x600 pikseļu, lai spēle būtu optimāla.
- Veiktspējas atkarība:
  - Spēle ir atkarīga no Bevy spēles dzinēja (0.14 vai jaunāka versija).
- Veiksmīga kompilēšana un izvietošana ir atkarīga no CI/CD darbplūsmai saderības ar:
  - Linux kompilācijām;
  - macOS kompilācijām;
  - Windows kompilācijām;
  - WebAssembly kompilāciju.
- Izplatīšanas atkarības:
  - Pastāvīga itch.io[6] platformas pieejamība spēļu izplatīšanai.
  - CI/CD darbplūsmai nepieciešamo kompilēšanas rīku un atkarību uzturēšana.
- Izstrādes atkarības:
  - Rust programmēšanas valoda (stabilā versija);
  - Cargo pakešu pārvaldnieks;
  - Nepieciešamie Bevy spraudņi un atkarības, kā norādīts projekta Cargo.toml failā.
- Lietotāja vides pieņēmumi:
  - Spēlētājiem ir pamata izpratne par labirinta navigāciju un mīklu risināšanas koncepcijām.
  - Lietotāji var piekļūt un lejupielādēt spēles no itch.io[6] platformas.
  - Spēlētājiem ir ievadierīces (tastatūra/pele), ar kurām kontrolēt spēli.

2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

2.1. Funkcionālās prasības

1. līmeņa datu plūsmas diagramma (skat. 2.1. att.) ilustrē galvenos procesus spēles „Maze Ascension“ sistēmā. Diagrammā attēloti septiņi galvenie procesi: ievades apstrādātājs, spēles stāvokļa pārvaldnieks, labirinta ģenerators, spēlētāja modulis, spēles līmeņu pārvaldnieks, atveidošanas jeb renderēšanas un skaņas jeb audio moduļi. Šie procesi mijiedarbojas ar vienu datu krātuvi – operatīvo atmiņu (RAM) – un vienu ārējo lietotāju – spēlētājs.

Ievades apstrādes modulis uztver un apstrādā spēlētāja ievades datus. Spēles stāvokļa modulis pārrauga vispārējo spēles stāvokli. Labirinta ģeneratora modulis izveido un pārvalda labirinta struktūras. Spēlētāja modulis apstrādā visas ar spēlētāju saistītās kustības, sadursmes un papildspēju mijiedarbības. Spēles līmeņu pārvaldnieks kontrolē līmeņu virzību un stāvokli. Renderēšanas un audio moduļi pārvalda attiecīgi vizuālo un audio izvadi.



2.1. att. **1. līmeņa DPD**

uzlabot diagrammu

2.1.1. Funkciju sadalījums moduļos

Tabulā 2.1. ir sniegts visaptverošs spēles funkcionalitātes sadalījums pa tās galvenajiem moduļiem. Katram modulim ir noteikti konkrēti pienākumi, un tas ietver funkcijas, kas veicina kopējo spēles sistēmu.

2.1. tabula **Funkciju sadalījums pa moduļiem**

Modulis	Funkcija	Identifikators
Ievades apstrādes modulis	Ievades notikumu apstrāde	
	Ievades stāvokļa atjaunināšana	
	Ievades validācija	
Spēles stāvokļa pārvaldības modulis	Spēļu stāvokļa pārvaldība	
	Spēles cilpas pārvaldība	
	Stāvokļu pāreju apstrāde	
	Spēles notikumu apstrāde	

Spēlētāja modulis	Kustības vadība	
	Sadursmju apstrāde	
	Papildsēju pārvaldība	
	Spēlētāju stāvokļa atjaunināšana	
Labirinta ģenerēšanas modulis	Labirinta būvētājs	LGMF01
Līmeņu pārvaldības modulis	Līmeņu ielāde	
	Progresa izsekošana	
	Pāreju apstrāde	
	Stāvokļa saglabāšana	
	Stāvokļa ielāde	
Renderēšanas modulis	Labirinta renderēšana	
	Spēlētāja renderēšana	
	Lietotājsaskarnes renderēšana	
	Vizuālo efektu renderēšana	
Audio modulis	Skaņas efektu atskaņošana	
	Mūzikas pārvaldība	
	Audio stāvokļu apstrāde	

2.1.2. Ievades apstrādes modulis

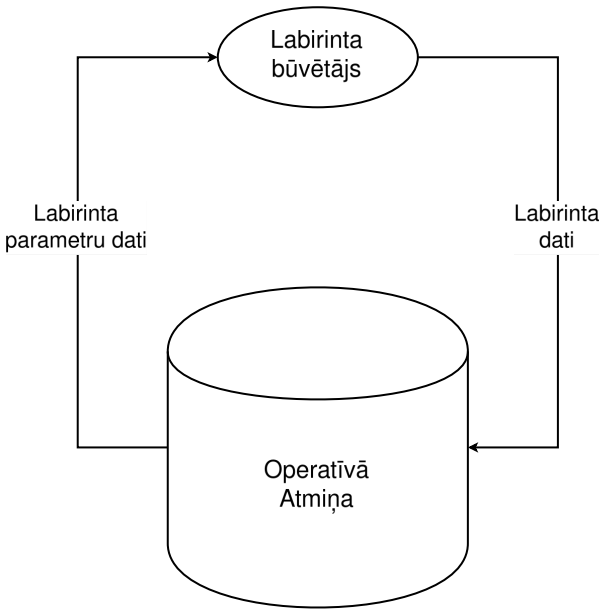
2.1.3. Spēles stāvokļa pārvaldības modulis

2.1.4. Spēlētāja modulis

2.1.5. Labirinta ģenerēšanas modulis

Apakšnodaļa ietver labirinta moduļa funkcijas. Moduļa funkcionalitāte ir izmantota sešstūraina labirinta ģenerēšanai. Moduļa funkciju datu plūsmas ir parādītas 2. līmeņa datu plūsmas diagrammā (skat. 3 att.) Labirinta būvēšanas funkcija ir aprakstītas atsevišķā tabulā (skat. LGMF01 tab.)

Modularitātes un atkārtotas lietojamības apsvērumu dēļ labirinta ģenerēšanas funkcionalitāte tika pārnesta uz ārēju bibliotēku „hexlib“[5]. Šis lēmums ļauj labirinta ģenerēšanas loģiku atkārtoti izmantot dažādos projektos un lietojumprogrammās, veicinot atkārtotu koda izmantošanu. Iekapsulējot labirinta ģenerēšanu atsevišķā bibliotēkā, to ir vieglāk pārvaldīt un atjaunināt neatkarīgi no galvenās lietojumprogrammas, nodrošinot, ka labirinta ģenerēšanas algoritma uzlabojumus vai izmaiņas var veikt, neietekmējot programmu.



2.2. att. **Labirinta ģenerēšanas moduļa 2. līmeņa DPD**

2.2. tabula **Labirinta būzētājs**

Funkcijas nosaukums
Labirinta būzētājs
Funkcijas identifikators
LGMF01
Apraksts
Izveido sešstūrainu labirintu ar norādītajiem parametriem.
Ievade
1) Rādiuss – Labirinta rādiuss. Obligāts parametrs, kas nosaka labirinta izmēru. 2) Sēkla – Neobligāta sēkla nejaušo skaitļu ģeneratoram. Ja norādīta, nodrošina reproducējamu labirinta ģenerēšanu ar vienādiem parametriem. Ja nav norādīta, tiek izmantota nejauša sēkla. 3) Sākuma pozīcija – Neobligāta sākotnējā pozīcija labirintā. Ja norādīta, labirinta ģenerēšana sāksies no šīs pozīcijas. Ja nav norādīta, tiek izvēlēta nejauša derīga sākuma pozīcija.
Apstrāde
1) Validē ievades parametrus: a) Pārbauda rādiusa esamību un derīgumu;

b) Validē sākuma pozīciju, ja tāda norādīta; 2) Izveido sākotnējo labirinta struktūru: a) Inicializē tukšu labirintu ar norādīto rādiusu; b) Katrai šūnai iestata sākotnējās (visas) sienas. 3) Validē stākuma prozīciju, ja tāda norādīta. 4) Ģenerē labirintu: a) Rekursīvi izveido ceļus, noņemot sienas starp šūnām; b) Izmanto atpakaļizsekošanu, kad sasniegts strupceļš.
Izvade
1) Jaucējtabulu, kas satur: a) Sešstūra koordinātes kā atslēgās; b) Sešstūra objekti ar: i) Pozīcijas koordinātēm (x, y); ii) Sienu konfigurāciju (8-bitu maska).
Paziņojumi
1) Lai izveidotu labirintu, ir jānorāda rādiuss. 2) Sākuma pozīcija ir ārpus labirinta robežām. 3) Neizdevās izveidot labirintu.

2.1.6. Līmeņu pārvaldības modulis

2.1.7. Renderēšanas modulis

2.1.8. Audio modulis

2.2. Nefunkcionālās prasības

2.2.1. Veiktspējas prasības

Uz sistēmas veiktspēju ir sekojošas prasības:

- Labirinta ģenerēšana: Jebkura izmēra labirintam jātiek uzģenerētam ātrāk par 1 sekundi.
- Spēles ielāde: Spēlei jāstartējas ātrāk par 3 sekundēm.
- Kadru ātrums: Spēlei jādarbojas ar vismaz 60 kadriem sekundē.

- Ievades apstrāde: Spēlētāja kustībām jātiek apstrādātām bez manāmas aizkaves (< 16ms).

2.2.2. Uzticamība

Uz sistēmas uzticamību ir sekojošas prasības:

- Kļūdu apstrāde: spēlei jāapstrādā kļūdas graciozi, bez sistēmas atteicēm.
- Saglabāšana: spēles progresam jātiek automātiski saglabātam pēc katra līmeņa.
- Atjaunošanās: spēlei jāspēj atjaunoties pēc negaidītas aizvēršanas.

2.2.3. Atribūti

2.2.3.1. Izmantojamība

Uz sistēmas izmantojamību ir sekojošas prasības:

- 90% jaunu lietotāju jāspēj lietot visas tiem pieejamās funkcijas bez palīdzības.
- Teksta fonta izmēram datoru ekrāniem jābūt vismaz 14 pikseļiem, labas salasāmības nodrošināšanai.

2.2.3.2. Drošība

Uz drošību risinājumiem ir sekojošas prasības:

- Spēles pirmkods ir iekļauts kopā ar izpildāmo bināro failu;
- Spēle nemodificē un nelasa lietotāja vai operētājsistēmas failus, izņemot izmantoto bibliotēku failus.

2.2.3.3. Uzturamība

Pret sistēmas izstrādājamo programmatūras uzturamību tiek izvirzītas sekojošās prasības:

- API dokumentācijas pārklājumam jābūt vismaz 80%.
- Koda testēšanas pārklājumam jābūt vismaz 70%.

2.2.3.4. Pārnesamība

- Platformas: spēlei jādarbojas uz 64 bitu Windows, Linux un macOS.
- Prasības: spēlei jādarbojas uz datora ar vismaz:
  - 4GB operatīvo atmiņa (RAM);
  - Integrēto grafisko karti;
  - Divu-kodolu procesoru.

2.2.4. Paplašināmība

- Labirinta ģenerēšana: jābūt iespējai viegli pievienot jaunus ģenerēšanas algoritmus,
- Līmeņu dizains: jābūt iespējai viegli pievienot jaunus līmeņus.
- Papildinājumi: koda arhitektūrai jāatbalsta jaunu funkciju pievienošana.

2.2.5. Ārējās saskarnes prasības

3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

3.1. Daļējs funkciju projektējums

pievienot funkciju

pievienot projektējumu +diagrammas

3.2. Saskarņu projektējums

saskarnes (UI/ UX)

4. TESTĒŠANAS DOKUMENTĀCIJA

Šajā nodaļā ir aprakstīta spēles „Maze Ascension“ testēšanas process. Testēšana tika veikta divos galvenajos virzienos – statiskā un dinamiskā testēšana, izmantojot gan automatizētus rīkus, gan manuālu pārbaudi.

4.1. Statiskā testēšana

Statiskā testēšana ir svarīga daļa no projekta kvalitātes nodrošināšanas. „Clippy“[6] tiek izmantots koda analīzei, meklējot potenciālas problēmas un neoptimālus risinājumus. Papildus noklusētajiem noteikumiem, tika aktivizēti stingrāki koda kvalitātes pārbaudes līmeņi: „pedantic“ režīms nodrošina padziļinātu koda stila pārbaudi, „nursery“ aktivizē eksperimentālās pārbaudes, un „unwrap_used“ un „expect_used“ brīdina par potenciāli nedrošu kļūdu apstrādi. Šie papildu noteikumi palīdz uzturēt augstāku koda kvalitāti un samazināt potenciālo kļūdu skaitu.

4.2. Dinamiskā testēšana

uztakstīt dinamisko testēšanu

4.2.1. Manuālā integrācijas testēšana

4.2.2. Automatizēti testi

5. PROGRAMMAS PROJEKTA ORGANIZĀCIJA

Kvalifikācijas darba prasības paredz, ka programmatūru un dokumentāciju autors veido patstāvīgi, vadoties pēc darba vadītāja norādījumiem.

uzrakstīt projekta organizāciju

5.1. Kvalitātes nodrošināšana

Augstas koda kvalitātes nodrošināšana ir jebkura projekta būtisks aspekts. Lai to panāktu, tiek izmantoti vairāki rīki un prakses, kas palīdz uzturēt tīru, efektīvu un uzticamu koda.

Viens no galvenajiem rīkiem, kas tiek izmantots ir „Clippy“[6], kas analizē iespējamās problēmas un iesaka uzlabojumus (skat. 4.1. Statiskā testēšana nodaļu).

Kopā ar „Clippy“ tiek arī izmantots „Rustfmt“[7], koda formatētājs, lai uzturētu vienotu koda formatējumu visā projektā. Šis rīks automātiski formatē kodu saskaņā ar Rust stila vadlīnijām[8].

Turklāt visas publiskās funkcijas un datu struktūras hexlab bibliotēkā ir dokumentētas[9]. Šajā dokumentācijā ir ietverti detalizēti apraksti un lietošanas piemēri, kas ne tikai palīdz saprast kodu, bet arī kalpo kā dokumentācijas testēšanas veids. Darbinot „cargo doc“[10], tiek ģenerēja un validēja dokumentācija, nodrošinot, ka piemēri ir pareizi un aktuāli.

Programmatūras prasības specifikācija ir izstrādāta, ievērojot LVS 68:1996 standarta „Programmatūras prasību specifikācijas ceļvedis“[1] un LVS 72:1996 standarta „Ieteicamā prakse programmatūras projektējuma aprakstīšanai“[2] standarta prasības.

5.2. Konfigurācijas pārvaldība

Pirmkods tiek pārvaldīts, izmantojot „git“[11] versiju kontroles sistēmu. Repozitorijs tiek izvietots platformā „GitHub“. Rīku konfigurācija ir definēta vairākos failos:

- „justfile“ – satur atkļūdošanas un laidiena komandas dažādām vidēm[12]:
  - atkļūdošanas kompilācijas ar iespējotu pilnu atpakaļsekošanu;
  - laidiena kompilācijas ar iespējotu optimizāciju.
- „GitHub Actions“ darbplūsmas, kas apstrādā:

- koda kvalitātes pārbaudes (vienībtesti, statiskie testi, formatēšana, dokumentācijas izveide).
- kompilācijas un izvietotošanas darbplūsma, kas:
  * izveido Windows, Linux, macOS un WebAssembly versijas;
  * publicē bināros failus GitHub platformā;
  * izvieto tīmekļa versiju itch.io[6] platformā.

Versiju specifikācija notiek pēc semantiskās versiju atlases[13] (MAJOR.MINOR.PATCH):

1) MAJOR – galvenā versija, nesaderīgas izmaiņas, būtiskas koda izmaiņas.
2) MINOR – atpakaļsaderīgas funkcionalitātes papildinājumi.
3) PATCH – ar iepriekšējo versiju saderīgu kļūdu labojumi.

5.3. Darbietilpības novērtējums

uzrakstīt darbietilpības novērtējumu

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Institūcija SIA "Latvijas standarts", *Programmatūras prasību specifikācijas ceļvedis*, nr. 68. 1996.

[2] Institūcija SIA "Latvijas standarts", *Ieteicamā prakse programmatūras projektējuma aprakstīšanai*, nr. 72. 1996.

[3] "CI/CD: The what, why, and how". [Tiešsaiste]. Pieejams: https://github.com/resources/articles/devops/ci-cd

[4] GitHub komanda, "Par laidieniem". [Tiešsaiste]. Pieejams: https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases

[5] Kristiāns Francis Cagulis, "Hexlab bibliotēka". [Tiešsaiste]. Pieejams: https://crates.io/crates/hexlab

[6] Rust Projekta Izstādātāji, "Clippy". [Tiešsaiste]. Pieejams: https://doc.rust-lang.org/clippy/usage.html

[7] Rust Projekta Izstādātāji, "Rustfmt". [Tiešsaiste]. Pieejams: https://github.com/rust-lang/rustfmt

[8] Rust Projekta Izstādātāji, "Rust stila ceļvedis". [Tiešsaiste]. Pieejams: https://doc.rust-lang.org/nightly/style-guide/

[9] Kristiāns Francis Cagulis, "Hexlab bibliotēkas dokumentācija". [Tiešsaiste]. Pieejams: https://docs.rs/hexlab/latest/hexlab/

[10] Rust Projekta Izstādātāji, "cargo-doc". [Tiešsaiste]. Pieejams: https://doc.rust-lang.org/cargo/commands/cargo-doc.html

[11] "Versijas kontroles sistēmas git dokumentācija". [Tiešsaiste]. Pieejams: https://git-scm.com/doc

[12] Casey Rodarmor, "Just programmētāja rokasgrāmata". [Tiešsaiste]. Pieejams: https://just.systems/man/en/

[13] "Semantiskā versiju veidošana". [Tiešsaiste]. Pieejams: https://semver.org/

PIELIKUMI

2079 words