LATVIJAS UNIVERSITĀTE

EKSAKTO ZINĀTŅU UN TEHNOLOĢIJU FAKULTĀTE

DATORIKAS NODAĻA

SPĒLES IZSTRĀDE, IZMANTOJOT BEVY SPĒĻU DZINĒJU

KVALIFIKĀCIJAS DARBS

Darba autors:

Kristiāns Francis Cagulis, kc22015

Darba vadītājs:

Mg. dat. Jānis Iljins

RĪGA 2025

ANOTĀCIJA

Kvalifikācijas darbā ir izstrādāta spēle „Maze Ascension“, kas piedāvā spēlētājiem izaicinājumu iziet cauri procedurāli 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, papildspējas.

ABSTRACT

The qualification work “Game development using Bevy game engine” 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.

Keywords:

Maze, computer 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-komponenšu sistēma (angl. Entity-Component-System)[1];

Entitāte – unikāls identifikators, kas apvieno saistītās komponentes;

Jaucējtabula[1] – jeb heštabula (angl. hash table)[2] ir datu struktūra, kas saista identificējošās vērtības ar piesaistītajām vērtībām;

Komponente – datu struktūra, kas satur tikai datus bez loģikas;

Notikums – īslaicīga ziņojuma struktūra, kas tiek izmantota komunikācijai starp sistēmām;

PPA – programmatūras projektējuma apraksts;

PPS – programmatūras prasību specifikācija;

Papildspēja – spēles mehānika, kas spēlētājam piešķir īslaicīgas priekšrocības vai papildu spējas (angl. power-up)[3];

Pirmkods – cilvēkam lasāmas programmēšanas instrukcijas, kas nosaka programmatūras darbību;

Procedurāla ģenerēšana – algoritmisks satura radīšanas process, kas automātiski ģenerē datus izpildes laikā, nevis izmanto manuāli, iepriekš veidotu saturu;

Renderēšana – process, kurā tiek ģenerēta vizuāla izvade;

Resurss – globāli pieejama datu struktūra, kas nav piesaistīta konkrētai entitātei;

Režģis – strukturēts šūnu izkārtojums, kas veido spēles pasaules pamata struktūru;

Sistēma – loģikas vienība, kas apstrādā entitātes ar specifiskām komponentēm;

Spēlētājs – entitāte, kas reprezentē lietotāja vadāmo objektu spēlē;

Sēkla – skaitliska vērtība, ko izmanto nejaušo skaitļu ģeneratora inicializēšanai;

Šūna – sešstūraina režģa viena pozīcija, kas definē telpu, kuru var aizņemt viena plāksne;

---

[1] https://lv.wikipedia.org/wiki/Jauc%C4%93jtabula

[2] https://en.wikipedia.org/wiki/Hash_table

[3] https://en.wikipedia.org/wiki/Power-up

WASM – WebAssembly – zema līmeņa assemblera tipa kods, kas var darboties modernos tīmekļa pārlūkos.

IEVADS

Nolūks

Šī programmatūras prasību specifikācija (PPS) ir izstrādāta, lai definētu un dokumentētu procedurāli ģenerētas sešstūru labirinta spēles „Maze Ascension“ programmatūras prasības, arhitektūru un tehnisko implementāciju. Dokumenta galvenais uzdevums ir nodrošināt skaidru un visaptverošu projekta aprakstu, kas kalpo gan kā tehniskā specifikācija, gan kā izpētes dokumentācija Bevy spēļu dzinēja iespēju demonstrēšanai.

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 procedurāli ģenerēti sešstūra labirinti, kas katrā spēlē rada unikālu vizuālu un navigācijas izaicinājumu. Procedurālās ģenerēšanas sistēma nodrošina, ka:

- katrs labirints tiek unikāli ģenerēts „uzreiz“[4], nodrošinot „bezgalīgu“[5] daudzveidību;
- labirinta sarežģītību var dinamiski pielāgot spēlētājam progresējot;
- uzglabāšanas prasības tiek samazinātas līdz minimumam, ģenerējot labirintus reāllaikā.

Spēlētāju uzdevums ir pārvietoties pa šiem procedurāli ģenerētajiem labirintiem, lai sasniegtu katra līmeņa beigas. Turpinot progresēt, spēlētāji saskaras ar arvien sarežģītākiem labirintiem, kuros nepieciešama stratēģiskā domāšana, izpēte un papildu prasmju 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-komponenšu sistēmas (turpmāk tekstā – ECS) arhitektūru. Tas ietver stabilu spēles vides sistēmu izstrādi, stāvokļa pārvaldības mehānismus un efektīvu Bevy iebūvēto funkciju izmantošanu.

[4] Attiecas uz gandrīz tūlītēju labirintu ģenerēšanu, kas notiek milisekunžu laikā.

[5] Lai gan sistēma izmanto `u64` sēklas, kas ir galīgas, iespējamo labirinta konfigurāciju skaits ir ārkārtīgi liels, tādējādi praktiskiem mērķiem nodrošinot praktiski bezgalīgu labirintu skaitu.

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“ [2] un LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai“ standarta prasības [3].

Pārskats

Šis dokuments sniedz detalizētu programmatūras prasību specifikāciju spēlei „Maze Ascension“, aprakstot tās arhitektūru, galvenās komponentes un to mijiedarbību. Dokumentā ir apkopota informācija par spēles tehnisko implementāciju, izmantojot Bevy spēļu dzinēju un ECS arhitektūru.

Ievada sadaļa iepazīstina ar projekta pamatkonceptu, definējot spēles galvenos mērķus un uzdevumus. Šajā nodaļā tiek aprakstīta spēles ideja – procedurāli ģenerēts labirints ar sešstūrainu lauku, kas piedāvā unikālu spēles pieredzi katrā spēles reizē. Tiek skaidrota arī spēles vertikālās progresijas sistēma un papildspēju mehānikas, kas padara spēli izaicinošāku un interesantāku.

Programmatūras prasību specifikācijas sadaļa detalizētāk apraksta sistēmas funkcionālās prasības un arhitektūru. Izmantojot datu plūsmas diagrammas, tiek ilustrēta sistēmas moduļu mijiedarbība un datu plūsmas starp tiem. Šajā sadaļā tiek aprakstīti pieci galvenie moduļi: spēles stāvokļa pārvaldības modulis, labirinta pārvaldības modulis, spēlētāja modulis, stāvu pārvaldības modulis un papildspēju modulis. Katram modulim ir detalizēts funkciju apraksts, kas dokumentēts, izmantojot funkciju tabulas.

Programmatūras projektējuma sadaļa sniedz detalizētu tehnisko specifikāciju. Datu struktūru projektējuma apakšsadaļā tiek aprakstītas ECS arhitektūras komponentes, notikumi un resursi. Daļējā funkciju projektējuma apakšsadaļā tiek detalizēta šūnu pārvaldības sistēma un

citas būtiskas funkcijas. Saskarņu projektējuma apakšsadaļā tiek aprakstīta lietotāja saskarnes arhitektūra un implementācija.

Testēšanas dokumentācijas sadaļa aptver gan statisko, gan dinamisko testēšanu. Statiskās testēšanas apakšsadaļā tiek aprakstītas koda kvalitātes pārbaudes metodes un rīki. Dinamiskās testēšanas apakšsadaļā tiek detalizētāk izskatīta gan manuālā integrācijas testēšana, gan automatizēto testu implementācija, sniedzot konkrētus piemērus un rezultātus.

Projekta organizācijas sadaļa apraksta projekta pārvaldības aspektus. Kvalitātes nodrošināšanas apakšsadaļa detalizē izmantotās metodes un rīkus koda kvalitātes uzturēšanai. Konfigurācijas pārvaldības apakšsadaļa apraksta versiju kontroles sistēmu un izstrādes procesu, bet darbietilpības novērtējuma apakšsadaļa sniedz projekta resursu un laika patēriņa analīzi.

Secinājumu sadaļā tiek apkopoti galvenie projekta sasniegumi, izaicinājumi un to risinājumi. Šeit tiek izvērtēta izvēlēto tehnoloģiju un metožu efektivitāte, kā arī sniegti ieteikumi turpmākai projekta attīstībai.

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, paralēlām sistēmām.

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,[6] 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“[7] un itch.io platformas,[8] kas ir populāra neatkarīgo spēļu platforma, kas ļauj viegli piekļūt un izplatīt spēles visā pasaulē.

1.4. Darījumprasības

Sistēmas izstrādē tiek izvirzītas sekojošas darījumprasības, kas nodrošinās kvalitatīvu lietotāja pieredzi:

- Līmeņu pārvaldība: Sistēma nodrošinās automātisku spēles līmeņu pārvaldību un vienmērīgu pāreju starp tiem, veidojot pakāpenisku grūtības pieaugumu.

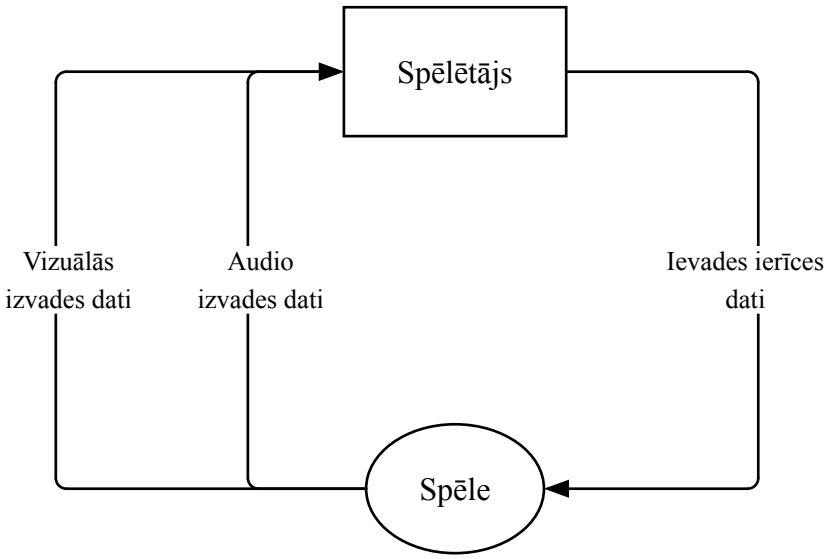
[6] https://github.com/resources/articles/devops/ci-cd
[7] https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases
[8] https://itch.io/

- Līmeņu ģenerēšana: Sistēma nodrošinās procedurālu līmeņu ģenerēšanu, radot unikālus sešstūrainus labirintus katram spēles stāvam, garantējot, ka katrs labirints ir pilnībā izejams.
- Tieša piekļuve: Spēle būs pieejama bez lietotāja konta izveides vai autentifikācijas, nodrošinot tūlītēju piekļuvi spēles saturam.
- Platformu atbalsts: Sistēma tiks izstrādāta ar daudzplatformu atbalstu, ietverot Linux, macOS, Windows un WebAssembly platformas.
- Kopienas integrācija: Izmantojot itch.io[8] platformu, tiks nodrošināta spēlētāju atsauksmju apkopošana un kopienas atbalsts.
- Nepārtraukta izstrāde: Izmantojot CI/CD risinājumus, tiks nodrošināta regulāra spēles atjaunināšana un uzturēšana.

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 lietotāju saistītās datu plūsmas ir attēlotas sistēmas nultā līmeņa DPD (sk. 1.1. att.).



1.1. att. **0. līmeņa DPD**

1.6. Vispārējie ierobežojumi

1) Izstrādes vides un tehnoloģijas ierobežojumi:
   a) Bevy dzinēja tehniskie ierobežojumi:

i) ECS arhitektūras specifika un tās ierobežojumi datu organizācijā;

ii) „Render Graph“[9] sistēmas ierobežojumi grafisko elementu attēlošanā;

iii) atkarība no „wgpu“[10] grafikas bibliotēkas iespējām.

b) Rust programmēšanas valodas ierobežojumi:

i) stingra atmiņas pārvaldība (angl. memory management) un īpašumtiesību (angl. ownership) sistēma;

ii) kompilācijas laika pārbaudes un to ietekme uz izstrādes procesu;

iii) WebAssembly kompilācijas specifika.

2) Platformu atbalsta ierobežojumi:

a) Nepieciešamība nodrošināt savietojamību ar:

i) darbvirsmas platformām (Linux, macOS, Windows);

ii) tīmekļa pārlūkprogrammām caur WebAssembly.

b) Platformu specifiskās prasības attiecībā uz:

i) grafikas renderēšanu;

ii) ievades apstrādi;

iii) veiktspējas optimizāciju.

Dokumentācijas izstrādei ir izmantots Typst rīks, kas nodrošina efektīvu darbu ar tehnisko dokumentāciju, ieskaitot matemātiskas formulas, diagrammas un koda fragmentus [4].

1.7. Pieņēmumi un atkarības

- Tehniskie pieņēmumi:
  - Spēlētāja ierīcei jāatbilst minimālajām aparatūras prasībām, lai varētu palaist uz Bevy spēles dzinēja balstītas spēles;
  - ierīcei jāatbalsta WebGL2[11] lai nodrošinātu pareizu atveidošanu [5];
  - tīmekļa spēļu spēlēšanai (WebAssembly versija) pārlūkprogrammai jābūt mūsdienīgai un saderīgai ar WebAssembly;

[9] https://docs.rs/bevy_render/latest/bevy_render/render_graph/struct.RenderGraph.htmlhttps://docs.rs/bevy_render/latest/bevy_render/render_graph/struct.RenderGraph.html

[10] https://wgpu.rs/

[11] https://registry.khronos.org/webgl/specs/latest/2.0/

  - ekrāna izšķirtspējai jābūt vismaz 800×600 pikseļi.
- Veiktspējas atkarība:
  - Spēle ir atkarīga no Bevy spēles dzinēja (0.15).
- 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[8] 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 platformas.
  - Spēlētājiem ir ievadierīces (tastatūra/pele), ar kurām kontrolēt spēli.

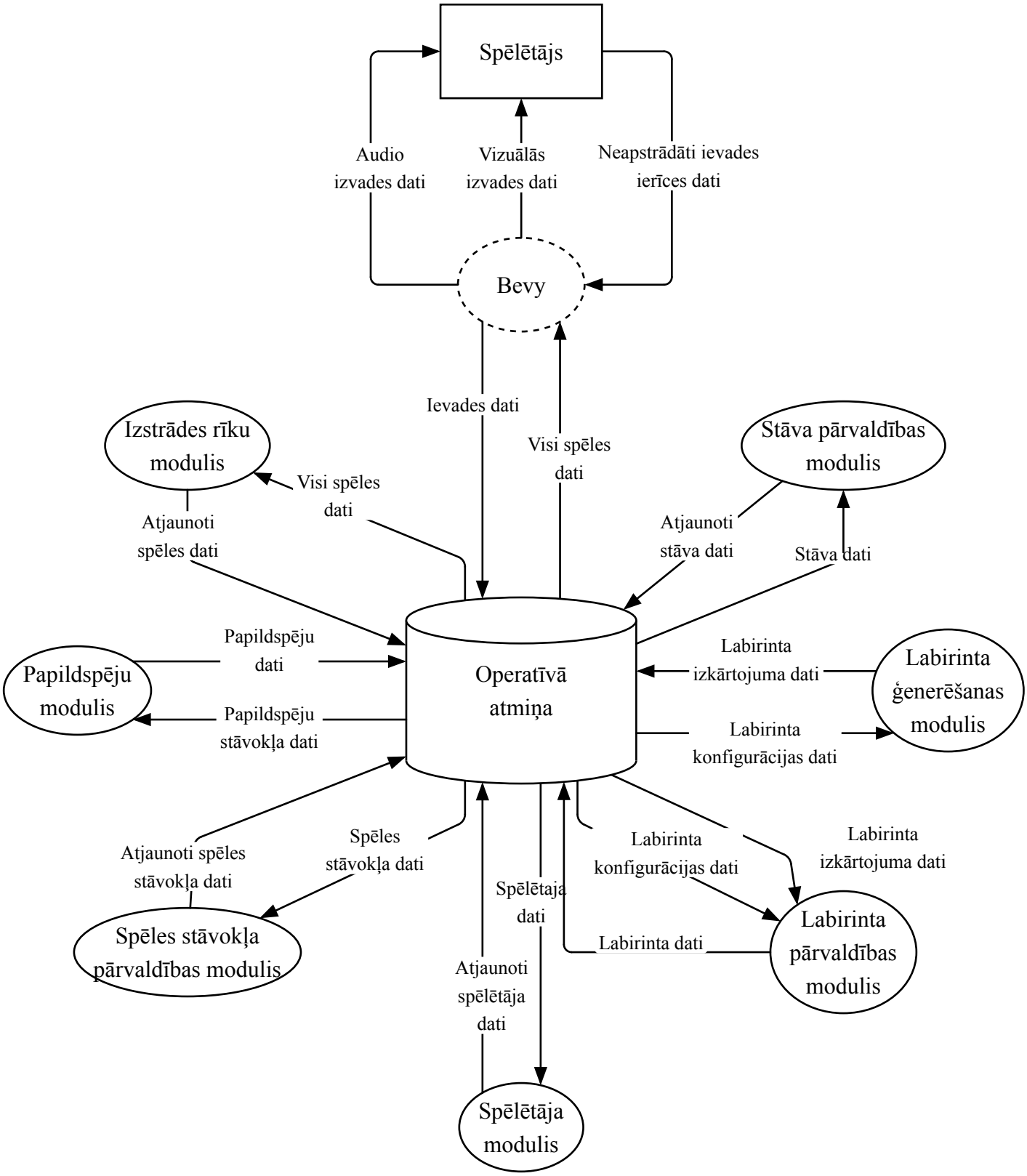
2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

2.1. Funkcionālās prasības

1. līmeņa datu plūsmas diagramma (sk. 2.1. att.) ilustrē galvenos procesus spēles „Maze Ascension“ sistēmā. Diagrammā attēloti seši galvenie procesi, viens izstrādes process un viens ārējs (bibliotēkas) process(-i): stāva pārvaldības modulis, labirinta ģenerēšanas un pārvaldības moduļi, spēlētāja modulis, spēles stāvokļa pārvalības modulis, papildspēju modulis un izstrādes rīku modulis. Šie procesi mijiedarbojas ar vienu datu krātuvi – operatīvo atmiņu (RAM) – un vienu ārējo lietotāju – spēlētājs.

Bevy spēļu dzinējs diagrammā ir attēlots kā ārējs process vairāku iemeslu dēļ. Pirmkārt, Bevy nodrošina pamata infrastruktūru spēles darbībai, ieskaitot ievades apstrādi, renderēšanu un audio atskaņošanu. Tā rezultātā visa lietotāja mijiedarbība ar spēli (tastatūras, peles ievade) vispirms tiek apstrādāta caur Bevy sistēmām, pirms tā nonāk līdz spēles specifiskajiem moduļiem.

Operatīvā atmiņa (RAM) ir vienīgā datu krātuve diagrammā, jo Bevy ECS arhitektūra balstās uz komponenšu datiem, kas tiek glabāti operatīvajā atmiņā un spēles stāvoklis netiek pastāvīgi saglabāts diskā.



2.1. att. **1. līmeņa DPD**

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.

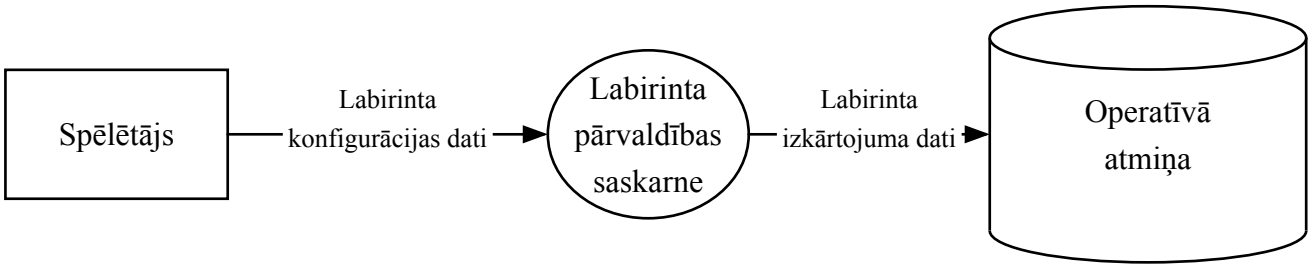
2.1. tabula **Funkciju sadalījums pa moduļiem**

Modulis	**Funkcija**	**Identifikators**
Izstrādes rīku modulis	Labirinta pārvaldības saskarne	IRMF01
Stāva pārvaldības modulis	Stāva kustība	SPMF01
	Stāvu pārejas apstrāde	SPMF02
Labirinta ģenerēšanas modulis	Labirinta būvētājs	LGMF01
Labirinta pārvaldības modulis	Labirinta ielāde	LPMF01
	Labirinta pārlāde	LPMF02
Spēlētāja modulis	Spēlētāja ielāde	SPMF01
	Spēlētāja ievades apstrāde	SPMF02
	Spēlētāja kustība	SPMF03
	Spēlētāja pāreja	SPMF04
Spēles stāvokļa pārvaldības modulis	Spēles sākšana	SSPMF01
	Atgriešanās uz sākumekrānu	SSPMF02
	Attēlot sākumekrānu	SSPMF03
Papildspēju modulis	Spēlētāja ievades apstrāde	PSMF01
	Ceļa rādīšanas papildspēja	PSMF02
	Sienu pārlēkšanas papildspēja	PSMF03

2.1.2. Izstrādes rīku modulis

Dotais modulis ir izstrādes rīks, kas paredzēts lietotāja saskarnes elementu attēlošanai un apstrādei, lai konfigurētu labirinta parametrus. Šis modulis, izmantojot „bevy_egui“[6] un „inspector-egui“[7] bibliotēkas, izveido logu „Maze Controls“ (labirinta vadības elementi), kurā tiek parādītas dažādas konfigurācijas opcijas, piemēram, sēkla, rādiuss, augstums, labirinta izmērs, orientācija un sākuma/galapunkta pozīcijas (sk 2.2. tab.). Lietotāji var mijiedarboties ar šiem vadības elementiem, lai mainītu labirinta izkārtojumu un izskatu. Modulis pārbauda, vai konfigurācijā nav notikušas izmaiņas, un izraisa attiecīgus notikumus, lai atjauninātu labirintu un spēlētāja pozīciju, kad notiek izmaiņas.

Svarīgi atzīmēt, ka šis modulis ir paredzēts lietošanai spēles izstrādes procesā. Laidiena versijās šī lietotāja saskarne nebūs pieejama, nodrošinot, ka gala lietotāji nevar piekļūt šīm uzlabotajām konfigurācijas opcijām.



2.2. att. **Izstrādes rīku moduļa 2. līmeņa DPD**

2.2. tabula **Labirinta pārvadības saskarne**

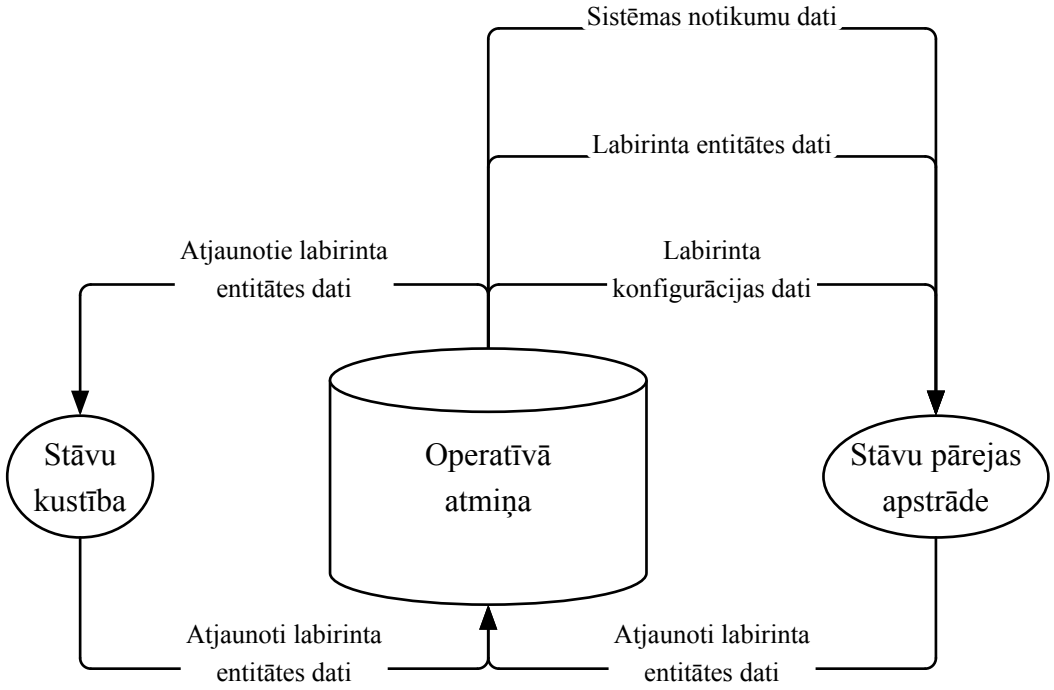
Funkcijas nosaukums
Labirinta pārvadības saskarne
Funkcijas identifikators
IRMF01
Apraksts
Apstrādā un izvada labirinta konfigurācijas vadības elementus lietotāja saskarnē.
Ievade
1) „`EguiContext`“ komponente;[12] 2) Labirinta konfigurācija un stāva komponentes saistībā ar pašreizējā stāva komponenti; 3) Globālais labirinta konfigurācijas resurss.
Apstrāde
1) Saņem „`EguiContext`“ komponenti no primārā loga. 2) Saņem labirinta konfigurāciju un stāvu komponentes no pašreizējā stāva. 3) Izveido jaunu „Maze Controls“ logu, izmantojot „egui“. 4) Ja globālais labirinta konfigurācijas resurss ir pieejams: a) Parāda galveno virsrakstu „Maze Configuration“. b) Pievieno vadības elementus: • Sēklai; • Rādiusam; • Augstumam; • Šešstūra izmēram; • Orientācijai; • Sākuma pozīcijai; • Beigu pozīcijai. c) Apstrādā izmaiņas un atjaunina labirintu un spēlētāju, ja radušās izmaiņas.

[12]https://docs.rs/bevy_egui/latest/bevy_egui/

Izvade
1) Atjaunināta labirinta konfigurācijas struktūra. 2) Atjaunināts labirints, ja radušās izmaiņas. 3) Atjaunināts spēlētāja novietojums, ja radušās izmaiņas.

2.1.3. Stāvu pārvaldības modulis

Stāvu pārvaldības modulis nodrošina vertikālo pārvietošanos starp spēles līmeņiem. Modulis sastāv no divām galvenajām funkcijām (sk. 2.3. att.): stāvu kustības (sk. 2.3. tab.) un stāvu pārejas apstrādes (sk. 2.4. tab.). Stāvu kustības sistēma nodrošina plūstošu vertikālo pārvietošanos starp līmeņiem, savukārt pārejas apstrādes sistēma koordinē pārejas starp pašreizējo un nākamo stāvu, reaģējot uz „TransitionFloor“ notikumu (sk. 3.5. tab.).



2.3. att. **Stāvu pārvaldības moduļa 2. līmeņa DPD**

2.3. tabula **Stāvu kustība**

Funkcijas nosaukums
Stāvu kustība
Funkcijas identifikators
SPMF01
Apraksts
Nodrošina plūstošu pāreju starp stāviem, pārvietojot tos vertikāli.

Ievade
1) Labirinta entitātes ar galamērķa Y pozīcijām.
Apstrāde
1) Aprēķina kustības attālumus. 2) Pārvieto stāvus uz galamērķi ar noteiktu ātrumu. 3) Atjaunina stāvu pozīcijas datus. 4) Noņem mērķa komponentes pēc sasniegšanas.
Izvade
1) Atjaunināto stāva pozīcijas.

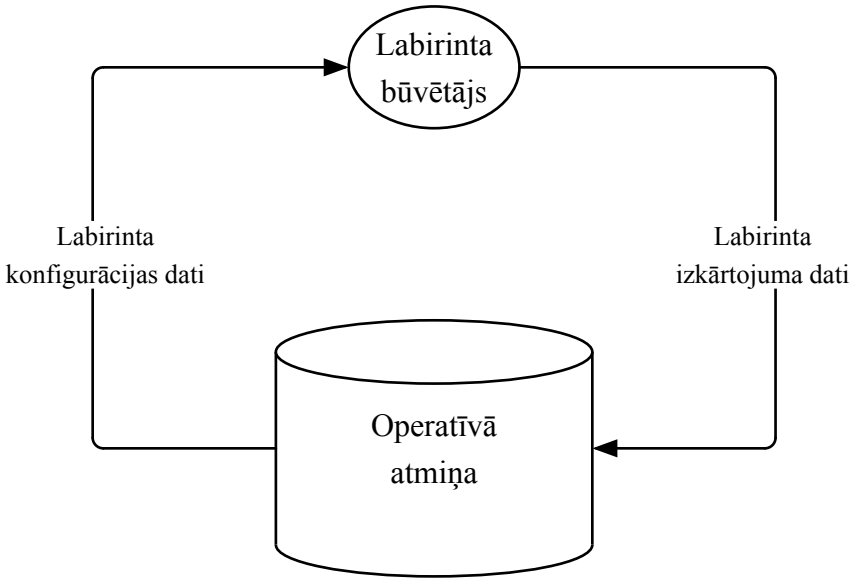
2.4. tabula **Stāvu pārejas apstrāde**

Funkcijas nosaukums
Stāvu pārejas apstrāde
Funkcijas identifikators
SPMF02
Apraksts
Apstrādā stāvu pārejas notikumus un koordinē pārejas starp stāviem.
Ievade
1) Pārejas notikums. 2) Stāvas entitātes. 3) Pašreizējais stāvs.
Apstrāde
1) Pārbauda vai ir aktīva pāreja. a) Ja ir, iziet no sistēmas un nedara neko. 2) Aprēķina galamērķu pozīcijas. 3) Pievieno mērķa komponentes stāvu entitātēm. 4) Atjauno stāvu statusus: a) Noņem pašreizējā stāva komponenti no pašreizējā stāva entitātes. b) Pievieno pašreizējā stāva komponenti nākamā stāva entitātei.
Izvade
1) Atjaunināts stāvs.

2.1.4. Labirinta ģenerēšanas modulis

Moduļa funkcionalitāte ir izmantota sešstūraina labirinta ģenerēšanai, balstoties uz „Hexagonal Grids“ rakstu [8], kas jau ir kļuvis par *de facto* standartu sešstūrainu režģu matemātikas un algoritmu implementācijai. Moduļa funkciju datu plūsmas ir parādītas 2. līmeņa datu plūsmas diagrammā (sk. 2.4. att.). Labirinta būvēšanas funkcija ir aprakstītas atsevišķā tabulā (sk. 2.5. tab.).

Modularitātes un atkārtotas lietojamības apsvērumu dēļ, labirinta ģenerēšanas funkcionalitāte ir izveidota kā ārēja bibliotēka „hexlib“.[13] 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.



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

2.5. tabula **Labirinta būvētājs**

Funkcijas nosaukums
Labirinta būvētājs
Funkcijas identifikators
LGMF01

[13] https://crates.io/crates/hexlab

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 sākuma pozīciju, 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

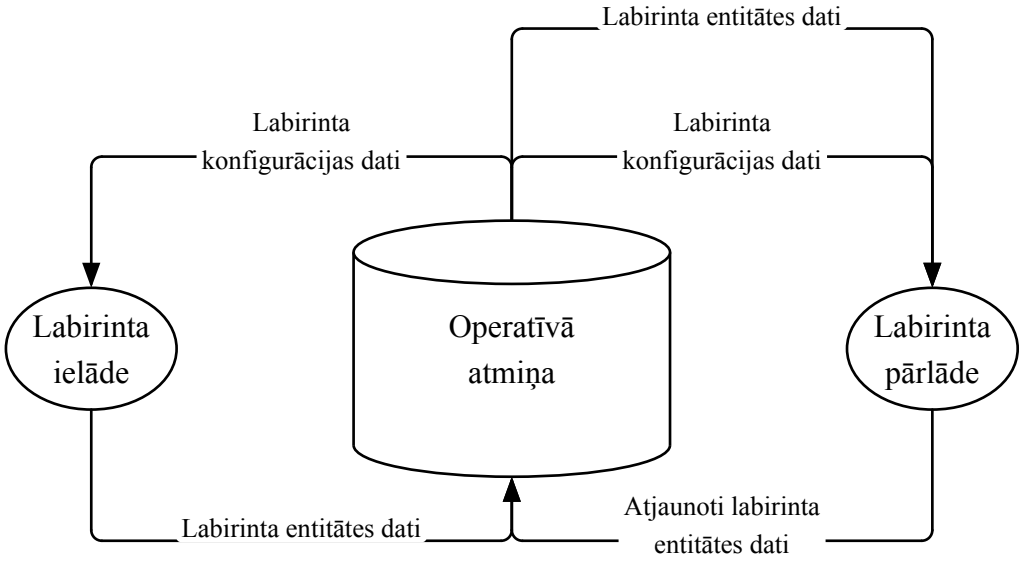
1) Lai izveidotu labirintu, ir jānorāda rādiuss.
2) Sākuma pozīcija ir ārpus labirinta robežām.
3) Neizdevās izveidot labirintu.

2.1.5. Labirinta pārvaldības modulis

Labirinta pārvaldības modulis ir atbildīgs par labirintu ģenerēšanu un pārvaldību katrā spēles stāvā. Moduļa funkciju datu plūsmas ir attēlotas 2. līmeņa datu plūsmas diagrammā (sk. 2.5. att.).

Modulis nodrošina divas galvenās funkcijas: labirinta izveidi (sk. 2.6. tab.) un labirinta atjaunošanu (sk. 2.7. tab.). Labirinta izveides funkcija tiek izsaukta, kad nepieciešams izveidot jaunu stāvu, pārbaudot, vai šāds stāvs jau neeksistē. Funkcija ģenerē jaunu labirintu, izmantojot norādīto konfigurāciju, un izvieto to atbilstošā augstumā spēles pasaulē.

Labirinta atjaunošanas funkcija ļauj pārģenerēt esošā stāva labirintu, saglabājot to pašu stāva numuru un pozīciju telpā nemainot entitātes identifikatoru.



2.5. att. **Labirinta pārvaldības moduļa 2. līmeņa DPD**

2.6. tabula **Labirinta ielāde**

Funkcijas nosaukums
Labirinta ielāde
Funkcijas identifikators
LPMF01
Apraksts
Izveido jaunu labirinta stāvu spēles pasaulē.
Ievade
1) Labirinta konfigurācija.

2) Globālā konfigurācija.
Apstrāde
1) Pārbauda, vai labirints šim stāvam jau eksistē. a) Ja eksistē, parāda 1. paziņojumu un iziet no sistēmas. 2) Ģenerē jaunu labirintu balstoties uz doto konfigurāciju. 3) Aprēķina vertikālo nobīdi jaunajam stāvam. 4) Izveido jaunu entitāti, kas pārstāv labirinta stāvu, pievienojot tam atbilstošās komponente. 5) Atkarībā no tā, vai tas ir pašreizējais stāvs, pievieno pašreizējā stāva komponenti. 6) Izveido jaunas entitātes, kas pārstāv labirinta šūnas, kā bērnu elementus labirinta entitātei. 7) Katrai labirinta šūnai atbilstoši labirinta konfigurācijai, izveido sienas bērnu entitātes.
Izvade
1) Labirinta entitāte.
Paziņojumi
1) „Stāvs x jau eksistē.“

2.7. tabula **Labirinta pārlāde**

Funkcijas nosaukums
Labirinta pārlāde
Funkcijas identifikators
LPMF02
Apraksts
Izveido jaunu labirinta stāvu spēles pasaulē.
Ievade
1) Stāva numurs, kuru atjaunot. 2) Labirinta konfigurācija. 3) Globālā konfigurācija.
Apstrāde
1) Pārbauda, vai labirints doto stāvu eksistē. a) Ja neeksistē, parāda 1. paziņojumu un iziet no sistēmas. 2) Ģenerē jaunu labirintu balstoties uz doto konfigurāciju.

3) Izdzēš visus labirinta entitātes pēcnācēju entitātes. 4) Izveido jaunas entitātes, kas pārstāv labirinta šūnas, kā bērnu elementus labirinta entitātei. 5) Katrai labirinta šūnai atbilstoši labirinta konfigurācijai, izveido sienas bērnu entitātes.
Izvade
1) Labirinta entitāte.
Paziņojumi
1) „Neizdevās atrast labirinta stāvu x.“

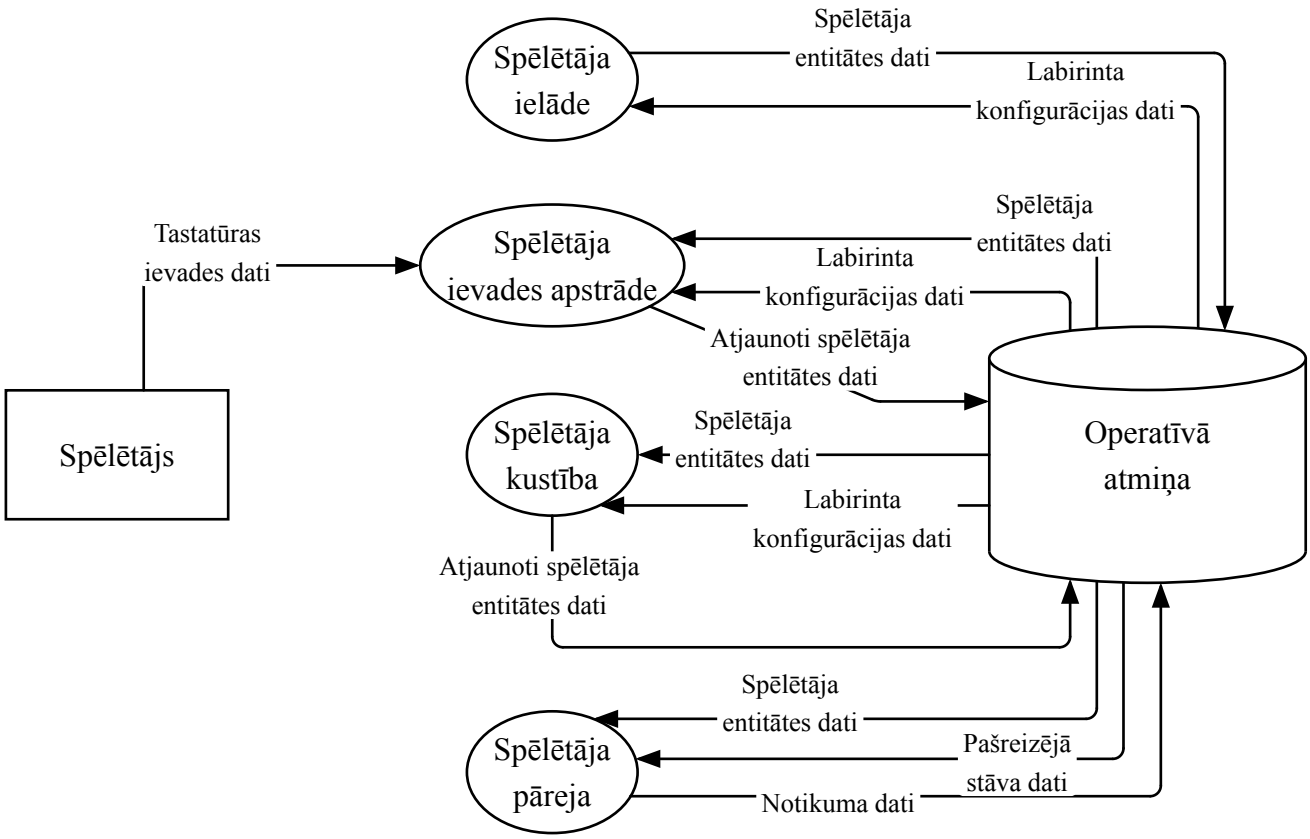
2.1.6. Spēlētāja modulis

Spēlētāja modulis ir atbildīgs par spēlētāja entītijas pārvaldību, kas ietver tās izveidi, kustību apstrādi un mijiedarbību ar spēles vidi. Moduļa datu plūsma ir attēlota 2. līmeņa datu plūsmas diagrammā (sk. 2.6. att.), kas parāda četras galvenās funkcijas un to mijiedarbību ar datu glabātuvi.

Spēlētāja kustība tiek realizēta divās daļās: ievades apstrāde (2.9. tab.) un kustības izpilde (2.10. tab.). Ievades apstrādes funkcija pārbauda tastatūras ievadi un, ņemot vērā labirinta sienu izvietojumu, nosaka nākamo kustības mērķi. Kustības izpildes funkcija nodrošina plūstošu pārvietošanos uz mērķa pozīciju, izmantojot interpolāciju[14] starp pašreizējo un mērķa pozīciju.

Stāvu pārejas apstrāde nepārtraukti uzrauga spēlētāja pozīciju attiecībā pret stāva izeju un sākumu. Kad spēlētājs sasniedz kādu no šiem punktiem, funkcija izsauc atbilstošu pārejas notikumu.

[14]Matemātiska metode, kas aprēķina starpvērtības starp diviem zināmiem punktiem.



2.6. att. **Spēlētāja moduļa 2. līmeņa DPD**

2.8. tabula **Spēlētāja ielāde**

Funkcijas nosaukums
Spēlētāja ielāde
Funkcijas identifikators
SPMF01
Apraksts
Izveido spēlētāja entitāti ar visām nepieciešamajiem komponentēm.
Ievade
1) Labirinta konfigurācija. 2) Globālā konfigurācija.
Apstrāde
1) Iegūst sākuma pozīciju no labirinta konfigurācijas. 2) Izveido spēlētāja entitāti ar pašreizējās pozīcijas komponenti.
Izvade
1) Spēlētāja entitāte.

2.9. tabula **Spēlētāja ievades apstrāde**

Funkcijas nosaukums
Spēlētāja ievades apstrāde
Funkcijas identifikators
SPMF02
Apraksts
Apstrādā spēlētāja tastatūras ievadi un aprēķina nākamo kustības šūnu.
Ievade
1) Tastatūras ievade. 2) Pašreizējā pozīcija. 3) Labirinta konfigurācija.
Apstrāde
1) Pārbauda vai ir aktīva kustība. a) Ja ir, iziet no sistēmas un nedara neko. 2) Nosaka kustības virzienu no ievades. 3) Pārbauda sienu šķēršļus. a) Ja kustības virzienā ir siena, iziet no sistēmas un nedara neko. 4) Aprēķina nākamo pozīciju.
Izvade
1) Kustības mērķa šūnas pozīcija.

2.10. tabula **Spēlētāja kustība**

Funkcijas nosaukums
Spēlētāja kustība
Funkcijas identifikators
SPMF03
Apraksts
Atjaunina spēlētāja pozīciju, veicot plūstošu pārvietošanos uz mērķa šūnas pozīciju.
Ievade
1) Kustības mērķis. 2) Kustības ātrums. 3) Pašreizējā pozīcija.

4) Labirinta konfigurācija.
Apstrāde
1) Aprēķina mērķa pozīciju pasaules koordinātēs. 2) Pārvieto spēlētāju uz mērķi ar noteiktu ātrumu. 3) Atjaunina pozīcijas datus.
Izvade
1) Atjaunināta pozīcija. 2) Transformācijas dati.

2.11. tabula **Stāvu pāreja**

Funkcijas nosaukums
Stāvu pāreja
Funkcijas identifikators
SPMF04
Apraksts
Pārbauda vai spēlētājs ir sasniedzis stāva izeju vai sākumu un inicializē vertikālo pāreju.
Ievade
1) Pašreizējā pozīcija. 2) Pašreizējais stāvs. 3) Labirinta konfigurācija.
Apstrāde
1) Pārbauda vai pozīcija sakrīt ar izeju. a) Ja sakrīt, izsauc pacelšanās notikumu uz iziet no sistēmas. 2) Pārbauda vai pozīcija sakrīt ar sākumu. a) Ja sakrīt, pārbauda vai iespējama nolaišanās. i) Ja ir iespējama, izsauc nolaišanās notikumu uz iziet no sistēmas.
Izvade
1) Stāva pārejas notikums.

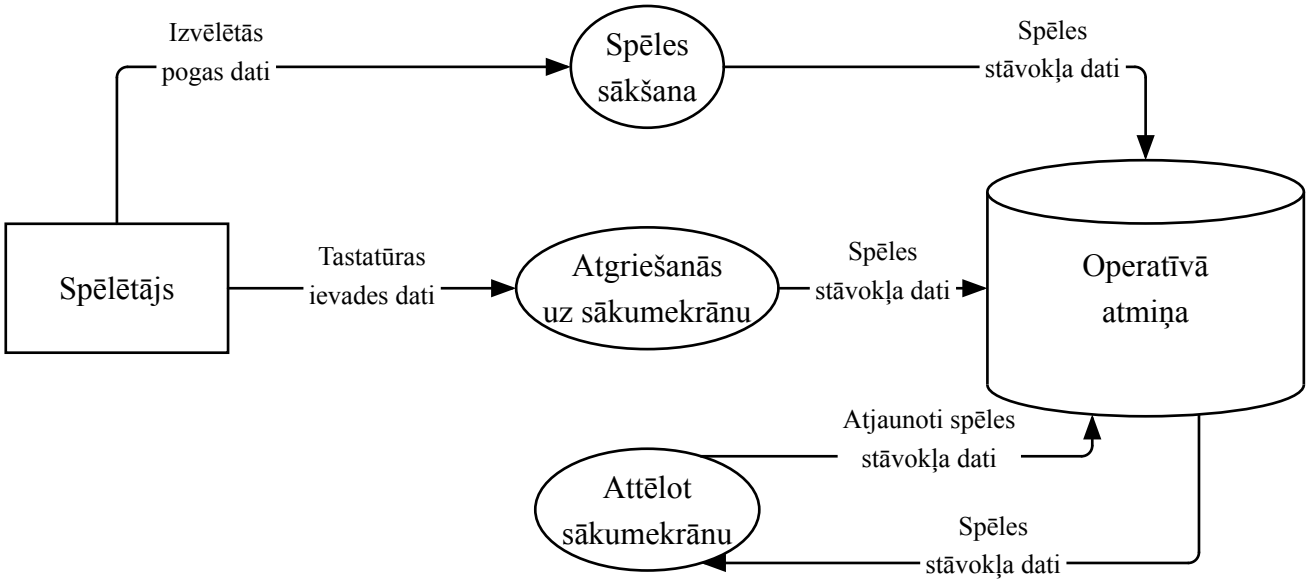
2.1.7. Spēles stāvokļa pārvaldības modulis

Spēles stāvokļa pārvaldības modulis nodrošina spēles dažādu stāvokļu pārvaldību un pārejas starp tiem. Modulis sastāv no trim galvenajām funkcijām: spēles sākšana (2.12. tab.),

atgriešanās uz sākumekrānu (2.13. tab.) un sākumekrāna attēlošanas (2.14. tab.). Katra no šīm funkcijām apstrādā specifiskus lietotāja ievades datus un atbilstoši atjaunina spēles stāvokli.

Moduļa 2. līmeņa DPD diagramma (sk. 2.7. att.) parāda, ka lietotājs mijiedarbojas ar sistēmu izmantojot divus galvenos ievades veidus: pogu izvēli sākumekrānā un „Escape“ taustiņa nospiešanu spēles laikā.

Spēles sākšanas funkcija inicializē nepieciešamos resursus un sistēmas, kad lietotājs izvēlas sākt jaunu spēli. Atgriešanās funkcija apstrādā lietotāja pieprasījumu pārtraukt aktīvo spēli un atgriežas uz sākumekrānu.



2.7. att. **Spēles stāvokļa pārvaldības moduļa 2. līmeņa DPD**

2.12. tabula **Spēles sākšana**

Funkcijas nosaukums
Spēles sākšana
Funkcijas identifikators
SSPMF01
Apraksts
Sākt spēles līmeni, ielādējot labirintu, spēlētāju un mūziku.
Ievade
1) Ekrāna stāvoklis
Apstrāde

1) Pievieno sistēmas spēles sākumā: • Izveido līmeni; • Izveido spēlētāju; • Sistēmas tiek darbinātas secīgi.
Izvade
Izvades datu sistēmai nav.

2.13. tabula **Atgriešanās uz sākumekrānu**

Funkcijas nosaukums
Atgriešanās uz sākumekrānu
Funkcijas identifikators
SSPMF02
Apraksts
Apstrādāt atgriešanos uz titulekrānu no gameplay režīma.
Ievade
1) Ekrāna stāvoklis 2) Nospiestie tastatūra taustiņi
Apstrāde
1) Pārbauda, vai ir nospiests „Escape“ taustiņš. a) Ja nav, iziet no sistēmas un nedara neko. 2) Pārbauda, vai pašreizējais stāvoklis spēle ir aktīva. a) Ja nav, iziet no sistēmas un nedara neko. 3) Samaina ekrāna stāvokli uz sākumekrānu.
Izvade
Izvades datu sistēmai nav.

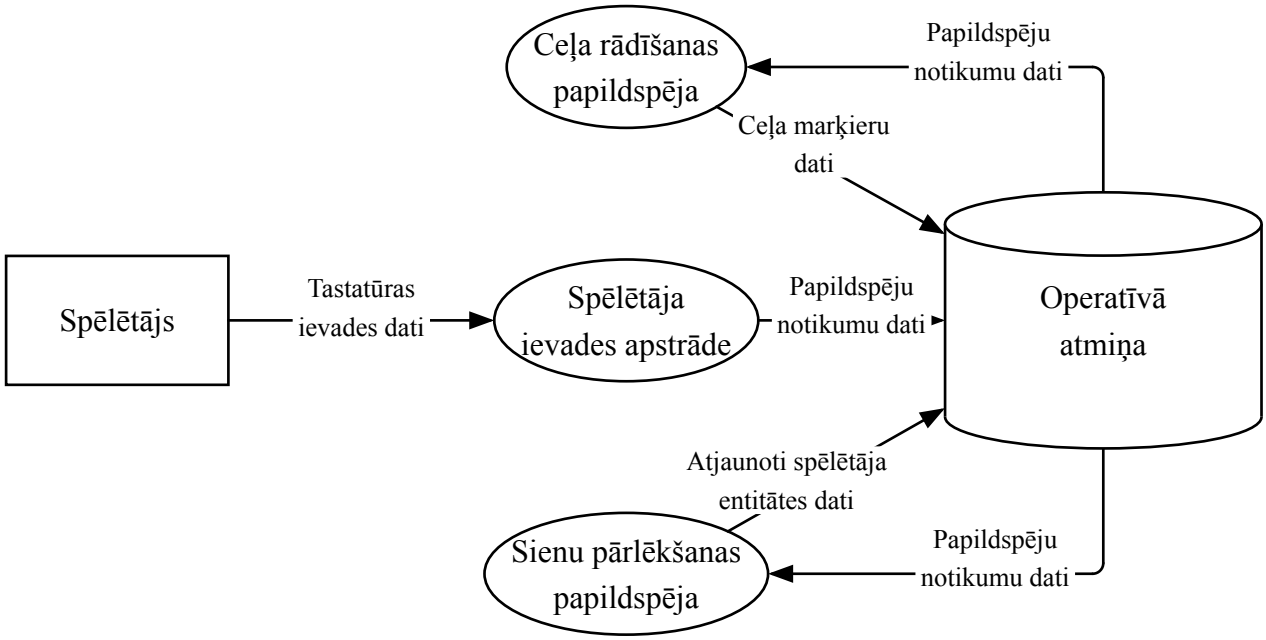
2.14. tabula **Attēlot sākumekrānu**

Funkcijas nosaukums
Attēlot sākumekrānu
Funkcijas identifikators
SSPMF03

Apraksts
Izveido un parāda nosaukuma ekrāna lietotāja saskarni ar interaktīvām pogām.
Ievade
1) Ekrāna stāvoklis
Apstrāde
1) Pievieno interaktīvas pogas: • pogu „Play“ ar pāreju uz spēli; • pogu „Exit“ (tikai platformām, kas nav WASM). 2) Pievieno novērotājus katrai pogai.
Izvade
Izvades datu sistēmai nav.

2.1.8. Papildspēju modulis

Papildspēju modulis nodrošina spēlētājam papildu iespējas labirinta izpētē, piedāvājot divas galvenās funkcijas: sienu pārlēkšanu un izejas ceļa parādīšanu (sk. 2.8. att.).



2.8. att. **Papildspēju moduļa 2. līmeņa DPD**

2.15. tabula **Spēlētāja ievades apstrāde**

Funkcijas nosaukums
Spēlētāja ievades apstrāde
Funkcijas identifikators

PSMF01
Apraksts
Apstrādā spēlētāja tastatūras ievadi un aktivizē attiecīgo papildspēju.
Ievade
1) Tastatūras ievade. 2) Globālā konfigurācija. 3) Papildspējas atdzišanas stāvoklis. 4) Atdzišanas laiks.
Apstrāde
1) Pārbauda lēkšanas papildspējas ievadi. a) Iegūst lēkšanas taustiņu no konfigurācijas. b) Ja taustiņš nospiests un nav atdzišanas periodā. i) Izsauc lēkšanas notikumu. 2) Pārbauda ceļa rādīšanas papildspējas ievadi. a) Iegūst ceļa rādīšanas taustiņu no konfigurācijas b) Ja taustiņš nospiests un nav atdzišanas periodā. i) Izsauc ceļa parādīšanas notikumu.
Izvade
1) Papilspējas notikums.

2.16. tabula **Ceļa rādīšanas papildspēja**

Funkcijas nosaukums
Ceļa rādīšanas papildspēja
Funkcijas identifikators
PSMF02
Apraksts
Rāda ceļu uz izeju noteiktu laiku periodu.
Ievade
1) Papilspējas notikums. 2) Pašreizējā spēlētāja pozīcija. 3) Labirinta konfigurācija. 4) Laiks.

Apstrāde
1) Aprēķina īsāko ceļu no pašreizējās pozīcijas līdz labirinta izejai. 2) Izveido ceļa marķieru entitātes. 3) Seko līdzi atlikušajam laikam. a) Ja laiks pārsniedz 10 sekundes, izdzēš ceļa marķierus.
Izvade
1) Ceļa vizualizācijas dati.

2.17. tabula **Sienu pārlēkšanas papildspēja**

Funkcijas nosaukums
Sienu pārlēkšanas papildspēja
Funkcijas identifikators
PSMF03
Apraksts
Ļauj pārlēkt pāri sienām.
Ievade
1) Lēciena pozīcijas dati 2) Papilspējas notikums. 3) Kustības mērķis. 4) Kustības ātrums. 5) Pašreizējā spēlētāja pozīcija. 6) Labirinta konfigurācija.
Apstrāde
1) Aprēķina mērķa pozīciju pasaules koordinātēs. 2) Pārbauda sienas esamību. 3) Pārvieto spēlētāju uz mērķi ar noteiktu ātrumu. 4) Atjaunina pozīcijas datus.
Izvade
1) Atjaunināta pozīcija. 2) Transformācijas dati.

2.2. Nefunkcionālās prasības

2.2.1. Veiktspējas prasības

Uz sistēmas veiktspēju ir sekojošas prasības:

- Jebkura izmēra labirintam jātiek uzģenerētam ātrāk kā 1 sekundē.
- Spēlei jāstartējas ātrāk par 3 sekundēm.
- Spēlei jādarbojas ar vismaz 60 kadriem sekundē.
- Spēlētāja kustībām jātiek apstrādātā bez manāmas aizkaves (< 16ms).

2.2.2. Uzticamība

Uz sistēmas uzticamību ir sekojošas prasības:

- Kļūdu apstrāde: spēlei jāapstrādā kļūdas graciozi, bez sistēmas atteicēm.

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

Spēlei jābūt saderīgai ar vairākām operētājsistēmām. Tas ietver Windows, Linux un macOS operētājsistēmu 64 bitu versiju atbalstu. Minimālās sistēmas prasības ir noteiktas, lai nodrošinātu plašu pieejamību, vienlaikus saglabājot veiktspēju:

- 4GB operatīvās atmiņas (RAM);
- Integrēta grafiskā karte;
- Divkodolu procesors.

3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

3.1. Datu struktūru projektējums

Spēle ir veidota, izmantojot Bevy spēles dzinēju, kas īstenu entitāšu-komponenšu sistēmu (ECS) arhitektūras modeli. Šis modelis sadala spēles loģiku trīs galvenajās daļās: entitātes jeb spēles objekti, komponentes jeb dati un sistēmas – loģika, kas darbojas ar entitātēm ar konkrētām komponentēm [1], [9], [10, nod. 14.7]. Šis modelis ļauj efektīvi apstrādāt datus un skaidri nodalīt atsevišķus pienākumus.

3.1.1. Komponentes

Komponentes Bevy ir vienkāršas datu struktūras, kuras var pievienot entitātēm. Tās nosaka spēles objektu īpašības un iespējas. Šajā spēlē komponentes tiek izmantotas, lai attēlotu dažādus spēles vienību aspektus, sākot ar to fiziskajām īpašībām un beidzot ar to vizuālo attēlojumu. Komponentes ir izstrādātas būt minimālām un mērķtiecīgām, ievērojot vienotas atbildības principus [11], [12]. Šīs komponentes var iedalīt trīs galvenajās grupās: ar stāvu/ līmeni, labirintu un spēlētāju saistītās komponentes, kā redzams 3.1. , 3.2. un 3.3. tabulās.

3.1.1.1. Stāva komponentes

Stāva komponentes pārvalda vertikālo progresu un kustību spēlē. Kā redzams 3.1. tabulā, šīs komponentes pārvalda stāvu numurus, pašreizējā stāva stāvokli un vertikālās kustības mehāniku.

3.1. tabula **Ar stāviem saistītās komponentes**

Komponente	Apraksts	Pielietojums
`Floor`	Stāva numurs	Identificē, kurai entitātei ir kurš stāvs.
`CurrentFloor`	Atzīmē pašreizējo stāvu	Identificē pašreizējo stāvu.
`FloorYTarget`	Stāva nākamā Y pozīcija	Identificē stāva Y koordināti, uz kuru tas jāpārvieto.

3.1.1.2. Labirinta komponentes

Labirinta struktūru pārvalda vairāki savstarpēji saistītas komponentes, kas ir atbildīgas par labirinta uzturēšanu (sk. 3.2. tab.).

3.2. tabula **Ar labirintiem saistītās komponentes**

Komponente	Apraksts	Pielietojums
`HexMaze`	Galvenais labirinta marķieris	Identificē labirinta entitāti un pieprasa nepieciešamās atkarības.
`Tile`	Apzīmē labirinta sešstūra šūnu	Identificē labirinta vietas, pa kurām var staigāt.
`Wall`	Apzīmē labirinta sienas	Identificē sadursmju robežas.
`MazeConfig`	Glabā labirinta parametrus	Konfigurē labirinta ģenerēšanu ar rādiusu, pozīcijām un izkārtojumu.
`Maze`	Glabā sešstūra labirinta datus	Glabā pilnu labirinta struktūru, izmantojot jaucējtabulu.
`Walls`	Apzīmē sienu konfigurāciju	Pārvalda sienas stāvokļus, izmantojot bitu karodziņus [13].

3.1.1.3. Spēlētāja komponentes

Spēlētāju kustību un mijiedarbību nodrošina specializētu komponenšu kopums. 3.3. tabulā ir redzamas sastāvdaļas, kas pārvalda ar spēlētāju saistītās funkcijas.

3.3. tabula **Ar spēlētāju saistītās komponentes**

Komponente	Apraksts	Pielietojums
`Player`	Apzīmē spēlētāja entitāti	Identificē spēlētāju un pieprasa nepieciešamās sastāvdaļas.
`CurrentPosition`	Glabā spēlētājs pozīciju	Nosaka pašreizējo atrašanās vietu labirintā.
`MovementSpeed`	Glabā kustības ātrumu	Nosaka spēlētāja pārvietošanās ātrumu.
`MovementTarget`	Glabā pārvietošanās mērķi	Pārvalda spēlētāju pārvietošanās virzienus.

3.1.2. Notikumi

Notikumi Bevy nodrošina paziņojumu apmaiņas mehānismu, kas nodrošina brīvu sasaisti starp sistēmām. Šī uz notikumiem balstītā arhitektūra ļauj dažādām sistēmas daļām spēles daļām sazināties bez tiešas atkarības. Notikumi ir īpaši noderīgi lietotāja ievades, fizikas mijiedarbības un spēles stāvokļa pāreju apstrādei [10, nod. 14.11]. Notikumi arī ir iedalīti trīs galvenajās grupās: ar labirintu, pāreju uz citu stāvu un ar spēlētāju saistīti notikumi, kas redzams 3.4. , 3.5. un 3.6. tabulās.

3.1.2.1. Labirintu notikumi

Labirinta notikumi pārvalda labirinta entitāšu dzīves ciklu spēlē. Kā redzams 3.4. tabulā, šie notikumi pārvalda labirinta izveidi un atjaunošanu.

3.4. tabula **Ar labirintiem saistīti notikumi**

Notikums	Apraksts	Pielietojums
`SpawnMaze`	Izveido jaunu labirintu	Inicializē labirintu ar norādīto grīdu un konfigurāciju.
`RespawnMaze`	Atjauno esošo labirintu	Atjauno labirintu.

3.1.2.2. Stāvu notikumi

Stāvu pārejas sistēma izmanto vienu uzskaitītu notikumu tipu (sk. 3.5. tab.), kas pārvalda vertikālo kustību starp stāviem.

3.5. tabula **Ar stāviem saistīti notikumi**

Notikums	Variants	Pielietojums
`TransitionFloor`	`Ascend`	Izraisa visu stāvu augšupejošu pāreju.
`TransitionFloor`	`Descend`	Izraisa visu stāvu lejupejošu pāreju.

3.1.2.3. Spēlētāju notikumi

Ar spēlētāju saistītie notikumi pārvalda spēlētāja entitātes dzīves ciklu (sk. 3.6. tab.). Līdzīgi kā labirintu notikumiem, šie apstrādā spēlētāja izveidošanu un atjaunošanu.

3.6. tabula **Ar spēlētaju saistīti notikumi**

Notikums	Apraksts	Pielietojums
`SpawnPlayer`	Izveido spēlētāja entitāti	Inicializē jaunu spēlētāju starta pozīcijā.
`RespawnPlayer`	Atjauno spēlētāju	Atiestata spēlētāju uz pašreizējā stāva sākuma pozīciju.

3.1.3. Resursi

Bevy resursi kalpo kā globāli stāvokļa konteineri, kuriem var piekļūt jebkura sistēma. Atšķirībā no komponentēm, kas ir piesaistīti konkrētām entitātēm, resursi nodrošina spēles mēroga datus un konfigurāciju. Tie ir īpaši noderīgi kopīgu stāvokļu un iestatījumu pārvaldībai, kas var ietekmēt vairākas sistēmas [10, nod. 14.6]. Spēle izmanto vienu resursu globālās konfigurācijas un stāvokļa pārvaldībai (sk. 3.7. tab.)

3.7. tabula **Globālie resursi**

Resurss	Apraksts	Pielietojums
`GlobalMazeConfig`	Labirinta vizuālie iestatījumi	Uzglabā globālos labirinta izskata parametrus.

Dotais resurss pārvalda labirinta vizuālo attēlojumu, ietverot tādus parametrus kā sešstūra lielums, sienu biezums un vertikālais augstums.

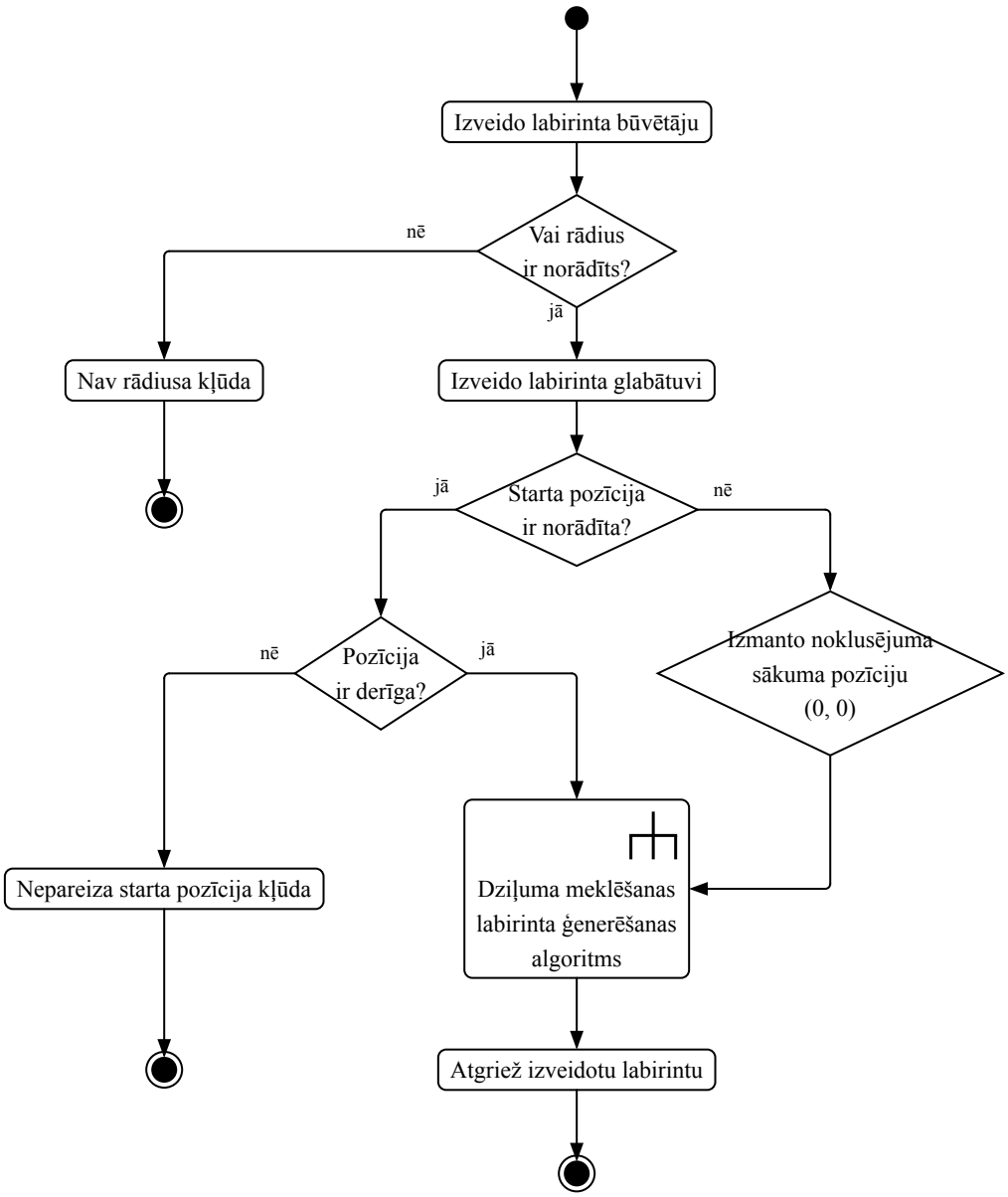
3.2. Daļējs funkciju projektējums

Labirinta ģenerēšanas process ir realizēts, izmantojot meklēšanas dziļumā algoritmu (angl. Depth-First Search, DFS)[15], kas ir viens no populārākajiem labirintu ģenerēšanas algoritmiem [14]. Labirinta ģenerēšanas process ir attēlots aktivitāšu diagrammā (sk. 3.1. att.), kas parāda algoritma galvenos soļus, sākot ar labirinta būvētāja inicializāciju un beidzot ar gatava labirinta atgriešanu.

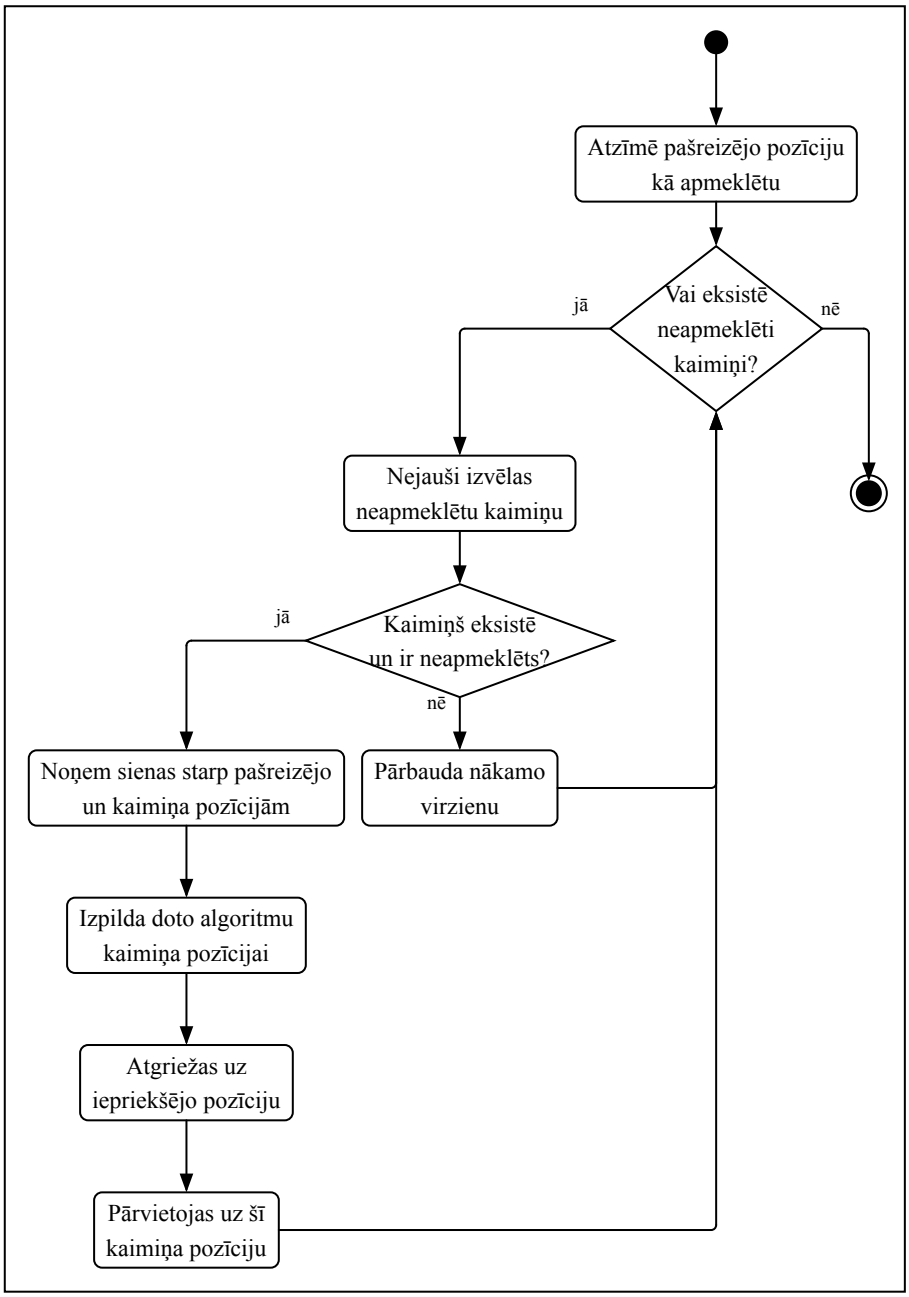
DFS algoritma implementācija (sk. 3.2. att.) izmanto rekursīvu pieeju, kur katrs rekursijas solis apstrādā vienu labirinta šūnu. Algoritms sākas ar sākotnējās pozīcijas atzīmēšanu kā apmeklētu un turpina ar nejaušu, neapmeklētu kaimiņu izvēli. Kad nejaušs kaimiņš ir izvēlēts, algoritms noņem sienu starp pašreizējo un izvēlēto šūnu, tad rekursīvi turpina procesu no jaunās pozīcijas. Šī pieeja nodrošina, ka izveidotais labirints ir pilnībā savienots un katrai šūnai var piekļūt no jebkuras citas šūnas.

Algoritma īpatnība ir tāda, ka tas veido labirintus ar zemu sazarošanās faktoru un gariem koridoriem, jo tas izpēta katru zaru pēc iespējas tālāk, pirms atgriežas un mēģina citu ceļu.

[15] https://en.wikipedia.org/wiki/Depth-first_search

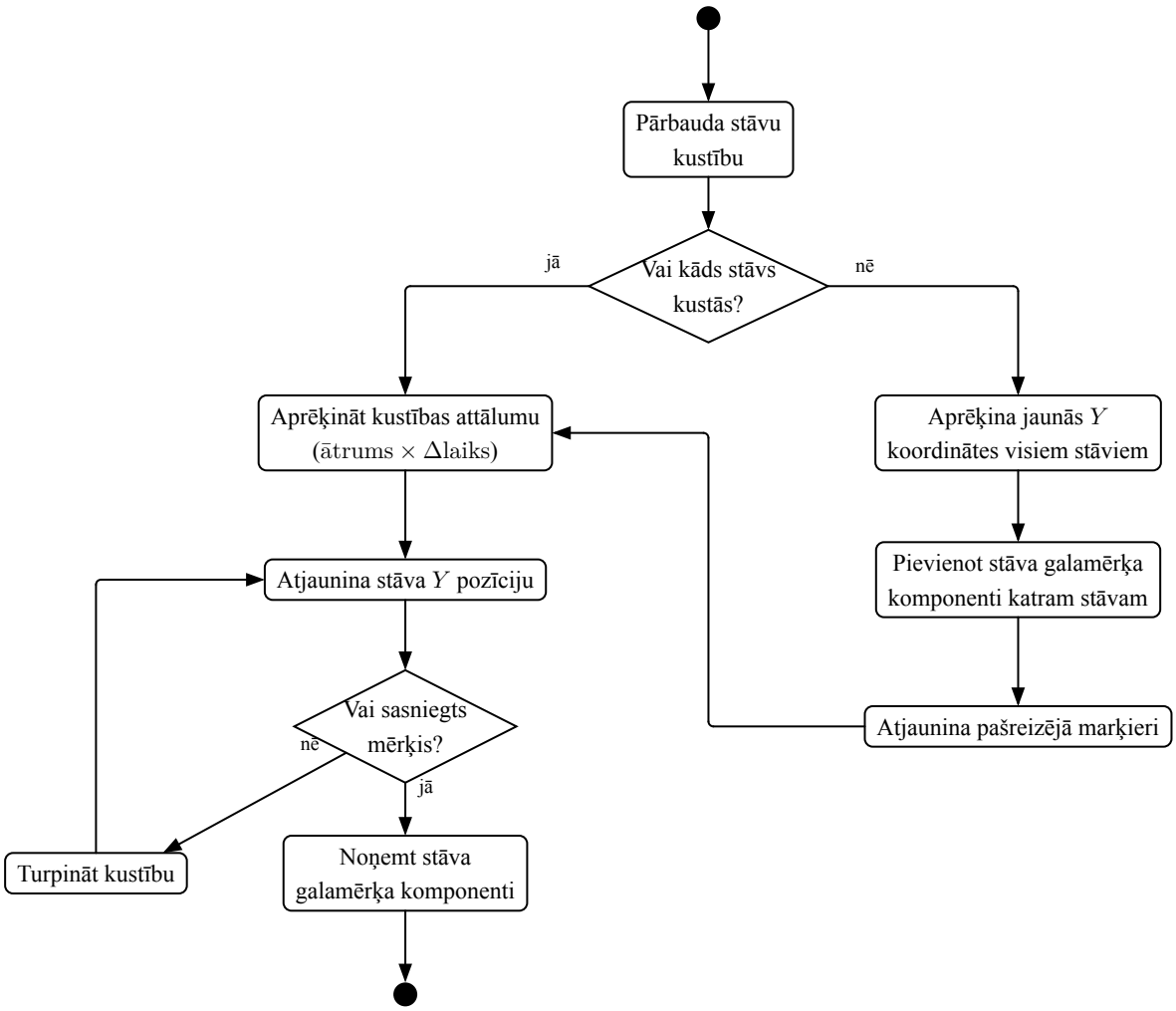


3.1. att. **Labirinta ģenerēšanas pārejas diagramma**



3.2. att. **Meklēšanas dziļumā labirinta ģenerēšanas algoritms (apakšaktivitāte)**

Stāva kustības aktivitāšu diagramma (sk. 3.3. att.) attēlo divus procesus stāvu pārvaldībai. Kreisajā pusē ir attēlota stāvu kustības loģika, kas sākas ar kustības stāvokļa pārbaudi. Ja kāds stāvs kustās, sistēma aprēķina kustības attālumu, balstoties uz ātrumu un laika deltu, un atjaunina stāva Y pozīciju. Šis process turpinās, līdz tiek sasniegts mērķis, pēc kā tiek noņemta stāva galamērķa komponente. Labajā pusē ir attēlota stāvu pārejas loģika, kas tiek izpildīta, kad neviens stāvs nekustās. Šī daļa aprēķina jaunās Y koordinātes visiem stāviem, pievieno tiem galamērķa komponentes un atjaunina pašreizējā stāva marķierus.



3.3. att. **Stāva kustības sistēma**

3.2.1. Plākšņu pārvaldības sistēma

Projekta sākotnējā plānošanas posmā tika apsvēra iespēja labirinta elementu pārvaldībai izmantot „bevy_ecs_tilemap“ bibliotēku.[16] Tomēr pēc rūpīgas izvērtēšanas tika secināts, ka tā neatbilst konkrētajam projekta lietojuma gadījumam sekojošu iemeslu dēļ:

1) Uz failiem balstīta plākšņu ielāde: „bevy_ecs_tilemap“ galvenokārt paļaujas uz plākšņu ielādi no ārējiem failiem. Šajā projektā ir nepieciešami dinamiski, procedurāli ģenerēti labirinti, tāpēc šī pieeja nav īsti piemērota.
2) Elastības ierobežojumi: bibliotēkas plākšņu datu struktūra nav viegli pielāgojama nepieciešamajai datu struktūrai, kurai ir nepieciešama sarežģītākām telpiskām attiecībām starp šūnām.

[16] https://crates.io/crates/bevy_ecs_tilemap

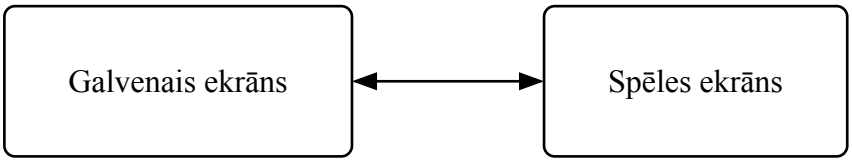
3) Prasības attiecībā uz sienu veidošanu: katrai sistēmas labirinta šūnai var būt 0-6 sienas, kas tiek ģenerētas nejauši. Šādu dinamisku sienu ģenerēšanas līmeni nav viegli sasniegt izmantojot „bevy_ecs_tilemap“.

Tā vietā, lai izmantotu „bevy_ecs_tilemap“, tika izlemts izstrādāt pielāgotu risinājumu, kas tieši integrējas ar labirinta ģenerēšanas algoritmu. Šī pieeja ļauj:

- vienkāršāku integrācija ar procedurālo labirintu ģenerēšanu;
- optimālāku veiktspēja projekta lietošanas gadījumam;
- lielāku kontroli pār labirinta vizuālo attēlojumu.

3.3. Saskarņu projektējums

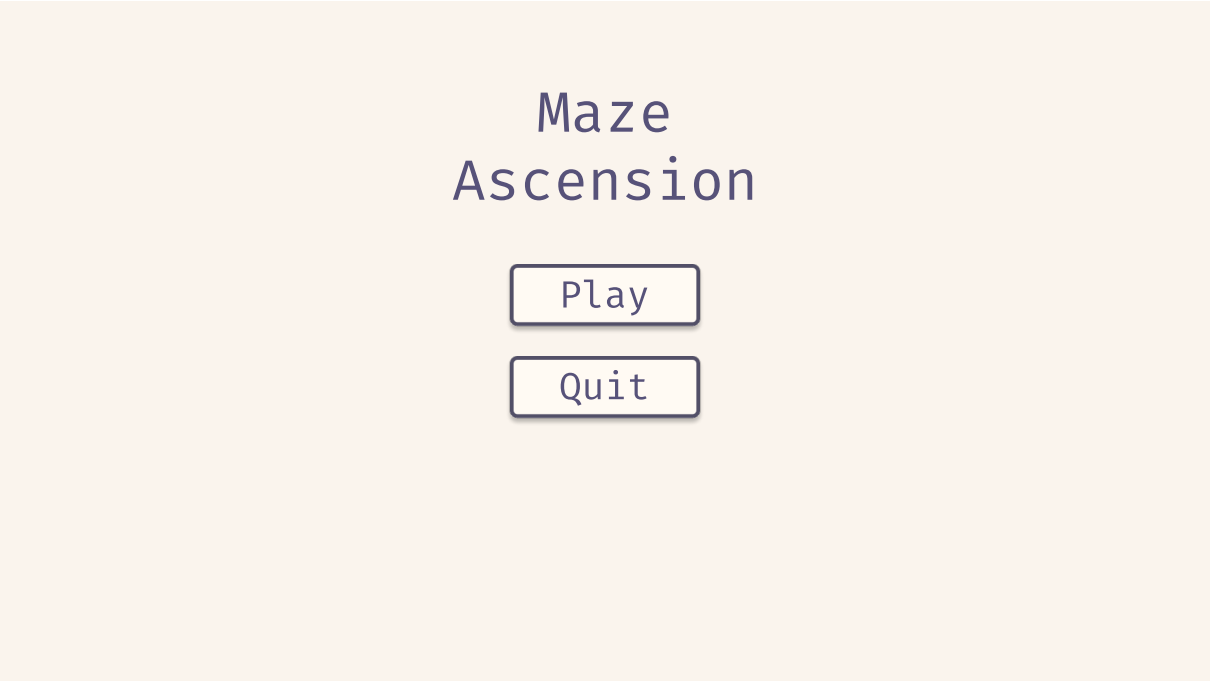
Spēles saskarņu projektējums ietver divus galvenos skatus (sk. 3.4. att.) – galveno izvēlni un spēles saskarni – un izstrādes rīkus.



3.4. att. **Ekrānskatu plūsmu diagramma**

3.3.1. Galvenā izvēlne

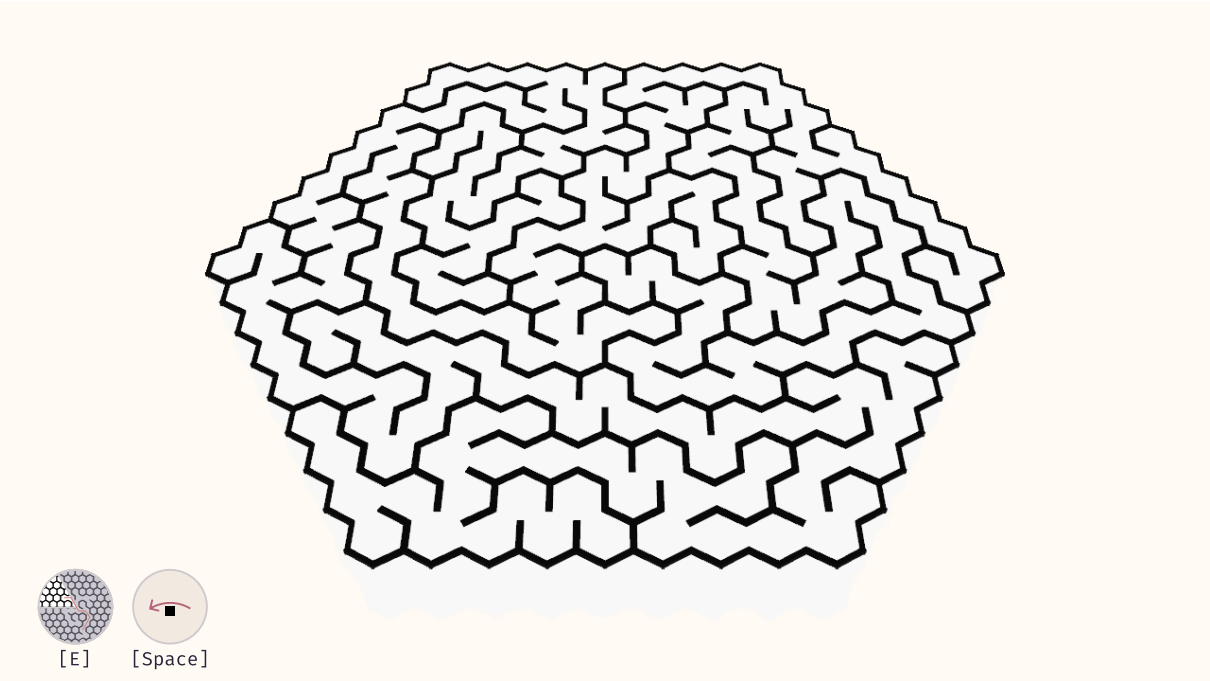
Galvenā izvēlne ir pirmais skats, ar ko saskaras lietotājs, uzsākot spēli (sk. 3.5. att.). Tā sastāv no spēles nosaukuma, „Play“ – sākt spēli pogas un „Quit“ – iziet pogas.



3.5. att. **Galvenās izvēlnes skats**

3.3.2. Spēles skats

Spēles skats apvieno pašu spēles pasauli ar minimālistisku lietotāja saskarni (sk. 3.6. att.). Centrālo daļu aizņem spēles pasaule ar sešstūra labirintu, kas veido spēles galveno interaktīvo elementu. Ekrāna kreisajā apakšējā stūrī ir izvietoti papildspēju statusa indikatori, kas sniedz spēlētājam vizuālu atgriezenisko saiti par pieejamajām spējām un to atjaunošanās laiku.



3.6. att. **Galvenās izvēlnes skats**

3.3.3. Izstrādes rīki

Izstrādes rīki, kas redzami 3.7. attēlā, ir implementēti izmantojot „bevy_egui“ bibliotēku[12]. Pirmais „Bevy Inspector Egui“ noklusētais skats [7], kas nodrošina detalizētu piekļuvi spēles entitāšu hierarhijai, komponenšu inspektoram un resursu pārvaldniekam. Otrs ir izvietots labirinta konfigurācijas panelis, kas ļauj kontrolēt labirinta izmēru, izkārtojumu un pielāgot ģenerēšanas parametrus,

3.7. att. **Izstrādes rīku saskarne ar labirinta konfigurācijas paneli**

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“ 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 (sk. 1. un 2. pielikumus) [15].

4.2. Dinamiskā testēšana

Lai novērtētu programmatūras uzvedību darbības laikā, tika veikta dinamiskā testēšana. Šī testēšanas pieeja apvieno gan manuālu testēšanu, izmantojot lietotāja saskarnes mijiedarbību, gan automatizētus testu komplektus, lai nodrošinātu visaptverošu spēles funkcionalitātes pārklājumu.

4.2.1. Manuālā integrācijas testēšana

Integrācijas testēšana ir veikta manuāli, mijiedarboties ar spēles saskarni. Katrs testa scenārijs ir dokumentēta strukturētas tabulas formātā, ievērojot būtisku informāciju, piemēram, test nosaukumu, unikālo identifikatoru, aprakstu, izpildes soļus, gaidāmo rezultātu un faktisko rezultātu (veiksmīga testa gadījumā apzīmēts ar „Ok“, bet neveiksmīgu – „Err“). Testu gadījumi ir detalizētāk aprakstīti 4.1. tabulā.

4.1. tabula **Manuālā testēšana**

Testa ID	**Testa nosaukums**	**Soļi**	**Sagaidāmais rezultāts**	**Faktiskais rezultāts**
MT01	Spēles palaišana	1) Palaist spēli 2) Gaidīt ielādi	Parādās galvenā izvēlne ar „Play“ pogu.	Ok

MT02	Labirinta ģenerēšana	1) Palaist spēli 2) Nospiest „Play“ pogu 3) Gaidīt ielādi	Tiek uzģenerēts sešstūrains labirints ar sienām.	Ok
MT03	Stāvu pāreja (uz augšu)	1) Nokļūt līdz beigu šūnai 2) Nospiest taustiņu „E“ 3) Novērot animāciju	1) Jauna stāva ģenerēšana 2) Plūstoša pāreja starp stāviem uz augšu	Ok
MT04	Stāvu pāreja (uz leju)	1) Nokļūt līdz sākuma šūnai 2) Nospiest taustiņu „E“ 3) Novērot animāciju	1) Jauns stāvs netiek ģenerēts 2) Plūstoša pāreja starp stāviem uz leju	Ok
MT05	Papildspēju aktivizēšana	1) Aktivizēt papildspēju 2) Gaidīt atjaunošanās laiku	1) Papildspēja aktivizējas 2) Sākas atjaunošanās laiks	Err
MT06	Izstrādes rīku atvēršana	1) Palaist spēli izstrādes režīmā 2) Nospiest „Play“ pogu	Parādās „egui“ panelis ar labirinta konfigurācijas opcijām	Ok
MT07	Labirinta parametru maiņa	1) Atvērt „egui“ paneli 2) Mainīt labirinta izmēru un citus parametrus	Tiek ģenerēts jauns labirints ar mainītajiem parametriem	Ok
MT08	Spēlētāja kustība	1) Izmantot „WASD“ kustības taustiņus 2) Mēģināt šķērsot sienas	1) Plūstoša kustība brīvajā telpā 2) Sadursmes ar sienām strādā	Ok
MT09	Windows palaišana	1) Kompilēt spēli Windows platformai 2) Palaist .exe failu 3) Veikt pamata funkcionalitātes testus	Spēle darbojas Windows vidē bez kļūdām	Ok
MT10	Linux palaišana	1) Kompilēt spēli Linux platformai 2) Palaist bināro failu 3) Veikt pamata funkcionalitātes testus	Spēle darbojas Linux vidē bez kļūdām	Ok

MT11	macOS palaišana	1) Kompilēt spēli macOS platformai 2) Palaist .dmg pakotni 3) Veikt pamata funkcionalitātes testus	Spēle darbojas macOS vidē bez kļūdām	Err
MT12	WASM palaišana	1) Kompilēt spēli WASM mērķim 2) Atvērt pārlūkā 3) Veikt pamata funkcionalitātes testus	1) Spēle ielādējas pārlūkā 2) Darbojas visas pamatfunkcijas	Ok

Divi testi no manuālās testēšanas plāna netika izpildīti. Papildspēju tests (MT05) netika izpildīts, jo šī funkcionalitāte netika implementēta projekta pirmajā versijā, atstājot to kā potenciālu nākotnes papildinājumu. Savukārt macOS platformas tests (MT11) netika izpildīts tehnisku ierobežojumu dēļ – projekta izstrādātājam nav pieejama Apple aparatūra.

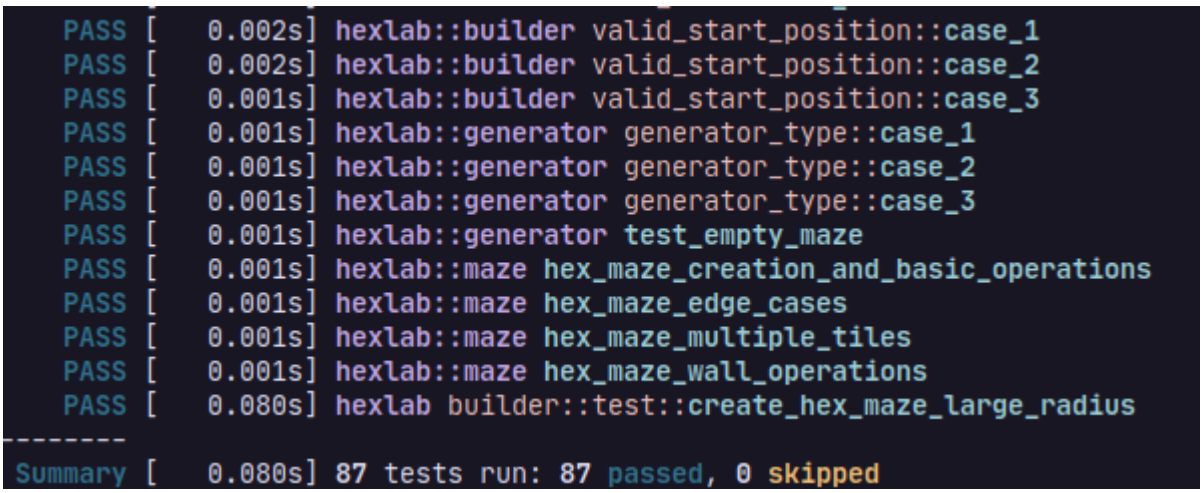
Tādējādi no divpadsmit definētajiem testiem desmit tika veiksmīgi izpildīti, viens netika implementēts (papildspējas), un viens tika daļēji izpildīts (macOS kompilēšana bez funkcionālās testēšanas).

4.2.2. Automatizēti testi

Automatizētā testēšanas sistēma plaši pārklāj bibliotēku „hexlab“, jo tā ir paredzēta publiskai lietošanai. Testēšanas stratēģijā ir ieviesti vairāki pārbaudes līmeņi: dokumentācijas testi no drošina piemēra koda pareizību, moduļu testi pārbauda iekšējo funkcionalitāti, savukārt testu mapē esošie vienībtesti un integrācijas testi pārbauda sarežģītākus gadījumus. Daļējs automatizēto testu izpildes rezultāts ir redzams 4.1. att., savukārt detalizēts testu izpildes pārskats ir redzams piejams pielikumā (sk. 6. pielikumu).

Izmantojot „cargo-tarpaulin“, testu pārklājums ir 81,69% (sk. 3. pielikumu), tomēr šis rādītājs pilnībā neatspoguļo faktisko pārklājumu, jo rīkam ir ierobežojumi attiecībā uz „inline“[17] funkcijām un citi tehniski ierobežojumi [16].

[17] https://doc.rust-lang.org/nightly/reference/attributes/codegen.html?highlight=inline



4.1. att. **Daļējs „hexlab“ bibliotēkas testu rezultāts**

Arī spēles kods saglabā stabilu testēšanas stratēģiju. Moduļu testi ir stratēģiski izvietoti līdzās implementācijas kodam tajā pašā failā, nodrošinot, ka katras komponentes funkcionalitāte tiek pārbaudīta izolēti. Šie testi attiecas uz tādām svarīgām spēles sistēmām kā spēlētāju kustība, sadursmju noteikšana, spēles stāvokļa pārvaldība u.c.

Visi testi tiek automātiski izpildīti kā nepārtrauktas integrācijas procesa daļa, nodrošinot tūlītēju atgriezenisko saiti par sistēmas stabilitāti un funkcionalitāti pēc katras koda izmaiņas.

5. PROGRAMMAS PROJEKTA ORGANIZĀCIJA

Kvalifikācijas darba izstrāde ir individuāls process, kur autors uzņemas pilnu atbildību par programmatūras izveidi un dokumentāciju.

Individuālā darba pieeja ļauj izvairīties no daudzām tipiskām projektu vadības problēmām. Tā kā nav nepieciešama koordinācija starp vairākiem izstrādātājiem, lēmumu pieņemšana ir tiešāka un ātrāka. Tas ir īpaši svarīgi, ņemot vērā kvalifikācijas darba ierobežoto laika periodu.

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“[15], kas analizē iespējamās problēmas un iesaka uzlabojumus (sk. 4.1. Statiskā testēšana nodaļu).

Kopā ar „Clippy“ tiek arī izmantots „Rustfmt“ [17], koda formatētājs, lai uzturētu vienotu koda formatējumu visā projektā. Šis rīks automātiski formatē kodu saskaņā ar Rust stila vadlīnijām [18].

Turklāt visas publiskās funkcijas un datu struktūras „hexlab“ bibliotēkā ir dokumentētas[18]. Šajā dokumentācijā ir ietverti detalizēti apraksti un lietošanas piemēri, kas ne tikai palīdz saprast kodu, bet arī atvieglo bibliotēkas testēšanu un kļūdu labošanu.

Programmatūras prasības specifikācija ir izstrādāta, ievērojot LVS 68:1996 standarta „Programmatūras prasību specifikācijas ceļvedis“ [2] un LVS 72:1996 standarta „Ieteicamā prakse programmatūras projektējuma aprakstīšanai“ standarta prasības [3].

5.2. Konfigurācijas pārvaldība

Pirmkods tiek pārvaldīts, izmantojot „git“[19] versiju kontroles sistēmu. Repozitorijs tiek izvietots platformā „GitHub“. Rīku konfigurācija ir definēta vairākos failos:

- „justfile“[20] – satur atkļūdošanas un laidiena komandas dažādām vidēm:

[18] https://docs.rs/hexlab/latest/hexlab/
[19] https://git-scm.com/doc
[20] https://just.systems/man/en/

  - atkļūdošanas kompilācijas ar iespējotu pilnu atpakaļsekošanu;
  - laidiena kompilācijas ar iespējotu optimizāciju.
- „GitHub Actions“ darbplūsmas, kas apstrādā [19]:
  - 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[8] platformā.

Versiju specifikācija notiek pēc semantiskās versiju atlases (MAJOR.MINOR.PATCH) [20]:

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

Projekta darbietilpības novērtēšanai tika izmantota QSM (angl. Quantitative Software Management, latv. kvantitatīvā programmatūras vadība) metodoloģija, kas balstās uz 550 verificētu programmatūras projektu datubāzi [21]. Izmantojot „tokei“ rīku [22], tika veikta detalizēta projekta koda analīze, kas parādija, ka „Maze Ascension“ projekts satur 2686 koda rindiņas (sk. 4. pielikumu), bet saistītā „hexlab“ bibliotēka – 979 rindiņas (sk. 5. pielikumu), kopā veidojot 3236 pirmkoda rindiņas, neiekļaujot tukšās rindiņas un komentārus.

Saskaņā ar QSM etalontabulu „Business Systems Implementation Unit (New and Modified IU) Benchmarks“, pirmās kvartiles projekti (25% mazākie no 550 biznesa sistēmu projektiem) vidēji ilgst 3,2 mēnešus, ar vidēji 1,57 izstrādātājiem un mediāna projekta apjomu – 1889 koda rindiņas [21]. Ņemot vērā, ka projekta autors ir students ar ierobežotu pieredzi, tiek izmantota pirmās kvartiles 50% diapazona augšējā robeža – 466 rindiņas personmēnesī. Tādējādi minimālais nepieciešamais koda apjoms trīs mēnešu darbam būtu $3 \times 466 = 1398$ rindiņas.

Projekta faktiskais koda apjoms (2906 rindiņas) vairāk nekā divkārt pārsniedz šo minimālo slieksni, kas apliecina projekta atbilstību trīs mēnešu darbietilpības prasībai. Turklāt jāņem vērā projekta papildu sarežģītības faktori:

- Bevy dzinēja un ECS arhitektūras apgūšana;
- Procedurālās ģenerēšanas algoritma izstrāde „hexlab“ bibliotēkai;
- „hexlab“ bibliotēkas izstrāde ar plašu dokumentāciju, ieskaitot API dokumentāciju, lietošanas piemērus un integrācijas vadlīnijas.

Šie faktori būtiski palielina projekta faktisko darbietilpību, jo prasa ne tikai koda rakstīšanu, bet arī izpēti, dokumentēšanu un optimizāciju.

6. SECINĀJUMI

Kvalifikācijas darba ietvaros tika izstrādāta trīsdimensiju spēle, izmantojot Bevy spēļu dzinēju un Rust programmēšanas valodu un tās dokumentācija. Projekta izstrādes gaitā tika sasniegti vairāki nozīmīgi rezultāti un gūtas vērtīgas atziņas.

Projekta galvenie sasniegumi ietver procedurāli ģenerēta sešstūraina labirinta implementāciju, kas balstās uz meklēšanas dziļumā (DFS) algoritmu. Šī funkcionalitāte tika veiksmīgi nodalīta atsevišķā „hexlab" bibliotēkā, kas padara to pieejamu atkārtotai izmantošanai citos projektos. Tika izveidota arī efektīva stāvu pārvaldības sistēma, kas nodrošina plūstošu pāreju starp dažādiem labirinta līmeņiem.

Bevy spēļu dzinēja izmantošana ļāva efektīvi implementēt entitāšu-komponenšu sistēmu (ECS), kas nodrošina labu veiktspēju un koda organizāciju. Tomēr tika konstatēts, ka Bevy ekosistēma joprojām ir aktīvās izstrādes stadijā, ko apliecina darba izstrādes laikā iznākusī jaunā versija (0.15). Šī versija ieviesa vairākas būtiskas izmaiņas, piemēram, „Required Components" (latv. nepieciešamo komponenšu) konceptu uzlabotu animāciju sistēmu un daudz ko citu, kas radīja nepieciešamību pielāgot esošo kodu [23]. Šāda strauja attīstība, no vienas puses, nodrošina jaunas iespējas un uzlabojumus, bet no otras puses, rada izaicinājumus saistībā ar dokumentācijas aktualitāti un bibliotēku savietojamību.

Izstrādes procesā īpaša uzmanība tika pievērsta koda kvalitātei un dokumentācijai. Tika izveidota detalizēta tehniskā dokumentācija, kas ietver gan sistēmas arhitektūras aprakstu, gan atsevišķu komponenšu funkcionalitātes skaidrojumu.

Projekta turpmākās attīstības iespējas ietver:

- papildu labirinta ģenerēšanas algoritmu implementāciju;
- spēles mehānikas paplašināšanu ar jaunām papildspējām;
- grafiskās kvalitātes uzlabojumus;
- tīkla spēles režīma ieviešanu.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] "ECS". Skatīts: 2024. gada12.septembrī. [Tiešsaiste]. Pieejams: https://en.wikipedia.org/wiki/Entity_component_system

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

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

[4] Mädje, Laurenz, Haug, Martin, un Typst Projekta Izstrādātāji, "Typst". [Tiešsaiste]. Pieejams: https://typst.app/

[5] Carter Anderson, "Bevy + WebGPU". Skatīts: 2024. gada20.septembrī. [Tiešsaiste]. Pieejams: https://bevyengine.org/news/bevy-webgpu/

[6] Vladyslav Batyrenko, "Bevy Egui bibliotēkas dokumentācija". Skatīts: 2024. gada26.septembrī. [Tiešsaiste]. Pieejams: https://docs.rs/bevy_egui/latest/bevy_egui/

[7] Jakob Hellermann, "Bevy Inspector Egui bibliotēkas dokumentācija". Skatīts: 2024. gada26.septembrī. [Tiešsaiste]. Pieejams: https://docs.rs/bevy-inspector-egui/latest/bevy_inspector_egui/

[8] Red Blob Games, "Hexagonal Grids". Skatīts: 2024. gada20.decembrī. [Tiešsaiste]. Pieejams: https://www.redblobgames.com/grids/hexagons/

[9] Bevy Projekta Izstādātāji, "Ievads Bevy ECS". Skatīts: 2024. gada12.septembrī. [Tiešsaiste]. Pieejams: https://bevyengine.org/learn/quick-start/getting-started/ecs/

[10] "Unofficial Bevy Cheat Book". Skatīts: 2024. gada14.septembrī. [Tiešsaiste]. Pieejams: https://bevy-cheatbook.github.io/

[11] "Single-responsibility principle". [Tiešsaiste]. Pieejams: https://en.wikipedia.org/wiki/Single-responsibility_principle

[12] "Separation of concerns". [Tiešsaiste]. Pieejams: https://en.wikipedia.org/wiki/Separation_of_concerns

[13] David Bernstein, *Patterns for Beginning Programmers*. lpp. 58–64. [Tiešsaiste]. Pieejams: https://pressbooks.lib.jmu.edu/programmingpatterns/

[14] "Maze Generation". [Tiešsaiste]. Pieejams: https://rosettacode.org/wiki/Maze_generation

[15] Rust Projekta Izstādātāji, "Clippy dokumentācija". [Tiešsaiste]. Pieejams: https://doc.rust-lang.org/stable/clippy/

[16] xd009642, "Tarpaulin rīks". Skatīts: 2024. gada18.decembrī. [Tiešsaiste]. Pieejams: https://crates.io/crates/cargo-tarpaulin

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

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

[19] GitHub komanda, "GitHub Actions dokumentācija". [Tiešsaiste]. Pieejams: https://docs.github.com/en/actions

[20] Tom Preston-Werner, "Semantiskā versiju veidošana". Skatīts: 2024. gada17.septembrī. [Tiešsaiste]. Pieejams: https://semver.org/

[21] QSM, "Software Project Performance Benchmark Tables". [Tiešsaiste]. Pieejams: https://www.qsm.com/resources/qsm-benchmark-tables

[22] XAMPPRocky, "Tokei rīks". [Tiešsaiste]. Pieejams: https://crates.io/crates/tokei

[23] Bevy Projekta Izstādātāji, "Bevy 0.15". [Tiešsaiste]. Pieejams: https://bevyengine.org/news/bevy-0-15/

PIELIKUMI

1. pielikums. Clippy rīka rezultāts „hexlab“ bibliotēkai

```
hexlab on  main [$] is  v0.5.3 via  v1.83.0 ❯ cargo clippy --all-features
    Finished `dev` profile [optimized + debuginfo] target(s) in 0.08s
```

2. pielikums. Clippy rīka rezultāts „Maze Ascension“ spēlei

```
maze-ascension on  main [$!] is  v1.0.1 via  v1.83.0 ❯ cargo clippy --all-features
    Finished `dev` profile [optimized + debuginfo] target(s) in 0.10s
```

3. pielikums. Tarpaulin rīka rezultāts „hexlab“ bibliotēkai

```
|| Uncovered Lines:
|| src/builder.rs: 115
|| src/generator/backtrack.rs: 29
|| src/maze.rs: 121, 153-154, 297, 334, 340-341, 352-353, 358-359, 377
|| src/tile.rs: 82, 92-94, 124
|| src/walls.rs: 68-69, 123, 160, 240-241, 277
|| Tested/Total Lines:
|| src/builder.rs: 29/30 +0.00%
|| src/generator/backtrack.rs: 15/16 +0.00%
|| src/generator/mod.rs: 2/2 +0.00%
|| src/maze.rs: 25/37 +0.00%
|| src/tile.rs: 10/15 +0.00%
|| src/walls.rs: 35/42 +0.00%
||
81.69% coverage, 116/142 lines covered, +0.00% change in coverage
```

4. pielikums. Tokei rīka rezultāts „Maze Ascension“ spēlei

```
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Rust                   60         3149         2686           71          392
 |- Markdown            22          246            0          207           39
 (Total)                           3395         2686          278          431
===============================================================================
 Total                  60         3149         2686           71          392
===============================================================================
```

5. pielikums. Tokei rīka rezultāts „hexlab“ bibliotēkai

```
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Rust                    9         1152          979           17          156
 |- Markdown             7          684            6          468          210
 (Total)                           1836          985          485          366
===============================================================================
 Total                   9         1152          979           17          156
===============================================================================
```

6. pielikums. Pilns „hexlab“ bibliotēkas testu rezultāts

```
Starting 87 tests across 4 binaries
    PASS [   0.062s] hexlab builder::test::create_hex_maze_tile_positions
    PASS [   0.062s] hexlab builder::test::create_hex_maze_various_radii::case_1
    PASS [   0.062s] hexlab builder::test::create_hex_maze_various_radii::case_2
    PASS [   0.062s] hexlab builder::test::create_hex_maze_various_radii::case_3
    PASS [   0.062s] hexlab builder::test::create_hex_maze_various_radii::case_4
    PASS [   0.062s] hexlab builder::test::create_hex_maze_various_radii::case_5
    PASS [   0.061s] hexlab builder::test::create_hex_maze_various_radii::case_6
    PASS [   0.061s] hexlab builder::test::maze_builder_new
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_connectivity::case_1
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_connectivity::case_2
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_connectivity::case_3
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_start_visited::case_1
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_start_visited::case_2
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_start_visited::case_3
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_walls_removed::case_1
    PASS [   0.061s] hexlab generator::backtrack::test::recursive_backtrack_walls_removed::case_2
    PASS [   0.060s] hexlab generator::backtrack::test::recursive_backtrack_walls_removed::case_3
    PASS [   0.002s] hexlab tile::test::bevy_tests::hex_tile_to_vec3
    PASS [   0.002s] hexlab tile::test::bevy_tests::hex_tile_to_vec2
    PASS [   0.002s] hexlab tile::test::hex_hex_into_tile
    PASS [   0.002s] hexlab tile::test::different_positions
    PASS [   0.002s] hexlab tile::test::hex_boundaries
    PASS [   0.004s] hexlab tile::test::hex_tile_creation_and_properties
    PASS [   0.004s] hexlab tile::test::hex_tile_display
    PASS [   0.004s] hexlab tile::test::hex_tile_from_hex
    PASS [   0.004s] hexlab walls::test::as_bits_all_walls
    PASS [   0.003s] hexlab walls::test::bit_manipulation
    PASS [   0.003s] hexlab walls::test::as_bits_multiple_walls
    PASS [   0.003s] hexlab walls::test::default_creates_closed_walls
    PASS [   0.003s] hexlab walls::test::empty_creates_no_walls
    PASS [   0.002s] hexlab walls::test::fill_adds_multiple_walls
    PASS [   0.002s] hexlab walls::test::fill_preserves_existing_walls
    PASS [   0.002s] hexlab walls::test::from_array_all_directions
    PASS [   0.002s] hexlab walls::test::from_array_conversion
    PASS [   0.002s] hexlab walls::test::from_array_duplicates
    PASS [   0.002s] hexlab walls::test::from_array_empty
    PASS [   0.002s] hexlab walls::test::from_array_multiple
    PASS [   0.002s] hexlab walls::test::from_array_single
    PASS [   0.002s] hexlab walls::test::from_edge_direction_conversion
    PASS [   0.002s] hexlab walls::test::from_edge_direction_flat_east
    PASS [   0.002s] hexlab walls::test::from_edge_direction_flat_north
    PASS [   0.005s] hexlab tile::test::hex_tile_wall_modifications
    PASS [   0.005s] hexlab walls::test::all_directions_creates_closed_walls
    PASS [   0.004s] hexlab walls::test::as_bits_empty
    PASS [   0.004s] hexlab walls::test::as_bits_single_wall
    PASS [   0.002s] hexlab walls::test::from_edge_direction_flat_north_west
    PASS [   0.004s] hexlab walls::test::from_edge_direction_flat_south
    PASS [   0.004s] hexlab walls::test::from_edge_direction_flat_south_east
    PASS [   0.004s] hexlab walls::test::from_edge_direction_flat_south_west
    PASS [   0.004s] hexlab walls::test::from_iterator
    PASS [   0.004s] hexlab walls::test::from_iterator_all_directions
    PASS [   0.004s] hexlab walls::test::from_iterator_empty
    PASS [   0.004s] hexlab walls::test::from_iterator_handles_duplicates
    PASS [   0.004s] hexlab walls::test::from_iterator_multiple
    PASS [   0.004s] hexlab walls::test::from_iterator_single
    PASS [   0.003s] hexlab walls::test::from_u8_conversion
    PASS [   0.003s] hexlab walls::test::insert_single_wall
    PASS [   0.003s] hexlab walls::test::new_created_closed_walls
    PASS [   0.002s] hexlab walls::test::remove_existing_wall
    PASS [   0.003s] hexlab walls::test::remove_nonexistent_wall
    PASS [   0.003s] hexlab walls::test::toggle_removes_wall
    PASS [   0.002s] hexlab walls::test::toggle_wall
    PASS [   0.005s] hexlab walls::test::from_iterator_duplicates
    PASS [   0.003s] hexlab::builder builder_without_radius
    PASS [   0.002s] hexlab::builder maze_size::case_4
    PASS [   0.002s] hexlab::builder different_seeds_produce_different_mazes
    PASS [   0.002s] hexlab::builder maze_size::case_5
    PASS [   0.002s] hexlab::builder generate_maze_with_different_types::case_1
    PASS [   0.001s] hexlab::builder maze_with_seed
    PASS [   0.002s] hexlab::builder invalid_start_position
    PASS [   0.002s] hexlab::builder maze_connectivity
    PASS [   0.002s] hexlab::builder maze_boundaries
    PASS [   0.002s] hexlab::builder maze_size::case_1
    PASS [   0.002s] hexlab::builder maze_size::case_2
    PASS [   0.002s] hexlab::builder maze_size::case_3
    PASS [   0.002s] hexlab::builder valid_start_position::case_1
    PASS [   0.002s] hexlab::builder valid_start_position::case_2
    PASS [   0.001s] hexlab::builder valid_start_position::case_3
    PASS [   0.001s] hexlab::generator generator_type::case_1
    PASS [   0.001s] hexlab::generator generator_type::case_2
    PASS [   0.001s] hexlab::generator generator_type::case_3
    PASS [   0.001s] hexlab::generator test_empty_maze
    PASS [   0.001s] hexlab::maze hex_maze_creation_and_basic_operations
    PASS [   0.001s] hexlab::maze hex_maze_edge_cases
    PASS [   0.001s] hexlab::maze hex_maze_multiple_tiles
    PASS [   0.001s] hexlab::maze hex_maze_wall_operations
    PASS [   0.080s] hexlab builder::test::create_hex_maze_large_radius
----------
 Summary [   0.080s] 87 tests run: 87 passed, 0 skipped
```

7. pielikums. Labirinta būvētāja implementācijas piemērs

```rust
/// A builder pattern for creating hexagonal mazes.
///
/// This struct provides a fluent interface for configuring and building
hexagonal mazes.
/// It offers flexibility in specifying the maze size, random seed, generation
algorithm,
/// and starting position.
///
/// # Examples
///
/// Basic usage:
/// ```
/// use hexlab::prelude::*;
///
/// let maze = MazeBuilder::new()
///     .with_radius(5)
///     .build()
///     .expect("Failed to create maze");
///
/// // A radius of 5 creates 61 hexagonal tiles
/// assert!(!maze.is_empty());
/// assert_eq!(maze.len(), 91);
/// ```
///
/// Using a seed for reproducible results:
/// ```
/// use hexlab::prelude::*;
///
/// let maze1 = MazeBuilder::new()
///     .with_radius(3)
///     .with_seed(12345)
///     .build()
///     .expect("Failed to create maze");
///
```

```rust
/// let maze2 = MazeBuilder::new()
///     .with_radius(3)
///     .with_seed(12345)
///     .build()
///     .expect("Failed to create maze");
///
/// // Same seed should produce identical mazes
/// assert_eq!(maze1.len(), maze2.len());
/// assert_eq!(maze1, maze2);
/// ```
///
/// Specifying a custom generator:
/// ```
/// use hexlab::prelude::*;
///
/// let maze = MazeBuilder::new()
///     .with_radius(7)
///     .with_generator(GeneratorType::RecursiveBacktracking)
///     .build()
///     .expect("Failed to create maze");
/// ```
#[derive(Default)]
pub struct MazeBuilder {
    radius: Option<u16>,
    seed: Option<u64>,
    generator_type: GeneratorType,
    start_position: Option<Hex>,
}

impl MazeBuilder {
    /// Creates a new [`MazeBuilder`] instance with default settings.
    #[inline]
    #[must_use]
    pub fn new() -> Self {
```

```
        Self::default()
    }

    /// Sets the radius for the hexagonal maze.
    ///
    /// The radius determines the size of the maze, specifically the number of
tiles
    /// from the center (0,0) to the edge of the hexagon, not including the
center tile.
    /// For example, a radius of 3 would create a maze with 3 tiles from
center to edge,
    /// resulting in a total diameter of 7 tiles (3 + 1 + 3).
    ///
    /// # Arguments
    ///
    /// - `radius` - The number of tiles from the center to the edge of the
hexagon.
    #[inline]
    #[must_use]
    pub const fn with_radius(mut self, radius: u16) -> Self {
        self.radius = Some(radius);
        self
    }

    /// Sets the random seed for maze generation.
    ///
    /// Using the same seed will produce identical mazes, allowing for
reproducible results.
    ///
    /// # Arguments
    ///
    /// - `seed` - The random seed value.
    #[inline]
    #[must_use]
```

```rust
    pub const fn with_seed(mut self, seed: u64) -> Self {
        self.seed = Some(seed);
        self
    }

    /// Sets the generator algorithm for maze creation.
    ///
    /// Different generators may produce different maze patterns and
characteristics.
    ///
    /// # Arguments
    ///
    /// - `generator_type` - The maze generation algorithm to use.
    #[inline]
    #[must_use]
    pub const fn with_generator(
        mut self,
        generator_type: GeneratorType,
    ) -> Self {
        self.generator_type = generator_type;
        self
    }

    /// Sets the starting position for maze generation.
    ///
    /// # Arguments
    ///
    /// - `pos` - The hexagonal coordinates for the starting position.
    #[inline]
    #[must_use]
    pub const fn with_start_position(mut self, pos: Hex) -> Self {
        self.start_position = Some(pos);
        self
    }
```

```
137
138         /// Builds the hexagonal maze based on the configured parameters.
139         ///
140         /// # Errors
141         ///
142         /// Returns [`MazeBuilderError::NoRadius`] if no radius is specified.
143         /// Returns [`MazeBuilderError::InvalidStartPosition`] if the start
144     position is outside maze
145         /// bounds.
146         ///
147         /// # Examples
148         ///
149         /// ```
150         /// use hexlab::prelude::*;
151         ///
152         /// // Should fail without radius
153         /// let result = MazeBuilder::new().build();
154         /// assert!(result.is_err());
155         ///
156         /// // Should succeed with radius
157         /// let result = MazeBuilder::new()
158         ///     .with_radius(3)
159         ///     .build();
160         /// assert!(result.is_ok());
161         ///
162         /// let maze = result.unwrap();
163         /// assert!(!maze.is_empty());
164         /// ```
165         pub fn build(self) -> Result<Maze, MazeBuilderError> {
166             let radius = self.radius.ok_or(MazeBuilderError::NoRadius)?;
167             let mut maze = create_hex_maze(radius);
168
169             if let Some(start_pos) = self.start_position {
170                 if maze.get(&start_pos).is_none() {
```

```
                return Err(MazeBuilderError::InvalidStartPosition(start_pos));
            }
        }

        if !maze.is_empty() {
            self.generator_type.generate(
                &mut maze,
                self.start_position,
                self.seed,
            );
        }

        Ok(maze)
    }
}
```

8. pielikums. Labirinta sienu reprezentācijas piemērs

```
/// A bit-flag representation of walls in a hexagonal tile.
///
/// `Walls` uses an efficient bit-flag system to track the presence or absence
of walls
/// along each edge of a hexagonal tile. Each of the six possible walls is
represented
/// by a single bit in an 8-bit integer, allowing for fast operations and
minimal memory usage.
///
/// # Examples
///
/// Creating and manipulating walls:
/// ```
/// use hexlab::prelude::*;
///
/// // Create a hexagon with all walls
/// let walls = Walls::new();
/// assert!(walls.is_enclosed());
```

```rust
///
/// // Create a hexagon with no walls
/// let mut walls = Walls::empty();
/// assert!(walls.is_empty());
///
/// // Add specific walls
/// walls.insert(EdgeDirection::FLAT_NORTH);
/// walls.insert(EdgeDirection::FLAT_SOUTH);
/// assert_eq!(walls.count(), 2);
/// ```
#[cfg_attr(feature = "serde", derive(serde::Serialize, serde::Deserialize))]
#[cfg_attr(feature = "bevy_reflect", derive(bevy_reflect::Reflect))]
#[cfg_attr(feature = "bevy", derive(Component))]
#[cfg_attr(feature = "bevy", reflect(Component))]
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub struct Walls(u8);

impl Walls {
    /// Insert a wall in the specified direction.
    ///
    /// # Arguments
    ///
    /// - `direction` - The direction in which to insert the wall.
    ///
    /// # Returns
    ///
    /// Returns `true` if a wall was present, `false` otherwise.
    ///
    /// # Examples
    ///
    /// ```
    /// use hexlab::prelude::*;
    ///
    /// let mut walls = Walls::empty();
```

```
    /// assert_eq!(walls.count(), 0);
    ///
    /// assert!(!walls.insert(1));
    /// assert_eq!(walls.count(), 1);
    ///
    /// assert!(walls.insert(1));
    /// assert_eq!(walls.count(), 1);
    ///
    /// assert!(!walls.insert(EdgeDirection::FLAT_NORTH));
    /// assert_eq!(walls.count(), 2);
    /// ```
    #[inline]
    pub fn insert<T>(&mut self, direction: T) -> bool
    where
        T: Into<Self>,
    {
        let mask = direction.into().0;
        let was_present = self.0 & mask != 0;
        self.0 |= mask;
        was_present
    }

    /// Removes a wall in the specified direction.
    ///
    /// # Arguments
    ///
    /// - `direction` - The direction from which to remove the wall.
    ///
    /// # Returns
    ///
    /// Returns `true` if a wall was present and removed, `false` otherwise.
    ///
    /// # Examples
    ///
```

```
272         /// ```
273         /// use hexlab::prelude::*;
274         ///
275         /// let mut walls = Walls::new();
276         ///
277         /// assert!(walls.remove(1));
278         /// assert_eq!(walls.count(), 5);
279         ///
280         /// assert!(!walls.remove(1));
281         /// assert_eq!(walls.count(), 5);
282         ///
283         /// assert!(walls.remove(EdgeDirection::FLAT_NORTH));
284         /// assert_eq!(walls.count(), 4);
285         /// ```
286         #[inline]
287         pub fn remove<T>(&mut self, direction: T) -> bool
288         where
289             T: Into<Self>,
290         {
291             let mask = direction.into().0;
292             let was_present = self.0 & mask != 0;
293             self.0 &= !mask;
294             was_present
295         }
296
297         /// Checks if there is a wall in the specified direction.
298         ///
299         /// # Arguments
300         ///
301         /// - `other` - The direction to check for a wall.
302         ///
303         /// # Examples
304         ///
305         /// ```
```

```
306         /// use hexlab::prelude::*;
307         ///
308         /// let mut walls = Walls::empty();
309         /// walls.insert(EdgeDirection::FLAT_NORTH);
310         ///
311         /// assert!(walls.contains(EdgeDirection::FLAT_NORTH));
312         /// assert!(!walls.contains(EdgeDirection::FLAT_SOUTH));
313         /// ```
314         #[inline]
315         pub fn contains<T>(&self, direction: T) -> bool
316         where
317             T: Into<Self>,
318         {
319             self.0 & direction.into().0 != 0
320         }
321
322         /// Toggles a wall in the specified direction.
323         ///
324         /// If a wall exists in the given direction, it will be removed.
325         /// If no wall exists, one will be added.
326         ///
327         /// # Arguments
328         ///
329         /// - `direction` - The direction in which to toggle the wall.
330         ///
331         /// # Returns
332         ///
333         /// The previous state (`true` if a wall was present before toggling,
334     `false` otherwise).
335         ///
336         /// # Examples
337         ///
338         /// ```
339         /// use hexlab::prelude::*;
```

```
    ///
    /// let mut walls = Walls::empty();
    ///
    /// assert!(!walls.toggle(0));
    /// assert_eq!(walls.count(), 1);
    ///
    /// assert!(walls.toggle(0));
    /// assert_eq!(walls.count(), 0);
    /// ```
    pub fn toggle<T>(&mut self, direction: T) -> bool
    where
        T: Into<Self> + Copy,
    {
        let mask = direction.into().0;
        let was_present = self.0 & mask != 0;
        self.0 ^= mask;
        was_present
    }
}

impl From<EdgeDirection> for Walls {
    fn from(value: EdgeDirection) -> Self {
        Self(1 << value.index())
    }
}

impl From<u8> for Walls {
    fn from(value: u8) -> Self {
        Self(1 << value)
    }
}

impl FromIterator<EdgeDirection> for Walls {
    fn from_iter<T: IntoIterator<Item = EdgeDirection>>(iter: T) -> Self {
```

```
            let mut walls = 0u8;
            for direction in iter {
                walls |= 1 << direction.index();
            }
            Self(walls)
        }
    }

    impl<const N: usize> From<[EdgeDirection; N]> for Walls {
        fn from(value: [EdgeDirection; N]) -> Self {
            value.into_iter().collect()
        }
    }
```

9. pielikums. Labirinta stāvu implementācijas piemērs

```
    /// Move floor entities to their target Y positions based on movement speed
    ///
    /// # Behavior
    /// - Calculates movement distance based on player speed and delta time
    /// - Moves floors towards their target Y position
    /// - Removes FloorYTarget component when floor reaches destination
    pub fn move_floors(
        mut commands: Commands,
        mut maze_query: Query<
            (Entity, &mut Transform, &FloorYTarget),
            (With<HexMaze>, With<FloorYTarget>),
        >,
        player_query: Query<&MovementSpeed, With<Player>>,
        time: Res<Time>,
    ) {
        let speed = player_query.get_single().map_or(100., |s| s.0);
        let movement_distance = speed * time.delta_secs();
        for (entity, mut transform, movement_state) in maze_query.iter_mut() {
            let delta = movement_state.0 - transform.translation.y;
            if delta.abs() > MOVEMENT_THRESHOLD {
```

```
            let movement = delta.signum() *
movement_distance.min(delta.abs());
            transform.translation.y += movement;
        } else {
            transform.translation.y = movement_state.0; // snap to final
position
            commands.entity(entity).remove::<FloorYTarget>();
        }
    }
}

/// Handle floor transition events by setting up floor movement targets
///
/// # Behavior
/// - Checks if any floors are currently moving
/// - Processes floor transition events
/// - Sets target Y positions for all maze entities
/// - Updates current and next floor designations
pub fn handle_floor_transition_events(
    mut commands: Commands,
    mut maze_query: Query<
        (Entity, &Transform, Option<&FloorYTarget>),
        With<HexMaze>
    >,
    current_query: Query<Entity, With<CurrentFloor>>,
    next_query: Query<Entity, With<NextFloor>>,
    mut event_reader: EventReader<TransitionFloor>,
) {
    let is_moving = maze_query
        .iter()
        .any(|(_, _, movement_state)| movement_state.is_some());
    if is_moving {
        return;
    }
```

```
441         for event in event_reader.read() {
442             let direction = event.into();
443             let Some(current_entity) = current_query.get_single().ok() else {
444                 continue;
445             };
446             let Some(next_entity) = next_query.get_single().ok() else {
447                 continue;
448             };
449             for (entity, transforms, movement_state) in maze_query.iter_mut() {
450                 let target_y = (FLOOR_Y_OFFSET as f32).mul_add(direction,
451 transforms.translation.y);
452                 if movement_state.is_none() {
453                     commands.entity(entity).insert(FloorYTarget(target_y));
454                 }
455             }
456             update_current_next_floor(&mut commands, current_entity, next_entity);
457             break;
458         }
```

10. pielikums. Labirinta ģenerācijas implementācijas piemērs

```
459 /// Spawns a new maze floor in response to a SpawnMaze trigger event
460 pub(super) fn spawn_maze(
461     trigger: Trigger<SpawnMaze>,
462     mut commands: Commands,
463     mut meshes: ResMut<Assets<Mesh>>,
464     mut materials: ResMut<Assets<StandardMaterial>>,
465     maze_query: Query<(Entity, &Floor, &Maze)>,
466     global_config: Res<GlobalMazeConfig>,
467 ) {
468     let SpawnMaze { floor, config } = trigger.event();
469 
470     if maze_query.iter().any(|(_, f, _)| f.0 == *floor) {
471         warn!("Floor {} already exists, skipping creation", floor);
472         return;
473     }
```

```
474
475         let maze = match generate_maze(config) {
476             Ok(m) => m,
477             Err(e) => {
478                 error!("Failed to generate maze for floor {floor}: {:?}", e);
479                 return;
480             }
481         };
482
483         // Calculate vertical offset based on floor number
484         let y_offset = match *floor {
485             1 => 0,              // Ground/Initial floor (floor 1) is at y=0
486             _ => FLOOR_Y_OFFSET, // Other floors are offset vertically
487         } as f32;
488
489         let entity = commands
490             .spawn((
491                 Name::new(format!("Floor {}", floor)),
492                 HexMaze,
493                 maze.clone(),
494                 Floor(*floor),
495                 config.clone(),
496                 Transform::from_translation(Vec3::ZERO.with_y(y_offset)),
497                 Visibility::Visible,
498             ))
499             .insert_if(CurrentFloor, || *floor == 1) // Only floor 1 gets
500     CurrentFloor
501             .id();
502
503         let assets = MazeAssets::new(&mut meshes, &mut materials, &global_config);
504         spawn_maze_tiles(
505             &mut commands,
506             entity,
507             &maze,
```

```
            &assets,
            config,
            &global_config,
        );
    }

    /// Spawns all tiles for a maze as children of the parent maze entity
    pub fn spawn_maze_tiles(
        commands: &mut Commands,
        parent_entity: Entity,
        maze: &Maze,
        assets: &MazeAssets,
        maze_config: &MazeConfig,
        global_config: &GlobalMazeConfig,
    ) {
        commands.entity(parent_entity).with_children(|parent| {
            for tile in maze.values() {
                spawn_single_hex_tile(
                    parent,
                    assets,
                    tile,
                    maze_config,
                    global_config,
                );
            }
        });
    }

    /// Spawns a single hexagonal tile with appropriate transforms and materials
    pub(super) fn spawn_single_hex_tile(
        parent: &mut ChildBuilder,
        assets: &MazeAssets,
        tile: &HexTile,
        maze_config: &MazeConfig,
```

```
    global_config: &GlobalMazeConfig,
) {
    let world_pos = tile.to_vec3(&maze_config.layout);
    let rotation = match maze_config.layout.orientation {
        // Pointy hexagons don't need additional rotation (0 degrees)
        HexOrientation::Pointy => Quat::from_rotation_y(0.0),
        // Flat-top hexagons need 30 degrees (pi/6) rotation around Y axis
        HexOrientation::Flat => Quat::from_rotation_y(FRAC_PI_6),
    };

    // Select material based on tile position: start, end, or default
    let material = match tile.pos() {
        pos if pos == maze_config.start_pos => assets
            .custom_materials
            .get(&RosePine::Pine)
            .cloned()
            .unwrap_or_default(),
        pos if pos == maze_config.end_pos => assets
            .custom_materials
            .get(&RosePine::Love)
            .cloned()
            .unwrap_or_default(),
        _ => assets.hex_material.clone(),
    };

    parent
        .spawn((
            Name::new(format!("Hex {}", tile)),
            Tile,
            Mesh3d(assets.hex_mesh.clone()),
            MeshMaterial3d(material),
            Transform::from_translation(world_pos).with_rotation(rotation),
        ))
        .with_children(|parent| {
```

```
576                 spawn_walls(parent, assets, tile.walls(), global_config)
577             });
578     }
579
580     /// Spawns walls around a hexagonal tile based on the walls configuration
581     fn spawn_walls(
582         parent: &mut ChildBuilder,
583         assets: &MazeAssets,
584         walls: &Walls,
585         global_config: &GlobalMazeConfig,
586     ) {
587         // Base rotation for wall alignment (90 degrees counter-clockwise)
588         let z_rotation = Quat::from_rotation_z(-FRAC_PI_2);
589         let y_offset = global_config.height / 2.;
590
591         for i in 0..6 {
592             if !walls.contains(i) {
593                 continue;
594             }
595
596             // Calculate the angle for this wall
597             // FRAC_PI_3 = 60 deg
598             // Negative because going clockwise
599             // i * 60 produces: 0, 60, 120, 180, 240, 300
600             let wall_angle = -FRAC_PI_3 * i as f32;
601
602             // cos(angle) gives x coordinate on unit circle
603             // sin(angle) gives z coordinate on unit circle
604             // Multiply by wall_offset to get actual distance from center
605             let x_offset = global_config.wall_offset() * f32::cos(wall_angle);
606             let z_offset = global_config.wall_offset() * f32::sin(wall_angle);
607
608             // x: distance along x-axis from center
609             // y: vertical offset from center
```

```
            // z: distance along z-axis from center
            let pos = Vec3::new(x_offset, y_offset, z_offset);

            // 1. Rotate around x-axis to align wall with angle
            // 2. Add FRAC_PI_2 (90) to make wall perpendicular to angle
            let x_rotation = Quat::from_rotation_x(wall_angle + FRAC_PI_2);
            let final_rotation = z_rotation * x_rotation;

            spawn_single_wall(parent, assets, final_rotation, pos);
        }
    }
}

/// Spawns a single wall segment with the specified rotation and position
fn spawn_single_wall(
    parent: &mut ChildBuilder,
    assets: &MazeAssets,
    rotation: Quat,
    offset: Vec3,
) {
    parent.spawn((
        Name::new("Wall"),
        Wall,
        Mesh3d(assets.wall_mesh.clone()),
        MeshMaterial3d(assets.wall_material.clone()),
        Transform::from_translation(offset).with_rotation(rotation),
    ));
}
```

DOKUMENTĀRĀ LAPA

Kvalifikācijas darbs „**Spēles izstrāde, izmantojot Bevy spēļu dzinēju**“ ir izstrādāts Latvijas Universitātes Eksakto zinātņu un tehnoloģiju fakultātē, Datorikas nodaļā.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai un/vai recenzentam uzrādītajai darba versijai.

Autors: **Kristiāns Francis Cagulis, kc22015 06.01.2025.**

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: **Mg. dat. Jānis Iljins 06.01.2025.**

Recenzents: **Artūrs Driķis**

Darbs iesniegs **06.01.2025.**

Kvalifikācijas darbu pārbaudījumu komisijas sekretārs (elektronisks paraksts)