|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | Stiliaus knyga – Vilniaus kolegija |  | |  |
| VILNIAUS KOLEGIJA  ELEKTRONIKOS IR INFORMATIKOS FAKULTETAS  PROGRAMINĖS ĮRANGOS KATEDRA | | | | | | |
|  | | | | | | |
| **ŽAIDIMAS ANGLIAKASYS** | | | | | | |
| INTELEKTIKOS SAVARANKIŠKAS DARBAS | | | | | | |
| PROGRAMŲ SISTEMOS (PI20 GRUPĖ)  6531BX028 | | | | | | |
| **STUDENTĖ** | |  | | | TOMA ŽILIŪTĖ | |
|  | | (Parašas)  2023-03 | | |  | |
|  | | |
| **DĖSTYTOJAS** | |  | | | D. SAVULIONIS | |
|  | | |
|  | | (Parašas)  2023-03 | | |  | |
| 2023 | | |

**TURINY****S**

ĮVADAS 6

1. INTELEKTIKA IR ŽAIDIMAI 7

1.1. Pasirinkimų medžiai 7

1.2. Elgesio medžiai 8

1.3. A\* paieškos algoritmas 10

2. AGENTO IR ŽAIDĖJO APRAŠAS 13

3. FUNKCINIAI/NEFUNKCINIAI REIKALAVIMAI IR DIAGRAMOS 15

3.1. Funkciniai ir nefunkciniai reikalavimai 15

3.2. Diagramos 15

4. PROGRAMINĖ REALIZACIJA 24

5. NAUDOTOJO INSTRUKCIJA 34

5.1. Diegimo instrukcija 34

5.2. Naudojimas 34

IŠVADOS 40

LITERATŪROS ŠALTINIAI 41

**Paveikslėlių sąrašas**

1 pav. Pavaizduotas pasirinkimų medis parodantis ar paskola bus duota. 7

2 pav. Priešo veiksmų pasirinkimo medis 8

3 pav. Parinkėjo mazgo žymėjimas 9

4 pav. Sekos mazgo žymėjimas 9

5 pav. Ankščiau pateiktas pasirinkimų medis, paverstas į elgesio medį 10

6 pav. Svorinis grafas, kurio pradinis mazgas yra A, o galutinis- D 11

7 pav. Nuo A iki D mazgo greičiausias kelias 12

8 pav. Žaidimo priešai 13

9 pav. Žaidėjo modelis 13

10 pav. Ne žaidėjo valdomas personažas 14

11 pav. Duobė 14

12 pav. Panaudos atvejai 15

13 pav. „Judėti į dešinę“ UML 16

14 pav. „Judėti į kairę“ UML 17

15 pav. „Judėti į viršų“ UML 18

16 pav. „Judėti į apačią“ UML 18

6 lent. „Sąveikauti su duobėmis“ aprašas 19

17 pav. „Sąveikauti su duobėmis“ UML 19

18 pav. „Sąveikauti su ne žaidėjo valdomu veikėju“ UML 20

19 pav. „Pulti į visus šonus“ UML 21

20 pav. „Pulti žaidėją“ UML 22

21 pav. „Gaudyti žaidėją“ UML 23

22 pav. „Bėgti nuo žaidėjo“ 23

23 pav. Pradinio lygio šakninis mazgas pavadinimu „World\_1“ su savo vaikiniais elementais ir objektais. „Player“, „MouseyBoi“, „Dialogue“ ir „Hole\_Enter“ yra atskirai sukurtos scenos su savo scenarijais ir šakniniais elementais, pridėtos prie „World\_1“ kaip vaikinės 24

25 pav. „\_ready()“, „animationPlay“ ir „\_physics\_process“ funkcijos 25

26 pav. „movement“, „attacking“ ir „attackAnimation“ funkcijos 25

24 pav. „Player“ scena 25

29 pav. „Dialogue“ funkcijos 26

28 pav. „Area2D“ funkcijos 26

27 pav. „\_on\_AnimationPlayer\_animation\_finished“ ir „\_on\_HurtBox\_are\_entered“ funkcijos 26

31 pav. „World\_2“ šakninis elementas su vaikiniais 27

30 pav. Vienas iš „Hole\_Enter“ scenarijų 27

33 pav. Gaunama ir pavaizduojama žaidėjo sveikatos informacija 28

32 pav. „World\_2“ scenarijus 28

35 pav. „World\_5“ „Hole\_Enter“ scenarijus 28

34 pav. „World\_5“ scenarijus 28

36 pav. „Bat“ priešo elementai 29

38 pav. „WonderTimer“ scenarijus, kuris generuoja priešo buvimą vienoje vietoje laiką 29

37 pav. Kito priešo aptykimas ir Žaidėjo aptykimas kodo palyginimas 29

39 pav. „\_on\_HurtBox\_area\_entered“, „\_physics\_process“, „\_on\_AnimationPlayer\_animation\_finished“ funkcijos 30

41 pav. „Bat“ agento elgesio medis 31

40 pav. „Bat“ agento „task“ funkcijos 31

43 pav. „Armor\_Slime“ elgesio medis 32

42 pav. „task\_Idal“ elgesio medžio funkcija 32

44 pav. Antro mygtuko scenarijus. Ant mygtuko paspaudus, žaidėjas atsiras pirmajame lygyje 32

46 pav. Vienintelė mygtuko funkcija, kuri nurodo, jog reikia užrakinti visus lygius ir vėl pradeda žaidimą iš naujo 33

45 pav. Vienintelė „Map“ scenos scenarijaus funkcija, kuri atrakina lygius, priklausant nuo skaičiaus, kurį gauna iš „LevelTracker“ 33

48 pav. Stovėjimas prie duobės 34

47 pav. Kalbėjimas su pele 34

49 pav. Lygių pasirinkimo ekranas 35

50 pav. Puolamas šikšnosparnis 35

53 pav. Šikšnosparnis susirado draugą, dabar jie puola kartu 36

52 pav. Šikšnosparnis bėga nuo žaidėjo 36

51 pav. Šikšnosparnis nukenčia 36

54 pav. Prieš užmušant visus priešus 37

55 pav. Užmušus visus priešus 37

56 pav. Pirmo lygio bosas 37

58 pav. Žaidėjas bėga nuo priešų paskutiniame lygyje 38

57 pav. Duobė, kurią panaudojus galima patekti į kitą lygį 38

60 pav. Laimėjimo langas 39

59 pav. „Game over“ ekranas 39

**Lentelių sąrašas**

1 lentelė. Žaidimo agentas 13

2 lent. „Judėti į dešinę“ aprašas 16

3 lent. „Judėti į kairę“ aprašas 16

5 lent. „Judėti į apačią“ aprašas 18

6 lent. „Sąveikauti su duobėmis“ aprašas 19

7 lent. „Sąveikauti su ne žaidėjo valdomu veikėju“ aprašas 20

8 lent. „Pulti į visus šonus“ aprašas 20

9 lent. „Pulti žaidėją“ aprašas 21

10 lent. „Gaudyti žaidėją“ aprašas 22

11 lent. „Bėgti nuo žaidėjo aprašas“ 23

**ĮVADAS**

Pagrindinė šio žaidimo idėja yra išlaisvinti po žeme užstrigusį angliakasį, užmušant priešus. Norint išsilaisvinti iš požemio, reikia kiekviename lygyje nugalėti didžiulius priešus, kurie yra pasislėpę uždarytose vietose, atskirai nuo visų kitų priešų. Vienintelis laimėjimo būdas- pasiekti paskutinį lygį, kuris yra užpildytas priešais ir juos visus nudėjus pasinaudoti atsiradusia duobe.

Šio žaidimo idėja kilo, norint sujungti kai kuriuos kitų žaidimų, kaip „Stardew valley“ ir ankstesnius „The Legend of Zelda“ frančizei priklausančias iteracijas, elementus kartu. Kuriant žaidimą buvo atsižvelgta į „Stardew valley“ žaidimo kasyklų lygius, kuriose reikia skaldyti akmenis ir apsiginti nuo įvairių priešų, kaip šlykštukai, mumijos, skeletai, didelės musės ar bitės, šikšnosparniai ir kiti. Skirtingai negu „Stardew valley“, kur galima priešus tiesiog ignoruoti ir tiesiog toliau rinkti sau reikalingus resursus. Šiame žaidime norint pereiti į kitą lygį būtinai reikia panaikinti visus to lygio priešus, o kartais net išspręsti galvosūkius. To nepadarius, niekada neatsiras kelias į kitą lygį. Taip pat skirtingai, negu „Stardew valley“, šiame sukurtame žaidime yra specialūs priešai, kurie yra dažnai vadinami „bosais“ (angl. boss).

Kalbant apie „The Legend of Zelda“, iš šios frančizės specifinių lygių, kurie žaidime kartais yra vadinami šventyklomis arba požemiais, kilo idėja žaidime pridėti „bosus“, kitaip sakant, didesnius ir stipresnius priešus, kuriuos reikia rasti ir nugalėti. Tai padarius, yra gaunamas prizas. Iš čia taip pat kilo mintis sukurti kokių nors galvosūkių, kurios žaidėjas turėtų išspręsti, norint atrasti „boso“ kambarį.

Šio darbo pagrindinis tikslas yra sukurti agentą, kuris būtų žaidėjo priešininkas, jį vis vytųsi ir bandytų sustabdyti žaidėjo bandymą pasiekti tikslą, naudojant įvairius puolimus.

Užduotys:

* Pateikti agento aprašą.
* Pateikti naudojamų algoritmų paaiškinimus.
* Suprojektuoti agentą.
* Sukurti žaidime naudojamą grafinį vaizdą.
* Sukurti žaidimą.
* Pateikti žaidimo instrukciją.

1. **INTELEKTIKA IR ŽAIDIMAI**

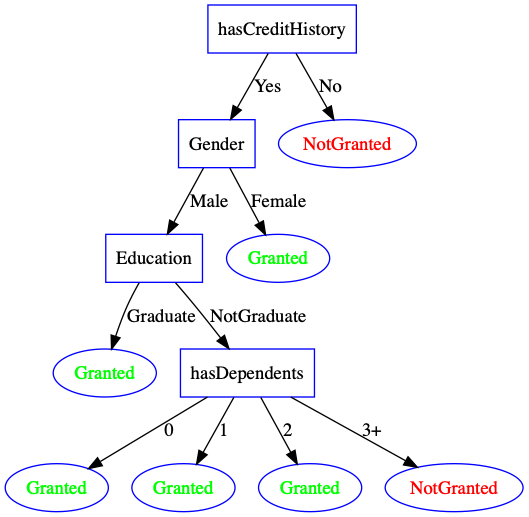
Intelektika, arba dažniau vadinamas dirbtinis intelektas, lengviausia suprasti kaip bandymą suteikti mašinoms galimybę priimti sprendimus, mokytis iš savo padarytų klaidų ir išspręsti įvairias užduotis. Tai yra bandymas atkurti žmogaus intelektualius gebėjimus įvairiose mašinose ar įrenginiuose.

Nors žodžiai „dirbtinis intelektas“ pirmą kartą buvo panaudotas net 1956 m., per paskutiniuosius 15 metų dirbtinis intelektas taip išpopuliarėjo, jog su juo galima susidurti beveik visuose savo gyvenimo aspektuose. Kompiuteriniai žaidimai yra vienas iš šių aspektų. Žaidimuose intelektika gali pasireikšti įvairiais būdais. Dažniausiai tai kokių nors uždavinių sprendimo užuominų davimas ar bandymas parodyti žaidėjui greičiausią būdą, kaip pasiekti norimą tašką. Intelektika žaidimuose taip pat pasireiškia veikėjų, kurių žaidėjas negali valdyti, forma. Šiuolaikiniuose žaidimuose šie veikėjai per visą žaidimo laikotarpį, priklausant nuo žaidėjo poelgių, keičiasi, mokosi. Tokiems rezultatams pasiekti buvo sukurta daug įvairių algoritmų.

* 1. **Pasirinkimų medžiai**

Vienas iš populiariausių būdų žaidime įgyvendinti intelektiką, tai naudojant pasirinkimų medžius. Tai yra algoritmas skirtas išspręsti įvairias klasifikavimo ar regresijos užduotis. Pasirinkimų medžiai, pagrinde, susideda iš šakninių mazgų, šakų, sprendimo mazgų ir lapų.

Šakninis mazgas yra medžio pati pradžia, jis neatkeliauja iš kokio nors kito mazgo ir yra padalinamas į ne vieną sprendimo mazgą. Sprendimo mazgas, arba dar kartais vadinama šaka, yra šakninio mazgo vaikas, kuris atspindi įvairius veiksmus, kurie gali būti atlikti, ar kokius nors pasirinkimus, kurie gali būti priimti agento. Lapas- tai pati žemiausia medžio dalis, kuri nurodo pasirinkimų rezultatą. Lapai yra toliau nebeišsišakojantys, pasirinkimų mazgų vaikai.



1 pav. Pavaizduotas pasirinkimų medis parodantis ar paskola bus duota.

Shape

Description automatically generatedPasirinkimų medžių įgyvendinimas žaidimuose, dažniausiai pasirodo priešų pavidalu. Tokiu atveju, pasirinkimų medis veikia tokiu principu: priešui reikia nuspręsti, kokį veiksmą atlikti, pirmiausia yra iškeliamas klausimas, ar žaidėjas yra matomas, jeigu ne, reikia vaikščioti aplinkui, kol jis bus surastas, jeigu taip- iškeliamas kitas klausimas, ar žaidėjas arti. Jeigu atsakymas yra ne, vėl iškeliamas klausimas, kiek sveikatos turi priešas priimantis šiuos sprendimus, jeigu mažai- laukti kitų priešų, jeigu daug, pulti žaidėją iš tolo. Esant situacijai kai žaidėjas jau buvo arti, jį pulti. Priešo pasirinkimo medis gali atrodyti taip:

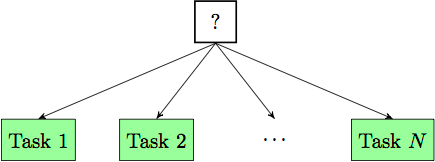
2 pav. Priešo veiksmų pasirinkimo medis

Viena iš priežasčių, kodėl pasirinkimų medžiai yra pakankamai dažnai naudojami žaidimuose yra dėl to, nes juos yra labai lengva suprasti ir pritaikyti. Jie gali būti naudojami norint sukurti priešą, kuris geba prisitaikyti prie esamos situacijos, arba bet kokį kitą veikėją, kuris nebus valdomas žaidėjo, bet reikia, kad šis galėtų elgtis skirtingai, priklausant nuo kokių nors sukurtų taisyklių ar nurodymų.

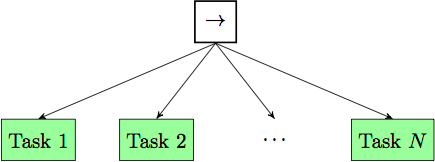
* 1. **Elgesio medžiai**

Nors ir pavadinimai su jau seniau minėtu algoritmu panašūs, vienintelį dalyką, ką jie turi bendro yra tai, jog jie yra pavaizduojami medžio, pavidalu. Šis algoritmas yra skirtas pateikti kokius veiksmus gali atlikti agentas, įvykus kokiam nors veiksmui.

Elgesio medžių veikimo principas yra pakankamai paprastas suprasti, per kiekvieną kadrą, sistema keliaus nuo medžio pačios viršūnės iki apačios, tikrinant kiekvieną mazgą. Visi mazgai gali sistemai grąžinti vieną iš trijų būsenų: vykdomas, sėkmė arba nesėkmė. Priklausant nuo grąžintos būsenos bus įvykdomas vienas ar kitas veiksmas.

Šio algoritmo struktūra taip pat prasideda nuo šakninio mazgo, po to jis yra išskirstomas į srauto valdymo mazgus, o šie į vykdymo mazgus. Srauto vykdymo mazgai gali būti vienas iš dviejų tipų, parinkiklio arba sekos mazgas. Priklausant nuo mazgo tipo, jo elgsena yra skirtinga. Esant parinkiklio mazgui, jis pasirenka patį pirmą savo turimą vykdymo mazgą ir tikrina ar tą veiksmą galima atlikti, jeigu ne, tikrina sekantį. Jis sustoja tikrinti ir grąžina sėkmės būseną tada, kai vienas iš vykdymo mazgų gražiną sėkmės būseną. Esant atvejui, kai nei vienas iš vaikų negrąžina sėkmės būsenos, mazgas yra laikomas nesėkme. Paprastumo dėlei, parinkiklio mazgus galima laikyti „ar“ sąlyga, kadangi šis mazgas teieško vieno vaikinio mazgo, kuris gražina sėkmę.

3 pav. Parinkėjo mazgo žymėjimas

Esant sekos mazgui, veikimo principas labai panašus, jis taip pat savo vaikinius mazgus iš eilės, iš kairės į dešinę, tikrina, tačiau skirtingai negu parinkiklio, norint jog mazgas grąžintų sėkmės būseną, reikia jog visi jo vaikiniai mazgai sugrįžtu sėkmingi. Dėl šios priežasties šį mazgo tipą, galima laikyti „ir“ sąlyga.

4 pav. Sekos mazgo žymėjimas

Kadangi abu mazgo tipai, vaikinius mazgus tikrina iš eilės yra labai svarbu, jog vaikiniai mazgai būtų sudėlioti iš eilės. Pavyzdžiui : aptinkamas žaidėjas, prieinama prie žaidėjo, puolama, o ne puolama, aptinkamas žaidėjas, prieinama prie žaidėjo.

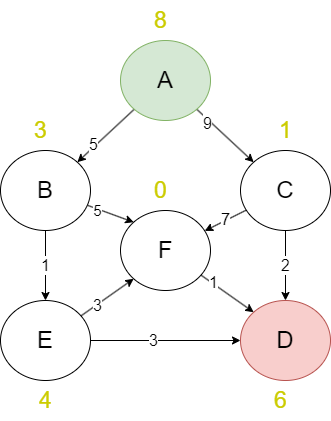
Štai kaip veikia šis medis: pirmiausia medis yra iškviečiamas, po to yra vykdoma patikra ar žaidėjas yra matomas, mazgas parenka pirmąjį savo vaiką, jeigu žaidėjas nėra matomas, mazgas grąžina sėkmės būseną ir veiksmas yra įvykdomas, o jeigu ne- mazgas parenka kitą savo vaiką. Kitas vaikas yra sekos mazgas, tad šis iš eilės parenka vaiką, kuris grąžina savo būseną, nuo kurios priklausant yra renkami arba nerenkami kiti sekos mazgo vaikai. Jeigu buvo grąžinta sėkmė, kitas vaikas, kuris yra parinkiklio mazgas, bus iškviestas. Šis mazgais taip pat iš eilės kvies savo vaikinius mazgus ir taip yra keliaujama medžiu vis žemiau, iki tol kol bus įvykdytas koks nors veiksmas, „attack“, „attack from a distance“, „do nothing“ ar „attack together“.

5 pav. Ankščiau pateiktas pasirinkimų medis, paverstas į elgesio medį

* 1. **A\* paieškos algoritmas**

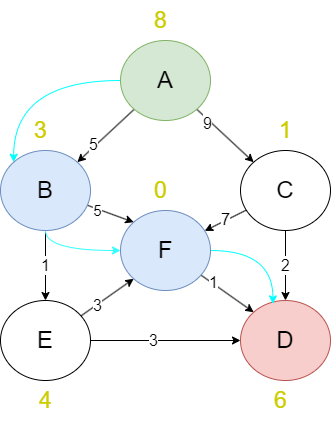
Tai yra dar vienas žaidimuose dažnai naudojamas algoritmas, kurio paskirtis yra surasti greičiausią kelią nuo taško A iki taško B. Norint rasti greičiausią kelią nuo vieno taško iki kito, galima naudoti įvairius algoritmus, pavyzdžiui, „Breadth First“ paiešką, kuris, norėdamas surasti greičiausią kelią, patikrina visus galimus kelius, arba Dijkstros sukurtas paieškos algoritmas, kurio pagerinta versija ir yra A\* paieška.

A\* viekimą lengviausiai galima paaiškinti naudojant svorinius grafus. Esant svoriniam grafui yra pasirenkama pradinė padėtis, dar kitaip vadinama pradiniu mazgu, ir norimą pasiekti tašką, galutinį mazgą. Algoritmas pritaiko trijų parametrų skaičiavimą norint atrasti greičiausią būdą pasiekti galutinį mazgą. Vienas iš šių parametrų yra g(n), kuris parodo kelio svorį, tai suprasti galima kaip, keik kainuos keliauti tuo keliu, pavyzdžiui nuo A iki C g(n) yra 9, o nuo A iki B, 5. Kitas parametras yra h(n), kuris parodo mazgo svorį, šią vertę galima suprasti taip: kiek kainuoja keliauti per tą mazgą. Pavyzdžiui C mazgo h(n) yra 1, o B- 3.



6 pav. Svorinis grafas, kurio pradinis mazgas yra A, o galutinis- D

Norint apskaičiuoti greičiausią kelią, naudojama formulė f(n)= g(n) +h(n). Pradedant nuo mazgo A yra du pasirinkimai, keliauti į mazgą C arba į mazgą B. Tam yra pritaikoma formulė, f(A-B)= 5+3 ir f(A-C)=9+1. Iš čia gauname atsakymus, jog A-B kelio visas svoris yra 8, o A-C yra 10. A\* algoritmas parenka toliau keliauti į B mazgą, nes 8<10. Norint toliau keliauti, vėl reikia apskaičiuoti, visos kelionės svorius B-F ir B-E keliams. f(B-F)=5+0 ir f(B-E)=1+4. Šiuo atveju, kadangi f(n) vertės gaunasi tos pačios, algoritmas nusprendžia į kurį mazgą keliauti pagal jų h(n) reikšmes. Jis keliaus į tą mazgą, kurio reikšmė yra mažesnė, taigi kadangi 0 yra mažiau už E, A\* pateks į F mazgą. Čia vėl pritaikomas algoritmas keliams F-C ir F-D. f(F-C)=1+7 ir f(F-D)= 6+1. F-D viso kelio svoris mažesnis tad ir pasirenkamas tas kelias. Taip pasiekiamas mazgas D. Nors pirmą kartą pažiūrėjęs asmuo gali teigti, jog greičiau būtų tiesiog keliauti A-C-D, tačiau pritaikius algoritmą, paaiškėja, jog greitesnis kelias iš tikrųjų yra A-B-F-D.



7 pav. Nuo A iki D mazgo greičiausias kelias

1. **AGENTO IR ŽAIDĖJO APRAŠAS**

Šiam darbui kuriamas agentas yra žaidėjo priešas. Jis gali būti arba šikšnosparnis arba šlykštukas (*angl. slime*). Jo pagrindinis darbas yra nugalėti žaidėją, stengiantis kuo mažiau nukentėti pačiam. Priklausant nuo aplink esančių kitų agentų ir nuo žaidėjo veiksmų, agentas gali bandyti pulti žaidėja arba nuo jo bėgti.

1 lentelė. Žaidimo agentas

8 pav. Žaidimo priešai



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agento tipas | Našumo rodiklis | Aplinka | Mechanizmas | Davikliai |
| Žaidėjo priešas | Žaidėjo nužudymas  Kuo mažesnis nukentėjimas nuo žaidėjo | Kiti žaidėjo priešai | Puolimas  Bėgimas | Akys |

Elgesio medžių dėka, šie agentai vaikščios aplink žemėlapį, o aptikus žaidėją, priklausant ar jis toli ar arti atliks kitus veiksmus. Jeigu žaidėjas bus arti, jį puls, o jeigu žaidėjas toliau nuo agento, priklausant nuo to, ar jo jėgos likutis yra didesnis negu 20 procentų visos jo galimos jėgos, vėl atliks skirtingus veiksmus. Jėgai esant mažiau negu 20 procentų pasitrauks toliau nuo žaidėjo, o esant daugiau, puls, bet tik tada, kai aplink bus kitų priešų, kurie taip pat puola žaidėją.

Žaidėjas, skirtingai negu priešas, bus kontroliuojamas žmogaus. Jį galima judinti į priekį, atgal, į abu šonus.



9 pav. Žaidėjo modelis

Shape

Description automatically generated with medium confidence Įprastai žaidimuose naudojamas ginklas yra kalavijas, tačiau šiame žaidime tai kirtiklis. Jį užsimojęs žaidėjas naudoja prieš įvairius priešus, norint juos nugalėti. Žaidėjui taip pat yra suteikiama galimybė sąveikauti su duobėmis, kurios bus vienintelis žaidėjo būdas patekti į kitą lygį. Pagrindinis veikėjas taip pat galės trumpai pakalbėti su vienu ne žaidėjo valdomu personažu, kurio pagrindinis darbas yra tiesiog žaidėjui perteikti informaciją, kaip laimėti žaidimą.

10 pav. Ne žaidėjo valdomas personažas

11 pav. Duobė

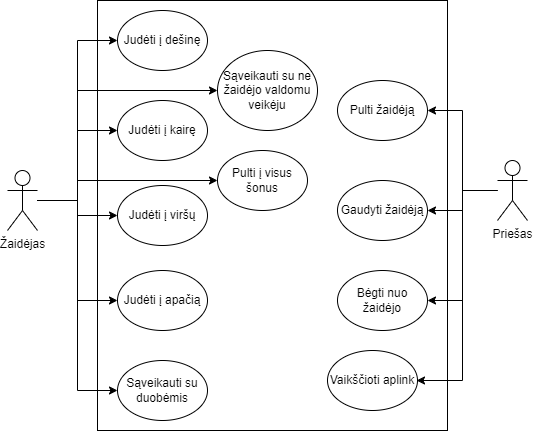


1. **FUNKCINIAI/NEFUNKCINIAI REIKALAVIMAI IR DIAGRAMOS**
   1. **Funkciniai ir nefunkciniai reikalavimai**

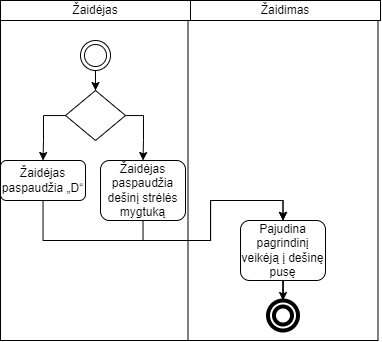
Funkciniai reikalavimai:

1. Žaidimas turi leisti žaidėjui judėti į dešinę
2. Žaidimas turi leisti žaidėjui judėti į kairę
3. Žaidimas turi leisti žaidėjui judėti į viršų
4. Žaidimas turi leisti žaidėjui judėti į apačią
5. Žaidimas turi leisti žaidėjui sąveikauti su duobėmis.
6. Žaidėjas turi leisti žaidėjui sąveikauti su ne žaidėjo valdomu veikėju.
7. Žaidimas turi leisti žaidėjui pulti į visus šonus
8. Žaidimas turi leisti priešui pulti žaidėją
9. Žaidimas turi leisti priešui gaudyti žaidėją
10. Žaidimas turi leisti priešui bėgti nuo žaidėjo

Nefunkciniai reikalavimai:

1. Privalo veikti stacionariuose ir nešiojamuose kompiuteriuose, kuriuose yra „Windows 10“
   1. **Diagramos**

12 pav. Panaudos atvejai

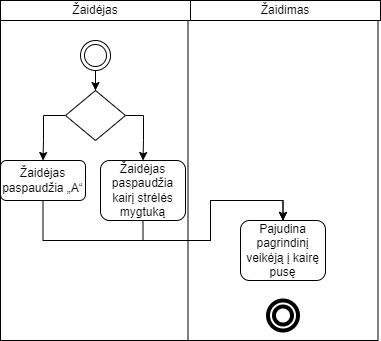
2 lent. „Judėti į dešinę“ aprašas

13 pav. „Judėti į dešinę“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Judėti į dešinę |
| **ID** | 1 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas judina pagrindinį veikėją į dešinę pusę |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Pagrindinis veikėjas yra vienoje vietoje |
| **Pagrindinis scenarijus** | 1. Naudotojas paspaudžia „D“ raidę 2. Žaidimas pajudina veikėją į nurodytą pusę |
| **Alternatyvūs scenarijai** | 1.a. Naudotojas paspaudžia dešinį strėlės mygtuką  1.a.1. Sistema pajudina veikėją į nurodytą pusę |
| **Po sąlygos** | Pagrindinis veikėjas pajudėjo į dešinę pusę |
| **Papildoma informacija** | - |

|  |  |
| --- | --- |
| **Pavadinimas** | Judėti į kairę |
| **ID** | 2 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas judina pagrindinį veikėją į kairę pusę |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Pagrindinis veikėjas yra vienoje vietoje |
| **Pagrindinis scenarijus** | 1. Naudotojas paspaudžia „A“ raidę 2. Žaidimas pajudina veikėją į nurodytą pusę |
| **Alternatyvūs scenarijai** | 1.a. Naudotojas paspaudžia kairį strėlės mygtuką  1.a.1. Sistema pajudina veikėją į nurodytą pusę |
| **Po sąlygos** | Pagrindinis veikėjas pajudėjo į kairę pusę |
| **Papildoma informacija** | - |

3 lent. „Judėti į kairę“ aprašas



14 pav. „Judėti į kairę“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Judėti į viršų |
| **ID** | 3 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas judina pagrindinį veikėją į viršų |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Pagrindinis veikėjas yra vienoje vietoje |
| **Pagrindinis scenarijus** | 1. Naudotojas paspaudžia „W“ raidę 2. Žaidimas pajudina veikėją į nurodytą pusę |
| **Alternatyvūs scenarijai** | 1.a. Naudotojas paspaudžia viršutinį strėlės mygtuką  1.a.1. Sistema pajudina veikėją į nurodytą pusę |
| **Po sąlygos** | Pagrindinis veikėjas pajudėjo į viršų |
| **Papildoma informacija** | - |

4 lent. „Judėti į viršų“

Diagram

Description automatically generated

15 pav. „Judėti į viršų“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Judėti į apačią |
| **ID** | 4 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas judina pagrindinį veikėją į apačią |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Pagrindinis veikėjas yra vienoje vietoje |
| **Pagrindinis scenarijus** | 1. Naudotojas paspaudžia „S“ raidę 2. Žaidimas pajudina veikėją į nurodytą pusę |
| **Alternatyvūs scenarijai** | 1.a. Naudotojas paspaudžia apatinį strėlės mygtuką  1.a.1. Sistema pajudina veikėją į nurodytą pusę |
| **Po sąlygos** | Pagrindinis veikėjas pajudėjo į apačią |
| **Papildoma informacija** | - |

5 lent. „Judėti į apačią“ aprašas

Diagram

Description automatically generated

16 pav. „Judėti į apačią“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Sąveikauti su duobėmis |
| **ID** | 5 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas gali atliktį kokį nors veiksmą su duobėmis |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Pagrindinis veikėjas yra viename lygyje |
| **Pagrindinis scenarijus** | 1. Naudotojas atsistoja ant duobės 2. Naudotojas paspaudžia tarpo klavišą 3. Žaidimas parodo lygių pasirinkimo ekraną |
| **Alternatyvūs scenarijai** | - |
| **Po sąlygos** | Žaidėjas pamato lygių pasirinkimo ekraną |
| **Papildoma informacija** | - |

6 lent. „Sąveikauti su duobėmis“ aprašas

Diagram

Description automatically generated

17 pav. „Sąveikauti su duobėmis“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Sąveikauti su ne žaidėjo valdomu veikėju |
| **ID** | 6 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas gali pasikalbėti su kitu žaidimo veikėju |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Žaidėjas nėra pasikalbėjęs su kitu žaidimo veikėju |
| **Pagrindinis scenarijus** | 1. Naudotojas prieina prie veikėjo. 2. Naudotojas paspaudžia tarpo klavišą 3. Žaidimas pradeda pokalbį tarp veikėjų |
| **Alternatyvūs scenarijai** | - |
| **Po sąlygos** | Žaidėjas pasikalbėjo su kitu žaidimo veikėju |
| **Papildoma informacija** | - |

7 lent. „Sąveikauti su ne žaidėjo valdomu veikėju“ aprašas

Diagram

Description automatically generated

18 pav. „Sąveikauti su ne žaidėjo valdomu veikėju“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Pulti į visus šonus |
| **ID** | 7 |
| **Trumpas aprašymas** | Procesas, kurio metu žaidėjas gali pulti priešus iš bet kurio šono |
| **Naudotojai** | Žaidėjas |
| **Prieš sąlyga** | Žaidėjas stovi priešais priešą |
| **Pagrindinis scenarijus** | 1. Žaidėjas prieina prie priešo 2. Žaidėjas paspaudžia kairį pelės mygtuką 3. Žaidimas parodo puolimo animaciją 4. Priešas yra nubloškiamas toliau nuo žaidėjo |
| **Alternatyvūs scenarijai** | - |
| **Po sąlygos** | Priešas buvo nublokštas toliau nuo žaidėjo |
| **Papildoma informacija** | - |

8 lent. „Pulti į visus šonus“ aprašas

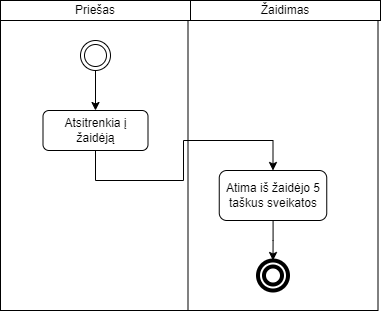
Diagram

Description automatically generated

19 pav. „Pulti į visus šonus“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Pulti žaidėją |
| **ID** | 8 |
| **Trumpas aprašymas** | Procesas, kurio metu priešas puola žaidėją |
| **Naudotojai** | Priešas |
| **Prieš sąlyga** | Priešas yra arti žaidėjo |
| **Pagrindinis scenarijus** | 1. Priešas atsitrenkia į žaidėją 2. Žaidimas atima iš žaidėjo 5 taškus sveikatos |
| **Alternatyvūs scenarijai** | - |
| **Po sąlygos** | Žaidėjas neteko šiek tiek sveikatos |
| **Papildoma informacija** | - |

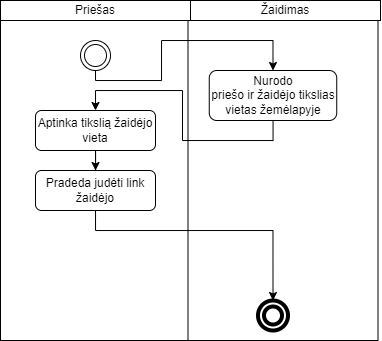
9 lent. „Pulti žaidėją“ aprašas



20 pav. „Pulti žaidėją“ UML

10 lent. „Gaudyti žaidėją“ aprašas

|  |  |
| --- | --- |
| **Pavadinimas** | Gaudyti žaidėją |
| **ID** | 9 |
| **Trumpas aprašymas** | Procesas, kurio metu priešas gaudo žaidėją |
| **Naudotojai** | Priešas |
| **Prieš sąlyga** | Priešas pamato žaidėją |
| **Pagrindinis scenarijus** | 1. Žaidimas nurodo priešo ir žaidėjo tikslias vietas žemėlapyje 2. Priešas aptinka tikslią žaidėjo vietą 3. Priešas pradeda judėti link žaidėjo |
| **Alternatyvūs scenarijai** | - |
| **Po sąlygos** | Priešas gali pulti žaidėją |
| **Papildoma informacija** | - |



21 pav. „Gaudyti žaidėją“ UML

|  |  |
| --- | --- |
| **Pavadinimas** | Bėgti nuo žaidėjo |
| **ID** | 10 |
| **Trumpas aprašymas** | Procesas, kurio metu priešas bėga nuo žaidėjo |
| **Naudotojai** | Priešas |
| **Prieš sąlyga** | Priešas buvo sužalotas |
| **Pagrindinis scenarijus** | 1. Priešas turi mažiau negu 40% savo sveikatos 2. Žaidimas nurodo priešo ir žaidėjo vietas žemėlapyje 3. Priešas aptinka į kurią pusę juda žaidėjas 4. Priešas juda priešinga kryptimi negu žaidėjas |
| **Alternatyvūs scenarijai** | - |
| **Po sąlygos** | Priešas pabėga nuo žaidėjo |
| **Papildoma informacija** | - |

11 lent. „Bėgti nuo žaidėjo aprašas“

Diagram

Description automatically generated

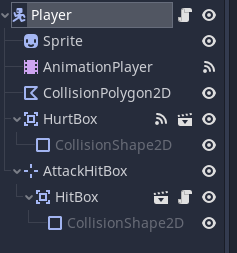
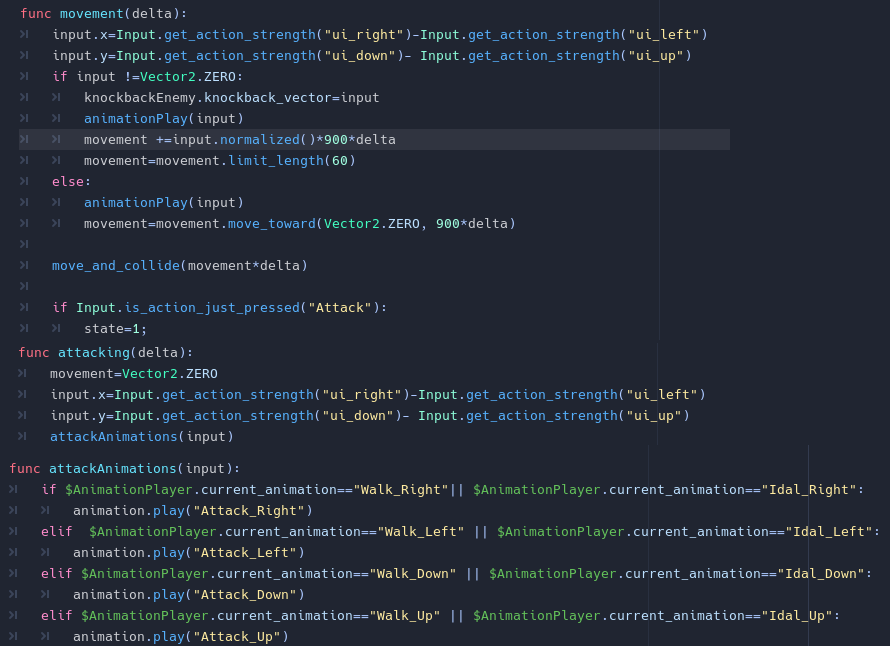
22 pav. „Bėgti nuo žaidėjo“

1. **PROGRAMINĖ REALIZACIJA**

Šiam žaidimui sukurti buvo naudojamas žaidimų variklis „Godot“ kartu su programavimo kalba „GDScript“. Kuriant žaidimui buvo naudojami 2D mazgai, scenos ir scenarijai (angl. Script). Norint sukurti vieną lygį, pirmiausia yra sukuriama nauja scena, jai priskiriamas šakninis mazgas, o mazgui- vaikiniai objektai ar elementai, kurie bus tame lygyje. Vaikiniai objektai gali būti kitos scenos su savo šakniniais ir vaikiniais elementais. Kiekvienam elementui ar objektui galima priskirti scenarijų, kuriame bus parašytas to daikto kodas.

23 pav. Pradinio lygio šakninis mazgas pavadinimu „World\_1“ su savo vaikiniais elementais ir objektais. „Player“, „MouseyBoi“, „Dialogue“ ir „Hole\_Enter“ yra atskirai sukurtos scenos su savo scenarijais ir šakniniais elementais, pridėtos prie „World\_1“ kaip vaikinės

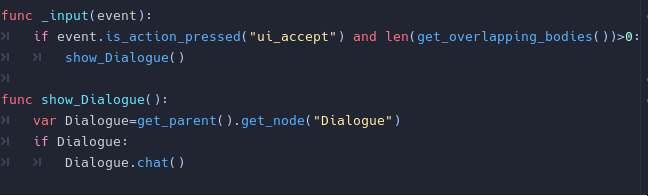
Pagrindinio veikėjo „Player“ scena susideda iš „Sprite“- žaidėjo paveiksliuko, „AnimationPlayer“- tai kas suteikia galimybę pritaikyti įvairias animacijas pagrindiniam veikėjui, „CollisionPlygon2D“- tai kas suteikia žaidėjui galimybę susidurti su sienomis ar priešais ir ant jų nelipti, „HurtBox“, kuris turi savo „CollisionShape2D“, šie du dalykai suteikia galimybę žaidėjui netekti jegos, kai jį puola priešas ir „AttackHitBox“ su savo vaikais „HitBox“ ir „CollisionShape2D“, kurie suteikia galimybę žaidėjui pulti priešą iš visų pusių.

Funkcija „ \_ready()“, įvykdoma kai tik objektas yra užkraunamas, joje yra nurodoma animacijos kintamojo reikšmė, pateikiama žaidėjo jėgos informacija į scenarijų, kuris yra atsakingas už šitos informacijos saugojimą ir pritaikyma kintamajam, kuris yra atsakingas už priešo judėjimą į tą pusę, kurią žiūri veikėjas, kai priešas yra sužeidžiamas. Taip pat yra „ animationPlay(input)“ ir „ attackAnimations (input)“ funkcijos, kurios atsakingos už veikėjo animacijų rodymą, priklausant nuo to, į kurią pusę veikėjas yra atsisukęs. „ attacking(delta)“ apskaičiuoja puolimo pusę, o „ \_on\_AnimationPlayer\_animation\_finished(anim\_name)“ nurodo, kokie veiksmai bus vygdomi, pasibaikus kuriai nors animacijai „ \_on\_HurtBox\_area\_entered(area)“ atima iš veikėjo jėgą, kai į nubrėžtą vietą patenka priešas. „ movement(delta)“ apskaičiuoja į kurią pusę turi judėti veikėjas, o „ \_physics\_process(delta)“ padeda valdyti vaikščiojimo ir puolimo būsenas, užtikrina, kad jos gali keistis.

25 pav. „\_ready()“, „animationPlay“ ir „\_physics\_process“ funkcijos

26 pav. „movement“, „attacking“ ir „attackAnimation“ funkcijos

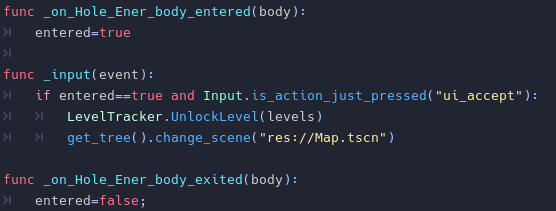
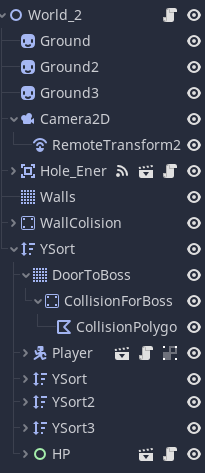
24 pav. „Player“ scena

„ MouseyBoi“ turi du elementus, kurie turi savo scenarijus. „Area2D“ patikrina ar žaidėjas yra atsistojęs tam tikroje zonoje ir ar jis paspaudė tarpo klavišą, jeigu taip, vykdoma funkcija „show\_Dialogue()“, kuri gauna duomenis iš „Dialogue“ objekto ir įvykdo funkciją „chat()“. Kitas „MouseyBoi“ elementas yra „Dialogue“, kurio scenarijuje yra aprašoma teksto gavimas iš .json failo, suteikiama galimybė paspaudus tarpo klavišą rodyti vis kitą sakinį ir vykdoma funkcija, kuri parodo pokalbį ekrane.

29 pav. „Dialogue“ funkcijos

28 pav. „Area2D“ funkcijos

27 pav. „\_on\_AnimationPlayer\_animation\_finished“ ir „\_on\_HurtBox\_are\_entered“ funkcijos

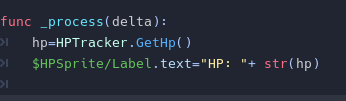
Taip pat yra scena „Hole\_Enter“, kuriam, priklausant nuo lygio, yra priskirti truputėlį besikeičiantys scenarijai. Visuose scenarijuose yra funkcijos „\_on\_Hole\_Ener\_body\_entered(body)“, „\_on\_Hole\_Ener\_body\_exited(body)“ ir „\_input(event)“. Pirmos dvi paminėtos funkcijos yra atsakingos už patikrą, kurios dėka yra nustatoma ar žaidėjas yra atsistojęs ant duobės ar ne, o „\_input“ atlieka patikrą, ar buvo žaidėjas atrastas prieš tai minėtoje funkcijoje ir jeigu taip, ar žaidėjas paspaudė tarpo klavišą. Jeigu į abu klausimus atsakymas yra taip, tai yra iškviečiamas lygių žemėlapio ekranas ir nusiunčiama atrakinto lygio informacija į scenarijų, kurio darbas yra saugoti tą informaciją. Visi šiam objektui priskiriami scenarijai tuo skiriasi, kad jie vis siunčia kitą lygių informaciją.

31 pav. „World\_2“ šakninis elementas su vaikiniais

30 pav. Vienas iš „Hole\_Enter“ scenarijų

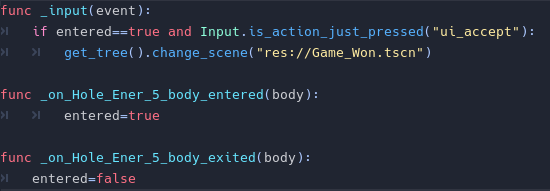
Šioje scenoje yra naudojami elementai kaip „Camera2D“, kurios paskirtis yra sekioti paskui veikėją, užtikrinti, kad žaidėjas mato lygį. „Ground“- tai tiesiog lygyje esanti žemė, o „Walls“ ir „ŽallCollision“, sienos, pro kurias žaidėjas negali praeiti. „YSort“, „YSort2“ ir „YSort3“ laiko šiame lygyje esančius priešų objektus, o „DoorToBoss“- parodo „boso“ duris ir užtikrina, kad žaidėjas nepatektų į „boso“ kambarį neužmušęs visų priešų.

Šio šakninio elemento scenarijuje yra patikrinama ar lygyje visi priešai yra užmušti ir jeigu taip, panaikina sieną, kuri užstoja iėjimą į „boso“ kambarį. Čia taip pat yra patikrinama ar „bosas“ yra užmuštas ir jeigu taip, parodoma duobė į kitą lygį.

Vienas iš šio pasaulio scenų yra „HP“, jos paskirtis yra ekrane pavaizduoti, kiek jėgos turi žaidėjas. Jos scenarijus gauną informaciją iš kito scenarijaus, kurio darbas yra laikyti bei atnaujinti informaciją apie žaidėjo jėgą.

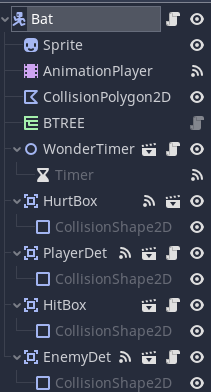
33 pav. Gaunama ir pavaizduojama žaidėjo sveikatos informacija

32 pav. „World\_2“ scenarijus

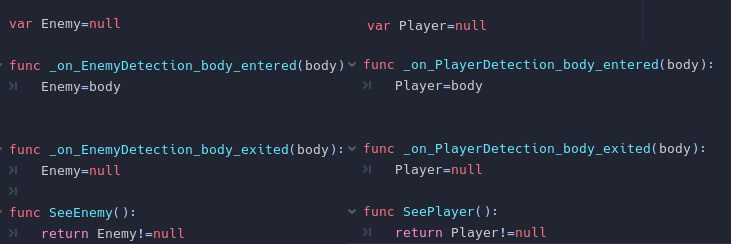
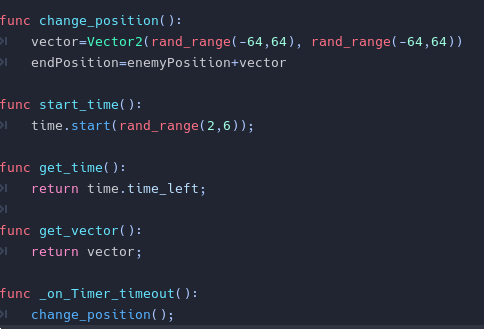
Beveik visų pasaulių scenarijai yra arba skiriasi tik vienu skaičiumi tikrinant ar visi priešai užmušti, tik pačio paskutinio lygio scenarijus neturi „boso“ mirties patikrinimo. Šio lygio „Hole\_Enter“ scenarijus taip pat skiriasi nuo kitų, nes vietoj lygio žemėlapio parodymo, jis parodo laimėjimo ekraną.

35 pav. „World\_5“ „Hole\_Enter“ scenarijus

34 pav. „World\_5“ scenarijus

Vienas iš priešų yra šišknosparnis. Šis objektas susideda iš element kaip „Sprite“- suteikia priešui paveiksliuką, „AnimatioPlayer“- leidžia priešui naudoti įvairias animacijas, „CollisionPolygon2D“, užtikdina, kad priešas nenuskristų ant sienų ar iš žemėlapio ribų, „BTREE“ leidžia įgivendinti elgesio medžio algoritmą, kurui dėka priešas įgauna pasirinkimų priėmimo galimybes, „WonderTimer“- nusako kas kiek laiko priešas, kai jis nežino kad kažkur yra žaidėjas, skraido aplinkui, „HurtBox“- nurodoma vieta, į kurią, jeigu žaidėjas puldamas pataiko, priešas netenka jėgos, „PlayerDetection“- suteikia galimybę matyti žaidėją, „HitBox“- į šią vietą žaidėjui patekus, jis neteks jėgos, „EnemyDetection“- suteikia galimybę matyti kitus priešus.

36 pav. „Bat“ priešo elementai

„EnemyDetection“ ir „PlayerDetection“ scenarijai labai panašūs. Skiriasi tik objektai kurių jie ieško.

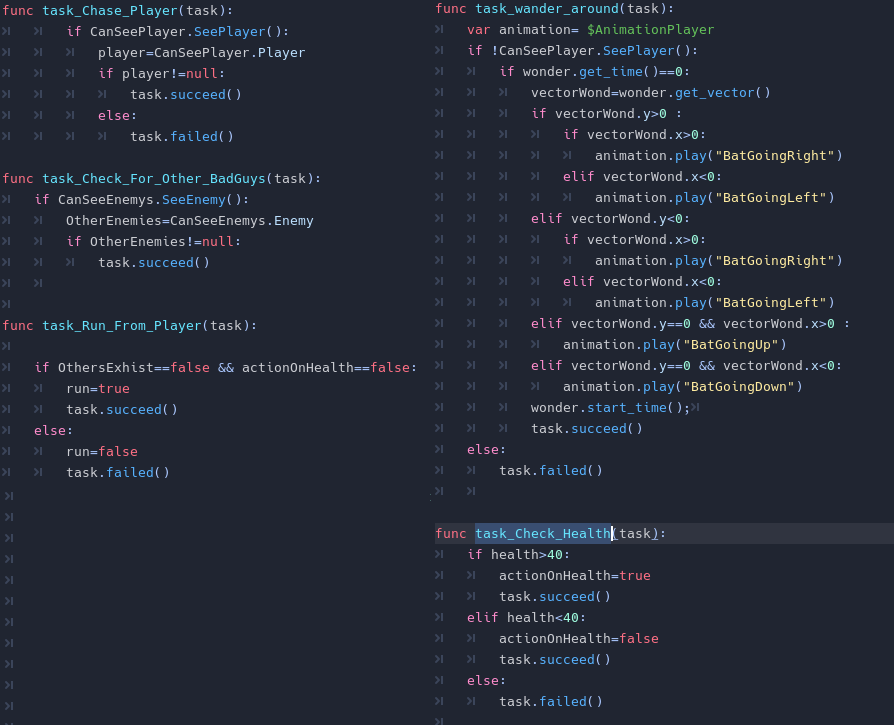
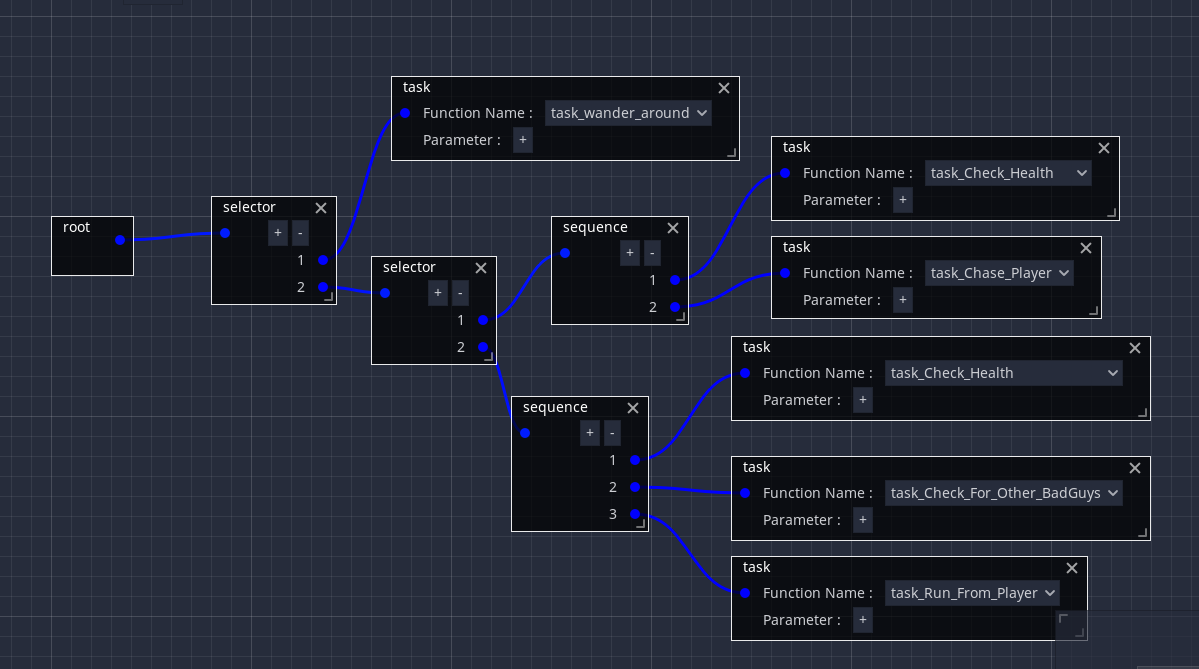
38 pav. „WonderTimer“ scenarijus, kuris generuoja priešo buvimą vienoje vietoje laiką

37 pav. Kito priešo aptykimas ir Žaidėjo aptykimas kodo palyginimas

Pats „Bat“ priešas turi daug funkcijų, „ \_physics\_process“ suteikia galimybę vaikščioti, „ \_on\_HurtBox\_area\_entered“ aptinka ar „Bat“ buvo sužeistas ir jeigu taip, iš jo atimti jėgą, „ \_on\_AnimationPlayer\_animation\_finished“ nurodo koks veiksmas turi eiti sustojus kokiai nors animacijai.

39 pav. „\_on\_HurtBox\_area\_entered“, „\_physics\_process“, „\_on\_AnimationPlayer\_animation\_finished“ funkcijos

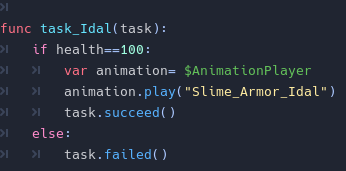
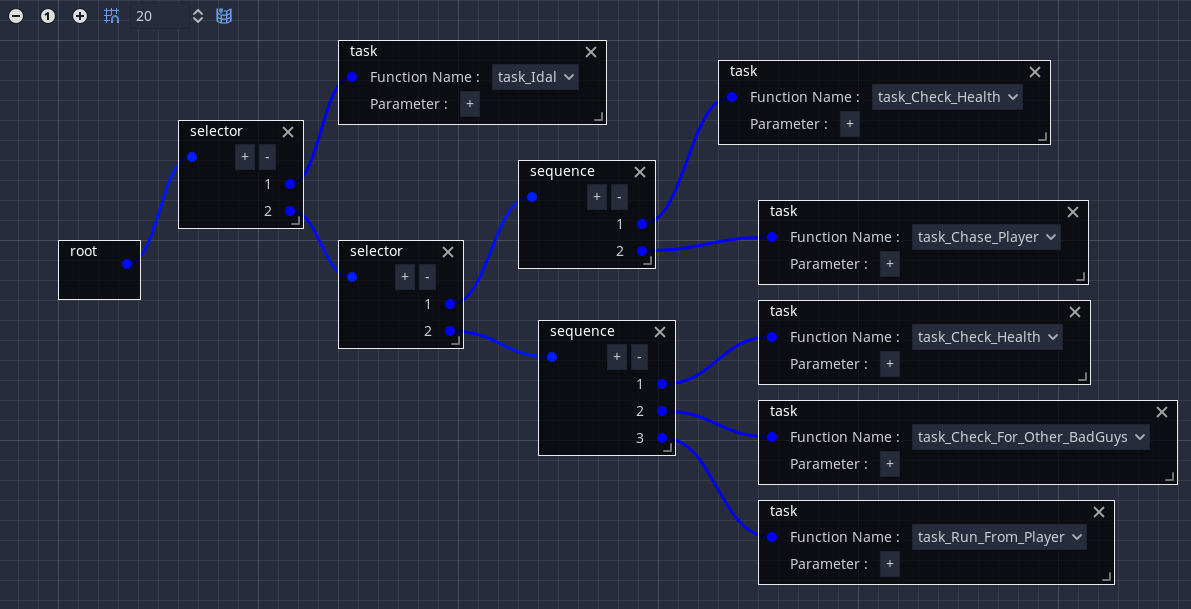
Kuriant priešus, buvo naudojamas elgesio medžio algoritmas. Norint nurodyti kokius veiksmus agentas turi atlikti, funkcijos turi prasidėti žodžiu „task“. „Bat“ agentas turi 5 „task“ funkcijas: „task\_Chase\_Player(task)“, „task\_Check\_For\_Other\_BadGuys(task)“, „task\_Run\_From\_Player(task)“, „task\_wander\_around(task)“ ir „task\_Check\_Health(task). Jų paskirtys yra gaudyti žaidėją, ieškoti kitų priešų, bėgti nuo žaidėjo, vaikščioti aplink ir tikrinti savo jėgą.

Kitas priešas yra šlykštukas arba „Slime“, jis turi tokius pat elementus kaip ir „Bat“ agentas, paprastos, „task“ funkcijos taip pat yra tos pačios. Elgiosio medis taip pat nesiskiria. Vieninteliai du skirtumai yra tas, jog šis priešas yra lėtesnis ir miršta greičiau negu „Bat“ priešas.

41 pav. „Bat“ agento elgesio medis

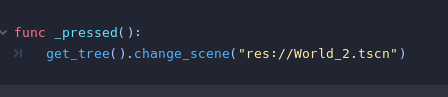
40 pav. „Bat“ agento „task“ funkcijos

Paskutinis iš priešų yra „Armor slime“, tai yra lėtesnis, sunkiau užmušamas „Slime preišas“. Jis skiriasi nuo kitų priešų tuo, kad jis prieš žaidėjo puolimą, jis atrodo kaip akmuo ir niekur nejuda. Vietoj to, kad turėtų „task\_ wander\_around“, jis turi „task\_Idal“ funkciją.

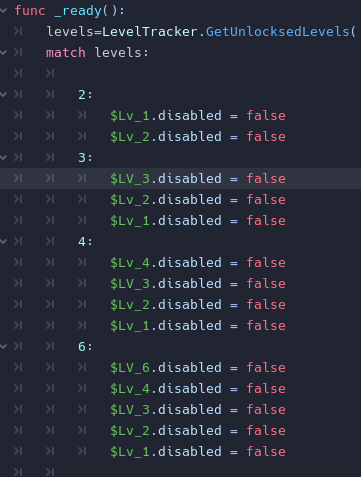
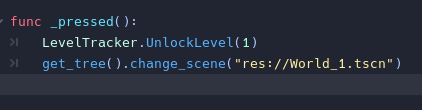
Žaidimui taip pat buvo sukurti 2 scenarijai, kurie nebuvo priskirti kokiam nors elementui, jie yra „HPTracker“ ir „LevelTracker“. Vieno iš jų paskirtis yra saugoti ir pateikti informaciją apie žaidėjo sveikatą, o kito saugoti ir pateikti informaciją, kiek lygių žaidėjas jau yra perėjęs.

43 pav. „Armor\_Slime“ elgesio medis

42 pav. „task\_Idal“ elgesio medžio funkcija

Viena iš scenų yra lygio pasirinkimas, jame yra 5 mygtukai, kiekvienas iš jų turi scenarijų, kurie yra panašūs. Skirtamas tarp jų yra tiesiog į kurį lygį mygtukas nuves.

44 pav. Antro mygtuko scenarijus. Ant mygtuko paspaudus, žaidėjas atsiras pirmajame lygyje

Jeigu žaidėjas netenka visos jėgos, žaidimas pasibaigia ir yra parodomas langas su mygtuku, kurį paspaudus žaidimas prasideda iš naujo.

46 pav. Vienintelė mygtuko funkcija, kuri nurodo, jog reikia užrakinti visus lygius ir vėl pradeda žaidimą iš naujo

45 pav. Vienintelė „Map“ scenos scenarijaus funkcija, kuri atrakina lygius, priklausant nuo skaičiaus, kurį gauna iš „LevelTracker“

1. **NAUDOTOJO INSTRUKCIJA**
   1. **Diegimo instrukcija**

Žaidimo įrašymas yra labai paprastas, tiesiog reikia du kartus spustelėti ant „CaveGame.exe“. Tai padarius, kartais iššoka įspėjimas ir yra paklausiama, ar tikrai norima leisti programai instaliuoti. Tuo atvėju kai šis langas pasirodo, reikia paspausti „accept“ arba „ok“.

* 1. **Naudojimas**

Norint kontroliuoti žaidėją, reikia spausti „WASD“ arba rodyklių klavišus. Tai padarius, veikėjas vaikščios. Norint pakalbėti su pirmame lygyje esančia pele- reikia prie jos prieiti paspausti tarpo klavišą. Norint patekti į pirmą lygį, reikia prieiti prie duobės ir paspausti tarpo klavišą.

48 pav. Stovėjimas prie duobės

47 pav. Kalbėjimas su pele

Paspaudę tarpo klavišą stovint šalia duobės, atsivers lygių pasirinkimo langas, lygiai yra atrakinami tik tada, kai lygis prieš tai yra pereinamas. Dabar yra atrakintas tik „Level1“ ir „Main cave“. Paspaudę ant „Main cave“ būsite sugrąžinti į pradinį lygį- kuriame yra pelė.

49 pav. Lygių pasirinkimo ekranas

Norint laimėti žaidimą, reikia pereiti visus lygius nemirštant, o norint pereiti lygį, reikia užmušti visus priešus, surasti „bosą“, jį nugalėti ir pasinaudoti duobe. Paspaudę ant „Level1“ atsirasite urve, kuriame yra priešai. Juos norint mušti, reikia prie jų prieiti ir į juos žiūrint, spausti kairį pelės mygtuką.

50 pav. Puolamas šikšnosparnis

Kai koks nors priešas netenka daugiau negu 40% savo jėgos ir aplink jį nėra kito priešo, kuris jam padėtų žaidėją nugalėti, jis bėga. Jeigu bebėgdamas suranda kitą žaidėjo priešą, ji pradeda pulti kartu.

53 pav. Šikšnosparnis susirado draugą, dabar jie puola kartu

52 pav. Šikšnosparnis bėga nuo žaidėjo

51 pav. Šikšnosparnis nukenčia

Užmušus visus priešus atsidaro „boso“ kambario durys.

54 pav. Prieš užmušant visus priešus



55 pav. Užmušus visus priešus

Įėjus į kambarį, reikia užmušti didelį priešą.

56 pav. Pirmo lygio bosas

Jį užmušus, kampe atsiranda duobė.

Priėjus prie jos ir paspaudus tarpo klavišą, galima pasirinkti „Level 2“. Naujame lygyje vėl reikia užmušti visus priešus, surasti „boso“ kambarį, užmušti „bosą“ ir judėti toliau. Jeigu sugebate pasiekti „Level 4“, šitame lygyje nėra „boso“, tačiau yra daugybė priešų, kurie mielu noru pasirūpins, jog jūs žaidimo nelaimėtumete.

58 pav. Žaidėjas bėga nuo priešų paskutiniame lygyje

57 pav. Duobė, kurią panaudojus galima patekti į kitą lygį

Jeigu žaidimą pralaimite, kus pasitiks „Game over“ langas, kuris suteikia galimybę, sužaisti žaidimą iš naujo.

Vis dėl to, jeigu laimėti pasiseka, jus pasitiks šis langas:

60 pav. Laimėjimo langas

59 pav. „Game over“ ekranas

**IŠVADOS**

Šiame darbe buvo pateiktas kuriamų agentų aprašas, pateikti įvairūs intelektikos algoritmai, sukurtas žaidimas bei pateikta jo instrukcija. Žaidimas tikrai nėra tobulas, tikrai toli nuo to, jį dar galima gerinti ir gerinti, išplėsti agentų elgesių medžius ar net pritaikyti kokį algoritmą, kurio pagalba žaidėjas galėtų matyti kure reikia eiti, norint rasti kitą priešą. Taip pat būtų galima pakeisti pačius „bosus“, suteikti jiems kokio nors individualumo.

**LITERATŪROS ŠALTINIAI**

1. B.J.Copeland „Artificial intelligence“ 2023m. Prieiga internetu: <https://www.britannica.com/technology/artificial-intelligence>
2. „AI- Path Finding“. Prieiga internetu: <https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/aitutorials/2pathfinding/AI%20-%20Simple%20Pathfinding.pdf>
3. [Nagesh Singh Chauhan](https://www.kdnuggets.com/author/nagesh-chauhan), KDnuggets „ Decision Tree Algorithm, Explained“ 2022m. Prieiga internetu: <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>
4. Debby Nirwan „ Designing AI Agents’ Behaviors with Behavior Trees“ 2020m. Prieiga internetu: <https://towardsdatascience.com/designing-ai-agents-behaviors-with-behavior-trees-b28aa1c3cf8a>
5. „ Decision Tree Introduction with example“ 2023m. Prieiga internetu: <https://www.geeksforgeeks.org/decision-tree-introduction-example/>
6. Jocelyn D’Souza „A Simple Guide to Decision Tree“ 2018m. Prieiga internetu:

<https://medium.com/@djocz/a-simple-beginning-to-decision-tree-2a1f992a3bec>

1. „Data Base Camp“ „What is a Decision Tree?“ 2022m. Prieiga internetu:

<https://databasecamp.de/en/ml/decision-trees#Structure_of_a_Decision_Tree>