

**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI  
POLITECHNIKI RZESZOWSKIEJ**

**Michał Majda**

Proceduralne generowanie map  
z wykorzystaniem w grze typu roguelike

**Praca dyplomowa inżynierska**

Opiekun pracy:  
dr hab. inż. Maciej Kusy, prof. PRz

Rzeszów, 2022



# Spis treści

<b>1. Charakterystyka porównawcza gier roguelike . . . . .</b>	<b>8</b>
1.1. Rogue . . . . .	8
1.2. Caves of Qud . . . . .	10
1.3. The Binding of Isaac . . . . .	13
<b>2. Szczegółowy opis gry . . . . .</b>	<b>17</b>
2.1. Główne założenia . . . . .	17
2.2. Opis menu głównego gry . . . . .	18
2.3. Sterowanie w grze . . . . .	18
2.4. Okno gry . . . . .	20
2.5. Opis poziomów . . . . .	21
2.5.1. Poziom 1 . . . . .	22
2.5.2. Poziom 2 . . . . .	22
2.5.3. Poziom 3 . . . . .	23
2.5.4. Poziom 4 . . . . .	24
2.5.5. Poziom 5 . . . . .	25
2.5.6. Poziom 6 . . . . .	25
2.6. Przedmioty oraz ich używanie . . . . .	25
2.6.1. Okna ekwipunku . . . . .	26
2.6.2. Zbroje i broń . . . . .	27
2.6.3. Mikstury . . . . .	27
2.6.4. Zwoje . . . . .	28
2.7. Tester proceduralnie generowanych poziomów . . . . .	29
<b>3. Technologie i implementacja . . . . .</b>	<b>32</b>
3.1. Technologie . . . . .	32
3.2. Implementacja . . . . .	33
3.2.1. Struktura projektu . . . . .	33
3.2.2. Architektura Entity Component System . . . . .	35
3.2.3. Pętla główna gry oraz jej stany . . . . .	40
3.2.4. System rozmieszczania przeciwników i przedmiotów na poziomach	42
3.3. Proceduralne generowanie poziomów . . . . .	44
3.3.1. Poziom pierwszy . . . . .	45

3.3.2.	Poziom drugi – Binary Space Partitioning	45
3.3.3.	Poziom trzeci – Cellular automata	48
3.3.4.	Poziom czwarty – Drunkard Walk	49
3.3.5.	Poziom piąty – Binary Space Partitioning	51
<b>Literatura</b>		<b>54</b>



# Wstęp

Roguelike to gatunek gier komputerowych posiadający elementy gier fabularnych RPG (role-playing game) [1], bazujący na dużej losowości rozgrywki [2]. Losowość w tych grach w głównej mierze opiera się na losowym rozmieszczeniu przeciwników i przedmiotów oraz losowo generowanych mapach. Inna ważną cechą gier Roguelike jest permadeath – śmierć gracza oznacza konieczność gry od początku. Klasyczne gry typu roguelike, z racji trudnej rozgrywki, nie były popularne, ale w ostatnich latach coraz popularniejsze są gry inspirujące się tym gatunkiem biorąc z niego tylko wybrane elementy. Z powodu standardowej dla tych gier słabej jakościowo grafiki oraz map tworzonych proceduralnie - to znaczy według algorytmów - zamiast ręcznie, gry te są stosunkowo proste w produkcji i nawet współcześnie możliwe do zrealizowania przez jedną lub dwie osoby. Nowsze gry posiadające elementy roguelike najczęściej rezygnują ze standardowego dla starszych tytułów systemu turowego oraz ruchu po kwadratowej siatce na rzecz gry w czasie rzeczywistym i swobodnego poruszania się. Czyni to nowych przedstawicieli gatunku bardziej przystępymi, co przekłada się na wzrost popularności [3].

Gatunek ten zapoczątkowany został przez grę Rogue w 1980 roku [4], od której wzięła się też nazwa gatunku roguelike - Rogue podobne. Rogue gracz eksploruje podziemia walcząc z przeciwnikami oraz zdobywając coraz lepszy ekwipunek. Celem ukończenia gry jest zdobycie amuletu Yendoru znajdującego się w najniższym poziomie. Grafika opatka jest o znaki tekstowe w kodowaniu ASCII [5], rozgrywka dzieje się w systemie turowym na kwadratowej siatce.

Mimo iż gatunek roguelike rozwinał się znacznie to wciąż powstają gry wierne klasycznym założeniom gatunku. Przykładem tego jest Caves of Qud z 2015 roku [6]. Gra ta posiada wszystkie cechy klasycznej rozgrywki, jednak standardową grafikę tekstową zastąpiono prostą grafiką typu *pixel art* [7].

Jednym z najbardziej popularnych przykładów nowoczesnych gier typu roguelike jest The Binding of Isaac z roku 2011 [8]. Z gatunku roguelike gra ta zaczerpnęła losowe generowanie map, przedmiotów i przeciwników oraz permadeath. W The Binding of Isaac rozgrywka dzieje się w czasie rzeczywistym, a ruch gracza nie jest ograniczony do siatki. Dzięki temu zabiegowi gra jest bardziej zręcznościowa i co za tym idzie - łatwiejsza.

Głównym celem pracy jest stworzenie gry z gatunku roguelike z proceduralnie generowanymi mapami, oraz omówienie użytych metod generacji map. Dodatkowym celem pracy jest omówienie najbardziej popularnych gier tego gatunku dostępnych w chwili obecnej na rynku.

Struktura pracy jest następująca: w rozdziale pierwszym dokonano szczegółowej charakterystyki porównawczej innych gier z gatunku roguelike: Rogue, Caves of Qud, The Binding of Isaac. Rozdział drugi opisuje grę roguelike stworzoną na potrzeby niniejszej pracy. W rozdziale trzecim zaprezentowano proces tworzenia tej gry oraz omówienie metod proceduralnego generowania map.

# 1. Charakterystyka porównawcza gier roguelike

W niniejszym rozdziale szczegółowo omówiono przykładowe gry z gatunku roguelike. Omówiono Rogue, Caves of Qud oraz The Binding of Isaac. Gatunek roguelike od swoich początków nie jest mocno popularny z powodu trudnej rozgrywki, skomplikowanego sterowania oraz prymitywnej oprawy graficznej. Powstały jednak popularne gry, które zaczepiają z gatunku tylko wybrane elementy, co w niektórych przypadkach oznacza płynną, prostszą rozgrywkę z elementami roguelike.

## 1.1. Rogue

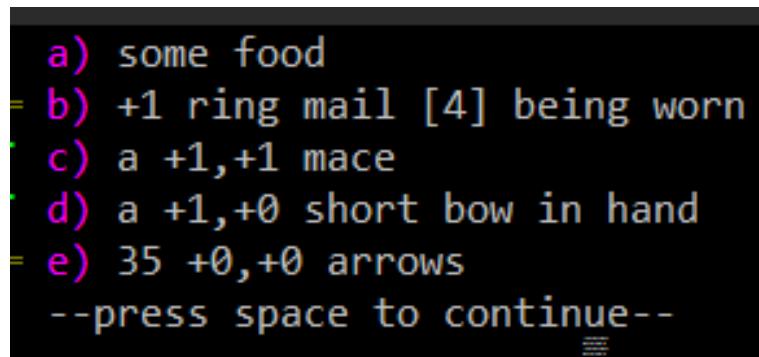
Rogue: Exploring the Dungeons of Doom to zaprogramowana w języku C gra pochodząca z 1980 roku, której twórcami są Michael Toy i Glenn Wichman. Gra ta zapoczątkowała gatunek Roguelike. Gracz wciela się w postać podróżującą w głąb podziemi osadzonych w fantastycznym, średniowiecznym świecie, w celu odnalezienia amuletu Yendoru. Rogue nie posiada wyboru i konfiguracji postaci – każdy gracz rozpoczyna grę tą samą postacią. W trakcie rozgrywki napotkać można wielu przeciwników, a w walce z nimi pomagają znajdowane na poziomach przedmioty i ekwipunek. Podczas eksploracji głębszych poziomów podziemi gracz napotyka trudniejszych przeciwników oraz lepsze przedmioty. Gra Rogue oparta jest o system turowy – gracz i jego przeciwnicy naprzemienne wykonują swoje ruchy. W wyniku tego gra pozwala na dowolnie długie przemyślenie każdego ruchu i taktyczne podejście do walki. Każda mapa przedstawiona jest za pomocą siatki kwadratów, dlatego też ruch gracza ograniczony jest do 8 kierunków: góra, dół, lewo, prawo i ukosy.

Z racji ograniczeń sprzętowych w czasach wydania gry Rogue za reprezentację graficzną odpowiadają litery i znaki ASCII w terminalu. Na rysunku 1.1 przedstawiono fragment rozgrywki w Rogue. Pomarańczowymi liniami oznaczone są ściany pokój, zielonymi kropkami podłogi a na szaro oznaczono korytarze. Żółta twarz reprezentuje postać gracza a przeciwnicy są oznaczani literami.



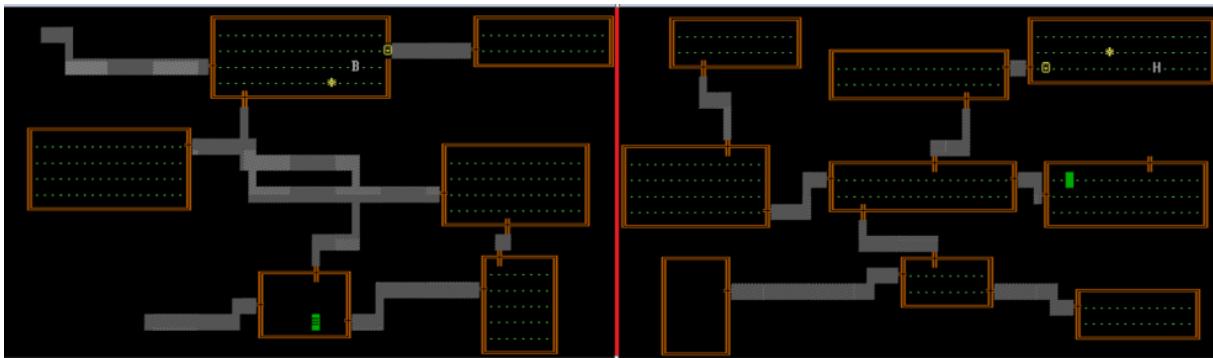
Rysunek 1.1: Widok główny gry Rogue

Do sterowania w Rogue używana jest tylko klawiatura. Do poruszania przeznaczone są klawisze klawiatury numerycznej a do wykonywania akcji litery (na przykład klawisz 'I' służy do wyświetlenia ekwipunku). Aby zaatakować gracz musi poruszyć się w stronę przeciwnika będąc tuż przy nim. Wyświetlanie ekwipunku w grze Rogue pokazano na rysunku 1.2. Manipulowanie przedmiotami w plecaku nie odbywa się w tym oknie, ale w głównym widoku gry za pomocą odpowiednich klawiszy odpowiadającym danemu przedmiotowi w ekwipunku. Przykładowo w celu wyrzucenia przedmiotu 'a +1, +1 mace' znajdującego się w ekwipunku na rysunku 1.2 trzeba wcisnąć klawisz 'd' odpowiadający za wyrzucanie przedmiotów i literę 'c' przypisaną obecnie do tego przedmiotu.



Rysunek 1.2: Ekran ekwipunku w Rogue

Rogue zawiera proste proceduralnie generowane mapy, przykładowe dwie mapy pokazano na rysunku: 1.3. Każda z map zawiera pokoje w układzie siatki 3x3, co szczególnie widać na mapie po prawej na rysunku 1.3. Maksymalna liczba pokojów to dziewięć, ale może być ich mniej. Pokoje są połączone ze sobą korytarzami, a część przejść może być ukryta przed graczem, co wymusza częste korzystanie z komendy przeszukiwania dostępnej pod klawiszem 'S'. Pokoje i korytarze mogą zawierać ukryte pułapki, które można wykryć używając odpowiedniej komendy. Jedyną zmianą na głębszych poziomach są małe labirynty zastępujące wybrane pokoje.



Rysunek 1.3: Przykładowe mapy w grze Rogue

Z racji sporego wieku gry Rogue ciężko jest określić jej popularność, jednak z pewnością można stwierdzić, że jest to kluczowa dla gatunku roguelike gra, która stworzyła podwaliny jego głównych cech. Świadczy o tym między innymi sama nazwa gatunku roguelike, która oznacza "podobne do Rogue".

## 1.2. Caves of Qud

Caves of Qud to współczesny przykład rozwoju klasycznej formuły roguelike. Gra tworzona w silniku Unity przez studio Freehold Games wydana została w 2015 roku, lecz rozwijana jest do dzisiaj (stan na luty 2022). Gra posiada wszystkie cechy klasycznej gry roguelike: działa w systemie turowym, posiada permadeath oraz mapa gry oparta jest o siatkę kwadratów. Twórcy dodali wiele nowoczesnych rozwiązań i znacznie rozwinięli wiele mechanik gry. Caves of Qud osadzone jest w postapokaliptycznym świecie, w którym zaawansowana technologia miesza się z fantastycznymi rasami oraz magią.

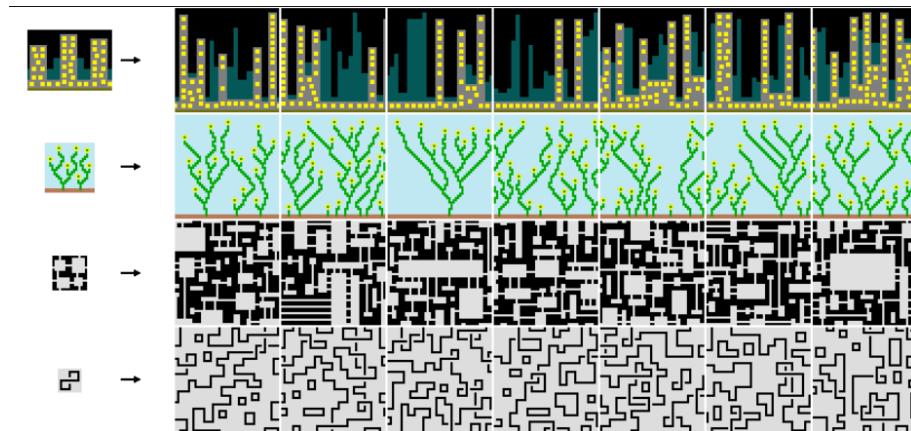
Jednym z głównych wyróżników Caves of Qud na tle innych gier tego typu jest znacząco rozwinięty system proceduralnie generowanych map. Gra posiada otwarty

świat o statycznie umiejscowionych obszarach, takich jak góry, pustynie oraz miasta. Na rysunku 1.4 przedstawiono przykładowe mapy wygenerowane w tej grze. Pomimo statycznego ustawienia konkretnych obszarów w grze, sam wygląd map jest proceduralnie generowany. Na wielu mapach mogą także zostać dodane obiekty takie jak ruiny lub obozowiska.



Rysunek 1.4: Przykładowe mapy w grze Caves of Qud. Po lewej wioska, po prawej dżungla z ruinami

Główna i najbardziej interesującą metodą używaną w tej grze jest generowanie map oparte o Wave Function Collapse (Załamanie funkcji falowej) [9]. Metoda ta pozwala na generowanie wielu różnych typów map za pomocą jednego algorytmu. Działanie Wave Function Collapse wykorzystuje wzorzec, na podstawie którego tworzy większy wzór lokalnie podobny do oryginalnego, co przedstawiono na rysunku 1.5 [10].



Rysunek 1.5: Przykład działania algorytmu opartego o Wave Function Collapse

Podobnie jak w grze Rogue, tutaj również do poruszania się używana jest klawiatura numeryczna oraz klawisze liter do wykonywania akcji. Caves of Qud umożliwia również granie wyłącznie za pomocą myszki. Gra oferuje bardziej przejrzystą grafikę typu *pixel art* w formacie 16x24 piksele na jeden kwadrat na mapie, co przedstawiono

na rysunku 1.6. Mylącym może być, iż gra jest reprezentowana graficznie przez siatkę pionowych prostokątów, lecz w rzeczywistości traktuje je jako kwadraty – ruch w każdym z ośmiu dostępnych kierunków zajmuje tyle samo czasu.



Rysunek 1.6: Zrzut ekranu z gry Caves of Qud

Caves of Qud znacznie rozwinęło interfejs użytkownika, co znaczaco wpływa na ułatwienie rozgrywki. Interfejs jest tu przejrzysty i prosty, dzięki czemu zmiana ekwipunku oraz używanie przedmiotów jest dużo łatwiejsze. Na rysunku 1.7 przedstawiono ekran ekwipunku.



Rysunek 1.7: Ekran ekwipunku w Caves of Qud

Interesującym rozwiązaniem w Caves of Qud jest system handlowy. W grze istnieje system głodu i pragnienia, gracz więc w celu uniknięcia śmierci musi nosić ze sobą zapasy wody oraz pożywienia. Zapasy te mają swoją wagę, więc można ich posiadać tylko ograniczoną liczbę. Walutą w tej grze jest woda pitna, która jest bardzo rzadko

spotykana w postaci źródeł. Zdecydowana większość wody w zbiornikach naturalnych jest niezdatna do picia. Z tego powodu gracz zbiera wodę nie tylko dla zaspokojenia pragnienia, ale również w celu handlu ze spotykanymi handlarzami.

Gra Caves of Qud znacznie rozwija aspekt RPG – odgrywania postaci. Dostępne są dwie rasy: mutant mający dostęp do fizycznych lub mentalnych mutacji oraz 'True Kin' będący niezmutowanym człowiekiem, który ma dostęp do cybernetycznych implantów. Poza typem wybiera się też pochodzenie postaci określające początkowy ekwipunek oraz bonusy do atrybutów. W trakcie gry gracz może dowolnie rozwijać swojego bohatera zwiększając wartości atrybutów takich jak siła, zręczność oraz inteligencja. Bohater uczy się też nowych umiejętności jak specjalizacja w posługiwaniu się konkretnym typem broni lub unikanie ataków.

W Caves of Qud proceduralne generowanie używane jest nie tylko do generowania map. W grze można znaleźć duże ilości proceduralnie tworzonych książek, z których część dotyczy także samej fabuły gry [11]. Oznacza to, że poza niektórymi, stałymi we wszystkich rozgrywkach, aspektami historii również część fabuły będzie inna w każdej nowej rozgrywce.

W odróżnieniu od większości gier tego typu, Caves of Qud posiada mocno rozwiniętą historię i rozbudowane główne zadania fabularne. Poza główną linią zadań gra oferuje też sporo losowych, które jednak są dosyć proste i najczęściej sprowadzają się do zdobycia konkretnego przedmiotu. Główne zadania są różnorodne, często polegają na odwiedzaniu fabularnych, rozbudowanych lokacji, obronie miasta przed atakiem lub nawet rozwiązywaniu zagadek.

Klasyczne gry typu roguelike posiadające zwykle dość wysoki poziom trudności rozgrywki wciąż uznawane są za dość niszowy gatunek. Poszczególne gry zyskują jednak relatywnie dużą popularność, czego przykładem jest gra Caves Qud, która w serwisie Steam posiada 4356 opinii z czego 95% jest pozytywnych (stan na luty 2022) [12].

### 1.3. The Binding of Isaac

Gra The Binding of Isaac została stworzona w roku 2011 przez Edmunda McMillena i Floriana Himsla, a następnie w odświeżonej wersji jako The Binding of Isaac: Rebirth w roku 2014. W niniejszej pracy skupiono się na omówieniu The Binding of Isaac: Rebirth, gdyż jest to nowsza i bardziej popularna odsłona. Gra ta jest przykła-

dem wzorowania się na gatunku roguelike, ale odejściu od części klasycznych założeń. Z gatunku zaczerpnięto losowo generowane mapy, losowe rozmieszczanie przedmiotów i przeciwników, oraz konieczność rozpoczęcia gry od początku w przypadku śmierci bohatera. Rozgrywka w The Binding Isaac dzieje się w czasie rzeczywistym, a ruch postaci nie jest ograniczony do siatki kwadratów. Z tego powodu gra jest bardziej dynamiczna, a walka zręcznościowa.

The Binding of Isaac odeszło od podziału mapy na kwadratową siatkę, ale wciąż korzysta z względnie prostej grafiki typu *pixel art*, co przedstawiono na rysunku: 1.8.



Rysunek 1.8: Przykład rozgrywki w The Binding of Isaac: Rebirth

Gracz wciela się początkowo w postać Isaaca, ale w trakcie rozgrywki może odblokować wiele nowych postaci, które różnią się początkowymi atrybutami – zdrowie, prędkość i siła ataku oraz posiadanymi przedmiotami. Gra nie posiada typowego ekwipunku takiego jak zbroja i broń, zamiast tego w grze zbiera się nieograniczoną liczbę przedmiotów, które zapewniają pasywne bonusy. Bonusy te wzmacniają postać na wiele różnych sposobów, od podstawowych zwiększających zdrowie lub atak po takie, które zmieniają pociski wstrzeliwane przez gracza na zupełnie inną postać. Jednym z głównym atutów The Binding of Isaac są możliwe kombinacje przedmiotów. Przykładowo jeśli gracz zdobędzie przedmiot, dzięki któremu pociski postaci będą się same kierowały na przeciwników, a następnie przedmiot, który zamienia pociski na promień lasera, to ten promień będzie zginany tak, aby zawsze trafiać przeciwników bez konieczności celowania, co pokazano na rysunku 1.9.



Rysunek 1.9: Przykład kombinacji przedmiotów w The Binding of Isaac: Rebirth

W grze The Binding of Isaac za sterowanie postacią odpowiadają klawisze 'W', 'A', 'S', 'D', a za strzelanie klawisze strzałek, do używania przedmiotów jest też używana spacja i klawisz 'Q'. Proste sterowanie jest kolejnym powodem przystępności tej gry.

W porównaniu do innych gier roguelike losowo generowane mapy w The Binding of Isaac są znacznie prostsze. Gra składa się z wielu poziomów podzielonych na pokoje, każdy z poziomów zakończony jest walką z silnym przeciwnikiem. Pokoje są wybierane losowo z puli przygotowanej przez twórców, a ich rozkład i połączenia są losowo generowane. Na rysunku 1.10 przedstawiono przykładowy układ poziomów. Z każdym kolejnym poziomem liczba pokojów jest coraz większa, ale jakość napotykanych przedmiotów nie wzrasta. Przedmioty są kompletnie losowe, dlatego czasem już nawet po pierwszym poziomie postać gracza może posiadać kilka najsilniejszych przedmiotów.



Rysunek 1.10: Przykładowe układy poziomów w The Binding of Isaac: Rebirth

Jedną z cech gier gatunku roguelike, które implementuje The Binding of Isaac jest permadeath. W porównaniu do innych tytułów, na których ukończenie trzeba

często poświęcić parę godzin, ukończenie The Binding of Isaac sprawnemu graczowi zajmie około godzinę. W wyniku tego rozpoczęcie rozgrywki od początku nie jest tak dotkliwe i zachęca do częstego, powtórnego przechodzenia gry.

The Binding of Isaac: Rebirth jest jedną z najbardziej popularnych gier z gatunku roguelike. Na serwisie Steam posiada 166244 opinii, w tym 97% pozytywnych (stan na luty 2022) [13]. Dowodzi to, że przeciętny gracz woli dynamiczną, rozgrywaną w czasie rzeczywistym rozgrywkę, od wolnej i taktycznej rozgrywki turowej. Kolejnym dowodem popularności tej gry jest wydana w 2018 roku gra planszowa The Binding of Isaac: Four Souls, która w tydzień zebrała ponad milion dolarów na serwisie Kickstarter [14].

## **2. Szczegółowy opis gry**

W tym rozdziale przedstawiono wszystkie elementy i możliwości gry stworzonej na potrzeby niniejszej pracy.

### **2.1. Główne założenia**

Zrealizowana na potrzeby niniejszej pracy gra spełnia podstawowe cechy klasycznych rozgrywek z gatunku roguelike:

- oparta jest o system turowy,
- zawiera plansze generowane proceduralnie,
- rozmieszczenie przeciwników i przedmiotów jest losowe,
- mapy są w postaci siatki kwadratów, ruch gracza ograniczony jest do poruszania się po nich,
- bohater posiada ekwipunek, który może założyć – zbroje i broń,
- gra posiada przedmioty, których można użyć – mikstury i magiczne zwoje,
- permadeath - śmierć kończy rozgrywkę, brak możliwości zapisu i wczytania stanu rozgrywki.

Projekt gry posiada też kilka bardziej nowoczesnych aspektów:

- grafika typu *pixel art*,
- rozwinięty, prostszy w użytkowaniu interfejs,
- system prezentowania z krokami proceduralnego generowania map.

Gra składa się z sześciu różnorodnych, proceduralnie generowanych poziomów. Gracz napotyka coraz trudniejszych przeciwników w kolejnych poziomach, a także znajduje coraz lepszy ekwipunek i przedmioty. Celem gry jest pokonanie ostatniego przeciwnika – potężnego szlamu, znajdującego się na ostatnim poziomie. Aby ukończyć grę gracz powinien eksplorować kolejne poziomy w celu odnalezienia silniejszych przedmiotów i ekwipunku niezbędnych przy walce z trudniejszymi przeciwnikami. Gra osadzona jest w fantastycznym, średniowiecznym świecie, w którym spotkać można rycerzy, gobliny i orków, a także magiczne mikstury i zwoje.

## 2.2. Opis menu głównego gry

Po uruchomieniu gry pierwsze, co zobaczy gracz to główne menu przedstawione na rysunku 2.1. Menu składa się z żółtego tytułu gry, oraz czterech opcji. Poruszać się po menu można za pomocą klawiszy strzałek, aktualnie wybrana opcja jest zaznaczona na zielono, a wybór opcji przeprowadzany jest klawiszem enter.



Rysunek 2.1: Menu główne gry

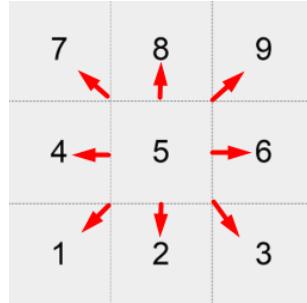
Menu składa się z następujących opcji:

- 'New Game' – rozpoczęcie nowej gry,
- 'Test Map Generators' – otwiera menu testera proceduralnego generowania map,
- 'Controls' – otwiera listę komend dostępnych w grze,
- 'Quit' – zamknięcie programu.

## 2.3. Sterowanie w grze

Główną metodą poruszania się postaci jest klawiatura numeryczna, co używane jest od dawna w gatunku roguelike. Jest to także najwygodniejsza kombinacja klawiszy przy możliwości ruchu w ośmiu kierunkach – dół, góra, prawo, lewo oraz skosy, schemat poruszania się przy użyciu klawiatury numerycznej przedstawiono na rysunku 2.2. Nie wszystkie klawiatury, a szczególnie te w laptopach, posiadają klawiaturę numeryczną, dlatego gra posiada też alternatywne sterowanie oparte o klawisze strzałek i klawisze liter. Klawisz '5' w przypadku klawiatury numerycznej używany jest do pomijania tury, co jest istotną możliwością w grze turowej. Jednokrotne naciśnięcie klawisza

odpowiedającego za ruch porusza postać gracza o jedno pole w danym kierunku, jeżeli nie jest ono zajmowane przez ścianę. Jeśli pole na które porusza się gracz zajmowane jest przez przeciwnika, to zamiast ruchu postać gracza wykona atak przeciwko temu przeciwnikowi, jest to standardowa metoda ataku w tego typu grach.



Rysunek 2.2: Schemat sterowania na klawiaturze numerycznej

Objaśnienia sterowania i lista dostępnych akcji w grze z menu głównego za pomocą opcji 'Controls' na rysunku 2.3 przedstawiono widok dostępny po wybraniu tej opcji.



Rysunek 2.3: Objaśnienia sterowania w grze

Poniżej wyjaśnienie wszystkich komend dostępnych w grze w kolejności z rysunku 2.3, Oznaczenie NumpadX oznacza klawisz cyfry X z klawiatury numerycznej. Oznaczenie 'X' oznacza klawisz litery lub znaku X:

- 1) Numpad8 / klawisz strzałki w góre – poruszenie się w góre,

- 2) Numpad2 / klawisz strzałki w dół – poruszenie się w dół,
- 3) Numpad4 / klawisz strzałki w lewo – poruszenie się w lewo,
- 4) Numpad6 / klawisz strzałki w prawo – poruszenie się w prawo,
- 5) Numpad7 / 'O' – poruszenie się na ukos – góra, lewo,
- 6) Numpad9 / 'P' – poruszenie się na ukos – góra, prawo,
- 7) Numpad1 / 'K' – poruszenie się na ukos – dół, lewo,
- 8) Numpad3 / 'L' – poruszenie się na ukos – dół, prawo,
- 9) Numpad5 / 'W' – pominięcie jednej tury,
- 10) ',' (klawisz przecinka) – zejście w dół po schodach,
- 11) '.' (klawisz kropki) – wejście w górę po schodach,
- 12) 'G' – podniesienie przedmiotu,
- 13) 'I' – otwarcie menu ekwipunku,
- 14) 'E' – pokazuje obecnie wyekwipowane przedmioty,
- 15) klawisz Enter – wybór opcji w menu,
- 16) klawisz Esc – powrót do menu głównego, kończy to obecną rozgrywkę,
- 17) klawiszem Enter powrócić można z menu ‘Controls’ do menu głównego.

## 2.4. Okno gry

Po rozpoczęciu nowej gry gracz zobaczy okno z wycinkiem mapy wycentrowanym na postać gracza, a także okno dziennika zdarzeń i pasek punktów życia, co pokazano na rysunku 2.4. Widok mapy jest powiększonym wycinkiem poziomu, przesuwającym się razem z graczem. Na mapie poza ścianami zaznaczone są też pola podłogi za pomocą białej kropki. Jest to przydatne do przeliczania odległości w grach opierających się o ruch na siatce kwadratów. W dzienniku zdarzeń zapisywane są wydarzenia z gry takie jak podnoszenie przedmiotu i ataki gracza lub przeciwników. Pasek życia posiada liczbową prezentację punktów życia w postaci [aktualne punkty

zdrowia / maksymalne punkty zdrowia], oraz graficzną reprezentację – czerwony pasek zmniejszający się wraz ze spadkiem zdrowia. Gracz posiada pole widzenia o określonym zasięgu, które działa w sposób realistyczny to znaczy będąc blisko wejścia do pokoju jest on w stanie zobaczyć tylko jego skrawek, a nie cały pokój, co widać na rysunku 2.4. Szarym kolorem zaznaczone są zapamiętane, odwiedzone wcześniej obszary, które aktualnie nie znajdują się w bezpośrednim polu widzenia bohatera. Na obszarach, które gracz pamięta, ale nie widzi ich obecnie zaobserwować można tylko układ podłóg i ścian. Przeciwników i przedmioty zauważać można tylko w bezpośrednim polu widzenia.



Rysunek 2.4: Widok główny gry. 1 - aktualny wycinek mapy, 2 - dziennik zdarzeń, 3 - aktualny poziom, 4 - pasek zdrowia. Żółta postać to gracz, a zielone postacie to gobliny – jeden z typów przeciwników

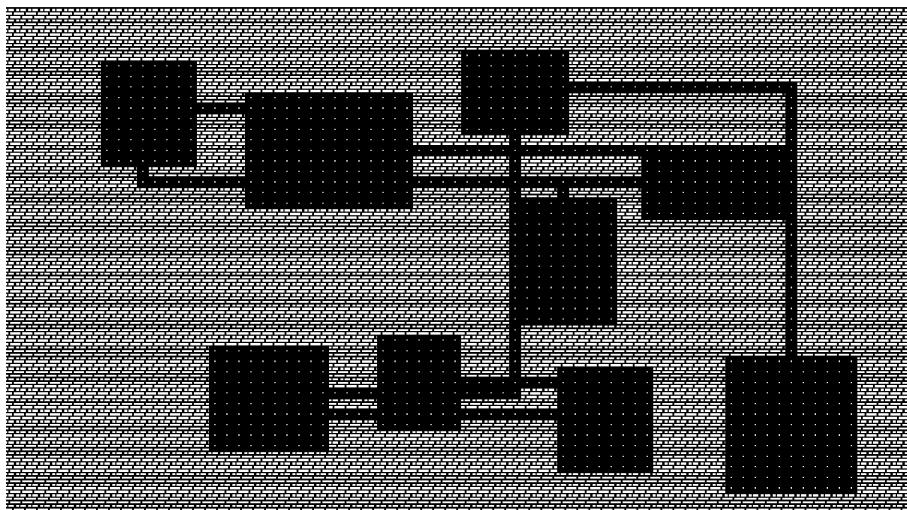
## 2.5. Opis poziomów

Gra składa się z sześciu poziomów oferujących różnorodnych przeciwników i przedmioty. W tej sekcji w celu prezentacji wyglądu poszczególnych poziomów posłużono się testerem proceduralnie generowanych map. Poziomy zaprezentowano w kolejności w jakiej występują w trakcie rozgrywki. Mapy na poziomach 1-5 są proceduralnie generowane, więc przy każdej nowej rozgrywce ich wygląd będzie podobny,

ale odmienny. Również rozmieszczenie przeciwników i przedmiotów na poziomach jest za każdym razem losowe. Akcja gry ma miejsce w podziemnych lochach połączonych z jaskiniami, kolejne poziomy są umieszczone coraz głębiej, dlatego są połączone ze sobą schodami, których położenie również jest losowe. Gracz ma możliwość swobodnego wracania do poprzednich poziomów.

### 2.5.1. Poziom 1

Poziom pierwszy to kilka prostych pokojów połączonych ze sobą korytarzami, przykładowy układ mapy przedstawiono na rysunku 2.5. Na poziomie tym napotkać można najprostszych przeciwników: goblin oraz mały szlam. Istnieje też mała szansa, że w pokoju z goblinami pojawi się też bardziej trudny przeciwnik – ork. Na poziomie tym znaleźć można podstawowy pancerz i miecz, a także mikstury uzdrawiające i zwoje uśpienia, które są szczególnie przydatne w walkach z bardziej trudnymi przeciwnikami w dalszych poziomach.

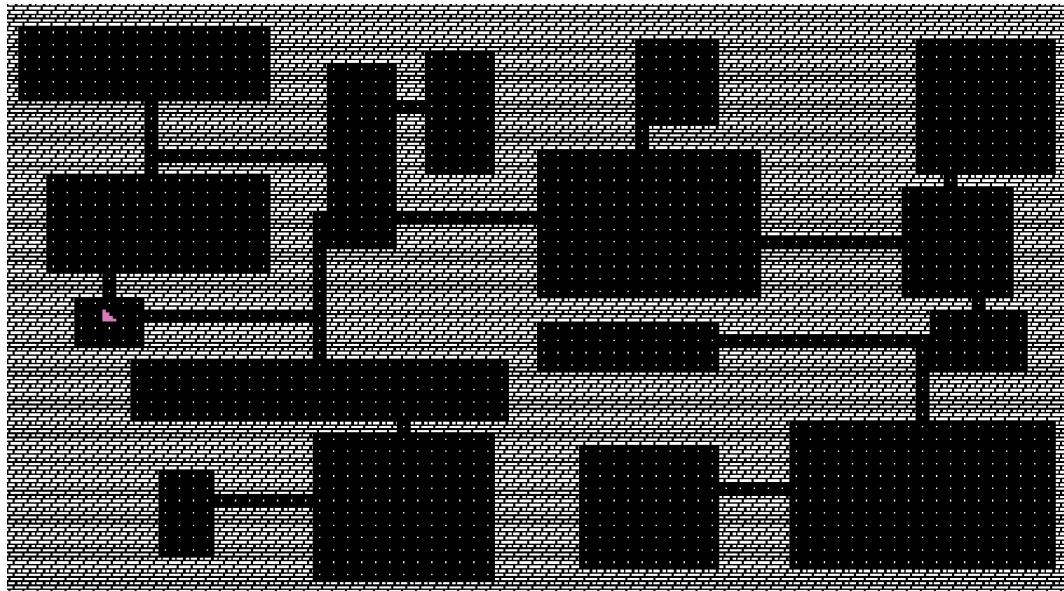


Rysunek 2.5: Przykładowy układ poziomu pierwszego

### 2.5.2. Poziom 2

Poziom drugi to bardziej rozwinięta wersja poziomu pierwszego z większą liczbą pokojów i inną metodą generacji. Przykładowy poziom drugi przedstawiono na rysunku 2.6. Można na nim spotkać gobliny z poprzedniego poziomu a także bardziej trudnych przeciwników: orków i rycerza. Szczególnie trudnym przeciwnikiem na tym poziomie jest rycerz, który wyposażony został w mocną zbroję i miecz. Przedmioty te można zdobyć po jego pokonaniu. W pokoju z rycerzem jest też szansa na znalezienie potężniejszych magicznych

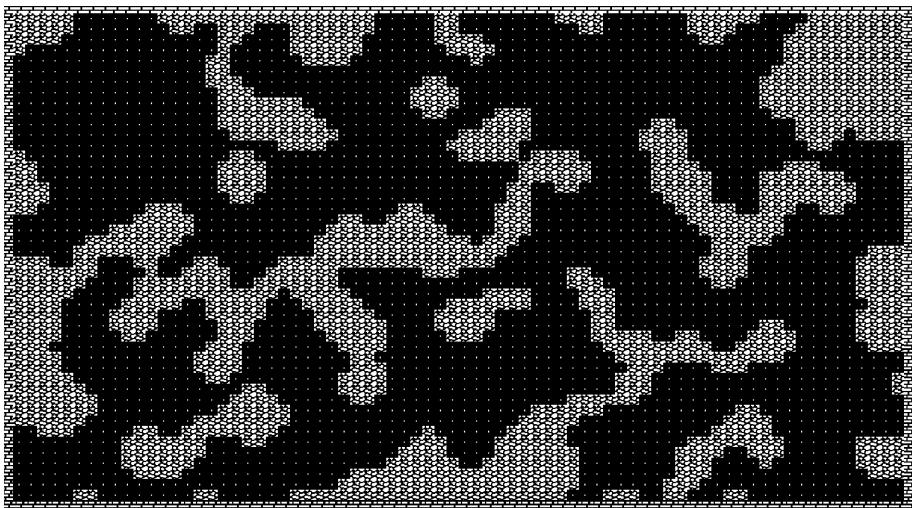
zwojów - kuli ognia, oraz teleportacji. Na poziomie tym można też znaleźć małe ilości mocniejszych mikstur uzdrowienia, oraz zwoje magicznych pocisków.



Rysunek 2.6: Przykładowy układ poziomu drugiego

### 2.5.3. Poziom 3

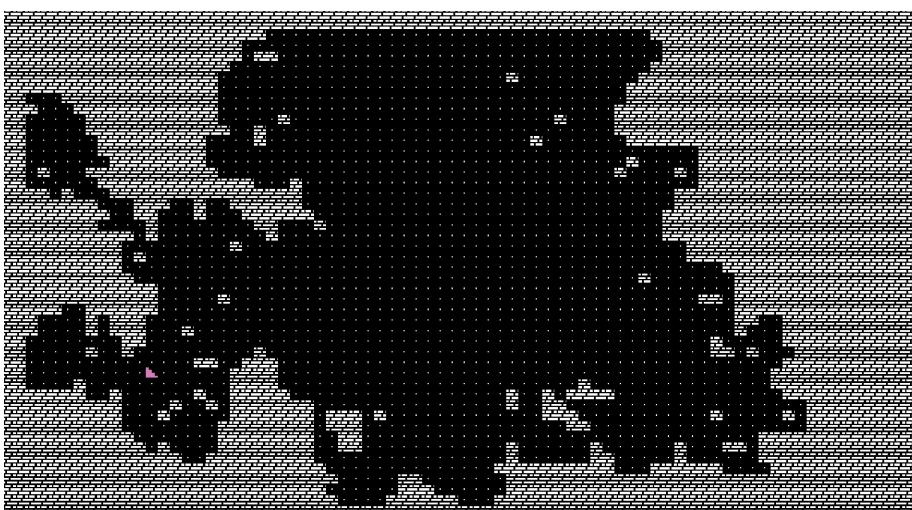
Poziom trzeci to duża jaskinia wypełniona masą goblinów i orków. Przykładowy widok poziomu trzeciego przedstawiono na rysunku 2.7. Mimo, że nie spotyka się tu trudniejszych przeciwników niż orkowie, to trudnością tego etapu jest duża ich liczba, oraz otwarty teren. Taki układ mapy, w przeciwieństwie do ciasnych korytarzy poprzednich poziomów, może łatwo doprowadzić do otoczenia gracza przez wielu przeciwników.



Rysunek 2.7: Przykładowy układ poziomu trzeciego

#### 2.5.4. Poziom 4

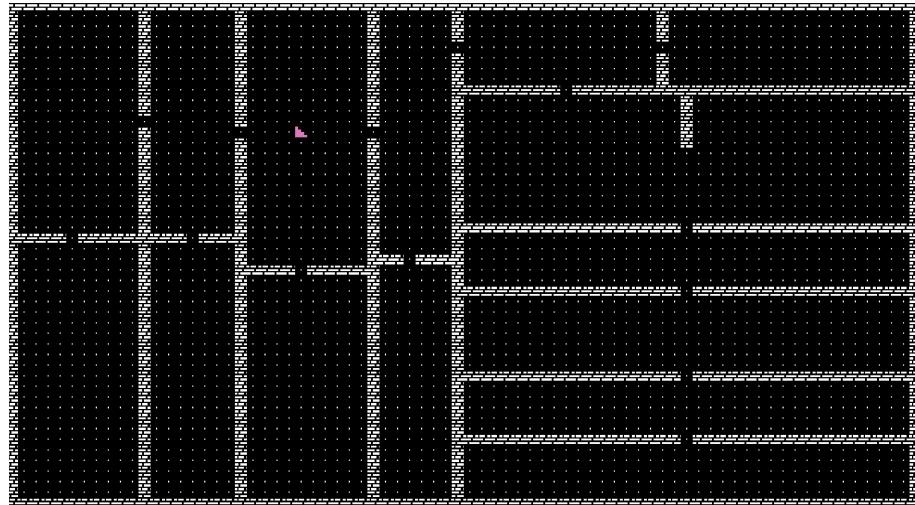
Poziom czwarty to kolejna jaskinia, tym razem z dużo większym obszarem w centrum, oraz okazjonalnymi ciasnymi odnogami. Przykładowy widok poziomu czwartego przedstawiono na rysunku 2.8. Na poziomie tym można spotkać dużą liczbę prostych małych szlamów z pierwszego poziomu, a także sporo ich większych odmian. Większy szlam poza byciem silniejszą odmianą posiada też specjalną zdolność – po śmierci rozpadnie się na kilka małych szlamów. Na tym poziomie można też znaleźć mocniejszą zbroję i miecz, a także duże mikstury uzdrowienia i zwoje obszarowego uśpienia.



Rysunek 2.8: Przykładowy układ poziomu czwartego

### 2.5.5. Poziom 5

Poziom piąty to duży obszar podzielony na pokoje bezpośrednio połączone ze sobą, bez korytarzy. Przykładową mapę poziomu piątego przedstawiono na rysunku 2.9. Na tym poziomie spotkać można najmocniejszych przeciwników: rycerzy oraz łotrzyków. W trakcie eksploracji poziomu piątego odnaleźć można najmocniejszą w grze zbroję oraz miecz, co będzie dla gracza szczególnie przydatne na poziomie szóstym.



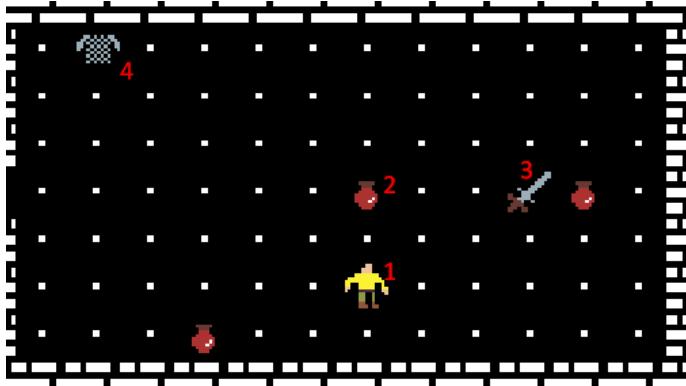
Rysunek 2.9: Przykładowy układ poziomu piątego

### 2.5.6. Poziom 6

Ostatni poziom jest wielką, otwartą przestrzenią. Na tym poziomie gracz znajduje się tylko jeden przeciwnik – potężny szlam, którego pokonanie jest celem gry. Potężny szlam jest silniejszą wariacją szlamów, na które rozpada się po śmierci, one z kolei po śmierci rozpadną się na małe szlamy. Aby ukończyć grę należy go pokonać oraz zejść schodami w dół.

## 2.6. Przedmioty oraz ich używanie

W grze gracz przez eksplorację znaleźć może wiele różnorodnych przedmiotów, których używanie jest konieczne w celu ukończenia gry. Przedmioty dzielą się na zbroje, broń, mikstury oraz magiczne zwoje. Aby podnieść przedmiot gracz musi przejść na zajmowane przez przedmiot pole a następnie wcisnąć klawisz 'G'. Na rysunku 2.10 przedstawiono pokój wypełniony przedmiotami.



Rysunek 2.10: Pokój z przedmiotami. 1 - postać gracza, 2 - mikstura zdrowia, 3 - broń, 4 - zbroja

### 2.6.1. Okna ekwipunku

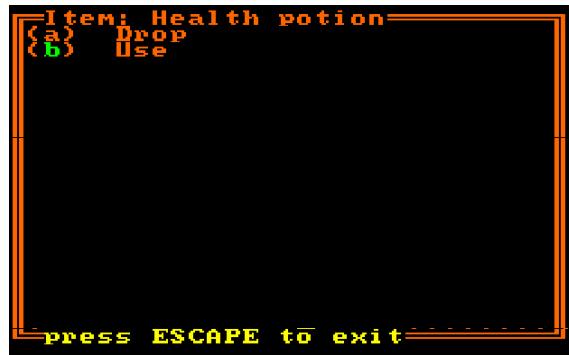
Gracz w każdej chwili może sprawdzić swój ekwipunek klawiszem 'I' – ekran zaprezentowano na rysunku 2.11. Wyboru przedmiotu dokonuje się za pomocą strzałek i klawisza Enter – litera po lewej stronie przy aktualnie wybranym przedmiocie jest zaznaczona na zielono. Innym sposobem wybrania przedmiotu jest wciśnięcie przypisanej mu po lewej stronie litery w nawiasach. Dopisek '<EQUIPPED>' po prawej stronie danego przedmiotu oznacza, że jest on aktualnie założony przez postać gracza. Zielone liczby oznaczają zasoby danego przedmiotu. Na rysunku 2.11 zaobserwować można, że gracz posiada cztery mikstury uzdrawiające 'Health potion'. Klawiszem Esc można wyjść z menu ekwipunku i powrócić do gry.



Rysunek 2.11: Okno ekwipunku

Po wybraniu przedmiotu wyświetla się menu wyboru akcji dostępnych dla każdego przedmiotu. Przykładowe dostępne akcje dla mikstury uzdrawiającej przedstawiono na rysunku 2.12. Każdy przedmiot można wyrzucić za pomocą akcji 'drop' – spowoduje to wyrzucenie przedmiotu na pole zajmowane przez gracza. W przypadku niezałożonych zbroi i broni dostępna jest opcja 'Equip' powodująca wyekwipowanie

danego przedmiotu. Konsekwentnie, założone przedmioty posiadają akcję 'UnEquip', która pozwala na ściągnięcie danego przedmiotu. Mikstury i zwoje posiadają akcję 'Use' pozwalającą na użycie danego przedmiotu. Niektóre przedmioty są używane od razu na postaci gracza, a inne pozwalają na wybranie celu.



Rysunek 2.12: Okno akcji przedmiotu z ekwipunku

### 2.6.2. Zbroje i broń

Każda postać w grze posiada poziom obrony, który obniża wartość otrzymywanych obrażeń. Poziom ten zwiększyć można za pomocą zbroi zakładanych na poszczególne części ciała: głowę, tors, ręce oraz nogi. W grze w kolejnych poziomach spotkać można coraz lepsze rodzaje zbroi, dzięki czemu postać gracza jest coraz wytrzymalsza.

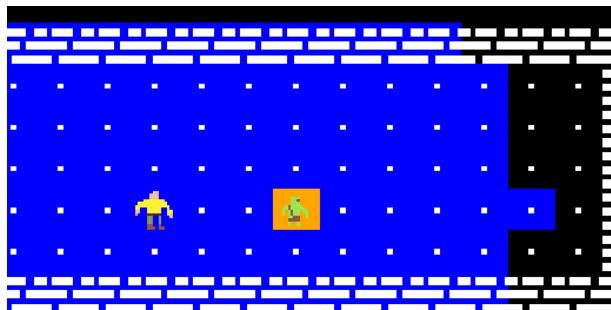
Postać gracza oraz przeciwnicy posiadają również wartość ataku, która określa liczbę zadawanych w trakcie akcji obrażeń. Wartość tę można zwiększyć zbierając broń rozsianą po poziomach. Lepsze rodzaje broni znaleźć można w późniejszych poziomach, dzięki czemu gracz, który dokładnie eksploruje plansze będzie zadawał coraz więcej obrażeń. Reprezentację graficzną zbroi oraz broni zaobserwować można na rysunku 2.10.

### 2.6.3. Mikstury

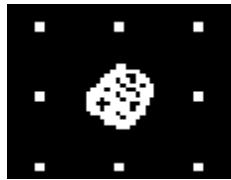
Postać gracza posiada określoną liczbę punktów zdrowia 'HP', które zmniejszają się wskutek ataków przeciwników. Aby zregenerować zdrowie gracz musi znaleźć mikstury uzdrawiające. W grze dostępne są dwie wersje tych mikstur – zwykła 'Health potion' oraz większa 'Great health potion', która leczy więcej punktów zdrowia. Użycie mikstur jest natychmiastowe, a celem zawsze jest postać gracza. Wygląd mikstur został pokazany na rysunku 2.10.

#### 2.6.4. Zwoje

Kolejnym typem przedmiotów są magiczne zwoje. Zwoje zapewniają wiele różnorodnych efektów, a ich użycie wymaga wyboru celu, co zaprezentowano na rysunku 2.13. Cel ataku zaznaczony jest na pomarańczowo, niebieskie pola oznaczają zasięg danego zwoju. Cel można wybrać przesuwając znacznik klawiszami używanymi do przesuwania postaci lub kliknięciem myszki na dane pole. Na rysunku 2.14 pokazano wygląd zwojów w grze.



Rysunek 2.13: Wybór celu magicznego zwoju



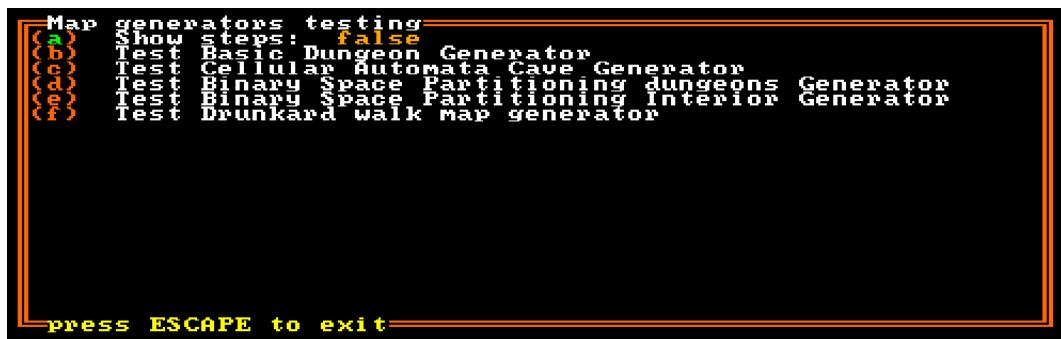
Rysunek 2.14: Magiczny zwój w grze

Dostępne w grze zwoje to:

- 'Magic missile scroll' – magiczny pocisk, zadaje obrażenia wybranemu celowi;
- 'Fireball scroll' – ognista kula, zadaje obrażenia przeciwnikom w małym promieniu od wybranego celu;
- 'Sleep scroll' – uśpienie, usypia wybrany cel na kilka tur – nie będzie się on poruszał ani atakował;
- 'Area sleep scroll' – uśpienie obszarowe, usypia kilka przeciwników w małym promieniu od wybranego celu;
- 'Teleport scroll' – teleportacja, natychmiast przenosi postać gracza w wybrane miejsce.

## 2.7. Tester proceduralnie generowanych poziomów

Gry roguelike opierają się w głównej mierze na proceduralnie generowanych poziomach, co oznacza generowanie map na podstawie algorytmów zawierających elementy losowości. Znaczy to na przykład, że użyty w niniejszej grze algorytm do generowania jaskiń zawsze wygeneruje poziom przypominający jaskinię, lecz za każdym razem inaczej wyglądający. Takie podejście do tworzenie poziomów pozwala znacznie zaoszczędzić czas, który w standardowej metodzie byłby spędzony na ręcznie tworzenie poziomów. Mapy proceduralnie generowane są jednak podatne na błędy – na przykład generowane są miejsca, do których nie da się dojść. W celu szybszego odnajdywania i naprawiania takich i podobnych pułapek, a także prezentacji możliwości zaimplementowanych algorytmów stworzono menu do testowania generatorów map, zaprezentowane na rysunku 2.15. Opcja ta dostępna jest w głównym menu gry po wybraniu 'Test Map Generators'.



Rysunek 2.15: Menu testera proceduralnie generowanych map

Klawiszami strzałek oraz klawiszem Enter wybiera się opcję, klawiszem Esc można powrócić do głównego menu gry. Opcje dostępne w menu to:

- a) 'Show steps' – zmiana opcji pokazania kroków generacji w trakcie testowania generatorów. 'true' oznacza włączenie pokazu kroków, 'false' wyłączenie,
- b) 'Test Basic Dungeon Generator' – testowanie generacji poziomu 1,
- c) 'Test Cellular Automata Cave Generator' – testowanie generacji poziomu 3,
- d) 'Test Binary Space Partitioning dungeons Generator' – testowanie generacji poziomu 2.

- e) 'Test Binary Space Partitioning Interior Generator' – testowanie generacji poziomu 5.
- f) 'Test Drunkard walk map generator' – testowanie generacji poziomu 4.

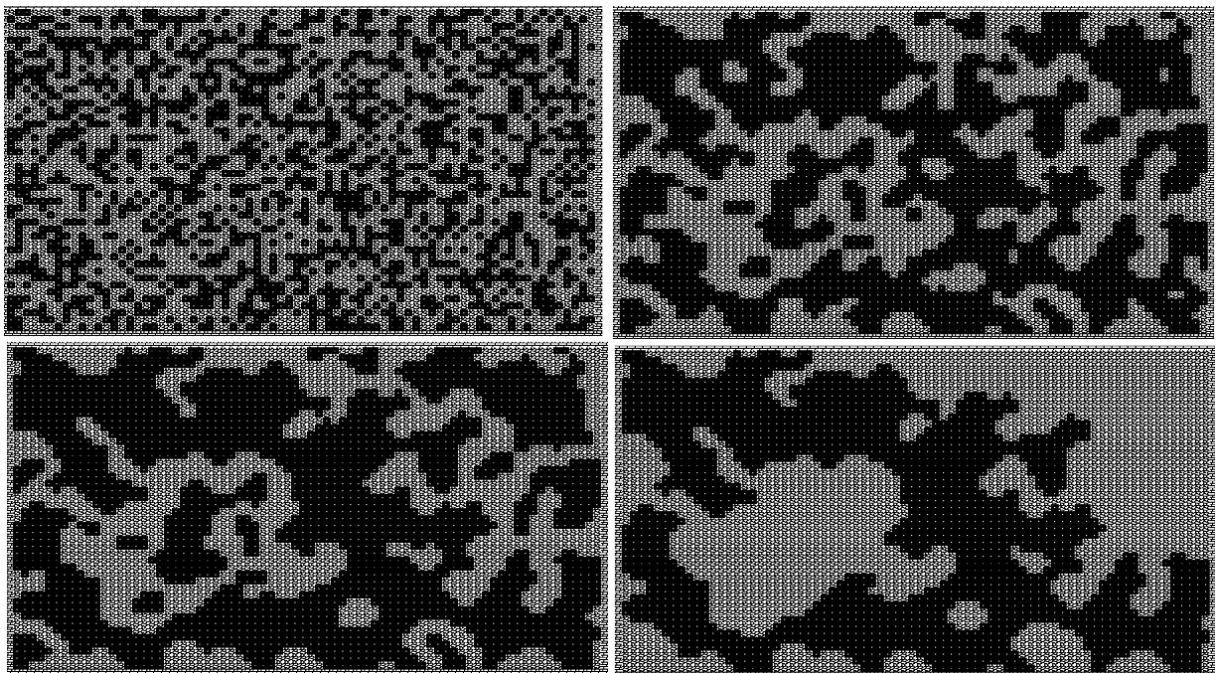
Po wybraniu generatora map otwiera się okno zaprezentowane na rysunku 2.16.

W głównej części zaprezentowany jest aktualny stan mapy w przypadku pokazu z krokami lub finalny efekt generacji planszy w przypadku wyłączenia kroków. W dolnej części okna znajdują się informacje o numerze aktualnego kroku oraz jego opisie. Klawiszem Esc powrócić można do menu testera, klawiszem spacji generuje się nową mapę lub przechodzi do kolejnego kroku.

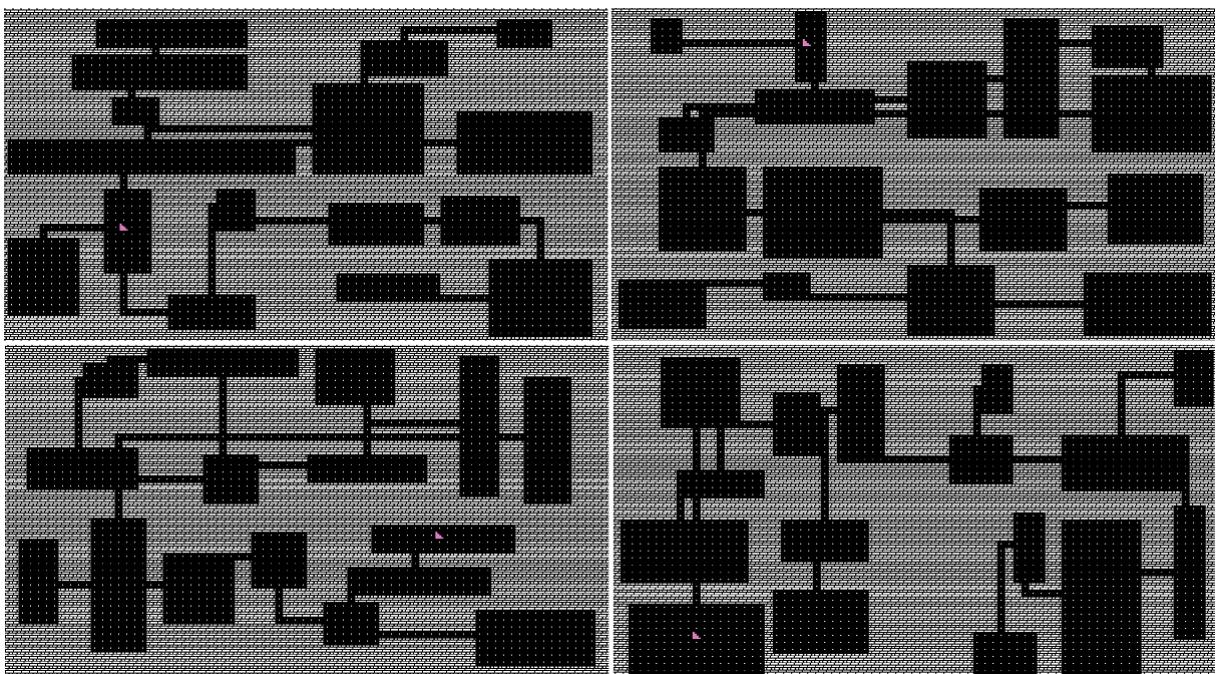


Rysunek 2.16: Okno testera wybranego generatora map

Na rysunku 2.17 przedstawiono część kroków generacji mapy do poziomu trzeciego, na rysunku 2.18 pokazano przykładowe wygenerowane mapy do poziomu drugiego.



Rysunek 2.17: Przykład testowania z krokami generacji mapy do poziomu trzeciego



Rysunek 2.18: Przykład testowania bez kroków generacji mapy do poziomu drugiego

### **3. Technologie i implementacja**

W niniejszym rozdziale opisano technologie, architekturę oraz algorytmy użyte do implementacji gry. W ostatnim podrozdziale szczegółowo omówiono użycie metody proceduralnego generowania map.

#### **3.1. Technologie**

Do zaprogramowania gry użyto języka Rust, będącego kompilowalnym językiem ogólnego przeznaczenia, powstały w 2010 roku [15]. Język Rust wydajnościowo jest porównywalny do języka C++, lecz jest to język nowocześniejszy i bezpieczniejszy [16]. Język jest dużo bardziej restrykcyjny już na poziomie komplikacji, dzięki czemu błędy są częściej wykrywane w trakcie komplikacji, niż dopiero po uruchomieniu programu. Użyto też wbudowanych w język narzędzi: 'cargo fmt' do formatowania oraz 'cargo clippy' wyszukującego niegroźne błędy niewykrywalne przez kompilator, ale utrudniające czytelność kodu. Te cechy oraz narzędzia języka Rust znacznie ułatwiają stosowanie zasad czystego kodu [17].

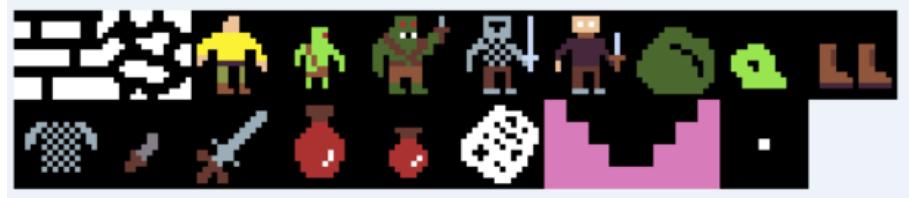
Do edycji kodu użyty został program Visual Studio Code [18] z rozszerzeniami dla języka Rust, przede wszystkim 'rust-analyzer' zapewniający kolorowanie składni, oraz automatyczne uzupełnianie kodu.

W programie użyto wielu tworzonych przez społeczność bibliotek open source – zapewniających prawo do korzystania, oraz dostęp do kodu źródłowego. Najważniejsze użyté biblioteki to:

- 'rltk' – tworzenie okna gry, obsługa grafiki oraz pętli głównej gry,
- 'specs' – implementacja architektury Entity Component System.

Gra posiada grafikę typu *pixel art* w formacie 16x16 pikseli. W oknie gry użyto dwukrotnie powiększonych obrazków w formacie 32x32 w celu polepszenia widoczności. Aby móc zmieścić całą mapę w jednym oknie w testerze proceduralnych generatorów map, użyty został oryginalny format 16x16. Do tworzenia grafiki użyto programu Aseprite [19] będącego wygodnym narzędziem graficznym wyspecjalizowanym w tworzeniu grafiki *pixel art*. Wszystkie obrazki użyte w grze znajdują się w folderze 'resources' w dwóch plikach, odpowiednio 'sprite\_sheet\_16x16.png' oraz 'sprite\_sheet\_32x32.png'. Taki format przechowywania obrazków zapewnia porządek w folderach gry. Dostęp

do konkretnego obrazka odbywa się przez określenie wymiarów oraz podanie indeksu. Aby uzyskać obrazek przedstawiający gracza z pliku 'sprite\_sheet\_16x16.png' przedstawionego na rysunku 3.1 należy określić rozmiary – 16x16 pikseli oraz podać indeks 2 – obrazki są indeksowane od lewej do prawej, pierwszy element ma indeks zero.



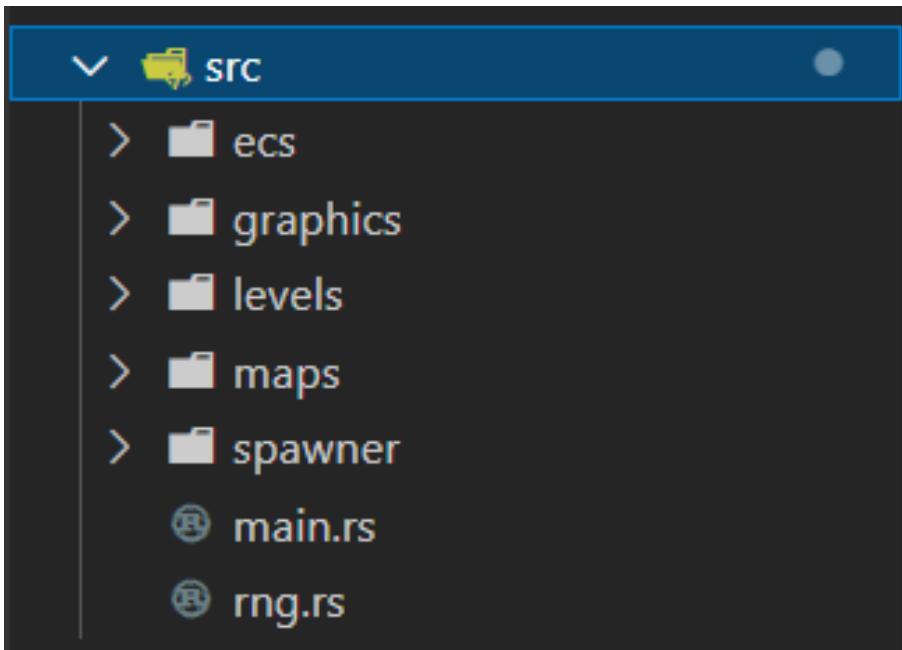
Rysunek 3.1: Grafiki użyte w grze. Od lewej: ściana, skały, postać gracza, gooblin, ork, rycerz, łotr, szlam, mały szlam, buty, zbroja, sztylet, miecz, duża mikstura zdrowia, mikstura zdrowia, magiczny zwój, schody w dół, schody w górę, podłoga

## 3.2. Implementacja

W niniejszym podrozdziale przedstawiono szczegóły implementacji, wraz z prezentacją fragmentów kodu źródłowego wybranych elementów gry.

### 3.2.1. Struktura projektu

Gra posiada wiele modułów spełniających odmienne zadania. Kod projektu po-dzielony został na wiele modułów i pod-modułów w celu lepszej organizacji. Główny folder programu zawiera foldery: 'resources' – zawierający grafiki użyte w grze oraz 'src' – zawierający kod źródłowy. Na rysunku 3.2 przedstawiono schemat głównych modułów kodu projektu. W folderze głównym znajduje się też plik Cargo.toml określający między innymi użyte biblioteki.



Rysunek 3.2: Schemat głównych modułów struktury projektu

Moduł 'ecs' zawiera podstawowy plik projektu 'game\_state.rs' obsługujący główną pętlę gry jako maszyna stanów. W tym module znajdują się też pod-moduły:

- 'components' – definicje komponentów,
- 'systems' – zawiera systemy, dzieli się na kolejne pod-moduły:
  - 'ai' – systemy sterujące komputerowymi przeciwnikami;
  - 'combat' – systemy związane z walką i zadawaniem obrażeń;
  - 'effects' – efekty takie jak leczenie i teleportacja;
  - 'inventory' – systemy zarządzania ekwipunkiem, podnoszenia i używania przedmiotów;
  - 'map' – systemy związane z mapą;
  - 'player' – systemy poruszania i akcji postaci gracza, oraz odczytywania wcisniętych klawiszy;
  - 'spawn' – system umieszczania obiektów na poziomie; /item 'view\_system' – system obliczania pola widzenia gracza oraz przeciwników.

Moduł 'graphics' odpowiada za grafikę: tworzenie okna gry, rysowanie mapy i okien interfejsu. Znajdują się w nim funkcje do rysowania mapy gry, oraz obiektów:

przedmiotów i postaci. Posiada pod-moduł 'menus' z definicją struktur odpowiedzialnych za poszczególne okna w grze, a także definicję cechy 'WindowOptionSelector', która implementuje większość tych struktur.

Moduł 'levels' posiada definicję struktury 'Level' odpowiedzialnej za poziom, oraz 'LevelManager' odpowiedzialnej za przechowywanie i tworzenie nowych poziomów.

W module 'map' zdefiniowana jest struktura przechowująca kształt mapy: 'Map', definicja typów pól na mapie oraz pomocnicze funkcje związane z mapą. Zawiera też pod-moduł 'generators' zawierający generatory poszczególnych typów map.

Moduł 'spawner' jest odpowiedzialny za tworzenie obiektów w grze takich jak: postać gracza, przeciwnicy oraz przedmioty. Zawiera też definicję struktur 'SpawnTable' odpowiadających za rozmieszczanie obiektów na konkretnych poziomach.

Plik 'main.rs' zawiera funkcję główną programu: 'main()'. W tej funkcji deklaruje się podstawowe konfigurację takie jak rozmiary map i okna gry, a także uruchamiana w niej jest pętla główna gry.

W pliku 'rng.rs' zawarto funkcje do losowania liczb używane w generatorach map, a także przy rozmieszczaniu na nich przeciwników i przedmiotów.

### 3.2.2. Architektura Entity Component System

Twórcy gier gatunku roguelike dzięki prostej grafice mogą poświęcić więcej czasu na tworzenie skomplikowanych i interesujących mechanik. Szczególnie przydatną w tworzeniu takich skomplikowanych mechanik jest architektura Entity Component System (Encja, Komponent, System), nazywana dalej ECS [20], skupiająca się na rozdzielaniu danych od systemów. Dzięki użyciu biblioteki 'specs' pominięto etap implementowania architektury ECS a skupiono się wyłącznie na jej użyciu.

Elementami ECS są :

- 'Entity' – encja, jest to indeks reprezentujący dany obiekt w grze na przykład gracza, przeciwnika lub przedmiot,
- 'Component' – komponent, są to części składowe obiektów w grze przypisywane do konkretnych encji,
- 'System' – części logiczne architektury, konkretne systemy aktualizują stany konkretnych komponentów,
- w niektórych implementacjach stosowany też jest czwarty element 'Event' – wy-

darzenie, służący do informowania konkretnych obiektów o takich wydarzeniach jak na przykład zadanie obrażeń lub użycie przedmiotu. W przypadku tej gry wydarzenie zostało zrealizowane za pomocą odpowiednich komponentów.

Komponenty to małe części składowe, których stan aktualizowany jest przez konkretne systemy. Komponenty tworzone są jako struktury implementujące cechę 'Component' z biblioteki 'specs'. Definicje komponentów znajdują się w pliku 'src/entities/components/mod.rs'. Przykładowe komponenty przedstawione na rysunku 3.3 to:

- 'BlocksTile' – struktura bez pól, informuje, że dany obiekt blokuje aktualnie zajmowane pole i nie może się na nim znaleźć inny obiekt posiadający 'BlocksTile',
- 'Hp' – informuje o maksymalnej i aktualnej liczbie punktów zdrowia. Posiadacze tego komponentu mogą zostać zranieni,
- 'CombatBaseStats' – statystyki bojowe: atak i obrona, informują odpowiednio o ilości zadawanych obrażeń w trakcie walki i o ilości blokowanych obrażeń,
- 'WantsToMeleeAttack' – komponent-wydarzenie dodawany do obiektu chcącego dokonać ataku, informuje o celu ataku,
- 'SufferDamage' – komponent-wydarzenie dodawany do obiektów, którym zadane zostały obrażenia. Ilości obrażeń przedstawione są jako wektor, gdyż w jednej turze jeden obiekt może zostać zaatakowany przez kilku przeciwników.

```

#[derive(Component, Debug, Clone)]
pub struct BlocksTile {}

#[derive(Component, Debug, Clone)]
pub struct Hp {
    pub max_hp: i32,
    pub hp: i32,
}

#[derive(Component, Debug, Clone)]
pub struct CombatBaseStats {
    pub attack: i32,
    pub defense: i32,
}

#[derive(Component, Debug, Clone)]
pub struct WantsToMeleeAttack {
    pub target: Entity,
}

#[derive(Component, Debug, Clone)]
pub struct SufferDamage {
    pub amount: Vec<i32>,
}

```

Rysunek 3.3: Przykładowe struktury komponentów

Na rysunku 3.4 pokazano przykładowe tworzenie encji za pomocą przydzielania jej odpowiednich komponentów. Tworzoną encją w tym przypadku jest postać gracza. Poniżej wyjaśniono znaczenie komponentów, z których składa się bohater, poza komponentami wyjaśnionymi wcześniej:

- 'Player' – struktura bez pól, informuje, że dany obiekt jest sterowany przez gracza. W grze powinien istnieć dokładnie jeden obiekt posiadający ten komponent;
- 'Movable' – obiekt z tym komponentem jest w stanie się poruszać. Zawiera on kierunek kolejnego ruchu lub 'None' w przypadku braku ruchu;
- 'View' – obiekt posiadający ten komponent posiada wzrok. Określa zasięg widzenia oraz aktualnie widziane pola na mapie;

- 'ViewMemory' – pozwala zapamiętywać zobaczone wcześniej pola na mapie.  
Zawiera listę odkrytych już pól,
- 'Position' – obiekt może zajmować określoną w tym komponencie pozycję na mapie;
- 'Name' – określa nazwę danego obiektu;
- 'Renderable' – obiekt z tym komponentem posiada przypisaną do niego grafikę i może być rysowany na mapie,
- 'BodyParts' – obiekt posiada części ciała na które może założyć ekwipunek.  
'default\_humanoid()' to domyślne części ciała dla postaci o ciele podobnym do ludzkiego: głowa, tors, ręce i nogi;
- 'Inventory' – pozwala obiekowi na podnoszenie i przechowywanie przedmiotów.

```

pub fn spawn_player(ecs: &mut World, x: usize, y: usize) -> Entity {
    ecs.create_entity()
        .with(components::Player { input: None }): EntityBuilder
        .with(components::Movable { move_dir: None }): EntityBuilder
        .with(components::View {
            range: 40,
            visible_tiles: HashSet::<rltk::Point>::new(),
            should_update: true,
        }): EntityBuilder
        .with(components::ViewMemory {
            seen_tiles: HashMap::default(),
            should_update: true,
        }): EntityBuilder
        .with(components::Position { x, y, level: 0 }): EntityBuilder
        .with(components::Name {
            name: "player".to_string(),
        }): EntityBuilder
        .with(components::Renderable {
            ascii: rltk::to_cp437('@'),
            texture: Some(2),
            fg: RGB::named(col: rltk::YELLOW),
            bg: RGB::named(col: rltk::BLACK),
            render_order: 0,
        }): EntityBuilder
        .with(components::Hp {
            max_hp: 300,
            hp: 300,
        }): EntityBuilder
        .with(components::CombatBaseStats {
            attack: 5,
            defense: 1,
        }): EntityBuilder
        .with(components::BodyParts::default_humanoid()): EntityBuilder
        .with(components::Inventory::new_empty()): EntityBuilder
        .build()
}

```

Rysunek 3.4: Funkcja tworząca encję gracza i przydzielająca mu odpowiednie komponenty

Systemy odpowiedzialne są za logikę gry, ich celem jest aktualizacja konkretnych komponentów. Na rysunku 3.5 przedstawiono system odpowiedzialny za przyjmowanie obrażeń. Systemy utworzone są jako puste struktury implementujące cechę 'System' z biblioteki specs. W implementacji tej cechy najpierw zdefiniowano typ 'SystemData' określający do jakich danych oraz w jaki sposób ma dostęp. Dla tego systemu przyznano dostęp do komponentów 'Hp' i 'SufferDamage'. Następnie zdefiniowano funkcję 'run' wymaganą przez cechę 'System'. W pierwszej linijce funkcji utworzono zmienne 'hps' - zbiór wszystkich komponentów 'Hp' oraz 'damages' - zbiór wszystkich komponentów 'SufferDamage'. Użyta w pętli for funkcja 'join' wyodrębnia komponenty znajdujące się na wszystkich zbiorach, na których została ona wywołana oraz gdzie

komponenty te przypisane są do wspólnego właściciela – Encji. Oznacza to, że każda iteracja pętli posiada dostęp do konkretnych komponentów ze zbiorów przekazanych do tej funkcji będących komponentami jednej Encji. Dla każdej Encji, która posiada te dwa komponenty odjęto sumę przyjętych obrażeń od aktualnego poziomu życia. Z racji, że komponenty 'SufferDamage' są komponentami-wydarzeniami, to można je usunąć po skończeniu pętli.

```
pub struct DamageSystem {}

impl<'a> System<'a> for DamageSystem {
    type SystemData = (
        WriteStorage<'a, components::Hp>,
        WriteStorage<'a, components::SufferDamage>,
    );

    fn run(&mut self, data: Self::SystemData) {
        let (mut hps: Storage<Hp, FetchMut<MaskedStorage<...>>>, mut damages: Storage<SufferDamage, FetchMut<...>>) = data;

        for (mut hp: &mut Hp, damage: &SufferDamage) in (&mut hps, &damages).join() {
            hp.hp -= damage.amount.iter().sum::<i32>();
        }

        damages.clear();
    }
}
```

Rysunek 3.5: System 'DamageSystem' odpowiedzialny za zadawanie obrażeń obiektom

W przypadku systemów ważna jest kolejność ich wywoływania. W trakcie tury przeciwników najpierw wywoływany jest system 'AISystem' odpowiedzialny za akcje przeciwników komputerowych. Następnie zadawane są obrażenia za pomocą 'MeleeCombatSystem', oraz omówionego wcześniej 'DamageSystem'. Na końcu uruchamiana jest funkcja 'delete\_the\_dead' odpowiedzialna za usuwanie postaci, których punkty życia osiągnęły zero.

### 3.2.3. Pętla główna gry oraz jej stany

Za stan gry odpowiada struktura 'State' z pliku 'game\_state.rs'. Implementuje ona funkcję 'tick()', która uruchamiana jest w nieskończonej głównej pętli gry. Działanie funkcji 'tick' zależne jest od aktualnego stanu gry opisanego za pomocą typu wyliczeniowego 'RunState'. Możliwe stany gry to:

- 'MainMenu' – wyświetlane jest menu główne gry;
- 'MapGenTesting(bool)' – wyświetlane jest menu testera proceduralnie generowanych map. Ta opcja zawiera dodatkową zmienną typu bool, ustawienie tej

zmiennej na true oznacza wyświetlanie testowanej mapy, false oznacza wyświetlenie menu testera;

- 'AwaitingInput' – wyświetlane jest okno gry, gra oczekuje na akcję gracza;
- 'PreRun' – uruchamiane są wszystkie podstawowe systemy gry, następny stan to 'AwaitingInput';
- 'PlayerTurn' – uruchamiane są wszystkie podstawowe systemy gry, następny stan to 'MonsterTurn';
- 'MonsterTurn' – uruchamiane są wszystkie podstawowe systemy gry, następny stan to 'AwaitingInput';
- 'ShowInventory' – wyświetlane jest okno ekwipunku, gra oczekuje na akcję gracza;
- 'ShowEquipment' – wyświetlane jest okno aktualnie założonych zbroj i broni, gra oczekuje na akcję gracza;
- 'ShowItemActions(Entity)' – wyświetlane jest okno akcji konkretnego przedmiotu, gra oczekuje na akcję gracza;
- 'Targeting(TargetingAction)' – gracz określa cel dla konkretnego przedmiotu określonego w TargetingAction, określony też jest zasięg danego przedmiotu;
- 'MoveLevel(usize)' – gracz przenosi się do poziomu o podanym indeksie. Jeśli poziom ten jeszcze nie istnieje, to 'LevelManager' go tworzy;
- 'GameOver' – wyświetlane jest okno informujące o przegraniu gry, co ma miejsce w razie śmierci postaci gracza. Gra czeka na wcisnięcie konkretnego klawisza po którym wraca do menu głównego;
- 'Controls' – wyświetlane jest okno informacji o sterowaniu;
- 'Won' – wyświetlane jest okno informujące o wygraniu gry. Gra czeka na wcisnięcie konkretnego klawisza po którym wraca do menu głównego.

### 3.2.4. System rozmieszczania przeciwników i przedmiotów na poziomach

W grze występuje wiele rodzajów przeciwników i przedmiotów, a z racji generowania map w sposób proceduralny nie jest możliwe określenie ich statycznych położen. W tym celu utworzono system określający jakie obiekty lub postacie występują na poziomach, oraz jakie są ich ilości oraz szansa wystąpienia. System ten został zdefiniowany w pliku 'spawner/spawn\_tables.rs' i składa się z następujących struktur w kolejności od najmniejszej części składowej: 'SpawnEntry', 'SpawnPack' oraz 'SpawnTable'.

Struktura 'SpawnEntry' przedstawiona na rysunku 3.6 składa się z pól:

- 'entity\_name' – nazwa obiektu. Nazwa ta musi być uwzględniona w funkcji 'spawner/mod.rs/spawn\_entity()', która odpowiada za umiejscowienie obiektu o danej nazwie na podanej pozycji,
- 'rng\_range' – określa zasięg od minimalnej do maksymalnej liczby obiektów, które mogą zostać utworzone w ramach danego 'SpawnEntry'. Dokładna liczba jest losowana w podanym zakresie,
- 'chance\_perc' – określa szansę w punktach procentowych na pojawienie obiektów z danego 'SpawnEntry'.

```
#[derive(Clone, Debug)]
impl SpawnEntry {
    pub fn new(entity_name: String, rng_range: (u32, u32), chance_perc: u32) -> Self {
        Self { entity_name, rng_range, chance_perc }
    }
}
```

Rysunek 3.6: Struktura 'SpawnEntry'

Każdy generator map posiada metodę zwracającą dla wygenerowanej przez niego mapy, wydzielone obszary nazywane 'spawn areas'. Są to obszary, na których może pojawić się grupa obiektów zdefiniowana w strukturze 'SpawnPack' zaprezentowanej na rysunku 3.7. Struktura ta określa minimalny obszar, na którym może pojawić

się ta grupa, szansę wystąpienia tej grupy, oraz maksymalną liczbę grup tego typu na danym poziomie.

```
pub struct SpawnPack {
    // this pack will not be spawned more times than `max_spawns`
    pub max_spawns: usize,
    pub spawns_counter: usize,
    // chance percentage % for this pack to be spawned, default to 100
    pub chance_perc: usize,
    pub entities: Vec<SpawnEntry>,
    pub min_area: usize,
}
```

Rysunek 3.7: Struktura 'SpawnPack'

Na rysunku 3.8 przedstawiono funkcję tworzącą 'SpawnPack' będący grupą goblinów. W obszarze 'spawn area' dla którego wylosowana zostanie ta grupa, pojawi się od dwóch do sześciu goblinów, od zera do jednej mikstury życia, oraz zwój uśpienia z dwudziestoprocentową szansą na pojawienie się.

```
pub fn goblins_pack() -> SpawnPack {
    SpawnPack {
        min_area: 8,
        entities: vec![
            SpawnEntry::new("Goblin".to_string(), 2, 6),
            SpawnEntry::new("Health potion".to_string(), 0, 1),
            SpawnEntry::new("Sleep scroll".to_string(), 1, 1).with_chance(20),
        ],
        ..SpawnPack::default()
    }
}
```

Rysunek 3.8: 'SpawnPack' grupa goblinów

Ostatnim elementem rozmieszczania przedmiotów i przeciwników jest struktura 'SpawnTable' przedstawiona na rysunku 3.9 , która określa możliwe do pojawienia się na danym poziomie grupy obiektów – 'SpawnPack'. Na rysunku 3.10 przedstawiono tablicę możliwych grup na poziomie pierwszym.

```
pub struct SpawnTable {
    pub level_type: LevelType,
    pub spawn_packs: Vec<SpawnPack>,
}
```

Rysunek 3.9: Struktura 'SpawnTable'

```
SpawnTable {
    level_type: LevelType::BasicDungeon,
    spawn_packs: vec![
        SpawnPack::blips_pack().with_max_spawns(5),
        SpawnPack::goblins_pack().with_max_spawns(2),
        SpawnPack::goblins_with_orc_pack()
            .with_max_spawns(1)
            .with_chance_perc(10),
        SpawnPack::armory_low_tier().with_max_spawns(1),
    ],
}
```

Rysunek 3.10: Przykład 'SpawnTable' dla poziomu pierwszego

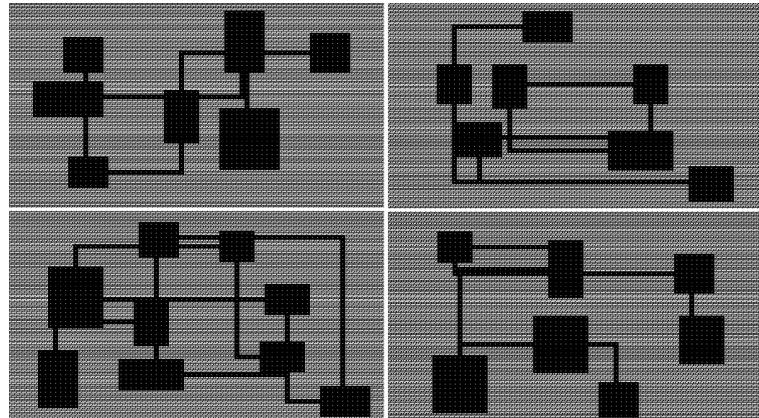
Po skończeniu generacji poziomu, generator oblicza obszary 'spawn area'. Następnie dla każdego takiego obszaru losowana jest grupa 'SpawnPack' z tablicy 'SpawnTable' przypisanej dla danego poziomu. Losowanie to odbywa się do momentu gdy wszystkim obszarom zostanie przypisany 'SpawnPack', albo każdy z nich osiągnie maksymalną liczbę pojawić się. W każdej przypisanej grupie następują dwa losowania dla każdego 'SpawnEntry' składającego się na tą grupę. Wynik pierwszego losowania decyduje o pojawienniu się danych obiektów, a drugie określa ich liczbę. Każdy obiekt, który pojawił się w ramach danego 'SpawnPack' zostaje umieszczony na losowym miejscu w przypisanym do tej grupy obszarze 'spawn area'.

### 3.3. Proceduralne generowanie poziomów

Proceduralna generacja oznacza tworzenie poziomów na podstawie algorytmów z użyciem losowości. Oznacza to, że przy każdym generatorze, mapy przez niego tworzone będą różne, ale będą tego samego typu. Każdy generator posiada strukturę konfiguracyjną, dzięki czemu łatwo można zmieniać jego zasady, co w połączeniu z testerem proceduralnie generowanych map pozwala na szybkie dostosowywanie parametrów w celu osiągnięcia konkretnego wyglądu planszy. Przykłady wygenerowanych map pochodzą z testera generowania map.

### 3.3.1. Poziom pierwszy

Generator używany do tworzenia mapy poziomu pierwszego jest jednym z najprostszych i polega na stochastycznym rozmieszczeniu pokojów o losowych wymiarach. Pokoje nie mogą na siebie nachodzić, ich liczbę i rozmiar można określić w konfiguracji generatora. Po umieszczeniu na mapie poziomów są one łączone ze sobą korytarzami, w kolejności w jakiej się pojawiły.



Rysunek 3.11: Przykład wygenerowanych map dla poziomu pierwszego

### 3.3.2. Poziom drugi – Binary Space Partitioning

Generator użyty w poziomie drugim oparty jest o metodę Binary Space Partitioning [21] – binarne dzielenie przestrzeni, określane też za pomocą skrótu BSP. Metoda ta opiera się o drzewo binarne [22], które zawiera kolejne pod-obszary mapy. Na rysunku 3.12 przedstawiono strukturę reprezentującą węzeł drzewa użytego w tej metodzie. Zawiera ona indeks węzła, poziom drzewa, na którym znajduje się ten węzeł, indeksy rodzica, siostry – węzła o wspólnym bezpośrednim rodzicu, opcjonalne indeksy dzieci, indeksy kolejnych rodziców aż do korzenia, obszar zajmowany przez ten węzeł, oraz opcjonalnie obszar zajmowany przez pokój umieszczony w tym węźle.

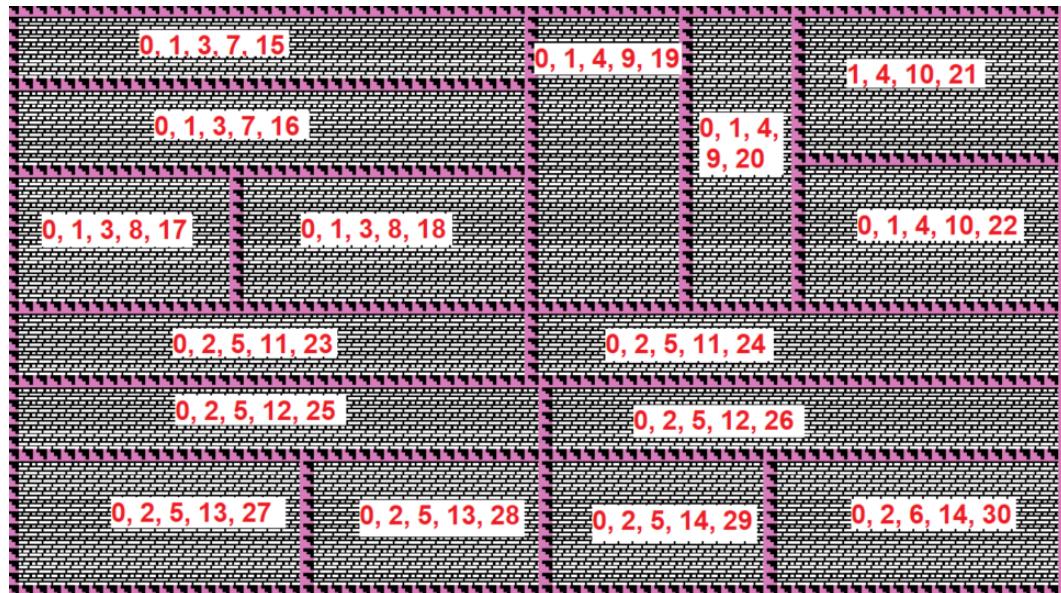
```
pub struct BSPNode {
    pub index: usize,
    /// 0 is root of tree
    pub tree_level: usize,

    /// index
    pub parent: usize,
    /// index
    pub sister: usize,
    /// indexes (Left, Right) / (Up, Down)
    pub children: Option<[usize; 2]>,

    /// indexes
    pub family: Vec<usize>,
    pub area: Rect,
    pub room: Option<Rect>,
}
```

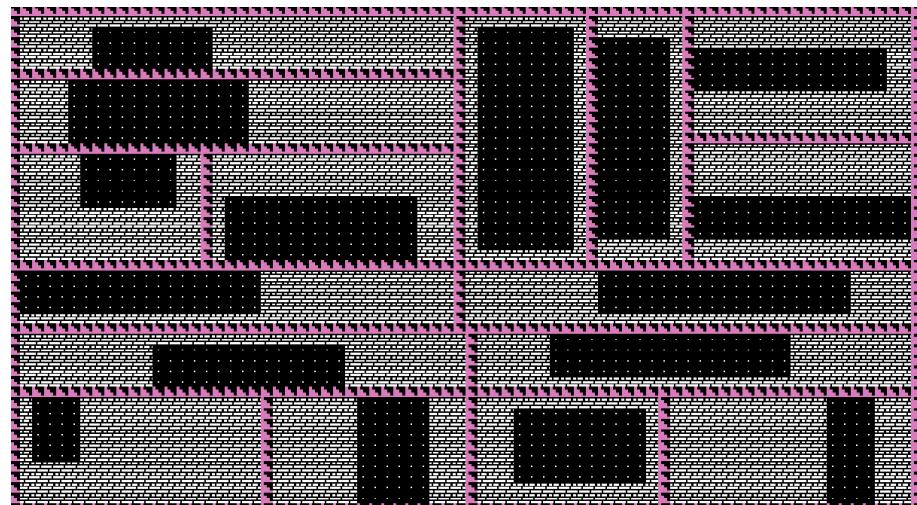
Rysunek 3.12: Węzeł drzewa binarnego użyty w metodzie Binary Space Partitioning

Korzeń drzewa zawsze zawiera obszar całej mapy. Następnie w losowym miejscu, zbliżonym do środka obecnego obszaru następuje podział na dwa obszary. Ten podział wykonywany jest losowo w sposób pionowy lub poziomy. Te dwa obszary są dziećmi obszaru, z którego się wywiodły. Taki podział następuje do momentu, gdy maksymalny, określony w generatorze poziom drzewa zostanie osiągnięty. Na rysunku 3.13 przedstawiono przykładowy podział mapy na obszary z przypisaniem obszarów do węzłów drzewa. Liczby w każdym obszarze oznaczają jego rodzinę. Zapisana pierwsza liczba od prawej strony to indeks węzła zawierającego ten obszar, następnie w lewo indeks jego rodzica, indeks rodzica rodzica i tak dalej. Węzły z indeksem zaczynającym się od 15 to liście drzewa – nie posiadają dzieci.



Rysunek 3.13: Przykład podziału map za pomocą metody Binary Space Partitioning

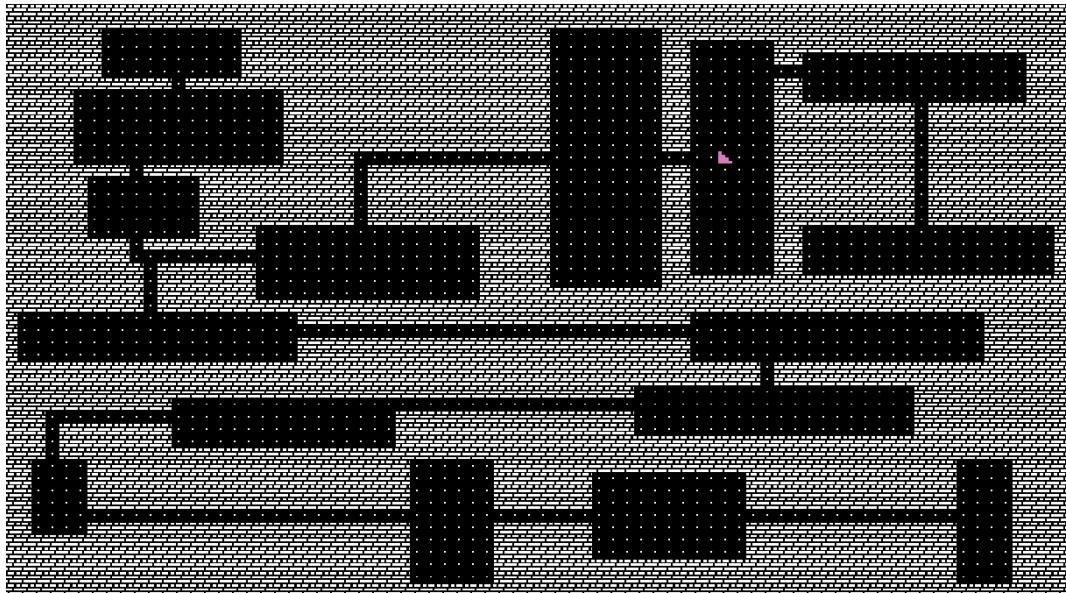
Następnie w każdym z obszarów zawartych w liściach drzewa umieszcza się pokój o losowych rozmiarach, przedstawiono to na rysunku 3.14.



Rysunek 3.14: Rozmieszczenie pokojów na mapie BSP

Ostatnim krokiem jest połączenie ze sobą pokojów za pomocą korytarzy. Dzięki użyciu drzewa binarnego można zastosować lepsze rozwiązanie niż "każdy z każdym", zapewniające, że z każdego pokoju można w jakiś sposób dostać się do każdego innego. Przechodząc od przedostatniego poziomu drzewa w góre najpierw łączy się ze sobą po dwa pokoje znajdujące się w obszarach będących dziećmi węzłów z tego poziomu. W kolejnych poziomach dla każdego węzła wybiera się losowych potomków będących

liścimi z obojga dzieci każdego węzła danego poziomu i łączy się zawarte w nich pokoje. Po skończeniu tego procesu wszystkie pokoje są połączone w sposób, który zakłada, że na pewno istnieje połączenie między jednym z pokojów zawartych w obszarze węzła o indeksie 3 a jednym z pokojów z obszaru należącego do węzła 4, gdyż węzły o indeksach 3 i 4 są bezpośrednimi dziećmi węzła o indeksie 1. Na rysunku 3.15 można zaobserwować, że w omówionym powyżej przypadku połączony został pokój z liścia o indeksie 18 z pokojem z liścia o indeksie 19.



Rysunek 3.15: Mapa wygenerowana za pomocą algorytmu binarnego podziału przestrzeni

### 3.3.3. Poziom trzeci – Cellular Automata

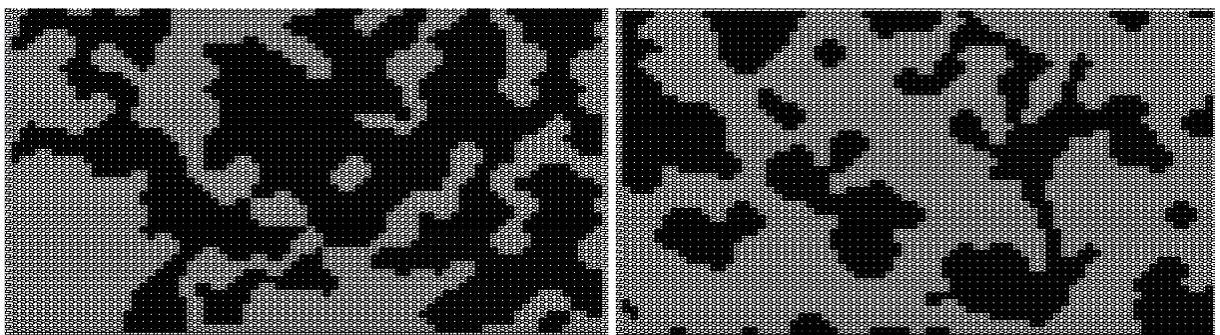
Generacja poziomu trzeciego oparta jest o automat komórkowy [23], będący kwadratową siatką pól mogących być żywymi lub martwymi, których kolejny stan zależy od poprzedniego oraz od ustalonych warunków. W kreowaniu mapy za pomocą tej metody za żywe uznano pola będące podłogą, a za martwe pola będące ścianą. W pierwszym kroku każde pole na mapie w losowy sposób staje się żywe lub martwe. W kolejnych krokach o ustalonej liczbie, stan każdej komórki na mapie zmienia się według tego czy jest ona żywa czy martwa, oraz ilu posiada żywych sąsiadów – pól stykających się z daną komórką. Opisuję to następujące reguły:

- jeśli komórka jest żywa oraz posiada mniej niż trzech żywych sąsiadów to w kolejnym kroku będzie martwa;

- jeśli komórka jest martwa i posiada więcej niż czterech żywych sąsiadów to w kolejnym kroku będzie żywa.

Ostatnim krokiem jest usunięcie wszystkich niepołączonych ze sobą obszarów, poza największym.

Za pomocą manipulowania zasadami tego algorytmu można w łatwy sposób zmieniać wygląd wygenerowanych plansz. Przykładowo na rysunku 3.16 po lewej stronie znajduje się mapa wygenerowana według standardowych zasad użytych w grze, a po prawej mapa, w której podłoga uznawana jest za martwą komórkę, a ściana za żywą.



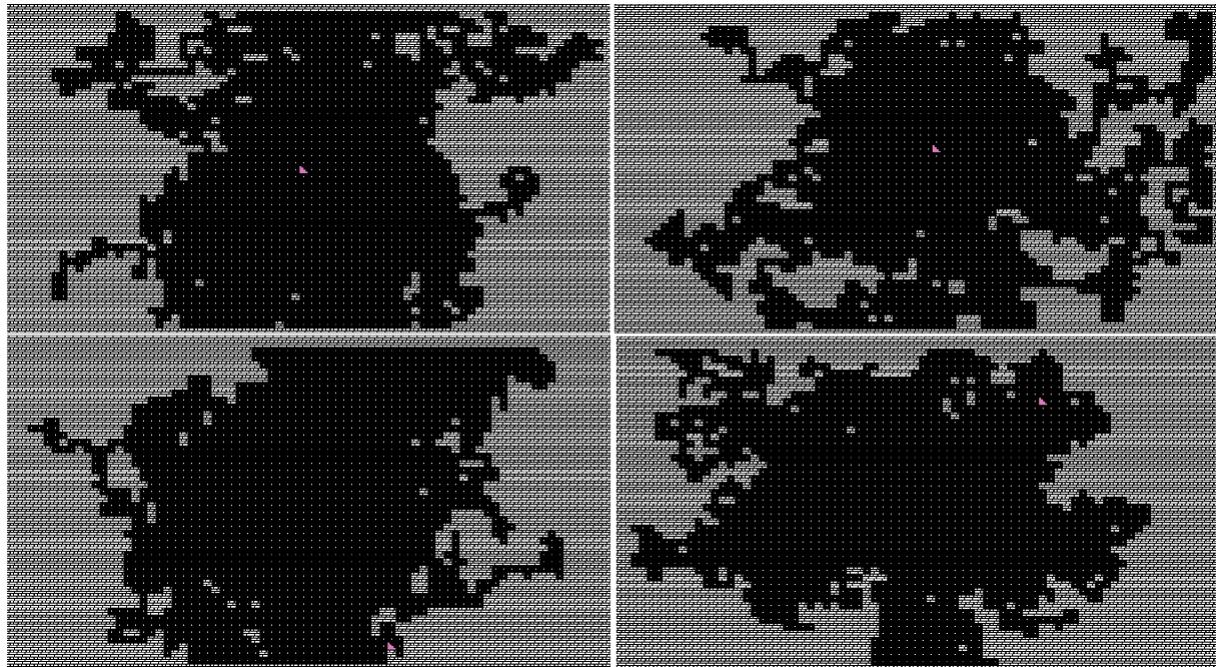
Rysunek 3.16: Przykład manipulacji zasad algorytmu opartego o automat komórkowy

### 3.3.4. Poziom czwarty – Drunkard Walk

Generator map użyty w poziomie czwartym oparty jest o algorytm Drunkard Walk. W celu przystępniejszego wytłumaczenia zasad tego algorytmu posłużono się postacią górnika, który chodząc po planszy wykopuje kolejne pola, zamieniając je w podłogi. W implementacji użytej na potrzeby stworzonej gry, algorytm wykonuje się w następujących krokach:

- wszystkie pola na mapie zamieniane są na ściany,
- następnie dopóki określona przestrzeń mapy nie będzie podłogą wykonuje się następujące kroki:
  - 1) w centralnym polu mapy umieszcza się górnika;
  - 2) w określonej liczbie ruchów górnik porusza się o jedno pole w losowym kierunku a każde pole, po którym przejdzie zamienia się w podłogę;
  - 3) jeśli górnikowi skończyły się ruchy, a mapa wciąż nie zawiera określonej liczby miejsc nie będących ścianami algorytm powraca do kroku 1.

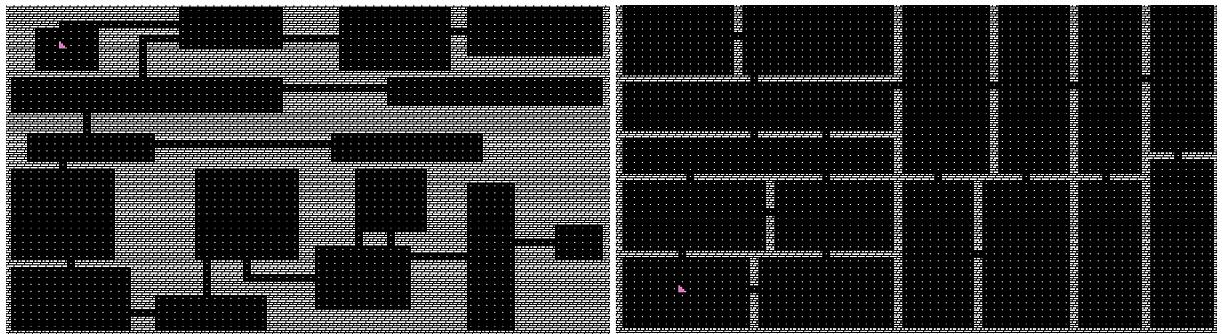
W wyniku tego algorytmu otrzymać można mapy, których przykłady zaprezentowano na rysunku 3.17



Rysunek 3.17: Przykład działania generatora map opartego o algorytm Drunkard Walk

### 3.3.5. Poziom piąty – Binary Space Partitioning

Poziom piąty korzysta z tej samej metody Binary Space Partitioning, co poziom drugi, lecz pokoje w nim zajmują całe obszary przypisane do węzłów będących liśćmi drzewa. Pozwala to na wygenerowanie mapy podzielonej na pokoje bezpośrednio ze sobą połączone, bez użycia korytarzy. Na rysunku 3.18 zaprezentowano porównanie mapy z poziomu drugiego z mapą z poziomu piątego.



Rysunek 3.18: Porównanie map poziomu drugiego po lewej i piątego po prawej, wygenerowanych za pomocą metody BSP

## Podsumowanie i wnioski końcowe

Roguelike to gatunek, który z racji częstego używania losowości pozwala na interesującą i unikalną rozgrywkę. W niniejszej pracy pokazano, że mimo prostej grafiki i interfejsu, gry te mogą oferować ciekawe mechaniki rozgrywki oraz różnorodne, proceduralnie generowane poziomy. Gry tego typu nie są popularne, ale istnieją nowoczesne, bardziej popularne gry inspirujące się tym gatunkiem.

W ramach niniejszej pracy zrealizowane zostały założone cele:

- zaprogramowano grę roguelike o założeniach zgodnych z klasycznymi rozgrywkami tego gatunku;
- omówiono implementację mechanizmów rozgrywki;
- w grze użyte zostały mapy generowane w sposób proceduralny;
- omówiono różne metody użytych generatorów proceduralnych map.

Zrealizowano też dodatkowy cel pracy – dokonanie charakterystyki porównawczej wybranych gier. Opisano grę uznawaną za pierwszą w gatunku roguelike – Rogue, grę Caves of Qud będącą nowoczesnym rozwinięciem klasycznych rozgrywek oraz The Binding of Isaac – grę implementującą wybraną część założeń gatunku.

Do gry wprowadzono nowe rozwiązania takie jak system testowania i prezentowania proceduralnie generowanych map, system rozmieszczania na mapach przeciwników i przedmiotów oparty o wydzielone obszary mapy oraz tablicę obiektów możliwych do pojawiения się na danym poziomie.

Podczas realizacji pracy napotkano kilka problemów. Pierwszy z nich związany był z wyświetlaniem aktualnego stanu mapy w trakcie pracy generatora w celu prezentacji kroków proceduralnego generowania map. Aby tego dokonać konieczne było zapisywanie w trakcie generacji aktualnych stanów mapy w kluczowych momentach do wektora będącego historią generacji. W każdym generatorze zaimplementowano możliwość tworzenia takiej historii, którą mógł on zwrócić po skończeniu generowania mapy. Wyświetlenie kroków polegało później na wyznaczeniu kolejnych pozycji z tego wektora.

Kolejna trudność polegała na tym, iż użyty rozmiar map oraz rozmiar kwadratów na mapie 16x16 pikseli okazał się mało wyraźny, co tworzyło problemy w rozróżnieniu poszczególnych pól na mapie gry. Aby uniknąć zmiany rozmiarów map, które wcześniej dostosowane były do rozmiaru okna zastosowano w trakcie gry znaczne przybliżenie wycentrowane na postać gracza i użyto grafik w formacie 32x32 piksele. Aby tester generatorów map wciąż mógł być w stanie przedstawić całą mapę na ekranie gry zastosowano oryginalne grafiki 16x16.

Gra wykonana na potrzeby niniejszej pracy została napisana w sposób umożliwiający łatwą jej rozbudowę. Aplikację można rozbudować o następujące elementy:

- większa liczba przedmiotów i ich efektów – obecnie w grze nie ma wielu przedmiotów. Dodanie nowych ich typów z pewnością urozmaiciłoby rozgrywkę. Jako przykład można dodać mikstury zwiększające atak lub obronę bohatera na kilka tur;
- nowe metody generowania map – pozwolą na dodanie do gry kolejnych poziomów, co przedłuży i urozmaici rozgrywkę. Możliwe jest też dodanie poziomów opartych o obecnie zaimplementowane metody, ale manipulując ich konfiguracją, lub używając kilku metod na jednej mapie;
- nowe typy przeciwników z interesującymi umiejętnościami – obecnie przeciwnicy poza szlamem nie różnią się od siebie niczym poza wartościami obrony i ataku oraz ekipunkiem. Interesującym dodatkiem do gry byłby przeciwnicy posiadający umiejętności, na przykład lodowy potwór mogący zamrozić gracza, przez co nie będzie on mógł się poruszać przez kilka tur;
- inteligentni przeciwnicy – w aktualnej wersji gry przeciwnicy mogą tylko podążać za graczem, jeśli jest on w ich polu widzenia orazatakować go, jeśli są bezpośrednio obok niego. Bardziej interesującym podejściem byłoby dodanie przeciwników, którzy nie zaatakują gracza, dopóki sami nie zostaną zaatakowani; przeciwnicy mogliby też uciekać przed graczem, jeśli zostanie im bardzo mało punktów zdrowia.

# Literatura

- [1] Felipe Pepe, The CRPG Book: A Guide to Computer Role-Playing Games, Bitmap Books, 2020.
- [2] Harris J.: Exploring Roguelike Games. CRC Press, 2020.
- [3] Daniel Doan, Why Are Modern Roguelike Games So Popular?, <https://gamedevlibrary.com/gamedev-thoughts-why-are-modern-roguelike-games-so-popular-53c1f5a05485>.
- [4] Michael Toy, Glenn Wichman, Ken Arnold, Rogue, 1980, <https://www.mobygames.com/game/rogue>
- [5] American Standards Association, American Standard Code for Information Interchange ASCII, 1963
- [6] Freehold Games, Caves of Qud, 2015, <https://www.cavesofqud.com>
- [7] Angie Kordic, What Exactly is Pixel Art and How Did It Come Back to Life ?, <https://www.widewalls.ch/magazine/pixel-art>
- [8] Edmund McMillen, Nicalis, The Binding of Isaac: Rebirth, 2014, <https://bindingofisaac.com>
- [9] Brian Bucklew (Freehold Games), Math for Game Developers: Tile-Based Map Generation using Wave Function Collapse in 'Caves of Qud', Game Developers Conference, <https://www.gdcvault.com/play/1026263/Math-for-Game-Developers-Tile> .
- [10] <https://github.com/mxgnn/WaveFunctionCollapse> .
- [11] Jason Grinblat (Freehold Games), Procedurally Generating History in 'Caves of Qud', Game Developers Conference, <https://www.gdcvault.com/play/1024990/Procedurally-Generating-History-in-Caves> .
- [12] [https://store.steampowered.com/app/333640/Caves\\_of\\_Qud/](https://store.steampowered.com/app/333640/Caves_of_Qud/)
- [13] [https://store.steampowered.com/app/250900/The\\_Binding\\_of\\_Isaac\\_Rebirth/](https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/)

- [14] Stefanie Fogel, ‘The Binding of Isaac’ Card Game Surpasses 1 Million on Kickstarter in Just Over a Week, <https://variety.com/2018/gaming/news/the-binding-of-isaac-four-souls-kickstarter-1202868455/>
- [15] Steve Klabnik , Carol Nichols, The Rust Programming Language, No Starch Press, 2019.
- [16] Ben Fenwick, C++ v. Rust -Speed, Safety, & Community Comparison, <https://devetry.com/blog/c-v-rust-speed-safety-community-comparison/> .
- [17] Robert C. Martin, Czysty kod. Podręcznik dobrego programisty, Helion, 2014.
- [18] Microsoft, Visual Studio Code, 2015, <https://code.visualstudio.com> .
- [19] Igara Studio S.A, Aseprite, 2001, <https://www.aseprite.org>.
- [20] Richard Lord, Why use an Entity Component System architecture for game development?, <https://www.richardlord.net/blog/ecs/why-use-an-entity-framework.html> .
- [21] [http://www.roguebasin.com/index.php/Basic\\_BSP\\_Dungeon\\_generation](http://www.roguebasin.com/index.php/Basic_BSP_Dungeon_generation) .
- [22] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Algorytmy i struktury danych, Helion 2003.
- [23] Krzysztof Kułakowski, Automaty komórkowe, Akademia Górnictwo-Hutnicza im. St. Staszica. Ośrodek Edukacji Niestacjonarnej, 2000.
- [24] Gene Peterson, Drunken Walk Algorithm, <https://blog.bitrageous.io/drunken-walk/> .

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza  
Wydział Elektrotechniki i Informatyki

Rzeszów, 2022

## **STRESZCZENIE PRACY DYPLOMOWEJ INŻYNIERSKIEJ**

### **PROCEDURALNE GENEROWANIE MAP Z WYKORZYSTANIEM W GRZE TYPU ROGUELIKE**

Autor: Michał Majda, nr albumu: EF-157100

Opiekun: dr hab. inż. Maciej Kusy, prof. PRz

Słowa kluczowe: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po polsku

RZESZOW UNIVERSITY OF TECHNOLOGY  
Faculty of Electrical and Computer Engineering

Rzeszow, 2022

### **BSC THESIS ABSTRACT**

### **TEMAT PRACY PO ANGIELSKU**

Author: Michał Majda, nr albumu: EF-157100

Supervisor: (academic degree) Imię i nazwisko opiekuna

Key words: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po angielsku