Spis treści

[**1.** **Wprowadzenie** 2](#_Toc529888825)

[**1.1.** **Tło pracy** 2](#_Toc529888826)

[**1.2.** **Cel i zakres pracy** 2](#_Toc529888827)

[**1.3.** **Gry typu roguelike** 2](#_Toc529888828)

[**2.** **Istniejące rozwiązania** 3](#_Toc529888829)

[**3.** **Założenia projektowe** 3](#_Toc529888830)

[**3.1.** **Wymagania funkcjonalne** 3](#_Toc529888831)

[**3.2.** **Wymagania niefunkcjonalne** 3](#_Toc529888832)

[**4.** **Projekt gry** 3](#_Toc529888833)

[**5.** **Implementacja** 3](#_Toc529888834)

[**5.1.** **Tworzenie labiryntu** 3](#_Toc529888835)

[**5.1.1.** **Koncepcja** 3](#_Toc529888836)

[**5.1.2.** **Algorytm generujący** 5](#_Toc529888837)

[**5.2.** **Gracz** 5](#_Toc529888838)

[**5.3.** **Przeciwnicy** 5](#_Toc529888839)

[**5.4.** **Pułapki** 7](#_Toc529888840)

[**5.5.** **Rozgrywka// ogolnie o polaczeniu wszytskigo** 7](#_Toc529888841)

[**5.6.** **Serializacja danych** 7](#_Toc529888842)

[**6.** **Prezentacja rozwiązania** 7](#_Toc529888843)

[**7.** **Podsumowanie** 7](#_Toc529888844)

# **Wprowadzenie**

# **Tło pracy**

# **Cel i zakres pracy**

Celem pracy jest utworzenie gry zręcznościowej typu roguelike na platformę Android. Gra skierowana jest do ludzi młodych, którzy wykazują największe zainteresowanie grami mobilnymi. Rozgrywka polega na przechodzeniu poziomów, które są labiryntami. Przejście poziomu polega na dotarciu żywym do wyjścia z labiryntu. Mapa wypełniona jest przeciwnikami, z którymi gracz nie może walczyć bezpośrednio. Wybór sposobu przejścia poziomu zależny jest od indywidualnych preferencji użytkownika. Wrogowie znajdujący się w labiryncie są dla gracza zagrożeniem, które może eliminować za pomocą pułapek. Gracz może wabić przeciwników do wcześniej przygotowanych sideł lub dynamicznie przedzierać się przez mapę zastawiając pułapki na goniących go już przeciwników. Musi mieć jednak na uwadze, że niepowodzenie, tj. zabicie przez potwory, skutkuje tzw. „permanentną śmiercią”. Oznacza to mus rozpoczęcia gry od samego początku, tracony jest cały dotychczasowy postęp. Ze względu na charakterystykę gry i brak fabuły możliwe jest przeprowadzenie krótkiej rozgrywki, np. w czasie przerwy.

# **Gry typu roguelike**

Gry typu roguelike są podgatunkiem gier RPG (Role-playing game). Gatunek RPG charakteryzuje się tym, że gracz wciela się w postać, odgrywając pewną role. Ten rodzaj gier jest komputerową wersja papierowych gier RPG takich jak „Dungeons and Dragons”, gdzie gracze tworzyli swoje postacie wraz ze statystykami oraz opisem a następnie rozgrywali scenariusze jako te postacie. Większość takich papierowych RPGów pochodzi ze Stanów Zjednoczonych, przez co nie są znane w Europie. Roguelike jako gatunek charakteryzuje się następującymi cechami:

* Podział gry na oddzielne poziomy
* Proceduralnie generowane poziomy
* Rozgrywka podzielona na tury
* Permanentna śmierć
* Prosta grafika
* Brak lub szczątkowa fabuła

Cechy te jak i nazwa gatunku pochodzą od gry „Rogue” (roguelike czyli podobny do rogue) która powstała w 1980 roku. W tej grze gracz przechodził przez labirynt pokoi, pokonując potwory oraz poszukując skarbu. Gra ta była odpalana w konsoli ponieważ grafikę stanowiły znaki ASCII



Grafika 2.2.1 przedstawia przykładowy ekran gry „Rogue”

By zaliczyć grę do danego gatunku nie musi ona posiadać wszystkich jego cech, nie są to dokładnie zdefiniowane granice. Dzisiejsze gry z gatunku roguelike wyglądają zupełnie inaczej od pierwowzoru w aspekcie graficznym. Jednak starają się zachować pozostałe cechy tego gatunku.

# **Istniejące rozwiązania**

# **Założenia projektowe**

# **Wymagania funkcjonalne**

# **Wymagania niefunkcjonalne**

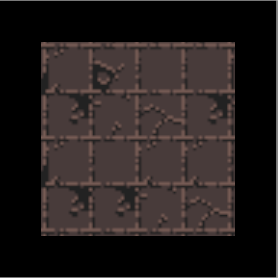
# **Projekt gry**

# **Implementacja**

# **Tworzenie labiryntu**

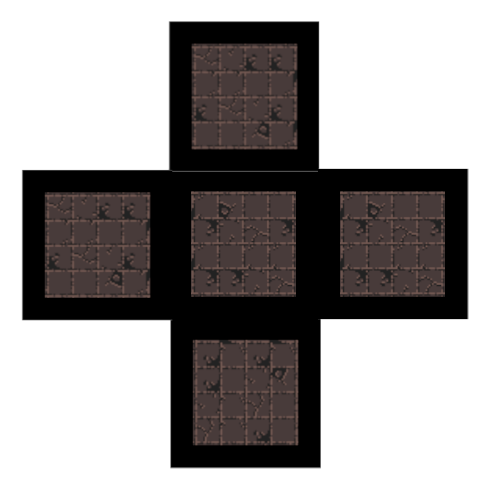
# **Koncepcja**

Podstawą do stworzenia labiryntu w grze jest graf nieskierowany oraz nieposiadający wag, gdzie maksymalny stopień dla każdego wierzchołka to 4 a minimalny 1. Sąsiadem każdego wierzchołka jest wierzchołek znajdujący się bezpośrednio obok danego, na linii poziomej i pionowej w graficznej reprezentacji grafu.



Grafika x.1

Grafika x.1 przedstawia graficzną reprezentację wierzchołka grafu. Czarna obwódka reprezentuje ścianę, element przez który gracz nie może przejść. Natomiast na obszarze brązowym wewnątrz ramki gracz może się poruszać.



Grafika x.2

Na grafice x.2 przedstawiony jest przykładowy graf składający się z pięciu wierzchołków. Element środkowy jest sąsiadem każdego innego elementu zawartego w grafie.

Wygenerowany labirynt musi spełniać kilka ograniczeń:

* musi być możliwe dotarcie do każdego wierzchołka grafu
* z każdego wierzchołka musi istnieć dokładnie jedna ścieżka do każdego innego wierzchołka

Ograniczenie 1 zostało określone, by wygenerowany labirynt posiadał taką samą liczbę wierzchołków jak graf z którego został wygenerowany. Sprawia to, że wielkość wygenerowanego labiryntu jest przewidywalna. Ograniczenie 2 zostało określone by zapewnić brak pętli w labiryncie, co uniemożliwi graczowi uciekanie przed przeciwnikami w łatwy sposób.

Algorytmem który umożliwia wygenerowanie labiryntu zgodnego z ograniczeniami jest zmodyfikowany algorytm Prima. Modyfikacja będzie polegała na zmianie kryterium wyboru wierzchołka, który ma zostać w następnej kolejności dodany do już utworzonego minimalnego drzewa rozpinającego. W podstawowej wersji algorytmu, wierzchołek był wybierany z wierzchołków osiągalnych z minimalnego drzewa rozpinającego. Wierzchołki te są dodawane do kolejki priorytetowej, gdzie najwyższy priorytet posiada wierzchołek o najmniejszym koszcie dotarcia do minimalnego drzewa rozpinającego. Wykorzystana implementacja nie posiada kolejki, a wybór następuje poprzez wylosowanie jednego z dostępnych wierzchołków. Modyfikacja algorytmu znacząco zmniejsza złożoność obliczeniową algorytmu. Podstawowa wersja algorytmu posiada złożoność

Od - O( |E| + |V| log|V| )

Do - O( |E| log|V| )

(gdzie |E| to liczba krawędzi a |V| to liczba wierzchołków)

w zależności od implementacji kolejki priorytetowej na której operacja są najbardziej złożone obliczeniowo. Zrezygnowanie z kolejki zmniejsza złożoność do O(|V|).

Do zaimplementowania algorytmu wykorzystano klasę Wierzchołek która posiada następujące pola:

* czyOdwiedzony – oznacza czy wierzchołek został odwiedzony, czyli jest potencjalnym kandydatem na element krawędzi. Może przyjmować wartości typu prawda fałsz
* listaSąsiedztwa – lista sąsiadujących wierzchołków.

wierzchołkiBrzegowe – lista wierzchołków, do których możemy się dostać z wierzchołków odwiedzonych, które znajdują się w liście krawędzieMDR

Implementacja zmodyfikowanego algorytmu Prima w pseudo-kodzie:

krawędzieMDR = pusta lista par wierzchołków

wierzchołkiBrzegowe = pusta lista wierzchołków

obecnyWierzchołek = losowy wierzchołek grafu

obecnyWierzchołek.czyOdwiedzony = prawda

**dla każdego** elementu z listy obecnyWierzchołek.listaSąsiedztwa:

wierzchołkiBrzegowe.Dodaj(element)

**dopóki** lista wierzchołkiBrzegowe nie jest pusta:

obecnyWierzchołek = losowy wierzchołek z wierzchołkiBrzegowe

wierzchołkiBrzegowe.Usuń( obecnyWierzchołek )

obecnyWierzchołek.czyOdwiedzony = prawda

krawędzieMDR.Dodaj ((obecnyWierzchołek , losowy odwiedzony element z listy

obecnyWierzchołek.listaSąsiedztwa) )

**dla każdego** elementu z listy obecnyWierzchołek.listaSąsiedztwa:

**jeśli** element.czyOdwiedzony == fałsz **i** sąsiad nie jest w wierzchołkiBrzegowe:

wierzchołkiBrzegowe.Dodaj( sąsiad )

Labirynt jest reprezentacją grafu o krawędziach znajdujących się w liście krawędzieMDR.

# **Algorytm generujący**

# **Gracz**

# **Przeciwnicy**

Przeciwnicy są obiektami w przestrzeni gry które są podzielone na typy. Każdy typ posiada unikalne cechy takie jak:

* wygląd,
* maxHP - liczba punktów życia,
* damage - obrażenia,
* minTimeBetweenAttacks- minimalny czas pomiędzy atakami,
* speedModifier - modyfikator szybkości poruszania,
* lostCounterMax – ilość sekund w których przeciwnik kontynuuje poruszanie się po zgubieniu gracza,
* playerFollowersCount – ilość „cieni” gracza za którymi podąża przeciwnik (dokładniej wyjaśnione w dalszej części)

Na początku każdego poziomu przeciwnicy są losowo rozmieszczani w labiryncie. Ich ustawienie musi spełniać pewne kryteria:

* nie mogą znajdować się na obszarze tego samego wierzchołka grafu co gracz
* na obszarze jednego wierzchołka nie może znajdować się więcej niż jeden przeciwnik / grupa przeciwników

Każdy poziom posiada taki sam algorytm rozmieszczania przeciwników, jednak różnią się dane wejściowe które są unikalne dla każdej mapy. Na dane wejściowe składa się lista par gdzie jednym elementem jest obiekt przeciwnika, natomiast drugim wartość liczbowa na podstawie której określa się prawdopodobieństwo wylosowania. Kolejnym elementem danych wejściowych jest całkowita liczba przeciwników którzy pojawią się na danym poziomie.

Implementacja algorytm wyboru kolejnego typu przeciwnika oraz wylosowania jego pozycji w pseudo-kodzie:

sumaPunktów = suma drugich elementów listy parametrów wejściowych

listaWierzchołków = wierzchołki gdzie liczba sąsiadów jest większa od 1

ilośćPrzeciwników **razy**:

losowaWartość = losowa wartość od 0 do sumaPunktów

sumaTymczasowa = 0

**długość(**parametryWejściowe**) razy (iterator i):**

sumaTymczasowa = sumaTymczasowa + parametryWejściowe[i].wartość

**jeśli** losowaWartość < sumaTymczasowa:

wylosowanyWierzchołek = losowy wierzchołek z listaWierzchołków

listaWierzchołków.Usuń(wylosowanyWierzchołek)

stwórz przeciwnika typu parametryWejściowe[i].typPrzeciwnika

**przerwij pętle**

Przeciwnicy, jako jedna z przeszkód jakie musi pokonać gracz by ukończyć poziom, powinni stanowić pewne wyzwanie. Przeciwnik stoi w miejscu do czasu, aż gracz podejdzie do niego na określoną odległość, która zależy od typu przeciwnika. Następnie zaczyna podążać za graczem według algorytmu przedstawionego poniżej. Przeciwnik może przestań podążać za graczem w dwóch przypadkach.

* Punkty życia przeciwnika spadną do zera
* Gracz ucieknie na odpowiednią odległość

Gracz posiada 7 „cieni”, są to niewidzialne obiekty poruszające się dokładnie tak jak gracz, jednak z pewnym opóźnieniem.

//rysunek z widocznymi cieniami

Implementacja algorytmu podążania za graczem w pseudo-kodzie. Algorytm wykonuje się w każdej klatce podążania przeciwnika za graczem:

forceToApply - wartość to siła przykładana do przeciwnika by się poruszał

playerFollowers - gracz oraz cienie ustawione w kolejności od gracza do najbardziej opóźnionego cienia

followerToFollow – numer cienia za którym powinien podążać przeciwnik, domyślnie -1

**długość(**playerFollowers**) razy (iterator i):**

**jeśli**  playerFollowers[i] znajduje się na tym samym wierzchołku co przeciwnik lub ,,,,,,,,,,,,,,,,,,,,,,,wierzchołku dostępnym z danego

followerToFollow = i

**przerwij pętle**

**jeśli** followerToFollow nie jest równe -1**:**

diff = playerFollowers[followerToFollow].pozycja – przeciwnik.pozycja

diff = wersor z diff

obrót przeciwnika w kierunku gracza

forceToApply = diff \* speedModifier

lostCounter = 0

**w przeciwnym wypadku:**

lostCounter = lostCounter + 1

**jeśli** lostCounter jest równy lostCounterMax \* 30 **:**

przestań śledzić gracza

# **Pułapki**

Gracz poruszając się po labiryncie napotyka przeciwników, niektórych można ominąć na drodze do zakończenia poziomu, inni stoją bezpośrednio na drodze gracza. Pułapki są jednym ze sposobów na interakcje gracza z przeciwnikami.

Pułapka jest to obiekt w świecie gry, który umieszcza gracz. Jej działanie zależy od komponentów z których jest zbudowana. Każda pułapka składa się z dwóch typów komponentów:

* parametryzowany moduł aktywujący, który określa w jaki sposób pułapka może zostać aktywowana, taki moduł musi być tylko jeden .
* komponenty oddziałujące, które określają w jaki sposób pułapka działa na przeciwnika oraz gracza. Takich elementów w pułapce może znajdować się wiele.

Pułapka posiada skuteczny zasięg czyli maksymalną odległość w jakiej może się znajdować przeciwnik lub gracz, by pułapka na niego zadziałała. Aktywacja pułapki polega na wykonaniu instrukcji znajdujących się w komponentach oddziałujących na przeciwnikach oraz graczu, jeśli znajdują się odpowiednio blisko. To, że pułapki działają nie tylko na przeciwników, ale także na gracza, wymusza większą ostrożność w rozmieszczaniu pułapek w labiryncie.

# **Rozgrywka// ogolnie o polaczeniu wszytskigo**

# **Serializacja danych**

# **Prezentacja rozwiązania**

# **Podsumowanie**