Spis treści

[**1.** **Wprowadzenie** 2](#_Toc529965660)

[**1.1.** **Tło pracy** 2](#_Toc529965661)

[**1.2.** **Cel i zakres pracy** 3](#_Toc529965662)

[**1.3.** **Gry typu roguelike** 4](#_Toc529965663)

[**2.** **Istniejące rozwiązania** 5](#_Toc529965664)

[**2.1.** **Generowanie proceduralne** 5](#_Toc529965665)

[**2.2.** **Mobilne gry roguelike** 5](#_Toc529965666)

[**3.** **Założenia projektowe** 7](#_Toc529965667)

[**3.1.** **Wymagania funkcjonalne** 7](#_Toc529965668)

[**3.2.** **Wymagania niefunkcjonalne** 7](#_Toc529965669)

[**4.** **Projekt gry** 7](#_Toc529965670)

[**4.1.** **Tytuł gry** 7](#_Toc529965671)

[**4.2.** **Opis gry** 7](#_Toc529965672)

[**4.2.1.** **Koncepcja gry** 7](#_Toc529965673)

[**4.2.2.** **Gatunek** 7](#_Toc529965674)

[**4.2.3.** **Grupa docelowa** 7](#_Toc529965675)

[**4.3.** **Rozgrywka i mechaniki** 7](#_Toc529965676)

[**4.3.1.** **Rozgrywka** 7](#_Toc529965677)

[**4.3.1.1.** **Postęp w grze** 7](#_Toc529965678)

[**4.3.1.2.** **Cel gry** 7](#_Toc529965679)

[**4.3.2.** **Mechaniki** 7](#_Toc529965680)

[**4.3.2.1.** **Poruszanie się gracza** 7](#_Toc529965681)

[**4.3.2.2.** **Przedmioty – wypdajace z przeciwnikow** 7](#_Toc529965682)

[**4.3.2.3.** **Akcje** 7](#_Toc529965683)

[**4.3.2.4.** **Walka** 7](#_Toc529965684)

[**4.3.2.5.** **Zdobywanie punktów** 7](#_Toc529965685)

[**4.3.2.6.** **Poziom trudności** 7](#_Toc529965686)

[**4.4.** **Zapisywanie stanu gry** 7](#_Toc529965687)

[**4.5.** **Poziomy** 7](#_Toc529965688)

[**4.5.1.** **Poziomy** 7](#_Toc529965689)

[**4.5.2.** **poziom treningowy** 7](#_Toc529965690)

[**4.6.** **Interfejs** 7](#_Toc529965691)

[**4.6.1.** **HUD** 8](#_Toc529965692)

[**4.6.2.** **System sterowania** 8](#_Toc529965693)

[**4.7.** **Elementy techniczne** 8](#_Toc529965694)

[**4.7.1.** **Wymagania sprzętowe** 8](#_Toc529965695)

[**4.7.2.** **Wymagania sieciowe** 8](#_Toc529965696)

[**4.8.** **Podsumowanie** 8](#_Toc529965697)

[**5.** **Implementacja** 8](#_Toc529965698)

[**5.1.** **Tworzenie labiryntu** 8](#_Toc529965699)

[**5.1.1.** **Koncepcja** 8](#_Toc529965700)

[**5.1.2.** **Algorytm generujący** 10](#_Toc529965701)

[**5.2.** **Gracz** 10](#_Toc529965702)

[**5.3.** **Przeciwnicy** 10](#_Toc529965703)

[**5.4.** **Pułapki** 12](#_Toc529965704)

[**5.5.** **Rozgrywka// ogolnie o polaczeniu wszytskigo** 12](#_Toc529965705)

[**5.6.** **Serializacja danych** 12](#_Toc529965706)

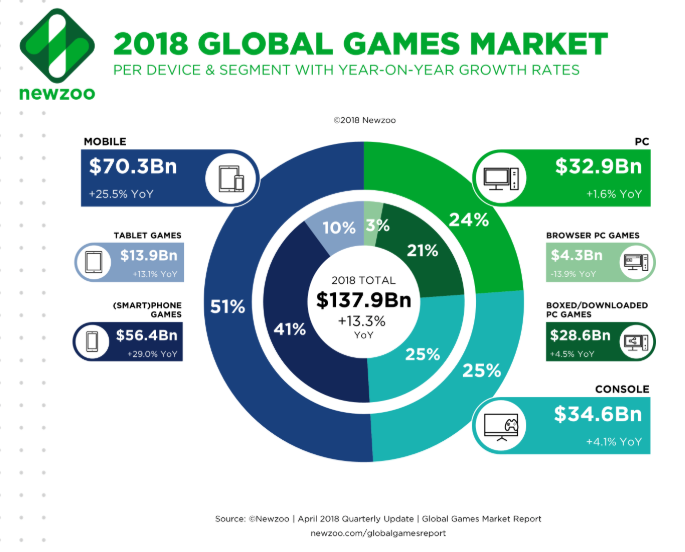
[**6.** **Prezentacja rozwiązania** 12](#_Toc529965707)

[**7.** **Podsumowanie** 12](#_Toc529965708)

# **Wprowadzenie**

# **Tło pracy**

Rynek gier z każdym rokiem zwiększa swoje dochody. Coraz większą część tego rynku zajmują gry mobilne. W 2018 roku jest to przewidywane 51% wraz z dochodami rzędu 70.3 miliarda dolarów. Rynek rozwija się bardzo dynamicznie a w szczególności jego segment mobilny, od ostatniego roku dochody z tego segmentu wzrosły o 25.5%. W porównaniu do innych segmentów: rynek PC wzrost 1.6% oraz konsole wzrost 4.1%, jest to wzrost bardzo duży.



Grafika 1.1 wykaz rynku gier wraz z dochodami

Prognozy na następne lata przewidują dalszy wzrost udziału gier mobilnych w rynku, do 59% w 2021 roku. Są to jednak przewidywania oparte na aktualnych trendach wzrostu dla segmentu mobilnego.

W takim środowisku istnieje duże zapotrzebowanie na produkty IT jakimi są gry mobilne. Dodatkowo łatwość publikacji takich aplikacji w „sklep play” sprawia, że nawet małe studia deweloperskie mogą dostarczać produkt do szerokiego grona odbiorców.

# **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.

# **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 rolę. Ten rodzaj gier jest komputerową wersją 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 uruchamiana w konsoli ponieważ grafikę stanowiły znaki ASCII



Grafika 1.2 przedstawia przykładowy ekran gry „Rogue”

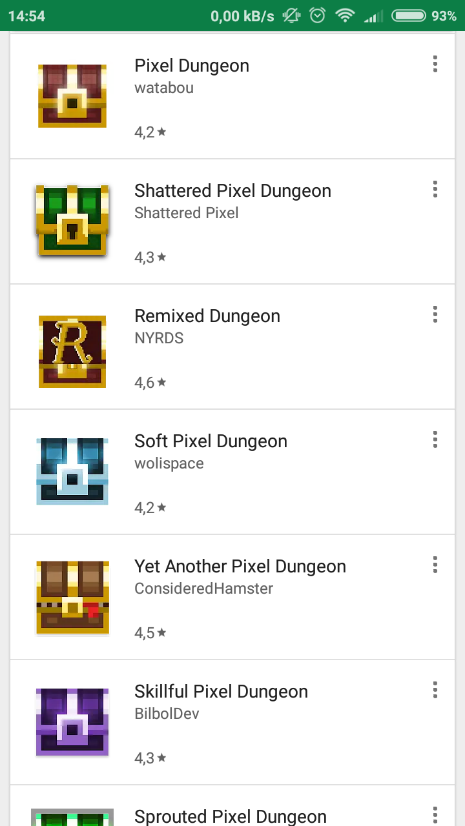
By zaliczyć grę do danego gatunku nie musi ona posiadać wszystkich jego cech, nie są to dokładnie zdefiniowane granice. Dzisiejsze gry z gatunku roguelike wyglądają zupełnie inaczej od pierwowzoru w aspekcie graficznym. Jednak starają się zachować pozostałe cechy tego gatunku.

# **Istniejące rozwiązania**

# **Generowanie proceduralne**

# **Mobilne gry roguelike**

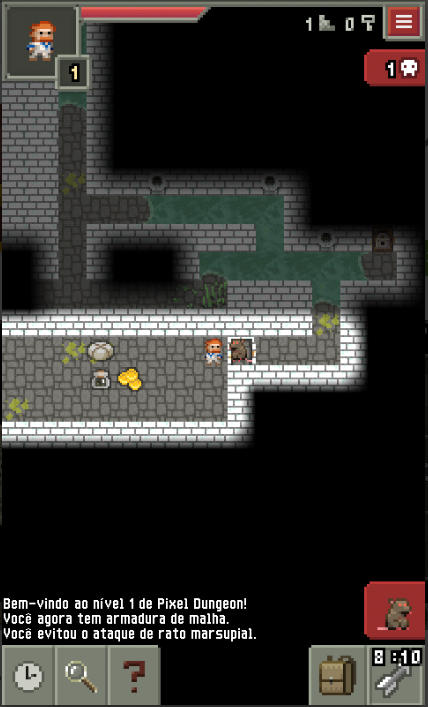
Gry tego typu są bardzo popularne. Istnieje gra, której kod jest dostępny opensource. Jest powszechnie wykorzystywana przez twórców jako baza dla nowego produktu. Po wprowadzeniu drobnych modyfikacji wydają oni grę jako inny, nowy produkt. Mimo wielu podobieństw gry powstałe w ten sposób zdobywają dużą popularność. Dowodzi to temu, że nawet niewiele zmieniona gra może znaleźć grono odbiorców.



Grafika 2.1 lista gier powstałych na bazie rozwiązania „Pixel Dungeon”

Przykładowe rozwiązania:

* Pixel Dungeon PL – jest to gra, która spełnia wszystkie założenia gatunku roguelike (które zostały opisane w poprzednim rozdziale). Na początku gra daje nam wybór klasy postaci wraz z krótkim opisem jej mocnych i słabych stron. Następnym krokiem jest już przechodzenie podziemi. Podziemia te są proceduralnie generowane. W pomieszczeniach gracz napotyka przeciwników oraz skarby. Cały poziom nie jest widoczny od początku, gracz odkrywa go podczas poruszania się. W grze obecny jest rozbudowany system ekwipunku z wieloma możliwymi do zdobycia przedmiotami. Gra posiada rozgrywkę w systemie turowym. Gra powstała w ramach tej pracy inżynierskiej nie posiada takiego systemu, gracz wykonuje wszystkie akcje w czasie rzeczywistym. Cel gry nie jest podany na początku rozgrywki, jest to prawdopodobnie przejście jak największej ilości poziomów, lub fabułę poznaje się w trakcie gry.



Grafika 2.2 wygląd gry pixel dungeon PL

* Caves (roguelike) –

# **Założenia projektowe**

# **Wymagania funkcjonalne**

# **Wymagania niefunkcjonalne**

# **Projekt gry**

# **Tytuł gry**

Tytuł gry to Trapper’s Quest. Tłumacząc z angielskiego oznacza to: zadanie trapera. Jest to bezpośrednie nawiązanie do głównej mechaniki walki w grze czyli zastawiania pułapek.

# **Koncepcja gry**

# **Gatunek**

Gra jest przedstawicielem gatunku roguelike, posiada także cechy gry zręcznościowej oraz strategicznej.

# **Grupa docelowa**

Grupą docelową są ludzie posiadający urządzenia z Androidem, powinni także lubić wyzwania. Gra jest w języku angielskim wiec jego znajomość jest również wymagana.

# **Graficzny aspekt gry**

Gra jest typu 2D oznacza to, że wszystkie obiekty widoczne dla gracza są płaskimi grafikami. Gracz widzi całą scenę z góry, taki widok pozwala pokazać większy obszar co daje graczowi więcej informacji. Grafiki wykorzystane do stworzenia gry to wielokąty dostępne w unity lub grafiki pobrane z Internetu.

# **Rozgrywka i mechaniki**

# **Rozgrywka**

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. W grze znajduje się kilka typów pułapek. Na początku gracz posiada dostęp tylko do jednej z nich, następne zdobywa osiągając określone poziomy punktów. Punkty dostaje się za pokonywanie przeciwników. 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 przymus 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.

# **Postęp w grze**

Gra jest zbudowana z poziomów które gracz przechodzi. Postępem jest przejście danego poziomu bez zginięcia.

# **Cel gry**

Celem gry jest przejście wszystkich poziomów.

# **Mechaniki**

# **Poruszanie się gracza**

Gracz może poruszać się po mapie w osi pionowej oraz poziomej. Nie może przechodzić przez ściany. Przez naciśnięcie odpowiedniego przycisku gracz może wykonać tzw. „roll”. Postać przez jedną sekundę porusza się w tym samym kierunku co podczas wciśnięcia przycisku z szybkością większą niż normalnie. W tym czasie nie może zmienić kierunku poruszania się.

# **Akcje**

# **Zdobywanie punktów**

Gracz zdobywa punkty przechodząc poziom. Za każdy poziom jest inna ilość punktów, zależna od ilości przeciwników na danym poziomie. Inny sposób na zdobywanie punktów to pokonywanie przeciwników, gdy przeciwnik umiera gracz otrzymuje punkty równe połowie punktów życie przeciwnika. Ilość punktów to zawsze liczba całkowita, w przypadku gdy przeciwnik posiada nieparzystą liczbę punktów zdrowia wartość zostaje zaokrąglona w górę.

# **Walka**

Walka z potworami jest głównym elementem rozgrywki, ponieważ jest to najczęściej wykonywana aktywność. Ponieważ gracz nie może bezpośrednio zadać przeciwnikowi obrażeń musi on wykorzystywać pułapki. Gracz musi jednak uważać ponieważ potwór zadaje obrażenia za pierwszym razem gdy dotknie gracza. Następne ataki mogą nastąpić co pewien okres czasu unikalny dla każdego typu przeciwnika.

# **Przedmioty – wypdajace z przeciwnikow**

# **Poziom trudności**

# **Zapisywanie stanu gry**

# **Poziomy**

# **Poziomy**

# **poziom treningowy**

# **Interfejs**

# **HUD**

# **System sterowania**

# **Elementy techniczne**

# **Wymagania sprzętowe**

# **Wymagania sieciowe**

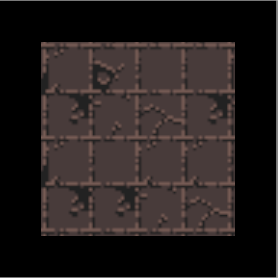
# **Podsumowanie**

# **Implementacja**

# **Tworzenie labiryntu**

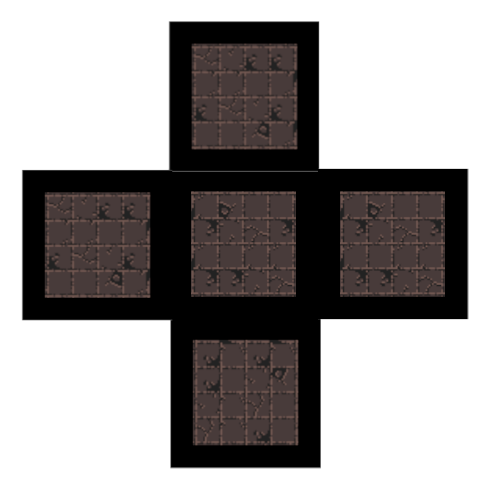
# **Koncepcja**

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 operacje 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 algorytmu 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 przestać 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**