Spis treści

[Streszczenie 3](#_Toc405465897)

[Abstract 4](#_Toc405465898)

[1 Wprowadzenie 5](#_Toc405465899)

[1.1 Wprowadzenie do dziedziny pracy dyplomowej 5](#_Toc405465900)

[1.2 Technologie, algorytmy, narzędzia i inne aplikacje 5](#_Toc405465901)

[1.3 Proces tworzenia pracy dyplomowej 5](#_Toc405465902)

[2 Projekt sytemu 5](#_Toc405465903)

[2.1 Cel i przeznaczenie systemu 5](#_Toc405465904)

[2.1.1 Założenia i cel tworzenia aplikacji 5](#_Toc405465905)

[2.1.2 Planowany sposób rozpowszechnienia aplikacji 5](#_Toc405465906)

[2.1.3 Ograniczenia dotyczące projektu aplikacji 6](#_Toc405465907)

[2.2 Specyfikacja wymagań użytkowych 6](#_Toc405465908)

[2.2.1 Ogólna charakterystyka systemu 6](#_Toc405465909)

[2.2.2 Zbiór wymagań użytkowych 6](#_Toc405465910)

[2.2.3 Zakres funkcjonalny (wymagania funkcjonalne) 6](#_Toc405465911)

[2.2.4 Środowisko pracy systemu 7](#_Toc405465912)

[2.2.5 Wymagania jakościowe 7](#_Toc405465913)

[2.2.6 Wymagania projektowo-wdrożeniowe 7](#_Toc405465914)

[2.2.7 Kryteria akceptacji 8](#_Toc405465915)

[2.3 Projekt systemu 8](#_Toc405465916)

[2.3.1 Projekt architektury systemu 8](#_Toc405465917)

[2.3.2 Przypadki użycia 8](#_Toc405465918)

[2.3.3 Diagram klas 8](#_Toc405465919)

[2.3.4 Projekt interfejsu użytkownika 8](#_Toc405465920)

[2.3.5 Wybór środowiska implementacji 8](#_Toc405465921)

[2.3.6 Projekt danych 8](#_Toc405465922)

[2.3.7 Rozszerzalność aplikacji 8](#_Toc405465923)

[2.3.8 Zastosowane algorytmy 8](#_Toc405465924)

[3 Implementacja, testowanie, walidacja i weryfikacja 8](#_Toc405465925)

[3.1 Implementacja systemu 8](#_Toc405465926)

[3.1.1 Diagram klas 8](#_Toc405465927)

[3.2 Testowanie 9](#_Toc405465928)

[3.3 Walidacja 9](#_Toc405465929)

[3.4 Weryfikacja założeń użytkowych aplikacji 9](#_Toc405465930)

[3.5 Podsumowanie i wnioski 9](#_Toc405465931)

[3.6 Możliwość dalszego rozwoju 9](#_Toc405465932)

[1 Generowanie proceduralne grach 9](#_Toc405465933)

[1.1 Geneza 9](#_Toc405465934)

[1.2 Najczęściej używane algorytmy 9](#_Toc405465935)

[1.3 Przykładowe zastosowania 9](#_Toc405465936)

[2 Generowanie map w grze The Mighty Marian 9](#_Toc405465937)

[2.1 Pożądany efekt, poprawność mapy 9](#_Toc405465938)

[2.2 Automat komórkowy - gra w życie 10](#_Toc405465939)

[2.3 Etapy procesu generowania mapy 11](#_Toc405465940)

[2.3.1 Etap labiryntu 11](#_Toc405465941)

[2.3.2 Położenie przejść między pokojami 13](#_Toc405465942)

[2.3.3 Etap pokoju 14](#_Toc405465943)

[2.3.4 Etap łączenia i wygładzania 15](#_Toc405465944)

[2.3.5 Etap erozji 15](#_Toc405465945)

# Streszczenie

# Abstract

# Wprowadzenie

## Wprowadzenie do dziedziny pracy dyplomowej

## Technologie, algorytmy, narzędzia i inne aplikacje

## Proces tworzenia pracy dyplomowej

# Projekt sytemu

## Cel i przeznaczenie systemu

### Założenia i cel tworzenia aplikacji

Celem naszego projektu jest stworzenie gry przy użyciu silnika Unity, która łączyć będzie elementy survivalu oraz klasycznego RPG. Planujemy stworzyć produkt, który zapewni użytkownikowi wiele godzin niezapomnianej rozrywki i wyjątkową grywalność, jednocześnie będąc wyzwaniem. Grafika będzie łączyć w sobie elementy 2D i 3D. Głównym elementem naszej gry, który ma znacząco poprawić grywalność i wydłużyć czas przyjemnej i pełnej niespodzianek rozgrywki, jest stworzenie map generowanych losowo.

### Planowany sposób rozpowszechnienia aplikacji

Początkowo aplikacja będzie dostępna do pobrania z serwisu dropbox dla osób posiadających link, który umieszczony będzie na portalach społecznościowych, i pod filmikami z gameplayu na serwisie Youtube.

Dodatkowo Unity udostępnia technologię Unity Web Player, która umożliwia uruchamianie gry w oknie przeglądarki internetowej, jeśli tylko użytkownik ma zainstalowaną odpowiednią wtyczkę. Unity nawet udostępnia generator, który produkuje gotową stronę HTML i odpowiedni kod JavaScript z osadzanym obiektem gry. W przyszłości planujemy stworzyć stronę internetową, na której będzie można zapoznać się z dokumentacją projektu, zagrać w grę, a dla użytkowników, którzy nie będą zainteresowani pobieraniem wtyczki Unity Web Player zostanie udostępniona do pobrania za darmo paczka z grą The Mighty Marian, w wersji na system operacyjny Microsoft Windows i Linux. Obecnie jednak przeszkodą są koszty związane z utrzymaniem serwera.

Ponieważ ten temat projektu został wybrany głównie w celu zdobycia cennego doświadczenia ze środowiskiem Unity i tworzeniem gier komputerowych, a nie stworzenia produktu komercyjnego, produkt końcowy dystrybułowany będzie głownie lokalnie, dla osób zainteresowanych, choć będzie również dostępny do pobrania w Internecie.

### Ograniczenia dotyczące projektu aplikacji

W związku z brakiem grafika w zespole, projekt nie będzie skupiał się na elementach graficznych. Postanowiliśmy w miarę mozliwości wykożystywać gotowe elementy graficzne. Dodatkowo jest to nasz pierwszy projekt tworzony w Unity, przez co częsć założeń uległa zmianie w miarę poznawania możliwości i ograniczeń tego środowiska.

Projekt jest testowany przez grupę naszych znajomych, przez co testy mogą być niepełne.

W unity kod jest bardzo silnie powiązany z obiektami w grze, więc diagram klas będzie diagramem obiektów i klas, gdyż tylko w ten sposób można pokazać zależności między nimi

## Specyfikacja wymagań użytkowych

### Ogólna charakterystyka systemu

Gra jest produkcją z pogranicza gatunków Survival, RPG. Stosuje proceduralną generację terenu, a główny nacisk położony będzie na rozgrywkę. Dzięki generowanemu terenowi każde podejście do produkcji oferuje inne doznania. Produkcja jest stworzona w technologii Unity, dzięki czemu posiada zapewnione przez środowisko skalowanie grafiki, w zależności od sprzętu na którym zostanie uruchomiona. Dzięki technologii Unity Web Player można umieścić ją w sieci. Przejście gry powinno zajmować około 30-40 minut, Gra ma działać na systemach operacyjnych wspieranych przez Unity, testowana jest pod systemem MS Windows.

### Zbiór wymagań użytkowych

Gra powinna działać na systemie windows 7 lub nowszym, jak i na systemach Linux wspieranych przez środowisko Unity.

Sterowanie w grze powinno odbywać sie przy pomocy klawiatury i myszki, w możliwie naturalny sposób.

Gra uruchamia się przez standardowy program startowy Unity. Wybór rozdzielczości i jakości efektów powinien mieć odzwierciedlenie w faktycznym wyglądzie gry, jednak powinno byc to zapewnione przez Unity.

### Zakres funkcjonalny (wymagania funkcjonalne)

Gracz ma możliwość poruszania się po świecie gry. Kamera powinna pokazywać świat w rzucie izometrycznym lub podobnym. Kamera powinna porusza się za bohaterem. Świat powinien być generowany proceduralnie. Gra będzie posiadać poziomy, celem bohatera jest dojście do ostatniego z nich, ich ilość będzie losowa. W wygenerowanym świecie pojawiają się wrogowie, są oni rozmieszczani losowo, w ilości zależnej od obecnego poziomu gry. Gra powinna posiadać elementy gry RPG: postać gracza powinna byc opisana statystykami, których zwiększanie powinno odpowiednio modyfikować rozgrywkę. Postać ma posiadac umiejętności, które gracz będzie mógł aktywować by zyskac przewagę w walce. Umiejętności powinny być odpowiednio zbalansowane. Dodatkowo w grze powinien być zaimpemetowany system ekwipunku. Przedmioty powinny odpowiednio modyfikować statystyki postaci. Gra może posiadac minigry które będą znacząco odbiegać od reszty rozgrywki.

### Środowisko pracy systemu

Dzięki zastosowanemu silnikowi gra jest między-platformowa i można ją uruchomić na najbardziej popularnych systemach: Windows, Linux i Macintosh. Jako że Mac jest najmniej popularnym systemem z powyższej trójki, a także z racji braku maszyny z tym systemem na której moglibyśmy przetestować grę, nasza gra działać będzie na systemie Windows 7 i nowszym, oraz na najpopularniejszych dystrybucjach Linuxa. Dodatkowo dzięki darmowej wtyczce unity web player gra może działać w przeglądarce. Obecnie unity web player jest kompatybilny z nowymi wersjami przeglądarek Internet Explorer, Firefox, Chrome, Safari, Opera, a wtyczka do instalacji potrzebuje systemu operacyjnego Windows XP/Vista/7/8.

### Wymagania jakościowe

Gra powinna być utrzymana w stylistyce 8-bitowej, więc grafika nie musi być najwyższej jakości. Gra powinna być stabilna, nie może się zawieszać, powinna być zapewniona obsługa błędów.

### Wymagania projektowo-wdrożeniowe

Projekt aplikacji powinien być gotowy do 15 października.

Wersja alfa gry, która zostanie rozeslana do wybranej grupy osób, zainteresowanych jej przetestowaniem powinna byc gotowa do 31 października.

Wersja beta powinna byc gotowa do 15 listopada. Wersja beta powinna już posiadać zaimplementowane wszystkie funkcjonalności gry. Wersja ta ponownie zostaniw wysłana do testerów.

Aplikacja powinna być wykonana w terminie do 2 grudnia, dokumentacja powinna być gotowa do 5 grudnia.

Program powinien zajmować mniej niż 0.5 GB miejsca na dysku, dzięki czemu będzie szybki w pobraniu, oraz jego ładowanie przez unity web player zajmie mniej czasu.

### Kryteria akceptacji

## Projekt systemu

### Projekt architektury systemu

### Przypadki użycia

### Diagram klas

### Projekt interfejsu użytkownika

### Wybór środowiska implementacji

### Projekt danych

### Rozszerzalność aplikacji

### Zastosowane algorytmy

# Implementacja, testowanie, walidacja i weryfikacja

## Implementacja systemu

### System RPG w grze The Mighty Marian

#### Statystyki opisujące postać

Główny bohater jest opisywany następującymi statystykami: siła, zręczność i inteligencja. Manipulacja nimi wpływa na parametry opisujące postać: punkty życia, manę, regenerację many, szybkość i obrażenia. Dodatkowo postać posiada poziom, punkty doświadczenia i punkty umiejętności.

#### Zdobywanie poziomów

By zdobyć poziom główny bohater musi miec odpowiednią liczbę punktów doświadczenia, obliczaną według wzoru: Ilość punktów doświadczenia potrzebna by zdobyć następny poziom doświadczenia = 100 \* poprzedni poziom doświadczenia

|  |  |
| --- | --- |
| Poziom | Łączna ilość punktów doświadczenia |
| 1  2  3  4  5  6  7  8  9  ... | 0  100  300  600  1000  1500  2100  2800  3600  ... |
| Rysunek 3.1: Tabela doświadczenia | |

Zdobycie każdego poziomu daje bohaterowi 5 punktów umiejętności które może wykożystać na zwiększenie jednej ze statystyk opisujących postać: siły, zręczności i inteligencji. Dodatkowo zdobycie poziomu dodaje 20 punktów maksymalnego życia, zwraca 20 punktów życia, oraz dodaje 20 punktów maksymalnej many i przywraca 20 many.

#### Zdobywanie statystyk

Punkty umiejętności można wykożystać do zwiększenia statystyk. Każdy punkt statystyki zwiększa parametry postaci:

Każdy punkt siły dodaje punkt obrażen z broni białej, zwiększa maksymalny poziom punktów życia o pięć i przywraca pięć punktów życia.

Punkty zręczności zwiększają obrażenia z broni dystansowej, oraz zwiększają prędkość z jaką postać porusza się po mapie.

Inteligencja zwiększa obrażenia z broni magicznej, zwiększa maksymalną pulę many, przywraca część obecnej many oraz przyspiesza jej regeneracje. Dodatkowo zwiększa punkty obrażeń zadawane przez czary.

#### Zdobywanie umiejętności

W miarę zdobywania statystyk, bohater zyskuje dostęp do umiejetności aktywnych. Postać ma pięć miejsc na umiejętności aktywne, są one uzupełniane pierwszą piątką odblokowanych umiejętności.

|  |  |  |  |
| --- | --- | --- | --- |
| Ilość punktów danej statystyki, potrzebna do odblokowania umiejętności. | Siła | Zręczność | Inteligencja |
| 10 | Wzmocnienie: Na dwie sekundy podwaja obrażenia zadawane bronią białą | Przyspieszenie: Przez krótki okres czasu postac porusza się szybicej | Erupcja: Główny bohater przy pomocy swojej umiejętności manipulacji magią powoduje lokalną erupcję lawy która zadaje obrażenia każdemu przeciwnikowi który znajdzie się na jej obszarze |
| 20 | Szał: Przez dwie sekundy postać odzyskuje życie równe sile jej ataku broniom białą. | Leczenie: Przywraca postaci część punktów życia | Wiązka błyskawic: Przez krótki czas podstawowy atak postaci zamienia si w wiązkę błyskawic, która zadaje obrażenia każdemu przeciwnikowi który zostanie nią trafiony |
| 30 | Ogłuszający krzyk: ogłusza przeciwników w otoczeniu bohatera na pięc sekund. | Przebijający strzał: Postać wypuszcza wiązkę energii która zadaje poważne obrażenia każdemu przeciwnikowi którego trafi. | Kula ognia: Bohater wystrzeliwuje ognisty pocisk który eksploduje po doleceniu do celu, lub po kolizji z pierwszym napotkanym obiektem, zadając poważne obrażenia wszystkim przeciwniką w polu rażenia. |
| Rysunek 3.2: Tabela umiejętności | | | |

Erupcja

|  |
| --- |
|  |
|  |

Szał

|  |
| --- |
|  |
|  |

Ogłuszający krzyk

|  |
| --- |
|  |
|  |

Przyspieszenie

|  |
| --- |
|  |
|  |

Przyspieszenie

|  |
| --- |
|  |
|  |

Leczenie

|  |
| --- |
|  |
|  |

Przebijający strzał

|  |
| --- |
|  |
|  |

Erupcja

|  |
| --- |
|  |
|  |

Wiązka błyskawic

|  |
| --- |
|  |
|  |

Kula ognia

|  |
| --- |
|  |
|  |

Umiejętności mają też czasy odnowienia, które są zależne od tego jak potężne one będą. Dodatkowo czary leczące mają dodatkowo zwiększony czas odnowy, by utrudnić rozgrywkę.

#### Obrażenia

Zadawane obrażenia są zależne od rodzaju broni jakiej używa postać gracza, jak i od odpowiedniej statystyki powiązanej z tą bronią. Bazowo bohater zadaje dziesięć obrażeń pojedyńczym atakiem, jednak znajdowanie lepszej broni i zwiększanie umiejętności podnosi ta wartość.

### Ekwipunek

W naszej produkcji zdecywodaliśmy się zastosować gotowy system ekwipunku. Na stronie unity asset store [TODO link] jest dostępny gotowy system, bardzo łatwy w implementacji. Inventory System [TODO link] stworzony przez Brackeys[TODO link] jest najlepszym darmowym rozwiązaniem, niestety z racji braku oficjalnej wersji która była by napisana w C#, zdecydowaliśmy się na zaimportowanie najnoweszej wersji (1.2.2) opartej o javascript. Wiązało się to z problemami z komunikacją między kodem napisanym w javascripcie, a kodem C#. Unity wymaga by w wypadku odwołań między różnymi językami, jeden z nich był skompilowany wcześniej [TODO link]. Niestety zastosowanie tego modułu przeszkodziło w implementacji niektórych funkcjonalności systemu ekwipunku, takich jak pokazywanie wartości obrażeć po najechaniu na przedmiot.

Screeny ekwipunku

|  |
| --- |
|  |
|  |

W grze są trzy rodzaje przedmiotów: miecz, strzała i różdżka. Każdy wrog po smierci ma 7.5% szans na upuszczenie jednego z tych przedmiotów. Każdy przedmiot ma inną wartość obrażeń.

Miecz zadaje obrażenia każdemu wrogowi którego trafi, działa na krótkim zasięgu. Jego obrażenia zależą od siły

|  |
| --- |
|  |
|  |

Strzały zadają obrażenia pierwszemu trafionemu wrogowi, ich obrażenia zależą od wartości zręczności naszego bohatera.

|  |
| --- |
|  |
|  |

Różdżka zadaje obrażenia na dystansie pierwszemu wrogowi którego trafi, obrażenia zależą od wartości inteligencji głównej postaci. Każdy atak kosztuje punkty many, więc może dojść do sytuacji gdy nasz bohater nie będzie mógł zaatakować, dlatego ważne jest posiadanie dodatkowej broni w ekwipunku.

|  |
| --- |
|  |
|  |

## Testowanie

## Walidacja

## Weryfikacja założeń użytkowych aplikacji

## Podsumowanie i wnioski

## Możliwość dalszego rozwoju

# Generowanie proceduralne grach

## Geneza

## Najczęściej używane algorytmy

## Przykładowe zastosowania

# Generowanie map w grze The Mighty Marian

W naszym projekcie każda rozgrywka polega na przejściu kilku, dziesięciu do piętnastu, poziomów jaskiń. Każda z jaskiń generowana jest niezależnie, przy użyciu algorytmu stworzonego na potrzeby projektu. Generowany obszar, pomimo tego, że w grze wizualnie reprezentowany jest w trzech wymiarach, na etapie generacji traktujemy, jako dwuwymiarowy. Mapa składa się z płytek, zwanych także komórkami, które mogą przyjmować dwie wartości, są podłogą lub nicością. Bohater i wrogowie mogą przebywać i poruszać się jedynie po komórkach podłogi.

Komórki podłogi na rysunkach reprezentowane będą przez jaśniejsze pola, natomiast komórki, po których postacie nie mogą się poruszać, kolorem ciemnym.

## Pożądany efekt, poprawność mapy

Celem algorytmu jest wygenerowanie mapy, która będąc wystarczająco skomplikowaną, aby gracz mógł się w niej zgubić i jednocześnie spójną, tak, aby wszystkie komórki podłogi, z których składa się mapa były osiągalne przez gracza. Drugi warunek jest konieczny do spełnienia, ponieważ gdyby postać Mariana i drabina umożliwiająca przejście między poziomami gry zostały umieszczone w innych „składowych spójności” mapy, to skończenie gry byłoby niemożliwe.

To, czy mapa jest spójna (nie istnieją w niej komórki podłogi, do których nie da się dotrzeć z każdej innej komórki podłogi) da się stosunkowo prosto sprawdzić algorytmicznie, to określenie stopnia skomplikowania i zawiłości korytarzy jest już zadaniem, którego w ramach tego projektu nie podjęliśmy się zrealizować. Zdecydowaliśmy się określać w początkowym etapie skomplikowane korytarze i traktować je, jako formę, na której działają kolejne kroki algorytmu. Daje nam to gwarancję mapy o pożądanym poziomie skomplikowania i podzielonej na logiczne obszary. Ten proces zostanie dokładniej opisany w dalszej części dokumentu.

## Automat komórkowy - gra w życie

W projekcie The Mighty Marian do generowania map został użyty automat komórkowy. Automat komórkowy o odpowiednich parametrach i kryteriach przeżycia komórek pozwala w niewielu krokach wygenerować ciekawe struktury podobne do jaskiń. Niestety to rozwiązanie ma również swoje ograniczenia.

Automat komórkowy to model matematyczny, w którym komórki znajdują się w jednym z określonych stanów. System składa się z pojedynczych komórek, znajdujących się obok siebie. Każda z komórek może przyjąć jeden ze stanów, przy czym liczba stanów jest skończona. Plansza, na której znajdują się komórki może być w dowolnej skończonej liczbie wymiarów.

Inicjalnie, w czasie t = 0, każda z komórek znajduje się w jednym z możliwych stanów. Ich stan w czasie t = 1, nowa generacja komórek, określony jest pewną funkcją matematyczną, zwykle zależną od stanu jej sąsiadek.

Na potrzeby tej pracy rozpatrywać będziemy automaty komórkowe w dwóch wymiarach, w których komórki mogą przyjąć jeden z dwóch stanów { żywa, martwa}. Oczywiście, automaty komórkowe mają o wiele więcej możliwych zastosowań i są używane chociażby przy symulowaniu ewolucji czy proceduralnym generowaniu tekstur.

|  |  |
| --- | --- |
| a) | b) |
|  |  |
| *Rysunek 2.2: Spójna mapa o niewielkim poziomie skomplikowania. Otwarty pokój, w którym gracz nie będzie miał czego odkrywać* | *Rysunek 2.1: Mapa niespójna, ale ciekawa pod względem grywalności* |

## Etapy procesu generowania mapy

Proces powstawania map można podzielić na pięć etapów. Produkt końcowy każdego z etapów jest danymi wejściowymi dla kolejnego etapu.

### Etap labiryntu

Proces generowania mapy rozpoczyna się od stworzenia labiryntu, który określi czy pomiędzy wybranymi pokojami występuje połączenie. Parametrami istotnymi w tej fazie jest liczba pokojów, jakie chcemy uzyskać. W poniższych przykładach wybrana liczba to 4 w wymiarze X i 4 w wymiarze Y, więc razem 16.

Na tym etapie mapę modelujemy za pomocą grafu prostego ważonego w następujący sposób:

*G(V,E) - graf nieskierowany*

*V* - zbiór wierzchołków - pojedynczy wierzchołek reprezentuje jeden pokój

*|V|= RoomsX \* RoomsY*

E - zbiór krawędzi - krawędź reprezenuje przejście między pokojami

Program generuje początkowy graf przejść między pokojami. Inicjalnie wszystkie możliwe przejścia między sąsiednimi pokojami istnieją.

|  |
| --- |
|  |
| Rysunek 2.3: Początkowy graf połączeń między pokojami |

W tak zamodelowanej przestrzeni wygenerowanie labiryntu łączącego pokoje sprowadza się do znalezienia minimalnego drzewa spinającego w grafie G. Drzewo spinające grafu jest grafem spójnym i acyklicznym, który zawiera wszystkie wierzchołki grafu oraz niektóre z jego krawędzi. Minimalne drzewo spinające jest drzewem spinającym, którego suma wag krawędzi jest najmniejsza ze wszystkich pozostałych drzew rozpinających danego grafu.. W danym grafie może istnieć może istnieć więcej niż jedno drzewo o tych własnościach. Z punktu widzenia grywalności nie ma znaczenia które wybierzemy, zatem wystarczy wskazać jedno z nich, a wagi krawędziom grafu możemy przypisać losowo. Do uzyskania minimalnego drzewa spinającego został użyty został algorytm Prima, wybrany ze względu na łatwość implementacji.

|  |
| --- |
|  |
| Rysunek 2.4: Graf przejść z przyporządkowanymi losowymi wagami na krawędziach |

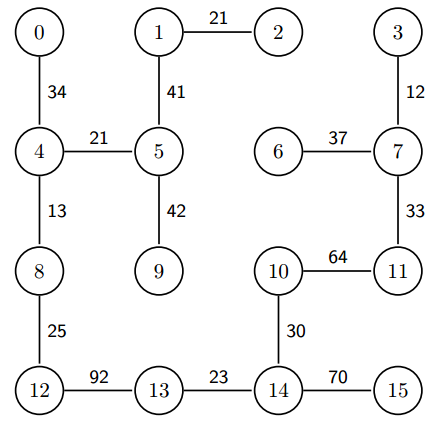
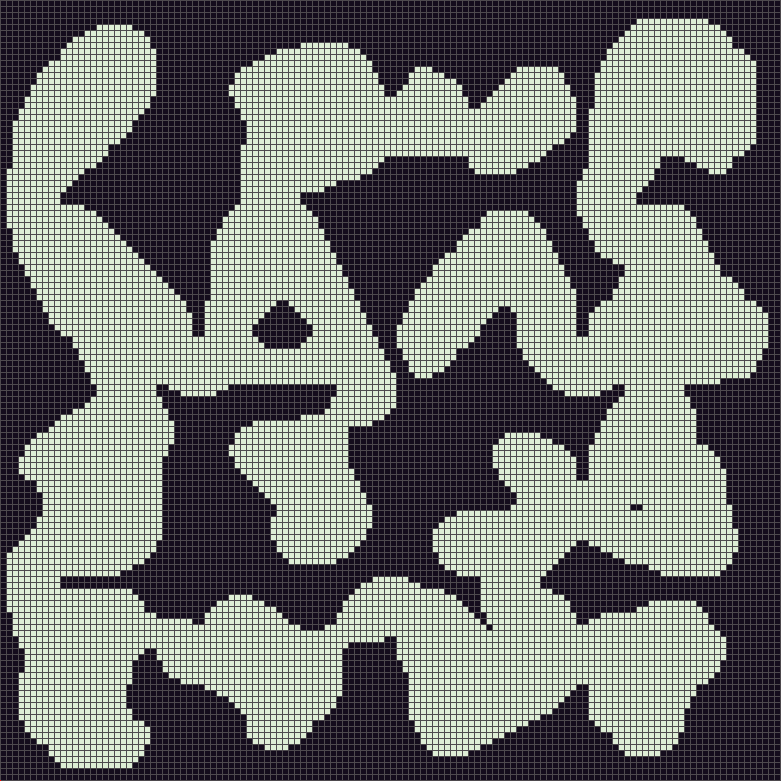
Krawędziom grafu G przyporządkowane zostają losowe wagi z zakresu (3,660). Wierzchołek startowy dla algorytmu również jest wybierany losowo.

Algorytm Prima oparty jest o metodę zachłanną. Można opisać go następująco:

1. Rozpoczynamy od grafu składającego się jedynie z wierzchołka startowego.
2. Krawędzie incydentne do wierzchołka umieszczamy na posortowanej wg. wag liście.
3. Zdejmujemy z listy krawędź o najmniejszej wadze i sprawdzamy, czy łączy wierzchołek wybrany z niewybranym. Jeśli tak, to znalezioną krawędź dodajemy do drzewa spinającego.
4. Dodajemy krawędzie incedentne z nowo wybranym wierzchołkiem do posortowanej listy.
5. Powtarzamy kroki 2 - 4 dopóki lista krawędzi nie będzie pusta.

Rysunek .: Przykładowa mapa możliwa do wygenerowania dla uzyskanego labiryntu

Rysunek .: Labirynt uzyskany po zastosowaniu algorytmu Prima na grafie przejść między pokojami



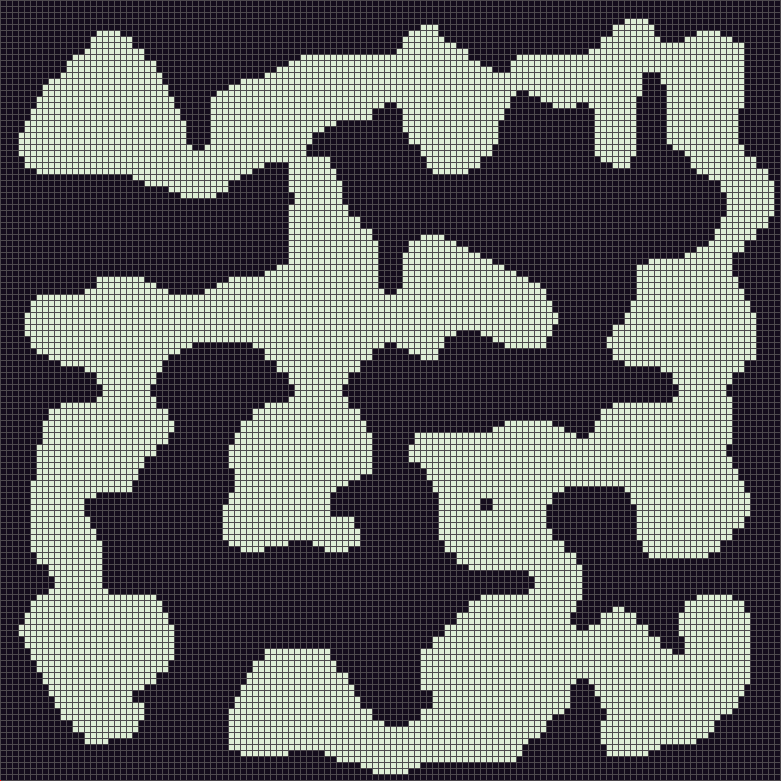
### Położenie przejść między pokojami

W określonym w poprzednim etapie drzewie spinającym wagi krawędzi zastępujemy losowo wartościami ze zbioru {1, 4}. Ta wartość określa, w którym miejscu pomiędzy pokojami utworzone zostanie przejście. Przejście to prostokąt o szerokości *rozmiar pokoju/4* i długości dwóch komórek. Przejścia są obliczane i umieszczane osobno w każdym pokoju. Dzięki wprowadzeniu różnorodności w położeniu przejścia, pomimo tego, że są one wszystkie tych samych rozmiarów, generowane mapy zyskują nieco na poziomie skomplikowania i dzięki temu urozmaicają rozgrywkę. Takie rozwiązanie pomaga ukryć przed graczem fakt, że poziom, po którym się porusza jest zwykłym labiryntem. Na rysunkach 2.7 i 2.8 porównano przykładową mapę z przejściami w tych samych miejscach w pokoju oraz mapę z czterostopniową różnorodnością w położeniu przejścia.

Już cztery stopnie różnorodności przejść tworzą wrażenie różnorodności i pomagają ukryć przed graczem to, że porusza się po prostu po labiryncie. Komórki oznaczone, jako należące do przejścia są zapisywane i wykorzystywane w kolejnym etapie.

Rysunek .: Mapa, gdzie przejścia między pokojami występują losowo na szerokościach oznaczanych przez wartości {1,4}

Rysunek .: Przykładowa mapa, w której przejścia między pokojami wygenerowano na szerokości oznaczonej przez wartość 4



### Etap pokoju

Etap pokoju jest decydujący dla kluczowy dla definicji ostatecznego kształtu korytarzy na mapie. Każdy z pokojów generowany jest osobno, przy pomocy automatu komórkowego. Po każdej iteracji algorytmu komórki należące do przejścia, określonego w poprzedniej fazie stają się podłogą. Na początku plansza pokoju wypełniania jest komórkami nicości. Komórki nieleżące na krawędzi planszy pokoju z prawdopodobieństwem 0,51 zamieniane są w komórki podłogi. Prawdopodobieństwo początkowe zostało wybrane eksperymentalnie, taka wartość daje najciekawsze rezultaty.

Automat komórkowy działa przez siedem iteracji. Żywa komórka, na rysunkach oznaczana kolorem czarnym, to nicość. Komórki jasne oznaczają podłogę. Pierwsze cztery to iteracje tworzące automatu o dość ciekawych regułach przejść między stanami. Jeżeli w sąsiedzctwie Moora o promieniu 2 liczba sąsiadów wynosi mniej niż 3 to komórka jest żywa. Jest również żywa, jeśli w sąsiedztwie Moora o promieniu 1 liczba sąsiadów wynosi 5678. W przeciwnym razie komórka zostaje podłogą. Po czterech iteracjach parametry automatu zmieniają się na 5678/5678 z sąsiedztwem Moora o promieniu 1. Zatem pierwsze cztery iteracje sprzyjają powstawaniu nowych komórek w miejscach opustoszałych i zapobiegają powstawaniu map nieciekawych i pustych. Pozostałe iterację pełnią rolę wygładzającą.

Po tym następuje sprawdzenie, czy uzyskany układ komórek jest spójny, to znaczy czy wszystkie komórki podłogi są połączone. Sprawdzane jest również, czy komórki podłogi stanowią co najmniej 30% powierzchni pokoju. Jeżeli dane ułożenie komórek nie spełnia któregoś z tych warunków rozwiązanie jest odrzucane i proces generowania rozpoczyna się od początku, z prawdopodobieństwem początkowym wystąpienia podłogi większym o 0,1. Proces powtarza się, aż nie powstanie plan pokoju spełniający wspomniane warunki. Dzięki zwiększanej początkowej ilości podłóg prawdopodobieństwo tego, że gracz będzie długo czekał na wygenerowanie mapy zmniejsza się. Takie zwiększanie prawdopodobieństwa umożliwia również generowanie nie tylko map, gdzie kształt pokoju jest od razu widoczny i obrysowany prostokątem, ale również pokojów o kształcie wąskich korytarzy, sterując wspomnianym prawdopodobieństwem początkowym, bez obawy o zbyt długi czas oczekiwania na rozwiązanie.

### Etap łączenia i wygładzania

### Etap erozji

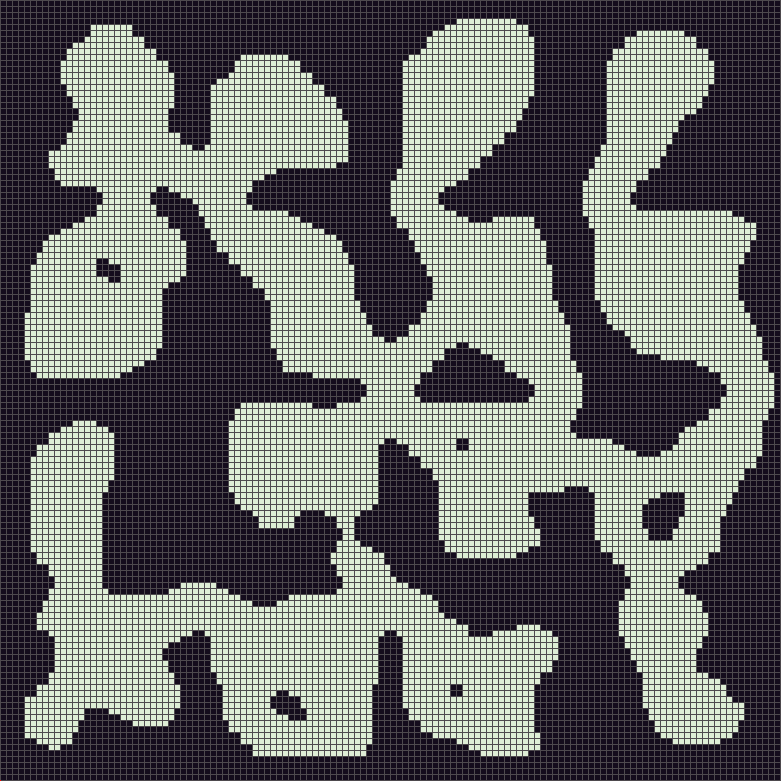
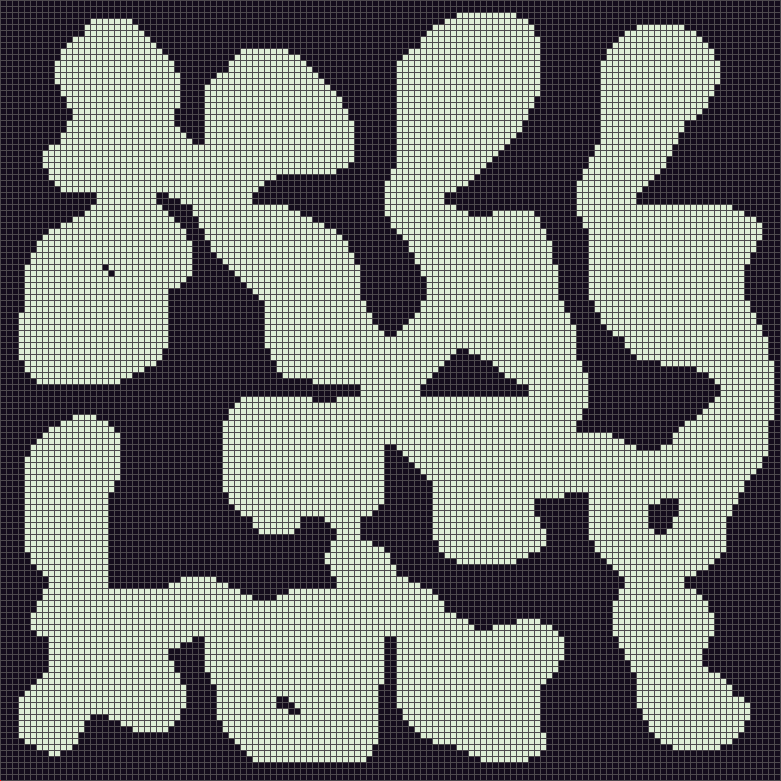
Ponieważ szerokość bohatera jest większa niż szerokość jednej komórki i koliduje on ze ścianami kołem o promieniu 1,5 komórki spójność mapy nie gwarantuje, że będzie ona grywalna. Wszystkie przejścia muszą być, co najmniej o szerokości trzech komórek, aby gracz mógł się przez nie przecisnąć.

Ponieważ nie znaleźliśmy stosunkowo łatwego w implementacji sposobu, aby upewnić się, że wszystkie przesmyki na mapie mają szerokość co najmniej trzech komórek, zdecydowaliśmy się w końcowym etapie poszerzyć wszystkie korytarze prewencyjnie. Ponieważ komórki mapy mogą należeć jedynie do dwóch kategorii, są albo podłogą, albo nicością, to mapę możemy potraktować jak obraz binarny. Poszerzenie korytarzy realizowane jest dzięki cyfrowemu przetwarzaniu obrazów binarnych, przy pomocy filtra erozyjnego.

Niestety, porównując rysunki 2.9 i 2.10 możemy zaobserwować, że zastosowanie filtra erozyjnego ma również swoje wady. Mapa na rysunku 2.10 ma ostrzejsze krawędzie i jest bardziej otwarta. Ponieważ lokalizacja miejsca zwężenia nie została zaimplementowana, filtr nie poszerza jedynie miejsca zwężenia, a ma wpływ na kształt całej mapy.

Rysunek .: Mapa z rysunku 2.9 po zastosowaniu filtru erozyjnego. Problematycznie wąskie przejście zwiększyło swoją szerokość do czterech komórek.

Rysunek .: Mapa przed zastosowaniem filtra erozyjnego. Można zaobserwować przejście o szerokości dwóch komórek, przez które bohater się nie przeciśnie



###### Efekt końcowy

Rozmieszczenie gracza i wrogów

Wyznaczenie początku i końca poziomu

Najkrótsza cieżka przejścia i jej konsekwenjce

Położenie początkowe gracza

Pozycje początkowe wrogów

Wyznaczenie pozycji handlarza

Efekt końcowy

###### **Tabela 1.1.** Wielkość czcionki stosowanej w nagłówkach

|  |  |  |
| --- | --- | --- |
| Poziom nagłówka | Przykład | Wielkość i styl czcionki |
| Nagłówek 1. stopnia | **1. Tytuł rozdziału** | 12 pkt, WERSALIKI, pogrubiona |
| Nagłówek 2. stopnia | ***1.1. Podtytuł rozdziału*** | 10 pkt, pogrubiona i kursywa |
| Nagłówek 3. stopnia | *1.1.1. Punkt podrozdziału* | 10 pkt, kursywa |

Nazwa tabeli jest umieszczona bezpośrednio nad nią, czcionka o wielkości 9 pkt, bez kropki

https://www.assetstore.unity3d.com/en/ asset store

https://www.assetstore.unity3d.com/en/#!/content/10384 inventory

http://brackeys.com/ strona typa który to robi

http://docs.unity3d.com/Manual/ScriptCompileOrderFolders.html odwołanie do skryptu js z c#