

**Paweł Kumorowski**

Nr albumu: 378452

***Projekt i wykonanie gry wideo "Treasure Hunter" w silniku Unity***

Praca magisterska na kierunku Informatyka

Praca wykonana pod kierunkiem

dra Marka Badury

w Katedrze Geometrii

Łódź 2023

Spis treści

[Wstęp 3](#_Toc137652464)

[Cel i założenia 3](#_Toc137652465)

[Terminologia 3](#_Toc137652466)

[Zawartość rozdziałów 3](#_Toc137652467)

[Rozdział I. Wprowadzenie 4](#_Toc137652468)

[Wykorzystane narzędzia 4](#_Toc137652469)

[Szersze omówienie założeń projektu 4](#_Toc137652470)

[Przykłady podobnych gier 5](#_Toc137652471)

[Rozdział II. Dokumentacja programistyczna 6](#_Toc137652472)

[Struktura projektu 6](#_Toc137652473)

[Zawartość folderów 6](#_Toc137652474)

[Spis skryptów 6](#_Toc137652475)

[Omówienie wybranych rozwiązań programistycznych 7](#_Toc137652476)

[Generowanie labiryntu pomieszczeń 7](#_Toc137652477)

[Tworzenie wnętrza pomieszczenia 10](#_Toc137652478)

[Rozdział III. Dokumentacja użytkownika 11](#_Toc137652479)

# Wstęp

## Cel i założenia

Celem pracy było zaprojektowanie oraz wykonanie gry wideo z gatunku roguelite. Głównymi aspektami, na które postawiono nacisk były proceduralne generowanie scenariuszy rozgrywki oraz zróżnicowanie przebiegu kolejnych podejść.

Projekt miał powstać z użyciem darmowych narzędzi oraz assetów dostępnych w Internecie. Miał on zawierać wszystkie funkcjonalności finalnej wersji gry w stopniu co najmniej podstawowym, wraz z gotowym fundamentem pod dalszy rozwój dostępnej zawartości.

W dzisiejszych czasach gatunek gier roguelite jest bardzo popularny. Powstaje wiele tytułów o zróżnicowanej tematyce, stale wnoszących świeże rozwiązania pod kątem rozgrywki. Niezaprzeczalną zaletą tego gatunku jest możliwość odniesienia komercjalnego sukcesu nawet niewielkim studiom. Gry tworzone są często z wykorzystaniem już gotowych silników, np. Unity lub Unreal Engine. Są one bardzo rozbudowane oraz stale rozwijane, zawierają więc najnowsze rozwiązania techniczne oraz w dużym stopniu upraszczają oraz przyspieszają proces twórczy.

## Terminologia

* Roguelite – gatunek gier wideo charakteryzujący się m.in. proceduralnym generowaniem świata oraz wysoką regrywalnością.
* Regrywalność – pojęcie określające jak bardzo atrakcyjne dla gracza jest kolejne przejście gry.
* Assety – gotowe elementy udostępnione do wykorzystania w projekcie.
* Prefaby – przygotowane wcześniej obiekty składające się z wielu komponentów, gotowe do wielokrotnego użycia.

## Zawartość rozdziałów

1. Wprowadzenie – dokładne omówienie celów i założeń pracy, wykorzystanych narzędzi oraz przykłady podobnych gier wideo.
2. Dokumentacja programistyczna – opis skryptów i rozwiązań, które są najistotniejsze z perspektywy tematu pracy, ogólna struktura pracy.
3. Dokumentacja użytkownika – instalacja, uruchomienie, poradnik oraz opis rozgrywki.

# Rozdział I. Wprowadzenie

## Wykorzystane narzędzia

Gra „Treasure Hunter” została wykonana w Unity. Silnik ten jest darmowy do celów niekomercyjnych, posiada obszerną dokumentację oraz bazę poradników tworzonych przez użytkowników. Istotnym jego elementem jest również integracja z Unity Asset Store, czyli bazą assetów gotowych do natychmiastowego wykorzystania.

Skrypty w Unity pisane są w języku C#. Do tego celu wykorzystano środowisko Visual Studio, które zawiera szereg ułatwień w ich tworzeniu, m.in. IntelliSense, czyli pomoc w uzupełnianiu kodu w postaci podpowiedzi, debugowanie, czy też możliwość uruchomienia go z poziomu Unity.

Do przechowywania historii oraz kopii zapasowej skorzystano z systemu kontroli wersji GIT oraz programu GIT Bash. Ułatwiło to również synchronizację pracy na różnych urządzeniach bez konieczności każdorazowego przenoszenia całego projektu.

Niektóre gotowe modele wymagały drobnych poprawek, do tego celu wykorzystano darmowy program Blender.

## Szersze omówienie założeń projektu

Gatunek roguelite zawiera kilka istotnych elementów w kontekście regrywalności. Duża ilość elementów losowych, tj. układ świata, przeciwnicy, przeszkody, czy chociażby niespodziewane wydarzenia, razem tworzą ogromną ilość możliwych kombinacji, które gracz może napotkać w każdej rozgrywce. Innym ważnym aspektem jest stopniowe odblokowywanie bądź ulepszanie ekwipunku, statystyk oraz umiejętności. Pozwala to na docieranie z każdym podejściem coraz dalej oraz daje dodatkowy cel w postaci poczucia rozwoju bohatera. W grze „Treasure Hunter” starano się oddać charakter wyżej wymienionych cech implementując owe systemy realizujące założenia gatunku roguelite.

W projekcie skupiono się w pierwszej kolejności na wpływie losowości na przebieg rozgrywki, zaimplementowano więc kilka systemów na niej opartych:

* Generowanie labiryntu pomieszczeń przy użyciu algorytmu DFS
* Wybór wnętrza dla każdego z pomieszczeń
* Pojawianie się opcjonalnych przeszkód
* Wybór scenariusza rozgrywki dla każdego pomieszczenia
* Przypisanie pomieszczeniom typów przeciwników i szans na ich pojawienie się

Aby rozgrywka była atrakcyjna z perspektywy gracza, musiała zostać dodana odpowiednia ilość zawartości, na którą składają się:

* 6 różnych pomieszczeń
* 5 typów przeciwników oraz łącznie 11 dostępnych wariantów
* 7 rodzajów broni oraz pancerz
* 4 dostępne scenariusze rozgrywki

Projekt posiada również gotową bazę pozwalającą na dodawanie nowej zawartości w prosty sposób.

Rozgrywka polega na ciągłym pokonywaniu hord potworów, z tego powodu również element walki wymagał odpowiedniej uwagi. Aby jej przebieg zależał w większym stopniu od umiejętności gracza, poza standardowymi statystykami decydującymi o przewadze dodano możliwość uniku. Dzierżona w danej chwili broń wpływa na poruszanie się, efektywność uników oraz styl gry. Gra w jasny sposób sygnalizuje odnoszone obrażenia poprzez migotanie postaci a celne trafienia przeciwników odrzucają ich z siłą zależną od siły ataku gracza i ich masy. Przeciwnicy kontrolowani są przez sztuczną inteligencję, do wyznaczania ścieżek skorzystano z systemu NavMesh.

Podczas rozgrywki gracz gromadzi monety, które pozwalają mu na ulepszanie oraz odblokowywanie arsenału broni. Wymagało to stworzenia prostego systemu zapisu pozwalającego na zachowywanie osiągnięć podczas kolejnych sesji gry.

Do stworzenia wszelkich elementów dźwiękowych oraz graficznych, tj. modele, animacje, tekstury, zostały wykorzystane darmowe rozwiązania. Jest to jednak element poboczny w tym projekcie, pozwalający jedynie na zobrazowanie działania wyżej opisanych systemów gry.

## Przykłady podobnych gier

* „Hades” – Supergiant Games
* „Cult of the Lamb” – Massive Monster
* „The Binding of Isaac” – Edmund McMillen

# Rozdział II. Dokumentacja programistyczna

## Struktura projektu

Do stworzenia „Treasure Hunter” została użyta jedynie jedna scena, na której od razu rozpoczyna się rozgrywka. Na początku poprzez skrypt generowane są elementy świata oraz wczytywane dane z zapisu poprzedniej rozgrywki. Każdy koniec gry, niezależnie czy zwycięstwo czy porażka dokonuje zapisu oraz restartuje scenę.

Unity pozwala na przypisywanie wartości zmiennym bezpośrednio w edytorze więc wszystkie niezbędne ustawienia pozwalające na dostosowywanie parametrów rozgrywki zostały tam udostępnione.

Elementy świata wykorzystywane w grze zostały przygotowane w formie prefabów. Część z nich będąca otoczeniem została użyta do utworzenia wnętrz pomieszczeń, natomiast pozostałe kontrolowane są skryptami i należą do nich m.in.: gracz, przeciwnicy, bronie, pociski.

## Zawartość folderów

* Animations – humanoidalne animacje używane przez różne jednostki
* Images – graficzne elementy interfejsu
* Imported – pobrane assety
* Materials – wybrane materiały i tekstury
* Prefabs – przygotowane elementy składające się na świat gry
* Scenes – scena gry
* Scripts – skrypty
* Settings – ustawienia Universal Render Pipeline
* Sounds – dźwięki
* TextMeshPro – importowaną paczkę obsługującą tekst

## Spis skryptów

* BulletController – kontroluje zachowanie pocisków wystrzelonych zarówno przez gracza jak i przeciwników.
* CameraMovement – ustawienie miękkiego śledzenia gracza kamerą.
* ChestController – dotyczy zachowania finalnej skrzyni z nagrodą oraz restartu gry po jej otworzeniu.
* DestroyableScript – pozwala na niszczenie niektórych elementów otoczenia.
* Enemies – pomocniczy skrypt losujący szanse pojawienia się konkretnych przeciwników.
* EnemyController – definiuje zachowanie przeciwników.
* LevelGenerator – przygotowuje świat gry tworząc labirynt pomieszczeń.
* MeleeHitbox – obsługuje kolizje w walce wręcz.
* PlayerCombat – zawiera statystyki gracza oraz zachowania związane z walką.
* PlayerController – odpowiada za obsługę przycisków kontrolujących gracza, poruszanie, uniki, możliwość interakcji z otoczeniem.
* PlayerInventory – system ekwipunku gracza, zawiera ilość posiadanych monet, broń, pancerz.
* PlayerSpawn – tworzy postać gracza w pomieszczeniu startowym.
* RoomChange – pomocniczy skrypt przechowujący informację o obecnym pomieszczeniu.
* RoomController – kontroluje zachowanie pomieszczeń oraz scenariuszy rozgrywki.
* RoomGenerator – umożliwia tworzenie pomieszczeń i ustawianie przejść między nimi.
* Save – zapisuje do oraz odczytuje z pliku zapisu.
* SoundController – przechowuje oraz odtwarza niektóre dźwięki.
* TorchController – obsługuje pochodnie będące jednym ze scenariuszy rozgrywki.
* UIController – obsługuje interfejs.

## Omówienie wybranych rozwiązań programistycznych

### Generowanie labiryntu pomieszczeń

Generowanie labiryntu pomieszczeń odbywa się z wykorzystaniem algorytmu DFS. W pierwszej kolejności tworzona jest siatka, przechowywana w jednowymiarowej liście. Składa się ona z komórek, które mają potencjalną szansę na bycie pomieszczeniem.



Następnie jako pierwszą przeglądaną komórkę ustawiamy wcześniej wybraną, będącą startem naszego labiryntu. Ścieżka będzie przechowywana w strukturze stosu.



Warunkami końca są dotarcie do pomieszczenia wybranego jako finalne oraz odwiedzenie maksymalnej liczby pomieszczeń. Każda przeglądana komórka dodawana jest do ścieżki lecz jeżeli nie ma żadnych wolnych sąsiadów a nie jest jednocześnie końcem labiryntu, nie może być jej częścią i zostaje zdjęta ze stosu. Kolejną rozpatrzaną w pętli komórką jest losowo wybierana z dostępnych sąsiadów.



Funkcja CehckNeighbors zwraca wszystkich sąsiadów danej komórki.



Funkcja AddPath dodaje połączenia między komórkami (pokojami). Sprawia, że odpowiednie drzwi w pokoju zostają otwarte umożliwiając przejście do jego sąsiadów. Skrypt zawiera również typ wyliczenia Direction, który posiada wszystkie cztery możliwe kierunki poruszania się po siatce pomieszczeń.

### Tworzenie wnętrza pomieszczenia

* Generowanie labiryntu pomieszczeń
* Tworzenie wnętrza pomieszczeń
* Losowanie przeciwników
* AI przeciwników
* Zapis

# Rozdział III. Dokumentacja użytkownika