



POLSKO-JAPOŃSKA AKADEMIA  
TECHNIK KOMPUTEROWYCH

## Proces tworzenia gry komputerowej z gatunku survival-horror na przykładzie projektu "Callisto"

Mateusz Jarecki S20854

Jan Rycielski S21651

Michał Konieczny S20856

Rozszerzona rzeczywistość i systemy immersywne

Programowanie gier

**Polsko-Japońska Akademia Technik Komputerowych**

Promotor:

mgr inż. Krzysztof Kowalski

Recenzent:

mgr inż. Rafał Masłyk

Warszawa, 9 września 2024

**Streszczenie** Praca inżynierska koncentruje się na projektowaniu i implementacji gry komputerowej w gatunku survival-horror, z naciskiem na zapewnienie graczom głębskiej immersji i emocjonujących wyzwań. Główne mechaniki przetrwania w grze zostały zaprojektowane tak, aby oferować skomplikowane wyzwania, które zmuszają graczy do myślenia strategicznego i zarządzania zasobami w nieprzewidywalnym środowisku.

Projekt został zrealizowany przy użyciu technologii programistycznych i środowiska 3D stworzonego za pomocą silnika Unity, który umożliwia efektywne tworzenie skomplikowanych scen i obiektów interaktywnych. Dodatkowo, zastosowano program Blender do tworzenia grafik 3D, co pozwoliło na szczegółowe opracowanie postaci i środowisk gry. Program Blender, jako narzędzie do modelowania, animacji i renderowania, umożliwił również dodanie realistycznych tekstur i efektów świetlnych, które są kluczowe dla budowania napiętej i przerażającej atmosfery typowej dla gier z gatunku horror.

Metody projektowe obejmują rozwój systemów zarządzania zasobami, które wymagają od graczy inteligentnego gospodarowania ograniczonymi surowcami, interakcje z otoczeniem, które mogą prowadzić do odkrywania nowych obszarów, które adaptują się do działań i strategii gracza, zwiększając poziom trudności i angażując w rozgrywkę. Projekt stanowi połączenie nowoczesnych technologii i kreatywnego projektowania, które razem tworzą unikatowe i pamiętne doświadczenie gry survival-horror.

**Keywords:** Proces tworzenia gier · Grafika 3D · Trzecio-osobowa gra survivalowa · Unity · Blender

**Abstract.** The engineering thesis focuses on the design and implementation of a computer game in the survival-horror genre, emphasizing deep immersion and thrilling challenges for players. The main survival mechanics in the game are designed to offer complex challenges that force players to think strategically and manage resources in an unpredictable environment.

The project was executed using programming technologies and a 3D environment created with the Unity engine, which enables the efficient creation of complex scenes and interactive objects. Additionally, Blender was used for creating 3D graphics, allowing for detailed development of game characters and environments. Blender, as a tool for modeling, animation, and rendering, also enabled the addition of realistic textures and lighting effects, which are crucial for building the tense and terrifying atmosphere typical of horror games.

The design methods include the development of resource management systems that require players to intelligently manage limited resources, interactions with the environment that can lead to the discovery of new areas, and environments that adapt to player actions and strategies, increasing the level of difficulty and engaging gameplay. The project combines modern technologies and creative design, together creating a unique and memorable survival-horror game experience.

**Keywords:** Process of creating games · 3D Graphics · Third person survival game.

# Table of Contents

1	Wstęp . . . . .	6
1.1	Cel pracy . . . . .	7
1.2	Motywacja . . . . .	7
1.2.1	Stworzenie własnej gry . . . . .	8
2	Game design . . . . .	8
2.1	Elementy rozgrywki . . . . .	8
2.1.1	Cel gry . . . . .	9
2.1.2	Zmiany pogodowe . . . . .	9
2.1.3	Opcje . . . . .	9
2.1.4	Proces tworzenia przedmiotów . . . . .	10
2.1.5	Rozwijanie bazy . . . . .	11
2.1.6	Przetapianie zasobów . . . . .	11
2.1.7	Zbieranie zasobów . . . . .	11
2.2	Kamera . . . . .	12
2.3	Ruch gracza . . . . .	13
2.4	Grafika 3D . . . . .	14
2.5	Oprawa Muzyczna . . . . .	20
3	Powiązane prace/gry/gatunki . . . . .	20
3.1	Cryofall . . . . .	20
3.2	Darkwood . . . . .	22
4	Narzędzia i metody . . . . .	23
4.1	Program Unity . . . . .	23
4.2	Program Blender . . . . .	25
4.3	Trello . . . . .	26
4.4	Scrum . . . . .	27
4.5	Github i Git . . . . .	28
5	Rezultaty . . . . .	30
5.1	Trzecioosobowa, izometryczna kamera . . . . .	30
5.2	Ruch gracza . . . . .	30
5.3	Interakcją gracza z otoczeniem . . . . .	31
5.4	Wydobywanie zasobów . . . . .	33
5.5	Wytwarzanie zasobów . . . . .	35
5.6	Wytapianie zasobów . . . . .	37
5.7	Zapis gry . . . . .	38

5.8 Ustawienia .....	39
5.9 Deszcz .....	41
5.10 Testy .....	43
6 Dyskusja .....	44
7 Wnioski i przyszłe prace .....	46
8 Bibliografia .....	47

## 1 Wstęp

W ramach pracy inżynierskiej skupiamy się na tworzeniu gry komputerowej w gatunku survival, której celem jest dostarczenie doświadczeń związanych z przygodą, przetrwaniem i taktycznym rozwiązywaniem problemów. Gra przenosi graczy do dynamicznie zmieniającego się świata, gdzie każda decyzja może wpływać na dalsze losy postaci.

Projekt wykorzystuje nowoczesne technologie programistyczne oraz zaawansowane możliwości silnika Unity, co pozwala na kreowanie złożonego środowiska trójwymiarowego. Dzięki temu możliwe jest stworzenie gry, która jest bogata w różnorodne mechaniki gry. Znaczącą rolę odgrywa tutaj projektowanie skomplikowanych systemów przetrwania, które wymagają od graczy zarówno zręczności, jak i strategicznego planowania.

Głównym założeniem jest zbudowanie świata, który stale stawia przed graczami nowe wyzwania. To wymusza na graczu stałą adaptację i podejmowanie szybkich decyzji. Oprawa graficzna oraz dźwiękowa mają za zadanie nie tylko zachwycić użytkownika, ale przede wszystkim wzmacnić odczucie realności i bezpośredniego zagrożenia, które są kluczowe w grach tego gatunku.

Projektowano przez gra jest próbą połączenia technologii z kreatywnym projektowaniem rozgrywki, mając na celu stworzenie doświadczenia, które jest zarówno wyzwaniem, jak i emocjonującą przygodą.

## 1.1 Cel pracy

Celem pracy inżynierskiej jest projektowanie i implementacja gry komputerowej w gatunku survival, która korzysta z nowoczesnych metod i technologii programistycznych oraz potencjału silnika Unity do stworzenia złożonego środowiska 3D. Centralnym punktem pracy jest nie tylko stworzenie gry, ale również analiza procesu jej tworzenia, która ma na celu zrozumienie i demonstrację wyzwań technicznych związanych z produkcją gier, oraz szanse eksploracji nowych możliwości technologicznych i narracyjnych.

### 1. Zaawansowane mechaniki gry

Celem jest stworzenie zaawansowanego systemu mechanik gry, które wykraczają poza standardowe rozwiązań znane z gatunku survival. Naszym zamiarem jest integracja skomplikowanych elementów rozgrywki, które zmuszają graczy do głębokiego myślenia analitycznego i strategicznego. Chcemy, aby gracze musieli efektywnie zarządzać zasobami, podejmować szybkie decyzje pod presją. Kluczowym aspektem będzie również rozwijanie systemów interakcji z przeciwnikami oraz pułapek, które będą wymagały od graczy nie tylko zręczności, ale przede wszystkim przemyślanego planowania i taktycznego podejścia.

### 2. Oprawa wizualna

Projekt zakłada zastosowanie najnowszych technik graficznych. Prace skupią się na optymalizacjach w dziedzinie oświetlenia, tekstur i modelowania, co ma na celu nie tylko wzmacnianie imersji, ale również zintensyfikowanie emocjonalnego wpływu gry na użytkownika. Dążymy do tego, aby każdy element wizualny, od mrocznych, klaustrofobicznych korytarzy po subtelne efekty świetlne, przyczyniał się do budowania napięcia i angażowania gracza w przeżywanie historii.

## 1.2 Motywacja

Celem projektu jest stworzenia nowej gry komputerowej, która ma na celu wypełnienie luki na rynku. Obecnie na rynku gier brakuje zaawansowanych technicznie tytułów, które jednocześnie oferują jakościową oprawę wizualną oraz mechaniki gry wymagające od graczy

skupienia, strategicznego myślenia i przemyślanego planowania. Nasze założenia projektowe wynikają z dokładnej analizy aktualnych trendów oraz potrzeb graczy, którzy poszukują bardziej angażujących i stylujących doświadczeń.

Projekt ten ma na celu nie tylko dostarczenie rozrywki, ale również zaoferować wyzwania. Poprzez nasz produkt chcemy zapewnić użytkownikom możliwość rozwijania umiejętności takich jak krytyczne myślenie, strategiczne planowanie oraz zdolność adaptacji do dynamicznie zmieniających się warunków wirtualnego świata. Podejście do projektowania gry kładzie nacisk na zaawansowane mechaniki

Projekt ten jest odpowiedzią na identyfikację specyficznej potrzeby rynkowej, oferując produkt, który jest nie tylko technologicznie zaawansowany, ale także bogaty w treści. Zmusza to graczy do myślenia i planowania kolejnych ruchów. Ostatecznym celem jest wprowadzenie na rynek gry, która będzie pionierem w swojej kategorii, inspirując tym samym inne podmioty do tworzenia gier, które są jednocześnie wyzwaniem i narzędziem edukacyjnym.

### **1.2.1 Stworzenie własnej gry**

Motywacją zespołu było stworzenie własnej produkcji oraz podjęcie wyzwania, które stawia taki projekt. Chcieliśmy sprawdzić swoje umiejętności, kreatywność i zdolność do pracy zespołowej. Zespół chciał zmierzyć się z różnymi aspektami tworzenia gry, od pisania scenariusza, przez wymyślanie treści, aż po projektowanie mechanik.

## **2 Game design**

### **2.1 Elementy rozgrywki**

Gra opiera się na rzemieślniczym tworzeniu przedmiotów oraz na zbieraniu potrzebnych surowców jako podstawy ich wytworzenia. Gracze są zachęcani do badania świata gry, w którym mogą znaleźć rozmaite materiały pomocne w dalszej produkcji narzędzi, broni czy innych użytecznych przedmiotów. Te działania są konieczne, aby przetrwać w trudnych warunkach gry. Rozbudowany system craftingu pozwala na szeroką gamę działań związanych z przetwarzaniem surowców.

Jednym z kluczowych elementów rozgrywki jest stworzony przez nas model craftingu, który umożliwia graczom tworzenie przedmiotów z zebranych surowców. W systemie tym gracze mają do dyspozycji piec do przetapiania, który służy do obróbki metali i innych materiałów wymagających wysokiej temperatury.

Gra zawiera możliwość interakcji z otoczeniem poprzez ścinanie drzew, kopanie węgla, rudy metali oraz zbieranie elementów z ziemi. Gracze mogą także pozyskiwać inne surowce naturalne, które są niezbędne do tworzenia bardziej zaawansowanych przedmiotów i narzędzi. W ten sposób, eksplorując świat gry i korzystając z rozbudowanego systemu craftingu, gracze są w stanie przetrwać i rozwijać się w trudnych warunkach, jakie oferuje gra.

### 2.1.1 Cel gry

W celu osiągnięcia końcowego zadania, jakim jest umieszczenie złotej figurki w wyznaczonym miejscu na mapie. Konieczne jest przebycie długiej drogi polegającej na zbieraniu surowców, z których można tworzyć nowe przedmioty. Na starcie gra wymaga zebrania podstawowych surowców takich jak kamienie i patyki, które są niezbędne do dalszego rozwoju. W późniejszych etapach gry, gracz jest w stanie stworzyć, piec, który umożliwia wytapianie rud żelaza. Proces ten umożliwia dalsze eksplotowanie zasobów naturalnych, w tym wydobycie rud złota. Ostatnim etapem jest przetopienie zebranych rud złota na sztabki, które są niezbędne do realizacji głównego zadania gry.

### 2.1.2 Zmiany pogodowe

Naszym planem było dodanie zmian pogodowych tak aby w czasie gry gracz mógł zaobserwować sytuację w której zaczyna padać deszcz. Poza efektem wizualnym nie ma on wpływu na mechaniki gry, ma jednak znaczenie wizualne oraz wpływa na odczucia gracza z zmieniającego się otoczenia.

### 2.1.3 Opcje

Celem było umożliwienie graczowi zmiany opcji ustawień graficznych takich jak:

- FullScreen
- Resolution

- Graphics (wcześniej predefiniowane ustawienia)
- Multisampling Anti-Aliasing Mode
- Texture Details
- Texture Filtering Mode
- VSync Count

Chcieliśmy również żeby gracz miał możliwość zmiany ustawień dźwięku niezależnie od siebie. W tym celu naszym planem było dodanie

- Master
- Music
- SFX

Dodatkowo naszym celem było umożliwienie graczowi automatyczne zapisywanie wybranych opcji ustawień do plików wewnętrznych gry, tak aby po ponownym włączeniu gry nie trzeba było ich ponownie zmieniać.

#### **2.1.4 Proces tworzenia przedmiotów**

W grze dodano system craftingu, który umożliwia graczom tworzenie nowych prostych przedmiotów, jak i zaawansowanych zestawów w ekwipunku niezbędnych dla rozgrywki. Dzięki temu systemowi gracz może wykorzystać zebrane materiały, aby dostosowywać swoje wyposażenie do napotkanych potrzeb i wyzwań w świecie gry.

Proces tworzenia przedmiotów składa się z kilku kluczowych etapów. Gracz musi zbierać różnorodne surowce i komponenty, które można znaleźć eksplorując świat gry. Surowce pochodzą z zasobów naturalnych. Każdy przedmiot w grze wymaga określonych składników, które są opisane w recepturach przedmiotów. Gracz może przeglądać dostępne receptury, aby zobaczyć, jakie materiały są potrzebne dotworzenia danego przedmiotu. Po zebraniu niezbędnych surowców, gracz musi sprawdzić swój ekwipunek oraz ma możliwość sprawdzić najbliższe skrzynie, które znajdują się w promieniu 3 jednostek od gracza. Jeżeli wszystkie potrzebne składniki są dostępne, gracz może przejść do kolejnego etapu. Gracz wykorzystuje zebrane surowce do stworzenia wybranego przedmiotu. Proces ten automatycznie używa wymagane składniki z ekwipunku oraz pobliskich skrzyń. Po stworzeniu nowego przedmiotu, najpierw jest sprawdzane, czy w ekwipunku gracza jest wolne miejsce na ten przedmiot. Jeśli tak, przedmiot

zostaje dodany bezpośrednio do ekwipunku. Jeśli jednak ekwipunek jest pełny, wtedy sprawdzana jest najbliższą skrzynią, która ma najmniej przedmiotów. Nowo wytworzony przedmiot zostaje umieszczony w tej skrzyni, zapewniając optymalne zarządzanie zasobami i miejscem.

### 2.1.5 Rozwijanie bazy

#### 2.1.6 Przetapianie zasobów

W grze został zaimplementowany piec, który daje możliwość przetapiania elementów wykopanych przez gracza na sztabki metali. Piec działał poprawnie i można było uzyskać pełną ilość wytwarzanego surowca, konieczne jest utrzymanie odpowiedniej temperatury. Można to osiągnąć poprzez dodanie odpowiedniej ilości paliwa, takiego jak patyki, węgiel. Temperatura w piecu jest kluczowa dla efektywności przetapiania surowców. Ilość dostępnego paliwa bezpośrednio wpływa na ilość uzyskanego produktu. Gdy dostępne jest tylko 50% potrzebnego paliwa, można otrzymać około 40-50% surowca. Jeśli paliwa jest około 70%, uzyskana ilość produktu wynosi od 50 do 70%. W przypadku, gdy paliwo stanowi pełne 100% wymaganej ilości, piec przetapia surowce na 100% produktu. Działanie pieca opiera się na dodaniu odpowiedniej ilości dodanego paliwa i dostosowywaniu efektywności przetapiania surowców. Proces ten motywuje graczy do zbierania i dostarczania odpowiedniej ilości niezbednego paliwa, aby osiągnąć maksymalną efektywność. W zależności od ilości dostępnego paliwa, piec dostosowuje ilość wytwarzanego produktu. Wpływa to na strategię gracza w zarządzaniu zasobami. Gracz musi dokładnie planować i zarządzać swoimi zasobami, co dodaje realizmu do gry. Konieczność utrzymywania odpowiedniej temperatury sprawia, że proces przetapiania staje się bardziej angażujący i skaplikowany, co zwiększa satysfakcję z osiągnięcia sukcesu w tworzeniu niezbędnych surowców.

#### 2.1.7 Zbieranie zasobów

W grze istnieją dwie możliwości zbierania przedmiotów. Pierwszy z nich polega na podnoszeniu przedmiotów, które leżą na ziemi. Gracz musi podejść do tych przedmiotów, takich jak małe kamienie lub gałęzie, i je podnieść. Jest to bezpośrednia i szybka metoda

zdobywania podstawowych surowców. Drugi sposób zbierania przedmiotów polega na niszczeniu większych obiektów na mapie, takich jak drzewa lub kamienie, aby pozyskać dane surowce. Proces jest bardziej złożony i wymaga użycia odpowiednich narzędzi, na przykład kilofa do niszczenia kamieni lub siekiery do ścinania drzewa. Każdy większy obiekt przeznaczony do niszczenia ma nad sobą pasek wytrzymałości, który wskazuje ile czasu trzeba uderzać dany przedmiot aby go zniszczyć. Pasek wytrzymałości kamienia lub drzewa jest widoczny dla gracza. Proces ten jest stopniowy, co oznacza, że gracz musi wielokrotnie uderzać obiekt. Im gracz jest bliżej zniszczenia obiektu, wtedy pojawiają się efekty wizualne. Sygnalizuje to postęp prac. Podnoszenie przedmiotów działa, dzięki użyciu korutyn. Kiedy gracz podejdzie do przedmiotu i zdeży się z nim, wtedy przedmiot znika z sceny gry i pojawia się w ekwipunku gracza. Mechanika niszczenia obiektów wprowadza element strategicznego zarządzania zasobami i narzędziami, wymagając od gracza podejmowania decyzji dotyczących optymalnego wykorzystania dostępnych narzędzi. Proces ten jest bardziej angażujący, trudniejszy i bardziej czasochłonny w porównaniu do prostego podnoszenia przedmiotów. Pozwala na zdobycie większych ilości surowców i bardziej wartościowych materiałów.

## 2.2 Kamera

Kamera została specjalnie przygotowana tak, aby rzut na świat był izometryczny. Rzut izometryczny to rodzaj rysunku technicznego, który pozwala na przedstawienie trójwymiarowego obiektu na płaszczyźnie dwuwymiarowej w taki sposób, że trzy osie przestrzenne są równo oddalone od siebie o 120 stopni. W rzucie izometrycznym wymiary wzdłuż każdej z trzech osi są skalowane jednakowo, co sprawia, że proporcje i kształty obiektów są zachowane bez zniekształceń perspektywicznych. Dzięki temu można dokładnie i czytelnie zobrazować przestrzenne relacje między różnymi elementami gry.

Kamera w naszej grze podąża za graczem, jednak obrót gracza wokół własnej osi nie wpływa na pozycję kamery. Jej odpowiednie umiejscowienie na scenie sprawia również że nie ingeruje ona z innymi obiekttami.

### 2.3 Ruch gracza

Celem było zaprojektowanie poruszania się postaci w taki sposób aby gracz nie odczuwał problemów z domyśleniem się w którym kierunku będzie poruszała się postać kiedy gracz przyciśnie przycisk odpowiedzialny za poruszanie się postaci w oczekiwana stronę. Chcieliśmy aby poruszanie się nie było zależne od obrotu postaci ani miejsca w którym znajduje się kursor gracza, co mogło by spowodować że postać zmieniała by kierunek po jednej klatce animacji wbrew oczekiwaniu gracza.

## 2.4 Grafika 3D

W naszej grze postanowiliśmy część modeli stworzyć przy użyciu programu Blender, zamiast korzystać z gotowych modeli, które można znaleźć online. To podejście daje nam pełną swobodę w tworzeniu zawartości gry, ale również pozwala nam wytwarzanie modele konkretne pod nasze potrzeby, co jest kluczowe dla utrzymania spójnej estetyki i charakteru gry.

Blender to narzędzie do modelowania 3D, które oferuje szeroki wachlarz funkcji umożliwiających tworzenie skomplikowanych i szczegółowych modeli. Jego otwarty charakter oraz społeczność sprawiają, że jest to idealne narzędzie dla twórców gier.

Nasze modele stworzyliśmy w stylu "low-poly", co pozwoliło na szybkie uzyskanie upragnionych efektów a jednocześnie zapewniło, że gra będzie działać płynnie. Styl low-poly, charakteryzujący się ograniczoną liczbą wielokątów, nadaje również grze unikalny, minimalistyczny wygląd, który jest obecnie bardzo popularny w branży gier.

Proces tworzenia modeli w Blenderze obejmował kilka etapów. Na początku tworzyliśmy podstawowe kształty, które następnie były modyfikowane i optymalizowane. Dzięki zastosowaniu tekstur oraz prostych materiałów, byliśmy w stanie uzyskać zamierzone wcześniej modele.

Niektóre materiały zostały również tworzone przy użyciu systemu materiałów oferowanym w programie Blender. System materiałów "Material Nodes" to narzędzie umożliwiające tworzenie zaawansowanych shaderów i tekstur poprzez łączenie ze sobą różnych funkcji i parametrów. Dzięki temu mogliśmy tworzyć materiały, które pasują do naszych modeli i stylu gry. Używanie "nodes" pozwoliło na większą kontrolę nad wyglądem końcowym materiałów, a także umożliwiło łatwe eksperymenty z różnymi efektami wizualnymi.

Warto również wspomnieć, że używanie Blenera pozwoliło nam na szybkie poprawki. Każdy model mógł być łatwo dostosowywany w zależności od feedbacku otrzymanego od zespołu testującego, co znacznie przyspieszyło proces ich tworzenia.

Dwa modele użyte w grze zostały pobrane ze strony Sketchfab i użyte na podstawie licencji CC Attribution. Są to dwa ostatnie modele, model pieca oraz model skrzynki.

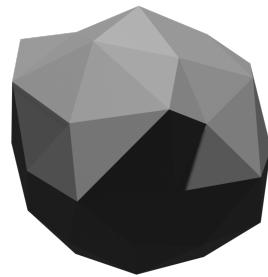


Fig. 1: Model kamienia Źródło: Opracowanie własne



Fig. 2: Model złota Źródło: Opracowanie własne

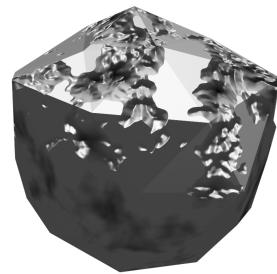


Fig. 3: Model żelaza Źródło: Opracowanie własne



Fig. 4: Modele patyków Źródło: Opracowanie własne



Fig. 5: Model drzewa (twisted) Źródło: Opracowanie własne



Fig. 6: Model drzewa (liscie) Źródło: Opracowanie własne



Fig. 7: Modele kilofów Źródło: Opracowanie własne



Fig. 8: Modele siekier Źródło: Opracowanie własne



Fig. 9: Model bazy Źródło: Opracowanie własne



Fig. 10: Model pieca Źródło: <https://sketchfab.com/3d-models/stylized-furnace-e7e9c0fa89724f8dba106f96bb4a9c8b>



Fig. 11: Model skrzyni Źródło: <https://sketchfab.com/3d-models/low-poly-chest-ede8d988e3724ed395bc20419040d2da>

## 2.5 Oprawa Muzyczna

W naszym projekcie starannie dobraliśmy muzykę oraz efekty dźwiękowe. Wybraliśmy różnorodne odgłosy lasu i otaczającej rzeczywistości, takie jak wiatr, trzaski gałęzi czy głosy dzikich zwierząt. Muzyka i dźwięki w grze mają nie tylko maja dostarczać wrażeń estetycznych, ale przede wszystkim są zaprojektowane tak, by wspierać napięcie i koncentrację gracza. Dzięki temu, gra zyskuje na realizmie, a rozgrywka staje się bardziej wciągająca i emocjonująca.

## 3 Powiązane prace/gry/gatunki

### 3.1 Cryofall

CryoFall to gra komputerowa, która łączy elementy survivalu z rozgrywką wieloosobową (MMO). Gra została opracowana przez studio AtomicTorch Studio i wydana przez Daedalic Entertainment. Premiera gry miała miejsce w kwietniu 2019 roku.

W CryoFall gracze znajdują się na odległej, obcej planecie po katastrofie swojego statku kosmicznego. Zadaniem graczy jest przetrwanie w surowym, nieznajomym środowisku, budowanie schronień,

zdobywanie zasobów, tworzenie narzędzi i broni, a także interakcja z innymi graczami. Rozgrywka skupia się na stopniowym postępie technologicznym, od prostych narzędzi po zaawansowaną technologię.

CryoFall wyróżnia się na tle innych gier survivalowych dzięki swojej rozbudowanej gospodarce i aspektom społecznym. Gra oferuje zaawansowany system craftingu, który pozwala na tworzenie coraz to nowszych i bardziej skomplikowanych przedmiotów i konstrukcji. Ważnym elementem gry jest też interakcja między graczami, która może przybierać formę handlu, współpracy lub konfliktu. Gra zachęca również do tworzenia własnych społeczności i osad, co dodaje dodatkową głębię społeczną i strategiczną.

Dodatkowo, CryoFall posiada atrakcyjną, choć prostą grafikę 2D, która jest dostosowana do dużej skali świata gry. Mechanizmy gospodarcze, takie jak handel i zarządzanie zasobami, w połączeniu z możliwością odkrywania nowych technologii, sprawiają, że gra oferuje bogate i złożone doświadczenie zarówno dla pojedynczego gracza, jak i dla większych grup.

CryoFall, dzięki swojemu połączeniu survivalu z elementami ekonomicznymi i społecznymi, stanowi unikalne podejście do gatunku MMO, co pozwoliło jej na wyróżnienie się na rynku gier. Gra nadal jest rozwijana i wspierana przez twórców, co przyciąga nowych graczy oraz utrzymuje zainteresowanie wśród dotychczasowej społeczności.



Fig. 12: Przedstawienie rozgrywki w CryoFall  
 Źródło:<https://muve.pl/sklep/digital/gry-niezalezne/cryofall,1534471>

### 3.2 Darkwood

Darkwood to niezależna gra komputerowa z gatunku horroru survivalowego, która zyskała uznanie dzięki swojej unikalnej atmosferze i innowacyjnym podejściu do rozgrywki. Stworzona przez polskie studio Acid Wizard Studio, gra miała swoją premierę w sierpniu 2017.

Fabuła gry osadzona jest w zdeformowanym, tajemniczym lesie, z którego nie ma ucieczki. Gracze przejmują kontrolę nad różnymi postaciami, które starają się przetrwać w tym przerząjącym środowisku, pełnym zagadek i niebezpieczeństw. Dni w Darkwood są przeznaczone na eksplorację i zbieranie zasobów, niezbędnych do przetrwania nocnych ataków przerząjących stworzeń i innych niepokojących zjawisk. W nocy, gracze muszą bronić swojego schronienia, co dodaje elementy strategii i planowania do rozgrywki.

Co wyróżnia Darkwood na tle innych gier z gatunku, to przede wszystkim atmosfera napięcia i grozy, potęgowana przez szczegółowe, ręcznie rysowane grafiki oraz efektowną ścieżkę dźwiękową. Gra korzysta z perspektywy z góry, co jest rzadkością w grach horrorowych, ale tutaj zastosowanie tej techniki tylko zwiększa uczucie niepokoju i niepewności.

Darkwood również wyróżnia się głębokim systemem decyzji i konsekwencji. Wybory dokonywane przez gracza mają realny wpływ na rozwój fabuły i dostępne zakończenia, co dodaje dodatkowej warstwy immersji. Wprowadzenie elementów roguelike, takich jak generowane procedurałem poziomy i stałe śmierci, zwiększa replayability, jednocześnie podnosząc poziom wyzwania.

Gra oferuje dużą swobodę w eksploracji i podejmowaniu decyzji, co pozwala każdemu graczu na kształtowanie swojego własnego, unikalnego doświadczenia. Dzięki temu oraz połączeniu elementów survivalu z psychologicznym horrorem, Darkwood stanowi wyjątkową pozycję na rynku gier, zdobywając uznanie zarówno wśród graczy, jak i krytyków. Gra stanowi doskonałym przykładem tego, jak innowacyjność i kreatywne podejście mogą odświeżyć i wzbogacić gatunek horrorów.



Fig. 13: Przedstawienie rozgrywki w Darkwood Źródło: <https://store.steampowered.com/app/274520/Darkwood/?l=polish>

## 4 Narzędzia i metody

### 4.1 Program Unity

Unity to środowisko deweloperskie, dla branży gier, które umożliwia tworzenie projektów od prostych po zaawansowane. Unity oferuje intuicyjny interfejs umożliwiający tworzenie złożonych scen i zarządzanie nimi. Deweloperzy mogą łatwo importować modele 3D, tekstury i efekty specjalne, wspierane przez dynamiczną integrację zewnętrznych pakietów graficznych i animacyjnych, co przyspiesza proces produkcyjny. Unity, wspiera język C#. API to pozwala na interakcję z elementami gry, zarządzanie danymi, obsługę wejścia użytkownika i symulacje fizyczne. Unity zapewnia narzędzie do tworzenia programowania logiki opartego na C#. W Unity każdy element gry jest traktowany jako "GameObject" wyposażony w komponenty takie jak skrypty, kolidery czy renderery, co pozwala na elastyczne projektowanie elementów gry. Unity posiada zaawansowane narzędzia do animacji, obsługujące animacje szkieletowe, umożliwiając deweloperom łatwą implementację płynnych i realistycznych animacji. Unity zostało wykorzystane do stworzenia gry, począwszy od najprostszych elementów aż po najbardziej skomplikowane aspekty rozgrywki. W Unity udało się nam zaprogramować poruszanie się gracza. Poruszanie to zostało zaimplementowane poprzez wykorzystanie komponentu Rigidbody oraz skryptu kontrolującego wejścia użytkownika, który

interpretuje ruchy klawiatury oraz myszy i odpowiednio dostosowuje pozycję oraz rotację postaci w grze.

Oprócz tego, stworzyliśmy bardziej skomplikowane mechanizmy, takie jak system tworzenia nowych przedmiotów. Skrypt odpowiedzialny za tworzenie przedmiotów sprawdza, czy gracz posiada odpowiednią liczbę wymaganych przedmiotów w ekwipunku lub pobliskiej skrzynce. Po spełnieniu warunków, skrypt pobiera te przedmioty i po zakończeniu procesu zwraca nowo utworzony przedmiot do ekwipunku gracza.

Dodatkowo, zaimplementowany został skrypt do przetapiania surowców, na przykład na sztabki żelaza. Dzięki niemu gracz może przetapiać surowce, wkładając do pieca patyki, które służą jako paliwo, oraz rudy metali, które po przetworzeniu zamieniają się w sztabki.

Unity zostało również wykorzystane do stworzenia mechaniki, która umożliwia graczowi zbieranie przedmiotów rozrzuconych po świecie gry. Przedmioty te można podnosić i dodawać do ekwipunku, co wzbogaca rozgrywkę i umożliwia dalsze przetwarzanie oraz tworzenie nowych elementów w grze. Unity oferuje możliwości konfiguracji oświetlenia i cieniowania, które są kluczowe dla estetyki gry. System umożliwia również tworzenie różnorodnych efektów wizualnych, w tym zaawansowanych systemów cząsteczkowych. System zarządzania dźwiękiem pozwala na precyjną kontrolę ścieżki dźwiękowej gry, włączając przypisywanie dźwięków do zdarzeń i kontrolę ich głośności. Unity jest wyposażone w zaawansowany system fizyki, który pozwala na realistyczne symulowanie ruchu i interakcji między obiektymi.

Wybór odpowiedniej technologii do realizacji projektu jest kluczowy dla jego sukcesu. Po dokładnej analizie dostępnych narzędzi, zdecydowaliśmy się na wykorzystanie silnika Unity. Poniżej przedstawiono uzasadnienie naszego wyboru oraz powody, dla których odrzuciliśmy alternatywne rozwiązania, takie jak Unreal Engine i Godot.

Decyzja o wyborze Unity opierała się na wcześniejszych doświadczeniach zespołu z tym narzędziem. Członkowie projektu mieli już podstawową wiedzę na temat środowiska programistycznego. Zespół posiadał doświadczenie w korzystaniu z Unity, obejmujące podstawy programowania w tym środowisku, obsługę eventów, poruszanie się

postaci, tworzenie grafiki 2D i 3D i podstawowe zasady fizyki w Unity. Umożliwiło to sprawną współpracę i realizację zadań.

Unreal Engine, choć jest potężnym narzędziem, został odrzucony z kilku powodów. Przede wszystkim, żaden z członków zespołu nie miał doświadczenia z tym środowiskiem programistycznym. To mogłoby opóźnić rozpoczęcie prac nad projektem. Ponadto, Unreal Engine jest bardzo rozbudowanym narzędziem, często wykorzystywanym do dużych, zaawansowanych projektów, był zbyt skomplikowany dla naszych potrzeb.

Godot również został odrzucony z powodu braku doświadczenia i znajomości środowiska w zespole. Brak wcześniejszej znajomości tego narzędzia mógłby znaczco wydłużyć prace nad projektem. Ponadto, Godot jest głównie używany do tworzenia gier 2D, podczas gdy nasz projekt skupiał się na grafice 3D. Z tego powodu uznaliśmy, że Godot nie będzie odpowiednim narzędziem dla potrzeb projektu.

## 4.2 Program Blender

Narzędzie do modelowania, animacji i renderowania grafiki 3D, którego używaliśmy, ma szerokie zastosowanie w różnych branżach, w tym w tworzeniu gier komputerowych. Dzięki ogromnym możliwościom to narzędzie pozwala na szczegółowe modelowanie obiektów, tworzenie zaawansowanych animacji oraz symulacji.

Blender został użyty do tworzenia kluczowych elementów środowiska gry, szaty graficznej oraz modeli 3D. Został wykorzystany do stworzenia niezbędnych elementów, takich jak piec. Dzięki niemu wygenerowaliśmy narzędzie dla gracza o odpowiedniej strukturze i realistycznym wyglądzie.

Okazał się niezbędny do modelowania większych elementów, takich jak piec, który w grze służy do przetapiania metali. Dzięki jego możliwościom udało się osiągnąć wysoki poziom detali, co znacząco przyczynia się do realizmu i immersyjności gry.

Również skrzynie, które służą do przechowywania przedmiotów zbieranych przez graczy, zostały zaprojektowane przy pomocy tego narzędzia. Modelowanie takich obiektów wymagało szczegółowej pracy nad teksturami i proporcjami, aby skrzynie wyglądały naturalnie i pasowały do estetyki reszty świata gry.

Najpierw przystąpiliśmy do zaplanowania i zebrania referencji dotyczących modelu, co obejmowało poszukiwanie inspiracji oraz analizę istniejących obiektów, aby lepiej zrozumieć proporcje, kształty i detale, które chcieliśmy odzwierciedlić w naszym projekcie.

Następnie rozpoczęliśmy tworzenie modelu w Blenderze, zaczynając od formowania podstawowych brył takich jak sześciany, cylindry czy sfery. Użyliśmy narzędzi do ekstrudowania, skalowania i obracania, aby nadać modelowi odpowiedni kształt i formę. W trakcie modelowania, korzystaliśmy z funkcji „Loop Cut” do dodawania dodatkowych linii podziału, co pozwoliło nam na dokładniejsze odwzorowanie detali i lepsze kontrolowanie topologii modelu. Dzięki temu mogliśmy uzyskać bardziej złożone kształty, które odpowiadały naszym początkowym założeniom.

Na końcu przystąpiliśmy do renderowania modelu, używając silnika renderującego Cycles lub Eevee, w zależności od potrzeb i oczekiwania co do finalnego wyglądu. W trakcie renderowania zwracaliśmy uwagę na ustawienia takie jak rozdzielcość, poziom próbek oraz denoising, aby uzyskać jak najlepszą jakość obrazu przy optymalnym czasie renderowania. Finalnie model został zapisany w wybranym formacie, gotowy do dalszej obróbki lub integracji w innych projektach.

Z wielu narzędzi dostępnych na rynku do tworzenia modeli 3D wybór padł na Blendera, ponieważ był wykorzystywany w wcześniejszych projektach wykonywanych przez nas. To przypieszyło i ułatwiło pracę. Wybraliśmy go ze względu na darmowy dostęp i charakter open source.

### 4.3 Trello

Trello to narzędzie do zarządzania projektami, które umożliwia użytkownikom organizowanie zadań i projektów poprzez interaktywne tablice. W Trello, zadania są reprezentowane przez karty, które można przemieszczać między listami odpowiadającymi różnym etapom procesu pracy. Każda karta może zawierać szczegółowe informacje, takie jak opisy, terminy realizacji, załączniki, a także mogą być przypisane do konkretnych osób lub zespołów.

W procesie tworzenia, wykorzystano Trello do organizacji pracy nad projektem, pilnowania terminów zadań oraz monitorowania postępów.

Dzięki temu narzędziu efektywnie można zarządzać różnymi aspektami projektu w czasie pracy, od projektowania grafiki, przez rozwój oprogramowania. Trello pozwoliło na utrzymanie porządku, co do bieżących i przyszłych zadań, co znaczco przyczyniło się do efektywnej współpracy i komunikacji w zespole.

Zarządzanie zadaniami w Trello jest intuicyjne i elastyczne, co umożliwia dostosowanie procesów pracy do specyficznych potrzeb projektu. Użytkownicy mogą łatwo dodawać nowe zadania, aktualizować ich status oraz przypisywać do nich odpowiedzialnych wykonawców. Trello oferuje również funkcjonalność sprawdzania, co już zostało wykonane, co jest w trakcie realizacji, a co jeszcze wymaga uwagi. To sprawia, że Trello jest nieocenionym narzędziem w zarządzaniu złożonymi projektami, takimi jak tworzenie gier komputerowych.

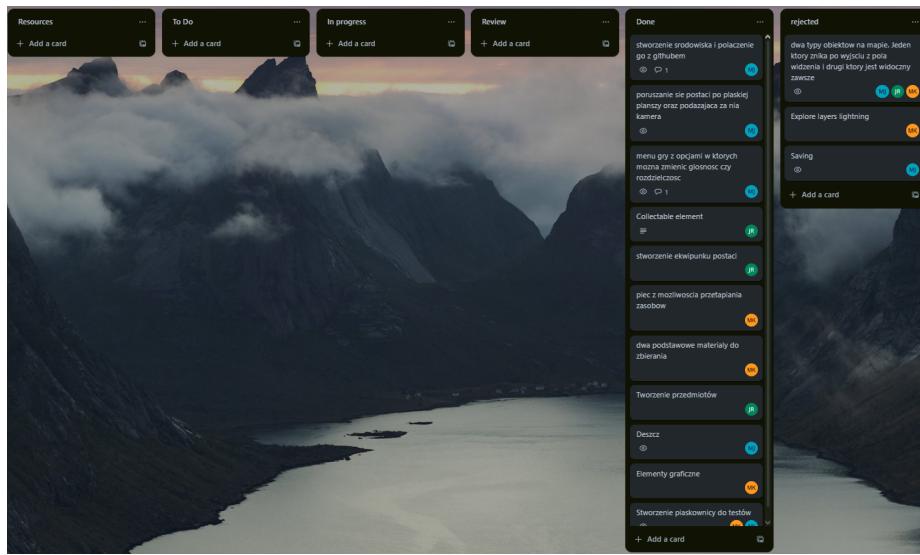


Fig. 14: Zrzut ekranu z trello

#### 4.4 Scrum

W ramach projektu zdecydowano się na zastosowanie metodyki Scrum. Metodyka Scrum koncentruje się na krótkich cyklach pracy, regularnych spotkaniach oraz ciągłej poprawie procesów i wyników.

Poniżej przedstawiono szczegóły tej metodyki oraz sposób jej wykorzystania w projekcie. W ramach Scrum praca była zorganizowana w dwutygodniowych sprintach. Każdy sprint miał jasno zdefiniowane cele, które należało osiągnąć. Sprinty pozwalały na regularne dostarczanie wyników, przegląd postępów i szybką adaptację do ewentualnych zmian. Stand-upy odbywały się co 3-4 dni. Na tych krótkich spotkaniach omawiano: Co zrobiono od ostatniego spotkania: Raportowano postępy w realizacji zadań. Co wymaga poprawy: Identyfikowano obszary wymagające poprawy lub napotkane problemy, aby szybko reagować na wyzwania. Co zostanie zrobione w przyszłości: Ustalano zadania na nadchodzące dni, zapewniając jasność co do dalszych działań. Po każdym sprincie organizowano retrospektyny, na których analizowano, co poszło dobrze, co można poprawić i jak usprawnić pracę w kolejnym sprincie. Te spotkania były kluczowe dla ciągłego doskonalenia procesów. Utrzymywano backlog produktu, który był listą wszystkich zadań i funkcji do wykonania. Backlog był regularnie aktualizowany i priorytetyzowany, aby zapewnić pracę nad najważniejszymi elementami projektu. Podczas sprintów utrzymywano stały kontakt dotyczący pracy inżynierskiej. Wykorzystywano do tego różne narzędzia komunikacyjne:

Messenger: Służył do szybkiej wymiany wiadomości i ustalania bieżących spraw.

Discord: Był głównym narzędziem do spotkań online. Dzięki Discordowi organizowano spotkania wideo i audio, dzielono się ekranem oraz pracowano wspólnie w czasie rzeczywistym. Metodykę Scrum wybrano ze względu na jej format, który idealnie pasował do potrzeb projektu inżynierskiego. Dwutygodniowe sprinty pozwalały na regularne pokazywanie wyników pracy, co zapewniało stały monitoring postępów.

#### 4.5 Github i Git

**Git** jest narzędziem typu open-source do kontroli wersji, które umożliwia zarządzanie zmianami w kodzie źródłowym podczas tworzenia oprogramowania. Narzędzie to pozwala na śledzenie zmian, cofanie się do wcześniejszych wersji kodu oraz efektywną współpracę między wieloma użytkownikami. Git został wykorzystany do współdzielenia się pracą oraz pracy nad kodem w tym samym czasie.

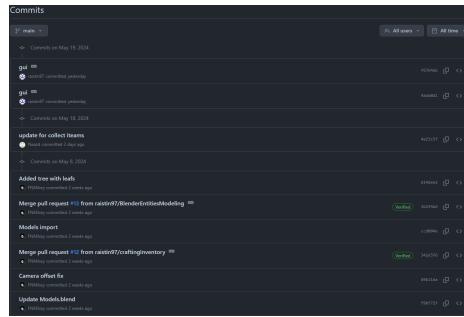


Fig. 15: Screenshot przedstawiający wykorzystanie systemu zarządzania wersją Git. Źródło: Opracowanie własne

**GitHub** jest internetową platformą hostingową, która wykorzystuje Git do zarządzania projektami programistycznymi. Platforma ta pozwala na przechowywanie projektów, współdzielenie zasobów oraz współpracę nad kodem w prosty i zorganizowany sposób. W projekcie wykorzystano Git i GitHub do pracy na wielu różnych urządzeniach oraz umożliwienia współpracy wielu osób nad różnymi etapami projektu jednocześnie. Dzięki Git każdy członek zespołu mógł pracować lokalnie nad elementami rozgrywki, tworząc osobne gałęzie kodu (branches), które później mogły być scalane bez ryzyka utraty pracy innych. GitHub z kolei ułatwił łączenie kodu gry (merging) i zarządzanie projektem poprzez centralizowanie wszystkich zasobów i historii zmian w jednym miejscu.

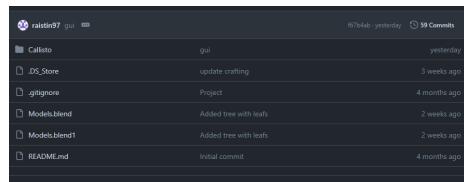


Fig. 16: Screenshot przedstawiający wykorzystanie systemu zarządzania wersją Git. Źródło: Opracowanie własne

Wymienione wyżej narzędzia ułatwiały zarządzanie złożonym projektem, synchronizację pracy oraz utrzymanie porządku i przejrzystości w procesie tworzenia gry.

## 5 Rezultaty

### 5.1 Trzecioosobowa, izometryczna kamera

#### 5.2 Ruch gracza

Ruch gracza został zaplanowany w taki sposób żeby był niezależny od obrotu postaci względem osi x-z. W tym celu kod pobiera jaki przycisk jest aktualnie wcisniety oraz pozycję myszki. Na podstawie tych zmiennych oblicza w którym kierunku powinna poruszać się postać aby był to odpowiedni kierunek względem osi x-z oraz dodaje do niego obrót o 45 stopni który jest związany z obrotem kamery względem gracza o dokładnie taką wartość.

---

```

float moveX = Input.GetAxisRaw("Horizontal");
float moveZ = Input.GetAxisRaw("Vertical");

Vector3 moveDirection =
    Quaternion.Euler(0, 45, 0) * new Vector3(moveX, 0f, moveZ);

Ray cameraRay = cam.ScreenPointToRay(Input.mousePosition);
Plane groundPlane = new Plane(Vector3.up, Vector3.zero);
float rayLength;

if (groundPlane.Raycast(cameraRay, out rayLength))
{
    mousePos = cameraRay.GetPoint(rayLength);

    Quaternion newRotation =
        Quaternion
            .LookRotation(new Vector3(mousePos.x -
                transform.position.x,
            0f,
            mousePos.z - transform.position.z));
    rigidBody.MoveRotation (newRotation);
}

```

---

Fig. 17: Ruch postaci względem kamery: Opracowanie własne

Dodatkowo żeby prędkość poruszania się postaci nie była zależna od prędkości procesora została zastosowana funkcja która mnoży prędkość gracza przez "Time.fixedDeltaTime" co umożliwia uzyskanie

predkości niezależnej od takich zmiennych. W tym fragmencie kodu zostaje nadana ostateczna predkość poruszania się postaci na co również ma wpływ to czy gracz wciska przycisk odpowiedzialny za bieg co zwiększa predkość półtora razy.

---

```
void Movement()
{
    rigidBody.MovePosition(rigidBody.position + movement * moveSpeed
                           * Time.fixedDeltaTime);
}
```

---

Fig. 18: Ostateczna prędkość poruszania się postaci: Opracowanie własne

### 5.3 Interakcją gracza z otoczeniem

Zdobywanie przedmiotów jest realizowane za pomocą prostej mechaniki zbierania przedmiotów. Proces ten umożliwia graczowi zdobywanie podstawowych elementów, takich jak patyki czy kamienie. Opiera się to na kilku prostych metodach. Mechanika zbierania przedmiotów polega na interakcji gracza z obiekta znajdującymi się w świecie gry. Gracz, przemieszczając się, ma możliwość zbierania różnych przedmiotów poprzez interakcję z nimi. W momencie, gdy gracz zbliży się do przedmiotu, przedmiot znika, pojawiając się w ekwipunku gracza. Odpowiedzialne jest za to przypisanie odpowiednich interaktywnych właściwości przedmiotą w grze i implementacja odpowiednich metod.

W kodzie została zaimplementowana metoda PickUpItems, która jest odpowiedzialna za podnoszenie itemów. Odbiera się to najpierw na sprawdzeniu czy ekwipunek posiada jeszcze wolne sloty. Jeśli, w ekwipunku są wolne miejsca, przedmiot zostaje dodany w pierwsze wolne miejsce. Całość jest oparta na wywoływaniu metody PickUpItem za pomocą OnTriggerEnter jeśli Player zderzy się z przedmiotem na którym nałożony jest colider z tagiem "Player" przedmiot zostanie zebrany

---

```
public void PickupItem(Item itemToPickup)
{
    if (inventoryManager.IsInventoryFull())
    {
        Debug.Log("Ekwipunek jest pełen");
        return;
    }

    if (inventoryManager.AddItem(itemToPickup))
    {
        Debug.Log($"{itemToPickup.itemName} został dodany.");
        gameObject.SetActive(false);
    }
    else
    {
        Debug.Log($"{itemToPickup.itemName} nie został dodany.");
    }
}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        if (item != null)
        {
            PickupItem(item);
        }
    }
}
```

---

Fig. 19: Podnoszenie przedmiotów: Opracowanie własne

## 5.4 Wydobywanie zasobów

W naszej grze zaimplementowaliśmy również mechanikę wydobywania zasobów. Stworzony został nowy skrypt który zajmuje się losowaniem szansy na wypadnięcie określonego surowca, z uwzględnieniem ogległości w której gracz znajduje się od obiektu:

---

```

void TryMine()
{
    float distance = Vector3.Distance(player.transform.position,
        transform.position);

    if (distance <= miningRange)
    {
        int randomValue = Random.Range(0, 100);

        if (randomValue < kamienChance)
        {
            SpawnItem(kamienPrefab);
        }
        else if (randomValue < kamienChance + srebroChance)
        {
            SpawnItem(srebroPrefab);
        }
        else if (randomValue < kamienChance + srebroChance + złotoChance)
        {
            SpawnItem(złotoPrefab);
        }

        PlayMiningParticles();

        TakeDamage();
    }
}

```

---

Fig. 20: Wydobywanie zasobów: Opracowanie własne

Metoda `TakeDamage()` jest odpowiedzialna za inkrementacyjne odbieranie wytrzymałości kopanego obiektu oraz za jego zniknięcie ze sceny po przekroczeniu limitu wytrzymałości.

Do animacji kopania został dodany efekt "Particle" zaimplementowany w metodzie `PlayMiningParticles()`, który jest widoczny pod-

czas kopania. Efekt ten został dodany, aby nadać grze więcej imersji i dać graczowi feedback odnośnie wykonywanych akcji. Dzięki niemu gracz będzie wiedział, że stoi wystarczająco blisko obiektu i wydobywanie przedmiotów odbywa się z sukcesem. Poniżej można zobaczyć implementację:

---

```

void PlayMiningParticles()
{
    if (miningParticlesPrefab != null)
    {
        Vector3 direction = (player.transform.position -
            transform.position).normalized;
        Vector3 spawnPosition = player.transform.position - direction *
            2f;
        Quaternion rotation = Quaternion.LookRotation(direction);

        GameObject particles = Instantiate(miningParticlesPrefab,
            spawnPosition, rotation);
        ParticleSystem particleSystem =
            particles.GetComponent<ParticleSystem>();
        if (particleSystem != null)
        {
            particleSystem.Play();
        }
        else
        {
            ParticleSystem[] childParticleSystems =
                particles.GetComponentsInChildren<ParticleSystem>();
            foreach (ParticleSystem ps in childParticleSystems)
            {
                ps.Play();
            }
        }

        Destroy(particles, particleSystem.main.duration +
            particleSystem.main.startLifetime.constantMax);
    }
}

```

---

Fig. 21: Efekt częsteszkowy podczas wydobywania zasobów: Opracowanie własne

Ten sam skrypt został użyty do wydobywania suroców z kamieni ale także do drzew, z których można pozyskać patyki.

## 5.5 Wytwarzanie zasobów

Proces wytwarzania nowych przedmiotów w grze opiera się na łączeniu podstawowych elementów w bardziej skomplikowane obiekty za pomocą określonych receptur. Najpierw konieczne jest zebranie wszystkich niezbędnych elementów. Po zgromadzeniu wymaganych surowców przeszukiwany jest ekwipunek gracza w celu znalezienia przedmiotów potrzebnych według receptury. W przypadku, gdy wszystkie wymagane przedmioty znajdują się w ekwipunku, możliwe jestowanie nowego przedmiotu.

---

```

public bool CheckRecipeIngredients(Recipe recipe, out string
    inventoryContents, out Dictionary<string, int>
    remainingChestItems, out Dictionary<string, int>
    usedChestItems, out Dictionary<string, int>
    missingItemsDict)
{
    remainingChestItems = chest.Openchest() ?
        chest.GetItemsFromChestWithCounts() : new Dictionary<string,
        int>();
    usedChestItems = new Dictionary<string, int>();
    missingItemsDict = new Dictionary<string, int>();

    Dictionary<string, int> inventoryCounts = GetInventoryCounts();
    inventoryContents = BuildInventoryContents(inventoryCounts,
        remainingChestItems);

    return VerifyAndUpdateCounts(recipe, inventoryCounts,
        remainingChestItems, usedChestItems, missingItemsDict, ref
        inventoryContents);
}

```

---

Fig. 22: Tworzenie przedmiotów: Opracowanie własne

W przypadku wykrycia brakujących przedmiotów w skrzyni, są one wykorzystywane do uzupełnienia braków w ekwipunku gracza, co umożliwia wytworzenie przedmiotu. Proces ten gwarantuje efektowne zarządzanie zasobami przez gracza, umożliwiając wykorzystanie dostępnych przedmiotów zarówno w ekwipunku, jak i w skrzyniach, w celu tworzenia nowych, bardziej skomplikowanych obiektów zgodnie z określonymi recepturami.

---

```

        public Dictionary<string, int> GetItemsFromChestWithCounts()
{
    Dictionary<string, int> itemCounts = new Dictionary<string, int>();

    if (chestSlots == null)
    {
        Debug.LogWarning("Miejsce jest puste");
        return itemCounts;
    }

    for (int i = 0; i < chestSlots.Length; i++)
    {
        InventorySlot slot = chestSlots[i];
        InventoryItem itemInSlot =
            slot.GetComponentInChildren<InventoryItem>();

        if (itemInSlot != null && itemInSlot.item != null)
        {
            string itemName = itemInSlot.item.itemName;
            if (itemCounts.ContainsKey(itemName))
            {
                itemCounts[itemName] += itemInSlot.count;
            }
            else
            {
                itemCounts[itemName] = itemInSlot.count;
            }

            Debug.Log($"Miejsce {i}: {itemName}, Ilosc:
                {itemInSlot.count}");
        }
        else
        {
            Debug.Log($"Miejsce {i} jest puste");
        }
    }
    foreach (var item in itemCounts)
    {
        Debug.Log($"Przedmiot: {item.Key}, Ilosc: {item.Value}");
    }

    return itemCounts;
}
private Dictionary<string, int> GetInventoryCounts()
{
    Dictionary<string, int> inventoryCounts = new Dictionary<string,
        int>();

    foreach (var slot in inventorySlots)
    {
        InventoryItem itemInSlot =
            slot.GetComponentInChildren<InventoryItem>();
        if (itemInSlot != null)
        {
            if (inventoryCounts.ContainsKey(itemInSlot.item.itemName))
            {
                inventoryCounts[itemInSlot.item.itemName] +=
                    itemInSlot.count;
            }
            else
            {

```

Gdy gracz posiada otwartą skrzynię w niedalekiej odległości, proces poszukiwania potrzebnych elementów rozpoczyna się od przeszukania plecaka gracza. Najpierw analizowane są zasoby znajdujące się bezpośrednio w ekwipunku gracza. Jeżeli okaże się, że w plecaku brakuje wymaganych przedmiotów, a skrzynia jest otwarta i znajduje się w zasięgu interakcji, następuje automatyczne przeszukanie skrzyni w celu zlokalizowania brakujących elementów. Procedura ta opiera się na założeniu, że gracz może przechowywać dodatkowe zasoby w skrzyni, co pozwala na efektywne zarządzanie posiadanymi przedmiotami. System ten umożliwia graczowi szybkie i wygodne zdobycie niezbędnych zasobów bez konieczności ręcznego przeglądania wszystkich dostępnych pojemników.

---

```
public void CraftItem(Recipe recipe)
{
    bool success = inventoryManager.TryCraftItem(recipe);
    if (success)
    {
        StartCreatingItem();
    }
}
```

---

Fig. 24: Tworzenie przedmiotów: Opracowanie własne

Metoda TryCraftItem analizuje zawartość plecaka i skrzyni gracza, sprawdzając, czy wszystkie niezbędne składniki są dostępne w wymaganych ilościach. Jeśli warunki te są spełnione, metoda informuje system, że wytworzenie przedmiotu jest możliwe, zwracając wartość true. Następnie metoda CraftItem może być wywołana w celu rzeczywistego stworzenia przedmiotu na podstawie dostarczonego przepisu.

## 5.6 Wytapianie zasobów

Gra pozwala na wytapianie przedmiotów za pomocą pieca. Proces ten wymaga dostarczenia paliwa, którym są patyki. Piec umożliwia przetapianie rud żelaza na sztabki żelaza. Dzięki temu procesowi gracz zyskuje niezbędne materiały do stworzenia przedmiotów o

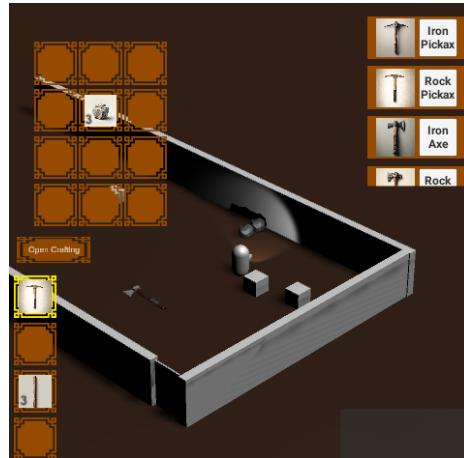


Fig. 25: Screenshot przedstawiający UI gry ekwipunku gracza i skrzyni we wczesnej fazie projektu. Źródło: Opracowanie własne

wyższej jakości. Przetapianie odbywa się poprzez umieszczenie rudy żelaza w piecu oraz dostarczeniu niezbędnego paliwa. Po określonym czasie, w wyniku działania pieca, ruda żelaza przetapiana jest w sztabki żelaza.

Metoda Smelt odpowiada za przetapianie rud na sztabki metali, zaczynając od sprawdzenia dostępności paliwa i rud metali. Jeśli obie te zasoby są dostępne, proces przetapiania rozpoczyna się i trwa przez określony czas, po którym ruda metalu zostaje przekształcona w sztabkę. W trakcie procesu ilość paliwa i rud jest stopniowo zmniejszana, a ilość sztabek metali wzrasta. Mechanizm ten umożliwia efektywne przekształcanie surowców w wartościowe materiały, co jest kluczowym elementem rozwoju ekwipunku w grze.

## 5.7 Zapis gry

Naszym celem było uzyskanie zapisu gry poprzez sposób niejawny czyli taki który uniemożliwia graczowi prostu dostęp do danych i możliwości ich zmiany co mogło by wprowadzić sytuacje w której gracz oszukuje gre. Niestety z powodu ograniczenia silnika jak i problemów w które silnik Unity posiada przy próbie konwersji swoich klas wewnętrznych na pliki niejawne pomysł ten został przełożony w czasie na następne etapy produkcji. Niemniej jednak zostały za-



Fig. 26: Screenshot przedstawiający UI pieca. Źródło: Opracowanie własne

implementowane obiekty które pod ten system będą służyć takie jak np przyciski i okna które będą odpowiadały za tworzenie nowej gry jak i za wczytanie gry już istniejącej.

## 5.8 Ustawienia

Oczekiwane efekty uzyskaliśmy poprzez użycie klas unity które dają dostęp do wewnętrznych elementów graficznych oraz zapis do pliku.

Żeby uniknąć sytuacji w której gracz mógł by wybrać rozdzielczość która nie jest obsługiwana przez jego komputer napisaliśmy pętle która w momencie uruchomienia sprawdza wszystkie dostępne rozdzielczości i dodaje je do listy która zostanie wyświetlona w polu wyboru rozdzielczości.

Żeby osiągnąć zapis oraz odczyt wybranych wcześniej ustawień musieliśmy zastosować funkcję "PlayerPrefs" która umożliwia zapis preferencji gracza w łatwy sposób. W tym celu napisaliśmy classy których zadaniem jest pobranie wybranej opcji graficznej bądź dźwiękowej i zapisanie jej do pliku a następnie w momencie uruchomienia gry na nowo bądź jej zmiany ponowne jej zapisanie lub odczytanie.

---

```

    IEnumerator Smelt()
    {
        InventoryItem fuelSlotItem =
            fuelSlot.GetComponentInChildren<InventoryItem>();
        InventoryItem oreSlotItem =
            oreSlot.GetComponentInChildren<InventoryItem>();
        if (fuelSlotItem == null || oreSlotItem == null)
        {
            Debug.Log("Nie ma paliwa albo rudy");
            yield break;
        }

        for (int i = 0; i < 5; i++)
        {
            yield return new WaitForSeconds(time / 5);
            if (fuelSlotItem != null)
            {
                fuelSlotItem.count--;
                if (fuelSlotItem.count <= 0)
                {
                    Destroy(fuelSlotItem.gameObject);
                    fuelSlotItem = null;
                }
                else
                {
                    fuelSlotItem.RefreshCount();
                }
            }
            if (oreSlotItem != null)
            {
                oreSlotItem.count--;
                if (oreSlotItem.count <= 0)
                {
                    Destroy(oreSlotItem.gameObject);
                    oreSlotItem = null;
                }
                else
                {
                    oreSlotItem.RefreshCount();
                }
            }
            InventoryItem productItem =
                productSlot.GetComponentInChildren<InventoryItem>();
            Debug.Log("Nazwa produktu" + productItem);
            if (productItem == null)
            {
                inventoryManager.SpawnNewItem(item, productSlot);
                Debug.Log("Liczba rud po zmniejszeniu: " +
                    productItem);
            }
            else
            {
                productItem.count+=productAmount;
                productItem.RefreshCount();
            }
        }
    }
}

```

---

Fig. 27: Przetapianie zasobów: Opracowanie własne

Listing 1.1: Sprawdzanie dostępnych rozdzielczości ekranu

```

1   resolutions = Screen.resolutions;
2   resolutionDropDown.ClearOptions();
3   List<string> options = new List<string>();
4   int currentResolutionIndex = 0;
5
6   for (int i = 0; i < resolutions.Length; i++)
7   {
8       string option = resolutions[i].width + " x " +
9           resolutions[i].height;
10      options.Add(option);
11
12      if (resolutions[i].width == Screen.currentResolution.width &&
13          resolutions[i].height == Screen.currentResolution.height)
14      {
15          currentResolutionIndex = i;
16      }
17
18      resolutionDropDown.AddOptions(options);

```

Fig. 28: Sprawdzanie dostępnych rozdzielczości ekranu

### 5.9 Deszcz

Żeby uzyskać deszcz zastosowaliśmy obiekt nazwany "Particles" który został wydłużony w odpowiednich osiach w celu uzyskania wrażenia spadającej kropli wody. Zostały na nim również nałożone odpowiednie systemy których celem jest spowodowanie aby wyglądał on realnie. W tym celu zostały zmienione ustawienia kolizji oraz ilości pojawiających się kropel.

Żeby osiągnąć efekt w którym pogoda zmienia się z słonecznej na deszczową został napisany kod którego celem jest losowe wybieranie wartości z zakresu pomiędzy 10min do 30min dla aktywności deszczu oraz z zakresu pomiędzy 5min do 60min dla braku jego aktywności.

Dzięki użyciu takiej funkcji gracz nie ma wrażenia że pogoda jest uzależniona od postępów fabuły czy wydań na mapie, a jest obiektem który jest niezależny od sytuacji dzierżących się w świecie gry i nie można przewidzieć dokładnie kiedy pogoda ulegnie zmianie oraz na jak długi czas pogoda się zmieni.

Żeby uniknąć sytuacji w której deszcz mógł by drastycznie wpływać na ilość klatek animacji wyświetlanych na ekranie zastosowa-

---

```

if (PlayerPrefs.HasKey("QualityLevel"))
{
    int qualityLevel = PlayerPrefs.GetInt("QualityLevel");
    QualitySettings.SetQualityLevel(qualityLevel);
    graphics.SetValueWithoutNotify(qualityLevel);
}

public void SetQuality(int quality)
{
    QualitySettings.SetQualityLevel(quality);
    PlayerPrefs.SetInt("QualityLevel", quality);
}

```

---

Fig. 29: Odczyt i zapis ustawień:  
[https://docs.unity3d.com/ ScriptReference/PlayerPrefs.html](https://docs.unity3d.com/ScriptReference/PlayerPrefs.html)

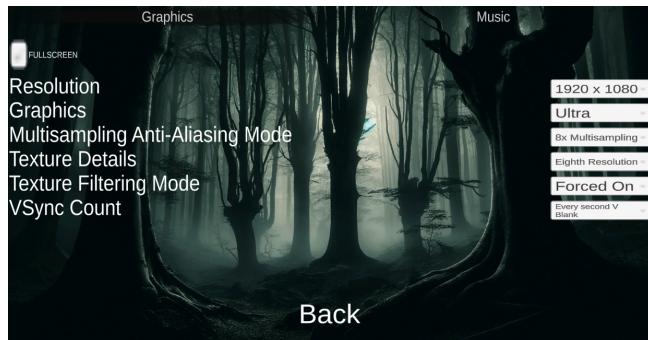


Fig. 30: Zakładka zmiany ustawień graficznych



Fig. 31: Zakładka zmiany ustawień dźwięku

---

```

private void SetNextToggleTime()
{
    if (isEffectActive)
    {
        timeToNextToggle = Random.Range(10 * 60, 30 * 60);
    }
    else
    {
        timeToNextToggle = Random.Range(5 * 60, 60 * 60);
    }
}

```

---

Fig. 32: Zmiana pogody: Opracowanie własne

iśmy rozwiązanie którego efektem jest mały obiekt "Particles" który znajduje się nad graczem i podąża za graczem znajdując się 15 jednostek nad nim. Dzięki takiemu rozwiązaniu efekt ten nie jest nałożony na cały obszar mapy a jedynie na fragment mapy na którym znajduje się gracz co zmniejsza jednocześnie ilość obliczeń które komputer musi przetworzyć.

---

```

private void UpdateEffectPosition()
{
    if (player != null)
        particleEffect.transform.position = new
            Vector3(player.position.x, player.position.y + 15,
                    player.position.z);
}

```

---

Fig. 33: Pozycjonowanie deszczu nad graczem: Opracowanie własne

## 5.10 Testy

Testowanie opierało się na wykonaniu kroków zgodnie ze scenariuszem testowym i osiągnięciu założonych poprawnych wyników. Testy były wykonywane tylko manualnie.

Testy pozwalały na znalezienie błędów, problemów w wykonywaniu kodu oraz w logice gry. Jednym z wykrytych błędów był

problem związany z liczbą elementów usuwanych podczas tworzenia nowych przedmiotów z ekwipunku i skrzyni. Usuwana była ta sama ilość przedmiotów zarówno z skrzyni, jak i z ekwipunku gracza, co powodowało zbyt duże zużycie zasobów na wytworzenie jednego produktu. W efekcie zużywano dwa razy więcej materiałów niż zakładała receptura.

Dzięki temu w procesie tworzenia gry można było od razu poddać tak poważne błędy naprawie i weryfikacji. Testowanie okazało się niezbędne w procesie tworzenia gry. Dzięki takiemu podejściu udało się osiągnąć założone cele. Pozwoliło to na identyfikację i naprawę błędów. Takie działania przyczyniły się do poprawy jakości gry.

## 6 Dyskusja

Chciałbym przedstawić dalsze pomysły i mechaniki, które planujemy dodać do gry. Mają one głównie na celu poprawić jakość rozgrywki. Chcemy się skupić na rozbudowie świata gry oraz zwiększyć zaangażowanie gracza, planując dodanie nowych możliwości rozwoju postaci, nowych przedmiotów oraz wyzwań.

- **Tworzenie nowych przedmiotów** stanowiło jedno z kluczowych wyzwań w procesie rozwoju gry. Jednym z głównych kroków tego procesu było ustalenie, która z skrzyń jest otwarta i leży najbliżej gracza. To umożliwiło prawidłowe zarządzanie zawartością oraz dostęp do niezbędnych przedmiotów. Rozwiążanie tego problemu wymagało implementacji metody sprawdzającej odległość skrzynki od gracza oraz wykorzystania słownika do zapisywania dostępnych przedmiotów i ich ilości. Podczas tworzenia przedmiotów napotkano problem związany z sprawdzaniem brakujących elementów potrzebnych do realizacji receptury. Aby rozwiązać ten problem konieczne było sprawdzenie zawartości skrzynek, jak i ekwipunku gracza. Proces ten polegał na porównaniu ilości przedmiotów w ekwipunku i skrzynce z niezbędnymi składnikami zapisanymi w recepturze. Problem został rozwiązany poprzez sprawdzenie w pierwszej kolejności ilości przedmiotów w ekwipunku i porównanie ich z wymaganą ilością w recepturze. Gdy ilość była zbyt mała, a skrzynka była otwarta i znajdowała się w odpowiedniej odległości, sprawdzano, czy w skrzynce znajduje się brakująca ilość

**Scenariusz Testowy 1: Dodawanie Przedmiotów do Ekwipunku**

**Cel:** Sprawdzenie, czy dodawanie różnych przedmiotów do ekwipunku działa poprawnie.

**Kroki:**

1. Użytkownik otwiera ekwipunek.
2. Użytkownik dodaje 1 kilof do ekwipunku.
3. Użytkownik sprawdza, czy kilof jest widoczny w ekwipunku.
4. Użytkownik dodaje 4 patyki do ekwipunku.
5. Użytkownik sprawdza, czy 4 patyki są widoczne w ekwipunku.
6. Użytkownik dodaje 1 topór do ekwipunku.
7. Użytkownik sprawdza, czy topór jest widoczny w ekwipunku.

**Oczekiwany wynik:**

Po dodaniu kilofa, patyków i topora, każdy z tych przedmiotów jest widoczny w ekwipunku w odpowiedniej ilości.

**Scenariusz Testowy 2: Przenoszenie Przedmiotów w Ekwipunku**

**Cel:** Weryfikacja, czy użytkownik może przenosić przedmioty ekwipunku między wszystkimi wolnymi slotami.

**Kroki:**

1. Użytkownik otwiera ekwipunek, który zawiera przynajmniej jeden przedmiot.
2. Użytkownik wybiera przedmiot do przeniesienia.
3. Użytkownik wybiera wolny slot docelowy.
4. Użytkownik przenosi wybrany przedmiot do wybranego wolnego slotu.
5. Użytkownik powtarza proces dla innych przedmiotów i slotów.

**Oczekiwany wynik:**

1. Użytkownik może swobodnie przenosić przedmioty między wolnymi slotami, a zmiany są odzwierciedlone w ekwipunku.

**Scenariusz Testowy 3: Stakowanie Przedmiotów w Ekwipunku**

**Cel:** Testowanie stackowania przedmiotów w ekwipunku.

**Kroki:**

1. Użytkownik otwiera ekwipunek.
2. Użytkownik dodaje do ekwipunku 4 patyki w jednym slocie.
3. Użytkownik próbuje dodać kolejne 4 patyki w jednym slocie.
4. Użytkownik dodaje do ekwipunku 1 kilof.
5. Użytkownik próbuje dodać kolejny kilof do tego samego slotu.
6. Użytkownik dodaje do ekwipunku 1 topór.
7. Użytkownik próbuje dodać kolejny topór do tego samego slotu.

Fig. 34: Screenshot scenariusza testowego

- **Wytapianie przedmiotów**, problemem pojawił się ze śledzeniem ilości zużywanych zasobów. Zastosowanie funkcji RefreshCount() pozwoliło na regularne zmniejszanie liczby przedmiotów w piecu aż do osiągnięcia ilości równej zero, co oznaczało pełne wykorzystanie zasobów. To umożliwiło wyświetlenie bieżącego stanu zasobów.
- **System walki** będzie odbywał się w turowym systemie walki. W tym systemie gracze i przeciwnicy wykonują akcje w ustalonej kolejności, która jest losowana na początku walki. Podczas swojej tury każdy uczestnik może zaatakować, zadając obrażenia przeciwnikowi, które są odejmowane od jego punktów życia (HP). Istnieje także możliwość wybrania obrony, co zwiększa odporność na obrażenia do końca tury. Gracze mogą również korzystać z zaklęć leczniczych, aby przywrócić HP sobie lub sojusznikom. Alternatywnie, można podjąć próbę ucieczki, co kończy udział w walce. Walka trwa, dopóki wszystkie jednostki jednej ze stron nie zostaną pokonane lub ktoś nie zdecyduje się uciec.
- **Przeciwnicy** będą pojawiać się w późniejszych lokacjach, a ich zadaniem będzie pilnowanie określonych obszarów lub zasobów. Każdy przeciwnik będzie miał wyznaczoną ścieżkę, po której będzie się poruszał, patrolując dany teren. Dzięki wsparciu AI, przeciwnicy będą mogli dokładniej i bardziej realistycznie reagować na obecność gracza, w tym efektywnie wyszukiwać go oraz podejmować działania w zależności od sytuacji. To umożliwia stworzenie bardziej dynamicznego i immersyjnego systemu interakcji z przeciwnikami.

## 7 Wnioski i przyszłe prace

Na podstawie przeprowadzonych prac można wyciągnąć szereg wartościowych wniosków, które mogą znacząco poprawić przyszłe projekty. Przede wszystkim, lepsze przygotowanie do pracy i bardziej szczegółowe rozplanowanie podziału zadań oraz obowiązków jest kluczowe. Rozłożenie zadań na mniejsze, bardziej zarządzalne fragmenty pozwala na efektywniejsze zarządzanie czasem oraz zasobami, co w konsekwencji prowadzi do zwiększenia wydajności pracy. Napewno kolejne prace

zyskałyby na przygotowaniu lepszego harmonogramu. Umożliwi to skuteczniejsze zarządzanie przyszłymi projektami. Pozwoli to na terminowe wykonanie poszczególnych etapów. Napewno przy przyszłym planowaniu nowych prac, trzeba uwzględnić dostęp do zasobów i uwzględnić wymagaia projektu. Takie podejście może znacznie usprawnić prace i czas realizacji. Napewno takie podejście zminimizuje ryzyko opóźnień i problemów technicznych. Kolejnym nasuwającym się istotnym wnioskiem jest zmiana sposobu myślenia o projekcie. Przy podejściu do tego projektu panowało myślenie jako pojedynczych metod i mechanik działających w grze, a nie całej wspólnej całości. Każdy fragment świata, kodu współdziała z innymi, co wymaga konsekwentnego podejścia do projektowania i implementacji. Wszystkie komponenty muszą współpracować ze sobą, jest kluczowe dla osiągnięcia spójności i funkcjonalności całego systemu.

## 8 Bibliografia

### List of Figures

1	Model kamienia Źródło: Opracowanie własne.....	15
2	Model złota Źródło: Opracowanie własne .....	15
3	Model żelaza Źródło: Opracowanie własne .....	15
4	Modele patyków Źródło: Opracowanie własne .....	16
5	Model drzewa (twisted) Źródło: Opracowanie własne .....	16
6	Model drzewa (liscie) Źródło: Opracowanie własne .....	17
7	Modele kilofów Źródło: Opracowanie własne .....	17
8	Modele siekier Źródło: Opracowanie własne .....	18
9	Model bazy Źródło: Opracowanie własne .....	18
10	Model pieca Źródło: <a href="https://sketchfab.com/3d-models/stylized-furnace-e7e9c0fa89724f8dba106f96bb4a9c8b">https://sketchfab.com/3d-models/stylized-furnace-e7e9c0fa89724f8dba106f96bb4a9c8b</a> .....	19
11	Model skrzyni Źródło: <a href="https://sketchfab.com/3d-models/low-poly-chest-ede8d988e3724ed395bc20419040d2da">https://sketchfab.com/3d-models/low-poly-chest-ede8d988e3724ed395bc20419040d2da</a> .....	20
12	Przedstawienie rozgrywki w CryoFall Źródło: <a href="https://muve.pl/sklep/digital/gry-niezalezne/cryofall,1534471">https://muve.pl/sklep/digital/gry-niezalezne/cryofall,1534471</a> .....	21

13 Przedstawienie rozgrywki w Darkwood Źródło: <a href="https://store.steampowered.com/app/274520/Darkwood/?l=polish">https://store.steampowered.com/app/274520/Darkwood/?l=polish</a> . . . . .	23
14 Zrzut ekranu z trello . . . . .	27
15 Screenshot przedstawiający wykorzystanie systemu zarządzania wersją Git. Źródło: Opracowanie własne . . . . .	29
16 Screenshot przedstawiający wykorzystanie systemu zarządzania wersją Git. Źródło: Opracowanie własne . . . . .	29
17 Ruch postaci względem kamery: Opracowanie własne . . . . .	30
18 Ostateczna prędkość poruszania się postaci: Opracowanie własne . . . . .	31
19 Podnoszenie przedmiotów: Opracowanie własne . . . . .	32
20 Wydobywanie zasobów: Opracowanie własne . . . . .	33
21 Efekt częsteszkowy podczas wydobywania zasobów: Opracowanie własne . . . . .	34
22 Tworzenie przedmiotów: Opracowanie własne . . . . .	35
23 Sprawdzenie dostępności przedmiotów w otoczeniu gracza: Opracowanie własne . . . . .	36
24 Tworzenie przedmiotów: Opracowanie własne . . . . .	37
25 Screenshot przedstawiający UI gry ekwipunku gracza i skrzyni we wczesnej fazie projektu. Źródło: Opracowanie własne . . . . .	38
26 Screenshot przedstawiający UI pieca. Źródło: Opracowanie własne . . . . .	39
27 Przetapianie zasobów: Opracowanie własne . . . . .	40
28 Sprawdzanie dostępnych rozdzielczości ekranu . . . . .	41
29 Odczyt i zapis ustawień: <a href="https://docs.unity3d.com/ScriptReference/PlayerPrefs.html">https://docs.unity3d.com/ScriptReference/PlayerPrefs.html</a> . . . . .	42
30 Zakładka zmiany ustawień graficznych . . . . .	42
31 Zakładka zmiany ustawień dźwięku . . . . .	42
32 Zmiana pogody: Opracowanie własne . . . . .	43
33 Pozycjonowanie deszczu nad graczem: Opracowanie własne . . . . .	43
34 Screenshot scenariusza testowego . . . . .	45