



UNIWERSYTET
IM. ADAMA MICKIEWICZA
w POZNANIU

Wydział Matematyki i Informatyki

Kamil Tyrek, Mateusz Hypś, Jakub Kozubal
Numer albumu: 434797, 434699, 434726

Projekt i implementacja gry
„The Lore: Story of the fallen warrior”

Project and implementation of game „The Lore: Story of the fallen warrior”

Praca inżynierska na kierunku **informatyka**
napisana pod opieką
dra Bartłomieja Przybylskiego

Poznań, luty 2021

Poznań, 13 lutego 2021 r.

Oświadczenie

Ja, niżej podpisany **Kamil Tyrek**, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt. *Projekt i implementacja gry*

„*The Lore: Story of the fallen warrior*” napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

- [TAK] wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM
- [TAK] wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

Poznań, 13 lutego 2021 r.

Oświadczenie

Ja, niżej podpisany **Mateusz Hypś**, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt. *Projekt i implementacja gry*

„*The Lore: Story of the fallen warrior*” napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

- [TAK] wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM
- [TAK] wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

Poznań, 13 lutego 2021 r.

Oświadczenie

Ja, niżej podpisany **Jakub Kozubal**, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt. *Projekt i implementacja gry*

„*The Lore: Story of the fallen warrior*” napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

- [TAK] wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM
- [TAK] wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

Streszczenie

Przedmiotem pracy jest przedstawienie projektowania gry platformowej „The Lore: story of the fallen warrior” w oparciu o silnik Unity. Gra posiada elementy zręcznościowe oraz zagadki logiczne, przez co przechodzenie kolejnych etapów gry wymaga szybkiego podejmowania logicznych decyzji.

Na początku zostanie przedstawiony zamysł projektu i w jaki sposób można nim zarządzać. Zaprezentowany będzie fundamentalizm pracy nad projektem, co jest ważne, czego należy przestrzegać i na czym się skupić. W jaki sposób zostaje podzielony projekt informatyczny oraz z czym wiążą się poszczególne etapy pracy. Następnie przedstawione zostaną metodyki, które od lat służą za zbiór drogowskazów w procesie realizacji projektu. Przedstawione zostanie również wykorzystanie tego w projekcie gry komputerowej „The Lore”.

W trzecim rozdziale będziemy mogli dowiedzieć się czym jest zagadka logiczna oraz poznać kilka przykładów. Opisany zostanie sposób obsadzenia mini-gier w grze opartej na silniku Unity. W kontekście projektu „The Lore”, przedstawione są algorytmy stojące za logiką zaimplementowanych mini-gier. Dodatkowo przedstawiono metodę projektowania mini-gry nie zawartej w projekcie - algorytmu generującego labirynt.

W ostatnim rozdziale przedstawiony zostanie opis procesu projektowania i tworzenia postaci oraz mechanik z nią związanych. Elementy te zostaną przedstawione począwszy od projektowania graficznego po tworzenie dodatkowych funkcjonalności takich jak drzewko umiejętności. Przedstawione zostaną również różnice między silnikiem fizyki w grze, a fizyką w prawdziwym świecie, problemy i rozwiązania fizyki oddziałującej na postać zarówno jeśli chodzi o sam ruch jak i o obiekty w terenie, które na zachowanie tej postaci wpływają.

Spis treści

Rozdział 1. Wstęp	8
1.1. Cel i założenia projektu	8
1.2. Organizacja pracy	9
1.3. Podział prac	9
1.3.1. Kamil Tyrek - tworzenie elementów logicznych i ich algorytmika	9
1.3.2. Mateusz Hypś - Zarządzanie projektem gry komputerowej z wykorzystaniem Agile	10
1.3.3. Jakub Kozubal - Fizyka postaci	10
1.4. Użyta technologia - Unity	10
Rozdział 2. Zarządzanie projektem gry komputerowej z wykorzystaniem Agile	12
2.1. Czym jest projekt?	12
2.1.1. Zakres	13
2.1.2. Zasoby	13
2.1.3. Czas	14
2.1.4. Pieniądze	15
2.2. Czym jest zarządzanie projektem?	16
2.3. Kim jest menedżer projektu?	17
2.4. Cykl życia projektu informatycznego	18
2.4.1. Faza planowania	19
2.4.2. Faza analizy	20
2.4.3. Faza projektowania	20
2.4.4. Faza implementacji	21
2.4.5. Faza testowania	21
2.5. Metodyki w projekcie	24
2.5.1. Podejście klasyczne a zwinne	25
2.5.2. Metoda waterfall	26
2.5.3. Metody zwinne czyli Agile	27
2.6. Wykorzystanie Agile w tworzeniu gry	30
2.6.1. Agile Game Development	30
2.6.2. Cykl życia projektu „The Lore”	34
2.7. Podsumowanie	43
Rozdział 3. Zagadki logiczne	44

3.1. Wprowadzenie do zagadek logicznych	44
3.1.1. Omówienie zagadnienia	44
3.1.2. Przykłady logicznych zagadek w prawdziwym świecie	44
3.1.3. Wdrożenie zagadek do gry opartej na silniku Unity	46
3.2. Przesuwane puzzle	48
3.2.1. Omówienie zagadnienia i ogólne założenia	48
3.2.2. Algorytmika	48
3.2.3. Przedstawienie przykładu w grze „The Lore”	52
3.2.4. Przedstawienie przykładu w innych grach	54
3.3. Zagadka z rurami	56
3.3.1. Omówienie zagadnienia	56
3.3.2. Algorytmika	56
3.3.3. Przedstawienie przykładu w grze „The Lore”	58
3.3.4. Przedstawienie przykładu w innych grach	60
3.4. Zagadka z otwieraniem zamku	61
3.4.1. Omówienie zagadnienia	61
3.4.2. Algorytmika	61
3.4.3. Przedstawienie przykładu w grze „The Lore”	63
3.4.4. Przedstawienie przykładu w innych grach	64
3.5. Labirynt	66
3.5.1. Omówienie zagadnienia	66
3.5.2. Algorytmika	66
3.5.3. Przedstawienie przykładu w innych grach	70
3.6. Podsumowanie	71
Rozdział 4. Fizyka postaci	72
4.1. Tworzenie postaci	72
4.1.1. Omówienie zagadnienia procesu tworzenia postaci	72
4.1.2. Proces graficzny tworzenia postaci	72
4.1.3. Wdrożenie postaci do projektu w Unity	74
4.1.4. Sposoby animowania postaci	74
4.2. Poruszanie się	77
4.2.1. Omówienie zagadnienia	77
4.2.2. Fizyka w grach 2D, a w prawdziwym świecie	77
4.2.3. Poruszanie się oraz kolizje postaci	78
4.2.4. Skakanie	80
4.2.5. Otoczenie wpływające na fizykę postaci	82
4.3. Umiejętności postaci	84
4.3.1. Omówienie zagadnienia umiejętności	84
4.3.2. Drzewko umiejętności	85
4.3.3. Umiejętności	89
4.4. Podsumowanie	91
Bibliografia	92

ROZDZIAŁ 1

Wstęp

1.1. CEL I ZAŁOŻENIA PROJEKTU

Celem projektu było stworzenie gry platformowej, zawierającej elementy zręcznościowe oraz łamigłówki. Głównym założeniem projektu jest gra, która zaciekawi swoją fabułą oraz trudnością. Wstępnie prosty zamysł samouczka, w większości gier jest raczej elementem wprowadzającym do rozgrywki, która powinna robić się trudniejsza w im dalszym etapie rozgrywki się znajdujemy, w przypadku naszej gry jest odwrócony. Samouczek poza wprowadzeniem użytkownika do poszczególnych elementów rozgrywki, którymi są między innymi: przedstawienie sterowania, funkcjonalności oraz wstępnych mechanik jest również wymagający. To od gracza zależy jakie umiejętności w trakcie rozgrywki będzie rozwijać, aby przejść kolejnych poziomów było dla niego łatwiejsze. Dużą wagę w projekcie przywiązujemy do mini-gier, które występują w trakcie przechodzenia poszczególnych poziomów. Występują dwa rodzaje mini-gier: opcjonalne, czyli te które przejść możemy w celu sprawdzenia siebie oraz zdobycia punktów doświadczenia, a także wymagane, które trzeba przejść, aby znaleźć się na dalszym etapie gry. Użytkownik posiada do dyspozycji punkty doświadczenia oraz ekwipunek. Punktami doświadczenia możemy rozporządzać bezpośrednio w drzewku umiejętności, w którym gracz ma możliwość wybrania odpowiednich ścieżek, zależnie od tego jaką strategię rozgrywki chce przyjąć. Warto pamiętać jednak, że im głębiej będziemy rozwijać daną gałąź, tym umiejętności będą bardziej pomocne, co sprawia, że gracz musi zastanowić się dobrze nad rozporządzeniem punktami doświadczenia. Ekwipunek służy do zdobywania przedmiotów potrzebnych w trakcie rozgrywki, m.in. do otwierania drzwi czy rozpoczęcia opcjonalnej mini-gry. W projekcie są wykorzystane 2 rodzaje kamer: *orthographic* i *perspective*, które skutkują wykorzystaniem różnych stylów do tworzenia poziomów. Wszystkie animacje w projekcie są tworzone z

użyciem technologii kinematyki odwrotnej co pozwala na dodawanie nowych animacji i poprawianie już istniejących w trakcie rozwoju projektu.

1.2. ORGANIZACJA PRACY

Charakter projektu sprawił, iż nie można w naszym zespole jasno podzielić typów zadań, które zostały zrealizowane w ramach projektu. Wynikiem tego jest to, iż nowe funkcjonalności są realizowane zazwyczaj przez jedną osobę od początku do końca, nie ma podziału, podobnego jak w przypadku aplikacji z frontendem i backendem.

Do koordynacji projektem wykorzystano metodykę zwinną - Agile. Wynikało to z doświadczenia pracy części zespołu w tej metodyce oraz przejrzystość i rozsądne koordynowanie pracą. Przy zarządzaniu realizacji zadań wspomagał nas serwis JIRA, który pozwala na wizualizację zaplanowanych obowiązków w wybranym czasie. Takie plany nazywane są w strategii Scrum sprintami. W pierwszym semestrze dwutygodniowe, a w drugim semestrze tygodniowe. Kod źródłowy zarządzany jest poprzez GitHub. Dla przejrzystości pracy, każdy commit na GitHub oznaczany jest identyfikatorem zadania z Jiry, dzięki któremu wchodzący w zadanie widzimy commit powiązany z jego rozwiązaniem.

Z racji wybranej metodyki, dokonano również przydzielenia odpowiednich ról członkom zespołu. Podział ról wygląda następująco:

- Kamil Tyrek - Development Team, Scrum Master
- Jakub Kozubal - Development Team, Product Owner
- Mateusz Hypś - Development Team

1.3. PODZIAŁ PRACY

1.3.1. Kamil Tyrek - tworzenie elementów logicznych i ich algorytmika

W trzecim rozdziale zostaną przedstawione sposoby tworzenia elementów logicznych. Część z przedstawionych łamigłówek została użyta w projekcie końcowym. Przykładem są tutaj przesuwane puzzle - rozgrywka polegająca na przesunięciu elementu, celem ułożenia poprawnego obrazka. Algorytmika stojąca za losowaniem kolejności elementów nie jest trywialna, ponieważ źle wylosowana kolejność puzzli powoduje, iż mogą być niemożliwe do ułożenia.

Zostanie przedstawiona też logika mini-gry z ustawieniem odpowiednich ruchów, celem połączenia dwóch końców rur. Podobnie jak w poprzednim przykładzie, do rozwiązania tej zagadki potrzebna jest odpowiednia liczba rur danego typu - pionowe, poziome, skrętne.

Następnym przykładem będzie logika stojąca za rozgrywką, która nie jest dostępna w końcowym projekcie, a chodzi tutaj o generowanie labiryntu. Problemem może być wygenerowanie ścieżki, aby możliwe było przejście z punktu A do punktu B. W tym rozdziale postaramy się przedstawić rozwiązanie tego problemu, przy użyciu odpowiednich algorytmów.

1.3.2. Mateusz Hypś - Zarządzanie projektem gry komputerowej z wykorzystaniem Agile

W rozdziale zostanie opisany proces zarządzania projektem z wykorzystaniem metodyki zwinnej. Omówione i porównane zostaną podejścia Agile i Agile Game Development oraz zastosowania tych metodyk, w porównaniu z innymi stosowanymi w praktyce. Przedstawione zostaną najważniejszych idee stojące za metodami Agile m.in. framework SCRUM. Następnie omówione zostanie zastosowanie tych metod w projekcie „The Lore”, z uwzględnieniem problemów w trakcie realizacji tego projektu.

1.3.3. Jakub Kozubal - Fizyka postaci

W tym rozdziale zostaną przedstawione sposoby tworzenia poruszania się postaci. Zaznaczone zostaną główne różnice między fizyką rzeczywistą, a tą stosowaną w grach jak i przedstawienie wielu sposobów rozwiązania tej samej funkcjonalności. Ponadto przedstawiony zostanie proces tworzenia postaci. Do tego pojawi się też opisanie problemów takich jak sterowanie postaci w powietrzu oraz oddziaływanie sił z otoczenia (m.in. poruszające się platformy). Zostanie także przedstawione i przeanalizowane działanie każdej umiejętności występującej w drzewku odpowiedzialnej za poruszanie się.

1.4. UŻYTA TECHNOLOGIA - UNITY

W naszym projekcie postanowiliśmy wybrać UNITY jako środowisko do stworzenia naszej gry. Wynikało to z możliwości jakie oferuje oraz z jakości i czytelności, stworzonej przez twórców, oficjalnej dokumentacji. Pozwala nam na tworzenie gier dwuwymiarowych czy trójwymiarowych oraz interaktywnych materiałów,

na przykład animacje czy wizualizacje. W razie problemów możemy również wykorzystywać oficjalne forum na którym rzesza użytkowników dzieli się swoimi wskazówkami a także oficjalny sklep – Asset Store, w którym możemy wykupić materiały potrzebne do naszej gry. UNITY działa na każdym systemie operacyjnym, tj. Windows, macOS oraz Linux. Gwarantuje nam również możliwość stworzenia aplikacji nie tylko na komputery osobiste, ale także przeglądarki internetowe, konsole gier video oraz urządzenia mobilne. Dzięki aktualizacji silnika do wersji 5.1.1 ta lista wzrasta do 22 platform sprzętowych, w tym gogle wirtualnej rzeczywistości takie jak Oculus Rift. W przeszłości można było tworzyć aplikacje w trzech językach:

- UnityScript (swego rodzaju pochodna JavaScript'u)
- C#
- Boo

Jednak wraz z piątą wersją silnika (wydaną w roku 2015) możliwość pisania w języku Boo została usunięta, pozostała tylko wsteczna kompatybilność w postaci możliwości komplikacji skryptów przez środowisko MonoDevelop. Podobny los dotknął UnityScript, którego wsparcie zakończyło się na wersji 2018.2 (najnowsza wersja stabilna to 2019.3.4). Z tych względów nasz wybór musiał paść na język C#, w którym zostały napisane wszystkie nasze skrypty. Jako jedyny jest wciąż wspierany przez autorów, co zaowocowało drastycznym wzrostem popularności wśród użytkowników.[1]

ROZDZIAŁ 2

Zarządzanie projektem gry komputerowej z wykorzystaniem Agile

2.1. CZYM JEST PROJEKT?

Aby odpowiednio przyswoić temat zarządzania projektem, należy zacząć od zrozumienia paru terminów, które są nieodłączną jego częścią.

Projekt jest to termin z pozoru banalny, w końcu spotykamy się z nim wielokrotnie, jednak wy tłumaczenie go może sprawiać problemy. Jedną z najlepszych oraz najbardziej wyczerpujących definicji na jaką można trafić, jest ta stworzona przez Roberta K. Wysockiego oraz Rudd'a McGary'ego. Są to Amerykańscy specjalści w temacie zarządzania projektami. W ich książce pod tytułem „Efektywne zarządzanie projektami” definiują projekt jako: „sekwencję niepowtarzalnych, złożonych i związanych ze sobą zadań, mających wspólny cel, przeznaczonych do wykonania w określonym terminie bez przekraczania ustalonego budżetu, zgodnie z założonymi wymaganiami”. [2]

Pomimo faktu, iż powyższa definicja jest najprawdopodobniej znacznie bardziej złożona od takiej, którą słyszymy na co dzień to jest ona bardzo cenna, ponieważ przedstawia najważniejsze cechy każdego projektu.[3]

Składa się z czterech podstawowych elementów, którymi menedżer projektu musi zarządzać symultanicznie. Trzeba pamiętać, że są one ze sobą powiązane[4]. Do wspomnianych elementów należą:

- Zakres
- Zasoby
- Czas
- Pieniądze

2.1.1. Zakres

Zakres jest zdecydowanie najważniejszym z podanych elementów. Jest on definicja, która opisuje co tak naprawdę projekt ma osiągnąć i co ma w nim zostać zrealizowane. Obrazuje rozmiar projektu, jego cele, a także wymagania. Zakres jest nie tylko najważniejszym, ale również najbardziej złożonym elementem. Jakakolwiek zmiana musi zostać odwzorowana w pozostałych trzech elementach. Jest to jeden z powodów, dla którego mówi się o współzależności między tą czwórką. Aby to lepiej zrozumieć, można sobie wyobrazić aplikację, która ma posiadać cztery główne funkcjonalności, zbudowana przez dwa zespoły z budżetem 40 tysięcy złotych. Jeśli zakres projektu zmieni się do przykładowo sześciu funkcjonalności, zadaniem menedżera projektu jest dostosowanie zasobów ludzkich, czasu oraz budżetu, w taki sposób aby cel ten został zrealizowany. [5]

2.1.2. Zasoby

Zasoby dzielą się na trzy kategorie:

- Zasoby ludzkie
- Wyposażenie
- Materiały

Zasoby ludzkie

Menadżer projektu musi upewnić się, że pracownicy posiadają odpowiednie umiejętności i narzędzia aby ukończyć dane im zadanie. Musi w pełni monitorować czy ma wystarczającą ilość zasobów ludzkich do konkretnego projektu tak aby go ukończyli w ustalonym czasie. Jego zadaniem jest również dopilnowanie aby każda osoba przypisana do zadania doskonale wiedziała i rozumiała co ma zrobić oraz znała wyznaczone terminy. W większych firmach wygląda to inaczej. Pracownicy są podzieleni na grupy, którymi zarządza *team leader*, jest on odpowiedzialny za większość obowiązków, które zostały wymienione powyżej. Wynika to z faktu, iż menedżer projektu nie jest w stanie zarządzać tak dużą grupą ludzi. Dużym atutem jest również to, że *team leader* najlepiej zna swój zespół, ich umiejętności, możliwości, a także czas, którym dysponują. Jest to również duże ułatwienie dla menedżera projektów, ponieważ nie musi komunikować się ani nadzorować wszystkich, wystarczy kontakt z *team leaderem* konkretnych zespołów.

Wyposażenie i materiały

Czasami dochodzi do sytuacji, w której project manager jest odpowiedzialny za pozyskiwanie materiałów i wyposażenie, którymi musi zarządzać w taki sposób aby zespół wykonywał swoją pracę w najefektywniejszy sposób. Nie jest to często spotykane zjawisko, w szczególności w dużych firmach w których podział obowiązków jest mocniej rozdzielony po wielu pracownikach.

2.1.3. Czas

Podobnie jak w przypadku zasobów, czas jest dzielony na trzy grupy

- Podział na zadania
- Harmonogram
- Ścieżka krytyczna (ang. Critical path)

Podział na zadania

Podział na zadania jest pierwszym z trzech kroków do pomyślnego zarządzania czasem. Zadania muszą być tworzone w przemyślany sposób, dobrze przeanalizowane, a także odpowiednio wytlumaczone, tak aby pracownik, który się go podejmie wszystko zrozumiał za pierwszym razem. Niespełnienie przynajmniej jednego z wyżej wymienionych warunków tworzenia zadań negatywnie wpływa na ich wykonanie. W wielu przypadkach łączy się to z opóźnieniami w realizacji projektu.

Harmonogram

Po pomyślnym stworzeniu zadań przechodzi się do zaplanowania harmonogramu. Tworzy się go poprzez ustalenie w odpowiedniej kolejności wszystkich zadań, które muszą zostać wykonane. Niektóre z nich można wykonywać sekwencyjnie, inne z nich mogą nakładać się na siebie, a jeszcze inne mogą zostać wykonane jednocześnie. Kluczem do sukcesu jest ich zrozumienie i poprawne grupowanie. Następnym ważnym czynnikiem jest zrozumienie zależności między nimi, ponieważ niektóre z zadań muszą zostać wykonane w pierwszej kolejności. Ostatnim krokiem tworzenia harmonogramu jest estymacja czasu potrzebnego do ich wykonania, a także przypisanie im odpowiednich zasobów.

Ścieżka krytyczna (ang. Critical path)

Niektóre zadania mają elastyczny termin rozpoczęcia oraz ich zakończenia. Jednak, istnieją również takie które tej elastyczności nie mają. Linia przechodząca przez zbiór takich zadań nazywana jest ścieżką krytyczną i wykorzystywana jest do monitorowania w jakim tempie zostają wykonywane zadania w projek-

cie. W zależności od podziału zadań, istnieje możliwość występowania wielu ścieżek krytycznych. Wszystkie zadania, które znajdują się na ich drodze muszą zostać wykonane w terminie, w przeciwnym razie występuje bardzo wysokie prawdopodobieństwo, że projekt nie zostanie ukończony na czas.

2.1.4. Pieniądze

Dla najefektywniejszego zarządzania kosztami projektu uwzględnia się jego:

- Koszty
- Wydatki związane z losowymi zdarzeniami
- Zyski

Koszty

Każde zadanie, ma określony koszt potrzebny do jego wykonania, najczęściej bazuje na wydatkach związanych z potrzebnymi zasobami. Każdy z tych wydatków jest estymowany i uwzględniany podczas przygotowania budżetu dla projektu.

Wydatki związane z losowymi zdarzeniami

Podobnie jak w estymacji czasu potrzebnego do wykonania konkretnych zadań, tak samo przy przygotowywaniu budżetu, należy uwzględnić pewien bufor. Zostanie on wykorzystany na wypadek losowych zdarzeń, które mogą się w nieoczekiwany momencie wydarzyć. Najprostszym przykładem, w firmie informatycznej może być problem techniczny związany ze sprzętem np. zepsuty komputer.

Zyski

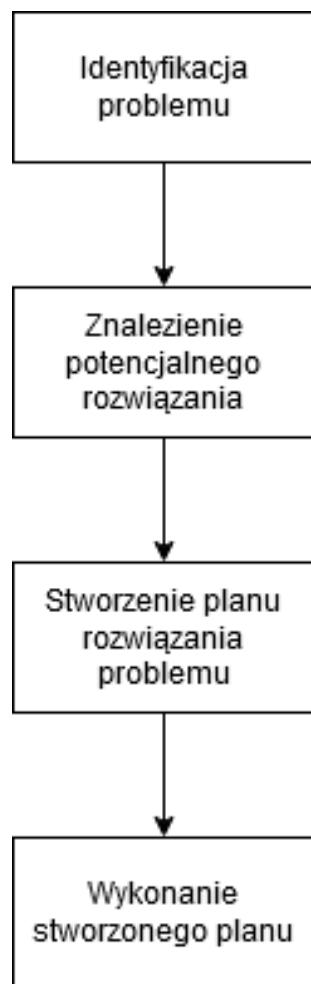
Zyski są to pieniądze, które firma planuje zarobić na wykonanym projekcie, bądź po każdym zakończonym zadaniu. Oczywiście aby projekt został uznany za opłacalny dla biznesu, budżet, który zawiera odpowiednio oszacowane koszty nie może przekraczać pewnego procentu planowanych zysków. Zadaniem menedżera projektu jest oczywiście zminimalizowanie jak najbardziej kosztów produkcji i jak największe zmaksymalizowanie zysku, które firma zarobi po wykonanym projekcie.

2.2. CZYM JEST ZARZĄDZANIE PROJEKTEM?

Wiedząc już dokładnie czym jest projekt, zrozumienie na czym polega zarządzanie nim staje się trywialne. W dużym uproszczeniu jest to proces, który za pomocą sprecyzowanego planu pozwala na osiągnięcie wyznaczonego przez ciebie lub twoją firmę celu. Stworzenie odpowiedniego planu, podzielonego na szereg kroków, jest tutaj znaczące, ponieważ do osiągnięcia wyznaczonego celu, wymagane jest ukończenie wszystkich zadań w odpowiedniej kolejności. Bardzo często, zarządzanie projektem jest porównywane do wchodzenia po schodach, bądź wspinania się po drabinie. Aby osiągnąć wyznaczony cel i dojść na samą górę, trzeba krok po kroku zaliczać każdy poziom. Podobnym trafnym porównaniem, jest podróż z jednego miejsca do drugiego. Należy rozpatrzyć różne trasy w celu zrozumienia, która z nich będzie najlepsza. Następnie wyestymować czas, potrzebny na podróż, przemyśleć sposoby w jaki się tam dostaniesz, czy to pieszo czy za pomocą pojazdu. Przygotowanie potrzebnego budżetu oraz przeanalizowanie potencjalnych zagrożeń, może również wpływać na końcowy rezultat. Jak widać, zarządzanie projektem nie występuje tylko w biznesie, jest to szeroko pojęte zdarzenie, którego doświadczamy każdego dnia. Jednak nawet na tak prostym przykładzie, można zrozumieć jak ważny jest to aspekt naszego życia, a także jak kluczowe ma efekty w biznesie.[6]

Zarządzanie projektem nie składa się tylko z planowania i pilnowania czy wszystko idzie zgodnie z założeniami. Jest to również dziedzina która ma na celu budowanie motywacji zespołu projektowego, zadbanie o właściwą komunikację między jego stronami, a także zrozumieniu ich potrzeb. W zależności od rozmiaru firmy proces taki potrafi być bardzo trudny.

Innym czynnikiem wchodząącym w skład zarządzania projektem jest analiza zagrożeń oraz praktyczna wiedza pozbywania się ich. Jest to tak zwana eliminacja ryzyka występująca podczas całego cyklu życia projektu. Ryzyko w projektach pochodzi głównie z niemożliwości wyeliminowania niechcianych incydentów oraz niepewności związanej z przyszłością. Do tego typu zdarzeń dochodzi na każdym kroku projektu, wynika to z dynamiki procesu, potencjalnych konfliktów między pracownikami, niespodziewanymi komplikacjami związanymi z zasobami, zmiennej wydajności pracy czy zwyczajnie błędnego planowania. Aby jak najbardziej ograniczyć ryzyko przy jednoczesnej maksymalizacji optymalizacji użycia zasobów zatrudnia się odpowiednią na to miejsce osobę, zwaną menadżerem projektu lub kierownikiem projektu.



Rysunek 2.1. Uproszczony proces realizacji projektu

Powyższy diagram obrazuje proces realizacji projektu w bardzo uproszczeniu formacie, ponieważ każdy etap kryje za sobą wiele procesów, które trzeba dodatkowo wykonać.

2.3. KIM JEST MENEDŻER PROJEKTU?

Menedżer projektu zwany również kierownikiem projektu czy PM'em od angielskiej nazwy project manager. Jest to specjalista, którego główną odpowiedzialnością jest rozpoczęcie cyklu życia projektu. Zaczyna od przygotowania planu działania a następnie wdrożenia go i zrealizowania aby następnie móc

zakończyć projekt z sukcesem. Jego podstawowym zadaniem jest zapewnienie wykonania wcześniej zaplanowanych celów, po to aby wydać produkt docelowy spełniający wszelkie wymagania. Kluczowymi obowiązkami są:

- wstępne zaplanowanie projektu, a następnie jego ustandaryzowanie
- wprowadzenie kluczowych zasad działania
- opracowanie logicznych i możliwych do zrealizowania celów
- odpowiednie rozporządzenie czasem oraz kosztami

W niektórych przypadkach zajmuje się również komunikacją z klientem, aby następnie przedstawić oraz wdrożyć wszelkie jego wymagania do życia projektu. Poza najważniejszymi obowiązkami, dobry kierownik projektu powinien również zadbać o dobrą organizację pracy oraz podkreślać jak ważna jest komunikacja pomiędzy członkami zespołów. Musi rozumieć, że należy skupić się na szukaniu rozwiązań problemów zamiast winnych, a także pozostać opanowanym nawet w najbardziej stresujących sytuacjach.

2.4. CYKL ŻYCIA PROJEKTU INFORMATYCZNEGO

Cykl życia projektu to podział na fazy przez które przechodzi nasz projekt od początku do końca. Zawiera on wszystkie operacje, zadania oraz kroki które konkretny projekt musi doświadczyć. Zaletami korzystania z podziału na fazy cyklu życia projektu jest jego efektywna organizacja oraz możliwość ustandaryzowania procesu systematyzacji i implementacji projektu. Pozwala na radzenie sobie ze złożonością projektów informatycznych, poprzez tworzenie i weryfikowanie modeli dziedziny aplikacyjnej i samego systemu informatycznego.[7] Cykl życia projektu informatycznego nie zawsze wygląda tak samo, może się różnić od siebie ilością faz i ich nazewnictwem. Najczęściej spotykanym podziałem jest zestawienie na 4 - 5 faz.[8] Są to:

- Faza planowania
- Faza analizy
- Faza projektowania
- Faza implementacji
- Faza testowania

2.4.1. Faza planowania

Fazę planowania nazywamy pierwszą fazą procesu tworzenia oprogramowania. Jest ona kluczowa, ponieważ należy się w niej dowiedzieć w jakim celu oprogramowanie ma powstać oraz co ważniejsze, jak ma powstać. Już na tym etapie powinny być znane wyniki analizy biznesowej oraz znajomość decyzji czy oprogramowanie może powstać. Ustala się w niej jakie wartości dla biznesu ma projekt oraz czy w ogóle jest wart podjęcia się tworzenia. Analizuje się czy zgromadzone zasoby ludzkie są w stanie go wykonać oraz, co często może wydawać się oczywiste, chociaż w istocie takie nie jest, czy końcowy produkt będzie możliwy do wdrożenia.

Na tym etapie są również zbierane informacje oraz wymagania od klienta aby programiści byli w stanie zdefiniować przeznaczenie systemu. Dochodzi do procesu tworzenia opisu systemu w kategoriach aktorów i przypadków użycia. Aktorzy są to byty zewnętrzne, których zadaniem jest interakcja z systemem, natomiast przypadkami użycia nazywamy ogólne sekwencje zdarzeń odzwierciedlające możliwe sytuacje pomiędzy aktorami a systemem, bazując na aspekcie jego funkcjonalności.

Tabela 1.4. Przykładowy przypadek użycia — PurchaseOneWayTicket

<i>Nazwa przypadku testowego</i>	<i>PurchaseOneWayTicket</i>
<i>Aktor uczestniczący</i>	<i>Traveler</i>
<i>Sekwencja zdarzeń</i>	<ol style="list-style-type: none"> 1. <i>Traveler</i> wybiera strefę, w której zlokalizowana jest stacja docelowa 2. <i>TicketDistributor</i> wyświetla cenę biletu. 3. <i>Traveler</i> wprowadza do automatu środki płatnicze w wysokości nie niższej niż cena biletu. 4. <i>TicketDistributor</i> drukuje żądanego bilet i wydaje ewentualną resztę.
<i>Warunek wstępny</i>	<i>Traveler</i> staje przedautomatem zlokalizowanym na stacji początkowej lub dowolnej stacji pośredniej.
<i>Warunek końcowy</i>	<i>Traveler</i> jest w posiadaniu żądanego biletu i zwrócone nadpłaty.
<i>Wymagania jakościowe</i>	Jeżeli transakcja nie została ukończona, a bezczynność <i>Travelera</i> trwa dłużej niż minutę, <i>TicketDistributor</i> zwraca ewentualną gotówkę wprowadzoną już przez <i>Travelera</i> w ramach bieżącej transakcji.

Rysunek 2.2. Źródło: Inżynieria oprogramowania w ujęciu obiektowym. UML, wzorce projektowe i Java, Bernd Bruegge, Allen H. Dutoit

2.4.2. Faza analizy

Kolejną fazą jest faza analizy, w której przede wszystkim ustala się kto będzie używał tworzonego systemu, co dany system ma robić oraz kiedy i gdzie będzie używany. Na tym etapie powstaje model, który jest wynikiem zgromadzenia wymagań klienta jak i również wyciągniętych wniosków z przeprowadzonych przypadków użycia (z ang. Use Cases). Po zakończeniu tej fazy, otrzymanym rezultatem jest model systemu wzbogacony o atrybuty operacji i skojarzenia, który jest spójny i wolny od niejednoznaczności. Ewentualne jego przejawy bądź niespójności modelu są omawiane podczas negocjacji z klientem.

2.4.3. Faza projektowania

W porównaniu do dwóch poprzednich etapów, w którym główny nacisk był nakładany na ideę stojącą za stworzeniem konkretnego projektu, tak w tym pojawia się po raz pierwszy kreowanie w jaki sposób ma to być zrealizowane. Tutaj powstają wstępne koncepcje wewnętrznej logiki działania projektu. Dochodzi do projektowania interfejsów między innymi interfejsu użytkownika, także dyskutowana jest architektura systemu. Zdarza się, że dochodzi do podziału na dwie mniejsze fazy: faza projektowania systemu oraz projektowanie obiektów:

- wybór platformy sprzętowej oraz systemu operacyjnego środowiska docelowego
- selekcja technologii użytych i sposobu przechowywania danych tworzonego oprogramowania
- podejmowane są również decyzje w sprawie globalnej kontroli przepływu sterowania oraz reguł polityki dostępu

Podobnie jak w poprzedniej fazie czyli fazie analizy rezultatem otrzymanym na koniec jest klarowny opis i reprezentacja budowanego systemu w postaci na przykład diagramu. Główną różnicą jest fakt, że otrzymane modele systemu nie są zrozumiałe dla klienta tak jak to było w fazie analizy. Są one bowiem stworzone za pomocą raczej zaawansowanych i ulepszonych technik, które przekraczają możliwości zrozumienia i zainteresowania przeciętnego klienta.

Wspomniana wcześniej faza projektowania obiektów ma na celu zaspakajenie luk pomiędzy modelem zbudowanym podczas etapu analizy, a platformą sprzętową i programową zdefiniowaną na etapie projektowania systemu. Podczas tej fazy zostaje uszczegółowione oraz sprecyzowane opisanie poszczególnych

obiektów i podsystemów w celu osiągnięcia rozszerzalności, zrozumienia oraz poprawy wydajności tworzonego systemu. Rezultatem tego etapu jest szczegółowy model obiektowy, który posiada szeroko rozwinięte spektrum wyjaśniające wszystkie elementy.

2.4.4. Faza implementacji

Faza implementacji to w rzeczywistości wprowadzenie w życie wszystkich planów które zostały dotychczas stworzone. W żargonie programistycznym można powiedzieć, że jest to tłumaczenie modeli na kod źródłowy aplikacji. Oznacza to programowanie każdego obiektu zaprojektowanego w poprzednich etapach, a następnie zintegrowanie ich w jeden działający pojedynczy system. Rezultatem tego etapu jest zaskleplenie przerwy pomiędzy skomplikowanymi i uszczegółowionymi diagramami, planami oraz modelami a skończonym i kompilowanym kodem źródłowym.

2.4.5. Faza testowania

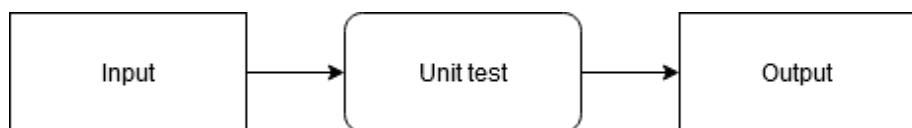
Ostatnią fazą jest faza testów która wykonywana jest już na stworzonym w poprzednim etapie działającym systemie. Polega na analizie założonych zachowań i funkcjonalności podczas projektowania a jego rzeczywistym odwzorowaniem w gotowym oprogramowaniu. To w niej wykonywana jest zdecydowana większość testów, jednak niektóre z nich były już obecne w poprzedniej fazie. Biorąc pod uwagę ilość różnych poziomów testów, poniższa grupa jest najpopularniejszą, jeśli chodzi o zdecydowaną większość projektów informatycznych. Testy dzielone są na:

- Testy jednostkowe
- Testy integracyjne
- Testy obciążeniowe
- Testy funkcjonalne
- Testy akceptacyjne

Testy jednostkowe

Testy jednostkowe to testy, których zadaniem jest sprawdzenie poprawności działania pojedynczego, wyizolowanego bytu. Są to najczęściej pojedyncze funk-

cje, jednostki bądź komponenty. Kluczową cechą tych testów jest brak integracji z API bądź bazą danych (jest to zadanie testów integracyjnych). Tego typu testy muszą działać bardzo szybko (liczone w setkach na sekundę), ponieważ odpalane są wiele razy przez programistów w fazie implementacji. Mają one nie tylko sprawdzać poprawność testowanego komponentu, ale przede wszystkim być dla deweloperów pewność siebie, że implementowane przez nich zmiany w kodzie nie doprowadzą do usterek funkcjonalnych.[9]



Rysunek 2.3. Uproszczony diagram działania testów jednostkowych

Testy integracyjne

Testy integracyjne są często łączone z testami jednostkowymi, ponieważ są one do siebie bardzo podobne. Diametralną różnicą jest fakt, że testy jednostkowe nie sprawdzają integralności między komponentami a API czy bazą danych, natomiast testy integracyjne już tak. Wykonywane są w celu wykrycia wad i defektów w interfejsach i komponentach oraz służą do testowania poprawnej interakcji między modułami lub systemami. Są one przeprowadzane w celu oceny zgodności systemu lub komponentów z określonymi wymaganiami funkcjonalnymi. Test integracyjny traktuje system jak czarną skrzynkę. Testują integralność bez zagłębiania się w szczegóły implementacji, co oznacza, że otrzymywana jest tylko informacja, jaka konkretna funkcjonalność powinna działać oraz co zostać zwrócone jako wynik. Weryfikowane jest czy otrzymany rezultat zgadza się z oczekiwany wynikiem bazującym na danych wejściowych. Podobnie jak testy jednostkowe, są tutaj sprawdzone zaimplementowane funkcje oraz poprawność interfejsów, które powinny zapewnić integralność systemu, mogą one również wskazać błędy w strukturach danych bądź problemy z dostępem do API lub bazy danych.[10]

Testy obciążeniowe

Testy obciążeniowe nazywane mogą być również testami wydajnościowymi ze względu na ich główny cel. Podczas ich wykonywania testowana jest próba obciążenia serwera, bazy danych oraz samego systemu naszej aplikacji, bazując na wcześniej przygotowanych scenariuszach użycia oraz w oparciu o

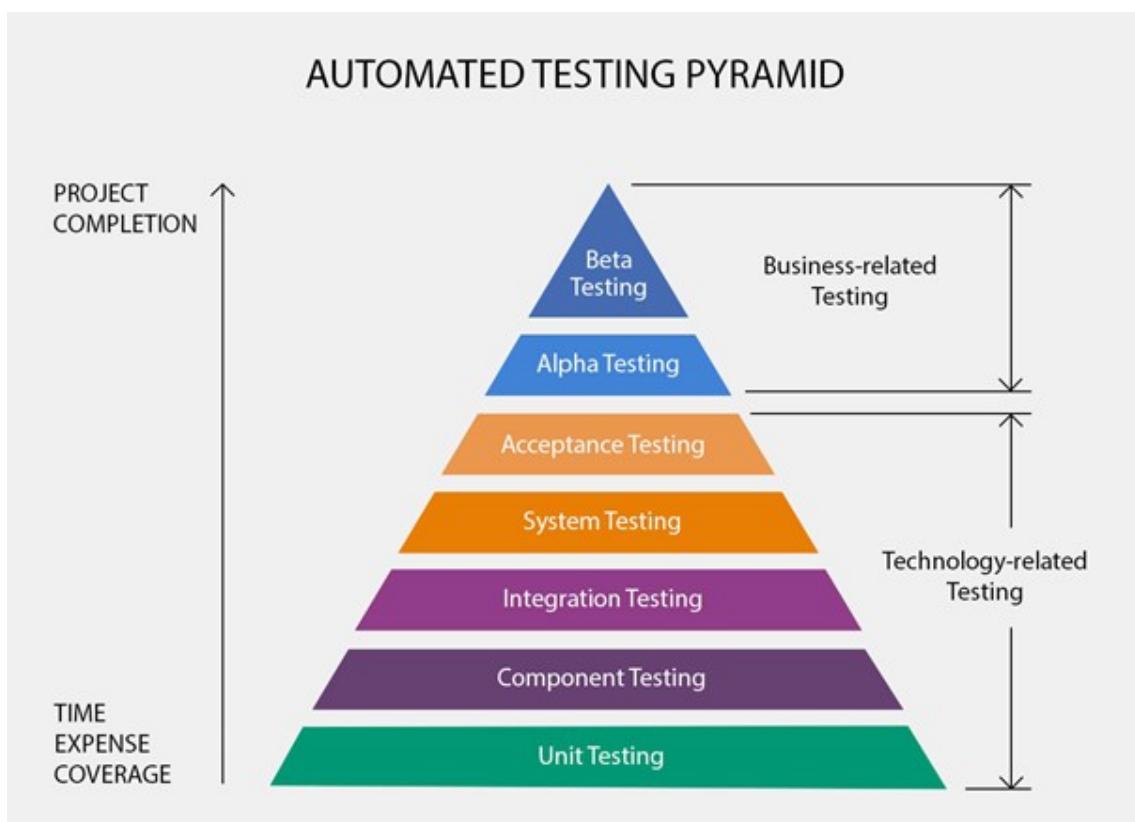
wygenerowanie wirtualnych użytkowników, których zadaniem jest wykonanie wcześniej stworzonych scenariuszy. Przykładowymi scenariuszami użycia są wszelkie działania, które może wykonać klient na przykład proces rejestracji czy logowania. Testy obciążeniowe mają za zadanie wskazać krytyczne punkty systemu, które w negatywny sposób wpływają na wydajność tworzonego systemu. Sprawdzają realny ruch na serwerze, który następnie pozwala na zdobycie punktu odniesienia oraz możliwości sprawdzenia czy nasz projekt spełnia wymogi wydajnościowe, brak wiedzy z tego zakresu może doprowadzić do braku akceptacji na wdrożenie.[11]

Testy funkcjonalne

Testy funkcjonalne są to testy które opierają się na analizie specyfiki funkcjonalnej tworzonego oprogramowania, systemu bądź modułu. Są one najczęściej wykonywane przez osoby które nie miały styczności z zaimplementowanym kodem. Testerzy mają za zadanie poddanie próbie stworzony system bez znajomości jego budowy. Głównym zamysłem nie jest przetestowanie konstrukcji bądź architektury lecz jego założenia funkcjonalne, to znaczy mają wykryć błędy zaimplementowanych funkcjonalności, które zostały zawarte w dokumentacji.[12]

Testy akceptacyjne

Testy akceptacyjne to inny gatunek testów, ponieważ nie skupiają się one na znalezieniu błędów czy pomyłek w implementacji systemu, mają one na celu potwierdzenie wykonania oprogramowania na odpowiednim poziomie. Służą do uzyskania formalnego potwierdzenia od klienta bądź osób zewnętrznych, że wykonany projekt spełnia kryteria jakościowe. Ważnym ich aspektem jest również weryfikacja czy nie brakuje założonych w dokumentacji funkcjonalności oraz czy procesy biznesowe przechodzą w prawidłowy sposób, taki aby zbudowały zaufanie wśród odbiorców. Następnym testowanym czynnikiem jest potwierdzenie czy aplikacja jest zgodna z dokumentacją. Testy akceptacyjne są ostatnim etapem weryfikowania poprawności zrealizowania systemu informatycznego, na jego podstawie menedżerowie odpowiedzialni za rozwój projektu podejmują decyzję o wdrożeniu do użytkowania produkcyjnego.[13]



Rysunek 2.4. Piramida procesu przeprowadzania testów

2.5. METODYKI W PROJEKCIE

Pojęciem metodyki nazywamy zbiór metod służących do zarządzania projektem. Mówiąc prościej są to metody dzięki którym jest możliwe zakończenie projektu z powodzeniem w wyznaczonym czasie. Korzystanie z metodyki pozwala na uzyskanie maksymalnej wydajności podczas tworzenia oprogramowania. Chociaż metodyk jest sporo i wybór zależy stricte od charakterystyki projektu to w ramach uproszczenia można je podzielić na grupy. Najogólniejszym i najpopularniejszym podziałem jest rozdzielenie na metodyki klasyczne (nazywane również tradycyjnymi) oraz zwinne. Do pierwszej grupy zaliczamy na przykład *waterfall*, natomiast do drugiej Agile.

2.5.1. Podejście klasyczne a zwinne

Aby umiejętnie wybrać odpowiednią metodykę do realizowanego projektu należy znać fundamentalne różnice między wcześniej wymienionymi grupami.

Metodyki klasyczne charakteryzują się wysokim stopniem planowania oraz uporządkowania. Projekty są podzielone na konkretne etapy a sam harmonogram prac oraz zakres i cel tworzonego projektu jest odgórnie znany. Natomiast metodyki zwinne uznawane są za zupełny kontrast metod tradycyjnych. Nie ma w nim konkretnego planu a praca przypomina swego rodzaju improwizację dostosowywaną do zmienności okoliczności oraz ich losowości. Na ich podstawie ukazywane są nieznane wcześniej zakres i cel projektu. Są to kluczowe różnice pomiędzy tymi metodykami, stąd często są porównywane jako uporządkowane kontra elastyczne.

Kolejnym faktorem jest rozmiar projektu, w metodykach klasycznych two-rzone są najczęściej duże i rozbudowane projekty. Wynika to z faktu, że wielkie przedsięwzięcia wymagają charakterystycznych dla podejścia tradycyjnych, organizacji oraz planowania. Tak jak zostało to wcześniej wspomniane sukcesywne realizacje projektu bazującym na harmonogramie oraz celach poznanych na początku projektu. W mniejszych projektach rozmiar jak i czas przedsięwzięcia są najczęściej diametralnie mniejsze i krótsze. Nie wymagają tak ścisłych reguł działania oraz konsekwencji a co za tym idzie metodyki zwinne doskonale do nich pasują.

Kontakt z klientem w metodyce klasycznej nie jest konieczny. Występuje on na początku w fazie zbierania informacji oraz specyfikacji wymagań do projektu. Jak również na samym końcu przy prezentacji projektu. W charakterze zwinnym podejście jest ponownie odmienne, kontakt z klientem jest ciągły, stąd też wspomniana wcześniej tendencja do zmian w trakcie projektu, bowiem klienci zgłaszają swoje opinie oraz zastrzeżenia w trakcie realizacji projektu, co często doprowadza do przekształcania zakresu projektu.

Zaangażowanie zasobów ludzkich w obu metodykach jest ponownie różne. W charakterystyce zwinnej nie ma ustalonej hierarchii pracowników, posiadają oni sporą autonomię a ich współpraca jest dostrzegalna na praktycznie wszystkich płaszczyznach. Sytuacja w metodyce tradycyjnej wprowadza hierarchię, w zespołach występuje osoba która nie wykonuje tych samych zadań co pozostali członkowie zespołu - lider. [14][15]

Poniższa tabela w skondensowany sposób przedstawia kontrast między dwoma metodykami.

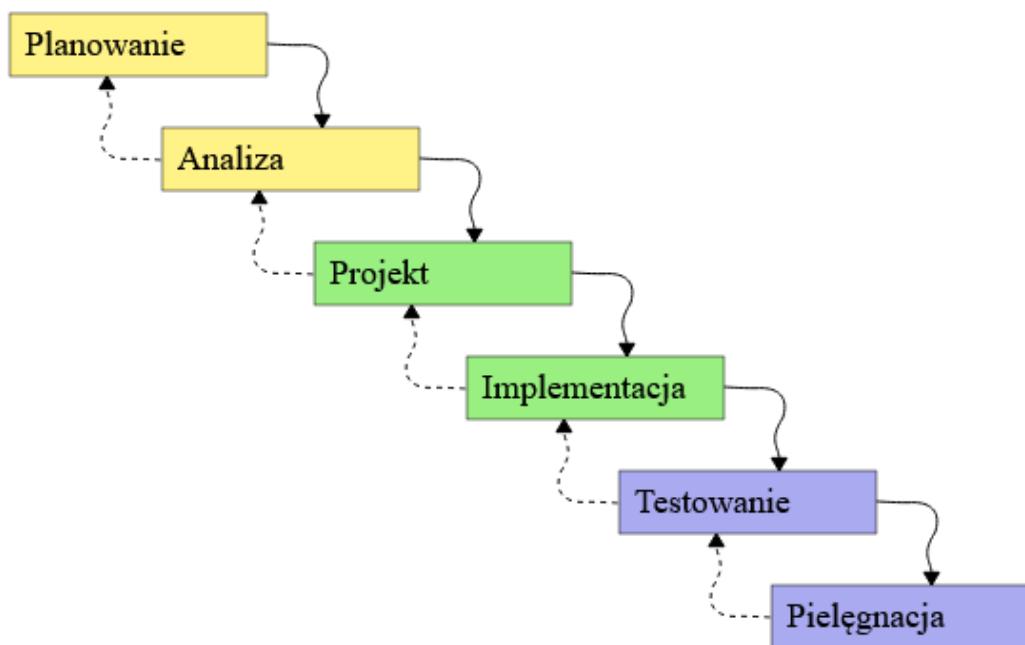
	Metodyki klasyczne	Metodyki zwinne
Oczekiwanie na produkt	Długi czas oczekiwania na gotowy produkt.	Małe iteracje pozwalające szybko uzyskać działający produkt.
Cel i zakres	Cel projektu jest znany i jasny, a jego zakres dokładnie uszczegółowiony w początkowych fazach projektu.	Polecane, gdy cel projektu nie jest do końca zdefiniowany. Kolejne elementy zakresu wybierane są i doszczegóławiane na początku kolejnych iteracji.
Wielkość projektu	Duży i bardzo duży.	Małe, innowacyjne, dobre dla startupów.
Zaangażowanie biznesu	Niezbędne w fazach początkowych (analiza, planowanie) oraz na końcu, przy testach i odbiorze produktu.	Reprezentant biznesu bierze czynny udział w pracach zespołu, określając i uszczegóławiając zakres.
Organizacja pracy	Zespoły biorące udział w poszczególnych etapach są rozłączne.	Istotą metodyk zwinnych jest bliska praca zespołów twórczych i biznesu, dlatego biznes jest członkiem zespołu. Wymagane jest duże zaangażowanie przez cały czas trwania projektu.
Zmiana	Mało odporne na zmiany zakresu i budżetu. Wymaga procedur zarządzania zmianą.	Odporne na zmianę. Polega na bieżącym reagowaniu na zmiany. Przed każdą kolejną iteracją zakres jest rewidowany.
Dokumentacja	Każdy etap dokładnie udokumentowany i zatwierdzony.	Dokumentacja na minimalnym poziomie niezbędnym do zapewnienia pracy zespołu.

Rysunek 2.5. Źródło: Narudo, Zwinne i tradycyjne zarządzanie projektem

2.5.2. Metoda waterfall

Metoda waterfall to przykład modelu kaskadowego, nazywanym również klasycznym, to jeden z wielu rodzajów procesów stworzenia oprogramowania. Polega na kaskadowym przechodzeniu pomiędzy ukończonymi wcześniej fazami. Zgodnie z wizją autora tej metodyki, o sukcesie projektu decydują dokładne zaplanowanie poszczególnych zadań oraz bezwzględne przestrzeganie terminów. Najważniejszym elementem tej metody jest szczegółowy plan oraz tworzona dokumentacja do każdej fazy projektu. Zakłada on sekwencyjną realizację ustal-

lonych etapów. Występują one kolejno po sobie i tworzą niezależne zadania. W wyniku braku możliwości przejścia do następnego etapu, przed zakończeniem poprzedniego występuje wysokie ryzyko pojawienia się błędów, które będą bardzo kosztowne w naprawie.[16]



Rysunek 2.6. Źródło: Wikipedia, Fazy modelu kaskadowego, Maciej Jaros -
Praca własna

2.5.3. Metody zwinne czyli Agile

Manifest Agile

Manifest agile wprowadzony w 2001 roku zaproponował i wprowadził nowe spojrzenie na możliwości wytwarzania oprogramowania. Przed wszystkim podkreślił wagę pracy zespołowej i interakcji międzyludzkiej nad dotychczasowej koncentracji na procesach i narzędziach. Zwrócił uwagę nad kontaktem z klientem oraz na tworzeniu roboczego oprogramowania zamiast wszechstronnej dokumentacji. Największym przeciwieństwem do dotychczasowych metod było zaakceptowanie powrotu do fazy implementacyjnej w celu zareagowania na

błędy i sugestie.[17][18]

Odkrywamy nowe metody programowania dzięki praktyce w programowaniu i wspieraniu w nim innych. W wyniku naszej pracy, zaczęliśmy bardziej cenić:

Ludzi i interakcje od procesów i narzędzi

Działające oprogramowanie od szczegółowej dokumentacji

Współpracę z klientem od negocjacji umów

Reagowanie na zmiany od realizacji założonego planu.

Oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej.

Rysunek 2.7. Źródło: Agile247, Manifest zwinnego programowania

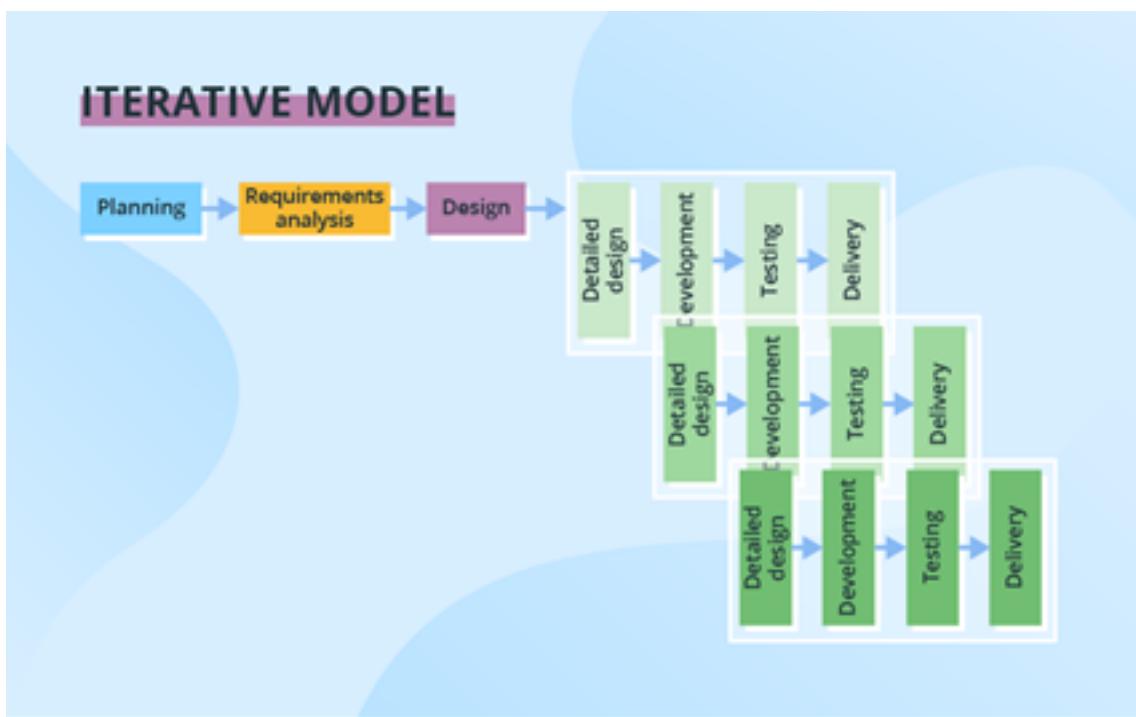
Generalnym założeniem stało się restrykcyjne zarządzanie procesem wytwarzania oprogramowania. Zakładana jest regularna kontrola wymagań jak i rozwiązań wraz z procesami adaptacji projektowych. Dużą wagę przywiązuje się do bezpośredniej komunikacji pomiędzy osobami z zespołu oraz codziennego kontaktu między sobą. W wyniku braku zastosowania jakiekolwiek hierarchii organizacyjnej, na barkach członków zespołu spoczywa większy ciężar dbania o systematykę i organizację pracy. Odpowiedzialność ta spoczywa bezpośrednio na osobach realizujących poszczególne zadania.

Czym jest model iteracyjny w Agile?

Podejście zwinne jest grupą metod, które bazują na tworzeniu oprogramowania opartym na stylu iteracyjno-przerostowym. Głównym jego założeniem jest tworzenie i wdrażanie systemu etapami. Pojedyncza iteracja kończy zrealizowany cykl życia projektu etapem testów. Wówczas zrealizowana część zostanie przekazana odpowiedniej grupie odpowiedzialnej ze przeprowadzenie sprawdzenia poprawności działania. Testowana jest również praktyczność zastosowania podczas realizacji faktycznych założeń biznesowych. Zdobywanie regularnej informacji zwrotnej pozwala na cykliczne korekty błędów, wprowadzanie zmian i ulepszeń funkcjonalności oraz precyzyjne reakcje na potrzeby klienta. Stąd, jak wspomniane wcześniej tendencję do ewolucji funkcjonalności w projekcie.

Pojedynczy cykl rozpoczyna się od fazy szczegółowego planowania zadań rozłożonych w konkretnym cyklu. Najczęściej przygotowywany jest przez szefów zespołów programistycznych lub kierowników projektowych. Następnym krokiem jest implementacja wcześniej przygotowanych założeń i funkcjonalności, po której następuje czas na testy i wdrożenie. To tutaj, docelowa grupa bada jakość przygotowanej części projektowej i decyduje o możliwości wdrożenia oraz przejścia do kolejnej iteracji.

Poniższy model obrazuje podejście iteracyjne, które jest charakterystyczne w metodyce Agile.



Rysunek 2.8. Źródło: ScienceSoft, Model iteracyjny [19]

2.6. WYKORZYSTANIE AGILE W TWORZENIU GRY

2.6.1. Agile Game Development

Agile Game Development to angielska nazwa na wykorzystanie podejścia zwanego w tworzeniu gry komputerowej. Samo podejście nie różni się niczym szczególnym od standardowego. Kluczowe jest natomiast pokazanie jak wiele możliwości wprowadza Agile do życia projektu gry. Głównym z nim jest wcześniej rozwinięty model iteracyjny. Dzięki niemu, części gry posiadające nowe funkcjonalności, znacznie częściej trafiają na testy do grupy docelowej. Takie podejście pozwoliło naprawić wiele błędów, których nie udało się wykryć w fazie implementacyjnej oraz poprawić działanie wielu funkcjonalności. Podczas realizacji zadań i rozwijania gry, programista może wpaść w pułapkę subiektywnej opinii dotyczącej zaimplementowanych rozwiązań. Jak to zwykle się zdarza bywają one błędne i posiadanie możliwości weryfikacji tego na tak wstępny etapie wprowadza ogromną wartość w projekcie. Pomimo przypisaniu takich elementów jak model iteracyjny czy częsty kontakt z klientem do podejścia zwanego, to w rzeczywistości są to nadzędne składowe metod, z których składa się metodyka Agile (przypominając metodyką to zbiór wcześniej założonych metod). W projekcie „The Lore” została wykorzystana fuzja trzech najpopularniejszych, ponieważ jest to najbardziej znana i wykorzystywana strategia w projektach komercyjnych.[20][21] Do tej grupy należą:

- SCRUM
- Kanban
- XP - Programowanie Ekstremalne

SCRUM

Scrum to zdecydowanie kluczowa i prawdopodobnie najważniejsza część Agile. To ona jest odpowiedzialna za zarządzanie procesem iteracyjnym w projekcie. Jednak w tym przypadku, pojedyncze iteracje nazywane są sprintami, czyli wcześniej zdefiniowanymi partiami pracy, posiadającymi przypisane do nich zadania do wykonania oraz z wydzielonym terminem. Sprinty trwają najczęściej od dwóch do czterech tygodni w zależności od zarządzeń czy planów biznesowych. W przypadku „The Lore” sprinty w początkowej fazie prac, czyli około trzech miesięcy trwały dwa tygodnie, a następnie zostało to zredukowane do jednego tygodnia. Wynikało to ze wzrostu jakości otrzymywanej

produkту, ponieważ był on częściej oddawany w ręce testerów, a także sam proces programistyczny mógł być lepiej rozplanowany. Dawało to możliwość przypisania sprintów do wykonania konkretnych funkcjonalności zamiast tworzenia przeładowanego sprintu z różnymi funkcjonalnościami. Tym samym wzrosła wydajność pracy, ponieważ grupa docelowa oczekiwana już gotowego uaktualnienia produktu. Kolejnym doskonałym elementem scrum'u jest wprowadzenie ról poszczególnych członków zespołu. Jak wcześniej było wspomniane, w podejściu zwinnym brakuje hierarchizacji w zespole, co może doprowadzić do katastrofy organizacyjnej.[22] Dlatego też Scrum wprowadza trzy funkcje:

- Scrum Master
- Właściciel produktu
- Zespół deweloperski

Scrum master to przede wszystkim osoba odpowiedzialna za pilnowanie przestrzegania zasad Scrumowych. Diametralnie wpłynął na wydajność i produktywność zespołu, ponieważ wielokrotnie podczas sprintów pilnował postępów prac oraz jakości ich wykonania. Zajmował się również komunikacją z źródłami zewnętrznymi po to aby pozostałe osoby z zespołu nie musiały tracić na to czasu i odrywać się od pracy. Do jego obowiązków należało również planowanie szczególnych spotkań, tak zwanych retrospektyw oraz przeglądów sprintów.

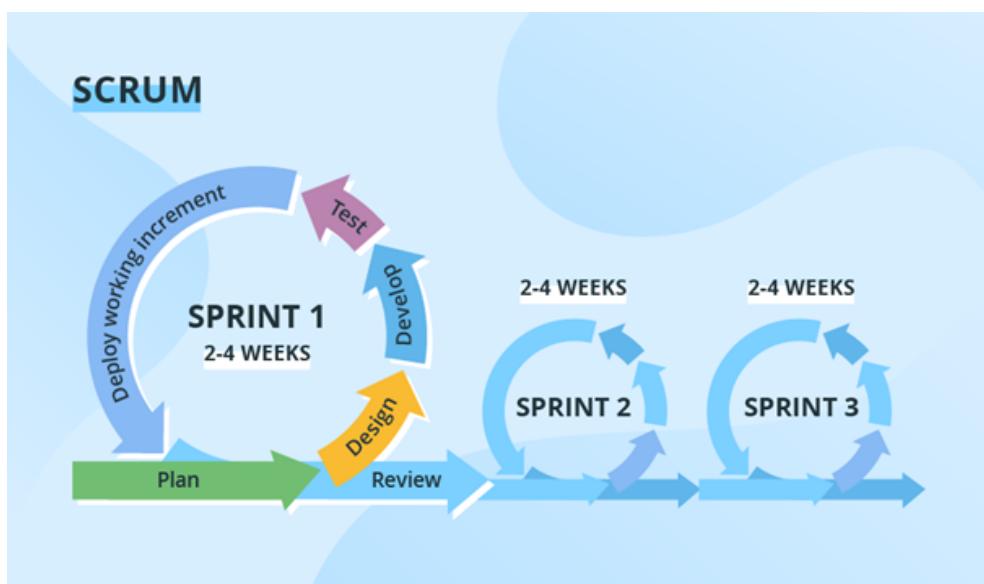
Retrospektywą nazywamy spotkaniem odbywającym się cyklicznie co 2-3 tygodnie w celu ustalenia organizacji pracy, dyskusji o odczuciach każdego członka zespołu na poszczególne tematy. Czasami bywały to rozmowy stricte związane z kodem, jednak bywały też takie, które dotyczyły jakości pracy bądź kontaktami interpersonalnymi. W dużym uproszczeniu był to czas dla zespołu aby mógł przedyskutować to co się podoba, nie podoba i jakie można podjąć akcje aby to zmienić.

Przegląd sprintu jest również cyklicznym spotkaniem, jednak odbywał się częściej – po każdym zakończonym sprintem. Jego głównym zadaniem była dyskusja członków zespołu o stworzonych przez nich funkcjonalnościach. Dzielona jest wiedza w jaki sposób konkretny deweloper podszedł do wykonania zadania oraz w jaki sposób go zrealizował. Dzięki temu, w sytuacji gdy inna osoba z zespołu otrzymała zadanie rozwoju tej funkcjonalności, w kolejnym sprintem, miała ona już bazę wiedzy wystarczającą do jego realizacji. Pozwala to na zaoszczędzenie czasu potrzebnego do zrozumienia działania kodu napisanego przez inne osoby. Podczas przeglądu sprintów tworzony jest również następny

sprint, w którego skład wchodzą zadania bazujące na tworzeniu funkcjonalności zgodnie z harmonogramem. Po wypełnieniu nowego sprintu konkretnymi zadaniami, dochodziło do rozmowy o przydzielaniu ich do konkretnej osoby. Głównymi czynnikami doboru zadań do członków zespołu było ich doświadczenie pracy z zadaniami łączonymi bądź zależnymi, które były wykonane w wcześniejszych sprintach.

Właścicielem produktu jest najczęściej osoba, poza zespołem, zlecająca wykonanie projektu, jednak w przypadku gry „The Lore”, ten tytuł spoczywał na barkach członka zespołu. Jego zadaniem było zadbanie o klarowną wizję projektu i pilnowanie czy tworzony projekt jest zgodnie z nią realizowany. Odpowiadał również za rozwój jak i maksymalizację wartości produktu. Robił to przy użyciu decydowania o kolejności wykonywania prac i wprowadzanych funkcjonalności. Zarządzał również ich priorytetem i decydował o treści tworzonych zadań. Wykonywał to wraz z Scrum Masterem, jednak to jego decyzja była najważniejsza. Projekt był tworzony i rozpisany zgodnie z jego wizją produktu finalnego.

Ostatnią grupą jest zespół deweloperski. Charakterystyką tej roli jest odpowiedzialność za dostarczenie, przygotowanej przez nich, gotowej do wydania aktualizacji projektu. Realizują backlog sprintu, przygotowanego wcześniej podczas przeglądu sprintów.

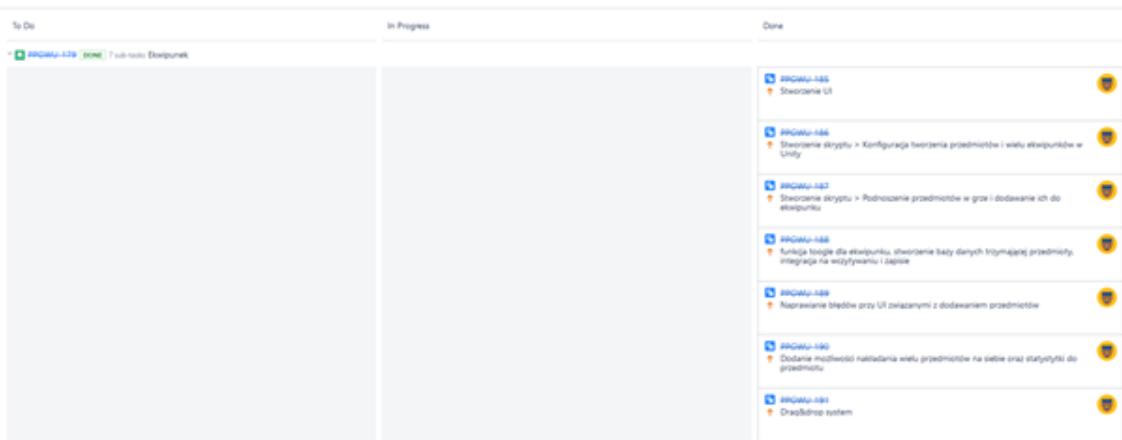


Rysunek 2.9. Źródło: ScienceSoft, SCRUM [19]

Na powyższym zdjęciu prezentowany jest cykl realizacji sprintów oraz poglądowa wizualizacja modelu pracy w Scrum'ie.

Kanban

Kanban to kolejna metoda w metodyce Agile. Jej głównym założeniem jest wizualizacja procesu realizacji zadań. Przy użyciu tak zwanej tablicy kanbanowej, było obrazowane na jakim etapie znajduje się poszczególne zadanie. W projekcie „The Lore”, tablica została podzielona na najprostszy układ tj. podział na czynności „do zrobienia”, „w trakcie realizacji” i „zrobione”. Jej celem było doprowadzanie do sytuacji, w której organizacja pracy poszczególnych osób z zespołu jest widoczna dla wszystkich. Dzięki temu proces realizacji zadań od siebie zależnych przechodził w procesie bezproblemowym i jawnym. Jednocześnie, dawało to wizualne potwierdzenie na jakim etapie pracy jest każdy członek zespołu. Przy niezadowalającym tempie pracy interweniował Scrum master lub właściciel produktu.[23]



Rysunek 2.10. Tablica kanbanowa w programie Jira. Przykład z projektu „The Lore”

„Jira to platforma do organizowania pracy i uruchamiania oraz monitoringu procesów w organizacjach.” [24] Pozwala na utworzenie i zarządzanie tablicą kanbanową w projektach.

Programowanie ekstremalne

Programowanie ekstremalne to już ostatnia metoda użyta w realizacji projektu „The Lore”, skupia się ona na adaptacji do częstych zmian wynikających z

cyklicznych aktualizacji projektu. Głównym jego założeniem, który został wykorzystany jest *Pair programming* czyli programowanie w parze lub nawet większej grupie ludzi. Dzięki wspólnemu planowaniu, dyskutowaniu o potencjalnym rozwiązaniu, a następnie zaimplementowaniu go, maleje czas wykonania zadania które normalnie byłoby tworzone przez pojedynczego dewelopera. Jest to szczególnie użyteczne gdy zbliża się termin zakończenia sprintu a zadanie jest we wczesnej fazie tworzenia lub gdy deweloper natrafił na ślepy zaułek i potrzebuje świeżego spojrzenia na zadanie.

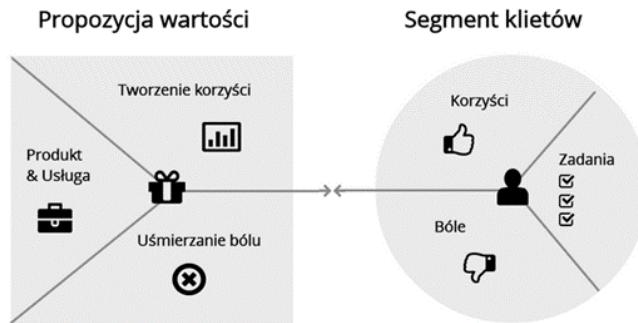
2.6.2. Cykl życia projektu „The Lore”

Faza zbierania informacji i analizy

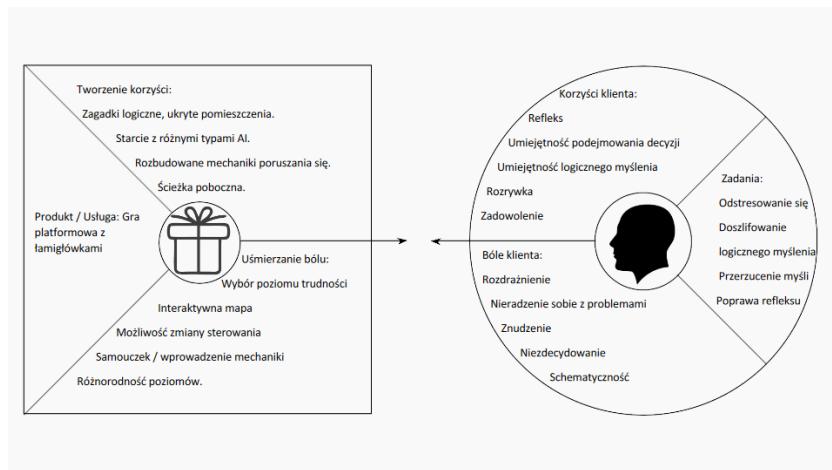
Podczas pierwszych faz realizacji projektu „The Lore”, była tworzona szeroka dokumentacja, której zadaniem było zebranie jak najwięcej informacji potrzebnych do obrania odpowiedniego kierunku prac. Dzięki niej właściciel produktu wraz z scrum masterem mieli potencjalną wizję na tworzony harmonogram wraz z przygotowaniem backlogu z zadaniami na najbliższe sprinty. Dokumentacja wraz z później przygotowanymi na jej podstawie *user stories* była w tej fazie kluczowa. W skład dokumentacji wchodziła wizja projektu, VPC oraz BMC.

Dokument wizji projektu - jego głównym założeniem jest przedstawienie przeznaczenia tworzonego oprogramowania oraz jego głównych cech i przyjętych założeń. Dokument dzieli się na sekcje wprowadzenia, celu projektu, badania rynku, użytkowników docelowych, szczegółowego opisu produktu wraz z zakresem i ograniczeniami.

Value Proposition Canvas (VPC) to angielska nazwa na przedstawienie unikalnej propozycji wartości modelu biznesowego. Jego celem jest łatwiejsza analiza i zaprojektowanie produktu wraz z jak najtrajniejszym skonfigurowaniem rozwiązań pod potrzeby klientów.[25]



Rysunek 2.11. Źródło: Product Vision, Value Proposition Canvas



Rysunek 2.12. Value Proposition Canvas dla projektu „The Lore”. Przykład z „The Lore”

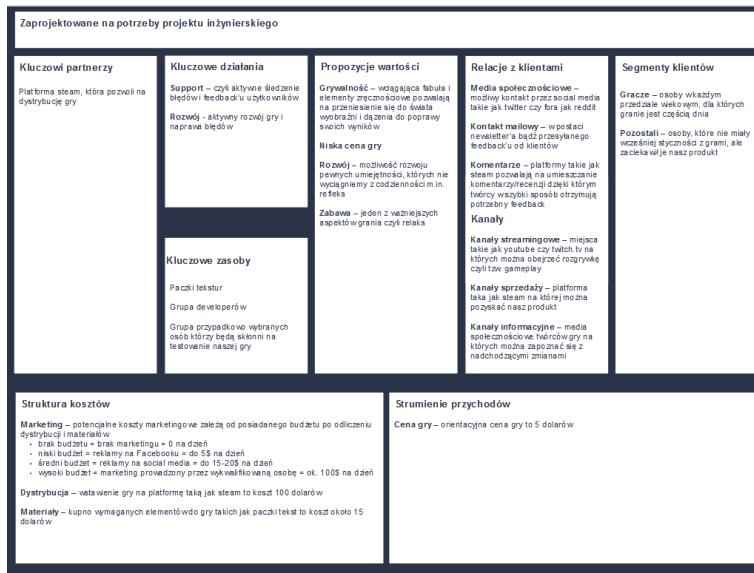
Powyżej przedstawiony jest przykład unikalnej propozycji wartości modelu biznesowego oraz już oficjalny model z projektu gry "The Lore".

Business Model Canvas (BMC) to podobnie jak VPC angielska nazwa na szablon modelu biznesowego. Jego zadaniem jest w wizualny sposób przedstawienie w jaki sposób planowany projekt ma przynieść zyski. Składa się z 9 podstawowych obszarów, które obrazują grupę docelowych użytkowników oraz w jaki sposób zostanie im dostarczony produkt. Jednocześnie pokazuje wszelkie korelacje z wydatkami jak i potencjalnymi zarobkami planowanego projektu.[26]

Partnerzy	Kluczowe działania	Propozycja wartości	Relacje z klientami	Segmenty klientów
Who jest naszym kluczowym partnerem? Jakiem zweryfikowane firmą lub organizacją są nam niezbędne do działania? Jakiem kluczowym zasoby i działania realizując nasi partnerzy?	Jakie działania musimy podporządkować, by dostarczyć naszym klientom propozycję wartości? Jakich działań wymagają nasze kanały dostarcza do klienta i nawiązujące z nim relacji?	Jaką wartością generujemy dla naszych klientów? Za co będą płacić? Co ich boli, a co ma dla nich kluczowe znaczenie? jakie problemy klientów rozwiązyjemy? Jaki produkty i usługi będziemy oferować?	Jakich relacji oczekujemy od naszych klientów? Czy to zakupu określonego produktu, a może szybkiej i automatycznej obsługi? Czy sprawią nawiązwanie relacji z klientami jest zintegrowany z potencjalnymi obiektami modelu biznesowego?	Who jest naszym klientem? Dla którego budujemy produkt/usługę? Ktora oferujemy wartości? Kto będzie płacić?
Kluczowe zasoby		Kanały		
Jakiem kluczowych zasobów pośredniczym, żeby zaoferować naszą propozycję wartości? Jakiem zasobów wymagają nasze kanały dostarcza do klienta i nawiązujące z nim relacji?		Gdzie bywają nasi klienci? Gdzie chcieliby spotkać naszych klientów? Z jakich kanałów będziemy korzystać przy nawiązaniu kontaktu z klientami?		
Struktura kosztów		Struktura przychodów		
Jakiem koszty generuje nasz model biznesowy? Jakiem nakładów finansowych generują kluczowe zasoby, działania, partnerzy?		Za co klienci są w stanie zapłacić? Za co nie będą płacić? Które elementy naszego produktu/usługi będą darmowe, a które płatne?		

Rysunek 2.13. Źródło: Product Vision, Business Model Canvas

Powyżej przedstawiony jest przykładowy szablon modelu biznesowego. Natomiast poniżej już oficjalny model z projektu gry "The Lore".



Rysunek 2.14. Business Model Canvas dla projektu „The Lore”. Przykład z „The Lore”



Rysunek 2.15. Backlog z zadaniami w programie Jira. Przykład z „The Lore”

Utworzone *user stories* zostały wykorzystane do stworzenia pierwszych zadań w backlogu projektu.

Faza projektowania

Po zakończeniu dwóch poprzednich faz, w których rozwijany był zamysł stworzenia gry komputerowej „The Lore”, przyszedł czas na skupienie się na tym w jaki sposób zostanie ona stworzona. Pierwszą rozterką było wybranie odpowiedniego środowiska. Pośród wielu możliwości na rynku, wybór padł na UNITY.

Miał on opinię najbardziej przyjaznego dla początkujących twórców gier komputerowych. Posiadał on również najlepiej rozwiniętą dokumentację techniczną i największą bazę użytkowników udzielających się na specjalistycznych forach. Unity posiada również autorski sklep z materiałami, które są nieocenione przy tworzeniu gry w ograniczonym czasie.

Po wybraniu środowiska trzeba było podjąć decyzję w sprawie używanego języka programowania. Tutaj jednak nie było to trudne, ponieważ z grupy Boo, UnityScript oraz C# tylko ten ostatni był nadal wspierany przez Unity oraz cieszył się największą popularnością wśród twórców. Również był to najlepiej znany język w grupie deweloperskiej.

Następnym etapem był wybór docelowej technologii oraz systemów na jakie ma trafić wdrożony projekt. Estymowany był tutaj dostępny czas oraz wymagania jakie kryją się za tworzeniem gry, jednocześnie na urządzenia mobilne, stacjonarne jak i konsole. Niezależnie od możliwości jakie posiada Unity dla budowania projektu na różne platformy, sama implementacja funkcjonalności stwarzała komplikacje. Wynikały one z unikalnych cech każdej platformy. Stąd decyzja o wykonaniu gry na pojedynczą platformę – stacjonarną, ale z kompatybilnością na trzy najpopularniejsze systemy operacyjne – Windows, Linux, MacOS.

Kolejnym etapem projektowania technicznego było dobranie odpowiedniej platformy wdrożeniowej. Na rynku jest ich dość sporo. Często różnice są marginalne i zależą od różnych twórców oraz cen za wdrożenie autorskiego projektu. Dylematem było wykorzystanie drogiego lecz jednocześnie najpopularniejszego miejsca na dystrybucję gry bądź wykorzystanie mało znanych platform, które udostępniają możliwość wdrożenia produktu za darmo. W związku z mocno ograniczonym budżetem projektowym, wybór musiał paść na platformę mniej znaną.

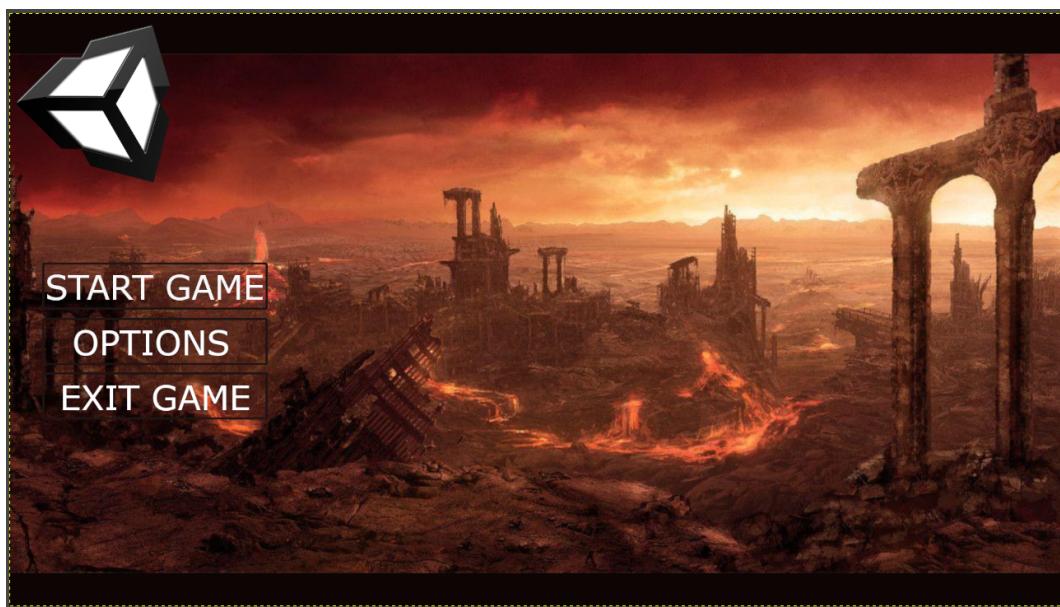
Następnym elementem, który musiał zostać wybrany, była platforma przechowywania plików. Tworząc projekt w pojedynkę, takie miejsce służy do bezpiecznego przechowywania progresu prac. Jednak przy pracy w zespole, tego typu miejsce służy również do współdzielenia i szybkiego dostępu do plików projektu. Najpopularniejszym na rynku środowiskiem jest Github. Wy magał on jednak specjalnej konfiguracji, ponieważ pliki przekazywane przez Unity są odmienne od standardowych. Przede wszystkim, sam edytor Unity tworzy setki tymczasowych plików, które nie zawsze powinny trafić na Githuba.

Odpowiednia konfiguracja pliku gitignore, którego celem jest nieprzesyłanie wpisanych tam rozszerzeń, typów plików czy folderów była kluczowa.

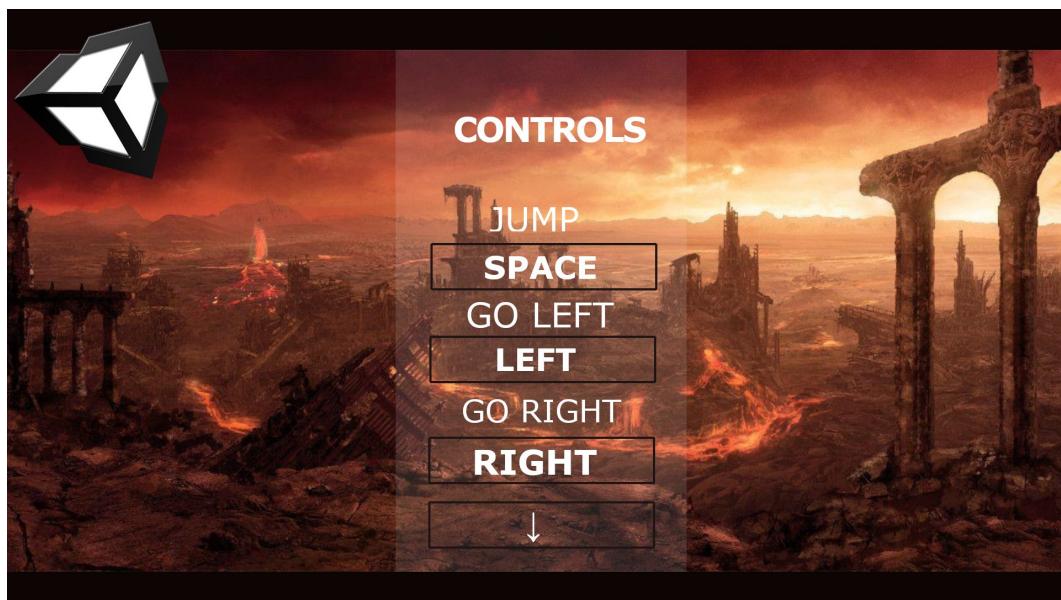
Kolejnym problemem, z którym trzeba było sobie poradzić były błędne referencje do specjalistycznych obiektów w Unity. Tego typu informacje wysyłane były na Githuba w specjalnych plikach z rozszerzeniem .meta, które rozwiązały wszelkie problemy z znikającymi obiektyami stworzonymi przez programistów.

Ostatnim i raczej największym problemem była konfliktowość wynikająca z edycji tych samych plików binarnych przez wielu programistów. Rozwiążanie konfliktów podczas prób scalania Githubowych gałęzi było niemożliwe. Stąd, kluczowa była komunikacja wewnętrz zespołowa wraz z odpowiednią organizacją pracy i umiejętnym podziałem zadań. W fazie projektowania zostały tworzone również pierwsze szkice interfejsów graficznych. Wiele z nich zostało zmodyfikowanych lub zupełnie zmienionych.

Poniżej przedstawione są zdjęcia prototypów interfejsów graficznych głównego menu oraz menu sterowania.

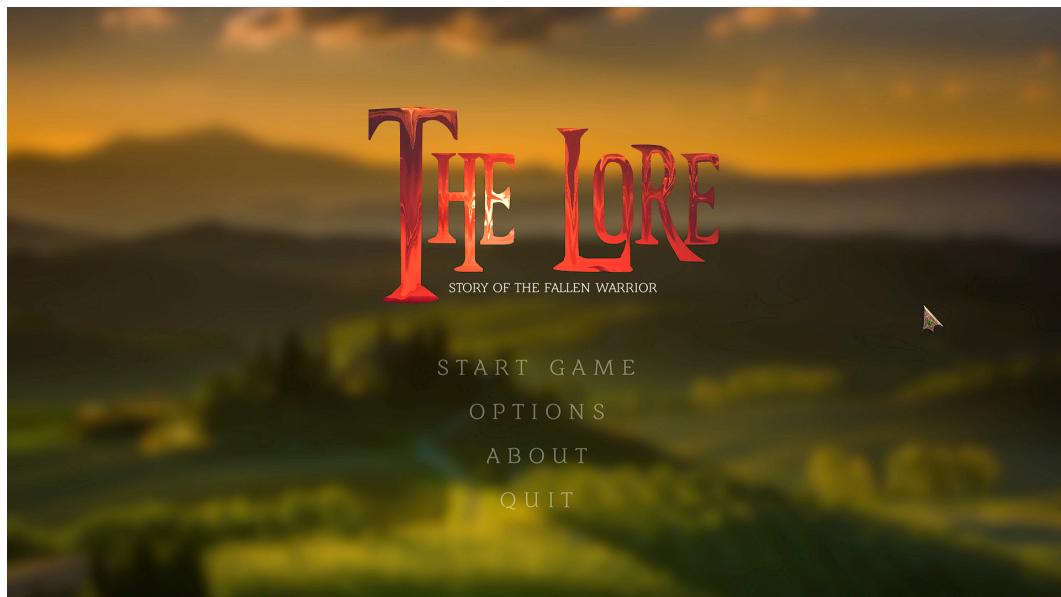


Rysunek 2.16. Prototyp interfejsu menu głównego. Przykład z „The Lore”

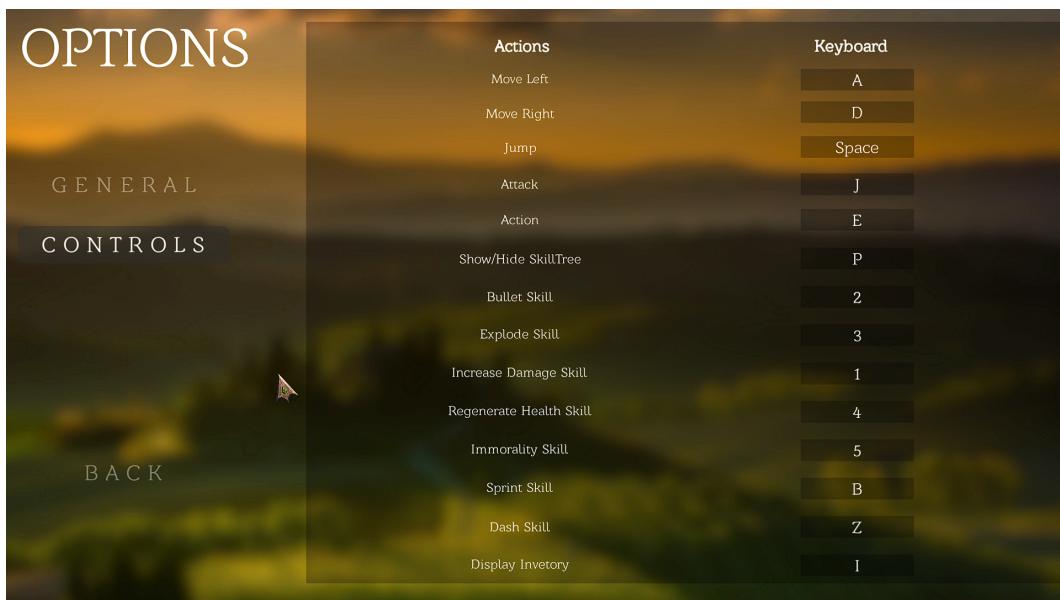


Rysunek 2.17. Prototyp interfejsu menu sterowania. Przykład z „The Lore”

Poniżej przedstawione są zdjęcia wdrożonych interfejsów graficznych głównego menu oraz menu sterowania.



Rysunek 2.18. Wdrożony interfejs menu głównego. Przykład z „The Lore”

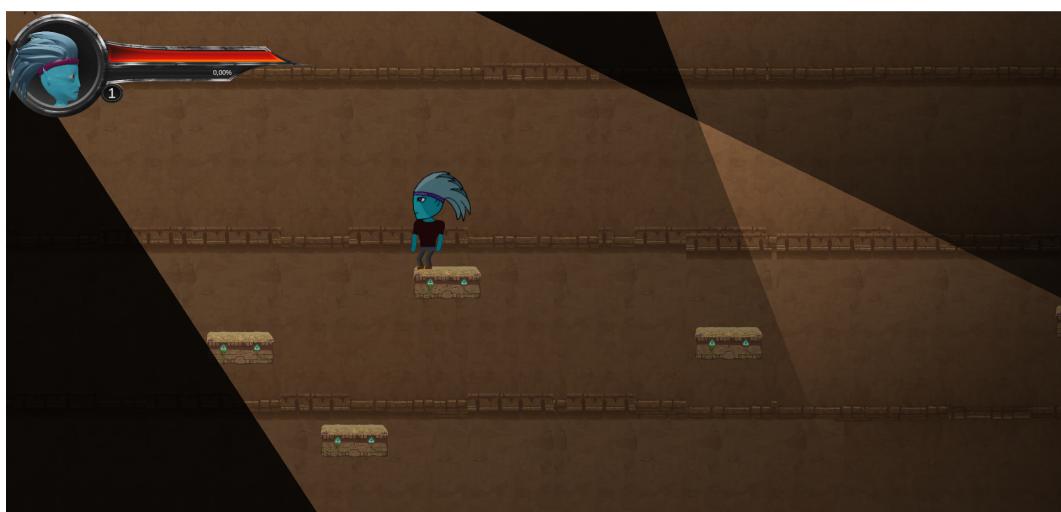


Rysunek 2.19. Wdrożony interfejs menu sterowania. Przykład z „The Lore”

Faza implementacji i testów

Faza implementacji wraz z testami była powtarzana wielokrotnie w wyniku metodyki Agile. Poprzez model iteracyjny, każdy sprint wykorzystywany był do zaimplementowania wcześniej zaprojektowanych funkcjonalności oraz przedstawienia ich grupie testerów. Dzięki nim, w projekcie gry „The Lore” zespół deweloperski był w stanie na bieżąco wprowadzać wszelkie poprawki wraz z sugestiami otrzymywany podczas testów. Przykładową funkcjonalnością jest element poziomu samouczka – poruszające się platformy. Zgodnie z powtarzającą się opinią o trudności w przechodzeniu tego elementu, zostały wprowadzone poprawki wraz z dodatkowymi ułatwieniami. Przykładowym rozwiązaniem było dodanie specjalistycznej fizyki, która miała na celu przyciąganie gracza w kierunku nadchodzącej platformy. Innym, bardziej wizualnym ułatwieniem było dodanie tras, po których poruszają się platformy. Ostatnim ułatwieniem było zmniejszenie prędkości poruszania się platform oraz powiększenie ich powierzchni.

Poniżej przedstawiono zdjęcia z pierwszej i ostatniej iteracji tworzonej funkcjonalności, która odzwierciedla zmiany po otrzymanych uwagach z fazy testów



Rysunek 2.20. Pierwsza iteracja ruchomych platform. Przykład z „The Lore”



Rysunek 2.21. Ostatnia iteracja ruchomych platform. Przykład z „The Lore”

To był jeden z wielu procesów poprawionych dzięki otrzymanej opinii podczas testów, jednak wystarczająco poważny żeby pokazać jak kluczowym było wybranie metodyki Agile w projekcie stworzenia gry komputerowej „The Lore”.

2.7. PODSUMOWANIE

Podsumowując, w niniejszym rozdziale pracy zostały przedstawione fundamentalne pojęcia oraz procesy realizacji projektu informatycznego. Poprzez zdefiniowanie i zobrazowanie podstawowego podziału cyklu życia projektu oraz wskazanie jednych z najważniejszych i najpopularniejszych zagadnień poszczególnych elementów przybliżono czytelnikowi życie projektu. Dzięki temu długiemu wprowadzeniu, możliwe było przedstawienie owego procesu na przykładzie gry "The Lore". Podkreślając wagę każdego członu projektu można zauważać jak niezbędną rolę ma cały proces, który pozwala z wyższym prawdopodobieństwem ukończyć projekt z sukcesem.

ROZDZIAŁ 3

Zagadki logiczne

3.1. WPROWADZENIE DO ZAGADEK LOGICZNYCH

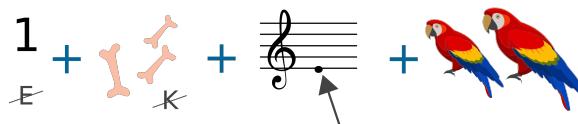
3.1.1. Omówienie zagadnienia

Zagadką logiczną określamy zadanie, którego celem jest odnalezienie odpowiedzi na pytanie, logiczne dojście do rozwiązania problemu czy też czasami abstrakcyjne myślenie. Możemy wyróżnić różne rodzaje zagadek, między innymi graficzne, czego przykładem jest znany, często używany w szkołach podstawowych rebus. Główną ideą takiej rozgrywki jest rozwój swoich intelektualnych możliwości przy dobrej zabawie. [27]

3.1.2. Przykłady logicznych zagadek w prawdziwym świecie

Rebus

Jednym z przykładów takiej zagadki może być rebus. To prosta rozgrywka, polegająca na odgadnięciu hasła na podstawie przytoczonego obrazka. Często treści w rebusach mogą być niejednoznaczne, co sprawia, iż nie są one trywialne i wymagają wielu kombinacji haseł, związanych z danym obrazkiem. [28]



Rysunek 3.1. Rebus. Źródło: Zespół autorski Politechniki Łódzkiej, licencja: CC BY 3.0

Piętnastka - przesuwane puzzle

Kolejnym przykładem może być też piętnastka, która w grze „The Lore” została zaimplementowana jako przesuwane puzzle rozmiarów 3 na 3. Temat ten szerzej poruszony będzie w dalszej części pracy, gdzie oprócz założeń związanych z rozwiązyaniem zagadki, przedstawiona zostanie logika idąca za zrealizowaniem tej zagadki w grze opartej na silniku Unity.



Rysunek 3.2. Piętnastka. Źródło: Wikipedia, Sliding Puzzle, Micha L. Rieser - Public Domain

Puzzle

Jedną z najpopularniejszych zagadek logicznych, z którą styczność miał zapewne każdy z nas, są puzzle. Jest to gra, w czasie której rozwiązujeMY problem polegający na ułożeniu z dostępnych elementów logicznego obrazka, który zazwyczaj jest dołączony do gry - w postaci fotografii umieszczonej w zestawie lub na pudełku.

Słowo „puzzle” pochodzące z XVI wieku, oznaczało "stan zagubienia". Sama gra ma swoje początki w wieku XVIII, gdy John Spilsbury, grawer i kartograf umieścił mapę na drewnie, następnie przepiował ją wokół konturów każdego kraju, który się na niej znajdował. To co powstało postanowił użyć do nauki geografii - ułożenie logicznej mapy uczyło jak wygląda mapa danego regionu. Taka pomoc dydaktyczna zdobyła szybko popularność, wszak była to nauka przez zabawę, stosowano ją dość często, nawet jeszcze w wieku XIX.

Trudność puzzli zależy zazwyczaj od kształtu elementów i ich ilości. Puzzle często mają specyficzny kształt, przez co nie każdy element pasuje do innego, co pozwala nam odrzucić możliwość połączenia niektórych par. Ilość elementów zwiększa ilość możliwych oraz logicznych kombinacji danych puzzli, przez co zwiększa się czas realizacji zadania. Największa układanka została wykonana przez firmę Educa, posiadała ona aż 42 000 elementów i przedstawia dość

bajeczny krajobraz, na którym znajdują się najpopularniejsze budynki z całego świata - Big Ben, Wieża Eiffela czy Krzywa Wieża w Pizie. Ułożenie tej układanki możemy liczyć w setkach godzin. [30]



Rysunek 3.3. Źródło: Pixnio, Przykład puzzli - Public Domain

3.1.3. Wdrożenie zagadek do gry opartej na silniku Unity

Zanim przedstawione zostaną dwa sposoby wdrażania zagadek, ze względu na sposób umieszczenia mini-gry w odpowiedniej hierarchii projektowej, warto poznać podstawowe pojęcia, które dotyczą projektu Unity.

Scena w Unity

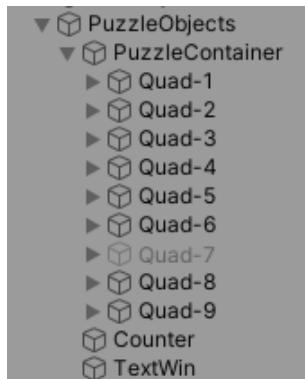
W Unity możemy podzielić naszą grę na sceny. Scena to obiekt zawierający nasze menu, czy dane środowisko gry (na przykład poziom gry). Dobrą praktyką jest, aby każdy level gry był osobną sceną, co skróci czas jego ładowania. W scenie możemy umieścić swoje obiekty, skrypty czy grafiki. [31]

Obiekt - GameObject

Obiekt, zwany w Unity GameObject, to klasa podstawowa dla wszystkich podmiotów w scenach Unity. Do każdego obiektu możemy przypinać kolejne obiekty, tworząc hierarchiczność, która może się przydać w odpowiedniej realizacji projektu. [32]

Komponent

Komponent jest klasą podstawową dla GameObject. Jest to klasa, przykładowo w języku C#, która dołączana jest do obiektu Unity, celem wykonania na nim jakiś operacji. [33]



Rysunek 3.4. Hierarchia obiektów. Przykład z projektu „The Lore”

Sposoby dodawania mini-gier

W toku pracy nad projektem „The Lore” rozpatrywano dwa sposoby dodawania mini-gier do gry:

- Każda mini-gra jest w osobnej scenie:
 - W przypadku rozbudowanych poziomów nie powiela szerokiej listy obiektów sceny,
 - Pozwala na mniejsze pobieranie zasobów w danej jednostce czasu - połączenie działającego poziomu z algorytmiką mini-gry, może być uciążliwe na słabszych komputerach.
- Każda mini-gra zawiera się w scenie poziomu gry:
 - Unikamy dłuższego ładowania się poziomu gry po zakończeniu mini-gry.

Po wewnętrznej dyskusji wybrano opcję wydzielenia do osobnej sceny. W toku testowania tego rozwiązania uznano, iż nie jest ono bardzo uciążliwe pod kątem czasu ładowania poziomu, zaś pozwala na zachowanie porządku w projekcie. Jak się okazało już w przypadku tworzenia poziomu pierwszego gry, ta decyzja była właściwa. Z racji sporego rozbudowania tej sceny i dużej liczby obiektów na niej zawartych, dodanie mini-gier do tej sceny mogłoby spowodować chaos organizacyjny. Dodatkowo mogłoby wydłużyć czas ładowania poziomu i spowodować pobieranie większych zasobów w czasie gry.

3.2. PRZESUWANE PUZZLE

3.2.1. Omówienie zagadnienia i ogólne założenia

Przesuwane puzzle to układanka, złożona zazwyczaj z kwadratowej liczby elementów, najczęściej jest to szesnaście pól. Pola są jednakowych rozmiarów i oznaczone są liczbami od 1 do ($n-1$), gdzie n to liczba dostępnych pól w układance. Jedno pole jest puste, pozwala to na przeniesienie sąsiednich elementów puzzli względem siebie. Rozgrywka kończy się, gdy ułożymy puzzle w odpowiedniej kolejności, według rosnącego porządku liczb lub powstania odpowiedniego obrazka. Trudno określić kto odpowiada za stworzenie zagadki. Wiadomym jest, że w 1878 roku pochodzący ze Stanów Zjednoczonych Samuel Loyd wypromował układankę, jednak prawdopodobnie nie jest to jego pomysł. Dość popularną nazwą na rozgrywkę jest "piętnastka", określająca ilość dostępnych pól w najpopularniejszym ułożeniu - 4x4. [29]

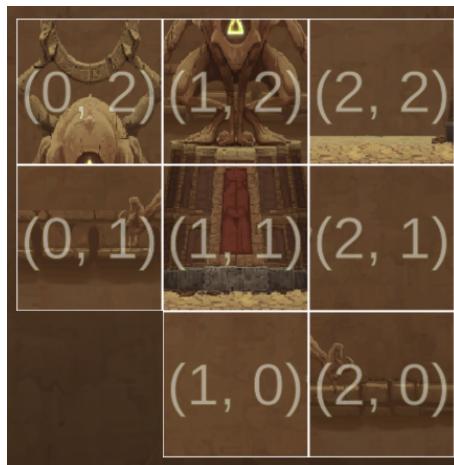
W grze „The Lore” gracze staną przed rozwiązyaniem zagadki gdzie do dyspozycji mają dziewięć pól. Podczas projektowania układanki w grze uznano, iż zagadka z piętnastoma elementami może być dość trudna, a korzyści płynące z rozwiązyania jej będą nieadekwatne do poświęconego czasu, stąd ilość pól jest mniejsza niż w najpopularniejszej wersji rozgrywki.

3.2.2. Algorytmika

Pojedynczy element puzzle - PuzzleBlock

Każdy pojedynczy element puzzla jest zainicjalizowany jako obiekt, który określamy jako PuzzleBlock. Obiekt posiada koordynaty, które określają jego położenie w przestrzeni na układance.

Tak jak na załączonym przykładzie, wartość x rośnie w prawą stronę, zaś Y w górę, gdzie x dotyczy położenia w poziomie, a y w pionie. W założeniach przedstawiona została logika, która mówiła, iż element może zostać przemieszczony wtedy i tylko wtedy, gdy pole obok niego jest wolne, czyli nie posiada żadnego PuzzleBlock - elementu z liczbą lub obrazkiem. Dla lepszego efektu wizualnego, w grze „The Lore” element porusza się stopniowo, aby sprawiał wrażenie, iż porusza się realistycznie. Z racji, iż w Unity każda akcja wykonuje się podczas pokazywania kolejnej klatki, element porusza się w minimalnym stopniu przez sekundę.



Rysunek 3.5. Koordynaty każdego pola. Przykład z gry „The Lore”

Plansza - Puzzle. Losowanie kolejności puzzli

Plansza rozgrywki posiada 8 elementów PuzzleBlock oraz jedno puste pole. Algorytm powinien wylosować dla 8 pól ich położenie na planszy - tak jak w wyżej przedstawionym przykładzie. W tym celu na początku losujemy dla każdego bloku wartość od 0 do 8. W C# możemy to zrobić przy użyciu funkcji `System.Random().Next(a, b)`, który losuje wartości z zakresu a, b). Przykładowo:

Wyciąg 3-1. Fragment klasy Puzzle.cs

```

1  private int randomPosition()
2  {
3      int pos = 0;
4      do
5      {
6          pos = new System.Random().Next(0, 9);
7      }
8      while (isOnBoard[pos]);
9      isOnBoard[pos] = true;
10     return pos;
11 }
```

Gdzie `isOnBoard[pos]` jest tablicą, która weryfikuje, czy dane pole nie jest już zajęte. Jeśli jest, ponownie losujemy wartość dla `PuzzleBlock`. Oczywiście to nie koniec - z tej wartości musimy stworzyć położenie w postaci (x, y) , gdzie x to położenie w poziomie, a y w pionie. W tym celu jedna z tych wartości będzie resztą dzielenia wylosowanej pozycji przez trzy (ilość pól w linii), a

druga ilorazem całkowitym pozycji i liczby trzy. Kolejną ważną rzeczą będzie weryfikacja, czy obecne ułożenie puzzli jest wykonalne.

Plansza - Puzzle. Weryfikacja kolejności puzzli

Jak się okazuje niektóre ułożenia puzzli powodują, iż nie ma żadnego rozwiązania tego problemu. Z taką sytuacją spotykaliśmy podczas pierwszych testów mini-gry puzzle.



Rysunek 3.6. Puzzle bez rozwiązania. Przykład z gry „The Lore”

Występuje tutaj dość prosta zasada. Łamigłówka przesuwanych puzzli z 8 elementami jest rozwiązywalna wtedy i tylko wtedy, gdy liczba inwersji stanu początkowego jest parzysta. Czym jest zaś w tym przypadku liczba inwersji?

Inwersją nazwiemy parę liczb, której wartości są w odwrotnej kolejności, niż w zakładanym stanie końcowym. [34] Przyjmijmy taką sytuację:

1	2	3
6	4	5
7	8	

W tym przypadku liczba inwersji wynosi dwa. Elementami zbioru inwersji są pary (6,4) oraz (6,5) - jak wiadomo, liczba 6 jest w kolejności po cyfrach 4 oraz 5. W tym przypadku rozwiążemy puzzle.

1	2	3
6	5	4
7	8	

W tym przypadku liczba inwersji wynosi już trzy. Elementami zbioru są pary (6,5), (6,4) oraz (5,4). Takich puzzli nie da się rozwiązać. W projekcie platformowej gry w Unity zastosowaliśmy prostą weryfikację tej sytuacji.

Wyciąg 3-2. Fragment klasy Puzzle.cs

```

1 int inversions = 0;
2 for (int i = 0; i < numbersOrdered.Length - 1; i++) {
3     for (int j1 = i + 1; j1 < numbersOrdered.Length - 1; j1++) {
4         if (numbersOrdered[j1] > numbersOrdered[i]) {
5             inversions++;
6         }
7     }
8 }
```

Gdzie **inversions**, to liczba znalezionych inwersji, a **numberOrdered** to lista kolejności puzzli w stanie wejściowym. Jeżeli wartość **x** znajduje się w liście **numberOrdered** przed wartością **y** i jest od niej większa, to wtedy zwiększamy licznik inwersji o jeden. Oczywiście pozostaje nam prosta weryfikacja, czy liczba inwersji jest nieparzysta - w takim wypadku puzzle będą nierozwiązywalne.

Wyciąg 3-3. Fragment klasy Puzzle.cs

```

1 if (inversions % 2 == 1)
2 {
3     pab.restartPuzzleButton();
4 }
```

Gdzie **pab** jest obiektem odwołującym się do akcji **PuzzleActionsButton**, zawierającym przycisk **restartButton** - ten sam, który dostępny jest w grze w celu zrestartowania obecnego ułożenia puzzli.

Każdy obiekt (puzzle) ma przypisane do siebie dwie akcje: **OnBlockPressed** i **OnFinishedMoving**. Pierwsza akcja dotyczy tego, co ma się dziać w momencie wybrania przez gracza danego elementu. W tym przypadku dany element dodawany jest do kolejki, której zadaniem jest kontrola poruszania się puzzli. Pozwala to uniknąć sytuacji, gdy użytkownik zdołałby w bardzo szybkim czasie wybrać dwa elementy - wtedy moglibyśmy być świadkami nachodzących się puzzli lub nawet wejścia dwóch elementów na jedno miejsce. Gdy element będzie pierwszy w kolejce, rozpoczęcie się ruch naszego elementu - wtedy też przypisane będą mu nowe koordynaty. Wtedy przychodzi czas na drugą akcję, **OnFinishedMoving**. Początkowo funkcja zwiększa naszą ilość ruchów o jeden. Następnie sprawdza, czy obecne ułożenie puzzli odpowiada oczekiwaniu wynikowi końcowemu. Jeśli tak, jesteśmy informowani o ilości zdobytego doświadczenia. Ponownie widzimy animację przesuwającej się płytki, tym razem zamkającej się. Jeśli nie, algorytm sprawdza, czy istnieją elementy w kolejce, w tym przypadku powtarza się część procedury wykonywanej w

OnBlockPressed - element porusza się na swoje miejsce, zmieniając koordynaty, następnie znów wykonując akcję **OnFinishedMoving**.

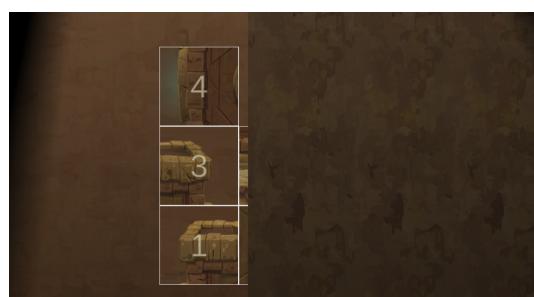
3.2.3. Przedstawienie przykładowu w grze „The Lore”

W grze „The Lore” zagadka przesuwanych puzzli pełni rolę opcjonalnej rozgrywki, a za jej rozwiązywanie gracz otrzymuje punkty doświadczenia. Mini-grę możemy rozpocząć znajdując się w pobliżu charakterystycznej płytki, wyróżniającej się podświetleniem.



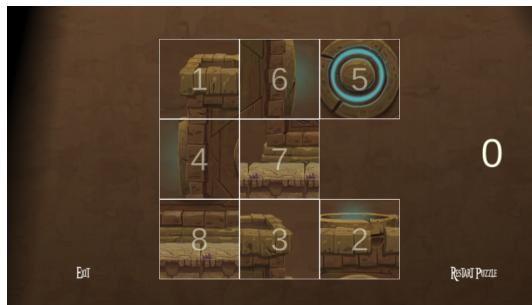
Rysunek 3.7. Miejsce rozpoczęcia puzzli. Przykład z projektu „The Lore”

W momencie naciśnięcia przycisku akcji kamera zbliża się do obiektu, czyli podświetlonej płytki. Następnie widzimy animację otwierającej się płytki sprawiającej wrażenie, iż ścianka jest przesuwana przez gracza.



Rysunek 3.8. Animacja rozpoczęcia puzzli. Przykład z projektu „The Lore”

Po skończonej animacji rozpoczynamy mini-grę. Oprócz bloczków z kolejnością, na ekranie widzimy również dwa przyciski i cyfrę zero.



Rysunek 3.9. Cały interfejs mini-gry puzzle. Przykład z gry „The Lore”

Gdzie **Exit** kończy rozgrywkę, przenosząc nas z powrotem w miejsce, w którym rozpoczęliśmy zagadkę, **Restart Puzzle** powoduje ponowne rozlosowanie puzzli, odwołując się do akcji w klasie **PuzzleActionsButtons**, czyli tej samej, która uruchamiana jest w przypadku, gdy puzzle nie mają rozwiązania. Dodatkowo, po prawej stronie widnieje informacja, ile ruchów do tej pory wykonałyśmy rozwiązuając zagadkę. Jest to dość istotna informacja dla gracza, gdyż to od ilości ruchów zależy ile punktów doświadczenia zdobędzie za rozwiązanie tej zagadki. Zastosowany wzór dla gry „The Lore” wygląda następująco:

$$y = \begin{cases} 5 & \text{gdy } x \geq 250 \\ 100 - (x/25) * 10 & \text{gdy } x < 250 \end{cases}$$

Gdzie **x** jest liczbą wykonanych ruchów. Po zakończeniu mini-gry jesteśmy informowani o ilości zdobytego doświadczenia. Ponownie widzimy animację przesuwającej się płytka, tym razem zamykającej się.



Rysunek 3.10. Wygrana rozgrywka puzzle. Przykład z gry „The Lore”

Każda mini-grę puzzle można wykonać w grze tylko jeden raz. Ponowne próby są niemożliwe, a o tym, iż zagadka została zakończona, jesteśmy informowani przez fakt, iż płytka symbolizująca mini-grę nie jest podświetlona.



Rysunek 3.11. Wygaszona płytka. Przykład z gry „The Lore”

3.2.4. Przedstawienie przykładu w innych grach

Nancy Drew: Hauting of Castle Malloy

„The Haunting of Castle Malloy” to gra dwuwymiarowa, przygodowa, osadzona w Irlandii. Postać wybiera się do tego kraju, aby zostać drużbą na ślubie swojego przyjaciela. Jadąc autem na uroczystość zostaje zaatakowany przez upiorną postać, przez co wpada do rowu. Jak się później okazuje, zaginął Pan Młody. Główny bohater, Nancy Drew chce odkryć tajemnicę, przechodząc przez świat pełny zagadek. Właśnie podczas poszukiwania swojego Pana Młodego, gracz natyka się na przesuwane puzzle.



Rysunek 3.12. Przesuwane puzzle. Przykład z gry „Nancy Drew: Hauting of Castle Malloy”

W czasie przechodzenia kolejnych etapów, użytkownik ma możliwość przyglądania się obiektem, które miją. To właśnie w tym momencie może natrafić na omawianą rozgrywkę. Przed mini-grą gracz widzi jaki jest stan wyjściowy puzzli.

Co warto podkreślić, w przeciwnieństwie do gry „The Lore”, na poszczególnych elementach puzzli nie widzimy cyfr oznaczających poprawną kolejność puzzli. Aby zakończyć zagadkę należy również posiadać ostatni element, który po ułożeniu obrazka umieszczamy w brakującym miejscu. [39]

Resident Evil 4

„Resident Evil 4” jest grą typu survival horror. Toczy się ona w fikcyjnym mieście Raccoon City, gdzie po sześciu latach kończy się śledztwo dotyczące nielegalnych badań i eksperymentów medycznych nad wirusem, prowadzone przez organizację Umbrella. Działania firmy spowodowały destabilizację tego miasta. Główny bohater, Leon S. Kennedy zostaje wyznaczony jako agent, który ma odnaleźć porwaną przez podejrzaną organizację, córkę prezydenta USA. Bohater przemierzając kolejne miasta odkrywa kolejne tajemnice związane z porwaniem. W toku gry zdarza się, iż gracz kieruje Ashley Graham, czyli porwaną córką prezydenta USA. To właśnie w czasie kontrolowania tej postaci możemy natrafić na mini-grę związaną z przesuwanym puzzlami. Bohaterka może natrafić na totem, gdzie widać osiem elementów. Dialogi bohaterki podpowiadają graczowi, iż elementy mogą ułożyć się w logiczną całość. W przeciwnieństwie do poprzedniego przykładu, nie mamy tutaj pokazanego efektu końcowego rozgrywki. Ostatni element, tak jak w „Nancy Drew: Haunting of Castle Malloy” musimy odnaleźć, aby zagadka została ukończona.



Rysunek 3.13. Przesuwane puzzle. Przykład z gry „Resident Evil 4”

3.3. ZAGADKA Z RURAMI

3.3.1. Omówienie zagadnienia

Zagadka z rurami jest rozgrywką, której celem jest ułożenie ścieżki z dostępnych kształtów rur. Rodzaj rur jest tutaj ważny, ponieważ oprócz kształtów pionowych czy poziomych, dostępne są inne rodzaje - skośne, skrętne, półokrągłe. Ścieżka powinna wyznaczać drogę wody począwszy od źródła, zazwyczaj zaznaczonego niebieskim kolorem, do końcowej rury. Rozgrywka kończy się, gdy woda dotrze do ostatniej rury w ścieżce, jeżeli będzie miejsce, w którym powinna kończyć się droga, gracz wygrywa. W innym wypadku ponosi porażkę. Poziom naładowania wody pełni rolę wyznacznika czasu w którym użytkownik musi ułożyć odpowiednią ścieżkę. Zazwyczaj w tego typu rozgrywkach, po pewnym czasie woda przenosi się z rury źródłowej do kolejnej, aby dać graczu pewien zapas czasowy na podjęcie odpowiednich decyzji. W niektórych grach gracz ma wybór rur, które dostępne są przez całą rozgrywkę - tak jest przykładowo w grze „The Lore”. Jednak są też cięższe przypadki, gdy mamy kolejkę kilku rur - wtedy musimy podejmować decyzje w oparciu o mniejszą pulę elementów.

3.3.2. Algorytmika

Mini-gra z rurami jest dość złożoną rozgrywką, dlatego zanim przedstawiony zostanie cały algorytm, warto na początku przytoczyć wszystkie obiekty, które pojawią się w czasie przedstawienia procedury.

Pipe

Jest to obiekt związany z rurą. To w tym miejscu określany jest typ rury, nadawana jest mu grafika oraz nazwa - ostatni element związany jest wyłącznie z częścią deweloperską, nie jest dostępne dla wszystkich użytkowników. W grze „The Lore” mamy dostępne kilka typów rur:

- Rury pionowe - 9 elementów,
- Rury poziome - 4 elementy,
- Rury skrętne lewo-góra - 1 element,
- Rury skrętne prawo-dół - 2 elementy,
- Rury dół-lewo - 1 element,
- Rury dół-prawo - 2 elementy.

Jak wcześniej wspomniano, liczba elementów poszczególnych typów rur odgry-

wa ważną rolę. Nie jest możliwe ułożenie bez możliwości zejścia na dół (czyli między innymi bez rur skrętnych).

PipeSlot

Jest miejscem, w którym może znaleźć się obiekt typu **Pipe**. Plansza, na której możemy kłaść elementy, jest rozmiarów 8x5, czyli tworzonych jest czterdzieści instancji obiektu **PipeSlot**. Jeżeli na danym miejscu znajduje się obiekt **Pipe**, wtedy możemy wykonywać na nim akcje - określa to zmienna **canDrag**. Do każdego obiektu przypisane są akcje **OnBeginDragEvent**, **OnEndDragEvent**, **OnDragEvent** oraz **OnDropEvent**. Związane są one ze systemem **Drag&Drop** czyli przeciagania i układania zadanych elementów. Jego działanie zostanie przedstawione w dalszej części prezentowanej algorytmiki.

Działanie algorytmu - UI

Klasa **UI** odpowiada za interakcje z widocznym interfejsem i interakcje z obiektami. Na samym początku, do każdego elementu **PipeSlot** przypisywane są jego akcje. Gdy to już się stanie, czyszczona jest lista **pipes** w której trzymane będą wszystkie obiekty **Pipe**. Ma to na celu zabezpieczenie algorytmu przed wczytaniem elementów z poprzedniej instancji rozgrywki. Następnie wszystkie obiekty **PipeSlot** jako obiekt **Pipe** nadany mają *null* – również z powodu zabezpieczenia przed niepożądanymi sytuacjami. Teraz czas na zainicjalizowanie dwóch najważniejszych **PipeSlot**, chodzi konkretnie o pierwszy i ostatni element. Dla tych miejsc przypisywana jest rura horyzontalna. Dodatkowo oba miejsca, pomimo iż znajduje się na nich rura, nie mogą być przeniesione – parametr **canDrag** ustawiony jest na *false*. Oczywiście chodzi to o nieingerowanie w początek i koniec algorytmu. Gdy to już się stanie, algorytm dodaje rury wszystkich typów do odpowiedniej tablicy - **pipes**. Dopiero wtedy następuje losowanie im miejsc. W tym miejscu następuje przypisanie do elementu **PipeSlot** obiektów **Pipe**. Oczywiście, w momencie losowania miejsc, procedura weryfikuje, czy na danym miejscu znajduje się już jakaś rura, dzieje się to w dość trywialny sposób.

Wyciąg 3-4. Fragment klasy **UI.cs**

```
1 | if (pipeSlots[x].Pipe == null)
```

Jeżeli warunek nie jest spełniony, następuje ponownie losowane miejsca dla zadanego obiektu **Pipe**. Gdy warunek zostanie spełniony, rozgrywka rozpoczyna się. W tym momencie zaczyna mijać czas, którego limit wyznacza zmienna **targetTime** – ustaliona na pięć sekund. Czas ten dotyczy przepływu wody przez jedną rurę. Co warto podkreślić, co każdą pełną sekundę zmienia się grafika danego obiektu **Pipe** – nadając efekt płynięcia wody.



Rysunek 3.14. Grafiki rury skrętnej. Przykład z projektu „The Lore”

W danym momencie procedura weryfikuje też, czy istnieje połączenie. Dzieje się to poprzez warunki:

Wyciąg 3-5. Fragment klasy UI.cs

```
1 | if (recentPipe.right && (i + 1) % 8 != 7 && pipeSlots[i + 1].Pipe.left
2 | && !cameFrom.Equals("right"))
```

Gdzie **recentPipe.right** to sprawdzanie czy obecna rura może skręcić w prawo, $((i + 1) \% 8 \neq 7)$ jest weryfikacją, czy znajdujemy się przy najbardziej wysuniętym na prawo **PipeSlot**. Następnie weryfikujemy, czy sąsiedni slot może być skrętny w lewo. Niestety, warunki nie są zbyt proste. Podobnie sprawdzamy to dla wszystkich stron, z odpowiednimi wartościami dla danych przypadków. Jeżeli minie czas, a dany **PipeSlot** nie będzie miał odpowiedniego połączenia, jesteśmy informowani o porażce. Jeżeli zaś algorytm wykryje, iż takie połączenie istnieje, a naszym obecnym indeksem na planszy jest 39 (liczba pól – 1), to algorytm przenosi nas do poziomu gry, z którego zaczynaliśmy rozgrywkę.

GameManager

Ostatnim ważnym obiektem w kontekście tej mini-gry jest **GameManager**. Odpowiada on za wszystkie akcje przypisane do **PipeSlot**. **BeginDrag**, **EndDrag** oraz **Drag** odpowiadają za przenoszenie danej rury - w tym momencie pozycja obiektu jest ściśle związana z kursem na ekranie. W momencie, gdy puścimy lewy przycisk myszy, wykonuje się akcja **Drop**. Tutaj weryfikujemy, czy **Pipe** został przeniesiony na **PipeSlot**, który nie posiada żadnej rury. Jeśli tak, do danego miejsca przypisana jest nowa rura. W innym wypadku rura wraca na swoje poprzednie miejsce.

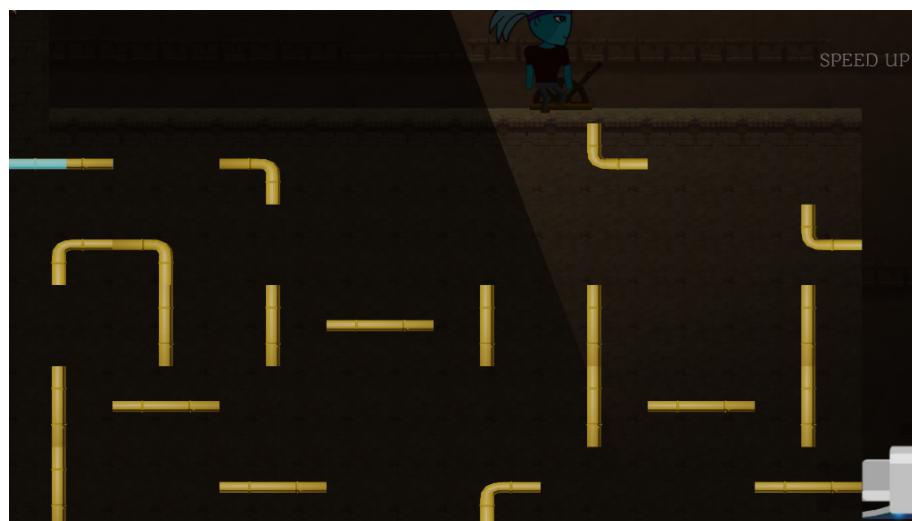
3.3.3. Przedstawienie przykładu w grze „The Lore”

Jest to kolejna mini-gra zawarta w grze „The Lore”, która jest elementem obowiązkowym do ukończenia poziomu "0", pełniącego rolę samouczka. Gracz zbliżając się do miejsca, w którym może rozpocząć rozgrywkę jest informowany poprzez komunikat o możliwości jej aktywacji.



Rysunek 3.15. Poziom 0 - Zagadka z rurami. Przyk³ad z projektu „The Lore”

Jak widzimy po prawej stronie znajduje siê do³, w którym widzimy dwa totemy oraz płytki strumień wody. W przypadku, gdyby gracz postanowi³ wskoczyć do dziury zostanie przeniesiony z powrotem w okolice przełącznika (widocznego na fotografii po lewej stronie). Gdy aktywujemy przełącznik przyciskiem akcji, kamera zmierza w dól, aby pokazać nam układ rur.



Rysunek 3.16. Zagadka z rurami. Przyk³ad z projektu „The Lore”

W lewym górnym rogu mamy źród³o strumienia, zaś w prawym dolnym rogu zakoñczenie (szara rura). Użytkownik sterując myszką mo¿e przek³adać rury o wybranym kszta³cie, tworzyć po³±czenia, celem stworzenia ścieżki, która jest konieczna do ukoñczenia mini-gry. Rozrywka zawsze generuje tą samą liczbê poszczególnych typów rur. Co warto podkre¶lić, liczba ta jest wa¿na -

jeżeli mamy zbyt mało elementów danego typu, to po prostu ułożenie ścieżki może być niemożliwe. Zbyt duża liczba elementów zaś pozwoliłaby na trywialne rozwiązanie, na przykład w kształcie litery L. W grze „The Lore” jest możliwość szybkiego zakończenia rozgrywki poprzez wybranie opcji **Speed Up**. Powoduje ona szybkie przeniesienie strumienia wody, do końcowej rury ułożonej przez nas ścieżki. Jeżeli odpowiednia droga została ułożona, mini-gra zostanie zakończona sukcesem, w przeciwnym przypadku porażką. Z tej racji przycisk spełnia rolę nie tylko szybkiego pominięcia rozgrywki, gdy jesteśmy już pewni zwycięstwa i nie chcemy tracić czasu, ale też zrestartowania rur.

3.3.4. Przedstawienie przykładu w innych grach

Pipe Mania

Gra logiczna stworzona przez *The Assembly Line* na platformy Amiga. Gra pojawiła się również na komputerach z systemem Windows. Rozgrywka polega na ułożeniu rurociągu z losowej puli rur. Kolejka rur, które otrzyma gracz wyświetla się po lewej stronie ekranu. Cała gra polega na ułożeniu ścieżki z dostępnych elementów, przy użyciu wszystkich rur. Liczba rur, które pozostały graczowi w kolejce wyświetlona jest w górnej części ekranu. Podobnie jak w grze „The Lore”, woda rozpoczyna swój bieg po pewnym czasie, gracz widzi ten czas w postaci paska po prawej stronie ekranu. [35]

Soda Pipes

Gra wzorowaną na poprzednim przykładzie jest „Soda Pipes”. Podobnie jak w poprzedniej grze, gracz musi ułożyć ścieżkę z rur, które losuje nam gra. W odróżnieniu do poprzedniczki, naszym celem jest zakończenie ścieżki w danym miejscu. Gra oprócz typowej rozgrywki oferuje również tryby gry, takie jak "puzzle mode" - pozwala on na układanie rur bez ograniczenia czasowego oraz trybu, w którym do pokonania jest 28 poziomów, które posiadają różne zadania. [44]

3.4. ZAGADKA Z OTWIERANIEM ZAMKU

3.4.1. Omówienie zagadnienia

Jest to bardzo powszechna mini-gra, która dodawana jest do wielu gier, które posiadają funkcjonalność posiadania ekwipunku. Jej celem jest otworzenie skrzyni, polega to na wykonaniu różnych operacji. Zazwyczaj zaczyna się od okrycia odpowiedniej sekwencji poruszania wytrychem, otwieranie zapadki poprzez poruszanie myszką, czy wcisnięcie odpowiedniego guzika, gdy postęp otwierania zamku jest na odpowiednim etapie. Otwarcie zamku pozwala graczom odblokować skrzynię, która zawiera elementy wyposażenia lub pozwala na otwarcie drzwi, które zaprowadzą nas do ważnych dla gry pomieszczeń. Z racji na charakter rozgrywki element ten bywa raczej opcjonalną metodą przejścia gry. Z tego względu w niektórych grach, posiadających system umiejętności, otwarcie niektórych zamków jest niemożliwe lub bardzo trudne, jeżeli nie rozwiniemy odpowiednich umiejętności.

3.4.2. Algorytmika

Pierwszym i najważniejszym dla całej mini-gry elementem jest wylosowanie przez grę sekwencji ruchów, która potrzebna jest do ukończenia rozgrywki. Odbywa się to w skrypcie **PickLockGenerateSequence**. Podczas projektowania generatora napotkano na problem związany z ciągłym losowaniem tych samych liczb. Powodem jest tutaj oczywiście synchronizacja, która przy ładowaniu scen potrafi sprawiać deweloperom problemy. Z tego powodu powstał ten skrypt, który wyróżnia się weryfikacją unikalności wylosowanej liczby. Wszystko rozpoczyna się od losowania liczby z zakresu <1,100).

Wyciąg 3-6. Fragment skryptu PickLockGenerateSequence.cs

```

1  private int randomSide()
2  {
3      int random = 0;
4      do
5      {
6          random = new System.Random().Next(1, 100);
7      } while (wasRandomed[random - 1]);
8      wasRandomed[random - 1] = true;
9      return random % 2;
10 }

```

Tablica **wasRandomed** to element, który inicjalizowany jest z samych wartości **false**. Ilość elementów jest równa **n**, czyli ilości możliwych wylosowanych liczb. Jeżeli liczba zostanie wylosowana ponownie, skrypt próbuje wylosować kolejną. Tak jak wspomiano gracz ma dwie możliwości ruchów, dlatego aby wszystko działało poprawnie funkcja zwraca **mod 2** z wylosowanego elementu. Wynikiem funkcji jest więc 0, co oznacza ruch w lewo oraz 1, co oznacza ruch w prawo. Losowanych jest pięć elementów, co tworzy nam tablicę ruchów które musi wykonać gracz.

Wyciąg 3-7. Fragment skryptu PickLockGenerateSequence.cs

```

1 for (int i = 0; i < numberOfWorks; i++)
2 {
3     moves[i] = randomSide();
4 }

```

Po wszystkim aktywowany jest komponent **Picklock**. Komponent ten odpowiada za wykrywanie jakich wyborów dokonał gracz. Jeżeli gracz wykonał dobry ruch, zostanie poinformowany o tym poprzez komunikat oraz zwiększy się licznik kontrolny **step** ilości dobrych ruchów. W przeciwnym razie licznik się wyzeruje. Dla efektu wizualnego, gracz widzi w którą stronę poruszył się jego wytrych - pozwala to łatwiej zapamiętać wybór. Odbywa się to poprzez zmianę pozycji elementu transform.

Wyciąg 3-8. Fragment skryptu Picklock.cs

```

1 picklockTransform.position += new Vector3(10 * direction, 0, 0) * 0.1f;

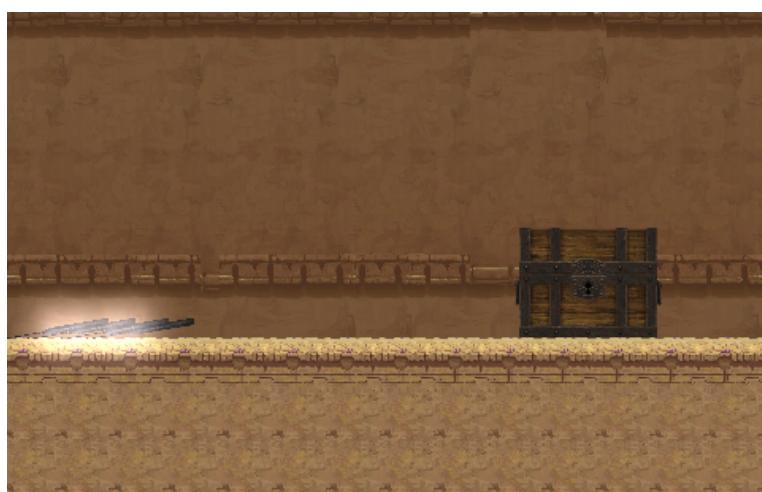
```

Gdzie **picklockTransform** to komponent typu transform, zawierający tablice

pozycji, rotacji oraz skalowania (odpowiednio parametry *position*, *rotation* i *scale*) obiektu. W tym przypadku interesuje nas tylko **position**.

3.4.3. Przedstawienie przykładowu w grze „The Lore”

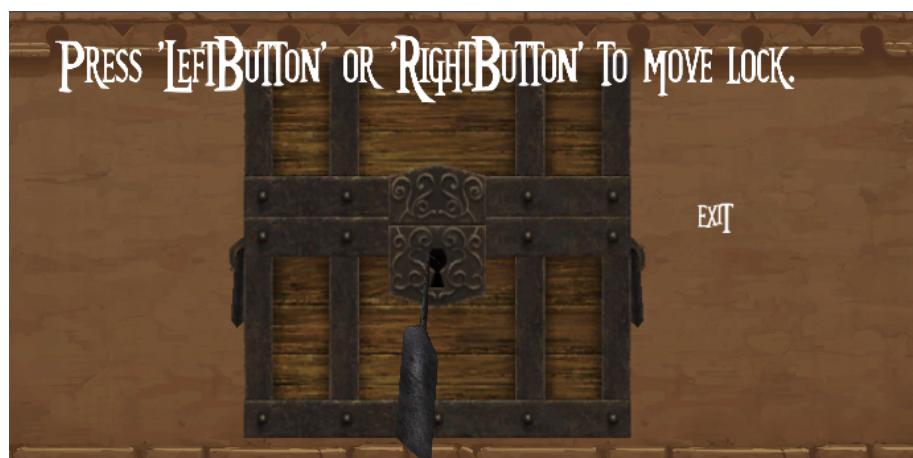
W grze „The Lore” otwieranie zamków dotyczy skrzyni, które położone są w różnych częściach mapy. Podczas poziomu samouczka jesteśmy informowani o możliwości rozpoczęcia mini-gry. Rozpoczęcie mini-gry wymaga posiadania wytrychów, które wyróżnione są na mapie białym podświetleniem.



Rysunek 3.17. Wytrychy i skrzynia. Przykład z projektu „The Lore”

Jeżeli chcemy otworzyć skrzynie nieposiadając w ekwipunku żadnego wytrychu, jesteśmy informowani o tym fakcie poprzez komunikat na ekranie. Gdy wciśniemy odpowiedni przycisk akcji, kamera zbliża się do skrzyni, po pewnym czasie rozpoczyna się mini-gra. Na wstępie jesteśmy informowani poprzez komunikat o sterowaniu w rozgrywce. Klawisze "Left/RightButton" są zależne od ustawień sterowania i dotyczą przycisków poruszania się graczem w lewo i prawo. Jeżeli naciśniemy któryś z tych przycisków, wytrych przesunie się zgodnie z naszym poleceniem. Po każdym ruchu jesteśmy informowani, czy był on odpowiedni.

Zadaniem gracza jest zgadnąć losowy ciąg elementów lewo/prawo o długości 5. Każdy zły ruch powoduje konieczność powtórzenia sekwencji. Nagrodą za rozwiązanie problemu są eliksiry przywracające punkty życia, które przydają się w czasie rozgrywki. Z racji, iż jest to rozgrywka nieobowiązkowa, gracz w każdej chwili może ją opuścić wybierając opcję **Exit**.



Rysunek 3.18. Mini-gra otwieranie skrzyni. Przykład z projektu „The Lore”

3.4.4. Przedstawienie przykładowu w innych grach

Mafia 2

Gra Mafia II osadzona jest w amerykańskim mieście rządzonym przez mafie. Głównym bohaterem jest Vito Scaletta, syn włoskich imigrantów. Los sprawia, iż bohater zmuszony jest do szybkiego zarobku, przez co wplata się w mafijne porachunki. Realizując zadania dla swoich pracodawców, nierzaz włamywał się do różnych pomieszczeń.



Rysunek 3.19. Otwieranie zamków. Przykład z gry „Mafia 2”

Gracz, może również w czasie gry włamywać się do zamkniętych pokoi lub budynków, celem szybszego przejścia poziomu, obejścia wrogów czy zdobycia pieniędzy. Dodatkowo mini-gra pozwala na odblokowanie zamkniętych pojazdów. W mini-grze gracz używa myszki. Widoczny jest zamek z trzema zapadkami, po prawej stronie widzimy wytrych przy pierwszej zapadce. Użytkownik sterując myszką wertykalnie musi ustawić zapadkę w takiej pozycji, aż będzie ona podświetlona na kolor zielony. Jeżeli gracz zrobi to w złym momencie, zostanie cofnięty do poprzedniej zapadki. [36]

Assassin's Creed Unity

W grze „Assassni's Creed Unity” wcielamy się w Arno Dorianę, członka tajnego bractwa Assasynów, którzy toczą odwieczną walkę z organizacją zwaną Templariusze. Gra osadzona jest w czasach rewolucji francuskiej w Paryżu. Bohater będąc w centrum najważniejszych wydarzeń osiemnastowiecznego Paryża nieraz potrzebuje włamać się do jakiegoś budynku, aby wykonać misję. W grze otwieranie zamków dotyczy pomieszczeń, które dają podobnie jak w poprzedniej grze, możliwość obejścia wrogów, czy też zdobycie ekwipunku oraz włamywania się do skrzyń - podobnie jak w grze „The Lore”. Rozpoczynając mini-grę widzimy ilość zapadek, błękitny pasek poruszający się wertykalnie oraz błękitny prostokąt. Zadaniem gracza jest wcisnąć przycisk akcji w momencie, gdy poruszający się znaczek znajdzie się w błękitnym polu. W przypadku, gdy gracz zrobi to w złym momencie, traci z ekwipunku wytrych. Warto też podkreślić, iż w grze istnieje system umiejętności, gdzie dwie umiejętności dotyczą poziomu otwierania zamków. Podczas gry spotkamy trzy poziomy trudności, które charakteryzują się większą ilością zapadek czy też szybszym poruszaniem się błękitnego paska. [37]



Rysunek 3.20. Otwieranie zamków. Przykład z gry „Assassin's Creed Unity”

Gothic 2

„Gothic 2” to gra osadzona w krainie Myrtana, w której toczy się walka pomiędzy siłami dobra i zła. Wcielamy się w byłego więźnia kolonii więziennej, który w wyniku pokonania Śniącego, tajemniczego potwora, niszczy magiczną barierę, która odcinała kolonię karną od reszty świata. Po upadku bariery bohater zostaje wybudzony przez nekromantę Xardasa, który informuje go o nowych zagrożeniach, zagrażające krainie Myrtana. Gracz sterując bohaterem może dopuszczać się przestępstw, pozwalają mu one na zdobycie odpowiedniego ekwipunku czy złota. Może się to odbywać między innymi poprzez plądrowanie skrzyni w chatach mieszkańców miasta Khorinis, czy też napotkanych, opuszczonych budynkach. Logika otwierania zamków, zarówno drzwi jak i skrzyni, jest dość podobna do zastosowanego systemu w grze „The Lore”. Gracz aby otworzyć daną skrzynię musi posiadać wytrychy, które może zakupić od kupców w mieście. Otwieranie skrzyni polega na odgadnięciu odpowiedniej sekwencji lewo – prawo, w przypadku pomyłki gracz może utracić wytrych. Co ciekawe, gra nie losuje kolejności sekwencji. Jest ona stała dla danej skrzyni w świecie gry, co może stanowić swego rodzaju ułatwienie - odpowiednie sekwencje można po prostu odnaleźć w internecie.

3.5. LABIRYNT

3.5.1. Omówienie zagadnienia

W pierwotnej definicji, labirynt oznacza budowlę, która charakteryzuje się układem dużej ilości pomieszczeń, które połączone są kretymi ciągami korytarzy. Oczywiście miało to na celu utrudnić dostęp do pewnego pomieszczenia osobom niepowołanym. Z podobnych powodów w niektórych grach mamy do czynienia z labiryntem, zmuszając w ten sposób gracza do szukania wskazówek, czy też zapamiętywaniu ścieżek, celem wyjścia z labiryntu.

3.5.2. Algorytmika

Założenia: algorytm z nawrotami

Do stworzenia labiryntu możemy użyć algorytmu z nawrotami. Początkowo powinniśmy stworzyć graf, gdzie każdy węzeł ma co najmniej jedno połączenie, to będzie nasz labirynt. Pomiędzy węzłami, które nie są połączone będzie

znajdować się ścianą, której użytkownik nie może przejść. Aby stworzyć taki graf, zaczynamy od generowania grafu bez ścieżek - możemy wyobrazić sobie, iż jest to po prostu prostokąt, podzielony na kilka kwadratów stworzonych przez ściany. Algorytm rozpoczyna się w pierwszym węźle, dodając go do stosu. Następnie wybiera pierwszego nieodwiedzonego sąsiada, tworząc w ten sposób pierwsze połączenie węzłów - z punktu widzenia gracza, usuwając pomiędzy nimi ścianę. Proces powtarzany jest aż do momentu, gdy trafimy do węzła, który nie ma żadnego nieodwiedzonego sąsiada. Gdy algorytm już dotrze do takiego miejsca, cofa się, jednocześnie usuwając elementy ze stosu, aż do momentu trafienia na węzeł, który posiada nieodwiedzonego sąsiada. Algorytm skończy się, gdy w naszym stosie nie będzie elementów. [38]

Implementacja: algorytm z nawrotami

Przejdźmy teraz do zaprojektowania takiej sytuacji. Na sam początek zdefiniujmy stany, które może posiadać każdy węzeł w naszym grafie (labiryntce). Jak wiemy, może mieć ściany z czterech stron.

Wyciąg 3-9. Stany danego węzła - skrypt C#

```
1 public enum NodeState  
2 {  
3     LEFT = 1,  
4     RIGHT = 2,  
5     UP = 4,  
6     DOWN = 8,  
7     VISITED = 128  
8 }
```

W ten sposób możemy łatwo określić wszystkie ściany, które otaczają nasz węzeł. Przykładowo, gdybyśmy chcieli przedstawić, iż nasz węzeł otacza ścianą lewa i górna, stworzylibyśmy stan **NodeState nodeState = NodeState.LEFT | NodeState.UP**. Jak wspomnieliśmy, chcemy, aby na początku wszystkie ściany tworzyły swego rodzaju siatkę, która odgradza każdy węzeł od innych. Generowanie jest w tym przypadku bardzo proste, wystarczy w odpowiedniej pętli wykonać odpowiednią liczbę iteracji **nodeWall[i] = NodeState.LEFT | NodeState.RIGHT | NodeState.UP | NodeState.DOWN**. Możemy również uznawać wartości za liczby binarne, LEFT jako 0001, UP 0010 i tak dalej. Oznaczmy też stan odwiedzonego pola jako 128 - reprezentacja bitowa 1000 0000. W przypadku, gdy chcemy dodać do naszego obiektu nowy stan, wystarczy wykonać operację dodania alternatywy do obecnego stanu. **node[x,y] |= NodeState.VISITED** Oczywiście w ramach większego porządku w kodzie warto określić **nodeWall**

jako tablicę dwuwymiarową, wyznaczając położenie w osi poziomej i pionowej. Pomoże nam to rozpoznać w jakim miejscu na labiryncie znajduje się dany węzeł. Jest to szczególnie przydatne, ponieważ będziemy tworzyć strukturę pod pozycję każdego obiektu. Pozycje, tak jak wyżej wspomniano, przedstawiamy w geometrii dwuosiowej, co oczywiście pomoże nam śledzić gdzie znajduje się dany obiekt.

Wyciąg 3-10. Pozycja obiektu - skrypt C#

```
1 | public enum NodePosition
2 | {
3 |     public int x;
4 |     public int y;
5 | }
```

Warto też stworzyć strukturę do przechowywania informacji o sąsiadach naszego węzła. Wystarczą wyżej przedstawione parametry **NodeState** i **NodePosition**. Dodatkowo w ramach kontroli powinniśmy stworzyć listę takich obiektów, które nie zostały jeszcze odwiedzone przez algorytm. Mając już takie struktury, kolejnym krokiem byłoby stworzenie metody, która zwraca nam stan danego sąsiada. Tak jak wspominano na początku, sąsiad będzie wybierany losowo, dla tego warto już na wstępie stworzyć obiekt **System.Random()**. Dzięki stworzonej wcześniej metodzie pobierania sąsiadów możemy go wybrać na przykład ze względu na kolejność na danej liście. Tworzymy też najważniejszy dla algorytmu z nawrotami stos. Każdy odwiedzony węzeł powinien być oznaczony wartością **NodeState.VISITED** i dodany do tego stosu. Wystarczy, aby były to obiekty **NodePosition**, jednak nic nie zaszkodzi, aby były to całe obiekty **NodeState**. Teraz dzięki pętli while, powinniśmy sprawdzać kolejne węzły, dopóki nasz stos nie będzie pusty. Jeżeli ostatnio dodany do stosu obiekt posiada co najmniej jednego sąsiada, to pobieramy ich listę, przy użyciu funkcji losującej i wybieramy dany element z posiadanej listy. Dodajemy do stosu wylosowany obiekt. Następnie powinniśmy usunąć każdą odwiedzoną ścianę.

Wyciąg 3-11. Usunięcie ściany - skrypt C#

```
1 | node[x ,y] &= ~neighbour.SharedWall;
```

Gdzie **SharedWall** mówi nam o tym, czy pomiędzy dwoma obiektami znajduje się ściana. Oczywiście, informacja o ścianie powinna zostać zaktualizowana również u sąsiada. W tej sytuacji są dwa wyjścia. Pierwszy z nich, z lekka naiwny i niezbyt przyjazne dla oka dewelopera to stworzenie czterech warunków, które definiują dla danego przypadku jak pobrać pozycję przeciwną, dla sąsiada po

lewej stronie - prawo, dla sąsiada na dole - góra i tak dalej. W ten sposób możemy zdobyć węzeł i zmienić jego stan, a następnie dodać nowy element do stosu. Drugim, rozsądniejszym pomysłem jest zadeklarowanie metody, która mogłaby od razu zwrócić nam **NodeState** naszego sąsiada, który od razu mógłby być szybko poprawiony. Z pomocą przychodzi tutaj prosta instrukcja `switch()`, która pozwala nam na zarządzanie czterema przypadkami, z którymi będziemy mieli do czynienia.

Wyciąg 3-12. Zwracanie stanu sąsiada - skrypt C#

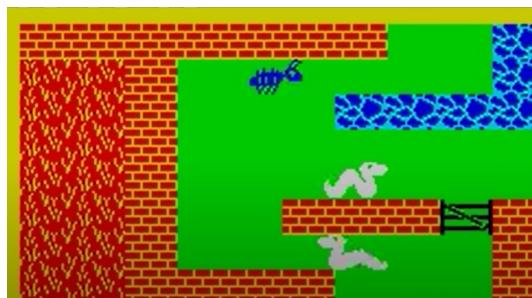
```
1 | switch(nodeState)
2 | {
3 |     case NodeState.RIGHT = return NodeState.LEFT;
4 |     case NodeState.LEFT = return NodeState.RIGHT;
5 |     case NodeState.UP = return NodeState.DOWN;
6 |     case NodeState.DOWN = return NodeState.UP;
7 | }
```

Algorytm gotowy. W tym momencie mamy stworzony generator labiryntu, który powinien jeszcze stworzyć algorytm odpowiedzialny za wygenerowanie w naszej grze labiryntu. Dzięki otrzymanym informacją nie sprawia to większych problemów. Mając informację o wszystkich węzłach, możemy stworzyć pętle, która tworzy obiekty ścian ze względu na ściany istniejące dla danego węzła. Z pomocą przychodzi tu obiekt **Instantiate**. Służy on do klonowania przekazanego obiektu (dla nas ściany). Dodatkowo metoda ta pozwala nam na określenie dokładnej pozycji oraz rotacji naszego elementu, co pozwoli nam na ułożenie ściany w wyznaczonym miejscu i pozwoli na obrócenie obiektu - ze względu czy jest ścianą pionową czy poziomą. Dokładny obiekt który jest nam potrzebny to **Instantiate(Object original, Vector3 position, Quaternion rotation)**. [40] Teraz używając czterech warunków, dla każdej strony, weryfikujemy, czy nasz węzeł sąsiaduje z daną ścianą. Dla każdej ściany (jeśli istnieje) tworzymy jej obiekt. Następnie powinniśmy określić pozycję - tutaj przychodzi z pomocą obiekt **Vector3**. Jest to trójelementowa struktura, która przydaje się w określaniu między innymi pozycji w trzech osiach - **Vector3(x, y, z)**. [41]. Pozycja jest tutaj zależna od naszych wymagań - powinniśmy dopasować ją do wielkości naszego obiektu. Z pomocą przychodzi Unity, które w sekcji Inspector pozwala łatwo zarządzać wielkością naszego obiektu. Rotacja potrzebuje zaś już czteroelementowego obiektu - **Quaternion**. Ze względu na posiadany przez nas obiekt ściany musimy sami określić, która oś powinna zostać zmieniona. W ten sposób powinniśmy mieć już widocznie wygenerowany labirynt.

3.5.3. Przedstawienie przykładowu w innych grach

Wiggler

„Wiggler” jest grą wydaną w 1985 roku. W grze sterujemy robakiem, którego zadaniem jest uciec z labiryntu składającego się z 256 poziomów, w ramach wyścigu w którym bierze udział. Każdy poziom (labirynt) jest stały - nie są one generowane w czasie gry. Gra została stworzona przez braci Kempthorne oraz David'a Vivian'a. [42]



Rysunek 3.21. Labirynt. Przykład z gry Wiggler

Dandy

Kolejną grą opartą na labiryntach jest **Dandy**. Gra powstała w 1983 roku na platformy Atari, przez Johna Howarda Palevicha, w ramach pracy licencjackiej na MIT. Gracz steruje postacią, poruszając się po lochach z wieloma poziomami, które połączone są ze sobą schodami. Niektóre fragmenty labiryntu zamknięte są przez drzwi, do których otwarcia potrzebne są klucze, rozsiane po planszy. Oprócz pokonywania kolejnych poziomów, gracz może pokonywać wrogów przy użyciu łuku. [43]

3.6. PODSUMOWANIE

Jak widać, projektowanie zagadek logicznych jako mini-gry może nie być aż tak trudne. Dużo zależy od podejścia autora i kreatywności w przewidywaniu szczególnych sytuacji - czego przykładem jest kwestia nieroziwiązywalnych przesuwanych puzzli. Pomocne jest tu też narzędzie Unity i zawarte w nim funkcjonalności. Dzięki odpowiednim dopasowaniom komponentów do obiektów, możemy łatwo podczas podglądu gry, testować produkt pod kątem potencjalnych błędów. Na pewno wielkim plusem jest tu też szeroka dokumentacja Unity, która pozwala nam łatwo znaleźć funkcje, czy obiekty, które mogą się przydać w toku tworzenia własnej rozgrywki.

ROZDZIAŁ 4

Fizyka postaci

4.1. TWORZENIE POSTACI

4.1.1. Omówienie zagadnienia procesu tworzenia postaci

Zagadnienie procesu tworzenia postaci polega na przygotowaniu grafik w odpowiedni sposób tak, żeby można było swobodnie wprowadzić i pracować na projekcie postaci w Unity, a także stworzenie odpowiednich komponentów przygotowanych bezpośrednio pod animowanie odpowiednich kończyn jak i późniejszej pracy na kolizjach. Chodzi więc o wszystkie aspekty przygotowania graficznego postaci jak i późniejszego wprowadzenia jej do projektu w taki sposób, żeby można było tworzyć na niej odpowiednie kolizje, animacje i skrypty odpowiadające za poruszanie się.

4.1.2. Proces graficzny tworzenia postaci

Tworzenie postaci zawsze zaczyna się od projektu graficznego, który jest kluczowym punktem do późniejszego animowania postaci. W tym miejscu warto zastanowić się nad ograniczeniami i korzyściami jakie daje nam stworzenie projektu w odpowiedni sposób. Istnieją 2 najbardziej popularne sposoby tworzenia graficznego postaci, które pozwalają nam na animowanie jej: Statyczny bohater, będący bazą do tworzenia animacji poklatkowych, które później używane są bezpośrednio w projekcie. Główną zaletą takiego rozwiązania są bardzo dobrze wyglądające animacje, wynikające z braku ograniczeń ilości zmian w postaci w trakcie animowania. Proces ten wymaga jednak bardzo dużo czasu poświęconego na modelowanie postaci w konkretnych pozach, symulowanie konkretnych

ruchów oraz tworzenie zmian w postaci takich jak np. poruszające się włosy przy skoku.

Alternatywą dla tego rozwiązania jest użycie kinematyki odwrotnej, która bazuje na systemie kości w ciele postaci, połączeń i ograniczeń ich ruchów. Spособ ten jest dużo bardziej wydajny czasowo, nie wymaga dużych umiejętności w animowaniu i pozwala na wprowadzanie nowych animacji wynikających z rozwoju projektu wewnątrz środowiska Unity. Animacja stworzona przez użycie kinematyki odwrotnej jest wykonywana w czasie rzeczywistym, przez co nie musimy tworzyć odpowiedniej ilości klatek, żeby wyglądała płynnie (ma tyle klatek ile gra w konkretnym momencie wykonywania animacji). W praktyce przekłada się to na konieczność tworzenia grafiki w odpowiedni sposób. Musimy podzielić postać na osobne warstwy poszczególnie odpowiadające za ilość kończyn, które powinny się ruszać w końcowym projekcie.



Rysunek 4.1. Projekt graficzny postaci w edytorze Gimp

Postać w naszym projekcie jest podzielona odpowiednio na głowę, tors, ręce i nogi. Ilość możliwych komponentów nie jest ograniczona. Tak przygotowany projekt graficzny w formacie PSB jest gotowy do importu w środowisku Unity, a poszczególne warstwy są widoczne jako oddzielne komponenty. Tym samym projekt jest gotowy do dalszej pracy.

4.1.3. Wdrożenie postaci do projektu w Unity

W celu zimportowania postaci w Unity z projektu graficznego potrzebujemy biblioteki o nazwie *PSDIImporter*, która pozwala na bezproblemowe wdrożenie projektu w tym formacie jako gotowy do pracy *sprite. W kolejnym etapie musimy nadać postaci kości odpowiadające konkretnym kończynom oraz geometrię postaci.



Rysunek 4.2. Widok postaci w środowisku Unity - kości

Geometria postaci to siatka, która definiuje to, w jaki sposób postać może się zginać w trakcie animowania, a kości powodują ograniczenie ruchów w taki sposób, aby były one naturalne jednocześnie łącząc ze sobą wcześniej powstałe warstwy. W celu uniknięcia przesuwania się torsu w trakcie poruszania rękami lub nogami należy wykluczyć zależność tych kości między sobą. W praktyce poruszając ręką nie ruszamy przy tym całym kręgosłupem. Następnym i ostatnim krokiem wdrożeniowym jest zapisanie zmian i przeniesienie wygenerowanej postaci do sceny oraz dodanie do każdej kończyny postaci kinematyki odwrotnej. Tak spreparowana postać jest gotowa do pracy w animatorze Unity.

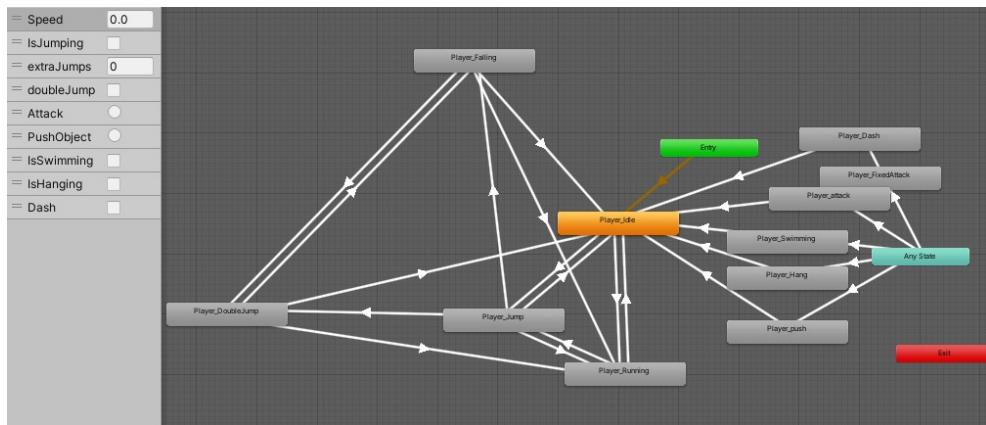
4.1.4. Sposoby animowania postaci

Głównym aspektem animacji jest stworzenie iluzji. Najlepiej postrzegane animacje to te, które w dobry sposób odwzorowują żywą postać.

*Sprite - Dwuwymiarowy obrazek używany w systemach grafiki komputerowej

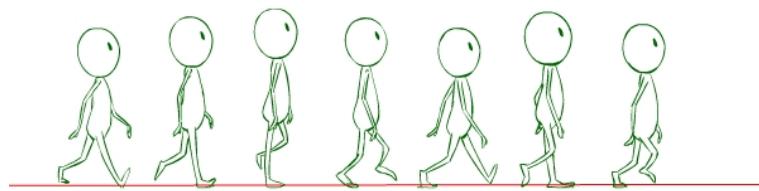
Dlatego więc głównym zadaniem animatora jest przestrzeganie zasad animacji w taki sposób, aby finalnie ruch postaci był jak najbardziej zbliżony do naturalnego ruchu człowieka. Zadanie to niejednokrotnie jest utrudnione, ponieważ mówimy jednak o wykreowanym świecie i narysowanych w programie postaciach, jednak zważając na wcześniej przygotowany system kości jesteśmy w stanie przynajmniej podobnie odwzorować sam ruch postaci. Proces animowania rozpoczynamy od stworzenia animatora, czyli obiektu odpowiedzialnego za obsługę i włączanie konkretnych animacji w danym momencie niejednokrotnie po spełnieniu warunków przejścia do konkretnej animacji. [45]

W naszym projekcie taki animator prezentuje się następująco:



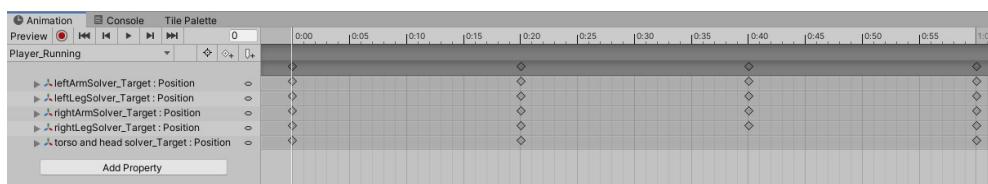
Rysunek 4.3. Widok animatora projektu „The Lore”

Po lewej stronie widoczne są warunki, które podpięte są do konkretnych stanów przejścia między animacjami, np. wartość zmiennej *IsJumping* służy do określenia, kiedy możemy przejść do animacji skoku. Animacje dodajemy do animatora poprzez stworzenie nowego pliku animacji, cały proces przebiega już na tym pliku. W oknie pojawia nam się oś czasu na której możemy zdefiniować kolejne etapy animacji.



Rysunek 4.4. Animacja chodzenia - przykład ze strony design.tutsplus.com [46]

W tym momencie korzystamy ze wcześniejszego przygotowania postaci. Animacja poklatkowa polega na przedstawieniu ruchu klatka po klatce, w praktyce dzięki animatorowi możemy zrobić przejścia tylko między bardziej istotnymi etapami animacji. Osobno animowany jest ruch ciała, głowy, kończyn, a wszystko zamiast kilkuset zdjęć sprowadza się do kilku stanów, które automatycznie przechodzą między sobą w naturalny sposób.



Rysunek 4.5. Animacja biegania - widok z projektu „The Lore”

Animacja biegania w naszym projekcie posiada tylko 4 stany, z czego ostatni stan jest taki sam jak pierwszy w celu zapętlenia animacji. W praktyce więc animowanie w Unity to próba przedstawienia w jak najbardziej rzeczywisty sposób czynności, które występują w świecie codziennym w taki sposób, żeby stworzyć iluzję realizmu. Jeśli jednak czynność nie występuje w realnym świecie jak np. umiejętności animacja jest robiona w taki sposób, żeby wyglądała jak najbardziej naturalnie w swojej abstrakcji.

4.2. PORUSZANIE SIĘ

4.2.1. Omówienie zagadnienia

Zagadnienie poruszania się dotyczy całego procesu skryptowego postaci. Elementy potrzebne do przygotowania postaci do środowiska deweloperskiego jak i wszelkie aspekty wprowadzenia postaci w ruch oraz jej reakcje na wejście wprowadzane przez użytkownika. Przedstawione zostaną problemy związane z tworzeniem poszczególnych funkcjonalności oraz sposoby ich rozwiązania. Zostanie również przedstawiony szczegółowy proces implementacji poszczególnych funkcjonalności jak i sposoby ich wdrożenia. W tym rozdziale również zostanie przedstawiona fizyka postaci oraz poszczególne rzeczy, które pokrywają się z fizyką świata rzeczywistego, a które od takiej fizyki odbiegają.

4.2.2. Fizyka w grach 2D, a w prawdziwym świecie

Pojęcie fizyki w grach składa się na dwa główne komponenty, jednym z nich jest silnik fizyki, natomiast drugim silnik kolizji. Pomimo, że silnik fizyki jest zależny od silnika kolizji nie zajmują się one tym samym. Silnik kolizji dzieli się na dwie części: detekcja kolizji oraz jej odpowiedzialność, która jest częścią silnika fizyki, z tego faktu oba te silniki przeważnie występują w jednej bibliotece. Wykrywanie kolizji występuje w dwóch formach: ciągłej i oddzielnej. Ogólnie używanie ciągłej detekcji jest bardzo kosztowne w zasoby oraz może skutkować dużymi spadkami wydajnościowymi, dlatego w grach staramy się używać jej tylko wtedy kiedy naprawdę jest potrzebna. W większości wypadków sprowadza się to do tego, że zamiast zatrzymywać użytkownika przed wbiegnięciem w ścianę, jego pozycja jest zmieniana już kiedy w te ścianę częściowo wbiegne, podobnie jest z podłożem. W prawdziwym świecie naturalnie nie istnieje możliwość wbiegnięcia w ścianę i ponownej teleportacji we wcześniejsze miejsce i nie jesteśmy ograniczani sposobami wydajnościowymi. Jeśli chodzi o wykrywanie kolizji z wykorzystaniem ruchomych obiektów, początków łamigłówek, kolizji z podłożem w celu możliwości skoku czy kontaktu z przeciwnikiem, używamy do tego pustego obiektu usytuowanego w postaci w celu sprawdzania kolizji od tego obiektu. Często wysunięty jest on w konkretnym kierunku, w którym w danym momencie musimy sprawdzać konkretne kolizje. Sprawdzanie kolizji to nic innego jak stworzenie otoczenia wokół naszego obiektu w postaci człowieka (wielkość takiego koła często zależy od konkretnej sytuacji do której chcemy go

użyć), a następnie sprawdzenie czy kolizja tego obiektu nie nachodzi na żaden inny obiekt z otoczenia. [47]

Wyciąg 4-1. Fragment klasy CharacterController.cs

```
1 | for (int i = 0; i < colliders.Length; i++)
2 | {
3 |     if (colliders[i].gameObject.name.Contains("Moving Platform")) {
4 |         onMovingPlatform = true;
5 |         currentPlatform =
6 |             = colliders[i].gameObject.GetComponent<MovingPlatform>();
7 |     }
8 |
9 |     if (colliders[i].gameObject != gameObject)
10 |    {
11 |        m_Grounded = true;
12 |    }
13 |
14 |     if (!wasGrounded)
15 |    {
16 |        animator.SetBool("IsJumping", false);
17 |    }
18 |
19 | }
```

W skrypcie jest sprawdzana konkretna warstwa lub konkretny obiekt, z którym oczekujemy konkretnego działania po kolizji. Ta część kodu odpowiada za sprawdzenie, czy postać dotyka ziemi w celu ustawienia zmiennej na odpowiadającą jej wartość, a także za sprawdzenie czy postać jest na poruszających się platformach w celu późniejszej zmiany jej rodzica. W tym momencie, kiedy kontakt został odkryty jesteśmy w stanie stwierdzić dokładnie kiedy występuje kolizja między dwoma konkretnymi obiektami i pracować na nich lub ograniczać je zależnie od tego.

4.2.3. Poruszanie się oraz kolizje postaci

Za poruszanie się w projekcie „The Lore” odpowiedzialne są dwie klasy: CharacterController.cs i PlayerMovement.cs. Ta pierwsza odpowiedzialna jest za sam ruch, skok i wyświetlanie postaci w dobrym kierunku, natomiast druga zajmuje się całą resztą od przesuwania przedmiotów po używanie umiejętności. Ze względu na to, że w projekcie użyty jest model bryły sztywnej Unity otwiera

nam wiele ścieżek pracy nad postacią. Aby wprowadzić postać w ruch musimy znać kierunek w jakim chcemy, żeby się poruszyła oraz docelową prędkość.

Wyciąg 4-2. Fragment klasy CharacterController.cs

```

1 | Vector3 targetVelocity = new Vector2(move * 10f, m_Rigidbody2D.velocity.y);
2 |
3 | m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity, targetVelocity,
4 |     ref m_Velocity, m_MovementSmoothing);

```

Zmienna *targetVelocity* odpowiedzialna jest za określenie konkretnego ruchu, który chcemy, żeby wykonała postać. Warto zauważać jednak, że końcowy wynik jest jeszcze wygładzany przy pomocy wbudowanej do wektora metody *SmoothDamp* [48], co oznacza, że ruch nie jest sztywny i ma swoje ograniczenia czasowe w trakcie jego wykonywania. W praktyce przenosi się to na bardziej intuicyjny sposób poruszania się jak i poprawę wizualną samego ruchu. Kolejną wcześniej wspomnianą funkcjonalnością, którą zajmuje się ta klasa jest zmienianie kierunku grafiki w zależności od kierunku w którym się poruszamy.

Wyciąg 4-3. Fragment klasy CharacterController.cs

```

1 | private void Flip()
2 | {
3 |     m_FacingRight = !m_FacingRight;
4 |     Vector3 theScale = transform.localScale;
5 |     theScale.x *= -1;
6 |     transform.localScale = theScale;
7 | }
8 | if (move > 0 && !m_FacingRight && !Statics.isOnRope)
9 | {
10 |     Flip();
11 | }
12 | else if (move < 0 && m_FacingRight && !Statics.isOnRope)
13 | {
14 |     Flip();
15 | }

```

Metoda *Flip()* odpowiedzialna jest za zmianę skali lokalnej postaci poprzez pomnożenie jej wartości przez -1 i wykonywana jest po sprawdzeniu wstępnych założeń.

Kolejnym sposobem implementacji ruchu jest odejście od używania wbudowanego modelu bryły sztywnej oraz wartości prędkości do niej przypisanej, zamiast tego używanie przejścia poprzez aktualizację wartości pozycji postaci.

Problemy jakie generuje to rozwiązywanie jednak to konieczność przeliczania prawa tarcia i ruchu oraz przypisywania ich w kodzie i analogicznej zmiany pozycji pokrywającej te prawa. Przez rozbudowany system fizyki w Unity takie rozwiązanie nie ma większego sensu i powoduje konieczność włożenia dodatkowej pracy. Niemniej jednak istnieją sytuacje w których takie rozwiązanie sprawdza się lepiej jak np. symulacja bujania się na linach czy nawet sam skok. Jednymi z głównych kolizji odpowiedzialnych za sam etap poruszania się są kolizje z podłożem oraz ze ścianami (ze względu na wymiar gry kolizja z sufitem nie jest brana pod uwagę). Podobnie jak zostało wspomniane wcześniej kolizje te definiują kiedy możemy przejść dalej oraz kiedy postać posiada możliwość skoku, która odnawia się za każdym razem, kiedy postać ponownie dotknie podłożu. Warto pamiętać, żeby oddzielić warstwy podłożu od warstw innych obiektów posiadających kolizje, gdyż może to przynieść niechciane efekty w postaci zwiększonej ilości skoków w obszarze obiektu posiadającego kolizje.

4.2.4. Skakanie

Skok jest kluczowym elementem w grach, niezależnie od tego czy mowa o grach platformowych czy nawet MMO-RPG skakanie jest nie tylko jednym lub jednym z głównych sposobów dostania się na wyższe poziomy ale również główną mechaniką dotyczącą przemieszczania się. Niektórzy użytkownicy preferują ciągły skok jako alternatywę dla biegania tylko dlatego, że wydaje się ona szybsza. Większość gier obala jednak tezę, że skok jest szybszy niż bieg ale są też takie gry, które specjalnie implementują taką mechanikę, jednym z takich przykładów jest Counter Strike: Global Offensive, gdzie poprzez umiejętne wykorzystanie skoku można się poruszać kilkukrotnie szybciej niż biegnąc. Skakanie jest preferowaną metodą poruszania się też ze względu na większą ilość możliwości ruchu w trakcie skoku. Czym tak naprawdę jest skok w grach 2D i jak dużo wspólnego ma z rzeczywistą fizyką? Bardzo dużo gier symuluje skok jak najbardziej zbliżony do rzeczywistej fizyki, istnieją jednak gry, które stawiają na nieco inne rozwiązanie. Dobrym przykładem jest znany tytuł gry platformowej Mario, ale również takie gry jak Limbo, Cuphead, speedrunners gdzie podczas skoku stawia się na nieco inne rozwiązanie. [49]



Rysunek 4.6. Przykład skoku z gry Super Mario Maker 2 [51]

Jak można zauważyć na obrazku, skok nie jest parabolą, tak jak byłoby to w przypadku realistycznej sytuacji. W niektórych grach 2D tworzy się symulację skoku polegającą na tym, że im dłużej trzyma się przycisk skoku tym dłużej jest się w powietrzu. W rzeczywistości polega to na ustaleniu czasu, w którym siła może działać na postać i ograniczenia jej górnego limitu. W praktyce wygląda to tak, że kiedy tylko przestaniemy trzymać przycisk skoku postać automatycznie zacznie lecieć w dół, a jeśli go przytrzymamy postać na chwilę zawiśnie w powietrzu. Stało się to na tyle powszechnie, że taka symulacja skoku wydaje się bardziej naturalna niż dodanie do niego wertykalnej siły jednorazowo, czyli zgodnie z prawami fizyki. [50]

W naszym projekcie postawiliśmy jednak na rzeczywistą symulację fizyki. Stworzenie skoku w taki sposób pozwala na większą przewidywalność miejsca lądowania, natomiast na mniejszą kontrolę samego skoku. Sama implementacja skoku w projekcie The Lore prezentuje się następująco:

Wyciąg 4-4. Fragment klasy CharacterController.cs

```

1  if (_Grounded && jump){
2      _Grounded = false;
3      m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce));
4      jump = false;
5      onMovingPlatform = false;
6  }
7  if (!_Grounded && jump && extraJumps > 0 && !Statics.isOnRope)
8  {
9      m_Rigidbody2D.velocity = new Vector2(m_Rigidbody2D.velocity.x, 0);
10     animator.SetBool("doubleJump", true);
11     m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce * extraJumpDifficulty));
12     jump = false;
13     extraJumps--;
14 }
```

Obsługa skoku w naszym projekcie polega zatem na jednorazowym dodaniu siły do postaci w trakcie skoku. Sprawa jest nieco bardziej skomplikowana jeśli chodzi o wielokrotny skok, ze względu na jednokrotne dodanie siły, zanim spróbujemy skoczyć drugi raz aktualna prędkość wertykalna musi zostać ustaliona na 0 ze względu na to, że skok dodaje wartość siły do aktualnej wartości działającej na postać, w trakcie wielokrotnego skoku bez ustawienia tej wartości na 0 przyczyniłoby się to do spowolnienia, a nie wykonania kolejnego skoku.

4.2.5. Otoczenie wpływające na fizykę postaci

W grach niejednokrotnie pojawiają się obiekty oddziałujące na fizykę naszej postaci. Ich symulacja często wiąże się ze słabym odwzorowaniem rzeczywistej fizyki. W tym podrozdziale głównym narzędziem, z którego korzystamy będzie wcześniej wspomniane wykrywanie kolizji między konkretnymi obiektami. Jeśli chcemy, żeby obiekt lub nasza postać zachowywała się w konkretny sposób po kontakcie ze sobą trzeba ten kontakt i konkretny obiekt najpierw wykryć. Dobre nazewnictwo warstw i obiektów przydaje się nie tylko do symulowania samej fizyki ale również do łatwego nawigowania po animacjach, a nawet włączania samej animacji z poziomu skryptu, a nie animatora. Na samym początku warto spojrzeć na skrypt służący do przesuwania skrzyń.

Wyciąg 4-5. Fragment klasy PlayerMovement.cs

```
1 void MoveObject()
2 {
3     if (Physics2D.OverlapCircleAll(actionPoint.position, boxMoveRange,
4         boxLayer).Length > 0 && !animator.GetBool("IsJumping"))
5     {
6         Physics2D.OverlapCircleAll(actionPoint.position, boxMoveRange,
7             boxLayer)[0].GetComponent<BoxMoving>()
8             .MoveBox(direction);
9             animator.SetTrigger("PushObject");
10    }
11 }
```

Jest to bardzo prosty skrypt, w którym jedyne siły dochodzące do dotychczas oddziałujących to waga skrzyń, które chcemy przesunąć. To właśnie ciężar skrzyń sprawia, że postać po zbliżeniu się do skrzyni i próbie jej przesunięcia znacznie spowalnia i skutkuje to ruchem skrzyni, której fizyka jest symulowana poprzez model bryły sztywnej dodany bezpośrednio do skrzyni. W naszym projekcie są jeszcze 2 obiekty oddziałujące na fizykę postaci, pierwszym z nich

są ruchome platformy. Mimo, że symulacja samego stania na platformie jest uproszczona ze względu na grywalność i jest ona niczym innym jak dodaniem platformy jako rodzica naszej postaci, co skutkuje tym, że platforma nie porusza się w kontekście postaci (ruch na platformie jest taki sam jak na ziemi) to dodana została siła poruszającej się platformy przy samym wyskoku. Jeśli obiekt, na którym się znajduje się postać jest w ruchu, to po wyskoku do góry postać nasza powinna poruszać się w tym samym kierunku i z taką samą prędkością co platforma w trakcie wyskoku. Dlatego skok w którakolwiek stronę będzie wypadkową siły skoku oraz prędkością poruszającej się aktualnie platformy.

Wyciąg 4-6. Fragment klasy PlayerMovement.cs

```
1 | if (onMovingPlatform && !m_Grounded)
2 | {
3 |     move += currentPlatform.velocityX;
4 |     onMovingPlatform = false;
5 | }
```

Przez to, że aktualna prędkość przyjmuje wartości ujemne, gdy porusza się w lewą stronę, a dodatnie gdy porusza się w prawą implementacja tego problemu ogranicza się do kilku linijek. Kolejnym obiektem oddziałującym na fizykę postaci w grze „The Lore” są liny. Mechanika lin polega na tym, że jeśli postać znajduje się wystarczająco blisko odpowiedniego obiektu oraz przytrzyma przycisk chwytu, przestanie spadać i zawiśnięcie na linie, ponadto w takiej sytuacji może poruszać całą linią i bujać się w celu przeskoczenia dalej. Problem ten jest dużo bardziej złożony, samo symulowanie lin w Unity działa bardzo dobrze, aczkolwiek dużo bardziej powszechnym rozwiązaniem jest ręczna symulacja i stworzenie do tego problemu animacji poklatkowej lub ewentualna zmiana pozycji postaci w taki sposób, aby uniknąć konkretnego usytuowania gracza względem liny i tym samym niepożądanego efektu. Problem ten w projekcie „The Lore” został rozwiązany poprzez przesunięcie obiektu szukającego kolizji z linami na przód gracza i zmniejszenie go w taki sposób, aby pole, w którym może się złapać gracz było małe, co likwiduje potencjalne problemy z niechcianym chwytem. Rozwiązanie takie sprawia, że mechanika jest poniekąd trudniejsza do ukończenia. Po testach użytkowników okazało się jednak, że wciąż nie jest ona wystarczająco trudna, dlatego usunięty został jeden z możliwych dodatkowych skoków po zeskokieniu z liny.

Wyciąg 4-7. Fragment klasy PlayerMovement.cs

```
1 void Grab()
2 {
3     Collider2D[] colliders = Physics2D.OverlapCircleAll(actionPoint.position,
4         attackRange, ropeLayer);
5     if (colliders.Length > 0 && !Statics.isOnRope)
6     {
7         recentCollider = colliders[0];
8         FixedJoint2D joint = colliders[0].gameObject.AddComponent<FixedJoint2D>();
9         joint.connectedBody = gameObject.GetComponent<Rigidbody2D>();
10        isOnRope = true;
11        Statics.isOnRope = true;
12        controller.extraJumps = 1;
13    }
14 }
```

Problem trzymania się liny został rozwiązyany przy użyciu połączenia stałego między postacią, a jednym z obiektów lin, ponieważ lina sama w sobie składa się z kilkunastu fragmentów połączonych, aby uniknąć błędów przy połączeniu brana jest pod uwagę tylko jedna z wykrytych przez skrypt kolizji. Warto zaznaczyć też, że bujanie liny jest możliwe ze względu na możliwość ruchu postaci w powietrzu, gdyby nie to trzeba byłaby zmodyfikować klasę dotyczącą poruszania się tak, aby ruch był możliwy również w trakcie kontaktu z linią. Połączenie stałe między postacią a linią zostaje zerwane w momencie puszczenia przycisku akcji, po czym użytkownikowi przysługuje jeszcze jeden skok.

4.3. UMIEJĘTNOŚCI POSTACI

4.3.1. Omówienie zagadnienia umiejętności

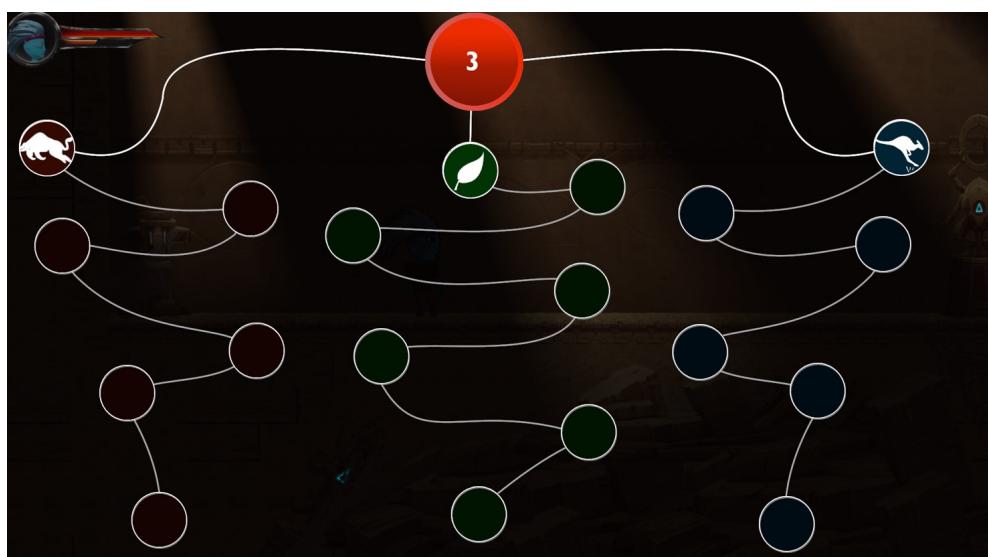
Umiejętności postaci to nowe mechaniki wprowadzone do gry w celu zarówno zwiększenia dynamiki rozgrywki jak i ulepszenia doznań użytkownika. Sekcja ta poświęcona będzie zarówno projekcie i mechanikach wewnętrz całego drzewka całego drzewka umiejętności jak i całemu projektowaniu i problemach związanych z poszczególnymi umiejętnościami. Przedstawione zostanie całe przygotowanie graficzne drzewka jak i różne sposoby na implementację tych samych umiejętności.

4.3.2. Drzewko umiejętności

Drzewko umiejętności to dodatkowa mechanika w grze pozwalająca użytkownikowi na rozwijanie poszczególnych gałęzi według własnych upodobań. To w tym miejscu gracz jest w stanie samemu określić w jaki sposób projektować będzie rozwój postaci oraz w jaki sposób przebiegać będzie jego rozgrywka. W celu zwiększenia dynamiki rozgrywki można postawić głównie na umiejętności związane z poruszaniem się, jeśli jednak użytkownik jest bardziej zapobiegawczym typem gracza bardziej prawdopodobnie będzie celował w przetrwanie i siłę.

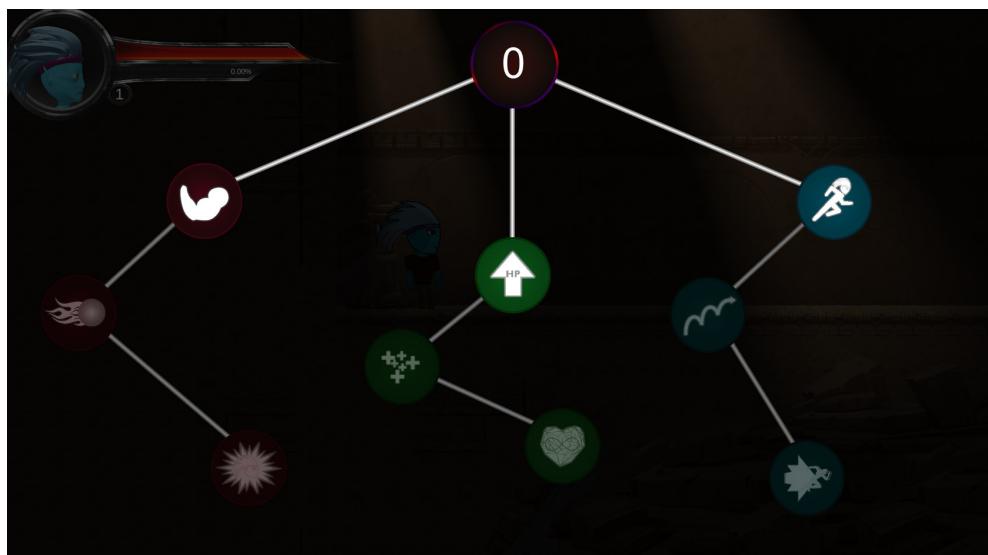
Wstępny prototyp drzewka, a widok końcowy

Drzewko umiejętności w projekcie „The Lore” w fazie prototypowania prezentowało się następująco:



Rysunek 4.7. Prototyp drzewka w projekcie „The Lore”

Jak się jednak okazało w kolejnych fazach pracy i testach użytkowników taka ilość umiejętności jest nieproporcjonalna do ilości treści i możliwości zdobytych poziomów w grze, dlatego w końcowej iteracji drzewo zostało ograniczone do 3 umiejętności na jedną gałąź. Widok końcowy bazujący na prototypie wygląda następująco:



Rysunek 4.8. Aktualne drzewko umiejętności w projekcie „The Lore”

Górny licznik wyświetla, ile umiejętności jesteśmy w stanie w danym momencie rozwinać tzn. ile dostępnych punktów umiejętności jest aktualnie odblokowanych. Liczba ta jest przeliczana w następujący sposób:

Wyciąg 4-8. Fragment klasy UI_SkillTree.cs

```
1 | text = (GLOBAL_DATA.Instance.Level - 1 - unlockedSkills)
```

W praktyce bez odblokowanej żadnej umiejętności mamy zawsze o jeden mniej punkt umiejętności niż aktualny poziom postaci. Przez to wystarczy odjąć od tego jeszcze długość listy obiektów typu **SkillButton** i końcowy wynik pokazuje aktualną ilość punktów. Podobne sprawdzenie używane jest w klasie **PlayerSkills.cs**, w celu weryfikacji czy użytkownik może daną umiejętność rozwinać. Kolejnym ważnym punktem jeśli chodzi o sprawdzanie poszczególnych umiejętności jest ich hierarchia, czyli odblokowywanie umiejętności z danej gałęzi w odpowiedniej kolejności. Tym zadaniem zajmuje się następujący fragment kodu:

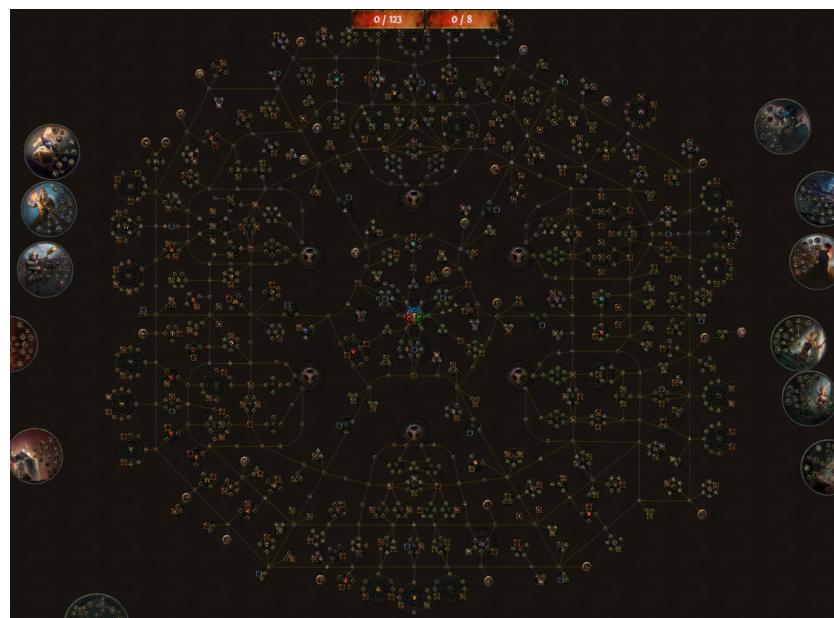
Wyciąg 4-9. Fragment klasy PlayerSkills.cs

```
1  public SkillType GetSkillRequirement(SkillType skillType) {  
2      switch (skillType) {  
3          //Blue Skills Requirements  
4          case SkillType.TripleJump: return SkillType.Sprint;  
5          case SkillType.Dash: return SkillType.TripleJump;  
6          //Red Skills Requirements  
7          case SkillType.Bullet: return SkillType.IncreaseDMG;  
8          case SkillType.Explode: return SkillType.Bullet;  
9          //Green Skills Requirements  
10         case SkillType.RegenerationHP: return SkillType.ExtraHP;  
11         case SkillType.Immortality: return SkillType.RegenerationHP;  
12     }  
13     return SkillType.None;  
14 }
```

Zwracana wcześniejsza umiejętność przy próbie odblokowania pozwala nam określić, która umiejętność jest warunkiem do odblokowania danej umiejętności, co znacznie ułatwia sprawdzanie, wystarczy przeszukać listę odblokowanych umiejętności i sprawdzić, czy wcześniejsza w hierarchii istnieje w danym kontekście.

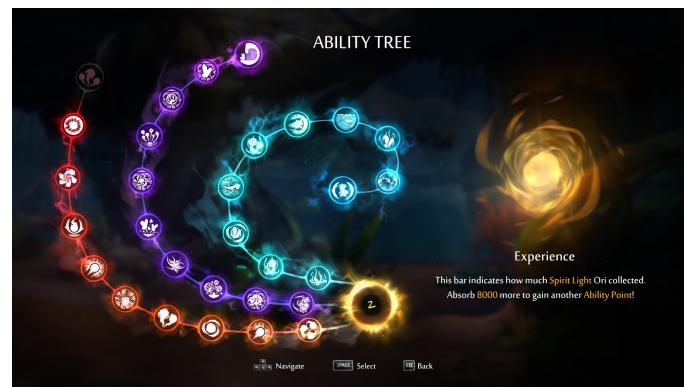
Przykłady z innych gier

Jest wiele gier, które wykorzystują drzewka umiejętności w wielu celach, niejednokrotnie jest to tylko po to, żeby ułatwić użytkownikowi rozgrywkę. Istnieją jednak takie gry, których podejście do drzewek umiejętności jest inne, są implementowane po to, żeby gracz poradził sobie na wyższych poziomach trudności lub nawet są mechaniką bez której samej gry nie jesteśmy w stanie przejść. Dobrym przykładem takiej gry jest Path of Exile w której liczba ścieżek w drzewku, które możemy wybrać jest naprawdę wielka, a samo drzewko prezentuje się następująco:



Rysunek 4.9. Drzewko Umiejętności z gry Path of Exile [52]

W naszym projekcie jednak skupiliśmy się na trochę prostszym modelu drzewka ze względu na ograniczoną liczbę poziomów oraz możliwości ich zdobycia, dobrym przykładem podobnego drzewka umiejętności jest gra Ori And The Blind Forest, w której drzewko to wygląda następująco:



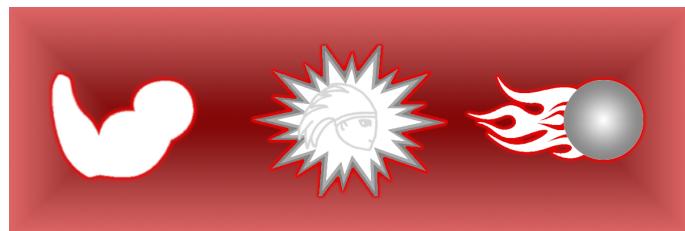
Rysunek 4.10. Drzewko Umiejętności z gry Ori And The Blind Forest

4.3.3. Umiejętności

W tej sekcji skupimy się głównie na umiejętnościach zaimplementowanych w projekcie, problemach(jeśli owe istniały) i innym sposobie rozwiązania tych samych problemów w trakcie tworzenia umiejętności.

Umiejętności siły

Zwiększenie mocy – odpowiedzialne za zwiększenie siły ataku podstawowego do 130% - 250% przez 2 sekundy. Wartość z tego przedziału jest losowana z każdym użyciem umiejętności. Eksplozja – Postać ładuje obszar dookoła siebie o określonym promieniu i zadaje obrażenia każdemu wrogowi znajdującemu się w tym obszarze, dodany do tego został system cząstek dla lepszej wizualizacji efektu. Strzał z kuli – Postać wystrzeliwuje horyzontalnie kulę leczącą ze stałą prędkością w kierunku, w którym aktualnie patrzy się postać, umiejętność ta jest zrobiona ciągłym aktualizowaniem pozycji kulki o daną, stałą wartość i zadawanie obrażeń każdemu przeciwnikowi, którego napotka na swojej ścieżce. Kolejnym sposobem rozwiązymania tego problemu jest stworzenie kuli jako bryły sztywnej oraz symulowanie trajektorii lotu kuli. Na potrzeby projektu oraz w danym kontekście oczekiwany rezultatem był jednak lot po prostej, poziomej drodze, dlatego zdecydowaliśmy się na pierwsze rozwiązanie.



Rysunek 4.11. Ikony umiejętności siły z projektu „The Lore”

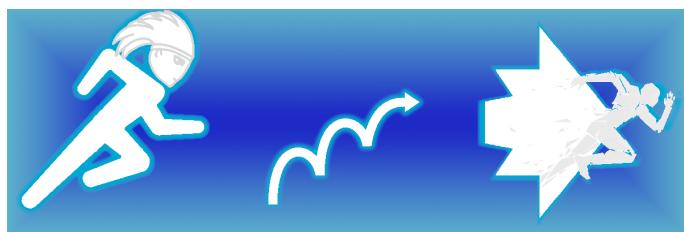
Umiejętności poruszania się

Umiejętności poruszania się służą głównie do łatwiejszego/płynniejszego przemieszczania się po planszy jak i większej opcjonalności przy unikaniu ataków przeciwników, 3 umiejętności dotyczące poruszania się w naszym projekcie to: Sprint – Polega na zwiększeniu prędkości gracza w trakcie trzymania klawisza, mechanikę tę możemy kojarzyć między innymi z takich gier jak Super Mario Bros. Potrójny skok – Polega na możliwości wykonania dodatkowego skoku w trakcie lotu, domyślna wartość to dwa. Umiejętność ta pozwala na doskoczenie we wcześniej niedostępne miejsca, a także na większą swobodę w locie Pęd – Trzecia i ostatnia umiejętność z gałęzi poruszania się polega na bardzo szybkim

przemieszczeniu się horyzontalnym gracza. Jest to często wykorzystywana mechanika w grach platformowych takich jak np. Ori And The Blind Forest. Istnieje kilka sposobów rozwiązyania tego problemu, najłatwiejszym z nich jest zamiana pozycji postaci, gdy tylko jest to możliwe, generuje to jednak dużo problemów związanych z ewentualnymi przeszkodami, które możemy spotkać na swojej drodze jak i z płynnością takiego przejścia. Kolejnym rozwiązańiem, które zostało użyte w naszym projekcie jest przez określona ilość czasu aktualizowanie konkretnej pozycji gracza, ponownie pojawia się tu problem wchodzenia w ściany, natomiast jest on w tym wypadku dużo łatwiejszy do rozwiązania. Przez to, że pozycja nie jest zmieniana jednokrotnie, a małymi krokami przez pewien odstęp czasu jesteśmy w stanie sprawdzać czy istnieje przed postacią jakaś przeszkoda w trakcie wykonywania się skryptu i ewentualne przestanie wykonywania umiejętności w trakcie kiedy ta przeszkoda jest przed postacią.

Wyciąg 4-10. Fragment klasy PlayerSkills.cs

```
1 | if (dashValue < dashForce && dash && !touchingWall)
2 | {
3 |     Dash();
4 | }
```

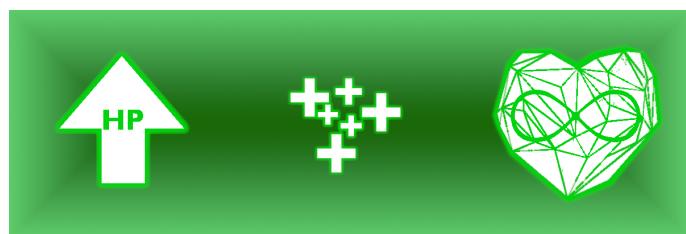


Rysunek 4.12. Ikony umiejętności poruszania się z projektu „The Lore”

Umiejętności przetrwania

Umiejętności przetrwania przydają się wtedy, kiedy bardzo dużo ginimy od przeciwników. Pozwalają one na dużo większą beztroskę w trakcie walki z przeciwnikami jak i na uzupełnianie swojego życia poza nią. W naszym projekcie wyróżniamy takie umiejętności jak: Zwiększenie życia – Jest to umiejętność pasywna zwiększająca na stałe poziom zdrowia gracza ze 100 do 125, co przyczynia się do tego, że jesteśmy w stanie przetrwać więcej ciosów od przeciwników tym samym gra bardziej wybacza nasze błędy. Odnowienie życia – Jest to umiejętność aktywna, w trakcie naciśnięcia przycisku do użycia umiejętności jesteśmy w stanie raz na 30 sekund odnowić swoje życie o 25, przyczynia się

to do spowolnienia rozgrywki jednocześnie dając możliwość odnowienia sobie życia przed lub w trakcie walki z mocniejszymi przeciwnikami. Nieśmiertelność – Jest to umiejętność aktywna pozwalająca nam przez 2 sekundy być odpornymi na każdy atak przeciwnika. Zastosowań tej umiejętności jest mnóstwo. Może ona posłużyć nam do ucieczki przed przeciwnikiem w kryzysowej sytuacji jak i pomóc nam w zadaniu mu dodatkowych ciosów nie tracąc przy tym punktów zdrowia.



Rysunek 4.13. Ikony umiejętności przetrwania z projektu „The Lore”

4.4. PODSUMOWANIE

Podsumowując w rozdziale tym został przedstawiony proces tworzenia postaci, przeniesienia jej do środowiska pracy oraz samą pracę na niej jeśli chodzi o poruszanie się i główne aspekty z tym związane. Warto pamiętać, że nie wszystkie gry odwzorowują fizykę w taki sposób jak wygląda ona rzeczywiście dlatego dobrze jest znać zagrania, którymi kierują się developerzy przy projektowaniu odpowiednich mechanik, a w grach 2D istnieje dużo ukrytych sposobów rozwiązywania problemów i innego przedstawienia fizyki do którego większość użytkowników jest przyzwyczajona przez co dokładna symulacja fizyki dla przeciętnego gracza wydaje się mniej poprawna niż użycie tych sztuczek.

Bibliografia

- [1] Wikipedia, Wolna Encyklopedia. Unity (silnik gry), dostęp w internecie 15.12.2020r.
[https://pl.wikipedia.org/wiki/Unity_\(silnik_gry\)](https://pl.wikipedia.org/wiki/Unity_(silnik_gry))
- [2] Robert K. Wysocki, Rudd McGary: *Efektywne zarządzanie projektami*. Wydanie III, ISBN: 83-7361-861-9, dostęp w internecie 25.01.2021r.
<http://pdf.onepress.pl/efzapr/efzapr-1.pdf>
- [3] CorazLepszyPortalBiznesowy. *Co to jest projekt?*, dostęp w internecie 25.01.2021r.
<https://www.corazlepszyportalbiznesowy.pl/art/co-to-jest-projekt>
- [4] NaRUDO. *Co to jest projekt i dlaczego się nim zarządza?*, dostęp w internecie 25.01.2021r.
<https://narudo.pl/co-to-jest-projekt/>
- [5] The balance careers. *Basic Project Management 101*, dostęp w internecie 26.01.2021r.
<https://www.thebalancecareers.com/project-management-101-2275338>
- [6] Frączkowski K.: *Zarządzanie projektem informatycznym*
Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2003.
ISBN 83-7085-731-0, dostęp w internecie 27.01.2021r
https://wwwdbc.wroc.pl/Content/998/fraczkowski_zarzadzanie_projektem.pdf
- [7] Bernd Bruegge, Allen H. Dutoit: *Inżynieria oprogramowania w ujęciu obiektowym. UML, wzorce projektowe i Java.*
Helion, ISBN: 978 - 83-24 6 -2872-8, dostęp 27.01.2021r.
- [8] Indeed. *5 Phases of the Project Management Life Cycle*, dostęp w internecie 28.01.2021r.
<https://www.indeed.com/career-advice/career-development/phases-of-project-management-life-cycle#3C>
- [9] Devpark. *Testowanie aplikacji – testy integracyjne, jednostkowe i smoke test,*

- dostęp w internecie 28.01.2021r.
<https://devpark.pl/pl/testowanie-aplikacji-testy\integracyjne-jednostkowe-i-smoke-testy/>
- [10] Wikipedia, Wolna Encyklopedia. *Testy integracyjne*, dostęp w internecie 28.01.2021r.
https://pl.wikipedia.org/wiki/Testy_integracyjne
- [11] Kaliop. *Testy obciążeniowe*, dostęp w internecie 28.01.2021r.
<https://www.kaliop.pl/Blog/Artykuly/Testy-obciazeniowe-czyli-jak-weryfikujemy-stabilne-dzialanie-aplikacji-internetowych>
- [12] TESTERKA. *Testy funkcjonalne*, dostęp w internecie 29.01.2021r.
<http://testerka.pl/testy-funkcjonalne/>
- [13] ForProgress. *Testy Akceptacyjne*, dostęp w internecie 29.01.2021r.
<https://forprogress.com.pl/testy-akceptacyjne/>
- [14] NaRUDO. *Zwinne i tradycyjne zarządzanie projektami*. dostęp w internecie 30.01.2021r.
<https://narudo.pl/zwinne-i-tradycyjne-zarzadzanie-projektam/>
- [15] global4net. *Waterfall czy Agile – którą metodę wybrać?*, dostęp w internecie 30.01.2021r.
<https://global4net.com/ecommerce/waterfall-czy-agile-która\metodyke-wybrac/>
- [16] Wikipedia, Wolna Encyklopedia. *Model kaskadowy*, dostęp w internecie 30.01.2021r.
https://pl.wikipedia.org/wiki/Model_kaskadowy
- [17] Agile247. *Czym jest agile?*, dostęp w internecie 31.01.2021r.
<https://agile247.pl/agile/>
- [18] Harvard Business Review. *Embracing Agile*, dostęp w internecie 31.01.2021r.
<https://hbr.org/2016/05/embracing-agile>
- [19] ScienceSoft. *8 Software Development Models: Sliced, Diced and Organized in Charts*. dostęp w internecie 30.01.2021r.
<https://www.scnsoft.com/blog/software-development-models>
- [20] Marionette. *Agile Game Development – A Quick Overview*. dostęp w internecie 01.02.2021r.
<https://marionettestudio.com/agile-game-development-quick-overview/>
- [21] starloop. *Best Agile Practices in Game Development*, dostęp w internecie 01.02.2021r.
<https://starloopstudios.com/best-agile-practices-in-game-development/>
- [22] Agile Hunters. *Role w Scrum*, dostęp w internecie 02.02.2021r.
<https://agilehunters.com/role-w-scrum/>

- [23] LUQAM. *Kanban – czym jest, cele i efekty*, dostęp w internecie 02.02.2021r.
<https://www.luqam.com/kanban-czym-jest-cele-i-zastosowanie/>
- [24] Żurkowski M: *Jira: wszystko, co potrzebujesz wiedzieć*, dostęp w internecie 02.02.2021r.
<https://blog.deviniti.com/pl/atlassian-pl/jira-wszystko-co-potrzebujesz-wiedziec/>
- [25] PRODUCTVISION, *Value Proposition Canvas – unikalna propozycja wartości modelu biznesowego*, dostęp w internecie 03.02.2021r.
<https://productvision.pl/2014/value-proposition-canvas-unikalna-propozycja-wartosci-modelu-biznesowego/>
- [26] PRODUCTVISION, *Business Model Canvas – szablon modelu biznesowego*, dostęp w internecie 03.02.2021r.
<https://productvision.pl/2014/business-model-canvas-szablon-modelu-biznesowego/>
- [27] Wikipedia, Wolna Encyklopedia.", *Zagadka* dostęp 07.01.2021r.
<https://pl.wikipedia.org/wiki/Zagadka>
- [28] Wikipedia, Wolna Encyklopedia.", *Rebus*, dostęp 07.01.2021r.
<https://pl.wikipedia.org/wiki/Rebus>
- [29] Wikipedia, Wolna Encyklopedia.", *15 puzzle*, dostęp 07.01.2021r.
https://en.wikipedia.org/wiki/15_puzzle
- [30] Wikipedia, Wolna Encyklopedia.", *Puzzle*, dostęp 07.01.2021r.
<https://pl.wikipedia.org/wiki/Puzzle>
- [31] Unity Documentation, *Creating Scenes*, dostęp 09.01.2021r.
<https://docs.unity3d.com/Manual/CreatingScenes.html>
- [32] Unity Documentation, *GameObject* dostęp 09.01.2021r.
<https://docs.unity3d.com/ScriptReference/GameObject.html>
- [33] Unity Documentation, *Component* dostęp 28.01.2021r.
<https://docs.unity3d.com/ScriptReference/Component.html>
- [34] Geeks for geeks. Check instance 8 puzzle solvable, dostęp 26.01.2021r.
<https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/>
- [35] Wikipedia, Wolna Encyklopedia." *Pipe Mania*, dostęp 27.01.2021r.
https://en.wikipedia.org/wiki/Pipe_Mania
- [36] Mafia Wiki, Fandom, *Lock Picking*, dostęp 28.01.2021r.
https://mafia.game.fandom.com/wiki/Lock_Picking
- [37] IGN, Assassin's Creed Unity Wiki Guide, *Lockpick*, dostęp 28.01.2021r.
<https://www.ign.com/wikis/assassins-creed-5-unity/Lockpick>
- [38] Qaz Wiki: *Algorytm generowania labiryntu - Maze generation algorithm*, dostęp 02.02.2021r.

- https://pl.qaz.wiki/wiki/Maze_generation_algorithm
- [39] Fandom Wiki, *The Haunting of Castle Malloy*, dostęp 03.02.2021r.
https://nancydrew.fandom.com/wiki/The_Haunting_of_Castle_Malloy
- [40] Unity Documentation, *Instantiate*, dostęp 04.02.2021r.
<https://docs.unity3d.com/ScriptReference/Object.Instantiate.html>
- [41] Unity Documentation, *Vector3*, dostęp 04.02.2021r.
<https://docs.unity3d.com/ScriptReference/Vector3.html>
- [42] Personal Computer News, *Issue 107*, dostęp 04.02.2021r.
http://www.personalcomputernews.co.uk/pcnb/html/107/personal_computer_news_107_\gameplay_spectrum_the_wriggler.html
- [43] "A History of Dandy Dungeon", Jack Palevich. Wersja archiwizowana z 04.11.2013r. Dostęp 04.02.2021 r.
<https://web.archive.org/web/20131104190048/http://jacks-hacks.appspot.com/dandy/\history.html>
- [44] "Soda Pipes", Moby Games. Dostęp 04.02.2021r.
<https://www.mobygames.com/game/windows/soda-pipes>
- [45] "Animation of 2d character in Unity", gamedevacademy. Dostęp 29.01.2021r.
<https://gamedevacademy.org/how-to-animate-a-2d-character-in-unity/>
- [46] "Animation for beginners", tutspplus. Dostęp 01.02.2021r.
<https://design.tutsplus.com/tutorials/animation-for-beginners-how-to-animate-a-character-running--cms-25730>
- [47] "How does a collision engine work", Stack Exchange. Dostęp 04.02.2021r.
<https://gamedev.stackexchange.com/questions/26501/how-does-a-collision-engine-work/26506#26506>
- [48] "SmoothDamp Documentation", Unity3d. Dostęp 05.02.2021r.
<https://docs.unity3d.com/ScriptReference/Vector2.SmoothDamp.html>
- [49] "Jumping in games", DaveTech. Dostęp 07.02.2021r.
<http://www.davetech.co.uk/gamedevplatformer>
- [50] "Begy Jumping", Jason Begy. Dostęp 08.02.2021r.
https://www.jasonbegy.com/uploads/5/0/7/7/50772065/begy_jumping.pdf
- [51] "Lets go physics", Wired. Dostęp 08.02.2021r.
<https://www.wired.com/2016/12/lets-go-physics-jumping-super-mario-run/>
- [52] "Path Of Exile Skill Tree", goldkk. Dostęp 08.02.2021r.
<https://www.goldkk.com/news/1294--path-of-exile-skill-tree-guide--passive-s>