



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
dr Bartłomieja Przybylskiego

Poznań, styczeń 2021

Poznań, 4 lutego 2021 r.

Oświadczenie

Ja, niżej podpisany **Kamil Tyrek, Mateusz Hypś, 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

Spis treści

Bibliography	4
Rozdział 1. Wstęp	5
1.1. Cel i założenia projektu	5
1.2. Organizacja pracy	6
1.3. Podział prac	7
1.3.1. Kamil Tyrek - tworzenie elementów logicznych i ich algorytmika	7
1.3.2. Mateusz Hypś - Zarządzanie projektem gry komputerowej z wykorzystaniem Agile	7
1.3.3. Jakub Kozubal - Fizyka postaci	7
1.4. Użyta technologia - Unity	8
Rozdział 2. Zarządzanie projektem gry komputerowej z wykorzystaniem Agile	9
2.1. Czym jest projekt?	9
2.1.1. Zakres	10
2.1.2. Zasoby	10
2.1.3. Czas	11
2.1.4. Pieniądze	12
2.2. Czym jest zarządzanie projektem?	12
2.3. Kim jest menedżer projektu?	14
2.4. Cykl życia projektu informatycznego	16
2.4.1. Metody pomocne w zarządzaniu projektami informatycznymi	16
2.4.2. Ryzyko oraz powody niepowodzeń projektów informatycznych	16
2.5. Zarządzanie projektem gry komputerowej - The Lore	16
2.5.1. Proces planowania projektu	16
2.5.2. Wykorzystane systemy i metody projektowe	16
2.5.3. Rola i wpływ kierownika projektu	16
2.5.4. Cykl życia projektu The Lore	16
Rozdział 3. Zagadki logiczne	17
3.1. Wprowadzenie do zagadek logicznych	17
3.1.1. Omówienie zagadnienia	17
3.1.2. Przykłady logicznych zagadek w prawdziwym świecie	17
3.1.3. Wdrożenie zagadek do gry opartej na silniku Unity	19

3.2. Przesuwane puzzle	21
3.2.1. Omówienie zagadnienia i ogólne założenia	21
3.2.2. Algorytmika	21
3.2.3. Przedstawienie przykładu w grze The Lore	25
3.2.4. Przedstawienie przykładu w innych grach	27
3.3. Zagadka z rurami	29
3.3.1. Omówienie zagadnienia	29
3.3.2. Algorytmika	29
3.3.3. Przedstawienie przykładu w grze The Lore	31
3.3.4. Przedstawienie przykładu w innych grach	33
3.4. Zagadka z otwieraniem zamku	34
3.4.1. Omówienie zagadnienia	34
3.4.2. Algorytmika	34
3.4.3. Przedstawienie przykładu w grze The Lore	35
3.4.4. Przedstawienie przykładu w innych grach	36
3.5. Labirynt	39
3.5.1. Omówienie zagadnienia	39
3.5.2. Algorytmika	39
3.5.3. Przedstawienie przykładu w innych grach	42
Rozdział 4. Fizyka postaci	44
4.1. Tworzenie postaci	45
4.1.1. Omówienie zagadnienia procesu tworzenia postaci	45
4.1.2. Proces graficzny tworzenia postaci	45
4.1.3. Wdrożenie postaci do projektu w Unity	45
4.1.4. Sposoby animowania postaci	45
4.2. Poruszanie się	45
4.2.1. Omówienie zagadnienia	45
4.2.2. Fizyka w grach 2D, a w prawdziwym świecie	45
4.2.3. Poruszanie się oraz kolizje postaci	45
4.2.4. Skakanie	45
4.2.5. Otoczenie wpływające na fizykę postaci	45
4.2.6. Umiejętności związane z poruszaniem się	45
Bibliografia	46

ROZDZIAŁ 1

Wstęp

1.1. CEL I ZAŁOŻENIA PROJEKTU

Celem projektu jest 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 wprowadzaniu poszczególnych elementów rozgrywki, którymi jest między innymi przedstawienie sterowania oraz funkcjonalności i wstępnych mechanik jest również wymagający, od gracza zależy jakie umiejętności w trakcie rozgrywki będzie rozwijać, aby przejście kolejnych poziomów było łatwiejsze (z perspektywy gracza). Dużą wagę w projekcie przywiązujemy do mini-gier, które występują w trakcie przechodzenia poszczególnych poziomów. Występują dwa rodzaje mini-gier: opcjonalne (te które przejść możemy w celu sprawdzenia siebie oraz zdobycia punktów doświadczenia) oraz wymagane, które trzeba przejść, aby znaleźć się w dalszym etapie gry. Użytkownik posiada do dyspozycji punkty doświadczenia, drzewko umiejętności oraz ekwipunek. Punkty doświadczenia możemy wydawać bezpośrednio w drzewku umiejętności, w którym zadaniem gracza jest wybranie odpowiednich umiejętności 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 decyzjami dotyczącymi rozwijania konkretnych umiejętności w drzewku. Ekwipunek służy do zdobywania przedmiotów potrzebnych w trakcie rozgrywki m.in. do otwierania drzwi czy rozpoczęcia opcjonalnej mini-gry. W projekcie są wykorzystane 2 style tworzenia poziomów: orthographic i perspective. Wszystkie animacje w projekcie są tworzone z użyciem technologii inverse

kinematics co pozwala na stałe dodawanie nowych animacji i poprawianie już istniejących.

1.2. ORGANIZACJA PRACY

Charakter projektu sprawił, iż nie można w naszym zespole jasno podzielić typów zadań, które realizujemy 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.

Jako metodykę pracy przyjęto Scrum. Głównymi powodami tej decyzji jest doświadczenie części zespołu w tej metodyce oraz przejrzystość i rozsądne zarządzanie pracą. Nie rozważano innych metodyk pracy. Przy zarządzaniu pracą wspomaga nas serwis JIRA, który pozwala zarządzać regularne sprinty – w pierwszym semestrze dwutygodniowe, w drugim semestrze tygodniowe.

Kod źródłowy zarządzany jest poprzez GitHub. Dla przejrzystości pracy, każdy commit na GitHub oznaczany jest id zadania na Jirze, dzięki czemu wchodząc w zadanie widzimy commit powiązany z jego rozwiązaniem. W momencie rozpoczęcia zadania deweloper, jeśli następuje potrzeba, tworzy podzadania do zadań na Jirze. Dotyczy to większych zadań, których wykonanie polega na tworzeniu większej ilości funkcjonalności. Dzięki temu realizując nowe zadania, o podobnej budowie, można sugerować się podobnym sposobem działania zadania. Po każdym commitcie programista wyznacza ile czasu poświęcił na zadanie, poprzez wbudowaną w Jirze funkcjonalność "Log Work". Gdy zadanie zostanie zakończone, programista oznacza je statusem "DONE".

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 PRAC

1.3.1. Kamil Tyrek - tworzenie elementów logicznych i ich algorytmika

W tym 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 ustwieniem odpowiednich ruch, 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. Netrywialnym problemem jest wygenerowanie takiej planszy, 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 metod, 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 wideo 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.

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 wyjaśnienie 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 specjaliści 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”. [1]

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

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

- Zakres
- Zasoby
- Czas
- Pieniądze

2.1.1. Zakres

Zdecydowanie najważniejszy z całej czwórki. Jest on definicją 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. Jakakolwiek zmiana musi zostać odwzorowana w pozostałych trzech elementach. Jest to jeden z powodów dla którego się mówi o współzależności między tą czwórką. Aby to lepiej zrozumieć, można sobie wyobrazić aplikacje, która ma posiadać cztery główne funkcjonalności, zbudowana przez dwa zespoły z budżetem 40 tysięcy złotych. Jeśli zakres projektu się zmieni do przykładowo sześciu funkcjonalności, zadaniem menedżera projektu jest dostosowanie zasobów ludzkich, czasu oraz budżetu w taki sposób aby cel ten został zrealizowany.

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 dużym zbiorowiskiem 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 wy tłumaczone, 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 listowanie 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 harmonogramy 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 projekcie. 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 do 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.

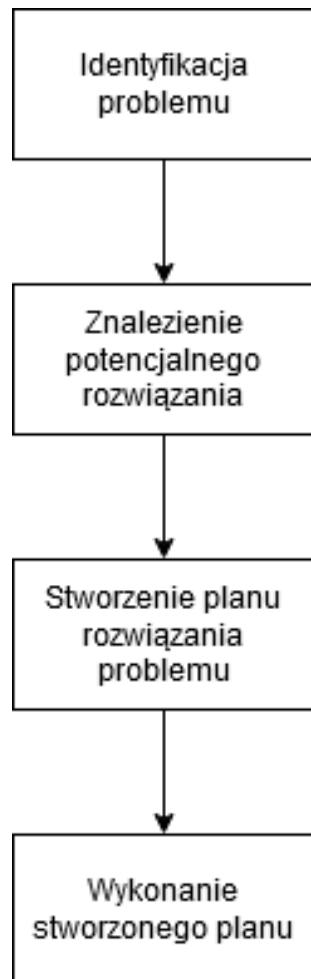
2.2. CZYM JEST ZARZĄDZANIE PROJEKTEM?

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 znaczący, ponieważ do osiągnięcia wyznaczonego celu wymagane jest ukończenie wszystkich zadań po drodze. 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 różną trasę w celu zrozumienia, która jest z nich będzie najlepsza. Następnie wyestymować czas, potrzebny na podróż, przemyśleć sposoby w jaki się tam dostaniesz czy to pieszo czy za pomocą pojazdu. Przygotować potrzebny budżet oraz przeanalizować potencjalne zagrożenia. jak widać zarządzanie projektem nie występuje tylko w biznesie, jest to szeroko pojęte zdarzenie, które 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.

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 nieuchcianych 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 uproszczonej 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 odpowiedzial-

nością jest rozpoczęcie cyklu życia projektu. Zaczyna od przygotowania planu działania a następnie wdrożenia go i zrealizowania aby następnie móc zamknąć 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

2.4.1. Metody pomocne w zarządzaniu projektami informatycznymi

2.4.2. Ryzyko oraz powody niepowodzeń projektów informatycznych

2.5. ZARZĄDZANIE PROJEKTEM GRY KOMPUTEROWEJ - THE LORE

2.5.1. Proces planowania projektu

2.5.2. Wykorzystane systemy i metody projektowe

2.5.3. Rola i wpływ kierownika projektu

2.5.4. Cykl życia projektu The Lore

ROZDZIAŁ 3

Zagadki logiczne

3.1. WPROWADZENIE DO ZAGADEK LOGICZNYCH

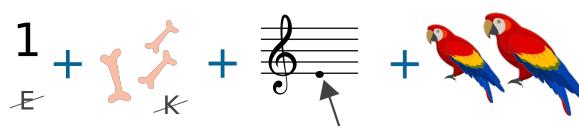
3.1.1. Omówienie zagadnienia

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. [2]

3.1.2. Przykłady logicznych zagadek w prawdziwym świecie

Rebus

Przywołanym już przykładem 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. [3]



Rysunek 3.1. 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 szerzej poruszony będzie w dalszej części pracy, gdzie oprócz założeń związanych z rozwiązaniem zagadki, przedstawiona będzie logika idąca za zrealizowaniem tej zagadki w grze opartej na silniku Unity.



Rysunek 3.2. 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 rozwiązajemy problem polegający na ułożeniu z dostępnych elementów logicznego obrazka, który zazwyczaj jest dołączony do gry - w postaci fotografii czy też po prostu znajduje na pudełku.

Słowo puzzle pochodzi z XVI wieku, które 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 znajdującego się na mapanie. 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, używano ją dość często, nawet jeszcze w wieku XIX.

Trudność puzzli zależy zazwyczaj od kształtu elementów i ich ilości. Jeżeli chodzi o kształt, puzzle często mają specyficzny kształt, przez co nie każdy element pasuje do innego, co pozwala nam odrzucić możliwości połączenia różnych par. Ilość elementów zwiększa ilość możliwych oraz logicznych kombinacji danych elementów, 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 Krzywą Wieżę w Pizie. Ułożenie tej układanki możemy liczyć w setkach godzin. [5]



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 zagedek ze względu na sposób umieszczenia mini-gry w odpowiedniej hierarchii projektowej, warto poznać podstawowe pojęcia, które dotyczą projektu Unity.

Scena w Unity

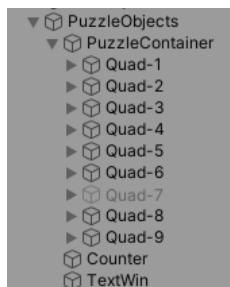
W Unity możemy podzielić naszą grę na sceny. Scena to obiekt zawierający nasze menu, czy dane środowisko gry (na przykład poziom gry). Dobrą praktyką jest, aby każdy level gry był osobą sceną, co skróci czas jego ładowania. W scenie możemy umieścić swoje obiekty, skrypty czy grafiki. [6]

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. [7]

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. [8]



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 minigra 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 algorytmką mini-gry może być uciążliwe na słabszych komputerach.
- Każda minigra zawiera się w scenie poziomu gry
 - Unikamy dłuższego ładowania się poziomu gry. Po zakończeniu mini-gry w przypadku osobnej sceny, cały poziom musi załadować się od nowa.

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 1. 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", określającą ilość dostępnych pól w najpopularniejszym ułożeniu - 4x4. [?]

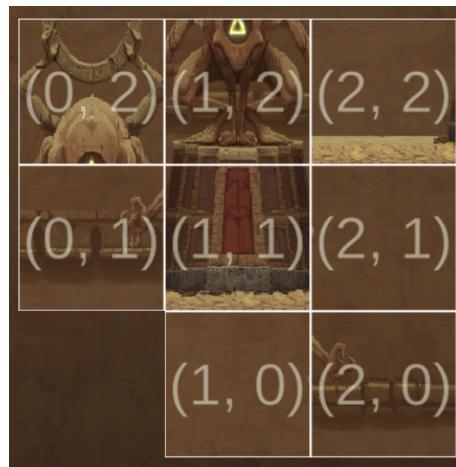
W grze The Lore gracze staną przed rozwiązaniem zagadki gdzie do dyspozycji mamy dziewięć pól. Podczas projektowania układanki w grze uznano, iż zagadka może być dość trudna, a korzyści płynące z rozwiązania jej będą nieadekwatne do poświęconego czasu, stąd ilość pól jest mniejsza niż w najpopularniejszej 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). 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 przypadkiem takiej sytuacji spotykaliśmy podczas pierwszych testów mini-gry puzzle.



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

Jak się okazuje, 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. [9] 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, rozpocznie 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 przesuwającą 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**.

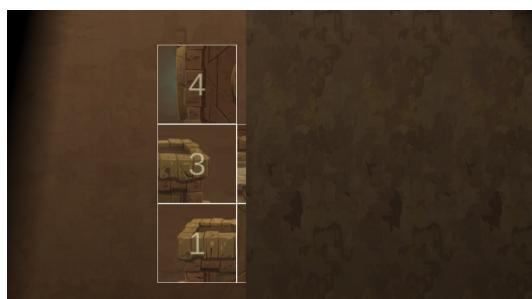
3.2.3. Przedstawienie przykładowu w grze The Lore

W grze The Lore zagadka przesuwanych puzzli pełni rolę opcjonalnej rozgrywki, za której 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 sprawiającą 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. Przykład: Gdzie Exit kończy rozgrywkę, przenosząc nas z powrotem w miejsce, w którym roz-



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

poczynaliśmy 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 wykonaliśmy rozwiązuając zagadkę. Jest to dość istotna informacja dla gracza, gdyż to od ilości ruchów zależy ile punktów doświadczenia zdobędzie za rozwiązywanie 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 animacje przesuwającej się płytki, tym razem zamykającej się.



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

Każdą 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: Haunting of Castle Malloy

The Haunting of Castle Malloy to gra dwuwymiarowa przygodowa osadzona w Irlandii. Postać wybiera się do tego kraju, aby zostać drużyną na ślubie swojego przyjaciela. Jadąc autem na uroczystość zostaje zaatakowany przez upiorną postać, przez co wpada do rowu. Jak się później okazuje, zginął 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. W czasie przechodzenia kolejnych etapów, użytkownik ma możliwość przyglądania się obiektem, które mija. To właśnie w tym momencie może natrafić na omawianą rozgrywkę. Przed mini-grą gracz widzi jaki jest stan wyjściowy puzzli. Co warto podkreślić, w przeciwieństwie do gry The Lore, na poszczególnych elementach puzzli nie widzimy cyfr označających poprawną kolejność puzzli. Aby zakończyć zagadkę należy również posiadać ostatni element, który po ułożeniu obrazka umieszczamy w brakującym miejscu. [14]

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 podejrzawaną organizację, córkę prezydenta USA. Bohater



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



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

przemierzając kolejne miasta odkrywa kolejne tajemnice związane z porwaniem. W toku gry zdarza się, iż gracz pokieruje 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 przeciwieństwie do poprzedniego przykładu, nie mamy tutaj pokazanego efektu końcowego rozgrywki. Ostatni element, tak jak w Nancy Drew: Hauting of Castle Malloy musimy odnaleźć, aby zagadka została ukończona.

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 - skońne, 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, woda przenosi się z rury źródłowej do kolejnej po pewnym czasie, 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 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, którego działania zostanie przedstawione w dalszej części przedstawienia 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, ponownie losowane miejsce dla obiektu **Pipe**. Gdy to już się stanie, 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 != 7)** jest weryfikacja, czy znajdujemy się przy najbardziej wysuniętym na prawo **PipeSlot**. Następnie weryfikujemy, czy sąsiadni 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. Odpoowiada 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 kursorem 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.

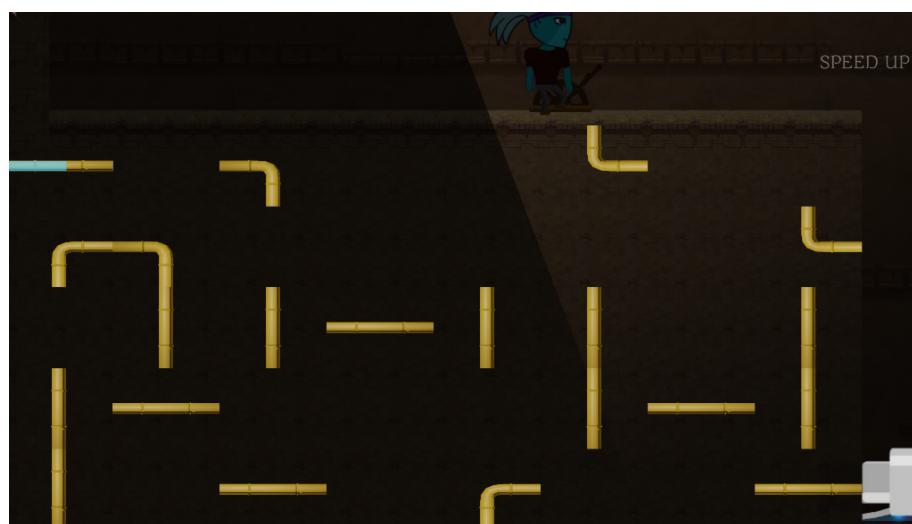
3.3.3. Przedstawienie przykładu w grze The Lore

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. Przykład z projektu The Lore

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



Rysunek 3.16. Zagadka z rurami. Przykład z projektu The Lore

W lewym górnym rogu mamy źródło strumienia, zaś w prawym dolnym rogu zakończenie (szara rura). Użytkownik sterując myszką może przekładać rury o wybranym kształcie, tworzyć połączenia, celem stworzenia ścieżki, która jest konieczna do ukończenia mini-gry. Rozrywka zawsze generuje tą samą liczbę poszczególnych typów rur. Co warto podkreślić, liczba ta jest ważna -

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. [10]

Soda Pipes

Grą 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. [19]

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, które polega na wykonaniu różnych operacji. Zazwyczaj jest od odkrycie odpowiedniej sekwencji poruszania wytrychem, otworzyć zapadki poprzez poruszanie myszką, czy wcisnięcie odpowiedniego guzika, gdy postęp otwierania zamku jest na odpowiednim etapie. Otworzenie zamku pozwala otworzyć skrzynię, zawierającą elementy wyposażenia lub otworzyć 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, otworzenie 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 się 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 wspominano 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. Odbiera się to poprzez zmianę pozycji elementu transform.

Wyciąg 3-8. Fragment skryptu Picklock.cs

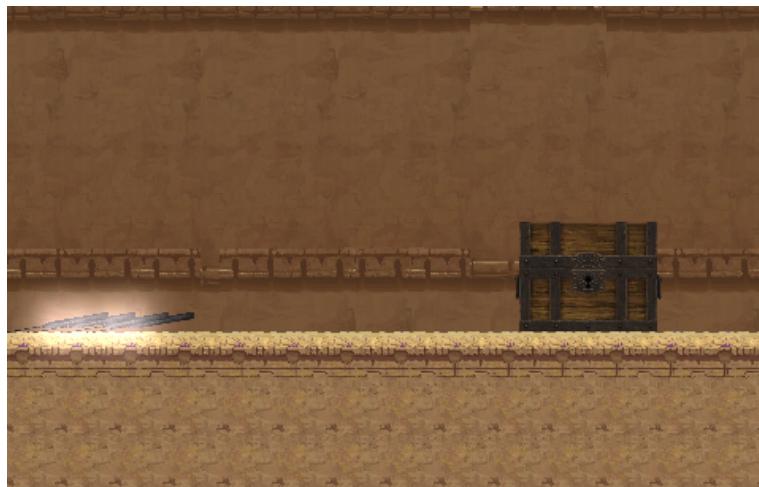
```
1 | picklockTransform.position += new Vector3(10 * direction, 0, 0) * 0.1f;
```

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 przykładu 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.

Jeżeli chcemy otworzyć skrzynie nieposiadając w ekwipunku żadnego wytrychu, jesteśmy informowani o tym fakcie poprzez komunikat na ekranie. Gdy



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

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 naciśnijemy 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 życia, które przydają się w czasie rozgrywki. Z racji, iż jest to rozgrywka nieobowiązkowa, gracz w każdej chwili może ją opuścić wybierając opcję **Exit**.

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 wpłata się w mafijne porachunki. Realizując zadania dla swoich pracodawców, nierzaz włamywał się do różnych pomieszczeń. 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 odblokowa-



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

nie 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. [11]

Assassin's Creed Unity

W grze Assassin's Creed Unity wcielamy się w Arno Dorianą, członka tajnego bractwa Assasynów, którzy toczą odwieczną walkę z organizacją zwaną Templariuszami. 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 poprzednie grze, możliwość obejścia wrogów, czy też zdobycie ekwipunku oraz włamywania się do skrzyni - 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. [12]



Rysunek 3.19. Otwieranie zamków. Przykład z gry Mafia 2



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, które grożą krainie Myrtana. Gracz sterując bohaterem może dopuszczać się przestępstw, które pozwalają mu 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

skrzynie musi posiadać wytrychy, które może zakupić od kupców w mieście. Otwieranie skrzyń 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

W pierwotnej definicji, labirynt oznacza budowlę, która charakteryzuje się układem dużej ilości pomieszczeń, które połączone są krętymi 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.

3.5.1. Omówienie zagadnienia

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 ściana, której użytkownik nie może przejść. Aby wygenerować taki graf, początkowo generujemy graf 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 trafiemy do węzła, który nie ma żadnego nieodwiedzonego sąsiada. Gdy już dotrze do takiego miejsca, cofa się, jednocześnie usuwając elementy ze stosu, aż do momentu, gdy trafi na węzeł, który posiada nieodwiedzonego sąsiada. Algorytm skończy się, gdy w naszym stosie nie będzie elementów. [13]

Implementacja: algorytm z nawrotami

Przejdźmy teraz do zaprojektowania takiej sytuacji. Na sam początek zdefiniujmy stany, które może posiadać każdy węzeł w naszym grafie (labiryncie). Jak wiemy, może mieć ściany 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 ściana lewa i góra, 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] |= WallState.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 **WallState**. 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 przyjazny 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 **WallState** 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. Zwarcanie stanu sąsiada - skrypt C#

```
1 | switch(nodeState)
2 | {
```

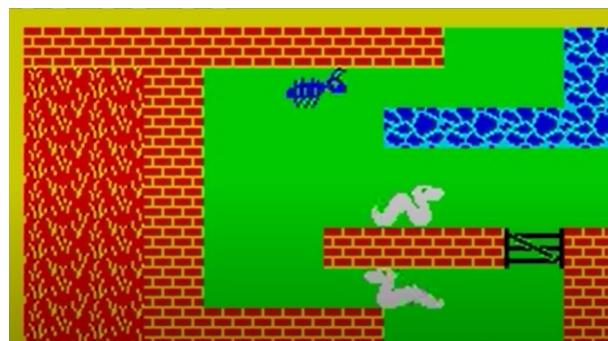
```
3     case NodeState.RIGHT = return NodeState.LEFT;
4     case NodeState.LEFT = return NodeState.RIGHT;
5     case NodeState.UP = return NodeState.DOWN;
6     case NodeState.DOWN = return NodeState.UP;
7 }
```

Algorytm gotowy. W tym momencie mamy stworzony generator labiryntu, który powinien zostać 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). Dodatkowa 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)**. [15] 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)**. [16]. Pozycja jest tutaj zależna od naszych wymagań - powinnismy 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 widoczny wygenerowany labirynt.

3.5.3. Przedstawienie przykładu 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 brazi Kempthorne oraz David'a Vivian'a. [17]



Rysunek 3.21. Labirynt. Przykład z gry Wriggler

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 schoami. 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. [18]

ROZDZIAŁ 4

Fizyka postaci

4.1. TWORZENIE POSTACI

4.1.1. Omówienie zagadnienia procesu tworzenia postaci

4.1.2. Proces graficzny tworzenia postaci

4.1.3. Wdrożenie postaci do projektu w Unity

4.1.4. Sposoby animowania postaci

4.2. PORUSZANIE SIĘ

4.2.1. Omówienie zagadnienia

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

4.2.3. Poruszanie się oraz kolizje postaci

4.2.4. Skakanie

4.2.5. Otoczenie wpływające na fizykę postaci

4.2.6. Umiejętności związane z poruszaniem się

Bibliografia

- [1] Robert K. Wysocki, Rudd McGary: Efektywne zarządzanie projektami. Wydanie III, ISBN: 83-7361-861-9, dostęp w internecie 26.01.2021r. <http://pdf.onepress.pl/efzapr/efzapr-1.pdf>
- [2] Wikipedia, Wolna Encyklopedia." Wikimedia Foundation, Inc. July 17, 2002, dostęp 07.01.2021r. <https://pl.wikipedia.org/wiki/Zagadka>
- [3] Wikipedia, Wolna Encyklopedia." Wikimedia Foundation, Inc. July 17, 2002, dostępny 07.01.2021r. <https://pl.wikipedia.org/wiki/Rebus>
- [4] Wikipedia, Wolna Encyklopedia." Wikimedia Foundation, Inc. July 17, 2002, dostępny 07.01.2021r. https://en.wikipedia.org/wiki/15_puzzle
- [5] Wikipedia, Wolna Encyklopedia." Wikimedia Foundation, Inc. July 17, 2002, dostępny 07.01.2021r. <https://pl.wikipedia.org/wiki/Puzzle>
- [6] Unity Documentation, dostęp 09.01.2021r. <https://docs.unity3d.com/Manual/CreatingScenes.html>
- [7] Unity Documentation, dostęp 09.01.2021r. <https://docs.unity3d.com/ScriptReference/GameObject.html>
- [8] Unity Documentation, dostęp 28.01.2021r. <https://docs.unity3d.com/ScriptReference/Component.html>
- [9] Geeks for geeks. Check instance 8 puzzle solvable, dostęp 26.01.2021r. <https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/>
- [10] Wikipedia, Wolna Encyklopedia." Wikimedia Foundation, Inc. July 17, 2002, dostępny 27.01.2021r. https://en.wikipedia.org/wiki/Pipe_Mania
- [11] Mafia Wiki, Fandom, dostęp 28.01.2021r. https://mafia.game.fandom.com/wiki/Lock_Picking
- [12] IGN, Assassin's Creed Unity Wiki Guide, dostęp 28.01.2021r. <https://www.ign.com/wikis/assassins-creed-5-unity/Lockpick>
- [13] Qaz Wiki: Algorytm generowania labiryntu - Maze generation algorithm, dostępny 02.02.2021r. https://pl.qaz.wiki/wiki/Maze_generation_algorithm
- [14] Fandom Wiki, The Haunting of Castle Malloy, dostęp 03.02.2021r. https://nancydrew.fandom.com/wiki/The_Haunting_of_Castle_Malloy

- [15] Unity Documentation, dostęp 04.02.2021r.
<https://docs.unity3d.com/ScriptReference/Object.Instantiate.html>
- [16] Unity Documentation, dostęp 04.02.2021r.
<https://docs.unity3d.com/ScriptReference/Vector3.html>
- [17] Personal Computer News, Issue 107, dostęp 04.02.2021r.
http://www.personalcomputernews.co.uk/pcnb/html/107/personal_computer_news_107_game
- [18] "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>
- [19] "Soda Pipes", Moby Games. Dostęp 04.02.2021r.
<https://www.mobygames.com/game/windows/soda-pipes>