AKADEMIA
TECHNICZNO-INFORMATYCZNA
W NAUKACH STOSOWANYCH

Akademia Techniczno-Informatyczna

w Naukach Stosowanych

Wydział Informatyki

Tomasz Maligranda

Nr albumu: 7232

Projektowanie i implementacja gry point and click z narracyjnymi elementami visual novel

Praca inżynierska

Kierunek: informatyka

Specjalność: Programowanie gier komputerowych

Nr albumu:7232

Praca wykonana pod kierunkiem:
mgr inż. Stanisław Lota

Wrocław 2025

Spis treści

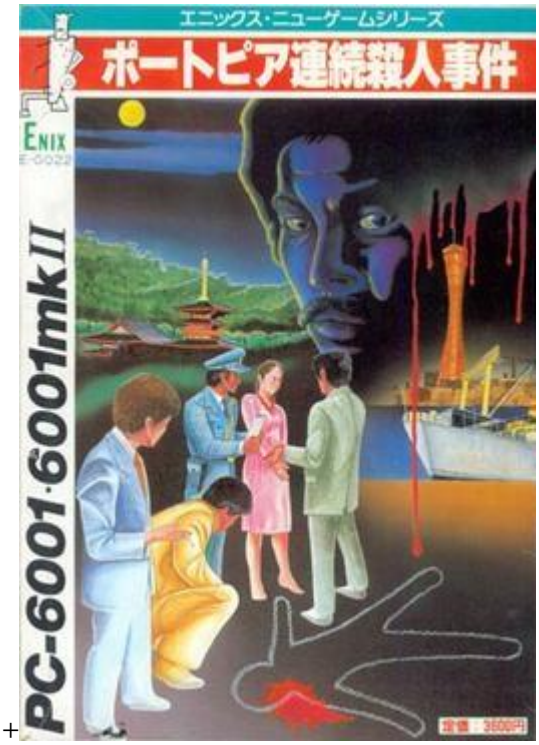
1. Wstęp

Projektowana gra łączy elementy *visual novel*[1], interaktywnych scen *point-and-click*[2] oraz mini gier logicznych, tworząc unikalne doświadczenie kryminalno-detektywistyczne. Projekt powstał w oparciu o wykorzystanie silnika Unity oraz dedykowanych skryptów w języku C#, co umożliwiło płynne zaimplementowanie warstwy narracyjnej, mechanik wyborów oraz różnych form interakcji z otoczeniem. Istotnym aspektem prac było opracowanie systemu obsługującego dialogi i obsługę scen, a także integracja mechanik – od identyfikacji podejrzanych osób na komisariacie, poprzez obsługę telefonu i radia, aż po zagadki w stylu *point-and-click*.

W trakcie realizacji projektu skoncentrowano się na kompleksowym podejściu do tworzenia gry: od określenia założeń i koncepcji, przez projektowanie mechanik oraz interfejsu, aż po wdrożenie logiki pozwalającej na swobodne przechodzenie między różnymi trybami rozgrywki. Efektem prac jest prototyp umożliwiający graczom eksplorację, podejmowanie decyzji, rozwiązywanie zagadek i rozwijanie fabuły w sposób nieliniowy. Ta baza może być w przyszłości rozszerzana o nowe funkcjonalności oraz stosowana jako fundamenty nowych projektów.

Visual novels, zwane często powieściami wizualnymi, wywodzą się z Japonii, gdzie gatunek ten zdobył popularność już w latach 80. XX wieku. Pierwsze produkcje, takie jak *The Portopia Serial Murder Case* (1983), położyły podwaliny pod ten gatunek, łącząc proste elementy interaktywne z rozbudowaną narracją. Visual novels stały się niezwykle popularne w Azji, a w ostatnich latach zyskały coraz większe uznanie na całym świecie, dzięki lokalizacjom popularnych tytułów i rozwojowi platform dystrybucji cyfrowej. Rysunek poniżej przedstawia okładkę gry uznawanej za pierwsza grę tego gatunku.

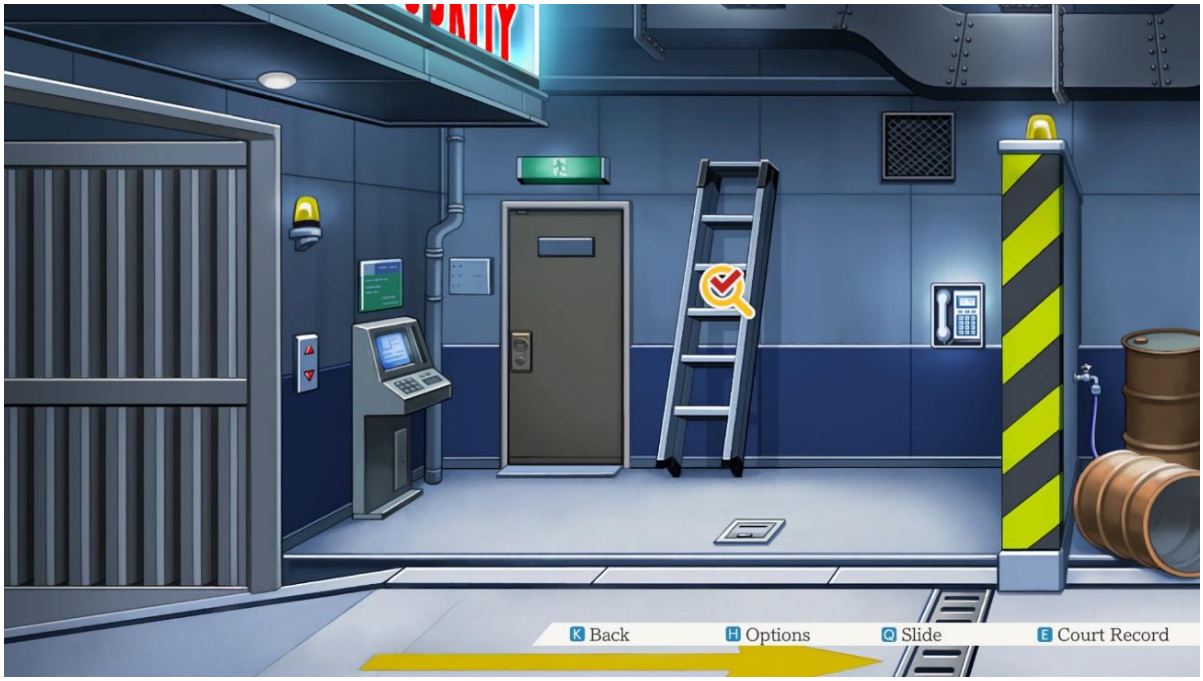
[1] Visual novel to interaktywna forma gry komputerowej lub opowieści, która łączy tekst, grafikę i dźwięk.
[2] Point-and-click to gatunek gier komputerowych, w którym gracz eksploruje świat, rozwiązuje zagadki i wchodzi w interakcje z obiektami lub postaciami, korzystając głównie z myszy do wskazywania i klikania.



Rysunek 1.1 The Portopia Serial Murder Case, © Square Enix 1983
https://upload.wikimedia.org/wikipedia/en/7/77/Portopia_cover_art.jpg (dostęp 07.11.2024)

Główną siłą gier *visual novels* jest ich zdolność do tworzenia emocjonalnych i angażujących historii, które często obejmują wybory moralne i wielowątkowe zakończenia. Gry te, wraz z *point-and-click'ami*, stanowią kluczową część szerszej kategorii tzw. *storygames*, czyli gier nierozerwalnie związanych z opowiadaniem historii. Jak zauważono: "We use the term 'storygame' throughout this volume to refer to the wider category of games fundamentally intertwined with story; but for our primary subject, this is too broad a label, while many more specific terms, such as 'point-and-click games,' 'visual novels,' or 'text adventures,' are too narrow, focusing on grouping games more by their interface elements or aesthetics than by a shared design." [1]Gry o gatunku *visual novels* i *point-and-click* świetnie współgrają, łącząc narrację z interaktywnością w sposób, który angażuje graczy zarówno intelektualnie jak i emocjonalnie. To harmonijne połączenie pozwala twórcom eksplorować różnorodne formy narracji, jednocześnie wzbogacając grę o elementy eksploracji i rozwiązywania zagadek, co otwiera nowe możliwości dla projektowania gier opartych na historii. Gry *point-and-click* to tytuły, w których gracz eksploruje świat gry poprzez klikanie na obiekty, aby wchodzić z nimi w interakcje, rozwiązuje zagadki i posuwa fabułę do przodu. Choć tradycyjnie uznaje się je za osobny gatunek, w kontekście współczesnych produkcji *visual novel* takie mechaniki są często integrowane w formie mini gier i zagadek logicznych aby wzbogacić rozgrywkę i zwiększyć możliwości interakcji. Dodanie takich elementów pozwala na

większe zaangażowanie gracza oraz pobudza logiczne myślenie, co wzmacnia poczucie spójności i interaktywności gry. Przykładami gier, które z powodzeniem łączą elementy *visual novel* i *point-and-click* są popularne tytuły takie jak *Phoenix Wright: Ace Attorney* oraz *Hotel Dusk: Room 215*. Produkcje te pokazują, że możliwe jest stworzenie interaktywnej narracji, angażuje gracza zarówno poprzez podejmowanie decyzji, jak i eksplorację otoczenia oraz rozwiązywanie zagadek. Celem projektu jest osiągnięcie podobnego połączenia tych gatunków, co pozwoli na urozmaicenie rozgrywki, między innymi poprzez wprowadzenie



Rysunek 1.2 Scena point and click z gry Phoenix Wright: Ace Attorney Trilogy CAPCOM 2019

mini gier, które wymagają logicznego myślenia oraz rozwiązania konkretnych problemów, a także pełnią funkcję przerywników fabularnych. Tego rodzaju mechaniki wzbogacają narrację i pozwalają na głębsze zaangażowanie gracza. Takie podejście ma na celu urozmaicenie rozgrywki, dodanie głębi oraz zwiększenie spójności gry.

2. Narzędzia, technologie i assety wykorzystane w projekcie

Podczas tworzenia gry kluczowym narzędziem był silnik Unity, który wybrano ze względu na jego elastyczność oraz bogate możliwości wprowadzania autorskich rozwiązań. Unity, w przeciwieństwie do silników skoncentrowanych na gatunku *visual novel* (jak Ren'Py), pozwolił na łatwe implementowanie niestandardowych mechanik, interaktywnych obiektów (telefon, radio) oraz sekwencji *point-and-click*. Dodatkowym atutem jest doświadczenie z tym środowiskiem oraz możliwość portowania gry na różne platformy – komputery osobiste, konsole czy systemy mobilne – jeżeli w przyszłości pojawi się taka potrzeba.

Do pisania kodu zostało użyte środowisko Visual Studio, które ściśle integruje się z Unity i zapewnia podpowiedzi składni oraz zarządzanie projektem w języku C#, który pozwala sprawnie tworzyć i modyfikować logikę gry, utrzymując spójny i zrozumiały kod źródłowy. W trakcie projektowania struktury historii oraz planowania mini gier wykorzystano narzędzia draw.io. Pozwoliło to w intuicyjny sposób tworzyć diagramy, drzewka decyzyjne, powiązania między scenami i notatki. Za pomocą prostych, wizualnych reprezentacji można było na bieżąco weryfikować i ulepszać koncepcje rozgrywki, optymalizować przepływ fabuły oraz powiązania pomiędzy elementami gry.

Do zarządzania kodem i kontrolą wersji zastosowano Git oraz platformę GitHub, co pozwoliło zabezpieczać postępy prac, tworzyć oddzielne gałęzie do testowania nowych funkcjonalności oraz łatwo wracać do wcześniejszych wersji, jeśli wprowadzone zmiany okazywały się nieskuteczne lub generowały błędy. Stosowanie systemu kontroli wersji zwiększyło stabilność i przejrzystość procesu. Dobór tych narzędzi i technologii był podyktowany przede wszystkim potrzebą zachowania elastyczności, skalowalności i wygody w rozwijaniu wszystkich elementów projektu. Unity zapewniło wszechstronną bazę do implementacji niestandardowych mechanik, Visual Studio ułatwiło pisanie i utrzymanie kodu, draw.io wspomogło warstwę koncepcyjną, a GitHub umożliwił sprawną i bezpieczną pracę nad kolejnymi iteracjami gry. W efekcie połączenie tych narzędzi pozwoliło na efektywną organizację procesu tworzenia oraz utrzymanie wysokiej jakości i spójności ostatecznego produktu.

Assety graficzne użyte w projekcie zostały przygotowane głównie przez grafików Ambre Buathier oraz Jon Kušar'a, którzy zapewnili unikalny wygląd i atmosferę gry. Przy pisaniu scenariusza wsparcie udzielił Ignacy Kitłowski. Część architektury silnika visual novel została oparta na poradniku przygotowanym przez kanał Stellar Studio na YouTube. Dźwięki i muzyka zostały przygotowane przez Phuwit'a Sangudomlert'a.

3. Projektowanie gry

Projektowanie gier to złożony proces, który łączy aspekty kreatywne, techniczne i narracyjne aby tworzyć spójną i angażującą rozgrywkę. Na projektowanie składa się opracowanie koncepcji gry, założeń projektowych, struktury historii, a także implementacja różnych mechanik i elementów wizualnych. Ważne jest, aby wszystkie te aspekty współgrały ze sobą, tworząc zbalansowaną i satysfakcjonującą całość. Gry są bowiem, "Games are an exercise of voluntary control systems, in which there is a contest between powers, confined by rules in order to produce a disequilibrial outcome" [2]To oznacza, że projektowanie gry wymaga skonstruowania systemów, które angażują gracza poprzez interakcję z kontrolowanym światem i wchodzenie w interakcję z wyzwaniami i regułami. Proces ten jest ukierunkowany na tworzenie sytuacji, które budują napięcie i zaskoczenie, a także wywołują emocje i zaangażowanie. W rezultacie gra powinna prowadzić do nieprzewidywalnych wyników, które zachęcają gracza do powrotu i ponownego odkrywania nowych możliwości. Dobre projektowanie gier to zatem sztuka łączenia różnych mechanizmów kontroli, fabuły oraz aspektów interaktywnych w sposób, który tworzy złożone i dynamiczne doświadczenie dla gracza.

3.1. Koncepcja gry

Projektowana gra to połączenie gatunków, łączące *visual novel* z elementami *point-and-click* oraz mini grami zagadkowymi. Celem było stworzenie angażującego doświadczenia, które nie tylko dostarcza wciągającej fabuły, ale także aktywnie angażuje gracza poprzez interaktywne mechaniki. W projekcie skupiono się na integracji tych elementów oraz ich implementacji w środowisku Unity, aby zapewnić spójność między narracją a rozgrywką.

Gra osadzona jest w latach 20. XX wieku w Wolnym Mieście Gdańsk. Wszystkie aspekty designu i środowiska zostały starannie dostosowane do realiów tamtego okresu, co nadaje grze unikalny klimat i autentyczność. Tematyka gry ma charakter kryminalno-detektywistyczny; gracz wciela się w postać detektywa prowadzącego śledztwa, rozwiązując zagadki i podejmując decyzje wpływające na przebieg fabuły oraz zakończenia – w tym dobre, złe i trzecie ukryte.

Struktura narracji w projektowanej grze opiera się na stylu *visual novel*, gdzie historia jest przedstawiana poprzez tekst i dialogi, a gracz podejmuje decyzje wpływające na dalszy przebieg fabuły. Taka forma prezentacji pozwala na głębokie zanurzenie się w opowieści i poznanie motywacji postaci. Zgodnie z zasadą: „Core game designs and mechanics are always directly linked to what the player character can do" [3] Podstawowe projekty

i mechaniki gier są zawsze bezpośrednio powiązane z tym, co może zrobić postać gracza. Gra została zaprojektowana w taki sposób, aby mechaniki interakcji idealnie współgrały z narracją i rolą gracza jako detektywa. Elementy *point-and-click* służą do eksploracji środowiska i zbierania informacji niezbędnych do postępu w grze. Przykładowo, podczas oględzin miejsca zbrodni gracz może odkryć, że narzędziem zbrodni był nóż. Dzięki otwartej sekwencji *point-and-click*, gracz ma możliwość znalezienia różnych wskazówek, ale może też je przeoczyć, co wpłynie na dalszy przebieg fabuły, na przykład podczas przesłuchania. Mini gry pełnią funkcję przerywników fabularnych i są zaprojektowane tak, aby wymagały logicznego myślenia, co dodaje głębi i angażuje gracza. Przykłady takich mini gier to dzwonienie przez telefon, gdzie gracz musi wybrać odpowiedni numer, aby skontaktować się z określoną postacią lub uzyskać ważne informacje, oraz identyfikacja złodzieja na komisariacie, gdzie gracz analizuje dokumenty i zdjęcia, aby wskazać sprawcę kradzieży sklepowej.

Rysunek 3.1 Scena wyboru sprawcy kradzieży sklepowej. Projekt własny

Największymi inspiracjami dla tego projektu były gry *Phoenix Wright* oraz *Beat Cop*. Strukturę narracji zaczerpnięto z *Phoenix Wright,* gdzie historia podawana jest w stylu *visual novel*, co pozwala na głębokie zanurzenie się w fabule oraz mechaniki *point-and-click,* w których gracz eksploruje sceny i zbiera informacje potrzebne w dalszej części gry. Na przykład, odkrycie narzędzia zbrodni na miejscu przestępstwa, które później jest kluczowe podczas przesłuchania. Taka struktura zachęca gracza do dokładnej eksploracji i zwiększa interaktywność rozgrywki.

Z *Beat Cop* zainspirowano się strukturą mini gier, które służą jako przerywniki w historii, dając graczowi chwilę wytchnienia i urozmaicając rozgrywkę. Przykłady to identyfikacja sprawcy kradzieży na podstawie dokumentów czy zdjęć, co wymaga analizy i logicznego myślenia. Takie podejście wzbogaca doświadczenie gracza, dlatego zaadaptowano je w realizowanym projekcie.

Jednym z unikalnych elementów gry jest wprowadzenie *hub'a* – centralnego miejsca, do którego gracz wraca po określonych momentach w fabule. W *hub'ie* gracz ma możliwość interakcji z różnymi obiektami i mini-grami, co jest rzadko spotykane w grach typu *visual novel*. Funkcje *hub'a* obejmują interakcje z otoczeniem, takie jak korzystanie z radia, telefonu czy zegara oraz ukryte mini gry, na przykład ukryta gra "w piętnastkę" w zegarze, która wymaga rozwiązania zagadki środowiskowej.

Interaktywne elementy, takie jak radio i telefon, wzbogacają doświadczenie gry. Radio umożliwia słuchanie muzyki z lat 20. XX wieku, co wzmacnia immersję w świecie gry. Telefon to unikatowa mechanika, która pozwala graczowi dzwonić na różne numery i uzyskiwać różne interakcje oraz informacje. Gracz korzysta z telefonu kilkukrotnie w trakcie gry, co dodaje głębi mechanicznej i narracyjnej. Poprzez integrację inspiracji z gier "*Phoenix Wright*" i "*Beat Cop*", a także wprowadzenie unikalnych elementów, takich jak *hub* i interaktywne obiekty, które zapewniają graczowi satysfakcję z eksploracji i podejmowania decyzji. Dużą uwagę poświęcono aspektom projektowym i programistycznym, dla utworzenia spójnej i przemyślanej mechaniki gry, która współgra z narracją.

3.2. Założenia projektowe

Projektowanie gry wymagało określenia szczegółowych założeń, które umożliwiłyby stworzenie spójnej i funkcjonalnej mechaniki, zgodnej z wizją projektu. W projekcie skupiono się na dwóch kluczowych aspektach opracowaniu silnika *visual novel* oraz zaprojektowaniu zestawu mini gier i interaktywnych elementów, które wzbogaciłyby rozgrywkę. Podstawą gry jest silnik *visual novel*, którego funkcjonalność wzorowano na popularnym silniku Ren'Py. W pierwszej kolejności rozpisano listę wymagań dotyczących tej części projektu. Silnik miał obsługiwać wyświetlanie postaci na ekranie, z możliwością płynnego przesuwania ich po ekranie lub animacji wprowadzających postać. Ważnym elementem było także umożliwienie zmiany *sprite'ów* postaci w celu ukazania różnych emocji i stanów, co pozwala na lepsze oddanie nastroju sceny i charakteru postaci. Ponadto, silnik miał obsługiwać

wyświetlanie teł odpowiednich dla danej sceny, odtwarzanie muzyki oraz efektów dźwiękowych (SFX) w celu wzmocnienia atmosfery i emocji towarzyszących rozgrywce.

Kolejnym istotnym wymaganiem było wyświetlanie tekstu dialogów w oknie dialogowym, wraz z imieniem postaci mówiącej, co ułatwia śledzenie fabuły i interakcji między postaciami. Silnik miał również umożliwiać pomijanie tekstu oraz automatyczne odtwarzanie dialogów, aby dostosować tempo gry do preferencji gracza. Dodatkowo, funkcjonalności takie jak czyszczenie okna dialogowego, wstrzymanie wykonywania komend czy ukrywanie interfejsu użytkownika były niezbędne do kontrolowania prezentacji treści i dynamiki scen. Kluczowym elementem silnika był system wyborów, który pozwala graczowi podejmować decyzje wpływające na przebieg fabuły. Aby umożliwić różne ścieżki fabularne i zakończenia, niezbędny był także system zapamiętywania wyborów gracza. Wszystkie komendy oraz teksty miały być obsługiwane za pomocą plików tekstowych, które są interpretowane przez silnik gry. Takie podejście umożliwia łatwą edycję i rozszerzanie treści gry bez konieczności modyfikowania kodu źródłowego.

Aby urozmaicić rozgrywkę i zwiększyć interaktywność, zaprojektowano również listę mini gier oraz funkcjonalności poza klasycznym systemem *visual novel*. Jednym z głównych elementów był interaktywny *hub* — centralne miejsce, do którego gracz wraca między kluczowymi momentami fabuły. W *hub'ie* gracz ma możliwość interakcji z różnymi obiektami, takimi jak radio, telefon czy zegar, co jest rzadko spotykane w grach typu *visual novel*.

Jedną z mini gier jest dzwonienie przez telefon, gdzie gracz może wybierać numery i prowadzić rozmowy, co wpływa na postęp w grze. Zaprojektowano interaktywne sceny z przedmiotami do interakcji, takimi jak gazeta czy książka telefoniczna, które pozwalają na zdobywanie dodatkowych informacji i poszlak. Dwie interaktywne sceny narracyjne typu *point-and-click* umożliwiają graczowi eksplorację środowiska i przeszukiwanie miejsc, co wpływa na rozwój fabuły.

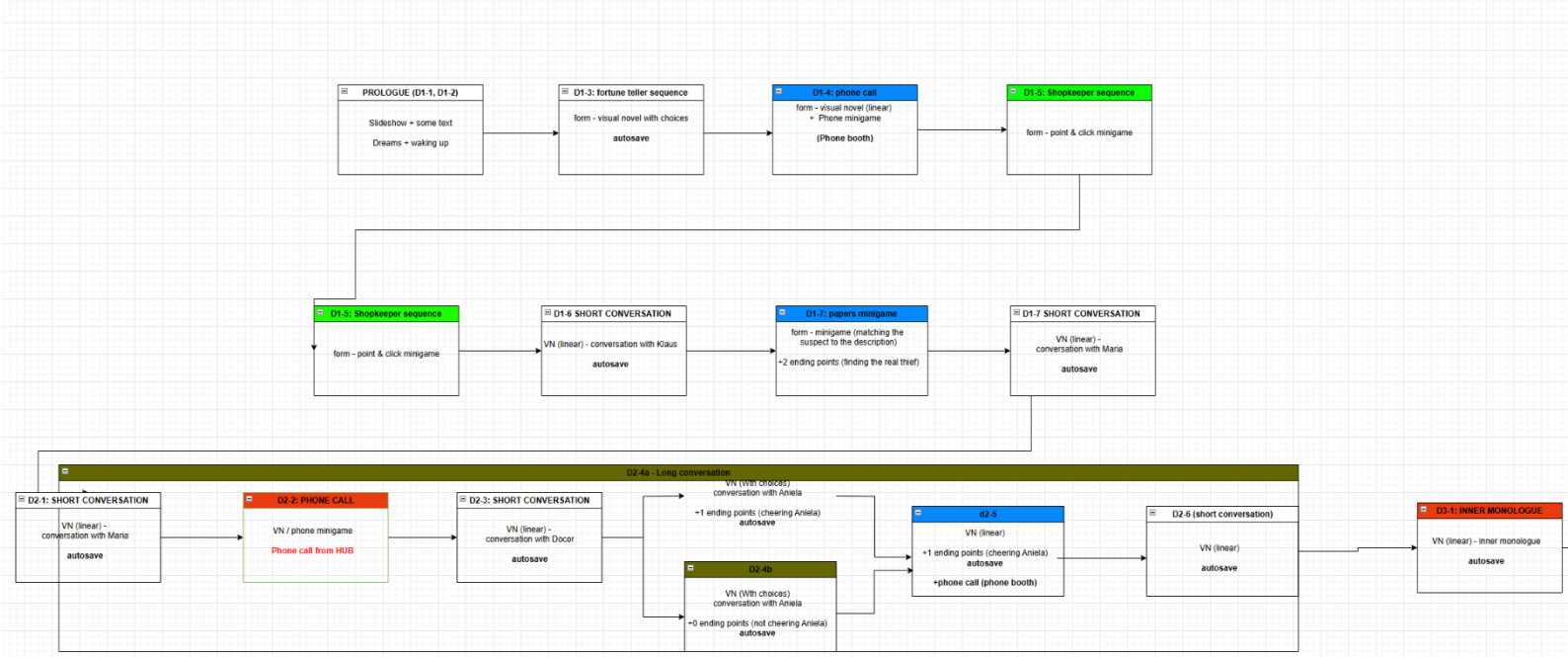
Rysunek 3.2Scena z telefonem znajdującym się w hub'ie. Projekt graficzny Ambre Buathier

Inną minigrą jest układanie puzzli ("piętnastka") ukrytą w zegarze, która wymaga rozwiązania zagadki środowiskowej aby ją odkryć. Scena z funkcjonalnym radiem z tamtych lat pozwala graczowi na słuchanie muzyki z okresu, co wzmacnia immersję i autentyczność świata gry. Wszystkie te elementy zostały zaprojektowane tak, aby w naturalny sposób integrowały się z główną fabułą, jednocześnie dostarczając dodatkowych wyzwań i satysfakcji z interakcji.

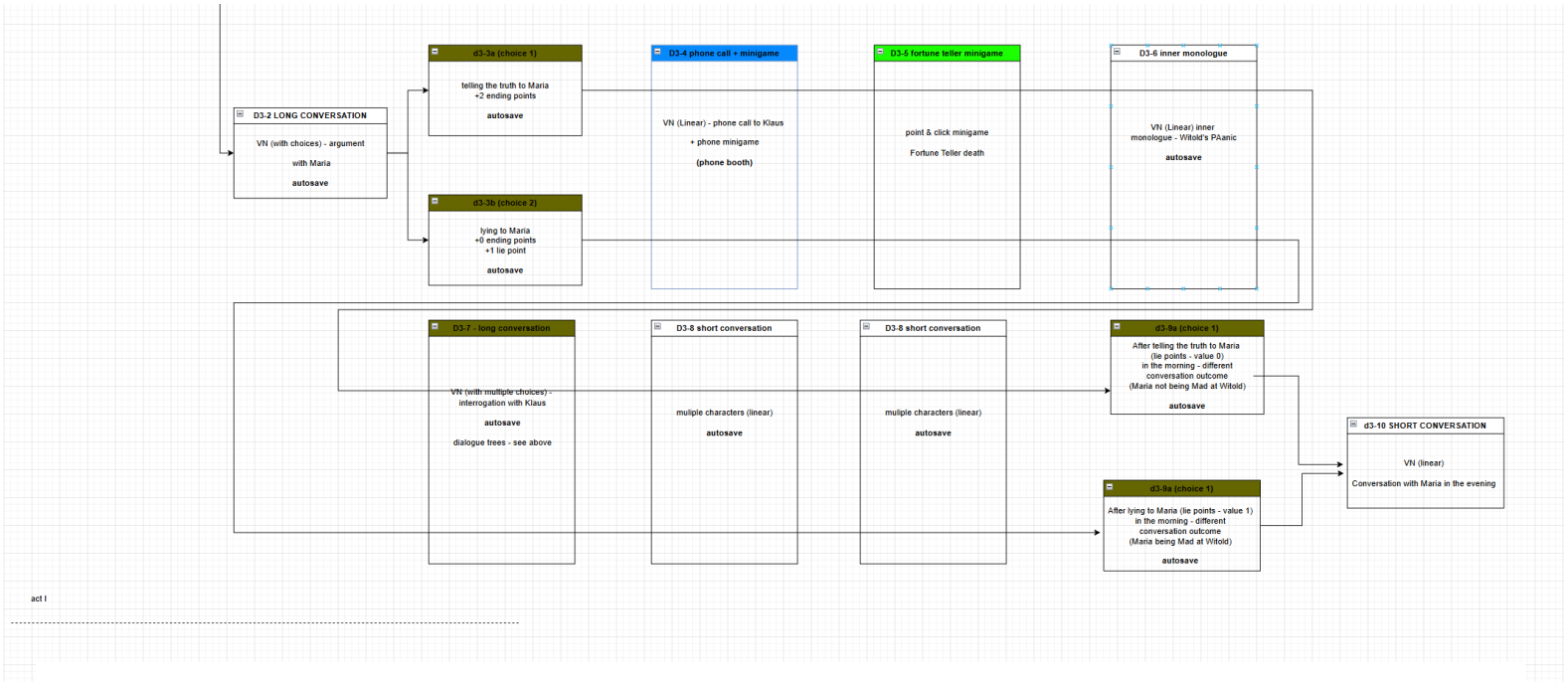
Silnik *visual novel* zaprojektowano z myślą o elastyczności i łatwości obsługi. Wykorzystano środowisko Unity oraz język C#, co pozwoliło na implementację zaawansowanych mechanik i łatwe zarządzanie scenami. Unity oferuje narzędzia do tworzenia zarówno grafiki 2D, jak i 3D, co było istotne dla wizualnej strony projektu. Narzędzia do zarządzania scenami umożliwiły płynne przechodzenie między różnymi etapami gry i kontrolowanie interakcji gracza.

Podczas projektowania gry wzięto pod uwagę również założenia techniczne dotyczące działania gry. Priorytetem była wydajność, dlatego gra została zoptymalizowana tak, aby działała płynnie na przeciętnych komputerach osobistych. Główną platformą docelową są komputery z systemem Windows, jednak dzięki możliwościom Unity, gra może być łatwo dostosowana do innych systemów operacyjnych. Struktura projektu została zaprojektowana w sposób skalowalny, co pozwala na łatwe dodawanie nowych treści takich jak dodatkowe sceny, postaci czy mini gry bez konieczności znaczących zmian w istniejącym kodzie.

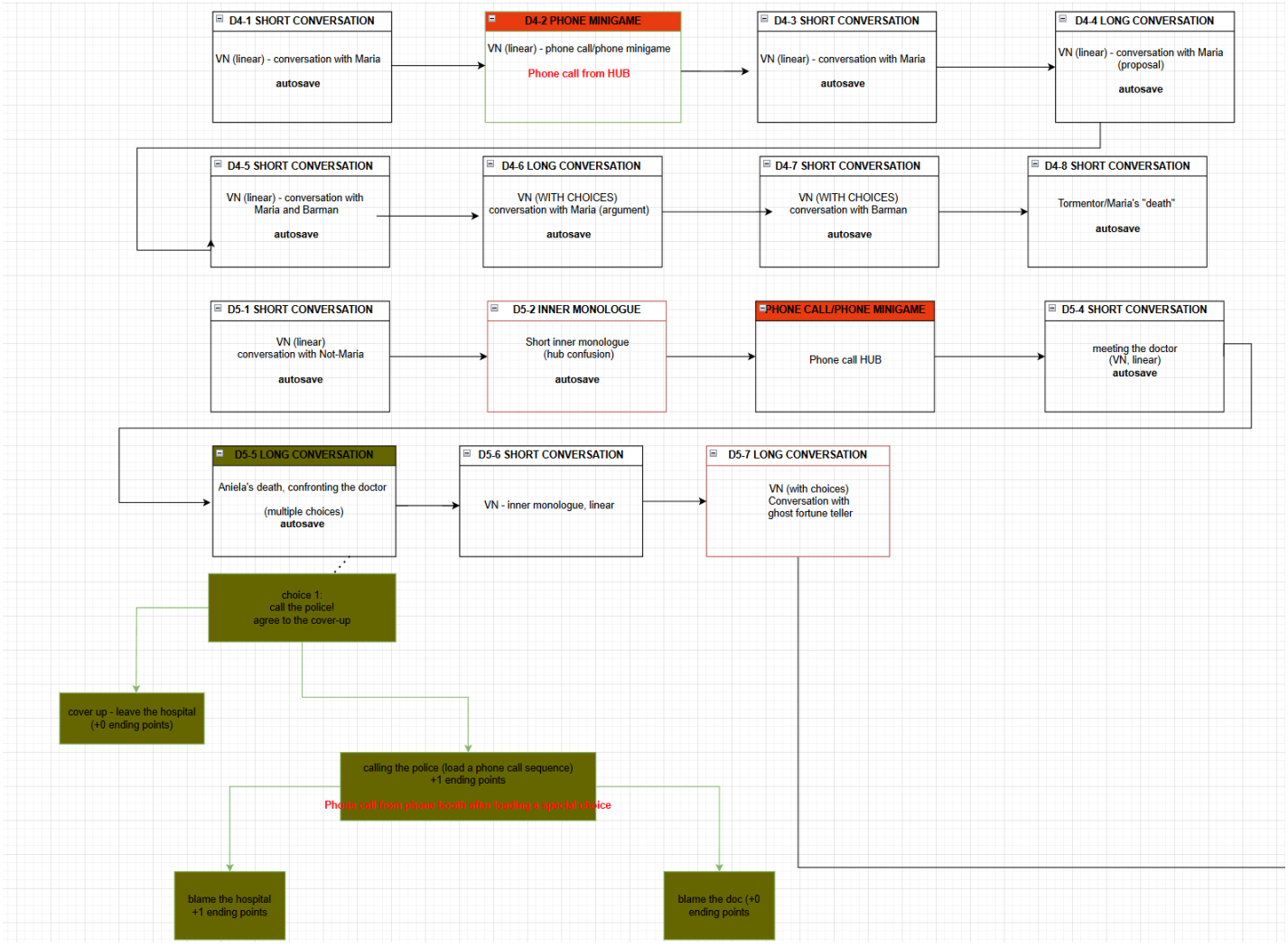
Założenia projektowe były kluczowe dla stworzenia gry, która w unikalny sposób łączy mechaniki *visual novel*, *point-and-click* oraz mini gier. Skupienie się na aspektach projektowych i programistycznych pozwoliło na stworzenie spójnej i przemyślanej mechaniki gry, która współgra z narracją i kontekstem fabularnym, dostarczając graczowi satysfakcjonującego i angażującego doświadczenia. Projektowana gra łączy w sobie liniową narrację z elementami nieliniowości, które pozwalają graczowi wpływać na rozwój fabuły oraz zakończenia. Struktura gry obejmuje dwa akty, a jej przebieg zależy od decyzji podejmowanych przez gracza szczególnie w kluczowych momentach, które mają bezpośredni wpływ na dalsze wydarzenia. Gra rozpoczyna się od pierwszego dnia w akcie pierwszym, wprowadzając gracza w realia Wolnego Miasta Gdańsk lat 20. XX wieku. Pierwsze dni są głównie liniowe, służąc jako wprowadzenie do świata gry postaci oraz podstawowych mechanik. Jednak w określonych momentach gra oferuje wybory, które wpływają na dialogi, relacje między postaciami i ostatecznie na zakończenie gry.



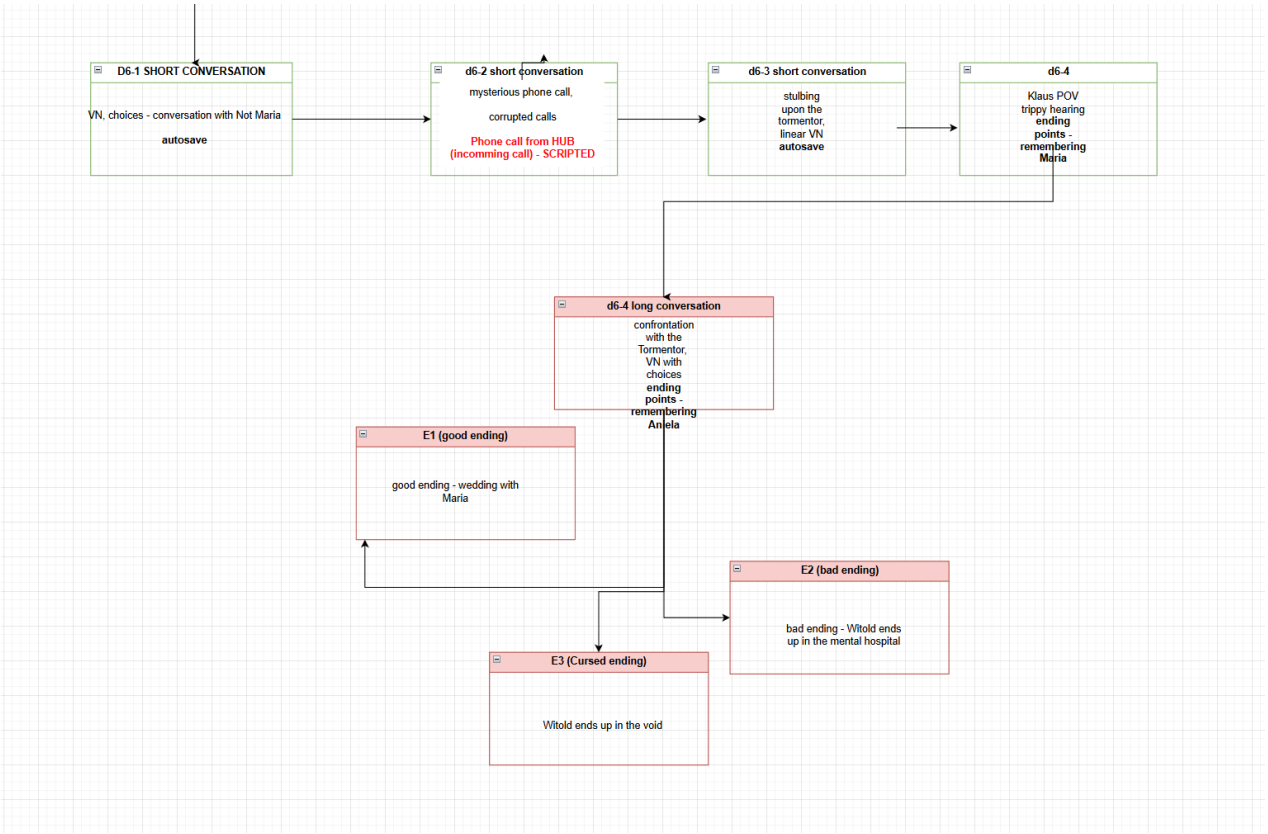
Rysunek 3.3 Diagram przedstawiający sekwencje pierwszych dwóch dni gry z programu draw.io



Rysunek 3.4 Diagram przedstawiający sekwencje dnia trzeciego z programu draw.io



Rysunek 3.5 Diagram przedstawiający sekwencje dnia czwartego oraz piątego z programu draw.io



Rysunek 3.6 Diagram przedstawiający sekwencje dnia szóstego wraz z rożnymi zakończeniami z programu draw.io

Na diagramach przedstawiających sekwencję gry można zauważyć, że poszczególne segmenty zostały oznaczone różnymi kolorami, co pomaga w identyfikacji ważnych momentów dla fabuły i struktury gry:

- **Niebieskie pola** reprezentują elementy zawierające mini gry, takie jak wybór podejrzanych na komisariacie czy dzwonienie przez telefon. Mini gry te są integralną częścią rozgrywki i muszą zostać ukończone, aby gracz mógł kontynuować fabułę. Przykładowo, podczas sceny na komisariacie gracz musi poprawnie zidentyfikować złodzieja na podstawie dostępnych dokumentów, co wpływa na zdobycie punktów potrzebnych do odblokowania dobrego zakończenia.
- **Żółte pola** wskazują na momenty, w których gracz dokonuje kluczowych wyborów wpływających na zakończenie gry. Choć w grze występuje wiele decyzji, te zaznaczone na żółto są szczególnie istotne dla rozwoju fabuły i relacji między postaciami. Na przykład: trzeciego dnia gracz staje przed wyborem, czy okłamać inną postać. Decyzja ta ma konsekwencje w późniejszych dialogach, zmieniając ich ton na bardziej pozytywny lub negatywny, co wpływa na ogólny odbiór postaci przez innych bohaterów.

- **Pomarańczowe pola** oznaczają sekwencje, w których gracz znajduje się w *hub'ie* centralnym miejscu umożliwiającym eksplorację i interakcję z różnymi obiektami. W *hub'ie* gracz może wykonywać określone czynności, takie jak korzystanie z radia, telefonu czy odkrywanie ukrytych mini gier, na przykład rozwiązanie zagadki w zegarze prowadzące do gry "piętnastka".

Trzeciego dnia gra oferuje bardziej rozbudowaną scenę dialogową z wieloma rozgałęzieniami. Gracz ma możliwość zdobycia większej liczby punktów wpływających na zakończenie, podejmując różne decyzje i angażując się w interakcje z innymi postaciami. W tej sekwencji wybory gracza mają bezpośredni wpływ na liczbę zdobytych punktów oraz na to, które zakończenie będzie dostępne pod koniec gry.

W trakcie całej rozgrywki gracz może zdobyć maksymalnie 14 punktów zakończenia. Aby odblokować dobre zakończenie, należy zgromadzić co najmniej 9 punktów. Punkty te są przyznawane za podjęcie właściwych decyzji w kluczowych momentach oraz za poprawne ukończenie mini gier, na przykład: prawidłowe wskazanie złodzieja podczas sceny na komisariacie czy uczciwe postępowanie wobec innych postaci zwiększa szanse na osiągnięcie pozytywnego finału.

Struktura gry została zaprojektowana tak, aby balansować między liniowością a nieliniowością. Liniowa sekwencja fabularna zapewnia spójność narracji i umożliwia graczowi zanurzenie się w świecie gry. Momentami jednak gra daje graczowi swobodę wyboru, co wpływa na rozwój postaci, relacje między nimi oraz ostateczne zakończenie historii. Takie podejście zachęca gracza do uważnej eksploracji dostępnych ścieżek i podejmowania przemyślanych decyzji.

Kluczowym elementem struktury gry jest system punktów zakończenia, który determinuje, jakie zakończenie będzie dostępne dla gracza. System ten został wprowadzony, aby zwiększyć możliwość ponownej rozgrywki i zachęcić gracza do odkrywania różnych możliwości fabularnych. Dzięki temu gracz może wielokrotnie przechodzić grę, podejmując inne decyzje i odkrywając nowe wątki oraz zakończenia. Struktura gry opiera się na połączeniu liniowej narracji z elementami nieliniowości, które umożliwiają graczowi wpływanie na przebieg fabuły i jej zakończenie. Poprzez wprowadzenie kluczowych wyborów, mini gier oraz interaktywnego *hub'a*, gra dostarcza angażującego doświadczenia, które zachęca do eksploracji i podejmowania świadomych decyzji. Celem było stworzenie spójnej struktury

gry, która integruje różne mechaniki i umożliwia graczowi aktywne uczestnictwo w kształtowaniu historii.

3.3. Założenia niefunkcjonalne

W ramach projektu zadbano o optymalizację kodu i zapewnienie płynności działania aplikacji na przeciętnych komputerach osobistych. System został zaprojektowany w środowisku Unity z myślą o kompatybilności z systemami Windows 10 i Windows 11, co umożliwia łatwą adaptację do nowych platform i wymagań. Istotne było także zapewnienie intuicyjnego interfejsu użytkownika, umożliwiającego bezproblemowe przechodzenie między różnymi trybami rozgrywki. Dodatkowym priorytetem była skalowalność projektu, umożliwiająca rozszerzanie treści – dodawanie nowych scen, postaci czy mini-gier – bez konieczności gruntownej przebudowy istniejącej architektury.

3.4. Założenia funkcjonalne

Podstawowym elementem jest silnik *visual novel*, odpowiedzialny za wyświetlanie postaci, tła oraz prezentowanie dialogów w dedykowanym oknie, z opcjami automatycznego i manualnego przechodzenia między kolejnymi liniami tekstu. System umożliwia dynamiczne sterowanie dzięki komendą sterujących – takich jak zmiana *sprite'ów,* odtwarzanie muzyki, efektów dźwiękowych czy zarządzanie czasem wyświetlania dialogów – na podstawie plików tekstowych. W projekcie zaimplementowano również mechanikę systemu wyborów, która umożliwia graczowi podejmowanie decyzji wpływających na rozwój fabuły oraz końcowe zakończenie. Integracja z interaktywnymi mini-grami, takimi jak scena dzwonienia przez telefon, interaktywne sceny *point-and-click* czy wybór podejrzanego na komisariacie, umożliwia zdobywanie dodatkowych informacji i punktów wpływających na przebieg gry. Każda z tych funkcji została opracowana w sposób modułowy, co pozwala na ich łatwe odłączenie i ponowne wykorzystanie w przyszłych projektach, niezależnie od gatunku czy specyfiki rozgrywki.

3.5. Projektowanie interaktywnych sekwencji i mini-gier

W procesie tworzenia gry jednym z kluczowych elementów było zaprojektowanie różnorodnych scen interaktywnych oraz mini gier, które uzupełniały wątek fabularny i zwiększały zaangażowanie gracza. Celem było wprowadzenie mechanik wykraczających poza

standardowy system *visual novel,* poprzez sekcje *point-and-click*, bardziej złożone łamigłówki oraz interakcje środowiskowe. Tego rodzaju podejście wpisuje się w obserwację, że: „Many games have very different rules during different parts of play. The rules often change completely from mode to mode, almost like completely separate games." [4] Wprowadzenie różnorodnych mechanik nie tylko wzbogaciło rozgrywkę, ale także zapewniło świeżość doświadczenia na każdym etapie gry. Pierwszym opracowanym modułem była sekwencja śledztwa w sprawie kradzieży butelki cennego koniaku ze sklepu. Ten fragment gry miał formę sceny *point-and-click*, w której gracz, wcielając się w detektywa, analizował tropy i rozmawiał ze świadkami. Początkowo istniała jedynie podstawowa lista przedmiotów do zbadania, obejmująca ślady butów, zbite szkło czy pozostawione fragmenty etykiety, pomagające nakierować na sprawcę. Z biegiem czasu scena została rozbudowana o wielostopniowy dialog ze sklepikarzem, kasę fiskalną, z której nic nie zginęło, sugestię o braku zainteresowania złodzieja pieniędzmi, rozbitą gablotę po skradzionej butelce, plakat z reklamą alkoholu umożliwiający wgląd w szczegóły produktu, pozostawioną czapkę typową dla pracowników fizycznych oraz ukryty pod znakiem fragment podłogi z odciskiem buta w rozmiarze ponad czterdzieści siedem. Te elementy tworzyły spójną, interaktywną scenę dającą graczowi swobodę eksploracji i stopniowego odkrywania prawdy o zdarzeniu.

Ważnym aspektem tej sceny jest projektowanie doświadczenia gracza, które wykracza poza samą treść fabularną. W tym kontekście warto odwołać się do słów: „There is a difference between designing the content and designing the end-user experience." [5] Podczas tworzenia gry nie wystarczy jedynie opracować zawartości, takich jak dialogi czy przedmioty do zbadania. Kluczowe jest również stworzenie takiego doświadczenia, które umożliwi graczowi płynne i intuicyjne interakcje z tymi elementami, zachowując przy tym spójność z ogólnym celem narracyjnym. Każdy element, od przedmiotów po dialogi musi być zaprojektowany z myślą o tym, jak wpłynie na odbiór całej sceny i na angażowanie gracza w rozwiązywanie zagadki.

Wykorzystane w scenie tropy stały się podstawą do kolejnej mini gry, w której zadaniem gracza było wybranie właściwego podejrzanego z listy dziewięciu potencjalnych sprawców. Podczas analizy dokumentów z posterunku policji gracz filtrował informacje, korzystając z wcześniej zebranych poszlak. Każdy z podejrzanych miał określone cechy, a tylko jedna osoba w pełni pasowała do ustalonego profilu. Aby nie uczynić zadania zbyt trudnym, przewidziano możliwość wytypowania dwóch osób, zwiększając szanse gracza na poprawny wybór. Miejscem akcji było interaktywne biurko, po którym można było

przesuwać dokumenty, korzystając ze strzałek ekranowych. Tego typu zaprojektowanie przestrzeni zapewniło płynną nawigację i łatwość zarządzania informacjami, nie ograniczając się do statycznego ekranu.

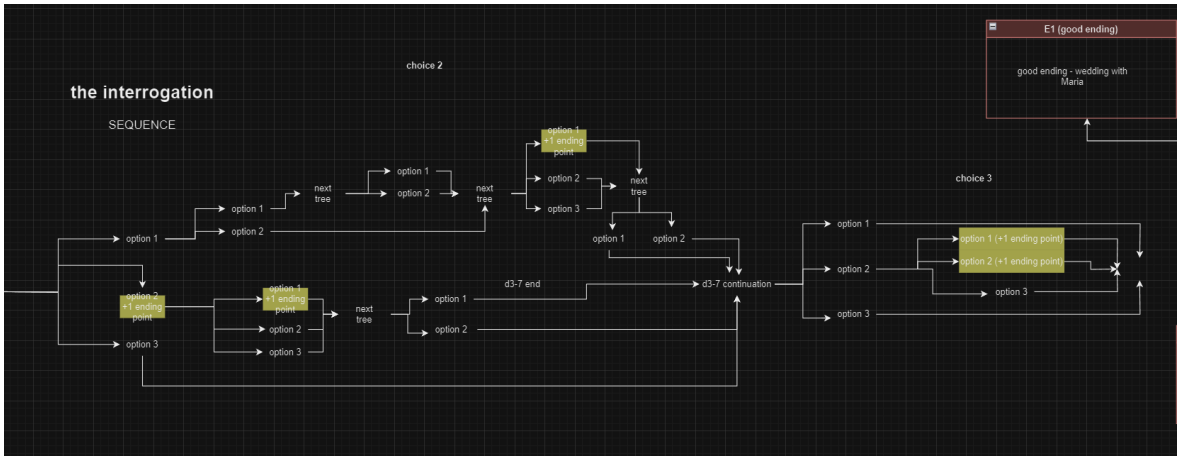
W efekcie, gracz musiał połączyć fakty z poprzedniej sceny z analizą dostępnych akt, aby wyselekcjonować właściwego przestępcę, co jednocześnie pogłębiało immersję w świat przedstawiony i wzmacniało wagę wcześniejszych działań. Przykład ten doskonale pokazuje, jak projektowanie samej treści gry – tropów, postaci, interakcji – jest nierozerwalnie związane z projektowaniem doświadczenia użytkownika, który na każdym etapie podejmuje decyzje i wpływa na rozwój narracji. Kolejnym interaktywnym elementem było radio umieszczone w centralnym *hub'ie*. Zaprojektowane z myślą o autentyczności, pozwalało na przełączanie się między stacjami z różnych regionów, od Warszawy przez Gdańsk, aż po francuskie i amerykańskie rozgłośnie, wszystkie prezentujące muzykę z lat 20. i 30. Wprowadzenie radia do gry miało na celu ożywienie *hub'u* i dostarczenie graczowi rozrywki oraz odprężenia między głównymi scenami śledczymi, a także podkreślenie specyficznego klimatu epoki i wielokulturowego charakteru ówczesnego Gdańska.

Rysunek 3.7 Rysunek przedstawia radio w stylu przedwojennym. Opracowanie graficzne Ambre Buathier

W późniejszym etapie rozwoju gry pojawiła się kolejna scena *point-and-click,* koncentrująca się na śledztwie związanym z zabójstwem romskiej wróżki. Podobnie jak w przypadku kradzieży, również i tutaj należało przeanalizować miejsce zbrodni, odnaleźć

właściwe tropy, takie jak narzędzie zbrodni, ślady krwi czy spalone karty tarota, które ofiara miała przy sobie. Wstępne wersje projektu uwzględniały elementy, które ostatecznie zostały zmienione, np. początkową obecność policyjnej taśmy zastąpiono sytuacją w której detektyw przybywa na miejsce przed innymi funkcjonariuszami. Ten proces iteracyjny pozwolił na stworzenie bardziej wiarygodnej i dynamicznej sceny, w której gracz samodzielnie dochodził do wniosków, jakie posłużą mu w późniejszym przesłuchaniu i docelowym wyjaśnieniu sprawy.

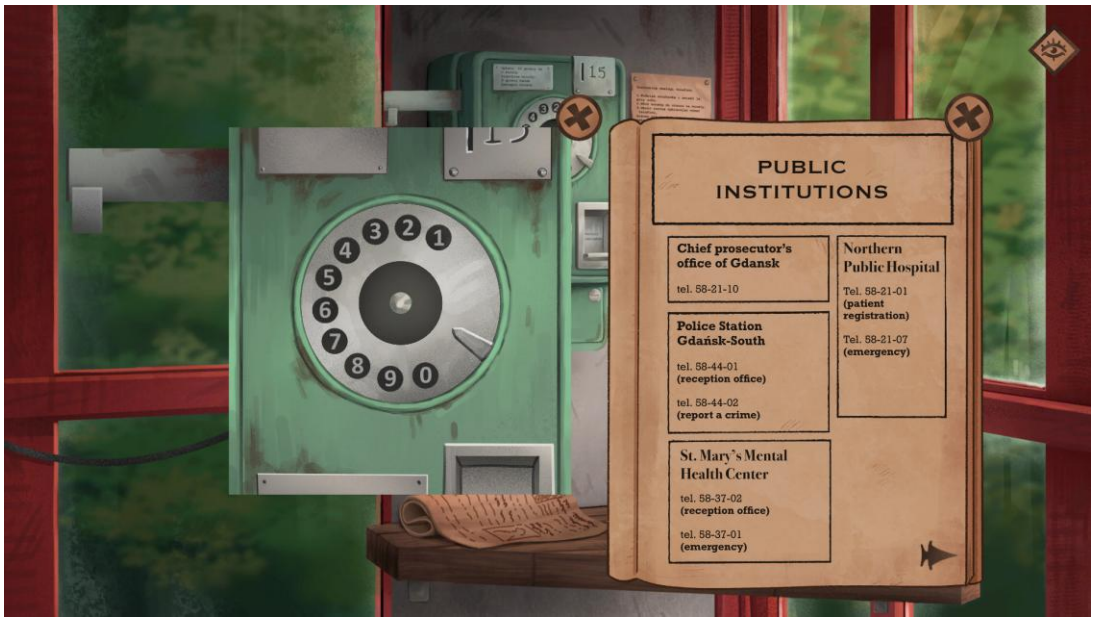
Następny moduł, wymagający od gracza logicznego myślenia i korzystania z uprzednio zebranych informacji, obejmował rozbudowane przesłuchanie podczas którego gracz musiał bronić się przed zarzutami zwierzchnika. Przesłuchanie zaprojektowano jako rozgałęzione drzewko dialogowe, w którym choć finał zawsze był podobny, poszczególne odpowiedzi umożliwiały zdobywanie punktów wpływających na zakończenie gry. Dobra argumentacja i właściwe wykorzystanie wiedzy zgromadzonej w poprzednich scenach przekładały się na wyższy wynik, a tym samym przybliżały gracza do pozytywnego finału opowieści.



Rysunek 3.8 Rysunek przedstawia diagram sceny przesłuchania

Ostatnim istotnym elementem, w pewnym sensie mini-grą samą w sobie, było dzwonienie przez telefon z autentycznie odtworzonym systemem wybierania numerów w stylu starych aparatów telefonicznych z okrągłą tarczą. Ten mechanizm pozwalał odtworzyć charakterystyczne czynności i wymagał od gracza większej precyzji i skupienia. Dodatkowo, sceny z telefonem umożliwiały przeglądanie książki telefonicznej, zawierającej istotne numery postaci i instytucji. Dzwonienie odblokowywało kolejne fragmenty fabuły, a jednocześnie pozwalało na wykonanie połączeń opcjonalnych, przygotowanych aby wzbogacić

świat gry. Takie rozwinięcie funkcjonalności telefonu podkreślało nieliniowy charakter rozgrywki, zachęcając gracza do eksploracji i odkrywania nowych interakcji.



Rysunek 3.9 Rysunek przedstawia telefon w stylu przedwojennym. Opracowanie graficzne Ambre Buathier

Wszystkie opisane interaktywne sekwencje i mini gry zostały zintegrowane z systemem *visual novel*, który odpowiadał za narrację, wyświetlanie dialogów oraz zarządzanie punktami decydującymi o zakończeniach. Ten system pozwalał na wyświetlanie postaci i scen oraz wprowadzanie komend takich jak *append* czy *wait*, a także na pomijanie i automatyczne odtwarzanie dialogów. Dodatkowo, zapewniał możliwość dynamicznej zmiany *sprite'ów* postaci czy wywoływania scen z innymi mechanikami gry w taki sposób, aby gracz płynnie przechodził od czytania dialogów do udziału w mini grach czy eksploracji otoczenia. Zliczane w tle punkty zapewniały interaktywne powiązanie między podejmowanymi decyzjami a dalszym rozwojem historii, wzmacniając wrażenie uczestnictwa w żywym, reagującym na działania gracza świecie.

W rezultacie, projektowanie interaktywnych sekwencji narracyjnych i mini gier stanowiło kluczowy element pracy nad grą. Tworzenie scen śledczych, łamigłówek, interaktywnych przedmiotów czy nietypowych mechanik, takich jak telefon z tarczą obrotową, w połączeniu z systemem *visual novel*, zaowocowało bogatszym doświadczeniem dla gracza. Przyjęta metodologia iteracyjnego rozwoju, stopniowego rozbudowywania każdej sceny i przemyślanego łączenia elementów narracyjnych z interaktywnymi, pozwoliła stworzyć spójną, a zarazem różnorodną rozgrywkę, która angażuje gracza na wielu poziomach.

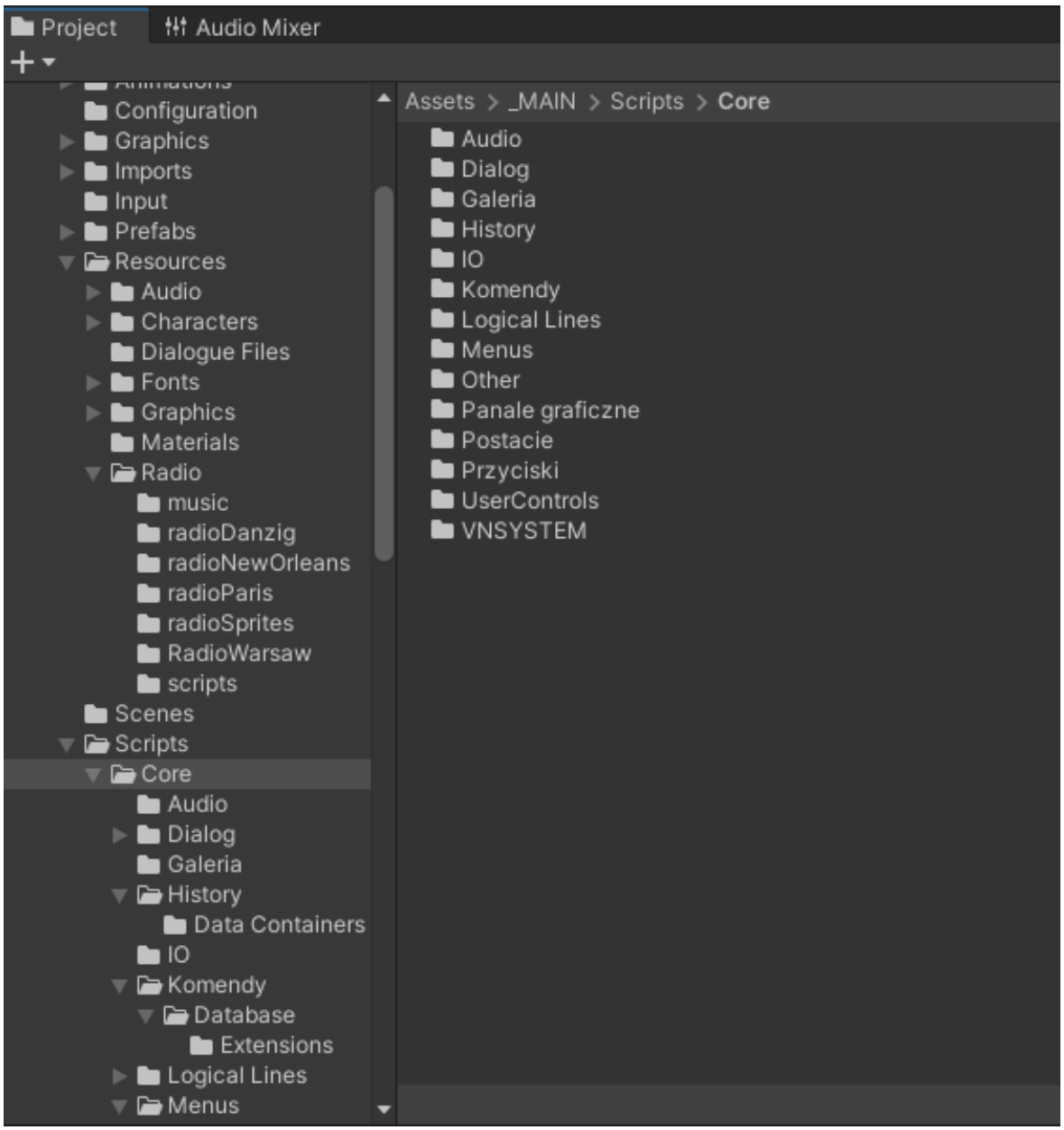
4. Implementacja

W procesie implementacji kluczowym celem było przeniesienie wszystkich założeń i pomysłów z dokumentów projektowych do działającej formy, tworząc spójny i interaktywny produkt. Choć część mini gier i mechanik została zaplanowana jeszcze na etapie projektowania, pewne elementy wymagały uzupełnienia i dopracowania już w trakcie prac programistycznych. Dzięki temu proces implementacji nie tylko stanowił realizację gotowych koncepcji, ale również stwarzał przestrzeń do wprowadzania nowych pomysłów i modyfikacji.

W dalszych podpunktach rozdziału zostaną szczegółowo omówione poszczególne aspekty tego etapu prac, takie jak przygotowanie systemu *visual novel* wraz z funkcjami wyświetlania *sprite'ów* postaci, interpretacją plików tekstowych zawierających dialogi i komendy sterujące narracją, czy też stworzenie dedykowanych scen w środowisku Unity dla określonych fragmentów rozgrywki. Istotnym elementem było również wdrożenie różnorodnych mini gier oraz konfiguracja interfejsu użytkownika w Unity, które uzupełniały główną narrację i wzbogacały całościowe doświadczenie. Wszystkie te zadania, choć oparte na wcześniejszych założeniach, zyskały swój ostateczny kształt podczas implementacji, co pozwoliło na dostosowanie rozgrywki do celów i wymagań projektu.

4.1 Implementacja systemu visual novel

W ramach implementacji kluczowych elementów gry konieczne było opracowanie kompleksowego systemu *visual novel*, który umożliwiłby płynne prezentowanie fabuły, kontrolę dialogów, zarządzanie wyborem ścieżek i podejmowaniem decyzji przez gracza, a także integrację z innymi komponentami, takimi jak mini gry czy interaktywne przedmioty. System ten został zaprojektowany jako zestaw skryptów i struktur logicznych, współpracujących z interfejsem użytkownika, dźwiękiem oraz obiektami Unity w celu stworzenia spójnego doświadczenia narracyjnego.



Rysunek 4.1 Struktura folderów skryptów systemu visual novel

Podstawowym zadaniem systemu *visual novel* jest interpretacja i wykonywanie poleceń zapisanych w plikach tekstowych, zawierających zarówno dialogi, jak i komendy kontrolne. Dzięki temu osoba projektująca fabułę może wygodnie tworzyć i modyfikować treść gry bez konieczności ingerencji w kod źródłowy. Komendy te obejmują m.in. ustawianie tła, wyświetlanie *sprite'ów* postaci, odtwarzanie muzyki i efektów dźwiękowych, wstrzymywanie narracji na określony czas, czyszczenie okienka dialogowego, a także zarządzanie logiką (taką jak nadawanie punktów za określone decyzje).

Funkcjonalność warstwy audio pozwala na odtwarzanie muzyki, dźwięków otoczenia oraz pojedynczych efektów dźwiękowych, np. odgłosu pukania do drzwi. Dźwięki można zapętlać lub odtwarzać jednorazowo, co ułatwia tworzenie zmieniającej się atmosfery scen. Interfejs użytkownika UI został zintegrowany z systemem dialogowym, dzięki czemu gracze mogą pomijać tekst, aktywować automatyczne odtwarzanie kolejnych linii dialogowych lub wybierać odpowiedzi za pomocą przycisków na ekranie. System postaci w *visual novel* opiera się na obiektach Unity, którym można przypisać *sprite'y* o różnej warstwowości. W zależności od potrzeb postać może składać się z kilku elementów (takich jak twarz i ciało) nakładających się na siebie, umożliwiając łatwą zmianę wyrazu twarzy czy pozy. W omawianym projekcie zastosowano uproszczoną wersję, wykorzystującą głównie pojedyncze *sprite'y*, co uprościło implementację oraz zarządzanie wyświetlanymi grafikami.

W tym kontekście ważnym aspektem implementacji jest sposób, w jaki system reaguje na różnorodne zdarzenia w grze. Zgodnie z zasadą: „When an event, message or command is received by a game object, it needs to respond to the event in some way. This is known as handling the event, and it is usually implemented by a function or snippet of script code called event handler. ” [6] W praktyce oznacza to, że każdy obiekt w grze, niezależnie od tego, czy jest to postać, przedmiot czy środowisko, musi być zaprojektowany tak, aby odpowiednio reagował na interakcje gracza, takie jak wybory w dialogach, kliknięcia czy aktywacje obiektów. Implementacja takich mechanizmów za pomocą dedykowanych funkcji event handler pozwala na spójną i dynamiczną reakcję obiektów na zachowanie gracza, co jest kluczowe dla zachowania płynności rozgrywki.

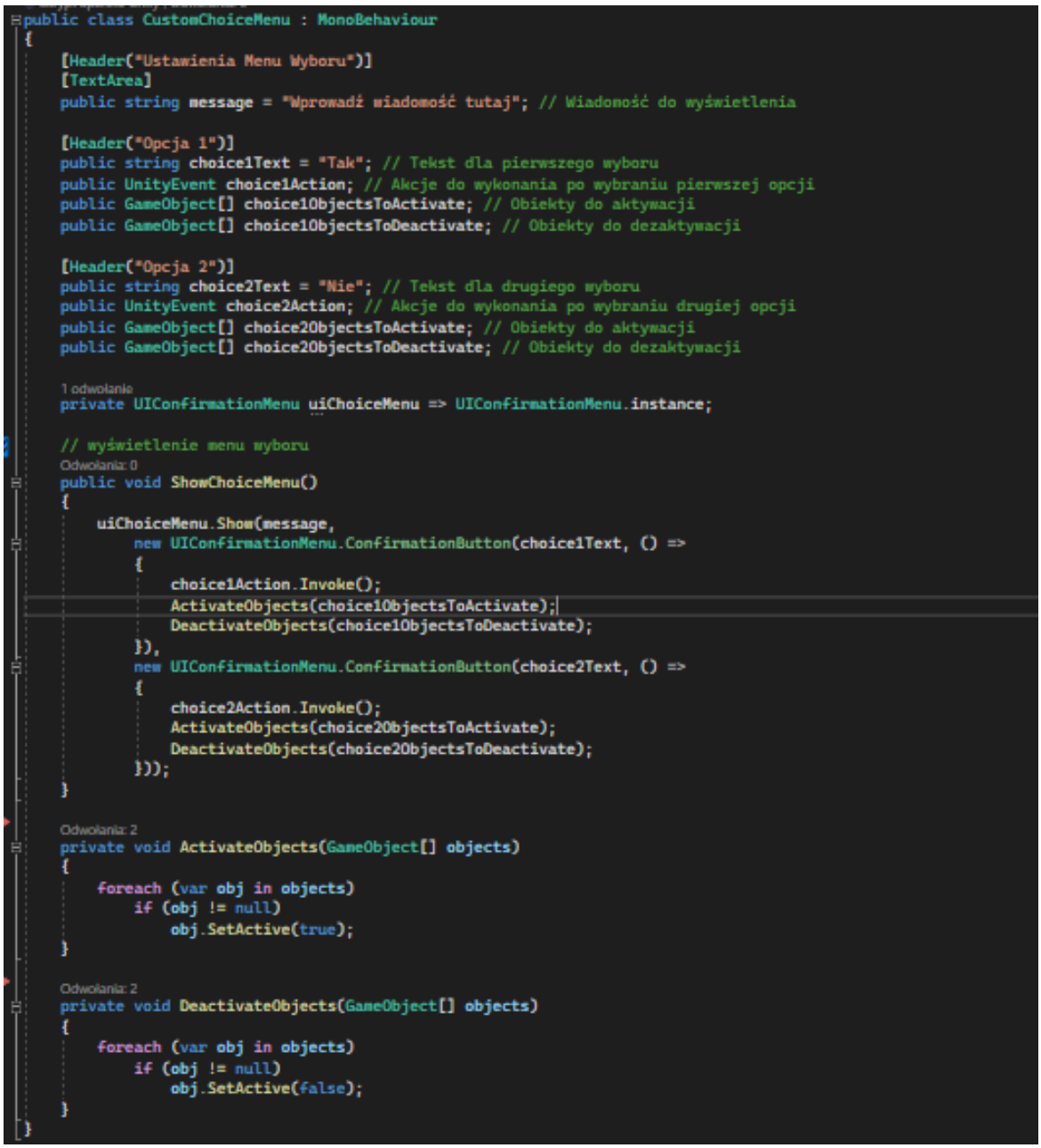
Poniżej przedstawiony został fragment kodu odpowiedzialny za interpretację przykładowych komend. Pierwsza metoda dezaktywuje komponent SpriteRenderer wybranego obiektu w scenie, co umożliwia chwilowe ukrycie postaci bez usuwania jej ze świata gry. Druga metoda wczytuje nową scenę, co jest przydatne np. po zakończeniu sekwencji

dialogowej, aby przejść do kolejnej lokacji lub *hub'u*. Trzecia metoda wykonuje automatyczny zapis stanu gry, pozwalając utrwalić postęp gracza.

```
1 odwołanie
private static void DeaktywujSpriteRenderer(string objectName)
{
    GameObject obj = GameObject.Find(objectName);
    if (obj != null)
    {
        SpriteRenderer spriteRenderer = obj.GetComponent<SpriteRenderer>();// Dezaktywacja komponentu SpriteRenderer na wskazanym obiekcie(postaci)
        if (spriteRenderer != null)
        {
        }
        else
        {
            Debug.LogWarning($"Obiekt '{objectName}' nie posiada komponentu SpriteRenderer.");
        }
    }
    else
    {
        Debug.LogWarning($"Obiekt '{objectName}' nie został znaleziony.");
    }
}
1 odwołanie
private static void LoadNextScene(string data)
{
    SceneManager.LoadScene(data);
}
// Wykonanie polecenia automatycznego zapisu
1 odwołanie
public static void AutoSaveCommand()
{
    if (VNGameSave.activeFile != null)
    {
        VNGameSave.activeFile.AutoSave();
        Debug.Log("Wykonano automatyczny zapis.");
    }
    else
    {
        Debug.LogWarning("Nie znaleziono aktywnego pliku zapisu. Nie można wykonać automatycznego zapisu.");
    }
}
1 odwołanie
private static IEnumerator HideDialogueSystem(string[] data)
```

Rysunek 4.2 Skrypt komend

Poza podstawowymi poleceniami, system *visual novel* obsługuje również menu wyborów. Przykładowo, po kliknięciu w obiekt na scenie gracz może otrzymać pytanie o wykonanie konkretnej akcji, a wybrana odpowiedź wpłynie na przebieg narracji, aktywowanie lub dezaktywowanie obiektów, a nawet wywołanie zewnętrznych funkcji. Poniższy przykładowy skrypt pokazuje, jak wykorzystać UnityEvent i interakcję z hierarchią obiektów Unity, aby sterować aktywnością komponentów gry zależnie od decyzji podjętych w menu wyboru.



```
public class CustomChoiceMenu : MonoBehaviour
{
    [Header("Ustawienia Menu Wyboru")]
    [TextArea]
    public string message = "Wprowadź wiadomość tutaj"; // Wiadomość do wyświetlenia

    [Header("Opcja 1")]
    public string choice1Text = "Tak"; // Tekst dla pierwszego wyboru
    public UnityEvent choice1Action; // Akcje do wykonania po wybraniu pierwszej opcji
    public GameObject[] choice1ObjectsToActivate; // Obiekty do aktywacji
    public GameObject[] choice1ObjectsToDeactivate; // Obiekty do dezaktywacji

    [Header("Opcja 2")]
    public string choice2Text = "Nie"; // Tekst dla drugiego wyboru
    public UnityEvent choice2Action; // Akcje do wykonania po wybraniu drugiej opcji
    public GameObject[] choice2ObjectsToActivate; // Obiekty do aktywacji
    public GameObject[] choice2ObjectsToDeactivate; // Obiekty do dezaktywacji

    1 odwołanie
    private UIConfirmationMenu uiChoiceMenu => UIConfirmationMenu.instance;

    // wyświetlenie menu wyboru
    Odwołania: 0
    public void ShowChoiceMenu()
    {
        uiChoiceMenu.Show(message,
            new UIConfirmationMenu.ConfirmationButton(choice1Text, () =>
            {
                choice1Action.Invoke();
                ActivateObjects(choice1ObjectsToActivate);
                DeactivateObjects(choice1ObjectsToDeactivate);
            }),
            new UIConfirmationMenu.ConfirmationButton(choice2Text, () =>
            {
                choice2Action.Invoke();
                ActivateObjects(choice2ObjectsToActivate);
                DeactivateObjects(choice2ObjectsToDeactivate);
            }));
    }

    Odwołania: 2
    private void ActivateObjects(GameObject[] objects)
    {
        foreach (var obj in objects)
            if (obj != null)
                obj.SetActive(true);
    }

    Odwołania: 2
    private void DeactivateObjects(GameObject[] objects)
    {
        foreach (var obj in objects)
            if (obj != null)
                obj.SetActive(false);
    }
}
```

Rysunek 4.3 Skrypt menu wyboru

System *visual novel* wykorzystuje pliki dialogowe do definiowania treści fabularnej i interakcji między postaciami. Takie podejście ułatwia rozbudowę scenariusza oraz implementację różnych mechanik narracyjnych. Przykładowo, jeśli gracz zadzwoni pod numer, który nie został przewidziany w fabule, system może automatycznie odtworzyć efekt dzwonienia, wstrzymać narrację, a następnie wygenerować komunikat o braku odpowiedzi, po czym gracz wraca do lokacji z budką telefoniczną.

Kluczowym elementem obsługi plików dialogowych jest skrypt *DialogueParser*, odpowiedzialny za interpretację i przetwarzanie wpisanych linijek tekstu. Jego głównym zadaniem jest analiza surowej linijki z pliku tekstowego i rozbicie jej na trzy główne komponenty:

postać wypowiadającą kwestię, treść dialogu oraz dodatkowe komendy sterujące. Proces ten jest realizowany w metodzie *Parse*, która wykorzystuje funkcję *RipContent* do identyfikacji struktury linijki dialogowej.

```
8       public class DialogueParser
9       {
10          private const string commandRegexPattern = @"[\w\[\]]*[^\s]\(";
11
12          public static DialogueLine Parse(string rawLine)
13          {
14              string commands) = RipContent(rawLine);
15              commands = TagManager.Inject(commands);
16              return new DialogueLine (rawLine, speaker, dialogue, commands);
17          }
18
19          private static (string, string, string) RipContent(string rawLine)
20          {
21              string speaker = "", dialogue = "", commands = "";
22              int dialogueStart = -1;
23              int dialogueEnd = -1;
24              bool isEscaped = false;
25              for (int i = 0; i < rawLine.Length; i++)
26              {
27                  char current = rawLine[i];
28                  if (current == '\\')
29                      isEscaped = !isEscaped;
30                  else if (current == '"' && !isEscaped)
31                  {
32                      if (dialogueStart == -1)
33                          dialogueStart = i;
34                      else if (dialogueEnd == -1)
35                          dialogueEnd = i;
36                  }
37                  else
38                      isEscaped = false;
39              }
40              Regex commandRegex = new Regex(commandRegexPattern);
41              MatchCollection matches = commandRegex.Matches(rawLine);
42              int commandStart = -1;
43              foreach (Match match in matches)
44              {
45                  if (match.Index < dialogueStart || match.Index > dialogueEnd)
46                  {
47                      commandStart = match.Index;
48                      break;
49                  }
50              }
51              if (commandStart != -1 && (dialogueStart == -1 && dialogueEnd == -1))
52                  return ("", "", rawLine.Trim());
```

Rysunek 4.4 Skrypt parsera dialogów

Mechanizm rozpoznawania dialogu opiera się na przeszukiwaniu ciągu znaków pod kątem odpowiednio sformatowanych treści. Algorytm iteruje przez tekst, analizując występowanie cudzysłowów, które oznaczają początek i koniec wypowiedzi. Dodatkowo, stosowana jest obsługa znaków ucieczki, co pozwala na poprawną interpretację znaków specjalnych wewnątrz dialogów. Równolegle, przy użyciu wyrażenia regularnego, system

wykrywa i wyodrębnia komendy sterujące znajdujące się poza zakresem samej wypowiedzi. Integracja *Dialogue Parser* z *Command Manager* umożliwia dynamiczne sterowanie przebiegiem gry poprzez obsługę poleceń zawartych w plikach scenariusza. Dzięki temu można np. zmieniać pozycję postaci na ekranie, odtwarzać dźwięki lub animacje w odpowiednich momentach dialogu. Ponadto, współpraca *parsera* z *History Manager* zapewnia, że zapisane stany gry przechowują nie tylko treść rozmów, ale również powiązane z nimi polecenia, co pozwala na dokładne odwzorowanie sceny po wczytaniu zapisu.

Przykładowy fragment pliku dialogowego:

```
SetLayerMedia(background BG_Phone_Lifted)
PlaySFX(sfx_calling)
wait(2)
StopSFX(sfx_calling)
PlaySFX(sfx_calling)
wait(2)
StopSFX(sfx_calling)
PlaySFX(sfx_calling)
wait(2)
StopSFX(sfx_calling)
Witold "(I can hear a signal, but apparently no one answers the phone. I wait for the call to end and then hang up.)"
loadnextscene("PhoneBoth")
```

Rysunek 4.5 Plik dialogowy z komendami

W omawianym systemie *visual novel* kluczową rolę odgrywają dedykowane menedżery, które ułatwiają zarządzanie poszczególnymi aspektami rozgrywki. *Character Manager* odpowiada za zarządzanie postaciami, umożliwiając m.in. ich pozycjonowanie na ekranie i zmianę wyrazów twarzy. *Command Manager* obsługuje interpretację komend sterujących, co pozwala na dynamiczną kontrolę przebiegu gry, jak wspomniano wcześniej w kontekście parsowania plików scenariusza. Równie istotne są *Audio Manager* oraz *History Manager*, które pełnią kluczową funkcję w zapewnieniu spójności rozgrywki i prawidłowego działania systemu zapisów. System historii *History Manager* przechowuje i zarządza *logami* dialogów, co jest istotnym elementem mechanizmu zapisów *save system*. Dzięki poprawnie zapisanym danym w strukturach możliwe jest dokładne odtworzenie wcześniejszych stanów gry, takich jak pozycja postaci, aktywne efekty dźwiękowe czy dialogi. Przykładowo, jeśli postać została przesunięta na lewą stronę ekranu kilka komend wcześniej, system historii zapewni, że po wczytaniu zapisu jej pozycja pozostanie niezmieniona. Analogicznie działa zarządzanie muzyką, jeśli ścieżka dźwiękowa została odtworzona przed kilkoma linijkami

kodu, system historii pozwoli na jej prawidłowe przywrócenie, co zapobiega przypadkowym zmianom w oprawie dźwiękowej po wczytaniu zapisu. Przykładem wykorzystania tego mechanizmu jest metoda przedstawiona na rysunku, odpowiedzialna za odtwarzanie zapisanych w historii ścieżek audio:

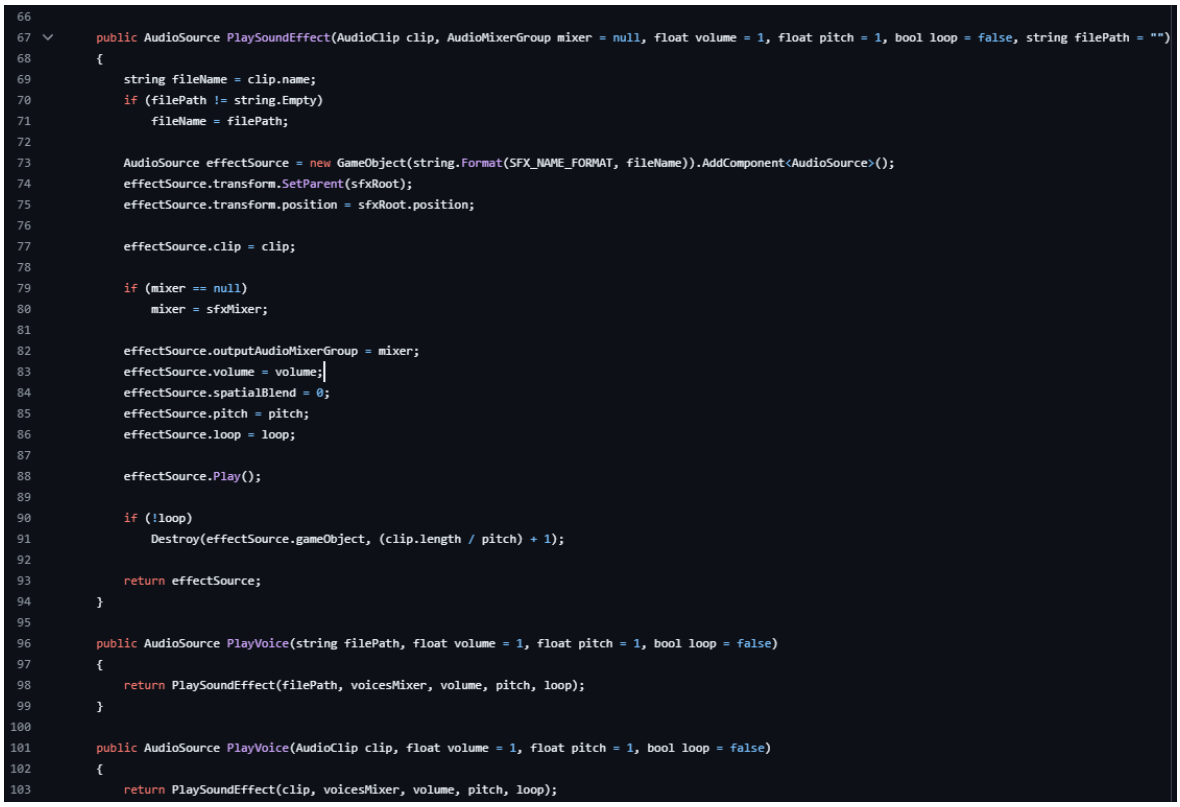
```
48          public static void Apply(List<AudioTrackData> data)
49          {
50              List<int> cache = new List<int>();
51
52              foreach (var channelData in data)
53              {
54                  AudioChannel channel = AudioManager.instance.TryGetChannel(channelData.channel, createIfDoesNotExist: true)
55                  if (channel.activeTrack == null || channel.activeTrack.name != channelData.trackName)
56                  {
57                      AudioClip clip = HistoryCache.LoadAudio(channelData.trackPath);
58                      if (clip != null)
59                      {
60                          channel.StopTrack(immediate: true);
61                          channel.PlayTrack(clip, channelData.loop, channelData.trackVolume, channelData.trackVolume, channel
62                      }
63                      else
64                          Debug.LogWarning($"History State: Could not load audio track '{channelData.trackPath}'");
65                  }
66
67                  cache.Add(channelData.channel);
68              }
69
70              foreach (var channel in AudioManager.instance.channels)
71              {
72                  if (!cache.Contains(channel.Value.channelIndex))
73                      channel.Value.StopTrack(immediate: true);
74              }
75          }
```

Rysunek 4.6 Rysunek przedstawia skrypt data container odpowiedzialny za obsługę audio

Kod ten pełni kluczową funkcję w przywracaniu stanu dźwięku podczas wczytywania gry. Najpierw iteruje przez listę zapisanych danych *audio data*, a następnie sprawdza, czy odpowiednie kanały dźwiękowe istnieją i czy odtwarzają właściwe ścieżki. Jeśli nie, system próbuje załadować odpowiedni plik audio z historii *HistoryCache.LoadAudio(channelData.trackPath)* i odtwarza go na właściwym kanale. Jeśli plik nie jest dostępny, generowane jest ostrzeżenie w konsoli deweloperskiej. Na końcu metoda przeszukuje listę aktywnych kanałów dźwiękowych w *Audio Manager* i zatrzymuje te, które nie znajdują się na liście cache, co zapobiega pozostawaniu niepotrzebnych dźwięków po wczytaniu zapisu.

Dzięki takiemu podejściu system *visual novel* gwarantuje, że stan gry po wczytaniu zapisu będzie odpowiadał rzeczywistej sytuacji sprzed jego utworzenia – zarówno pod względem wizualnym (dzięki *Character Manager*), jak i dźwiękowym (*Audio Manager* oraz *History Manager*).

Obsługa efektów dźwiękowych i dialogów w systemie *visual novel* opiera się na dynamicznym tworzeniu źródeł dźwięku, które są następnie konfigurowane i odtwarzane w odpowiednim momencie gry. Kluczową rolę odgrywa tutaj metoda *PlaySoundEffect*, odpowiedzialna za inicjalizację i kontrolę efektów dźwiękowych.



```
66
67      public AudioSource PlaySoundEffect(AudioClip clip, AudioMixerGroup mixer = null, float volume = 1, float pitch = 1, bool loop = false, string filePath = "")
68      {
69          string fileName = clip.name;
70          if (filePath != string.Empty)
71              fileName = filePath;
72
73          AudioSource effectSource = new GameObject(string.Format(SFX_NAME_FORMAT, fileName)).AddComponent<AudioSource>();
74          effectSource.transform.SetParent(sfxRoot);
75          effectSource.transform.position = sfxRoot.position;
76
77          effectSource.clip = clip;
78
79          if (mixer == null)
80              mixer = sfxMixer;
81
82          effectSource.outputAudioMixerGroup = mixer;
83          effectSource.volume = volume;
84          effectSource.spatialBlend = 0;
85          effectSource.pitch = pitch;
86          effectSource.loop = loop;
87
88          effectSource.Play();
89
90          if (!loop)
91              Destroy(effectSource.gameObject, (clip.length / pitch) + 1);
92
93          return effectSource;
94      }
95
96      public AudioSource PlayVoice(string filePath, float volume = 1, float pitch = 1, bool loop = false)
97      {
98          return PlaySoundEffect(filePath, voicesMixer, volume, pitch, loop);
99      }
100
101     public AudioSource PlayVoice(AudioClip clip, float volume = 1, float pitch = 1, bool loop = false)
102     {
103         return PlaySoundEffect(clip, voicesMixer, volume, pitch, loop);
```

Rysunek 4.7 Skrypt Audio Manager

Po wywołaniu tej funkcji tworzony jest nowy obiekt *GameObject*, do którego przypisywany jest komponent *AudioSource*. Jego parametry, takie jak głośność, wysokość tonu i możliwość zapętlenia, są konfigurowane zgodnie z przekazanymi argumentami. Jeśli nie podano konkretnego miksera dźwięku, metoda przypisuje domyślną grupę miksującą dla efektów specjalnych. Dźwięk jest odtwarzany natychmiast po utworzeniu źródła, a jeśli nie został ustawiony jako zapętlony, jego obiekt zostaje automatycznie usunięty po zakończeniu odtwarzania, co pozwala uniknąć zbędnego zużycia pamięci i zasobów systemowych. Analogicznie działa metoda *PlayVoice*, jednak została ona zoptymalizowana do odtwarzania dialogów postaci, wykorzystując osobny mikser głosowy. Dzięki temu możliwe jest oddzielenie efektów dźwiękowych od ścieżek dialogowych, co pozwala na lepszą kontrolę nad balansem audio w grze. Integracja tych funkcji z *History Manager* umożliwia ich poprawne przywracanie po wczytaniu zapisu gry, co zapobiega nagłym przerwom w odtwarzaniu

dźwięku lub jego nieoczekiwanym zmianom po powrocie do wcześniejszego stanu rozgrywki.

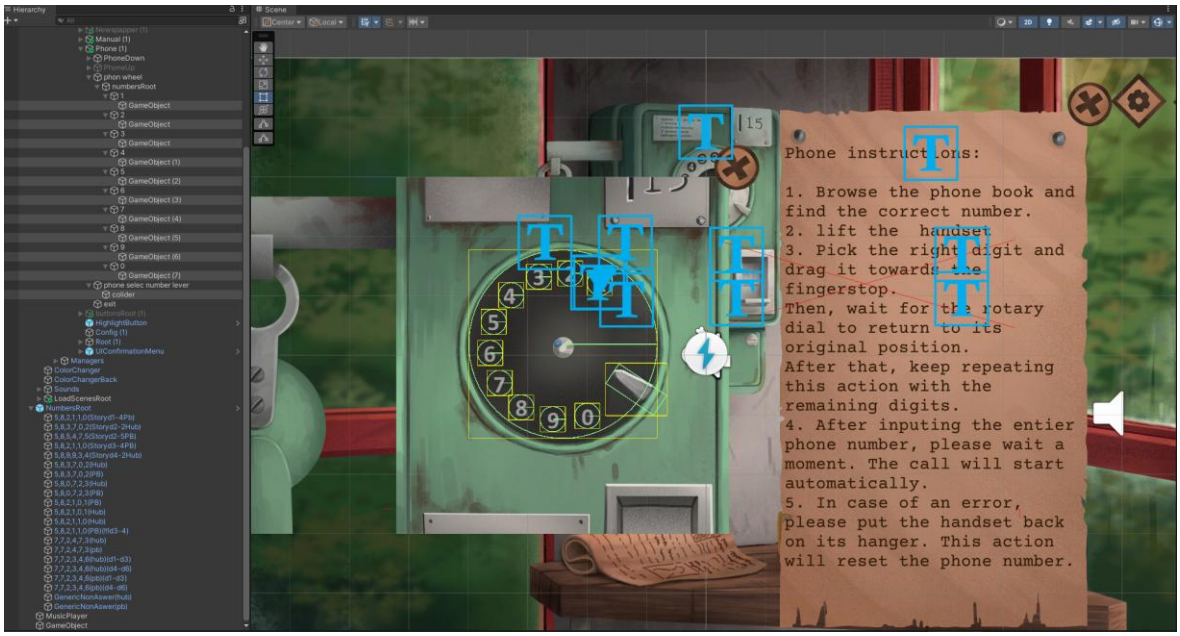
W ten sposób system *visual novel* staje się elastycznym fundamentem narracyjnym gry łączy interfejs, grafikę, dźwięk i logikę fabularną, umożliwiając sprawne dodawanie nowych wątków, scen czy interakcji bez konieczności ciągłego modyfikowania kodu źródłowego. W dalszych częściach pracy zostaną omówione inne aspekty implementacji, które współpracują z tym systemem, tworząc pełne i angażujące doświadczenie dla gracza.

4.2. Implementacja mechanik interaktywnych i mini-gier

W poprzednim rozdziale przedstawione zostały koncepcje i założenia dotyczące interaktywnych scen oraz mini gier. Na etapie implementacji każdy z tych elementów wymagał przełożenia idei i projektów na konkretne skrypty, komponenty oraz logikę osadzoną w środowisku Unity. W dalszych podpunktach tego rozdziału zostanie omówione w jaki sposób stworzono poszczególne mechaniki i mini gry, skupiając się na wykorzystaniu zasobów Unity (obiekty, *colidery, eventy*) i kodu w języku C#

Dwie główne sceny związane z telefonem – budka telefoniczna oraz mieszkanie głównego bohatera – bazują na podobnych zasadach interakcji. Każda z nich zawiera elementy takie jak książka telefoniczna z numerami, gazeta z informacjami fabularnymi, a także instrukcja obsługi telefonu, która ma ułatwić graczom zrozumienie mechaniki. W prawym górnym rogu ekranu dostępna jest ikonka „oka", dzięki której można podświetlić interaktywne przedmioty, jeśli graczowi umknęło, dokąd należy zadzwonić lub które elementy sceny są istotne. Wdrożenie tej sceny w Unity wykorzystuje standardowe komponenty silnika, takie jak przyciski i system zdarzeń (*Event System*). Na przykład kliknięcie w książkę telefoniczną powoduje aktywację obiektu zawierającego jej zawartość, umożliwiając graczowi przeglądanie numerów. W ten sposób całe sterowanie opiera się o prosty system: naciśnięcie przycisku wyzwala unity event, który aktywuje lub dezaktywuje wybrane obiekty. Dzięki temu gracz może swobodnie przełączać się między poszczególnymi elementami sceny (książka, gazeta, instrukcje, telefon). Kluczowym aspektem tej mini gry jest sama logika wybierania numeru na telefonie rotacyjnym. Scena z telefonem zawiera serię przycisków i *colider'ów* reprezentujących poszczególne cyfry oraz elementy ograniczające zakres obrotu tarczy. Aby zadzwonić pod wskazany numer, gracz przeciąga tarczę, a system sprawdza, z którym *colider'em* nastąpiła kolizja. Każdy „wykręcony" numer jest zapisywany w pamięci skryptu, a po wprowadzeniu pełnej kombinacji sprawdzane jest czy

istnieje pasująca sekwencja wywołująca określoną akcję – np. uruchomienie dialogu lub małej sceny fabularnej.



Rysunek 4.8 Scena przedstawia hierarchie oraz podświetlone colidery które odpowiadają za logikę telefonu

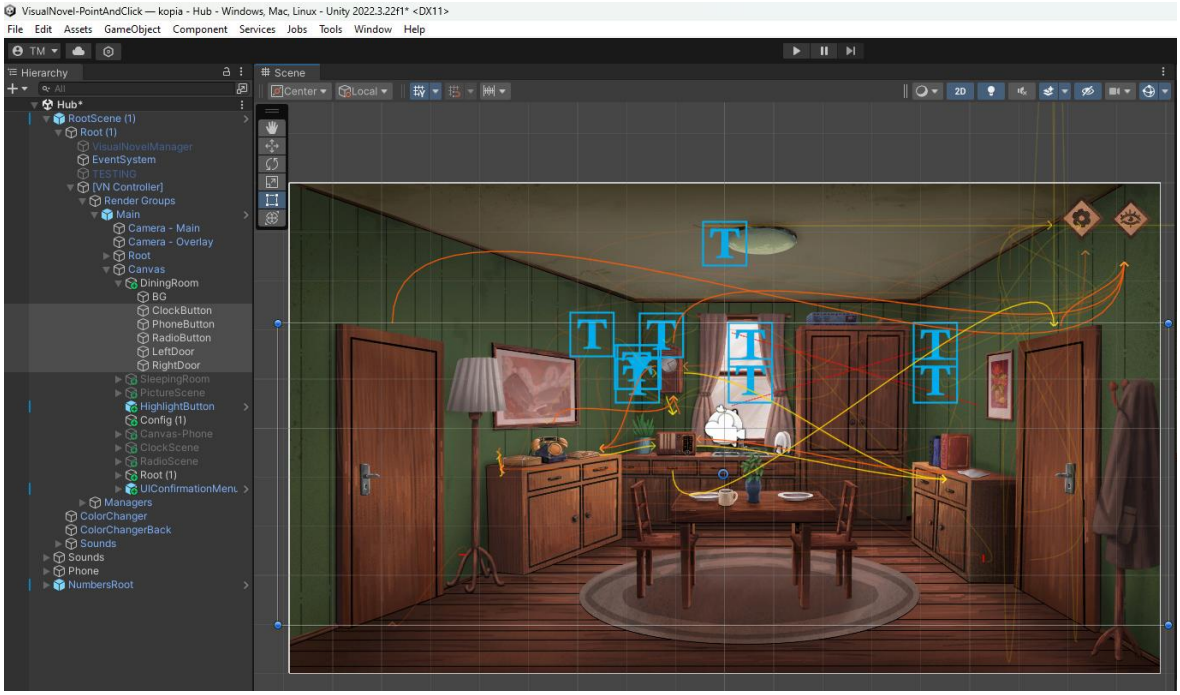
Poniższy fragment kodu ilustruje logikę obsługi telefonu. W momencie kolizji wyzwalana jest reakcja: odtworzenie dźwięku, zarejestrowanie wybranej cyfry, a następnie sprawdzenie kombinacji zdefiniowanych w tablicy *numberCombinationMappings*. Jeśli kombinacja jest zgodna, wywoływana jest powiązana akcja, np. przejście do kolejnej sceny lub odtworzenie określonego dialogu.

```
Unity Message | Odwołania: 0
private void OnTriggerEnter2D(Collider2D collision)
{
    string tag = collision.gameObject.tag;

    if (tag != "Untagged")
    {
        Debug.Log(tag);
        PlaySelectionSound(); // Odtwórz dźwięk przy wyborze cyfry

        if (rotaryDial.isReturning == false)
        {
            StartCoroutine(rotaryDial.ReturnToStartPosition());
        }
        rotaryDial.isHeld = false;

        tagsCollected.Add(tag);

        string tagsString = string.Join(",", tagsCollected);
        Debug.Log("Aktualna kombinacja numerów: " + tagsString); // Log informujący o aktualnej kombinacji

        // Znajdujemy pasującą kombinację i wywołujemy powiązaną akcję
        foreach (var mapping in numberCombinationMappings)
        {
            if (mapping.numberCombination == tagsString)
            {
                Debug.Log("Znaleziono pasującą kombinację: " + tagsString);
                Debug.Log("Wywołanie powiązanej akcji"); // Log informujący o wywołaniu akcji

                // Wywołujemy akcję z opóźnieniem
                StartCoroutine(ExecuteActionWithDelay(mapping.associatedAction, 2f));
                break;
            }
        }

        // Dezaktywujemy odpowiedni obiekt
        int tagNumber;
        if (int.TryParse(tag, out tagNumber) && tagNumber >= 0 && tagNumber < objects.Count)
        {
            objects[tagNumber].SetActive(false);
        }
    }
}

1 odwołanie
private void PlaySelectionSound()
{
    if (selectionSound != null)
    {
        selectionSound.Play();
    }
    else
    {
        Debug.LogWarning("AudioSource 'selectionSound' nie jest przypisany.");
    }
}

Odwołania: 0
public void SetVolume(float volume)
{
    if (audioMixerGroup != null)
    {
        audioMixerGroup.audioMixer.SetFloat("SelectionSoundVolume", Mathf.Log10(volume) * 20); // Przeliczamy głośność na skalę decybeli
    }
}

Odwołania: 0
public void ResetTags()
{
    tagsCollected.Clear();
    Debug.Log("Tagi wyczyszczone");
}
```

Rysunek 4.9 Skrypt ogranicznik tarczy numerycznej

Poza scenami związanymi z telefonem w grze pojawia się również *hub* – centralne miejsce, do którego gracz wraca w kluczowych momentach rozgrywki. *Hub* pełni funkcję węzła łączącego różne elementy rozgrywki, zapewniając dostęp do interaktywnych przedmiotów oraz mini gier. W jego obrębie umieszczono kilka obiektów, takich jak radio, zegar z ukrytą min igrą w “piętnastkę” (w wariancie 8), telefon czy drzwi prowadzące do innych pomieszczeń i miejsc fabularnych. Podobnie jak w przypadku scen z telefonem, logika *hub’a* opiera się przede wszystkim na prostych interakcjach typu kliknij i aktywuj/dezaktywuj

obiekt



Rysunek 4.10 Scena hub'u z hierarchia i podświetlonymi interaktywnymi przyciskami. Projekt graficzny Jon Kušar

Radio w *hub'ie* ma zapewnić nie tylko atmosferę, ale również możliwość prostego eksperymentowania z dźwiękiem i stacjami. Zaprojektowano je z dwoma pokrętłami: jedno kontroluje głośność, a drugie umożliwia zmianę stacji. Wybrane stacje mogą zawierać muzykę z różnych krajów i stylów, co pozwala graczowi wczuć się w klimat lat 20. XX wieku. Mechanizm działania radia jest zbliżony do telefonu – interakcja za pomocą przycisków i eventów Unity – jednak w tym przypadku zastosowano przesuwany suwak (*CircleSlider*) reprezentujący skalę wyboru stacji. Podczas kręcenia pokrętłem suwaka zmienia się pozycja wskaźnika, a kod oblicza, którą stację powinien aktualnie odtworzyć. Poniżej przedstawiony fragment kodu ilustruje sposób zmiany stacji i dostosowywania głośności. Metody *OnSliderValueChanged i AdjustRadioVolume* reagują na wejście użytkownika i odpowiednio modyfikują parametry radia, np. aktualnie odtwarzaną stację lub poziom głośności w decybelach. Każda stacja to osobny kanał dźwiękowy, którego aktywność jest kontrolowana przez przypisane skrypty i *AudioMixery*.

```
Skrypt aparatu Unity | Odwołania: 0
public class Radio : MonoBehaviour
{
    public AudioMixer audioMixer;
    public List<RadioStation> radioStations;
    public int currentStation;
    public float waitBeforeOff = 15;
    public CircleSlider circleSlider;
    public RectTransform arrowImage;
    public float arrowStartPositionY;
    public float arrowEndPositionY;

    Unity Message | Odwołania: 0
    void Start()
    {
        circleSlider.OnValueChanged.AddListener(OnSliderValueChanged);
        PlayRadio();
    }

    Odwołania: 0
    public void AdjustRadioVolume(float sliderValue)
    {
        float volumeInDecibels = SliderValueToDecibels(sliderValue);
        audioMixer.SetFloat("Volume", volumeInDecibels);
    }

    1 odwołanie
    private float SliderValueToDecibels(float sliderValue)
    {
        // Interpolacja liniowa między -80 dB a 20 dB
        return Mathf.Lerp(-80f, 20f, sliderValue / 100f);
    }

    Odwołania: 2
    public void PlayRadio()
    {
        for (int i = 0; i < radioStations.Count; i++)
        {
            if (i != currentStation)
            {
                radioStations[i].audioSource.volume = 0;
                radioStations[i].waitTime = waitBeforeOff;
                radioStations[i].currentRadio = false;
            }
        }
        radioStations[currentStation].audioSource.volume = 1;
        radioStations[currentStation].stopRadio = false;
        radioStations[currentStation].currentRadio = true;
    }
    1 odwołanie
    public void OnSliderValueChanged(float value)
    {
        int stationCount = radioStations.Count;
        float segmentSize = 100f / stationCount;

        for (int i = 0; i < stationCount; i++)
        {
            if (value >= i * segmentSize && value < (i + 1) * segmentSize)
            {
                currentStation = i;
                break;
            }
        }
        PlayRadio();
        UpdateArrowPosition(value);
    }
    1 odwołanie
    void UpdateArrowPosition(float sliderValue)
    {
        float normalizedValue = sliderValue / 100f;
        float newYPosition = Mathf.Lerp(arrowStartPositionY, arrowEndPositionY, normalizedValue);

        arrowImage.anchoredPosition = new Vector2(arrowImage.anchoredPosition.x, newYPosition);
    }
```

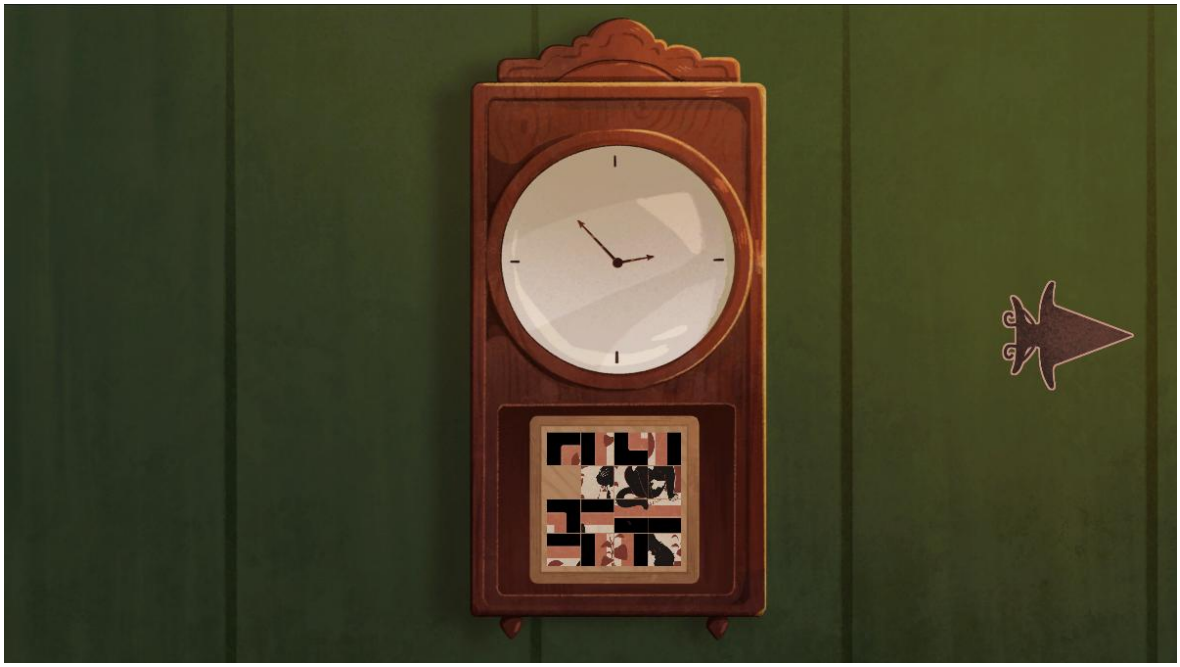
Rysunek 4.11 Skrypt odpowiadający za część logiki radia

Zegar w *hub'ie* pełni podwójną rolę: z jednej strony wskazuje aktualny czas systemowy gracza, a z drugiej skrywa mini grę logiczną. Mechanizm wskazówek zegara odwołuje się do czasu systemowego, a ich pozycja jest aktualizowana w każdej klatce. Dzięki temu gracz może odczuć immersję związaną z rzeczywistym upływem czasu, co wzmacnia wrażenie autentyczności świata gry.

```
using UnityEngine;

Skrypt aparatu Unity | Odwołania: 0
public class Clock : MonoBehaviour
{

    public Transform hourHand;
    public Transform minuteHand;

    Unity Message | Odwołania: 0
    void Update()
    {
        // Pobierz aktualny czas z systemu
        System.DateTime currentTime = System.DateTime.Now;

        // Oblicz kąt obrotu dla wskazówki godzinowej
        float hourRotation = (currentTime.Hour % 12) * 30f + currentTime.Minute * 0.5f; // 30 stopni na każdą godzinę (360/12) + minuta ma wpływ na pozycję godzinową
        hourHand.localRotation = Quaternion.Euler(0, 0, -hourRotation);

        // Oblicz kąt obrotu dla wskazówki minutowej
        float minuteRotation = currentTime.Minute * 6f; // 6 stopni na każdą minutę (360/60)
        minuteHand.localRotation = Quaternion.Euler(0, 0, -minuteRotation);
    }
}
```

Rysunek 4.12 Skrypt zegara

Pod tarczą zegara ukryta jest mini gra w „piętnastkę” (w wariancie 8), która jest opcjonalną łamigłówką stanowiącą dodatkowe wyzwanie dla gracza. Wybierając interakcję z zegarem, gracz może przejść do sceny z łamigłówką. Rozwiązanie puzzli może nagrodzić gracza dodatkowymi informacjami lub dostępem do nowych elementów fabuły. Obsługa tej mini gry została zaimplementowana w osobnym skrypcie, bazującym na podobnej logice co opisane wcześniej mechaniki – aktywowane i dezaktywowane obiekty, zarządzanie stanem gry oraz reagowanie na ruchy gracza.



Rysunek 4.13 Scena przedstawiająca zegar z odsłonięta tarcza do gry w piętnastkę

W *hub’ie* znajdują się również drzwi prowadzące do innych pomieszczeń lub obszarów fabularnych. Podobnie jak w przypadku innych elementów, interakcja z drzwiami odbywa się przez kliknięcie przycisku. Zależnie od stanu fabularnego mogą one prowadzić do

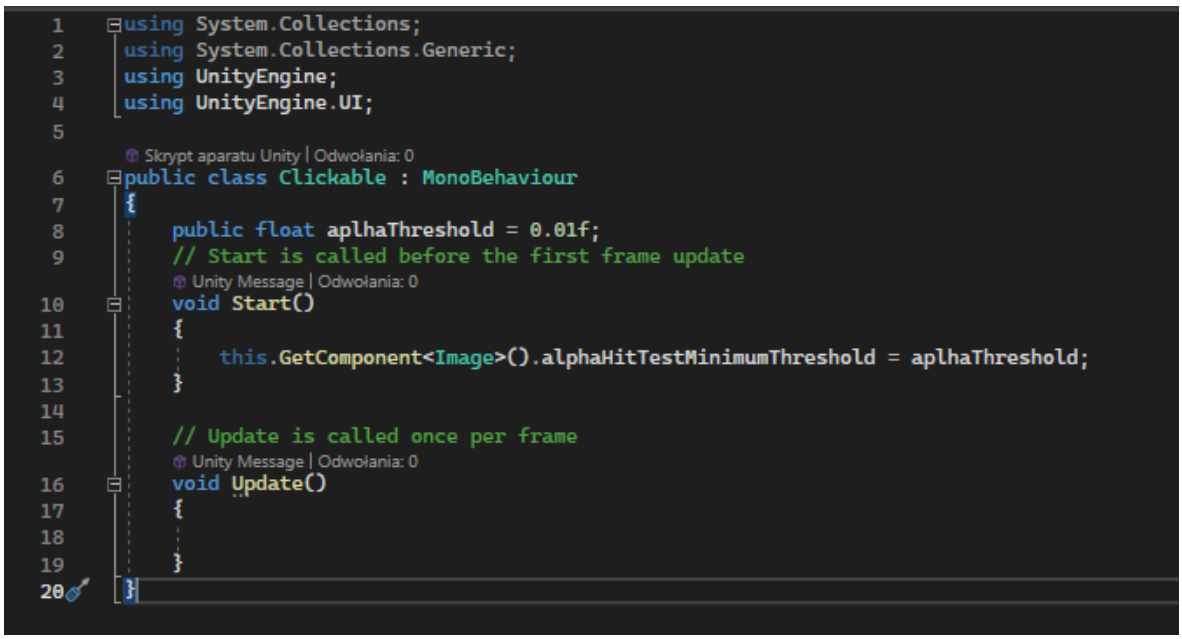
nowych scen, uruchamiać dialogi lub aktywować dodatkowe elementy gry. Ta elastyczność pozwala dynamicznie zmieniać funkcję *hub'a* w zależności od postępów gracza.

Implementacja *hub'a* sprowadza się do integracji wielu składowych: radia, telefonu, zegara z mini grą, a także drzwi i innych interaktywnych obiektów. Każdy element został zaimplementowany jako oddzielny zestaw skryptów i obiektów Unity, a następnie powiązany za pomocą prostych eventów i kliknięć. W ten sposób *hub* działa jako centralny punkt, w którym gracze mogą odpocząć, eksplorować dodatkowe funkcje gry oraz przejść do kolejnych etapów przygody.

4.3. Implementacja scen typu point-and-click.

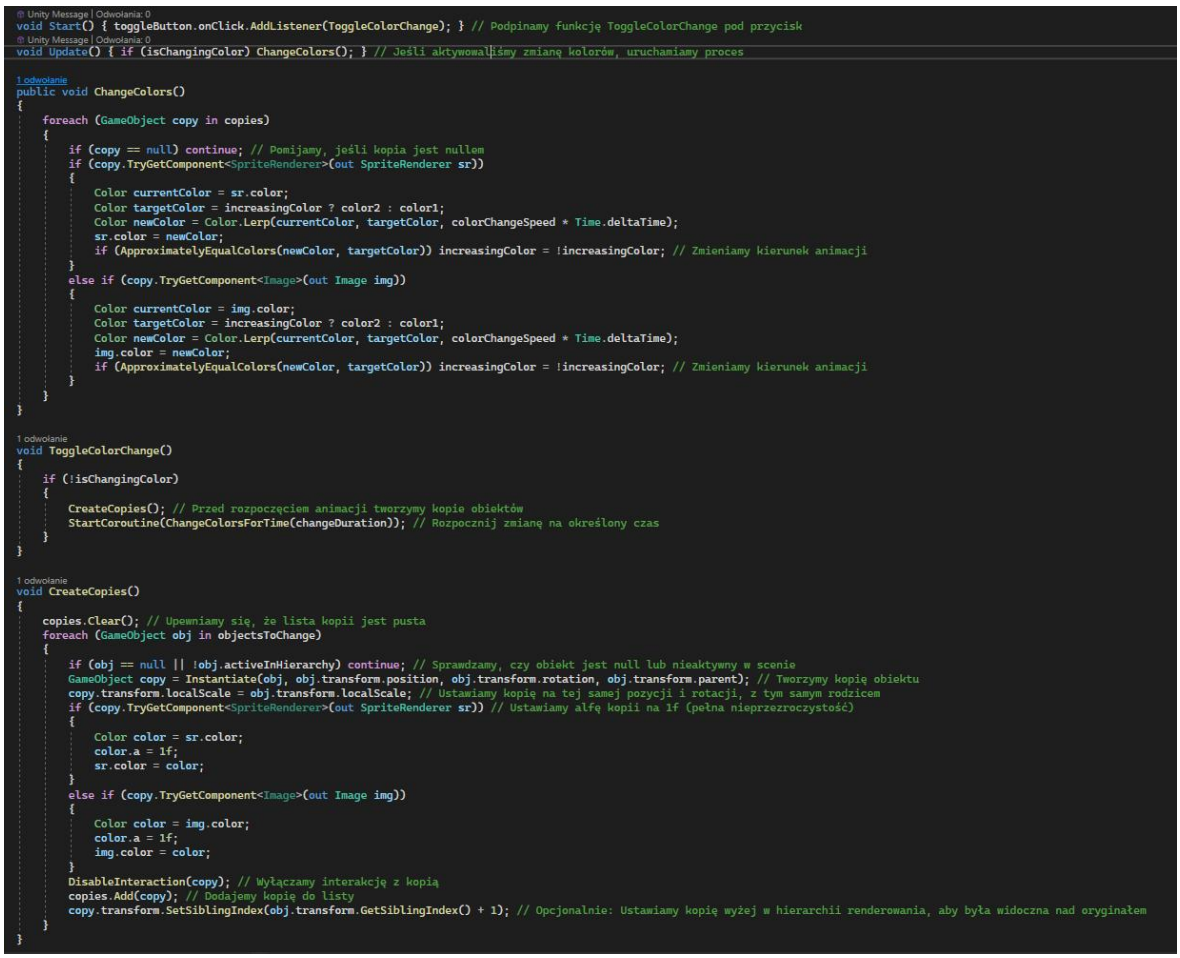
W grze występują dwie sceny *point-and-click*, których celem jest umożliwienie graczowi eksploracji otoczenia, zbierania informacji i interakcji z przedmiotami prowadzącymi do dalszego rozwoju fabuły. Choć każda z tych scen prezentuje inny kontekst i zawartość, opierają się one na zbliżonej logice: wykorzystują obiekty typu *Button* w Unity oraz komponenty umożliwiające interakcję z grafiką przedstawioną w formie *sprite'ów*. Dzięki temu gracz może klikać bezpośrednio na elementy graficzne, a nie jedynie w obrębie prostokątnych obszarów przycisków, co zwiększa precyzję i naturalność interakcji.

Aby osiągnąć możliwość klikania po powierzchni *sprite'a* zamiast standardowego obszaru przycisku, zastosowano komponent Image z ustawionym progiem przezroczystości a*lphaHitTestMinimumThreshold*. Wykorzystując ten mechanizm, kliknięcia rejestrowane są wyłącznie na widocznej części *sprite'a*, pomijając przezroczyste fragmenty. Poniższy kod prezentuje prostą klasę *Clickable*, która ustawia minimalny próg przezroczystości, poniżej którego kliknięcia nie są wykrywane:



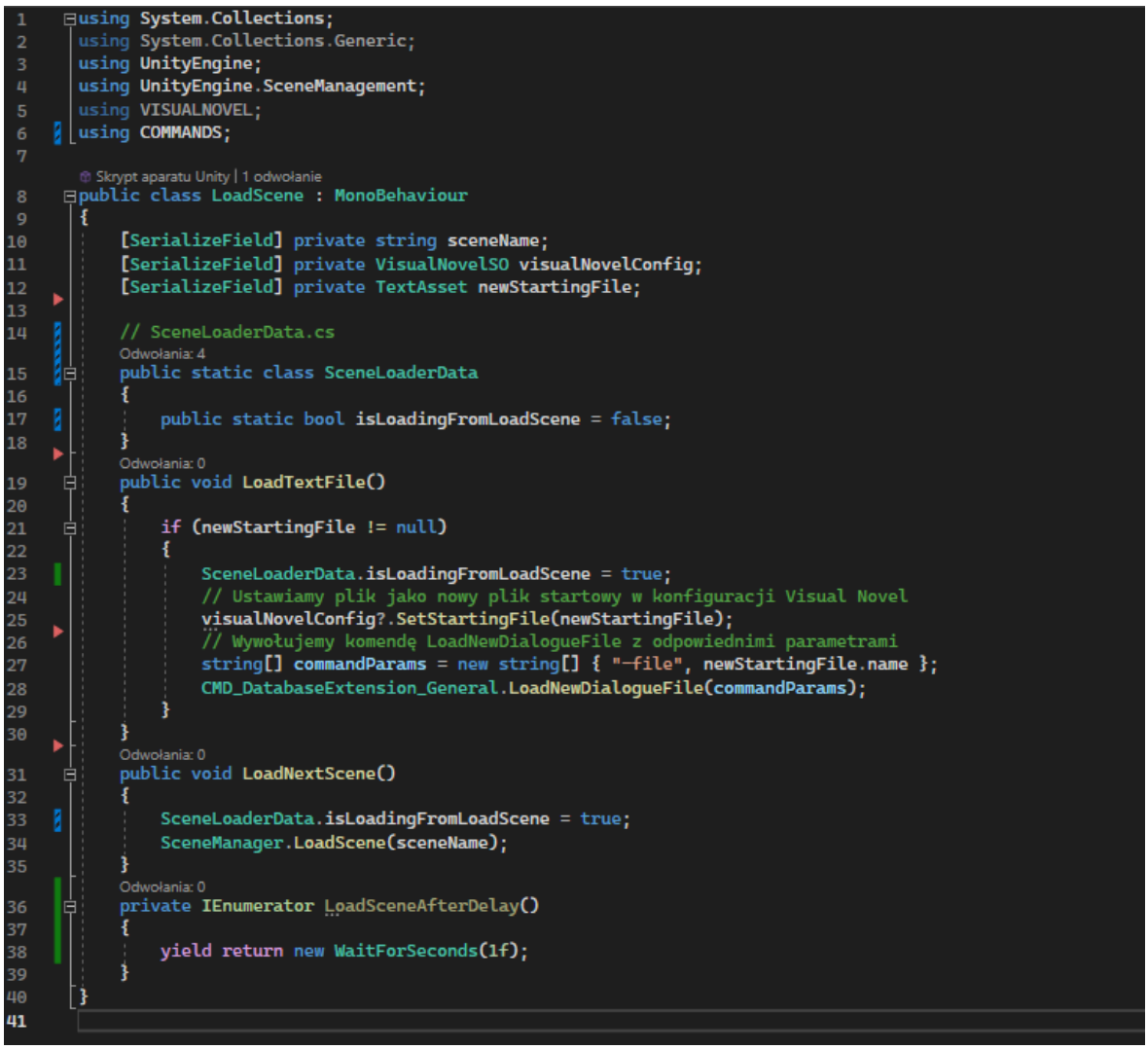
Rysunek 4.14 Skrypt odpowiadający za logikę przezroczystych przycisków

W scenach *point-and-click* ważną rolę pełni ikona „oka”, która pomaga graczowi odnaleźć istotne elementy, gdy nie jest pewien, co zrobić dalej. Naciśnięcie przycisku z ikoną aktywuje funkcję tworzenia kopii interaktywnych obiektów z pełną przezroczystością i pulsującym efektem zmiany koloru. Efekt pulsowania realizowany jest poprzez płynną interpolację wartości kolorów w czasie, tworząc wrażenie rozjaśniania i przyciemniania elementów. Kopie te wyświetlają się nad oryginalnymi obiektami, ale są pozbawione interaktywności, aby uniknąć niepożądanych błędów i wprowadzania gracza w błąd. Po upływie określonego czasu kopie zostają usunięte.



Rysunek 4.15 Skrypt podświetlania obiektów

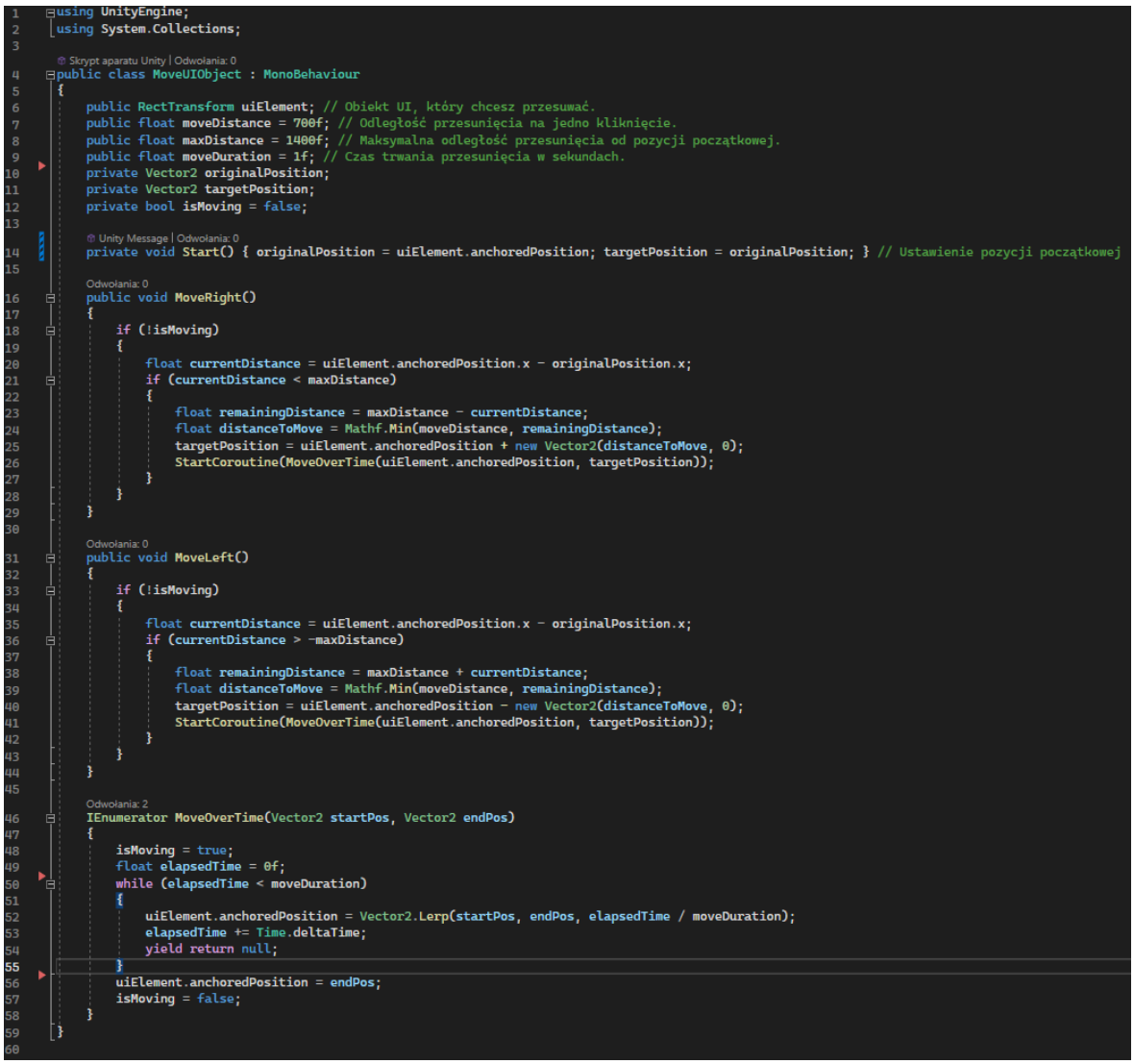
Kopie obiektów mają zwiększoną wartość alfa, co czyni je w pełni widocznymi, a ich kolor jest płynnie zmieniany, aby stworzyć efekt pulsowania. Poniższy kod przedstawia fragment skryptu odpowiedzialnego za wczytanie nowego pliku tekstowego ze scenariuszem i zmianę sceny. Po wywołaniu metody *LoadTextFile()* system *visual novel* ładuje określony plik, a *LoadNextScene()* przenosi gracza do pożądanej sceny. Zastosowano ten kod w przyciskach na scenach, które mają przenieść gracz na scenę z silnikiem, gdzie jest możliwość prowadzenia narracji.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using VISUALNOVEL;
using COMMANDS;

Skrypt aparatu Unity | 1 odwołanie
public class LoadScene : MonoBehaviour
{
    [SerializeField] private string sceneName;
    [SerializeField] private VisualNovelSO visualNovelConfig;
    [SerializeField] private TextAsset newStartingFile;

    // SceneLoaderData.cs
    Odwołania: 4
    public static class SceneLoaderData
    {
        public static bool isLoadingFromLoadScene = false;
    }
    Odwołania: 0
    public void LoadTextFile()
    {
        if (newStartingFile != null)
        {
            SceneLoaderData.isLoadingFromLoadScene = true;
            // Ustawiamy plik jako nowy plik startowy w konfiguracji Visual Novel
            visualNovelConfig?.SetStartingFile(newStartingFile);
            // Wywołujemy komendę LoadNewDialogueFile z odpowiednimi parametrami
            string[] commandParams = new string[] { "-file", newStartingFile.name };
            CMD_DatabaseExtension_General.LoadNewDialogueFile(commandParams);
        }
    }
    Odwołania: 0
    public void LoadNextScene()
    {
        SceneLoaderData.isLoadingFromLoadScene = true;
        SceneManager.LoadScene(sceneName);
    }
    Odwołania: 0
    private IEnumerator LoadSceneAfterDelay()
    {
        yield return new WaitForSeconds(1f);
    }
}
```

Rysunek 4.16 Skrypt odpowiadający za wczytanie plików tekstowych oraz wybranej sceny

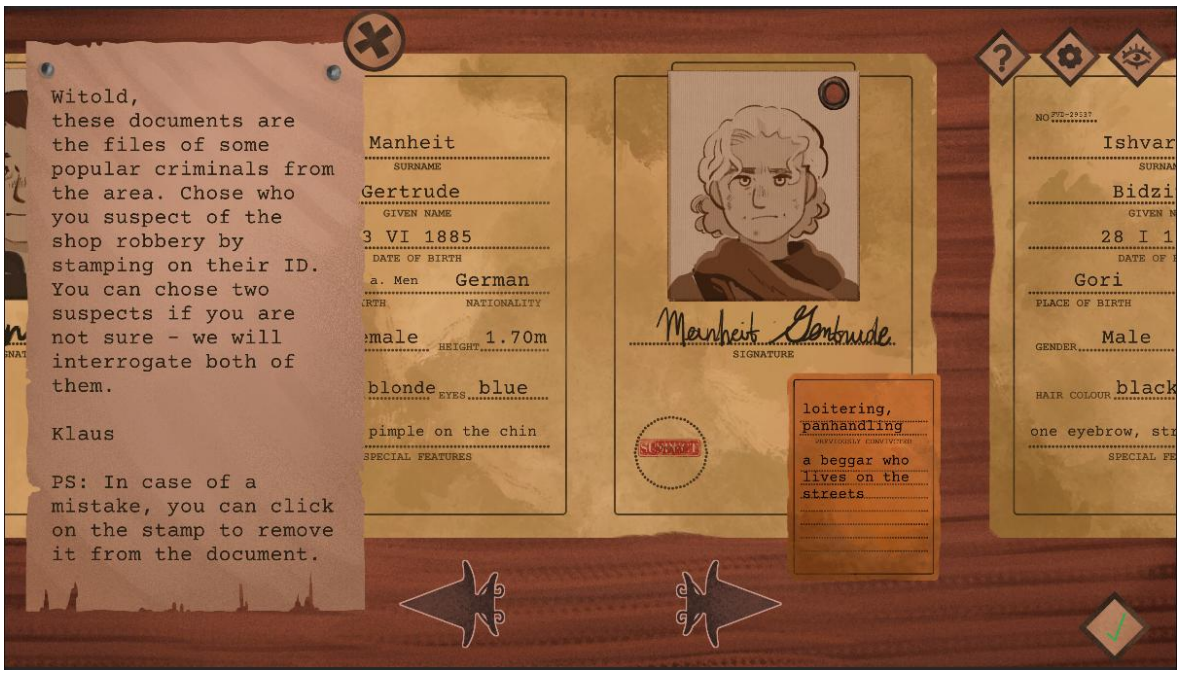
Scena z wyborem podejrzanych na komisariacie jest kolejną mini grą, którą gracz musi rozwiązać, korzystając z przedstawionych wcześniej mechanik. Podobnie jak w innych przypadkach, jej logika opiera się na interaktywnych elementach UI, lecz tym razem kluczową rolę odgrywa zarządzanie pozycją wyświetlanych dokumentów oraz wyborem określonych podejrzanych. Na scenie znajduje się zestaw 9 dokumentów dotyczących potencjalnych sprawców kradzieży sklepowej. Dokumenty te można przesuwać w lewo i w prawo za pomocą przycisków w interfejsie. Każde kliknięcie przesuwa widok o zdefiniowaną wartość, dzięki czemu gracz może przeglądać całą listę podejrzanych. W implementacji zastosowano kod, który sprawdza aktualną pozycję obiektu i uniemożliwia przesuwanie go w nieskończoność. W rezultacie widok przesuwa się jedynie w wyznaczonych granicach, co poprawia ergonomię i czytelność mechaniki.



```
using UnityEngine;
using System.Collections;

Skrypt aparatu Unity | Odwołania: 0
public class MoveUIObject : MonoBehaviour
{
    public RectTransform uiElement; // Obiekt UI, który chcesz przesuwać.
    public float moveDistance = 700f; // Odległość przesunięcia na jedno kliknięcie.
    public float maxDistance = 1400f; // Maksymalna odległość przesunięcia od pozycji początkowej.
    public float moveDuration = 1f; // Czas trwania przesunięcia w sekundach.
    private Vector2 originalPosition;
    private Vector2 targetPosition;
    private bool isMoving = false;

    Unity Message | Odwołania: 0
    private void Start() { originalPosition = uiElement.anchoredPosition; targetPosition = originalPosition; } // Ustawienie pozycji początkowej

    Odwołania: 0
    public void MoveRight()
    {
        if (!isMoving)
        {
            float currentDistance = uiElement.anchoredPosition.x - originalPosition.x;
            if (currentDistance < maxDistance)
            {
                float remainingDistance = maxDistance - currentDistance;
                float distanceToMove = Mathf.Min(moveDistance, remainingDistance);
                targetPosition = uiElement.anchoredPosition + new Vector2(distanceToMove, 0);
                StartCoroutine(MoveOverTime(uiElement.anchoredPosition, targetPosition));
            }
        }
    }

    Odwołania: 0
    public void MoveLeft()
    {
        if (!isMoving)
        {
            float currentDistance = uiElement.anchoredPosition.x - originalPosition.x;
            if (currentDistance > -maxDistance)
            {
                float remainingDistance = maxDistance + currentDistance;
                float distanceToMove = Mathf.Min(moveDistance, remainingDistance);
                targetPosition = uiElement.anchoredPosition - new Vector2(distanceToMove, 0);
                StartCoroutine(MoveOverTime(uiElement.anchoredPosition, targetPosition));
            }
        }
    }

    Odwołania: 2
    IEnumerator MoveOverTime(Vector2 startPos, Vector2 endPos)
    {
        isMoving = true;
        float elapsedTime = 0f;
        while (elapsedTime < moveDuration)
        {
            uiElement.anchoredPosition = Vector2.Lerp(startPos, endPos, elapsedTime / moveDuration);
            elapsedTime += Time.deltaTime;
            yield return null;
        }
        uiElement.anchoredPosition = endPos;
        isMoving = false;
    }
}
```

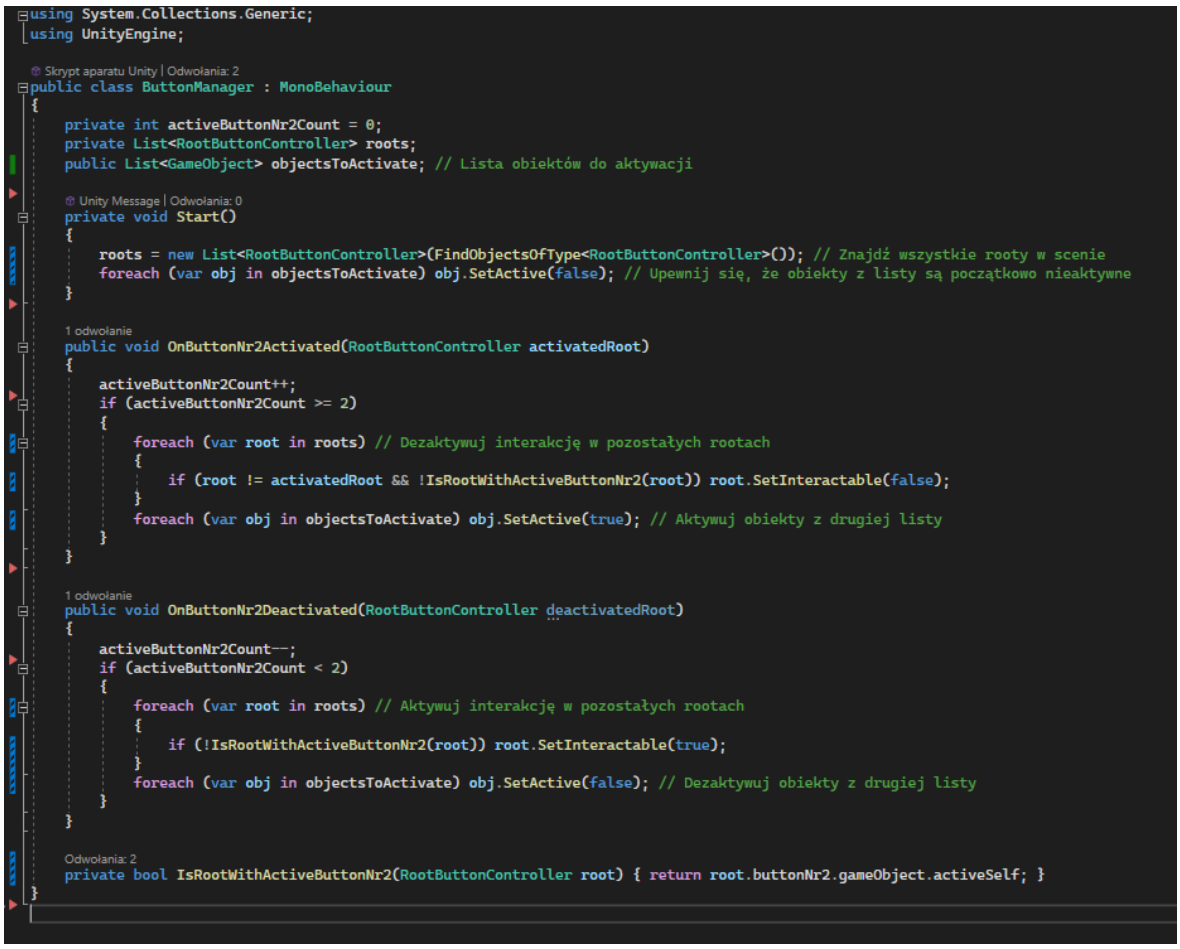
Rysunek 4.17 Skrypt z logiką przesuwania obiektów

Przykładowy kod wykorzystuje korutyny do płynnego przesuwania interfejsu oraz limity dystansu *maxDistance*, aby kontrolować zakres ruchu. Dzięki temu gracze mogą swobodnie przeglądać listę podejrzanych, a jednocześnie nigdy nie "wyjadą" poza ramy sceny. Podobnie jak w innych scenach, gracz ma dostęp do ikonki z pomocą – znak zapytania, który po kliknięciu aktywuje dodatkową stronę z wyjaśnieniem zasad mini gry. Pomaga to w przypadku, gdy gracz poczuje się zdezorientowany lub zapomni, w jaki sposób należy dokonać wyboru podejrzanych. To kolejny przykład wykorzystania prostych *event'ów* w Unity: kliknięcie przycisku aktywuje obiekt z informacjami i wyświetla go na ekranie.



Rysunek 4.18 Scena z wybieranymi dwoma podejrzanymi oraz aktywnym przyciskiem do progresji

Najważniejszym elementem tej mini gry jest mechanizm wyboru podejrzanych. Gracz może wytypować maksymalnie dwie osoby. W tym celu każdy podejrzany ma przypisany specjalny przycisk w interfejsie, który po kliknięciu oznacza daną osobę jako wybraną. Aby kontrolować tę logikę, zastosowano menedżer przycisków *ButtonManager*, który nadzoruje stan aktywowanych opcji i odpowiednio reaguje. Jeśli gracz wybierze jednego podejrzanego, nic szczególnego się nie dzieje – można nadal wybierać spośród pozostałych. W momencie, gdy gracz zaznaczy drugiego podejrzanego, menedżer przycisków sprawdza, czy dwa różne przyciski są aktywne. Jeśli tak, dezaktywuje interakcję z pozostałymi przyciskami, uniemożliwiając wybór trzeciej osoby. Jednocześnie aktywuje specjalne obiekty UI (np. przycisk potwierdzenia wyboru), które pozwalają graczowi przejść dalej z fabułą. Poniższy fragment kodu przedstawia logikę menedżera przycisków. Korzysta on z listy *RootButtonController* – skryptów przypisanych do przycisków podejrzanych – i sprawdza ich stan. Gdy dwa różne przyciski są aktywne, menedżer aktywuje określone obiekty i wyłącza możliwość interakcji z pozostałymi, zabezpieczając mini grę przed błędami i niejasnościami w rozgrywce.



```
using System.Collections.Generic;
using UnityEngine;

Skrypt aparatu Unity | Odwołania: 2
public class ButtonManager : MonoBehaviour
{
    private int activeButtonNr2Count = 0;
    private List<RootButtonController> roots;
    public List<GameObject> objectsToActivate; // Lista obiektów do aktywacji

    Unity Message | Odwołania: 0
    private void Start()
    {
        roots = new List<RootButtonController>(FindObjectsOfType<RootButtonController>()); // Znajdź wszystkie rooty w scenie
        foreach (var obj in objectsToActivate) obj.SetActive(false); // Upewnij się, że obiekty z listy są początkowo nieaktywne
    }

    1 odwołanie
    public void OnButtonNr2Activated(RootButtonController activatedRoot)
    {
        activeButtonNr2Count++;
        if (activeButtonNr2Count >= 2)
        {
            foreach (var root in roots) // Dezaktywuj interakcję w pozostałych rootach
            {
                if (root != activatedRoot && !IsRootWithActiveButtonNr2(root)) root.SetInteractable(false);
            }
            foreach (var obj in objectsToActivate) obj.SetActive(true); // Aktywuj obiekty z drugiej listy
        }
    }

    1 odwołanie
    public void OnButtonNr2Deactivated(RootButtonController deactivatedRoot)
    {
        activeButtonNr2Count--;
        if (activeButtonNr2Count < 2)
        {
            foreach (var root in roots) // Aktywuj interakcję w pozostałych rootach
            {
                if (!IsRootWithActiveButtonNr2(root)) root.SetInteractable(true);
            }
            foreach (var obj in objectsToActivate) obj.SetActive(false); // Dezaktywuj obiekty z drugiej listy
        }
    }

    Odwołania: 2
    private bool IsRootWithActiveButtonNr2(RootButtonController root) { return root.buttonNr2.gameObject.activeSelf; }
}
```

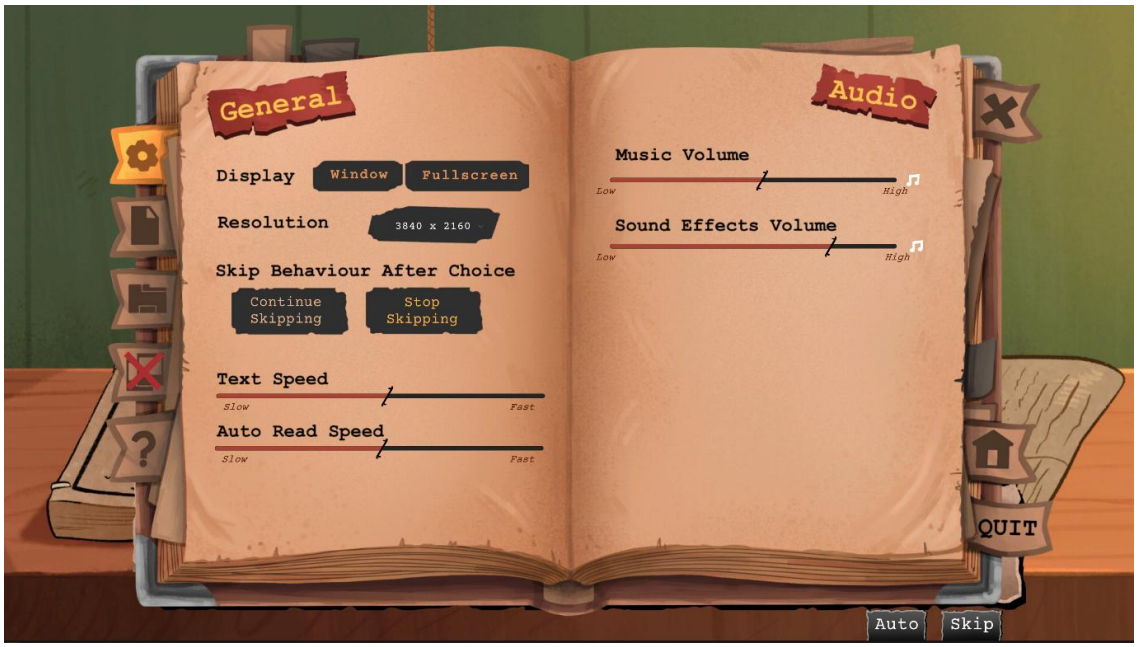
Rysunek 4.19 Skrypt menedżera przycisków

Dzięki temu zestawowi mechanik, gracz może w uporządkowany sposób dokonać wyboru dwóch podejrzanych, którzy jego zdaniem pasują do zgromadzonych dowodów. Po zatwierdzeniu wyboru gra może przejść do kolejnej sekwencji fabularnej, nagradzając gracza za poprawną dedukcję lub konsekwentnie rozwijając narrację, jeśli wybór okaże się nieprecyzyjny. W efekcie scena z dokumentami na komisariacie stanowi istotne połączenie między aspektami narracyjnymi a interaktywnymi, pozwalając graczowi realnie wpłynąć na postęp w dochodzeniu.

4.4. Logika interfejsu użytkownika

W trakcie rozwoju gry kluczowym elementem stała się implementacja logicznego i intuicyjnego interfejsu użytkownika. System menu, okien konfiguracyjnych, stron zapisu/wczytywania oraz pomocy musiał zostać zintegrowany z główną rozgrywką, zapewniając graczowi łatwy dostęp do najważniejszych opcji i narzędzi bez konieczności wychodzenia z gry. Dzięki temu możliwe było zagwarantowanie płynnego doświadczenia odbiorcy – od

wprowadzania zmian w ustawieniach graficznych czy audio, po przeglądanie pomocy i powracanie do menu głównego.



Rysunek 4.20 Panel ustawień w grze. Projekt graficzny Jon Kušar

W interfejsie gracze mogą zmieniać podstawowe ustawienia gry, takie jak rozdzielczość ekranu, tryb wyświetlania (okno lub pełny ekran), głośność muzyki i efektów dźwiękowych. Dodatkowo przewidziano opcje wpływające na sposób czytania tekstu: szybkość pojawiania się dialogów, opcje automatycznego pomijania tekstu czy decyzje dotyczące tego, czy tekst powinien wstrzymać funkcję pomijania po dokonaniu wyboru w grze. Dostęp do funkcji zapisu i wczytywania stanu gry zapewnia osobna zakładka, natomiast zakładka pomocy zawiera informacje o sterowaniu oraz mechanikach rozgrywki, tak aby gracz mógł w każdej chwili odświeżyć sobie najważniejsze aspekty działania gry.

Rysunek 4.21 Menu w grze zakładka zapisu postępu. Opracowanie Jon Kušar

Do zarządzania logiką wyświetlania i przełączania stron menu służy dedykowany menedżer menu *VNMenuManager*. Jego celem jest obsługa przejść pomiędzy poszczególnymi ekranami – np. z głównego menu do menu zapisu/wczytywania czy do zakładki pomocy – oraz odpowiednia aktywacja i dezaktywacja określonych obiektów interfejsu. Poniższy, skrócony fragment kodu prezentuje najważniejsze funkcje tego menedżera:

```
Skrypt aparatu Unity | Odwołania: 2
public class VNMenuManager : MonoBehaviour
{
    public static VNMenuManager instance; private MenuPage activePage = null; private bool isOpen = false;
    [SerializeField] private CanvasGroup root; [SerializeField] private MenuPage[] pages; private CanvasGroupController rootCG;
    Odwołania: 3
    private UIConfirmationMenu uiChoiceMenu => UIConfirmationMenu.instance; // Listy do aktywacji/dezaktywacji obiektów
    [SerializeField] private List<GameObject> objectsToActivateOnLoadPage; [SerializeField] private List<GameObject> objectsToDeactivateOnLoadPage;
    [SerializeField] private List<GameObject> objectsToActivateOnSavePage; [SerializeField] private List<GameObject> objectsToDeactivateOnSavePage;
    [SerializeField] private List<GameObject> objectsToActivateOnGalleryPage; [SerializeField] private List<GameObject> objectsToDeactivateOnGalleryPage;

    Unity Message | Odwołania: 0
    private void Awake() { instance = this; } // Inicjalizacja instancji

    Unity Message | Odwołania: 0
    void Start() { rootCG = new CanvasGroupController(this, root); } // Ustawienie kontrolera CanvasGroup

    Odwołania: 0
    public void Click_Home()
    {
        uiChoiceMenu.Show("Return to the main menu?", new UIConfirmationMenu.ConfirmationButton("Yes", () =>
        {
            VN_Configuration.activeConfig.Save(); UnityEngine.SceneManagement.SceneManager.LoadScene(MainMenu.MAIN_MENU_SCENE);
        }), new UIConfirmationMenu.ConfirmationButton("No", null));
    }
    Odwołania: 5
    private MenuPage GetPage(MenuPage.PageType pageType) { return pages.FirstOrDefault(page => page.pageType == pageType); } // Znalezienie strony na podstawie typu

    Odwołania: 0
    public void OpenSavePage()
    {
        var page = GetPage(MenuPage.PageType.SaveAndLoad); var slm = page.anim.GetComponentInParent<SaveAndLoadMenu>();
        slm.menuFunction = SaveAndLoadMenu.MenuFunction.save; OpenPage(page); ActivateObjects(objectsToActivateOnSavePage); DeactivateObjects(objectsToDeactivateOnSavePage);
    }

    Odwołania: 0
    public void OpenLoadPage()
    {
        var page = GetPage(MenuPage.PageType.SaveAndLoad); var slm = page.anim.GetComponentInParent<SaveAndLoadMenu>();
        slm.menuFunction = SaveAndLoadMenu.MenuFunction.load; OpenPage(page); ActivateObjects(objectsToActivateOnLoadPage); DeactivateObjects(objectsToDeactivateOnLoadPage);
    }

    Odwołania: 0
    public void OpenConfigPage() { var page = GetPage(MenuPage.PageType.Config); OpenPage(page); }

    Odwołania: 0
    public void OpenHelpPage() { var page = GetPage(MenuPage.PageType.Help); OpenPage(page); }

    Odwołania: 0
    public void OpenGalleryPage()
    {
        var page = GetPage(MenuPage.PageType.Gallery); OpenPage(page);
        ActivateObjects(objectsToActivateOnGalleryPage); DeactivateObjects(objectsToDeactivateOnGalleryPage);
    }

    Odwołania: 5
    private void OpenPage(MenuPage page)
    {
        if (page == null) return;
        if (activePage != null && activePage != page) activePage.Close();
        page.Open(); activePage = page;
        if (!isOpen) OpenRoot();
    }
```

Rysunek 4.22 Skrypt zarządzania menu

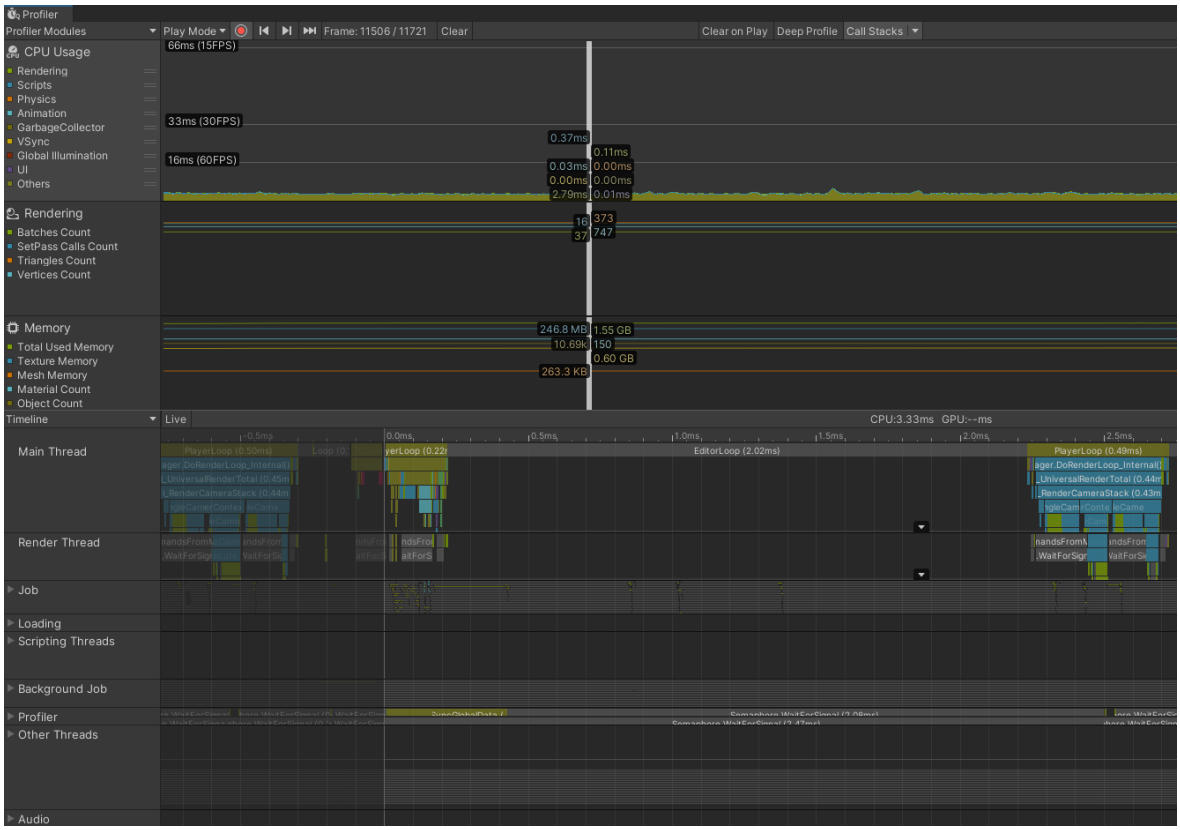
Wykorzystanie *CanvasGroup* i *MenuPage* pozwala na płynne animowanie i przełączanie ekranów, a poprzez wywoływanie odpowiednich metod, na przykład *OpenSavePage(), OpenLoadPage()* menedżer może dostosować interfejs do aktualnie wybranej funkcji. Dodanie narzędzi typu *UIConfirmationMenu* umożliwia wyświetlanie komunikatów potwierdzających ważne decyzje, takie jak powrót do menu głównego czy nadpisanie istniejących zapisów stanu gry.

Logika interfejsu i zarządzania menu jest istotnym aspektem implementacji gry. Pozwala ona na elastyczne reagowanie na potrzeby gracza, oferowanie rozbudowanych opcji konfiguracyjnych oraz łatwe przemieszczanie się pomiędzy różnymi ekranami i funkcjami. Dzięki przemyślanej architekturze kodu i prostym eventom opartym na Unity, możliwa jest łatwa rozbudowa interfejsu i dodawanie kolejnych elementów w przyszłości

4.5. Testy i optymalizacja

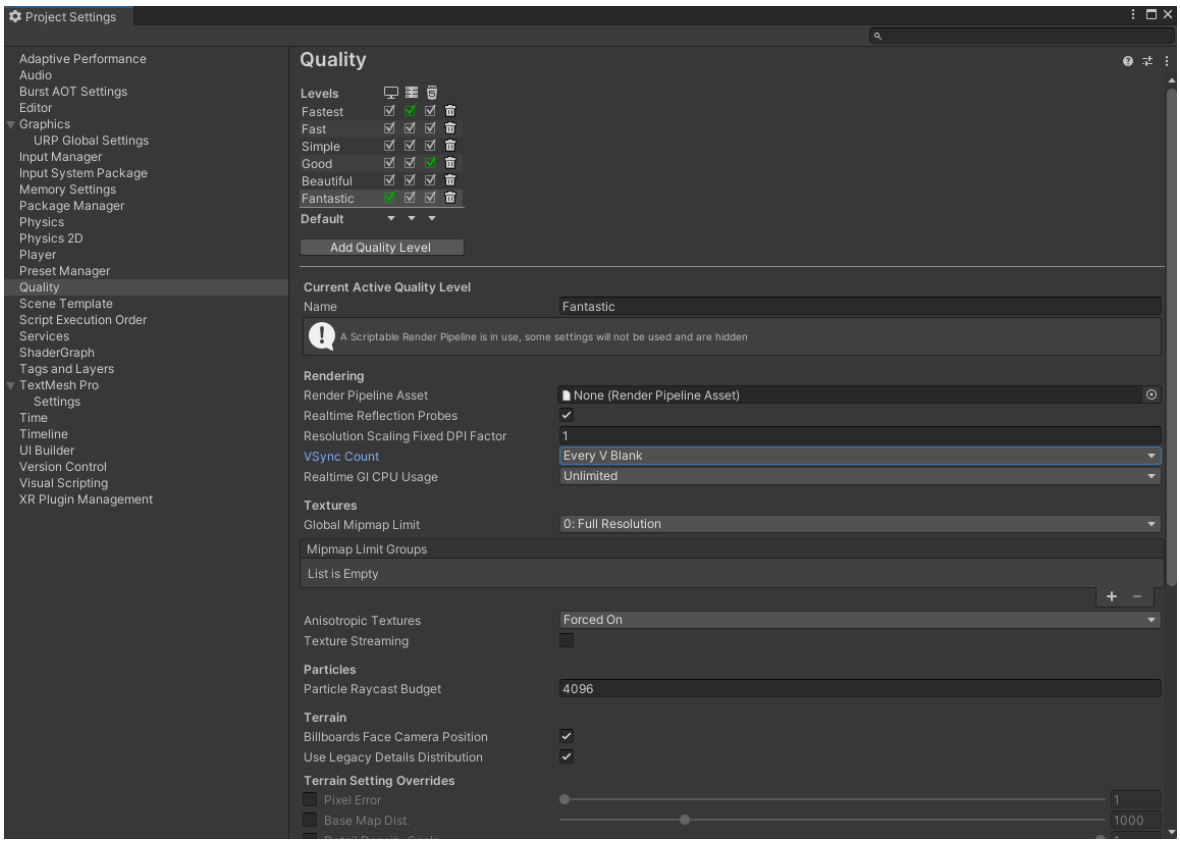
Proces testowania i optymalizacji stanowił kluczowy etap projektu, mający na celu zapewnienie stabilności oraz płynności działania gry na różnych konfiguracjach

sprzętowych. Gra, jako produkt typu *visual novel* z elementami *point-and-click* oraz mini-grami, nie wymagała bardzo dużej mocy obliczeniowej, co umożliwiło jej płynne działanie nawet na starszych laptopach z zintegrowanymi kartami graficznymi. Na przykład, testy przeprowadzone na komputerze wyposażonym w kartę 4070 Ti oraz procesor i5-13600KF wykazały, że gra renderuje znacznie powyżej 400 fps; jednak w ostatecznym buildzie zastosowano synchronizację pionową *v-sync*, która automatycznie ogranicza liczbę klatek do częstotliwości odświeżania monitora. Takie rozwiązanie pozwala zoptymalizować wykorzystanie zasobów, zapobiegać rozrywaniu animacji i utrzymać stabilną, płynną rozgrywkę. Testy wykonane na laptopie z procesorem AMD Ryzen 3 5425U z zintegrowaną kartą graficzną potwierdziły, że poza krótkimi okresami ładowania nowych scen, liczba klatek nie spada poniżej 60 fps.



Rysunek 4.23 profil wydajności w Unity Profiler

Istotnym elementem testów była weryfikacja poprawności ustawienia numerów telefonicznych we wszystkich scenach, w których wywoływana jest rozmowa telefoniczna. Poprawne przypisanie scen końcowych miało kluczowe znaczenie – błędne ustawienie komend mogło prowadzić do niezamierzonej teleportacji między scenami na przykład



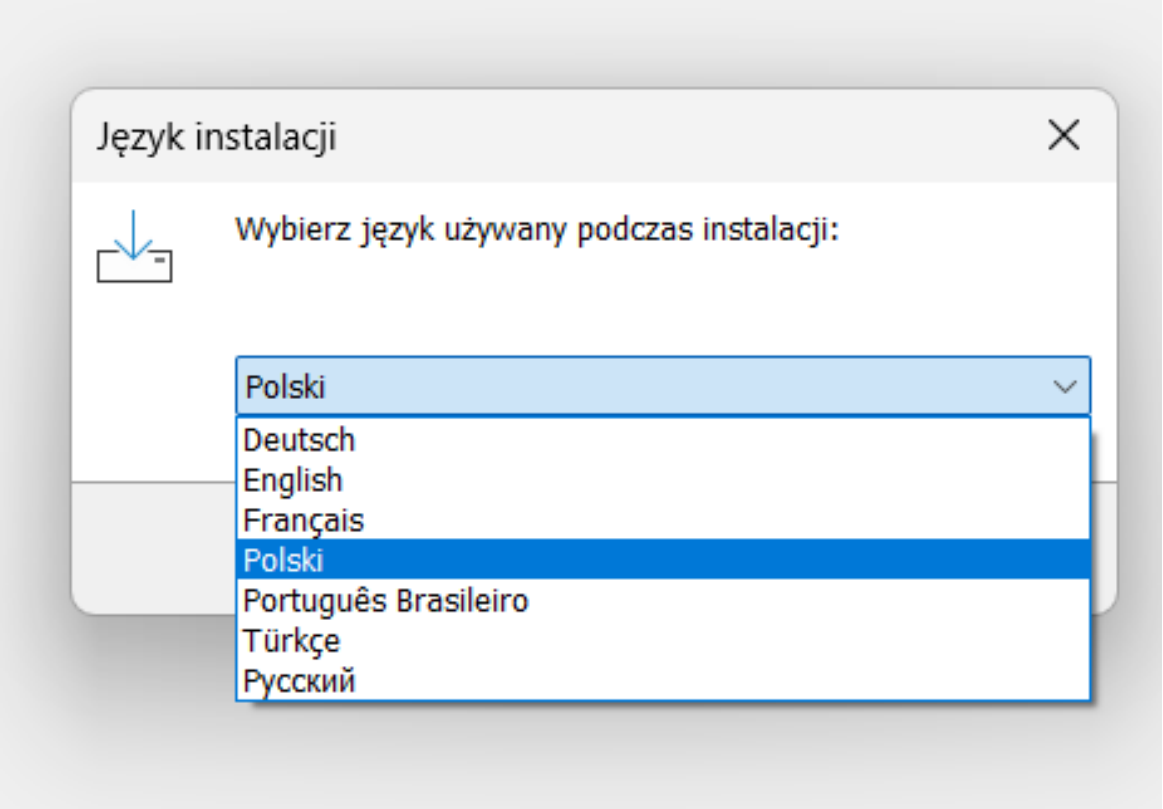
Rysunek 4.24 Ustawienia Vsync

przenoszenia gracza z dnia 3 do dnia 2, co zostało wykryte i naprawione podczas testów. Dodatkowo, podczas testów stwierdzono, że interfejs użytkownika nie skalował się poprawnie na rozdzielczościach poniżej Full HD. Problem ten został rozwiązany poprzez modyfikację ustawień *Canvas* w Unity, co umożliwiło prawidłowe wyświetlanie elementów UI na różnych ekranach. Kolejnym wykrytym błędem było nieprawidłowe zachowanie tarczy numerycznej – obracała się ona wielokrotnie, zanim powróciła do pozycji startowej. Błąd ten, wynikający z niedokładnego wycentrowania *sprite'a* tarczy, został usunięty dzięki wprowadzeniu kodu wymuszającego powrót do pozycji początkowej, gdy tarcza znajdzie się bardzo blisko tej pozycji.

Testy i optymalizacja wykazały, że gra działa płynnie i stabilnie na różnych konfiguracjach sprzętowych, a wdrożone mechaniki zostały starannie przetestowane pod kątem błędów. Ostatecznie gra charakteryzuje się wysoką wydajnością, co potwierdzają przeprowadzone testy, a wszystkie zaimplementowane elementy zostały zoptymalizowane, by zapewnić graczowi satysfakcjonujące i bezproblemowe doświadczenie.

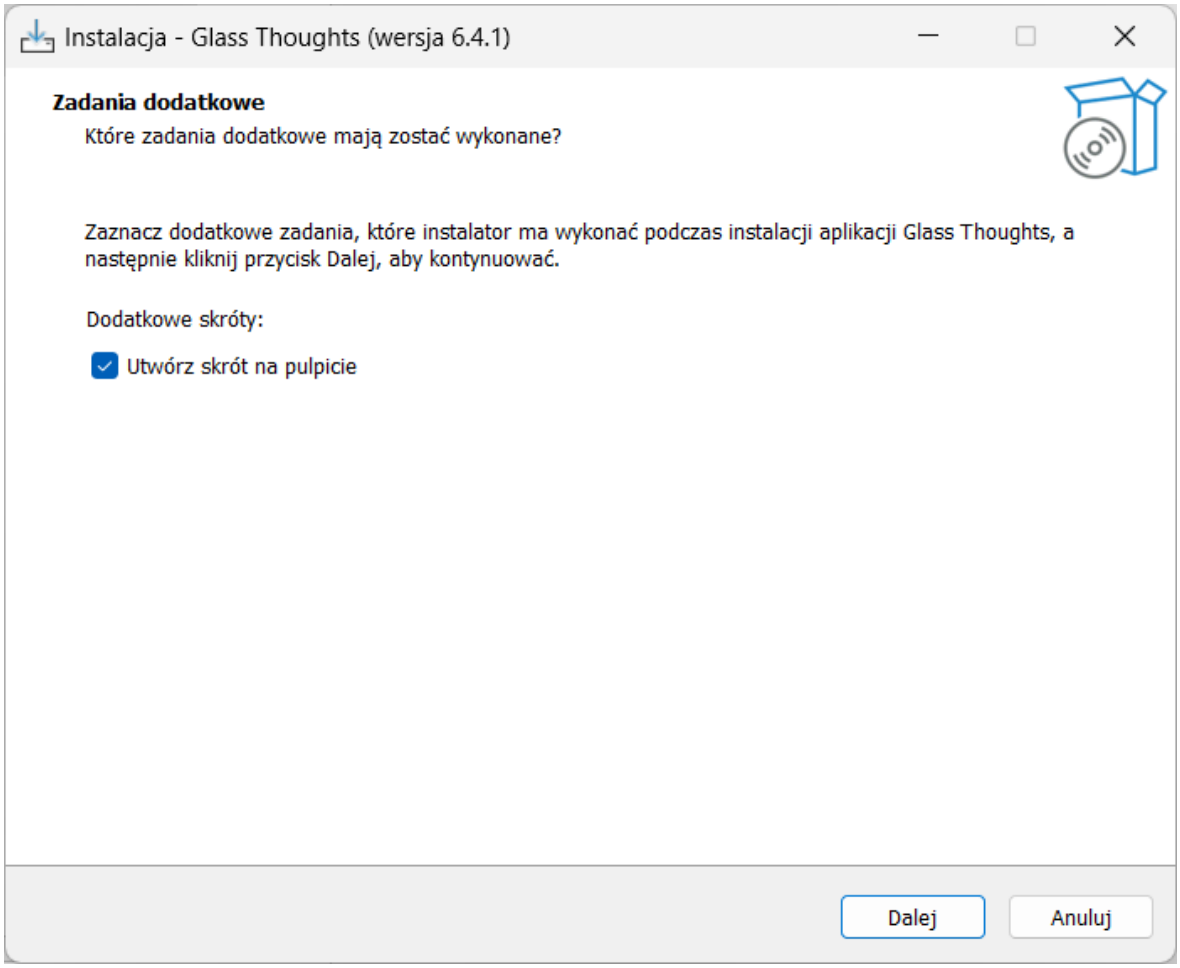
4.6. Proces implementacji instalatora

Aplikacja została opracowana z myślą o systemie Windows 11, zapewniając pełną kompatybilność, jednak dzięki standardowym funkcjom kompilacji w silniku Unity, gra działa również na systemie Windows 10, co potwierdzono podczas testów stabilności. Aby ułatwić pobieranie i zmniejszyć rozmiar plików, przygotowano dedykowany instalator stworzony przy użyciu programu InnoSetup. Instalator eliminuje konieczność ręcznej konfiguracji oraz rozpakowywania plików, co sprawia, że proces instalacji jest szybki i intuicyjny.



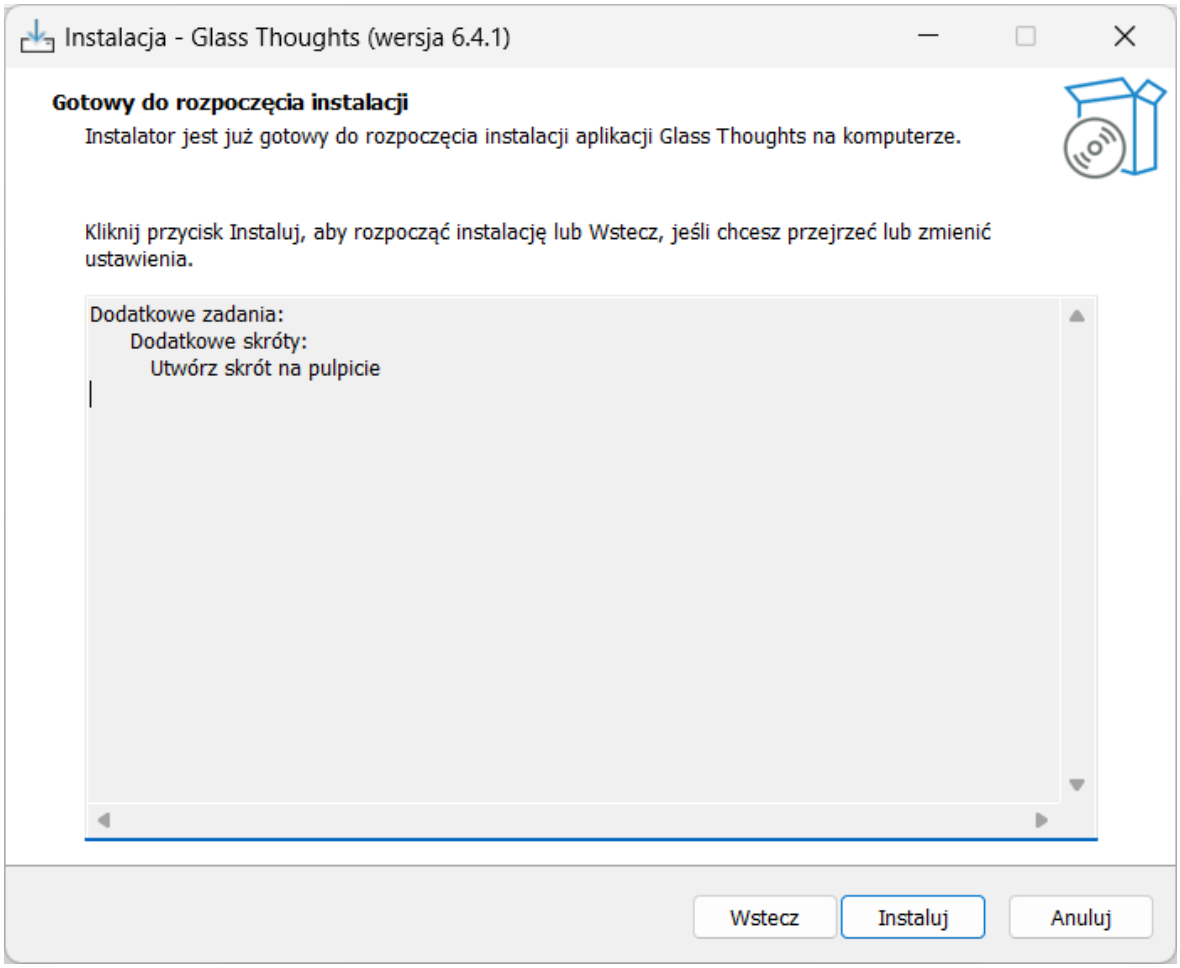
Rysunek 4.25 Wybór języka w instalatorze gry

Instalacja przebiega w kilku prostych krokach, zaprojektowanych w sposób przystępny dla użytkownika. Pierwszym etapem jest wybór języka interfejsu instalatora – do dyspozycji użytkownika dodano kilka popularnych języków europejskich, co ułatwia instalację osobom, które nie posługują się językiem polskim.



Rysunek 4.26 Tworzenie skrótu na pulpicie

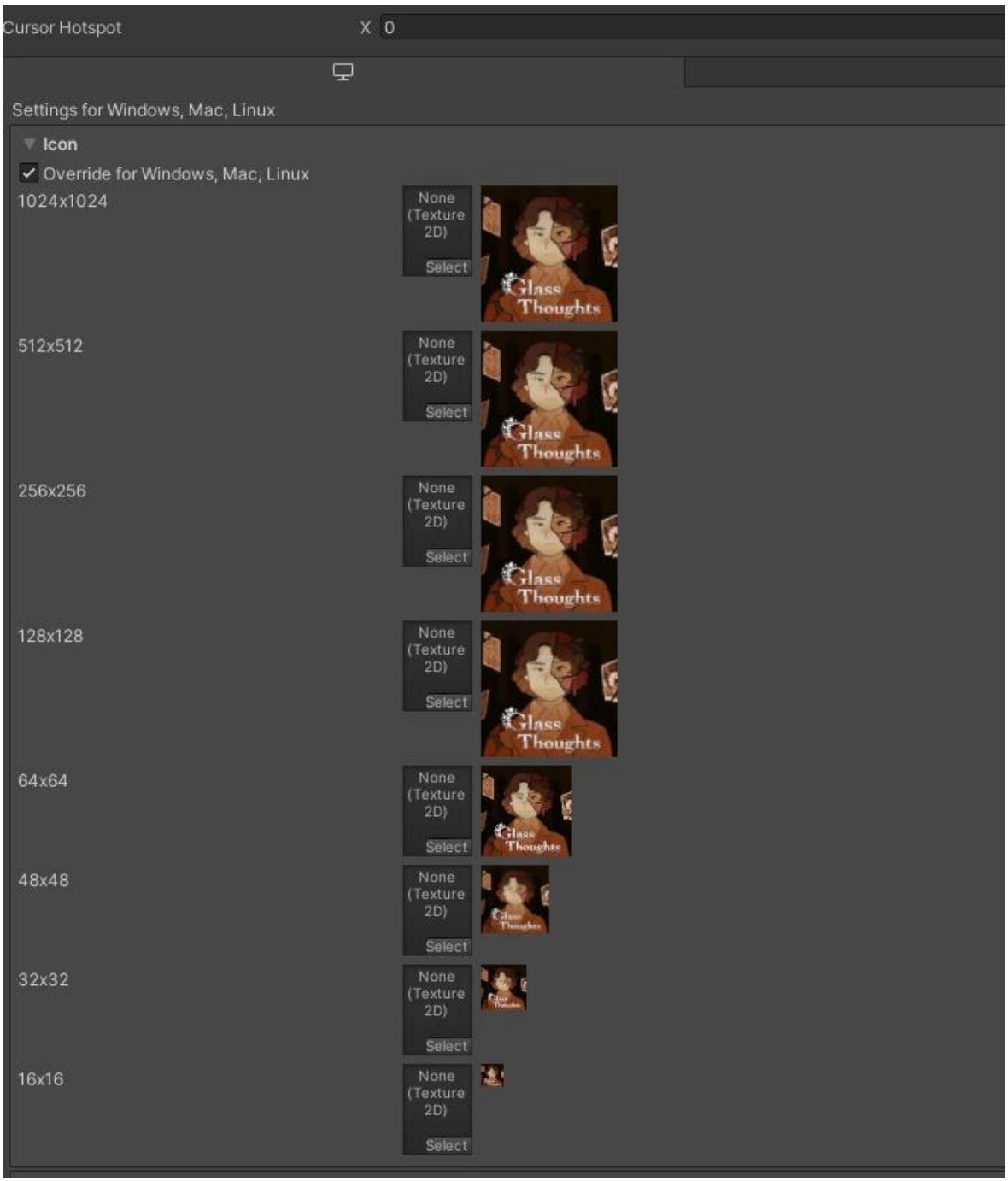
Dodatkowo istnieje możliwość wyboru dodatkowych funkcji, które mogą zostać uwzględnione podczas instalacji. Przykładem jest automatyczne utworzenie skrótu do aplikacji na pulpicie, co pozwala na szybkie uruchomienie gry bez konieczności przeszukiwania katalogów systemowych. Następnie wyświetlane jest podsumowanie konfiguracji, informujące o gotowości do rozpoczęcia instalacji. Na tym etapie użytkownik może kontynuować proces, klikając przycisk „Instaluj” lub wrócić do wcześniejszych ustawień, korzystając z opcji „Wstecz”.



Rysunek 4.27 Okienko instalacji w instalatorze

Po zakończeniu instalacji pojawia się możliwość natychmiastowego uruchomienia gry, a aplikacja zostaje zainstalowana z dostosowaną ikoną, wyróżniającą ją na tle innych programów w systemie. Ikona została dodana poprzez silnik Unity w zakładce „Icon for Windows", co umożliwia personalizację wyglądu aplikacji w systemie operacyjnym.

Rysunek 4.28 Ikonka gry na pulpicie ekranu. Projekt graficzny ikony Ambre Buathier



Rysunek 4.29 Zakładka ikon w silniku Unity. Projekt graficzny ikony Ambre Buathier

5. Kierunek dalszego rozwoju

W obecnej formie gra stanowi kompleksowe doświadczenie łączące elementy *visual novel, point-and-click* oraz mini gier logicznych o charakterze detektywistycznym. Mimo to istnieje wiele możliwości dalszej rozbudowy projektu, które mogą znacząco wzbogacić wrażenia płynące z rozgrywki. Jednym z głównych kierunków jest ponowne zaimplementowanie oraz rozszerzenie funkcjonalności galerii w menu gry. Pierwotnie planowana opcja pozwalałaby graczom przeglądać odblokowane ilustracje, *concept arty* i dodatkowe materiały związane z fabułą czy postaciami. Obecnie funkcja ta została wyłączona z powodu trudności technicznych, jednak w przyszłości planowane jest jej ponowne wdrożenie. Udostępnienie galerii dałoby graczom dodatkową motywację do eksploracji, odblokowywania sekretów i podejmowania różnych decyzji fabularnych.

Kolejną możliwością jest portowanie gry na inne platformy, w tym urządzenia mobilne działające pod systemami Android i iOS. Unity jako elastyczny i wieloplatformowy silnik, ułatwia przeprowadzenie procesu dostosowania interfejsu i sterowania do ekranów dotykowych. Tym samym zwiększyłaby się dostępność gry dla szerszego grona odbiorców, a gracze mogliby cieszyć się rozgrywką na swoich smartfonach czy tabletach. Wreszcie, stworzone podczas prac systemy – takie jak mechanika obsługi telefonu, integracja scen *point-and-click* czy moduł zarządzania dialogami w stylu *visual novel* – mogą zostać wykorzystane ponownie w innych projektach. Udostępnienie ich w formie komercyjnych pakietów (np. na Unity Asset Store) albo wykorzystanie jako bazy dla nowych gier, pozwoliłoby na dalszy rozwój uniwersalnych rozwiązań i szybsze prototypowanie kolejnych tytułów.

6. Podsumowanie

Warto podkreślić, że proces tworzenia gier nie podlega jednolitym standardom – każda produkcja wymaga indywidualnego podejścia do mechanik, narracji i technologii. Jak trafnie zauważono: „A video game can be just about anything, from a two-dimensional iPhone puzzler to a massive open-world RPG with über-realistic graphics, and it shouldn't be too shocking to discover that there are no uniform standards for how games are made. Lots of video games look the same, but no two video games are created the same way." [7]Różnorodność metodologii projektowania była wyraźnie widoczna również w tej pracy – integracja wielu gatunków, mechanik i systemów wymagała indywidualnego podejścia do implementacji każdego elementu, co doprowadziło do stworzenia gry wyróżniającej się kompleksowym systemem interpretatora tekstu oraz interaktywnymi przerywnikami. Każda z opracowanych mechanik została zaprojektowana specjalnie na potrzeby tego projektu i napisana od podstaw, co pozwoliło na pełną kontrolę nad ich działaniem oraz spójność całej produkcji. Dzięki przemyślanej architekturze projektu oraz komponentowej strukturze, poszczególne systemy mogą być łatwo integrowane w innych grach. System *visual novel*, radio oraz mechanika wykonywania połączeń telefonicznych zostały zaprojektowane modułowo, co umożliwia ich łatwe odłączenie i ponowne wykorzystanie w przyszłych produkcjach. Takie rozwiązanie znacząco zwiększa uniwersalność opracowanych mechanizmów i pozwala na ich adaptację w różnorodnych projektach, niezależnie od gatunku czy konwencji. Praca ta zaowocowała nie tylko stworzeniem rozbudowanej gry, ale przede wszystkim opracowaniem kompleksowych mechanik oraz zaawansowanych systemów, które mogą znaleźć zastosowanie w kolejnych projektach. Skrypty zostały zoptymalizowane pod kątem stabilności i wydajności, a łączna objętość kodu przekracza 600 tysięcy znaków, co świadczy o skali przedsięwzięcia oraz ogromnym nakładzie pracy włożonym w jego realizację. Całość produkcji stanowiła wieloaspektowe i czasochłonne przedsięwzięcie, wymagające nie tylko szczegółowego planowania, ale także integracji licznych elementów w celu stworzenia spójnego i angażującego świata. Dzięki modularnej architekturze i elastyczności zastosowanych rozwiązań, projekt ten może stanowić solidną bazę do dalszego rozwoju oraz inspirację dla kolejnych gier opartych na podobnych mechanikach. Ostateczny rezultat jest świadectwem ogromu włożonej pracy oraz dowodem na to, że dobrze zaprojektowane systemy mogą być nie tylko skuteczne w obrębie jednej produkcji, ale także stanowić fundament dla przyszłych, bardziej zaawansowanych projektów.

Spis rysunków

Bibliografia

[1] M. J. S. A. Reed A, „ADVENTURE GAMES PLAYING THE OUTSIDER,” New York, Bloomsbury Publishing Inc, 2020, p. 24.

[2] S.-S. B. Avedon E, „The Study of Games,” New York, John Wiley & Sons, 1971 , p. 405.

[3] S. E, „VIDEO GAME STORYTELLING,” New York , Watson-Guptill , 2014 , p. 166.

[4] S. J, „The Art. Of Game Design A Book of Lenses,” Burlington , Morgan Kaufmann Publishers, 2008 , p. 147.

[5] K. R, „Theory Of Fun For Game Design 2nd Edition,” Sebastopol , O'Reilly Media, 2013 , p. 166.

[6] G. J, „Game Engine Architecture Second Edition,” Boca Raton , CRC Press, 2014 , p. 940.

[7] S. J, „BLOOD, SWEAT, AND PIXELS,” New York , HarperCollins , 2017 , p. 8.