System Antyplagiat

Tytuł
Dok1.docx
Autor/zy
Mikołaj Łapiński

Data raportu **7.01.2025**

Raport z wykrywania treści AI

Wyniki oparte są na przewidywaniu czy tekst został utworzony przy użyciu narzędzi sztucznej inteligencji opartych na GPT tj. GPT-2, GPT-J, GPT-NEO, GPT-3 lub GPT-4, Przykładowe narzędzia to ChatGPT, Bing, Gemini, Jasper..

Średnie prawdopodobieństwo wykorzystania AI **94%**

Algorytm określa prawdopodobieństwo wykorzystania AI dla poszczególnych fragmentów dokumentu. Ogólny wskaźnik przedstawia średnie prawdopodobieństwo wykorzystania AI dla całego dokumentu. Wynik 90% należy traktować jako 90% pewności, że treść została stworzona przy użyciu narzędzi opartych o AI.

Na tekście zaznaczono prawdopodobieństwo wykorzystania AI dla poszczególnych fragmentów dokumentu i pogrupowano je w 5 zakresów zgodnie z legendą.

- 0 - 20%
- 21 - 40%
- 41 - 60%
- 61 - 80%
- 81 - 100%

Zawartość treści AI w tekście **96%**

Wskaźnik określa jaka część tekstu została zakwalifikowana jako treść wytworzona przy użyciu narzędzi opartych na AI. Za treści wytworzone przez AI system uznaje fragmenty, dla których prawdopodobieństwo wykorzystania AI przekracza wartość progową, domyślnie 0.8 (w skali od 0 do 1). Wynik 90% oznacza, że 90% tekstu zostało stworzone przez sztuczną inteligencję.

Szczegóły

W tabeli poniżej znajdują się fragmenty, uszeregowane od tych, które najprawdopodobniej zostały wygenerowane przez model językowy AI, do tych o najmniejszym prawdopodobieństwie.

LP	FRAGMENT	WYKRYWANIE TREŚCI AI	LICZBA SŁÓW
1	Rozdział 1 Analiza istniejących rozwiązań: 1. Cha...	97%	1287
2	3.2 Architektura projektu narzędzia wspomagającego...	95%	109
3	Ryc.4.2.5. Przedstawienie zewnętrzne, jako blok fu...	95%	1792
4	4.1.2. Ryc. 4.1.2. Logika przypisania tekstu. Ź...	94%	452
5	Po stworzeniu przycisku może on być wielokrotnie p...	93%	727
6	Dzięki temu narzędzie może dynamicznie przetwarzać...	90%	270
7	Ryc. 4.1.1. Wartości w Base_widget_dialog. Źródło:...	88%	74
8	W tej sekcji zaprezentowano ich ogólny podział i z...	83%	137
9	Dodatkowo taki przemyślany schemat pozwala powiela...	80%	22
10	Pozwalają one na tworzenie mechanizmów umożliwiają...	78%	46
11	Jest to użyteczne, gdy element interfejsu użytkown...	78%	36
12	Każda z tych zmiennych odpowiada za określony elem...	78%	34
13	wyzwalanie działań pomiędzy poszczególnymi fazami ...	77%	34
14	Takie przyciski tworzymy na podstawie bazowego wid...	77%	8

15	W przypadku braków (np. braku tekstu) funkcja gene...	75%	45

Rozdział 1 Analiza istniejących rozwiązań:
1. Charakterystyka istniejących narzędzi do tworzenia dynamicznych kampanii w grach komputerowych
Tworzenie dynamicznych kampanii w grach komputerowych wymaga odpowiednich narzędzi, które umożliwiają zarządzanie narracją, interakcjami gracza oraz zmianami w rozgrywce. Na rynku dostępnych jest kilka narzędzi, które w różnym stopniu pozwalają na realizację tych zadań.
Jednym z popularniejszych narzędzi jest RPG Maker, to program stworzony z myślą o amatorskich twórcach gier RPG. Umożliwia on korzystanie z gotowych zasobów i szablonów, co czyni go łatwym w obsłudze nawet dla osób bez doświadczenia w programowaniu. Narzędzie to posiada wbudowany system wydarzeń, który pozwala na tworzenie prostych interakcji i narracji. Jednakże jego ograniczenia są widoczne w bardziej złożonych projektach - brak elastyczności i zaawansowanych mechanizmów sprawia, że RPG Maker nadaje się głównie do tworzenia gier w jednym określonym gatunku.
Innym rozwiązaniem jest Unity, to wszechstronny silnik gier, który umożliwia tworzenie dynamicznych kampanii poprzez skrypty w języku C#. Dzięki bogatej dokumentacji i szerokim możliwościom technologicznym Unity stało się jednym z najpopularniejszych narzędzi w branży. Pomimo swojej elastyczności, Unity wymaga zaawansowanych umiejętności programistycznych. Tworzenie dynamicznych kampanii wymaga pisania wielu linii kodu oraz budowania systemów narracyjnych praktycznie od podstaw, co znacząco wydłuża czas pracy.
Warto także wspomnieć o Twine, to narzędzie specjalizujące się w interaktywnych opowieściach. Jest intuicyjne i proste w obsłudze, co czyni je idealnym dla projektów narracyjnych, w których tekst stanowi główny element rozgrywki. Twine umożliwia łatwe tworzenie wyborów gracza oraz interakcji, jednak ogranicza się do narracji tekstowej i nie oferuje wsparcia dla bardziej zaawansowanych funkcji, takich jak animacje, efekty dźwiękowe czy złożona mechanika gry.
Ostatnim narzędziem jest Unreal Engine, to jeden z najpotężniejszych silników gier, który dzięki systemowi Blueprintów pozwala na tworzenie zaawansowanych mechanik bez potrzeby pisania kodu w językach programowania. Unreal Engine oferuje szerokie możliwości integracji dynamicznych tekstów z innymi elementami gry, takimi jak animacje czy dźwięki. Jednak brak dedykowanego systemu do zarządzania tekstami sprawia, że proces ten wymaga dużej liczby manualnych kroków, co może być czasochłonne i mniej intuicyjne.
2. Słabe strony analizowanych rozwiązań
Każde z omówionych narzędzi ma swoje ograniczenia, które mogą wpłynąć na efektywność tworzenia dynamicznych kampanii.
W przypadku RPG Maker główną wadą jest brak wsparcia dla bardziej zaawansowanych mechanik. Narzędzie to zostało zaprojektowane z myślą o prostych grach RPG i trudno je zastosować w innych gatunkach.
Unity, choć bardzo elastyczne, wymaga od użytkowników umiejętności programistycznych, co może stanowić barierę dla mniej doświadczonych twórców. Tworzenie dynamicznych kampanii w Unity często wymaga pisania długich i skomplikowanych skryptów, co zwiększa czas produkcji.
Twine, mimo intuicyjności, jest ograniczone do narracji tekstowej i nie oferuje narzędzi pozwalających na łatwą integrację z bardziej zaawansowanymi mechanikami gier, takimi jak animacje czy dźwięki.
Z kolei w Unreal Engine brakuje wbudowanego systemu zarządzania sekwencjami tekstów oraz mechanizmów wyborów gracza. Tworzenie dynamicznych tekstów wymaga wielu kroków, takich jak ręczne budowanie widgetów i integrowanie ich z logiką gry, co może być czasochłonne.
3. Propozycje unikania słabych stron w projekcie
Unreal Engine 5 jest zaawansowanym narzędziem do tworzenia gier, oferującym nowoczesne technologie, takie jak Lumen i Nanite, które znacząco podnoszą jakość wizualną oraz wydajność. Mimo tych zalet, w kontekście tworzenia dynamicznych kampanii nadal istnieją pewne ograniczenia, które mogą wpłynąć na proces deweloperski.
Jednym z głównych wyzwań jest brak wbudowanego, dedykowanego systemu do zarządzania dynamicznymi kampaniami. Twórcy muszą samodzielnie implementować mechanizmy narracyjne, co wymaga tworzenia kompleksowych struktur w Blueprintach lub pisania kodu w języku C++. Choć system Blueprintów znacząco ułatwia budowanie logiki gry, bardziej zaawansowane mechanizmy, takie jak rozbudowane drzewa dialogowe czy dynamiczne zmiany fabularne, mogą wymagać dodatkowej pracy i optymalizacji.
W Unreal Engine 5 proces tworzenia dynamicznych tekstów i wyborów gracza wymaga wielu kroków. Twórcy muszą ręcznie tworzyć widgety, zarządzać wyświetlaniem tekstów oraz integrować je z innymi elementami gry, takimi jak animacje czy efekty dźwiękowe. Brak narzędzi do automatyzacji tych procesów wydłuża czas pracy i może być trudny dla mniej doświadczonych programistów.
Dodatkowo, zarządzanie dużymi ilościami danych, takimi jak sekwencje tekstów czy interakcje gracza, może być wyzwaniem w przypadku projektów o większej skali. Twórcy muszą zaprojektować własne systemy przechowywania danych i zarządzania nimi, co wymaga dodatkowego wysiłku.
Mimo zaawansowanych technologii, takich jak Nanite i Lumen, Unreal Engine 5 nadal wymaga dużej dbałości o optymalizację w przypadku gier o dużych otwartych światach lub złożonych kampaniach dynamicznych. Zarządzanie zasobami oraz utrzymanie płynności działania gry pozostają kluczowym wyzwaniem w kontekście projektów, które wykorzystują dynamiczne zmiany fabularne.
Rozdział 2 Charakterystyka narzędzi technologicznych
2.1 Charakterystyka narzędzi technologicznych
Do stworzenia mojego projektu użyłem głównie dwóch programów:
1. Unreal Engine 5 - Został wybrany jako główny silnik gry, a Blueprinty pełniły kluczową rolę w implementacji logiki gry. Blueprinty w Unreal Engine 5 to wizualny system skryptowy, który umożliwia tworzenie zaawansowanej logiki bez potrzeby pisania tradycyjnego kodu. Cała logika gry, w tym interakcje, generowanie dynamicznych tekstów oraz jednolitość interfejsów, została zaimplementowana właśnie za pomocą Blueprintów. Dzięki temu proces tworzenia gry stał się szybszy i bardziej intuicyjny, co umożliwiło łatwe modyfikowanie oraz iterowanie różnych elementów w grze.
Jednym z kluczowych elementów, które zostały zaimplementowane, były funkcje do dynamicznego pisania tekstu oraz tworzenia wyborów, które ułatwiły implementację dynamicznej kampanii. Te funkcje umożliwiają płynne wprowadzanie zmieniających się tekstów w grze i oferowanie graczowi opcji wyboru, co jest fundamentem dla rozwoju narracji w dynamicznych kampaniach.
Kluczowe funkcje Unreal Engine 5:
1. Łatwa implementacja logiki gry w trakcie projektowania.
2. Szybsze wyszukiwanie błędów i ich naprawa.
3. Ułatwienie dla osób, które będą próbowały tworzyć własne funkcje bazujące na istniejącym kodzie.
4. Dynamiczne generowanie tekstu i wybory gracza, umożliwiające tworzenie rozbudowanych scenariuszy i kampanii.
2. Blender - Został wybrany do tworzenia zasobów 3D, takich jak modele postaci, obiektów i tekstury, które następnie zostały zaimportowane do Unreal Engine 5 w celu dalszej obróbki i implementacji. Blender jest darmowym, open-source'owym narzędziem do modelowania 3D, które oferuje

rozbudowane funkcje umożliwiające tworzenie wysokiej jakości zasobów wizualnych, w tym renderowanie, symulację fizyczną oraz teksturowanie.
W projekcie wykorzystano głównie następujące funkcje Blendera:
1. Modelowanie 3D - Tworzenie postaci, obiektów i środowisk, które zostały zaimportowane do Unreal Engine 5.
2. Teksturowanie - Tworzenie tekstur i map, które nadawały realizmu postaciom i obiektom 3D, co miało kluczowe znaczenie dla estetyki projektu.
3. Export do Unreal Engine 5 - Blender zapewnia łatwą integrację z Unreal Engine 5 dzięki opcji eksportu modeli w standardowych formatach (np. FBX), co umożliwiło płynne przenoszenie zasobów 3D między tymi narzędziami.

Oba programy były bardzo pomocne w trakcie pracy, jednak eksport pomiędzy Blenderem a Unreal Engine 5 wiązał się z pewnymi trudnościami. Oto niektóre z ograniczeń:
1. Materiały i shadery - Różnice w systemach materiałów między Blenderem a Unreal Engine powodują konieczność rekonstrukcji materiałów w Unreal Engine. Tekstury, takie jak mapy normalnych czy albedo, są poprawnie przenoszone, jednak niestandardowe shadery wymagają ponownego stworzenia w Unreal Engine.
2. Animacje materiałów - Animacje materiałów muszą być zaimplementowane na nowo w Unreal Engine, ponieważ oba silniki korzystają z różnych systemów animacji materiałów.
3. Transformacje i skale - Różnice w jednostkach miary mogą powodować problemy z wielkością obiektów po imporcie. Często konieczne jest dokonanie korekty w Unreal Engine, aby zachować odpowiednią skalę w grze.
Rozdział 3 Założenia projektowe i architektura projektu:
3.1 Wymagania projektowe narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych
Wymagania dla projektowanego narzędzia opierają się na potrzebie przyspieszenia procesu implementacji dynamicznych tekstów i wyborów w grze, a także na ułatwieniu integracji z istniejącymi systemami Unreal Engine. Narzędzie powinno umożliwiać szybkie tworzenie i modyfikowanie treści tekstowych bez potrzeby zaawansowanego kodowania, co jest szczególnie istotne w dynamicznych kampaniach, gdzie treści zmieniają się w zależności od decyzji gracza.
Kluczowym wymaganiem było zapewnienie możliwości sterowania akcjami i zdarzeniami w trakcie wyświetlania tekstu lub po jego zakończeniu, co umożliwia wzbogacenie rozgrywki o dodatkowe interakcje i dynamiczne reakcje. Istotnym elementem projektu było także uwzględnienie kompatybilności z systemem Blueprintów, co pozwala na łatwe wdrażanie i rozbudowę narzędzia w środowisku Unreal Engine.
Ponadto narzędzie musi być elastyczne, aby obsługiwać różnorodne typy tekstów, wybory gracza oraz inne formy interakcji, jak np. wyzwalanie działań pomiędzy poszczególnymi fazami wyświetlania tekstu (pisanie, zatrzymanie, zakończenie). Ważnym aspektem jest także zoptymalizowanie narzędzia pod kątem Unreal Engine 5.3, co zapewnia pełne wykorzystanie dostępnych funkcji, takich jak mechanizmy Blueprintów i wbudowane timery.
3.2 Architektura projektu narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych.

Architektura narzędzia została zaprojektowana w sposób modułowy, co ułatwia jego integrację z różnymi aspektami rozgrywki. Centralnym elementem narzędzia jest widget, który zarządza wszystkimi funkcjami związanymi z dynamicznymi tekstami. Widget ten pełni kluczową rolę w wyświetlaniu tekstów oraz w synchronizacji z innymi elementami gry, takimi jak timery czy akcje wywoływane przez gracza.
Każdy menedżer, zaprojektowany jako komponent Actor, przechowuje sekwencję tekstów i oferuje funkcje zarządzania nimi w sposób zbliżony do uproszczonego drzewa dialogowego. Sekwencje te umożliwiają programiście sterowanie zdarzeniami w trakcie wyświetlania tekstu oraz definiowanie czasu, po którym tekst ma zniknąć z ekranu.
Funkcje wyboru są integralnym elementem narzędzia. Pozwalają one na tworzenie mechanizmów umożliwiających graczowi wybór z wielu opcji w grze. Dzięki specjalnie przygotowanym funkcjom w Blueprintach programista może z łatwością konfigurować przyciski wyboru oraz ich zachowanie w zależności od decyzji gracza.
Narzędzie obsługuje stringi, które są przechowywane jako sekwencje znaków definiowane przez programistę. Dzięki temu narzędzie może dynamicznie przetwarzać i wyświetlać teksty w czasie rzeczywistym.
Zoptymalizowana architektura umożliwia szybką implementację widżetów oraz ich integrację z resztą gry. Dzięki tym usprawnieniom, programiści mogą oszczędzać czas i skupić się na rozwijaniu narracji oraz innych aspektów gry, które wymagają interakcji z graczami.
Rozdział 4 Implementacja projektu:
4.1 Implementacja poszczególnych komponentów narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych

Pakiet Funkcji pisania tekstu został zaprojektowany jako modularny zestaw funkcji i komponentów, które współpracują ze sobą w celu usprawnienia procesu implementacji pod dynamiczną narrację. Każdy z elementów systemu pełni określoną rolę i integruje się z innymi funkcjami Unreal Engine 5, co umożliwia tworzenie złożonych interakcji przy minimalnym wysiłku programistycznym. W ramach projektu szczególną uwagę poświęcono widgetom odpowiedzialnym za obsługę interfejsu użytkownika, menedżerowi tekstów zarządzającemu logiką narracji oraz funkcji wyborów gracza, które zapewniają interaktywność.
Widget, stanowiący kluczowy element wizualny narzędzia, został zaprojektowany w systemie Blueprintów Unreal Engine. Obsługuje on wyświetlanie tekstów w dynamiczny sposób, umożliwiając zastosowanie efektów wizualnych, takich jak pisanie tekstu na ekranie. Menedżer tekstów, zaimplementowany jako Actor Component, przechowuje i zarządza sekwencjami tekstów w formie stringów. Umożliwia to programiście kontrolowanie wyświetlanych treści oraz przypisywanie do nich odpowiednich akcji, takich jak odtwarzanie dźwięków, zmiana stanu gry lub uruchamianie animacji.
Funkcja wyborów gracza pozwala na tworzenie opcji interakcji, które gracz może wybierać, a ich implementacja jest łatwa dzięki wbudowanym funkcjom Blueprintów.
W kolejnych sekcjach opisano szczegółowo implementację każdego z tych elementów, ze szczególnym uwzględnieniem ich integracji oraz zastosowania w dynamicznych kampaniach narracyjnych.
Główny Widget do pisania tekstu, jak zostało wspomniane, to główny element projektu. Jego konfiguracja jest całkiem prosta. Na poniższej ilustracji (Ryc. 4.1.1) przedstawiono strukturę zmiennych w Base_widget_dialog. Każda z tych zmiennych odpowiada za określony element interfejsu użytkownika, taki jak wyświetlany tekst lub obrazek postaci. Dzięki takiemu podejściu możliwe jest dynamiczne przypisywanie wartości w trakcie gry, co ułatwia konfigurację i modyfikację dialogów.

Ryc. 4.1.1. Wartości w Base_widget_dialog. Źródło: Opracowanie własne.

1. Tekst - przypisuje się do niego wartość pola tekstowego, które ma być edytowane.
2. Obrazek Gracza - przypisuje się do niego obrazek, który ma być edytowany, przez funkcje

Dzięki zastosowaniu takich rozwiązań, dołączone funkcje w menadżerze tekstu mają stałą implementację, jakie pole tekstowe i obrazek jest przez nich edytowane, a dzięki temu że dam widget jest bazą pod inne widgety, które będą od niego tworzone. Dodatkowo taki przemyślany schemat pozwala powielać i tworzyć różne wersje widgetów dialogu. Przykładowy poprawny układ zaimplementowania takiego widgetu jest przedstawiony na Ryc. 4.1.2.

Ryc. 4.1.2. Logika przypisania tekstu. Źródło: Opracowanie własne.

Ryc. 4.1.2 przedstawia szczegółową logikę przypisania zmiennych "Tekst" i "Obrazek Gracza" w widgetach dialogowych. Dzięki zastosowaniu selektorów zmiennych, programista może łatwo przypisać elementy wizualne, takie jak "Text Block" i "Image", bez konieczności ręcznego modyfikowania widgetu. Taka konstrukcja umożliwia szybkie tworzenie i dostosowywanie dialogów w grze, co znacząco skraca czas potrzebny na implementację dynamicznych kampanii.
Przykładowy i podstawowym układ takiego widgetu został przedstawiony na Ryc. 4.1.3

Ryc. 4.1.3. Podstawowy układ widgetu odpowiedzialnego za pisanie tekstu. Źródło: Opracowanie własne
Ryc. 4.1.3 przedstawia podstawowy przykład umiejscowienia elementów "Text Block" i "Image". Dzięki elastyczności układu wizualnego, projektant ma możliwość dowolnego rozmieszczenia tych komponentów oraz dostosowania ich parametrów wizualnych, takich jak czcionka, kolory czy rozmiary, do potrzeb projektu.

Kolejną główną funkcjonalnością jest Menadżer Tekstu (Base_Actor_Component), który pełni rolę centralnego komponentu zarządzającego dynamicznymi sekwencjami tekstów w grze. Menadżer ten odpowiada za przechowywanie, obsługę oraz synchronizację tekstów wyświetlanych w interfejsie użytkownika. Dzięki modularnej konstrukcji możliwa jest łatwa integracja z innymi elementami gry, takimi jak animacje, efekty dźwiękowe czy mechanizmy wyborów gracza. Menedżer pozwala również na wyzwalanie akcji po każdym zakończonym działaniu przypisanej funkcji, co umożliwia tworzenie bardziej złożonych scenariuszy narracyjnych.
Narzędzie to zostało zaimplementowane jako komponent Actor w Unreal Engine, co umożliwia jego wielokrotne wykorzystanie i elastyczne dostosowanie do różnych scenariuszy narracyjnych. Działanie Menedżera Tekstu opiera się na tworzeniu ścieżek wykonywanych czynności, które obejmują zmianę tekstu, dynamiczne tworzenie widgetów lub ich usuwanie zgodnie z potrzebami narracji. Struktura logiczna funkcji przypomina drzewo, w którym każdy tekst stanowi główną linię (pień), a dodatkowe akcje mogą odgałęziać się jako pomniejsze procesy (gałęzie). Funkcje wyboru, opisane w kolejnych sekcjach, pozwalają na tworzenie bardziej zaawansowanych drzew narracyjnych z wieloma możliwościami interakcji dla gracza.
Menadżer Tekstu posiada w sobie wiele funkcji odpowiadających za pisanie tekstu, które są nazwane "Macro_pisania...". Nazewnictwo to zostało przyjęte, aby jednoznacznie określać ich zadania oraz procesy wykonywane przed i po ich działaniu. Makra te pozwalają na elastyczne zarządzanie tekstami w zależności od potrzeb narracyjnych gry. Rozróżniamy cztery główne makra:
1. macro_pisania_od_widgetu
To macro służy do obsługi tekstu, który jest przypisany do istniejącego już widgetu. Pozwala na dynamiczne wprowadzanie zmian w treści tekstu bez konieczności tworzenia nowego widgetu. Jest przydatne w sytuacjach, gdy tekst w jednym elemencie interfejsu użytkownika musi być aktualizowany
2. wielokrotnie.macro_pisania_ze_stworzeniem_widgetu
To macro tworzy nowy widget przed rozpoczęciem wyświetlania tekstu. Jest stosowane, gdy tekst musi być wyświetlany w nowym kontekście, np. gdy gracz otwiera nowe okno dialogowe.
3. macro_pisania_ze_stworzeniem_widgetu_ze_skasowaniem_po_napisaniu
Macro to tworzy nowy widget, wyświetla tekst, a następnie usuwa widget po zakończeniu wyświetlania. Funkcja ta jest używana w scenariuszach, gdzie tekst jest jednorazowy i nie wymaga dalszego utrzymywania widgetu w pamięci.
4. macro_pisania_od_widgetu_ze_skasowaniem_po_napisaniu
W tym przypadku tekst jest wyświetlany w istniejącym już widgetcie, ale widget zostaje usunięty po zakończeniu wyświetlania. Jest to użyteczne, gdy element interfejsu użytkownika jest potrzebny tylko na czas wyświetlania danego tekstu.
Bardziej szczegółowy opis działania poszczególnych makr, takich jak macro_pisania_od_widgetu, macro_pisania_ze_stworzeniem_widgetu czy macro_pisania_ze_stworzeniem_widgetu_ze_skasowaniem_po_napisaniu, zostanie przedstawiony w podrozdziale "4.2 Wyjaśnienie kluczowych fragmentów kodu". W tej sekcji zaprezentowano ich ogólny podział i zastosowanie, aby podkreślić elastyczność i wszechstronność Menedżera Tekstu.

Kolejną kluczową funkcjonalnością narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych jest funkcja wyboru, która umożliwia graczowi podejmowanie decyzji w trakcie rozgrywki. Funkcja ta została zaprojektowana jako integralna część Menedżera Tekstu i opiera się na specjalnie przygotowanych widgetach wyboru. Dzięki temu mechanizmowi gracz ma możliwość wyboru spośród zdefiniowanych opcji dialogowych, co wpływa na dalszy rozwój narracji lub przebieg rozgrywki.
Mechanizm funkcji wyboru działa w sposób modularny, co pozwala na łatwą konfigurację liczby opcji wyboru, ich treści oraz akcji, które są wywoływane w zależności od decyzji gracza. Elastyczność tego rozwiązania sprawia, że może być ono używane zarówno w prostych dialogach, jak i w bardziej złożonych scenariuszach narracyjnych, gdzie decyzje gracza mają bezpośredni wpływ na dalszy rozwój fabuły.
Podstawą działania funkcji są przyciski. Takie przyciski tworzymy na podstawie bazowego widgetu "Base_widget_wybór". Po stworzeniu przycisku może on być wielokrotnie powielany i dostosowywany do potrzeb konkretnej sceny lub dialogu. Obsługa tych przycisków odbywa się za pomocą dedykowanych funkcji macro w Menedżerze Tekstu, które odpowiadają za przypisywanie treści i akcji każdej opcji wyboru. Rozróżniamy cztery główne makra:

1. macro_wyboru_input_button_to_macro
To macro odpowiada za powiązanie przycisku z odpowiednią akcją. Po aktywowaniu przycisku wywoływana jest funkcja zdefiniowana w Menedżerze Tekstu, która wykonuje przypisaną logikę, np. kontynuację dialogu lub zmianę stanu gry.
2. macro_wyboru_create_with_out_class
Macro to umożliwia dynamiczne tworzenie przycisków wyboru bez konieczności wcześniejszego definiowania ich klas. Jest szczególnie przydatne w

sytuacjach, gdy liczba opcji wyboru jest zależna od kontekstu fabularnego lub decyzji gracza.
3. macro_wyboru_input_button_to_macro_with_cooldown
Działa podobnie jak macro_wyboru_input_button_to_macro, jednak wprowadza dodatkową funkcjonalność - cooldown. Po aktywacji przycisku wprowadzana jest czasowa blokada uniemożliwiająca jego ponowne użycie przez określony czas.
4. macro_wyboru_create_with_out_class_with_cooldown
Jest to połączenie funkcjonalności dynamicznego tworzenia przycisków i mechanizmu cooldown. Przyciski są generowane w czasie rzeczywistym i mogą być czasowo blokowane po użyciu.
Bardziej szczegółowy opis działania poszczególnych makr, takich jak macro_wyboru_input_button_to_macro, macro_wyboru_create_with_out_class, czy z cooldownami, przedstawiony w podrozdziale "4.2 Wyjaśnienie kluczowych fragmentów kodu". W tej sekcji zaprezentowano ich ogólny podział i zastosowanie, aby podkreślić elastyczność i wszechstronność Menedżera Tekstu.

Ryc.4.13. Wartości pojedynczego przycisku wyboru. Źródło: Opracowanie własne

Ryc. 4.1.3 przedstawia szczegółowe ustawienia widgetu wyboru (Base_widget_wyboru), który pełni kluczową rolę w funkcji wyboru. Wartości i właściwości tego widgetu zostały zaprojektowane w sposób umożliwiający elastyczną konfigurację dla różnych scenariuszy w grze.

Ryc.4.1.4. Struktura wyboru. Źródło: Opracowanie własne

Widget korzysta ze struktury Wybór, przedstawionej na Ryc. 4.1.4. Struktura ta pozwala na przypisanie konkretnych wartości każdemu przyciskowi wyboru. W jej skład wchodzą m.in.:
1. Tekst_do_przycisku_wyboru: Pole tekstowe typu String, które definiuje treść wyświetlaną na danym przycisku wyboru.
2. Wyznaczona_opcja_przycisku: Pole typu Enum (w tym przypadku Switche), które określa, jaka akcja lub wyjście jest związane z danym wyborem.
Implementacja typu Enum (Switche) pozwala projektantom łatwo określić, które wyjście odpowiada za konkretny wybór. Dzięki temu programista może jasno opisać każde wyjście, co ułatwia późniejsze przypisanie odpowiednich akcji.
Dodatkowo, widget zawiera funkcję opartą na evencie Construct. Funkcja ta automatycznie sprawdza, czy dla danego przycisku wyboru został zdefiniowany tekst. Jeśli tekst istnieje, jest on przypisywany do właściwego pola przycisku, co przyspiesza proces konfiguracji widgetu.
W Base_widget_wyboru znajduje się również niestandardowe zdarzenie (Custom Event) o nazwie "Zapisz Zmianę Nazwy". Zdarzenie to zostało zaprojektowane w celu ułatwienia kontrolowania zmian tekstu w przyciskach wyboru. Po podłączeniu go do zdarzenia Pre Construct w widgetach, do których dodawane są przyciski, możliwe jest automatyczne zarządzanie aktualizacjami tekstu, co dodatkowo usprawnia proces projektowania.

- Wyjaśnienie kluczowych fragmentów kodu

Projekt narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych opiera się na szeregu funkcji i makr, które zostały zaprojektowane w celu zapewnienia elastyczności, wydajności oraz łatwości integracji z istniejącymi systemami Unreal Engine 5. W niniejszym podrozdziale szczegółowo omówiono kluczowe elementy kodu, które odgrywają centralną rolę w realizacji funkcjonalności narzędzia.
Omówienie rozpoczyna się od bardziej ogólnych elementów, takich jak funkcje wspierające makra i ich logikę, a następnie przechodzi do coraz bardziej szczegółowych mechanizmów, które odpowiadają za konkretne aspekty działania systemu. Wśród opisanych fragmentów znajdują się zarówno podstawowe funkcje obsługujące widgety i logikę wyboru, jak i zaawansowane makra, takie jak Funkcja_czekania, które umożliwiają implementację dynamicznych interakcji oraz zarządzanie czasem w grze.
Poniżej przedstawiono szczegółowe wyjaśnienie każdego z kluczowych elementów kodu, które ilustrują, jak poszczególne komponenty współpracują w celu zapewnienia kompleksowego wsparcia dla dynamicznych kampanii.

Universalna_funkcja
Funkcja ta pełni rolę pomocniczą dla makr, stanowiąc bank pamięci dla poszczególnych kopii makr. Podczas testów można było zauważyć, że każda instancja makra przechowuje swoją pamięć niezależnie, co rozwiązało problem związany z nadpisywaniem danych. Funkcja jest używana głównie do zapisywania wartości typu bool i wyzwalania zdarzeń (Event). Po zapisaniu wartość pozostaje dostępna w pamięci, co czyni funkcję kluczową dla dynamicznych operacji w systemie.

Ryc.4.2.1. Przedstawienie zewnętrzne, jako blok funkcji Universalna_funkcja. Źródło: Opracowanie własne

Ryc.4.2.2. Przedstawienie kodu funkcji Universalna_funkcja. Źródło: Opracowanie własne

Wybór
Wybór to funkcja wspierająca działanie makr wyboru. Umożliwia szybkie tworzenie przycisków z klasy Base_widget_wybór i ich natychmiastowe dodawanie do interfejsu. Funkcja przyjmuje dane klasy przycisku oraz tworzy widget, który zostaje zapisany w zmiennej tymczasowej i przekazany dalej. Podczas działania funkcji następuje również przypisanie zdarzenia (Bind to Event), co umożliwia jej elastyczne użycie w różnych scenariuszach.

Ryc.4.2.3. Przedstawienie zewnętrzne, jako blok funkcji Wybór. Źródło: Opracowanie własne

Ryc.4.2.4. Przedstawienie kodu funkcji Wybór. Źródło: Opracowanie własne

Pisanie
Funkcja Pisanie służy do edycji widgetów Base_widget_dialog. Weryfikuje poprawność zaimplementowanego tekstu oraz sprawdza obecność innych

elementów, takich jak obrazy. W przypadku braków (np. braku tekstu) funkcja generuje komunikaty diagnostyczne dla programisty. Choć nie zwraca wyników na zewnątrz, jej zastosowanie pozwala na dynamiczne dostosowywanie dialogów i ich wizualnej reprezentacji w interfejsie. Funkcja może być używana np. w sekwencerach do tworzenia spersonalizowanych dialogów lub scen przerywnikowych.

Ryc.4.2.5. Przedstawienie zewnętrzne, jako blok funkcji Pisanie. Źródło: Opracowanie własne

Ryc.4.2.6. Przedstawienie kodu funkcji Pisanie. Źródło: Opracowanie własne

Create_widget

Prosta funkcja umożliwiająca szybkie tworzenie widgetów Base_widget_dialog. Funkcja może być używana w scenariuszach, takich jak inicjalizacja dialogów lub przygotowanie widgetów na potrzeby cut-scen.

Ryc.4.2.7. Przedstawienie zewnętrzne, jako blok funkcji Create_widget . Źródło: Opracowanie własne

Ryc.4.2.8. Przedstawienie kodu funkcji Create_widget. Źródło: Opracowanie własne

Zrzucenie_widgetu

Funkcja ta umożliwia bezpieczne usuwanie widgetów. Zawiera dodatkowe mechanizmy, takie jak ukrywanie wskaźnika myszy i resetowanie sterowania. Działa na przekazanym widgetcie, który ma zostać usunięty, i opcjonalnie przyjmuje kontroler, który ma być używany po usunięciu widgetu.

Ryc.4.2.9. Przedstawienie zewnętrzne, jako blok funkcji Zrzucenie_widgetu. Źródło: Opracowanie własne

Ryc.4.2.10. Przedstawienie kodu funkcji Zrzucenie_widgetu. Źródło: Opracowanie własne

Funkcja_do_oczekiwania

Macro to obsługuje dwa wejścia: podstawowe "Wejscie_podstawowe” i programowalne "Wejście po wykonaniu”. Używa funkcji Universalna_funkcja do zarządzania stanem i przełączania między etapami działania. Może przechowywać stan funkcji oraz reagować na sygnały wywołujące jej kolejne operacje. To wszechstronne rozwiązanie umożliwia kontrolę przepływu logiki w systemie, szczególnie w scenariuszach wymagających dynamicznego oczekiwania.

Ryc.4.2.11. Przedstawienie zewnętrzne, jako blok macro Funkcja_do_oczekiwania . Źródło: Opracowanie własne

Ryc.4.2.12. Przedstawienie kodu funkcji Funkcja_do_oczekiwania. Źródło: Opracowanie własne

Funkcja_czekania (cooldown)

To macro opiera się na mechanizmie timera, który odlicza czas do osiągnięcia określonej wartości. Funkcja oferuje trzy wyjścia:
1. Po oczekiwaniu - sygnał po zakończeniu odliczania.
2. Początek oczekiwania - sygnał wysyłany na początku działania.
3. Odświeżanie czasu - sygnał aktualizujący stan w trakcie działania.
Dodatkowo, funkcja udostępnia zmienną Czas który pozostał, która może być używana do wizualnego wyświetlania odliczania. Macro to zostało zaprojektowane do elastycznego zarządzania przepływem czasu w systemie.

Ryc.4.2.11. Przedstawienie zewnętrzne, jako blok macro Funkcja_czekania (cooldown). Źródło: Opracowanie własne

Ryc.4.2.12. Przedstawienie kodu funkcji Funkcja_czekania (cooldown). Źródło: Opracowanie własne

Makra do pisania tekstu

Makra te umożliwiają dynamiczną edycję treści wyświetlanych w widgetach Base_widget_dialog. Ich głównym celem jest zarządzanie tekstem oraz obrazkami w interfejsie gry, zapewniając płynność narracji i łatwość obsługi. Każde macro zostało zaprojektowane w sposób umożliwiający różnorodne scenariusze ich wykorzystania, co czyni je niezbędnymi narzędziami w realizacji dynamicznych kampanii.

Elementy wspólne dla wszystkich makr pisania tekstu:
1. Wejście podstawowe
To główne wejście sygnałowe, które inicjalizuje działanie makra. Zapewnia ono prawidłową kolejność wykonywania operacji w systemie.
2. Reset
Pozwala na zresetowanie stanu funkcji w celu ponownego użycia. Jest to szczególnie przydatne w przypadku, gdy teksty mają być odtworzone wielokrotnie, np. w dialogach NPC. Jeśli sygnał resetu nie zostanie wysłany, macro pominie aktualnie zapisany tekst i przejdzie do kolejnej operacji.
3. In czas oczekiwania
Parametr umożliwiający ustawienie czasu, przez jaki gra będzie oczekiwać na kolejne działanie. Wartość domyślna 0.0 oznacza brak oczekiwania, a wartości ujemne wymuszają interakcję gracza przed kontynuacją, np. kliknięcie.
4. Tekst i Obrazek postaci
To podstawowe dane wejściowe. Tekst definiuje treść dialogu, a Obrazek postaci pozwala na wyświetlenie obrazu powiązanego z daną wypowiedzią. Oba elementy są kluczowe dla funkcji Pisanie, która odpowiada za dynamiczne wprowadzanie tych danych do interfejsu.

Przeznaczenie poszczególnych wariantów makr:
Macro_pisania_od_widgetu

Służy do edycji istniejących widgetów dialogowych. Jest przydatne w sytuacjach, gdy tekst lub obraz ma być dynamicznie zmieniony w już istniejącym elemencie interfejsu.

Ryc.4.2.13. Przedstawienie zewnętrzne, jako blok macro Macro_pisania_od_widgetu. Źródło: Opracowanie własne

Ryc.4.2.14. Przedstawienie kodu funkcji Macro_pisania_od_widgetu. Źródło: Opracowanie własne

Wejście
1. Widget in: Istniejący widget, który ma zostać edytowany.
Wyjścia
1. Wszystkie standardowe wyjścia związane z czasem (Po oczekiwaniu, Początek czekania, Odświeżanie czasu, Czas który pozostał).

Macro_pisania_ze_stworzeniem_widgetu
To macro tworzy nowy widget dialogowy przed wyświetleniem tekstu. Jest idealne w scenariuszach, gdy potrzebne są nowe elementy dialogowe, np. w przypadku rozpoczęcia nowej sceny.

Ryc.4.2.15. Przedstawienie zewnętrzne, jako blok macro Macro_pisania_ze_stworzeniem_widgetu. Źródło: Opracowanie własne

Ryc.4.2.16. Przedstawienie kodu funkcji Macro_pisania_ze_stworzeniem_widgetu. Źródło: Opracowanie własne

Wejścia
1. Klasa do stworzenia: Określa klasę widgetu.
2. Kontroler tekstu: Opcjonalnie przypisuje kontroler zarządzający tekstem.
Wyjścia
1. Widget tekstu: Zwraca utworzony widget.

Macro_pisania_ze_stworzeniem_widgetu_ze_skasowaniem_po_napisaniu
Umożliwia stworzenie nowego widgetu, wyświetlenie tekstu, a następnie jego usunięcie po zakończeniu działania.

Ryc.4.2.17. Przedstawienie zewnętrzne, jako blok macro Macro_pisania_ze_stworzeniem_widgetu_ze_skasowaniem_po_napisaniu. Źródło: Opracowanie własne

Ryc.4.2.18. Przedstawienie kodu funkcji Macro_pisania_ze_stworzeniem_widgetu_ze_skasowaniem_po_napisaniu. Źródło: Opracowanie własne
Wejścia
1. Wszystkie elementy z ze_stworzeniem_widgetu.
2. Kontroler podstawowy: Kontroluje proces usuwania widgetu.
Wyjścia
1. Standardowe wyjścia.

Macro_pisania_od_widgetu_ze_skasowaniem_po_napisaniu
To macro obsługuje istniejące widgety, ale usuwa je po zakończeniu działania. Jest to przydatne w przypadku elementów interfejsu, które mają być używane tylko tymczasowo.

Ryc.4.2.17. Przedstawienie zewnętrzne, jako blok macro Macro_pisania_od_widgetu_ze_skasowaniem_po_napisaniu. Źródło: Opracowanie własne

Ryc.4.2.18. Przedstawienie kodu funkcji Macro_pisania_od_widgetu_ze_skasowaniem_po_napisaniu. Źródło: Opracowanie własne
Wejścia
1. Widget in: Istniejący widget.
2. Kontroler podstawowy: Steruje procesem usuwania.
Wyjścia
1. Standardowe wyjścia.

Makra wyboru
Makra te odpowiadają za zarządzanie interakcjami gracza w formie wyborów dialogowych. Umożliwiają tworzenie, modyfikowanie i obsługę przycisków pochodzących z klasy Base_widget_wybór, co pozwala na dynamiczne sterowanie narracją.

Elementy wspólne dla wszystkich makr wyboru:
1. Wejście podstawowe
Punkt startowy dla działania makra, odpowiedzialny za inicjalizację całego procesu tworzenia i obsługi wyborów.
2. Reset

Resetuje stan przycisków, pozwalając na ponowne dokonanie wyboru. Jest to przydatne w scenariuszach, gdy decyzje gracza muszą być powtarzane lub zmieniane.
3. Przyciski_do_interakcji_podczas_wyboru
Zbiór przycisków, które gracz może wybierać w trakcie interakcji. Każdy przycisk odpowiada za jedną opcję dialogową.
4. Uruchomienie_konfiguracji_przycisków_po_wyborze
Pozwala na ustawienie zestawu przycisków, które pojawią się po dokonaniu wyboru. Ułatwia to tworzenie bardziej skomplikowanych sekwencji wyborów.
5. Event
Zdarzenie, które można przypisać do niestandardowych funkcji, umożliwiając dodatkową logikę na poziomie programistycznym.

Przeznaczenie poszczególnych wariantów makr:
Macro_wyboru_input_button_to_macro
Obsługuje istniejące przyciski, przypisując im odpowiednie akcje. Jest to najbardziej podstawowy wariant, używany w sytuacjach, gdy przyciski zostały wcześniej stworzone.

Ryc.4.2.19. Przedstawienie zewnętrzne, jako blok macro Macro_wyboru_input_button_to_macro . Źródło: Opracowanie własne

Ryc.4.2.20. Przedstawienie kodu funkcji Macro_wyboru_input_button_to_macro. Źródło: Opracowanie własne

Wejścia
1. Wejscie_W_czescniej_zaimplementowanych_przycisków: Lista istniejących przycisków.
Wyjścia
1. wyj_Wybór_exec: Wyjście dla kontynuacji działania.
2. wyj_Wartość_wyboru: Zmienna wynikająca z wyboru gracza.

Macro_wyboru_create_with_out_class
Tworzy nowe przyciski wyboru na podstawie klasy Base_widget_wybór.

Ryc.4.2.21. Przedstawienie zewnętrzne, jako blok macro Macro_wyboru_create_with_out_class. Źródło: Opracowanie własne

Ryc.4.2.22. Przedstawienie kodu funkcji Macro_wyboru_create_with_out_class. Źródło: Opracowanie własne

Wejścia
1. Clasa_base_wyboru: Klasa widgetu do utworzenia.
2. Pisanie_tekstu: Tekst przypisywany do przycisku.
Wyjścia
1. OutputPin: Identyfikator utworzonego przycisku.

Macro_wyboru_input_button_to_macro_with_cooldown
Dodaje mechanizm cooldown do obsługi istniejących przycisków, zapobiegając natychmiastowemu ponownemu użyciu.

Ryc.4.2.23. Przedstawienie zewnętrzne, jako blok macro Macro_wyboru_input_button_to_macro_with_cooldown. Źródło: Opracowanie własne

Ryc.4.2.24. Przedstawienie kodu funkcji Macro_wyboru_input_button_to_macro_with_cooldown. Źródło: Opracowanie własne

Wejścia
1. Wszystkie z input_button_to_macro.
2. in_czas_oczekiwania: Czas cooldownu.
Wyjścia
1. Wszystkie standardowe wyjścia z dodatkiem cooldownu.

Macro_wyboru_create_with_out_class_with_cooldown
Łączy tworzenie nowych przycisków z mechanizmem cooldown.

Ryc.4.2.25. Przedstawienie zewnętrzne, jako blok macro Macro_wyboru_create_with_out_class_with_cooldown. Źródło: Opracowanie własne

Ryc.4.2.26. Przedstawienie kodu funkcji Macro_wyboru_create_with_out_class_with_cooldown. Źródło: Opracowanie własne
Wejścia
1. Wszystkie z create_with_out_class.
2. Odgórnie_wybrany_przycisk: Przydziela konkretny przycisk po upływie cooldownu.
Wyjścia
1. Wszystkie standardowe wyjścia.
Rozdział 5 Prezentacja wyników testów
5.1 Plan testów jednostkowych

Testy jednostkowe przeprowadzono w celu oceny poprawności działania funkcji narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych. Testy miały charakter iteracyjny i były prowadzone głównie przez autora podczas implementacji poszczególnych funkcji. Dodatkowo przeprowadzono test z udziałem osoby trzeciej, której zadaniem było ocenić intuicyjność narzędzia oraz jego praktyczne zastosowanie.

Metodologia testów

Proces testowania obejmował trzy główne etapy:

1. Tworzenie i testowanie pojedynczych funkcji

Funkcja pisania została zrealizowana jako pierwsza i stanowiła podstawę dla kolejnych elementów, takich jak makra do pisania z różnymi rozszerzeniami. Podobnie w przypadku funkcji wyboru - najpierw stworzono podstawowy mechanizm przypisywania akcji do przycisków, a następnie dodano obsługę cooldownu oraz dynamicznego tworzenia widgetów.

2. Iteracyjne testowanie funkcji po ich implementacji

Każda nowa funkcjonalność była testowana pod kątem poprawności działania. Błędy identyfikowane podczas testów były eliminowane w kolejnych iteracjach lub pod koniec projektu w ramach pomniejszych testów. Stare funkcje i makra były często zmieniane podczas implementacji nowych, ponieważ ich wcześniejsze wersje okazywały się niewystarczające dla potrzeb projektu.

3. Test zewnętrzny

Narzędzie zostało udostępnione jednej osobie trzeciej, która nie miała wcześniejszej styczności z jego konstrukcją. Tester korzystał z narzędzia zgodnie z przygotowaną instrukcją i wypełnił ankietę, oceniając intuicyjność oraz użyteczność systemu.

Wyniki testów

Testy wewnętrzne wykazały poprawność większości funkcji, jednak wskazały również na obszary wymagające dopracowania:

2. W trakcie testów zauważono, że funkcje cooldown w pewnych warunkach nie działały poprawnie, co wymagało wprowadzenia dodatkowych warunków zabezpieczających.

3. Integracja makr wyboru z dynamicznym tworzeniem widgetów początkowo była problematyczna, jednak została usprawniona w kolejnych iteracjach.

Przed przystąpieniem do testów zewnętrznych przygotowano instrukcję obsługi narzędzia. Zawierała ona szczegółowy opis funkcji i makr, w tym sposoby ich użycia oraz przykłady zastosowań. Całość napisana była w sposób przystępny, unikając zbyt technicznego języka, co miało ułatwić zrozumienie narzędzia przez osoby trzecie.

Wyniki testu zewnętrznego były bardziej krytyczne:

1. Tester wskazał na brak intuicyjności narzędzia, co utrudniało jego obsługę dla osoby bez wcześniejszego doświadczenia z systemem.

2. Zauważono niedostateczną szczegółowość dokumentacji, która nie wyjaśniała w pełni działania niektórych funkcji i mechanizmów.

3. Pozytywnie oceniono jednak elastyczność narzędzia i jego zdolność do dynamicznego tworzenia interfejsów, co pozwala na różnorodne zastosowania w grach.

Analiza wyników

Mimo że testy jednostkowe potwierdziły techniczną poprawność narzędzia, brak intuicyjności oraz niedostateczna dokumentacja pozostały istotnymi wyzwaniami. Po przeprowadzeniu testów wprowadzono jedynie niewielkie poprawki w nazewnictwie funkcji, a sama instrukcja nie została zaktualizowana. Narzędzie w obecnym stanie jest funkcjonalne, jednak wymaga dalszych prac nad poprawą użyteczności i udokumentowaniem jego działania, co powinno być uwzględnione w przyszłych iteracjach projektu.

5.2 Testy integracyjne

W celu przeprowadzenia testów integracyjnych narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych stworzono grę testową.

Platforma ta umożliwiła kompleksową weryfikację funkcjonalności narzędzia w realistycznym środowisku, pozwalając na ocenę poprawności działania poszczególnych funkcji oraz ich wpływu na immersję gracza.

Podczas projektowania gry testowej kluczowe było dobranie odpowiedniego formatu, który nie tylko pozwalałby na szczegółowe przetestowanie możliwości narzędzia, ale także ukazywałby jego potencjalne zastosowanie w praktyce. W tym celu zdecydowano się na odwzorowanie mechanik oraz narracyjnych rozwiązań inspirowanych grą "The Stanley Parable".

Gra "The Stanley Parable" wyróżnia się unikalnym podejściem do narracji oraz interakcji z graczem. Rozgrywka polega na poruszaniu się postacią w środowisku biurowym, podczas gdy narrator na bieżąco komentuje wybory gracza oraz jego działania. Pomimo ograniczonych możliwości interakcji z otoczeniem, każda decyzja, taka jak wybór jednych z dwóch drzwi, prowadzi do nowych gałęzi fabularnych. Narrator dynamicznie reaguje na decyzje gracza, co tworzy wrażenie ciągłego wpływu na rozwój historii.

Mechanika gry opiera się na prostych, ale znaczących wyborach, takich jak skręcenie w lewo lub w prawo, wchodzenie po schodach, korzystanie z windy czy nawet zabranie ze sobą wiadra. Każda z tych decyzji prowadzi do innego zakończenia. Taka struktura fabularna sprawiła, że "The Stanley Parable" idealnie nadaje się jako inspiracja dla projektu testowego.

W testowej grze odwzorowano te mechaniki, jednocześnie wprowadzając zmiany dostosowane do celów testowania narzędzia. Gracz wciela się w postać Marioli, pracowniczki biurowej, która, podobnie jak Stanley, porusza się w ograniczonej przestrzeni biura. Zamiast wyborów dokonywanych poprzez ruch postaci, gracz podejmuje decyzje za pomocą interfejsu wyboru. Mechanizm ten pozwala na lepsze zaprezentowanie funkcji narzędzia oraz jego zastosowań w tworzeniu dynamicznych narracji.

Dodatkowo wprowadzono większe możliwości interakcji z otoczeniem. Postać może kliknąć na obiekt lub miejsce, co powoduje reakcję narratora w formie komentarza. Funkcjonalność ta została zrealizowana dzięki zastosowaniu makr pisania tekstu, które umożliwiają zsynchronizowanie wyświetlania tekstu z narracją dźwiękową. Dzięki rozbudowanym opcjom wyjść makr pisania, narrator może rozpocząć swoją wypowiedź jeszcze przed wyświetleniem tekstu na ekranie, co znacząco zwiększa immersję i spójność narracyjną gry.

5.3 Analiza wyników testów