**Akademia Finansów i Biznesu Vistula**

**Wydział Informatyki, Grafiki i Architektury**

**Kierunek studiów Informatyka**

**Mikołaj Łapiński**

Numer albumu: 59357

**PROJEKT I IMPLEMENTACJA NARZĘDZIA WSPOMAGAJĄCEGO TWORZENIE DYNAMICZNYCH KAMPANII W GRACH KOMPUTEROWYCH PRZY UŻYCIU UNREAL ENGINE Z ZASTOSOWANIEM METODYKI SDRUM W REALIZACJI PROJEKTU**

Praca inżynierska

napisana pod kierunkiem

Promotor – mgr. Jacek Mochyła

Warszawa, 2025

**Spis treści**

[Wstęp 3](#_Toc187869796)

[Rozdział 1 Analiza istniejących rozwiązań 5](#_Toc187869797)

[1.1. Charakterystyka istniejących narzędzi do tworzenia dynamicznych kampanii w grach komputerowych 5](#_Toc187869798)

[1.2. Słabe strony analizowanych rozwiązań 5](#_Toc187869799)

[1.3. Propozycje unikania słabych stron w projekcie 6](#_Toc187869800)

[Rozdział 2 Charakterystyka narzędzi technologicznych 8](#_Toc187869801)

[2.1. Narzędzia zastosowane w pracy 8](#_Toc187869802)

[Rozdział 3 Założenia projektowe i architektura projektu 10](#_Toc187869803)

[3.1. Wymagania projektowe narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych 10](#_Toc187869804)

[3.2. Architektura projektu narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych 10](#_Toc187869805)

[Rozdział 4 Implementacja projektu 12](#_Toc187869806)

[4.1. Implementacja poszczególnych komponentów narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych 12](#_Toc187869807)

[4.2. Wyjaśnienie kluczowych fragmentów kodu 18](#_Toc187869808)

[Rozdział 5 Prezentacja wyników testów 39](#_Toc187869809)

[5.1. Plan testów jednostkowych 39](#_Toc187869810)

[5.2. Testy integracyjne 40](#_Toc187869811)

[5.3. Wnioski z testów 42](#_Toc187869812)

[6. Podsumowanie 43](#_Toc187869813)

[Bibliografia 44](#_Toc187869814)

# Wstęp

Dynamiczne kampanie w grach komputerowych stały się istotnym elementem współczesnej branży gier wideo, umożliwiając graczom większą swobodę w podejmowaniu decyzji oraz wpływ na rozwój fabuły. W odróżnieniu od tradycyjnych, liniowych scenariuszy, dynamiczne kampanie oferują nieliniową narrację. Decyzje podejmowane przez gracza wpływają więc na dalszy przebieg gry. Taka struktura fabularna zwiększa immersję, co sprawia, że gracz czuje się częścią świata przedstawionego w grze.

Jednym z wyzwań przy tworzeniu dynamicznych kampanii jest efektywne zarządzanie interakcjami gracza oraz implementacja narzędzi wspomagających budowę nieliniowej narracji. Wymaga to zastosowania odpowiednich funkcji, takich jak dynamiczne pisanie tekstów, elastyczne systemy wyborów i synchronizacja narracji z innymi elementami gry, takimi jak animacja czy dźwięk. Złożoność takich systemów sprawia, że projektowanie i implementacja dynamicznych kampanii są czasochłonne i wymagają użycia zaawansowanych narzędzi deweloperskich.

W ramach niniejszej pracy inżynierskiej stworzono funkcje wspomagające tworzenie dynamicznych kampanii w grach komputerowych. Funkcje zaprojektowano z myślą o ułatwieniu pracy twórcom gier poprzez uproszczenie procesu implementacji narracji i zwiększenia elastyczności w dostosowywaniu mechanik gry do potrzeb konkretnego projektu. Autor opracował rozwiązania, które nie tylko poprawiają efektywność pracy, ale również podnoszą jakość narracji. Oferuje to graczom bogatsze i bardziej angażujące doświadczenia.

W niniejszej pracy zastosowano funkcje dynamicznej narracji, które pozwalają na elastyczne wyświetlanie tekstu w grze. Problemem w takich rozwiązaniach jest zapewnienie tej elastyczności. Tradycyjne systemy wyświetlania tekstu w grach, czy to w formie dialogów, opowiadania historii, czy też opcji wyboru są często tworzone w sposób statyczny. Generuje to dodatkowe obciążenie projektu o dodatkową zawartość. Proponowane przez autora funkcje rozwiązują ten problem dzięki przemyślanej strukturze i elastycznym zmiennym oraz prostym w użyciu narzędziom. Pozwalają one programistom i projektantom skupić się na bardziej złożonych aspektach projektu.

Praca składa się z pięciu rozdziałów przedstawiających kolejne etapy realizacji projektu. W rozdziale pierwszym i drugim zaprezentowano analizę możliwych silników gier oraz istniejących rozwiązań wspierających dynamiczne kampanie. W rozdziale trzecim i czwartym przedstawiono projektowanie oraz implementację funkcji. W czwartym rozdziale szczegółowo opisano główne funkcje, poczynając od ich podstawowych składowych aż po pełne makra. W piątym rozdziale przedstawiono proces testowania i analizę wyników testów.

Układ pisemnej pracy został zoptymalizowany dzięki konsultacjom z narzędziem wspierającym redakcję treści. Pozwoliło to zachować spójność strukturalną tekstu. Proces pisania oraz merytoryczna zawartość pracy zostały ostatecznie ukształtowane bezpośrednio przez autora, który odpowiada za opracowanie koncepcji i jej realizację.

# Rozdział 1 Analiza istniejących rozwiązań

## 1.1. Charakterystyka istniejących narzędzi do tworzenia dynamicznych kampanii w grach komputerowych

Tworzenie dynamicznych kampanii w grach komputerowych wymaga odpowiednich narzędzi. Umożliwiają one zarządzanie interakcjami gracza, zmianami w rozgrywce oraz narracją. Na rynku dostępnych jest kilka takich narzędzi, które pozwalają na realizację tych zadań.

Popularnym narzędziem jest RPG Maker, stworzony z myślą o twórcach gier RPG. Umożliwia on korzystanie z gotowych szablonów i zasobów. Jest łatwy w obsłudze nawet dla osób bez doświadczenia w programowaniu. RPG Maker ma wbudowany system wydarzeń, co pozwala na tworzenie prostych narracji i interakcji. Brak zaawansowanych mechanizmów oraz elastyczności sprawia, że nadaje się on głównie do tworzenia gier w jednym określonym gatunku.

Innym rozwiązaniem jest Unity, wszechstronny silnik gier umożliwiający tworzenie dynamicznych kampanii poprzez skrypty w języku C#. Bogata dokumentacja i duże możliwości technologiczne sprawiły, że Unity stało się jednym z najpopularniejszych narzędzi w branży. W odróżnieniu od RPG Makera, Unity wymaga zaawansowanych umiejętności programistycznych. Tworzenie dynamicznych kampanii wymaga pisania wielu linii kodu i budowania systemów narracyjnych praktycznie od podstaw, a to znacząco wydłuża czas pracy.

W interaktywnych opowieściach specjalizuje się także Twine. Jest to narzędzie proste w obsłudze i intuicyjne, przeznaczone do projektów narracyjnych, w których tekst stanowi główny element rozgrywki. Twine umożliwia tworzenie interakcji oraz wyborów gracza. Ogranicza się jedynie do narracji tekstowej. Nie oferuje wsparcia dla zaawansowanych funkcji, jak efekty dźwiękowe, animacje czy złożona mechanika gry.

Ostatnim narzędziem jest Unreal Engine, jeden z najpotężniejszych silników gier, który dzięki systemowi Blueprintów pozwala na tworzenie zaawansowanych mechanik bez potrzeby pisania kodu w językach programowania. Unreal Engine umożliwia integrację dynamicznych tekstów z animacją i dźwiękiem. Brak dedykowanego systemu do zarządzania tekstami sprawia, iż proces ten wymaga dużej liczby manualnych kroków.

## 1.2. Słabe strony analizowanych rozwiązań

Każde z omówionych narzędzi ma swoje ograniczenia, które mogą wpłynąć na efektywność tworzenia dynamicznych kampanii.

W przypadku RPG Makera, główną wadą jest brak wsparcia dla bardziej zaawansowanych mechanik. Narzędzie zostało zaprojektowane z myślą o prostych grach RPG i praktycznie nie ma zastosowania 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, ograniczone jest do narracji tekstowej i nie oferuje narzędzi pozwalających na łatwą integrację z animacją i dźwiękiem.

W Unreal Engine brakuje wbudowanego systemu zarządzania sekwencjami tekstów oraz mechanizmów wyborów gracza. Tworzenie dynamicznych tekstów wymaga ręcznego budowania widgetów i integrowanie ich z logiką gry.

## 1.3. 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 to, w kontekście tworzenia dynamicznych kampanii nadal istnieją pewne ograniczenia, które mogą wpłynąć na proces deweloperski.

Brak wbudowanego, dedykowanego systemu do zarządzania dynamicznymi kampaniami sprawia, że twórcy muszą samodzielnie implementować mechanizmy narracyjne. Wymaga to tworzenia kompleksowych struktur w Blueprintach lub pisania kodu w języku C++. System Blueprintów ułatwia budowanie logiki gry, ale bardziej zaawansowane mechanizmy, jak rozbudowane drzewa dialogowe czy dynamiczne zmiany fabularne, najczęściej wymagają dodatkowej pracy i optymalizacji.

W Unreal Engine 5 proces tworzenia dynamicznych tekstów i wyborów gracza realizowany jest w wielu krokach. Twórcy muszą ręcznie tworzyć widgety, zarządzać wyświetlaniem tekstów oraz integrować je z pozostałymi elementami gry.

Zarządzanie dużymi ilościami danych, takimi jak sekwencje tekstów czy interakcje gracza, stanowi wyzwanie w przypadku projektów o większej skali. Twórcy muszą wówczas zaprojektować własne systemy przechowywania danych i zarządzania nimi.

Mimo technologii Nanite i Lumen, Unreal Engine 5 nadal wymaga dużej dbałości o optymalizację w przypadku gier o dużych otwartych światach lub złożonych kampaniach dynamicznych. W projektach wykorzystujących dynamiczne zmiany fabularne konieczna jest zatem dbałość o zarządzanie zasobami i utrzymanie płynności działania gry.

# Rozdział 2 Charakterystyka narzędzi technologicznych

## 2.1. Narzędzia zastosowane w pracy

Do stworzenia projektu użyto dwóch programów:

1. Unreal Engine 5 – został wybrany jako główny silnik gry, a Blueprinty pełniły kluczową rolę w implementacji logiki gry. Blueprinty jako wizualny system skryptowy, umożliwiły 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. Funkcje umożliwiły płynne wprowadzanie zmieniających się tekstów w grze i oferowanie graczowi opcji wyboru, co jest podstawą rozwoju narracji w dynamicznych kampaniach.

O wyborze Unreal Engine 5 zadecydowały następujące czynniki:

* Łatwa implementacja logiki gry w trakcie projektowania.
* Szybsze wyszukiwanie błędów i ich naprawa.
* Ułatwienie dla osób, które będą próbowały tworzyć własne funkcje bazujące na istniejącym kodzie.
* Dynamiczne generowanie tekstu i wybory gracza, umożliwiające tworzenie rozbudowanych scenariuszy i kampanii.

1. 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, otwartoźródłowym narzędziem do modelowania 3D, które umożliwia tworzenie wysokiej jakości zasobów wizualnych, w tym renderowanie, symulację fizyczną oraz teksturowanie.

W projekcie wykorzystano następujące funkcje Blendera:

* Modelowanie 3D umożliwiające tworzenie postaci, obiektów i środowisk, które zostały zaimportowane do Unreal Engine 5.
* Teksturowanie do tworzenia tekstur i map, które nadały realizmu postaciom i obiektom 3D, co miało znaczenie dla estetyki projektu.
* Eksport modeli, który umożliwił płynne przenoszenie zasobów 3D między Blenderem a Unreal Engine 5.

Oba programy były bardzo pomocne w trakcie pracy, jednak eksport pomiędzy Blenderem a Unreal Engine 5 wiązał się z pewnymi trudnościami. Różnice w systemach materiałów obu programów narzuciły konieczność rekonstrukcji materiałów w Unreal Engine. Tekstury, takie jak wektor normalny czy albedo, były poprawnie przenoszone, jednak niestandardowe shadery wymagały ponownego stworzenia w Unreal Engine. Animacje musiały zostać zaimplementowane na nowo w Unreal Engine, ponieważ oba silniki korzystają z różnych systemów animacji materiałów. Biorąc pod uwagę, że różnice w jednostkach miary mogły powodować problemy z wielkością obiektów po imporcie, często konieczne było 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 opierały się na potrzebie przyspieszenia procesu implementacji dynamicznych tekstów i wyborów w grze, a także na ułatwieniu integracji z istniejącymi wersjami Unreal Engine. Narzędzie powinno umożliwiać szybkie tworzenie i modyfikowanie treści tekstowych bez potrzeby zaawansowanego kodowania. Ma to znaczenie w dynamicznych kampaniach, gdzie treści zmieniają się w zależności od wyborów dokonywanych przez gracza.

Kluczowym wymaganiem było zapewnienie możliwości sterowania akcjami i zdarzeniami w trakcie wyświetlania tekstu lub po jego zakończeniu. Miało to na celu wzbogacenie rozgrywki o dodatkowe interakcje. Istotnym elementem projektu było także uwzględnienie kompatybilności z systemem Blueprintów, co pozwoliło na łatwe wdrażanie i rozbudowę narzędzia w środowisku Unreal Engine.

Ponadto narzędzie musiało 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 było także zoptymalizowanie narzędzia pod kątem Unreal Engine 5.3, co zapewniło pełne wykorzystanie dostępnych funkcji, takich jak mechanizmy Blueprintów i wbudowane timery.

## 3.2. Architektura projektu narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych

Architekturę narzędzia zaprojektowano w sposób modułowy, co ułatwiło jej integrację z różnymi aspektami rozgrywki. Centralnym elementem narzędzia był widget zarządzający wszystkimi funkcjami związanymi z dynamicznymi tekstami. Widget pełnił 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

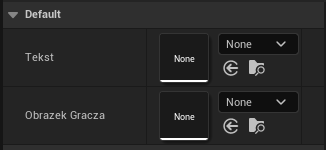
## 4.1. Implementacja poszczególnych komponentów narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych

W niniejszej pracy pakiet funkcji pisania tekstu zaprojektowano jako modularny zestaw funkcji i komponentów. Współpraca poszczególnych funkcji usprawniła proces implementacji pod dynamiczną narrację. Poszczególne elementy systemu mogły integrować się z innymi funkcjami Unreal Engine 5, co umożliwiło to tworzenie złożonych interakcji przy niewielkim wysiłku programistycznym.

Widget był odpowiedzialny za obsługę interfejsu użytkownika, menedżera tekstów zarządzającego logiką narracji oraz funkcje wyborów gracza, które zapewniły interaktywność. Obsługiwał wyświetlanie tekstów w dynamiczny sposób. Umożliwiło to zastosowanie efektów wizualnych w postaci pisania tekstu na ekranie. Menedżer tekstów zaimplementowano jako Actor Component. Posłużył on do przechowywania i zarządzania sekwencjami tekstów w formie stringów. Umożliwiło to kontrolowanie wyświetlanych treści. Ponadto, pozwoliło na przypisywanie do nich odtwarzania dźwięków, zmiany stanu gry i uruchamiania animacji. Funkcja wyborów gracza pozwoliła stworzyć opcje interakcji, spośród których gracz może dokonać wyboru.

W dalszej części pracy opisano szczegółowo implementację każdego z tych elementów, uwzględniając ich integracje oraz zastosowanie w dynamicznych kampaniach narracyjnych.

Widget do pisania tekstu, jak zostało wspomniane, stanowił główny element projektu. Jego konfiguracja była prosta i nie przysporzyła żadnych problemów. Na poniższej ilustracji (Ryc. 4.1.1) przedstawiono strukturę zmiennych w Base\_widget\_dialog. Każda ze zmiennych odpowiadała za dany element interfejsu użytkownika, czyli wyświetlany tekst lub obrazek postaci. Takie podejście umożliwiło dynamiczne przypisywanie wartości w trakcie gry, co ułatwiło konfigurację i modyfikację dialogów.

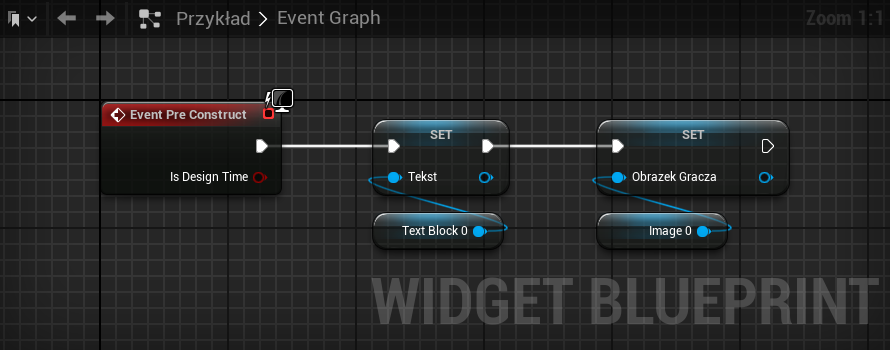


**Ryc. 4.1.1. Wartości w Base\_widget\_dialog**

Źródło: Opracowanie własne.

Do zmiennej „Tekst” przypisuje się wartość pola tekstowego, które ma być edytowane. Natomiast do zmiennej „Obrazek Gracza” przypisuje się obrazek, który ma być edytowany przez funkcje.

Dzięki zastosowaniu takich rozwiązań, dołączone funkcje w menadżerze tekstu miały stałą implementację tego, jakie pole tekstowe i obrazek były przez nie edytowane. Dany widget stał się 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 przedstawiono na ryc. 4.1.2.



**Ryc. 4.1.2. Logika przypisania tekstu**

Źródło: Opracowanie własne.

Na powyższej rycinie przedstawiono 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. Podstawowym układ takiego widgetu przedstawiono na ryc. 4.1.3.

Obraz zawierający zrzut ekranu, Prostokąt, kwadrat

Opis wygenerowany automatycznie

**Ryc. 4.1.3. Podstawowy układ widgetu odpowiedzialnego za pisanie tekstu**

Źródło: Opracowanie własne

Elastyczność układu wizualnego umożliwia projektantowi dowolne rozmieszczenie komponentów oraz dostosowanie ich parametrów wizualnych do potrzeb projektu. Można w ten sposób dopasować ich rozmiar, czcionkę czy kolory.

Kolejną funkcjonalnością jest Menadżer Tekstu (Base\_Actor\_Component). Jest to centralny komponent zarządzający dynamicznymi sekwencjami tekstów w grze. Odpowiada za przechowywanie, obsługę oraz synchronizację tekstów wyświetlanych w interfejsie użytkownika. Modularna konstrukcja umożliwia integrację z animacjami w grze, efektami dźwiękowymi oraz mechanizmami wyborów gracza. Menedżer pozwala również na wyzwalanie akcji po każdym zakończonym działaniu przypisanej funkcji, dzięki czemu można tworzyć bardziej złożone scenariusze narracyjne.

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. Każdy tekst stanowi główną linię (pień), a dodatkowe akcje mogą odgałęziać się jako pomniejsze procesy (gałęzie). Funkcje wyboru, opisane w dalszej części pracy, pozwalają na tworzenie bardziej zaawansowanych drzew narracyjnych z wieloma możliwościami interakcji dla gracza.

Menadżer Tekstu zawiera funkcje odpowiadające za pisanie tekstu, które zostały 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 – 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 – 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 – 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 – 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.

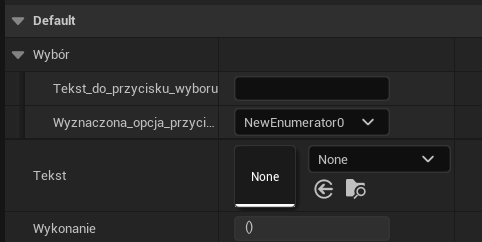
Kolejną funkcjonalnością narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych jest funkcja wyboru, która umożliwia graczowi podejmowanie decyzji w trakcie rozgrywki. Została ona zaprojektowana jako integralna część Menedżera Tekstu i opiera się na specjalnie przygotowanych widgetach 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. Dzięki temu łatwo można skonfigurować liczbę opcji wyboru, ich treść oraz akcje, które są wywoływane w zależności od decyzji gracza. Tego rozwiązania można używać 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 tworzy się 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 – 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 – 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 – połączenie funkcjonalności dynamicznego tworzenia przycisków i mechanizmu cooldown. Przyciski są generowane w czasie rzeczywistym i mogą być czasowo blokowane po użyciu.

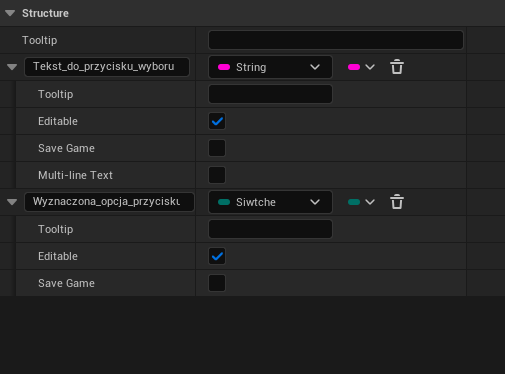
Bardziej szczegółowy opis działania poszczególnych makr zostanie przedstawiony w kolejnym podrozdziale. W tej sekcji zaprezentowano ich ogólny podział i zastosowanie, aby podkreślić elastyczność i wszechstronność Menedżera Tekstu.



**Ryc. 4.1.3. Wartości pojedynczego przycisku wyboru**

Źródło: Opracowanie własne.

Na rycinie 4.1.3 przedstawiono szczegółowe ustawienia widgetu wyboru (Base\_widget\_wyboru), który pełni kluczową rolę w funkcji wyboru. Wartości i właściwości tego widgetu zaprojektowano w sposób umożliwiający elastyczną konfigurację dla różnych scenariuszy w grze.



**Ryc. 4.1.4. Struktura wyboru**

Źródło: Opracowanie własne.

Widget korzysta ze struktury wyboru przedstawionej na ryc. 4.1.4. Struktura ta pozwala na przypisanie konkretnych wartości każdemu przyciskowi wyboru. W jej skład wchodzą m.in.:

* Tekst\_do\_przycisku\_wyboru – pole tekstowe typu String, które definiuje treść wyświetlaną na danym przycisku wyboru.
* 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.

Widget zawiera także 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 zaprojektowano 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.

## 4.2. Wyjaśnienie kluczowych fragmentów kodu

Projekt narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych opiera się na funkcjach i makrach zaprojektowanych w celu zapewnienia elastyczności, wydajności oraz łatwości integracji z istniejącymi wersjami Unreal Engine 5. W niniejszym podrozdziale szczegółowo omówiono kluczowe elementy kodu, które odgrywają centralną rolę w realizacji funkcjonalności narzędzia.

Omówiono ogólne elementy, takie jak funkcje wspierające makra i ich logikę, a następnie bardziej szczegółowe mechanizmy, które odpowiadają za konkretne aspekty działania systemu. Wśród opisanych fragmentów znalazły 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 ilustruje, jak poszczególne komponenty współpracują w celu zapewnienia kompleksowego wsparcia dla dynamicznych kampanii.

**Universalna\_funkcja**

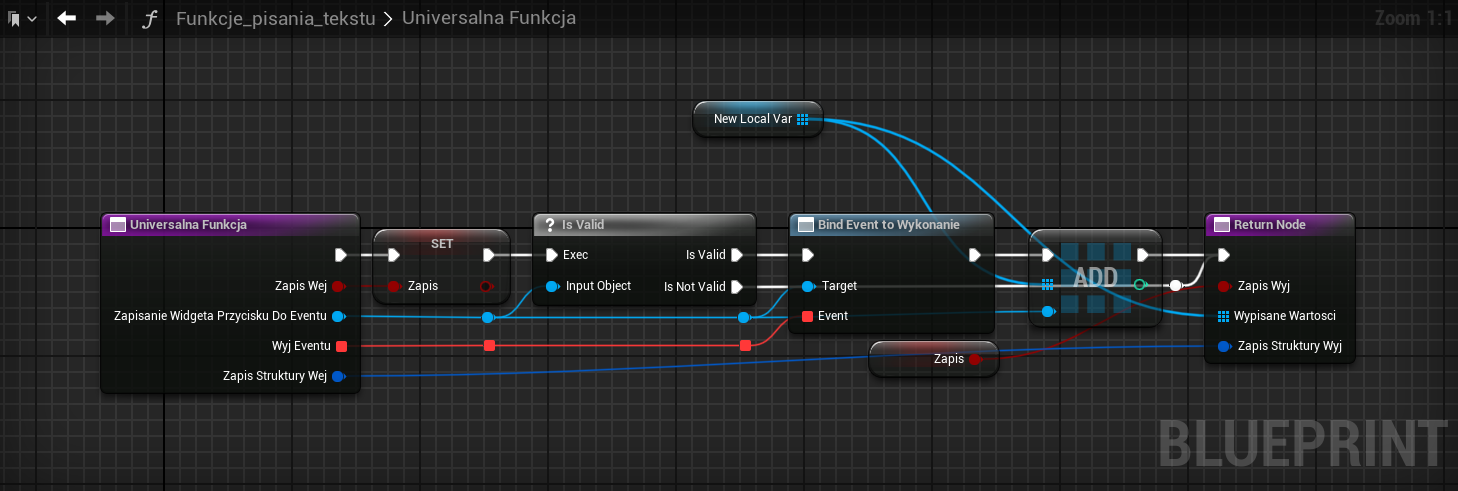
Funkcja ta pełni rolę pomocniczą dla makr, stanowiąc bank pamięci dla poszczególnych kopii makr. Podczas testów zauważono, że każda instancja makra przechowuje swoją pamięć niezależnie, co rozwiązało problem związany z nadpisywaniem danych. Funkcja jest używana głównie do zapisywania wartości typu bool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

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

**Ryc. 4.2.1. Przedstawienie zewnętrzne, jako blok funkcji Universalna\_funkcja**

Źródło: Opracowanie własne.

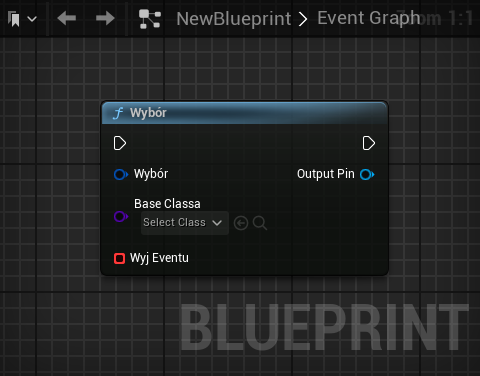


**Ryc. 4.2.2. Przedstawienie kodu funkcji Universalna\_funkcja**

Źródło: Opracowanie własne

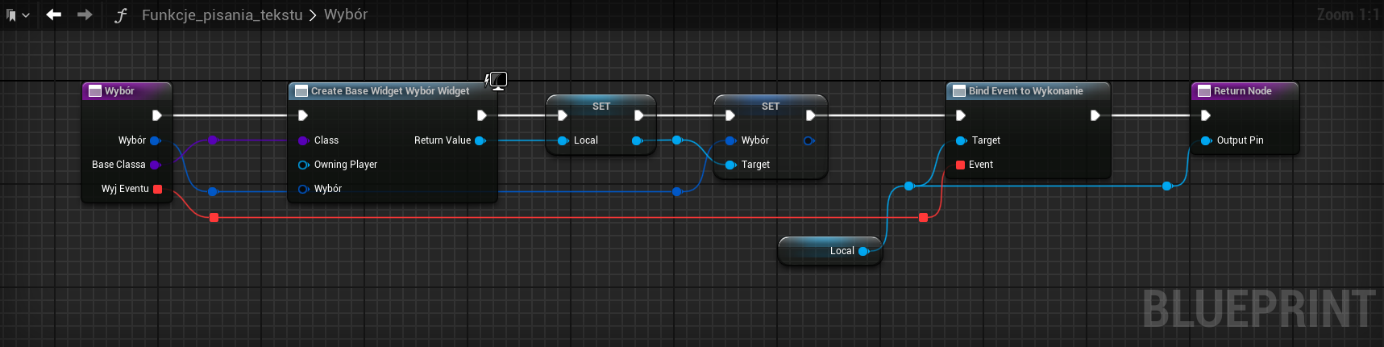
**Wybór**

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

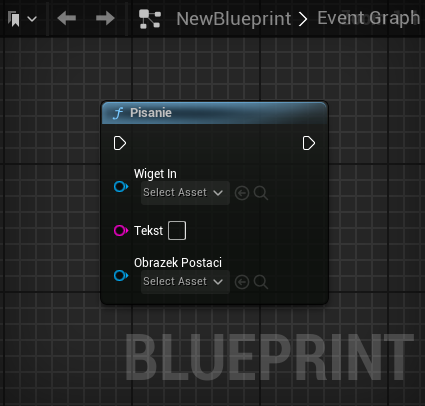


**Ryc. 4.2.4. Przedstawienie kodu funkcji Wybór**

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.

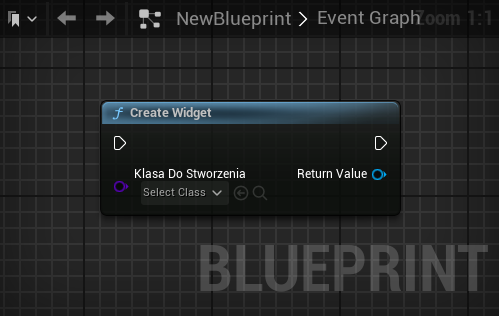


**Ryc. 4.2.6. Przedstawienie kodu funkcji Pisanie**

Źródło: Opracowanie własne.

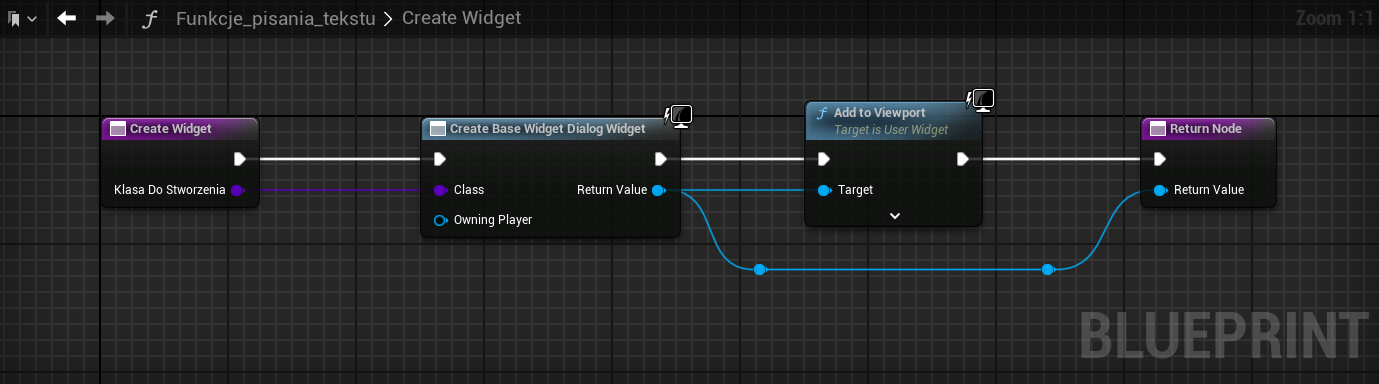
**Create\_widget**

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.

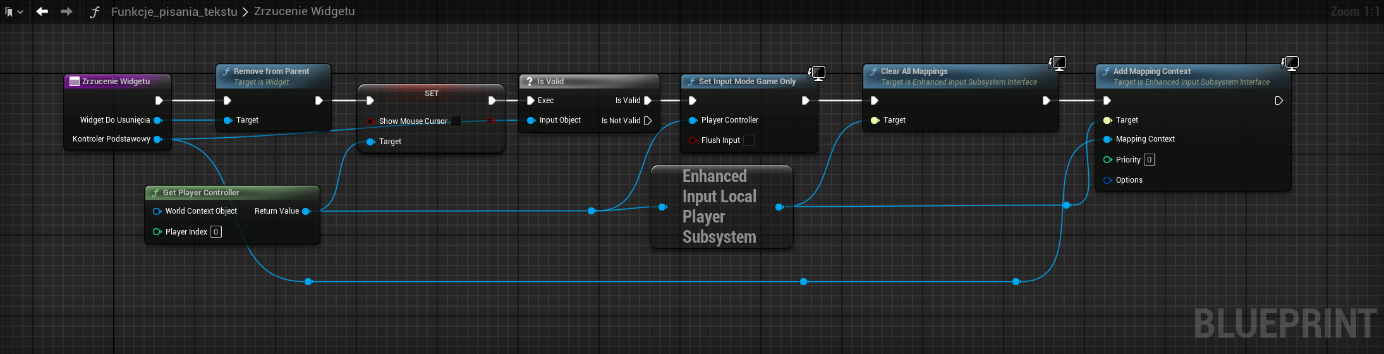
**Zrzucenie\_widgetu**

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.

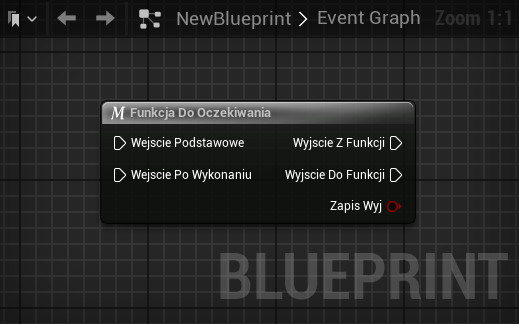


**Ryc. 4.2.10. Przedstawienie kodu funkcji Zrzucenie\_widgetu**

Źródło: Opracowanie własne.

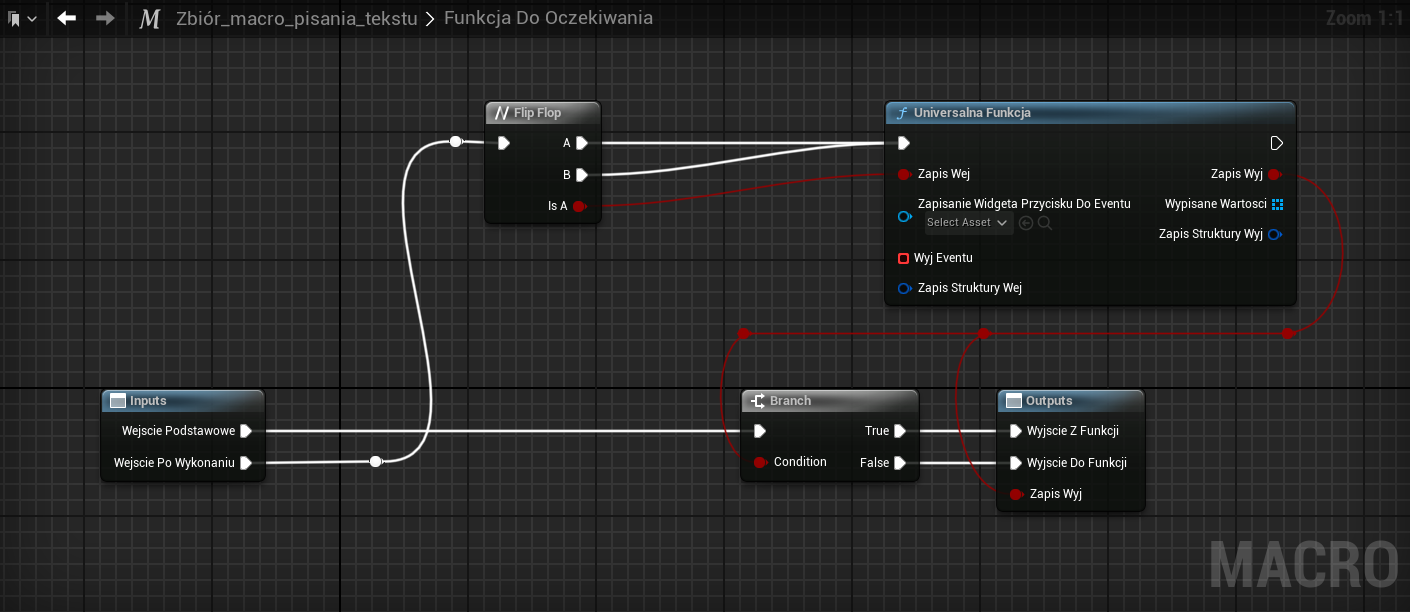
**Funkcja\_do\_oczekiwania**

Makro 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**

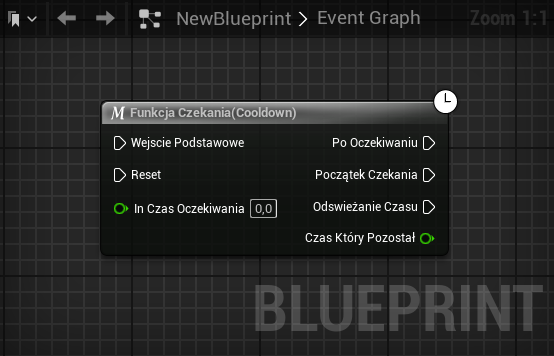
Źródło: Opracowanie własne.

**Funkcja\_czekania (cooldown)**

To makro opiera się na mechanizmie timera, który odlicza czas do osiągnięcia określonej wartości. Funkcja oferuje trzy wyjścia:

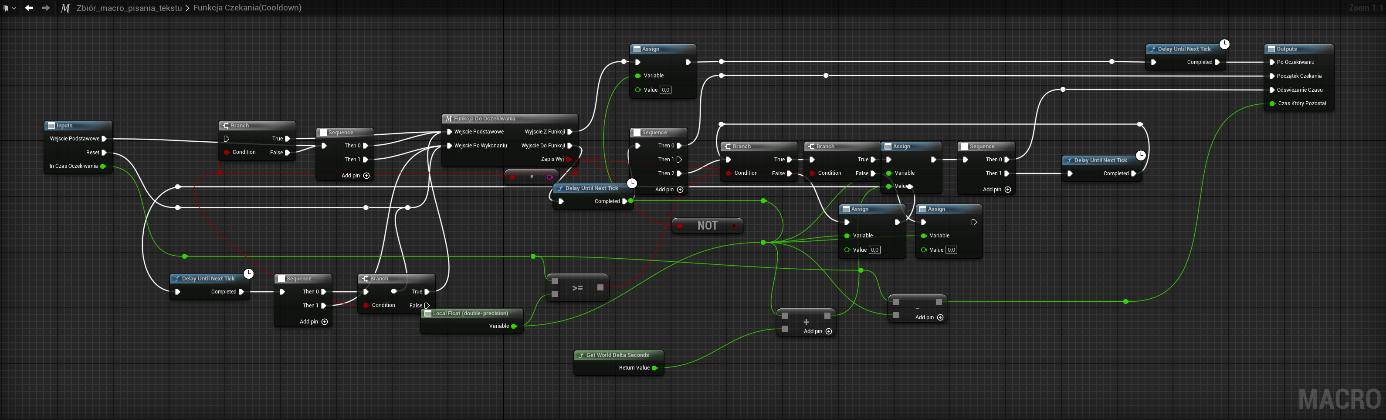
* Po oczekiwaniu – sygnał po zakończeniu odliczania.
* Początek oczekiwania – sygnał wysyłany na początku działania.
* Odświeżanie czasu – sygnał aktualizujący stan w trakcie działania.

Dodatkowo, funkcja udostępnia zmienną „Czas który pozostał”, która może być używana do wizualnego wyświetlania odliczania. Makro to zostało zaprojektowane do elastycznego zarządzania przepływem czasu w systemie.



**Ryc. 4.2.11. Przedstawienie zewnętrzne jako blok makro 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 makro zaprojektowano w sposób umożliwiający różnorodne scenariusze ich wykorzystania, co czyni je niezbędnymi narzędziami w realizacji dynamicznych kampanii.

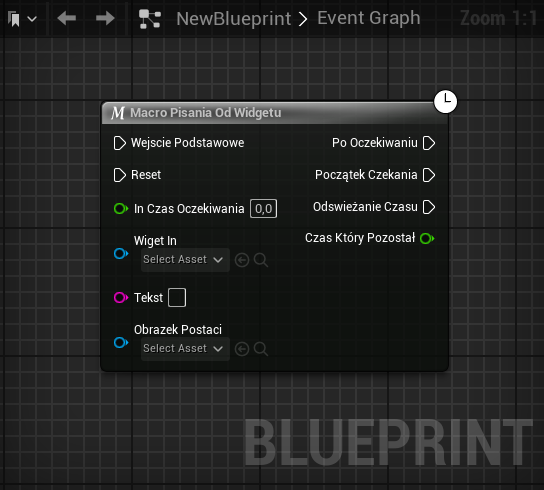
Elementy wspólne dla wszystkich makr pisania tekstu:

1. Wejście podstawowe – główne wejście sygnałowe, które inicjalizuje działanie makra. Zapewnia prawidłową kolejność wykonywania operacji w systemie.
2. Reset – pozwala na zresetowanie stanu funkcji w celu ponownego użycia. Jest to przydatne w przypadku, gdy teksty mają być odtworzone wielokrotnie, np. w dialogach NPC. Jeśli sygnał resetu nie zostanie wysłany, makro 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 – 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.

Na rycinie 4.2.13 przedstawiono wygląd zewnętrzny funkcji, w której wejścia i wyjścia zostały opisane następująco:

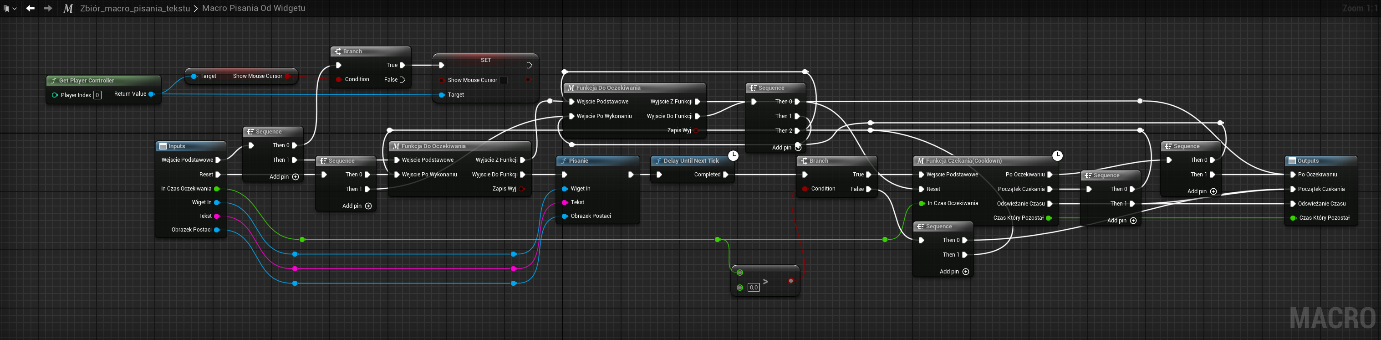
Wejścia

* Widget in – istniejący widget, który ma zostać edytowany.

Wyjścia

* Wszystkie standardowe wyjścia związane z czasem (Po oczekiwaniu, Początek czekania, Odświeżanie czasu, Czas który pozostał).

Natomiast na rycinie 4.2.14 przedstawiono to, jak funkcja wygląda wewnątrz.

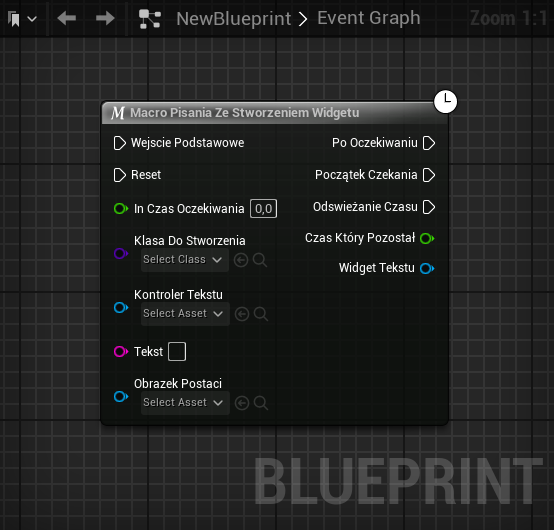


**Ryc. 4.2.14. Przedstawienie kodu funkcji Macro\_pisania\_od\_widgetu**

Źródło: Opracowanie własne.

**Macro\_pisania\_ze\_stworzeniem\_widgetu**

To makro 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 makro Macro\_pisania\_ze\_stworzeniem\_widgetu**

Źródło: Opracowanie własne.

Na rycinie 4.2.15 przedstawiono wygląd kodu funkcji, w której wejścia i wyjścia opisano w następujący sposób:

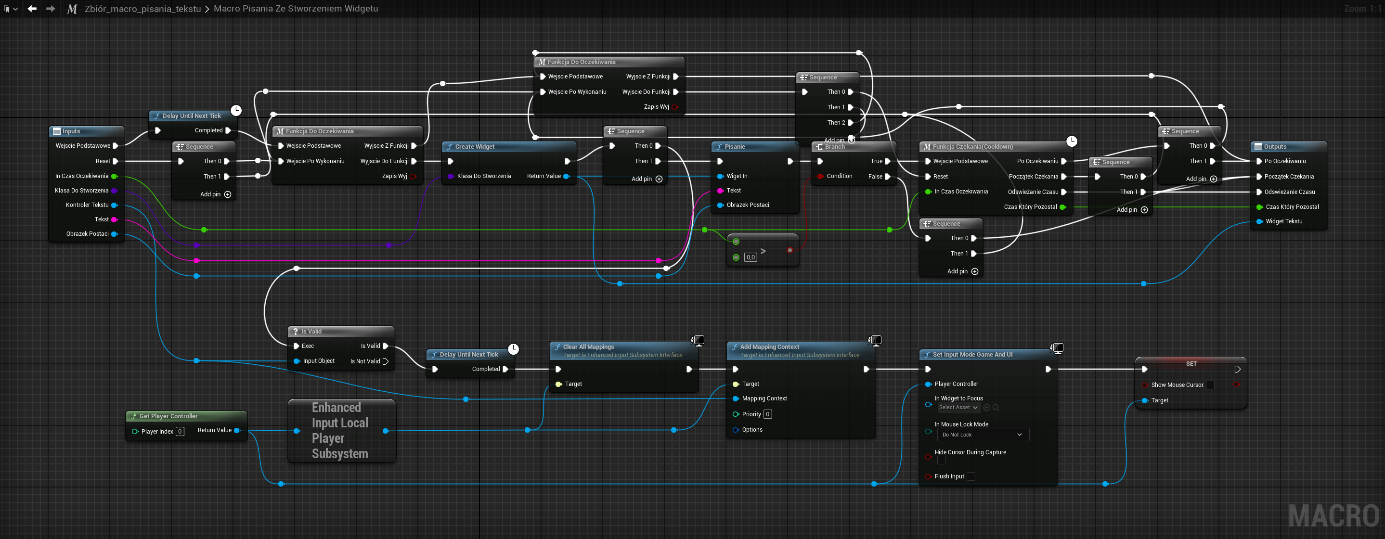
Wejścia

* Klasa do stworzenia – określa klasę widgetu.
* Kontroler tekstu – opcjonalnie przypisuje kontroler zarządzający tekstem.

Wyjścia

* Widget tekstu – zwraca utworzony widget.

Natomiast na rycinie 4.2.16 zaprezentowano wygląd wewnętrzny kodu funkcji.

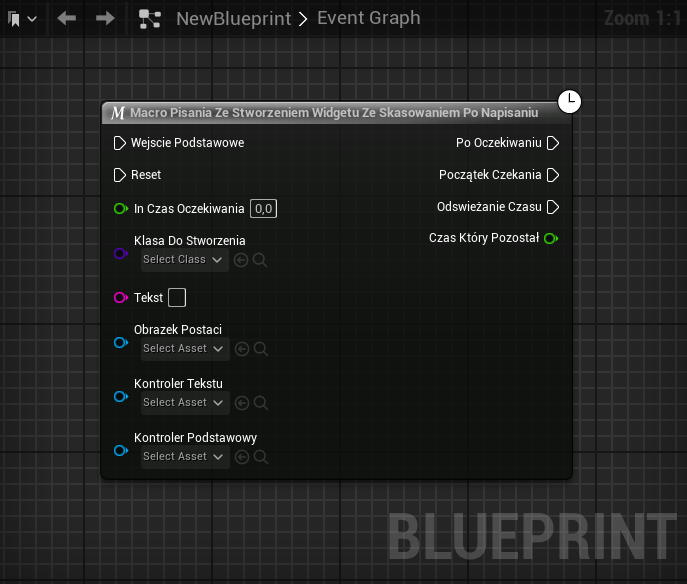


**Ryc. 4.2.16. Przedstawienie kodu funkcji Macro\_pisania\_ze\_stworzeniem\_widgetu**

Źródło: Opracowanie własne.

**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.

Na rycinie 4.2.17 przedstawiono wygląd zewnętrzny sekwencji, w której wejścia i wyjścia opisano w następujący sposób:

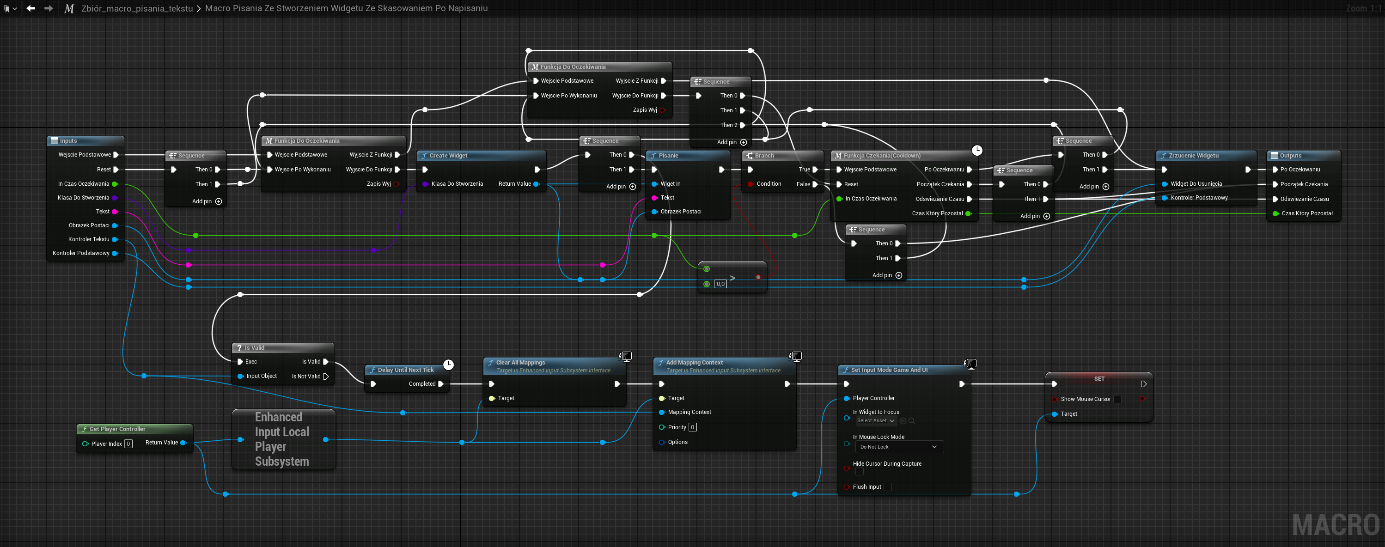
Wejścia

* Wszystkie elementy z ze\_stworzeniem\_widgetu.
* Kontroler podstawowy – kontroluje proces usuwania widgetu.

Wyjścia

* Standardowe wyjścia.

Natomiast na rycinie 4.2.18 zaprezentowano wnętrze kodu funkcji.

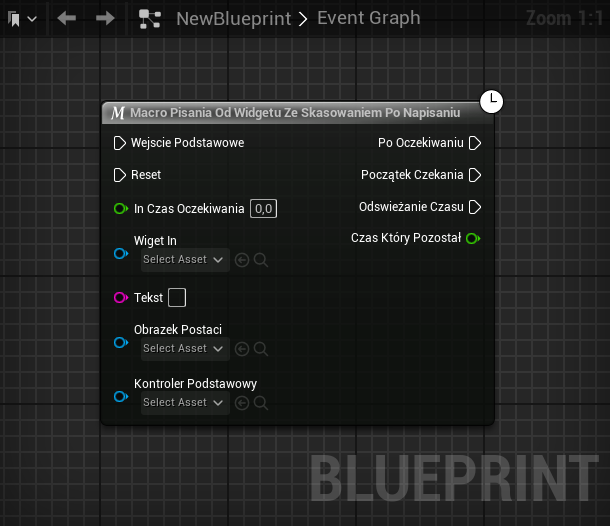


**Ryc. 4.2.18. Przedstawienie kodu funkcji Macro\_pisania\_ze\_stworzeniem\_widgetu\_ze\_skasowaniem\_po\_napisaniu**

Źródło: Opracowanie własne.

**Macro\_pisania\_od\_widgetu\_ze\_skasowaniem\_po\_napisaniu**

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



**Ryc. 4.2.19. Przedstawienie zewnętrzne jako blok macro Macro\_pisania\_od\_widgetu\_ze\_skasowaniem\_po\_napisaniu**

Źródło: Opracowanie własne.

Na rycinie 4.2.19 przedstawiono zewnętrzną formę funkcji, w której wejścia i wyjścia opisano następująco:

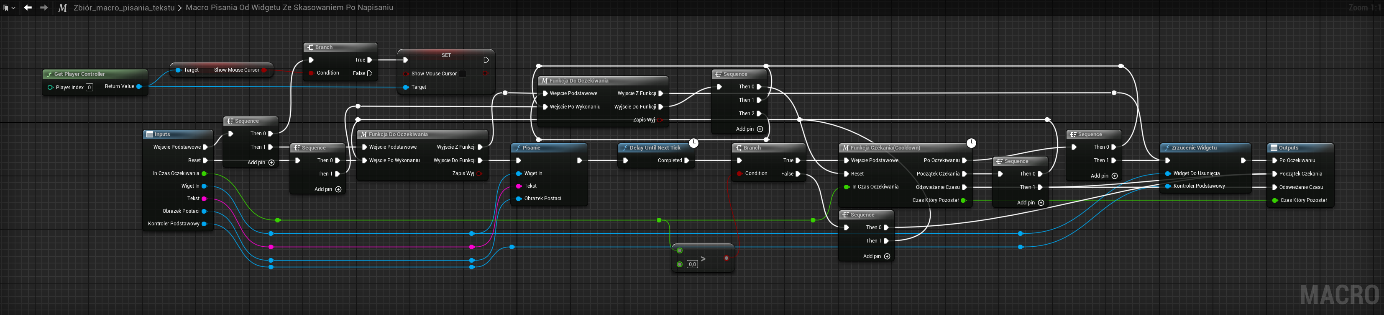
Wejścia

* Widget in – istniejący widget.
* Kontroler podstawowy – steruje procesem usuwania.

Wyjścia

* Standardowe wyjścia.

Natomiast na rycinie 4.2.20 zaprezentowano wnętrze kodu funkcji.



**Ryc. 4.2.20. Przedstawienie kodu funkcji Macro\_pisania\_od\_widgetu\_ze\_skasowaniem\_po\_napisaniu**

Źródło: Opracowanie własne

**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.

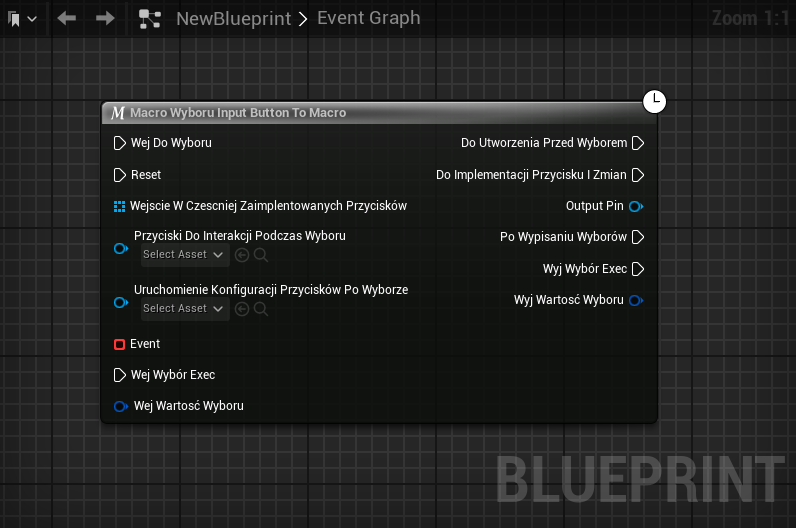
Elementy wspólne dla wszystkich makr wyboru:

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. 4.2.21. Przedstawienie zewnętrzne jako blok macro Macro\_wyboru\_input\_button\_to\_macro**

Źródło: Opracowanie własne.

Na rycinie 4.2.21 przedstawiono zewnętrzny kod funkcji, w której opisano wejścia i wyjścia następująco:

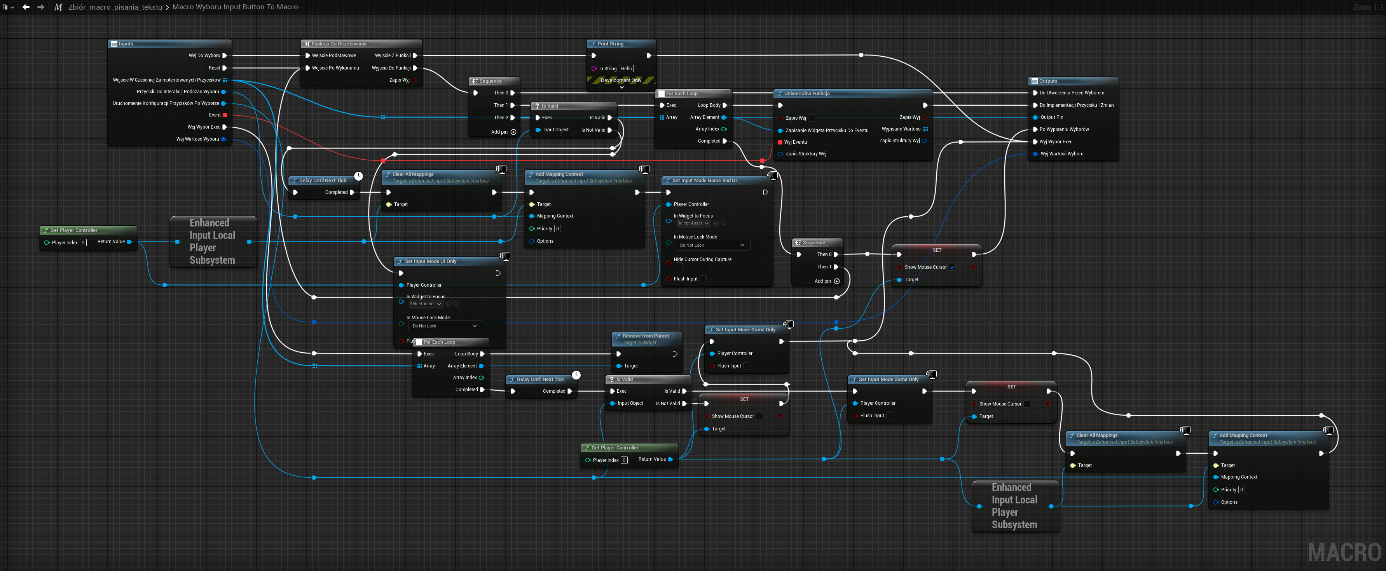
Wejścia

* Wejscie\_W\_czescniej\_zaimplementowanych\_przycisków – lista istniejących przycisków.

Wyjścia

* wyj\_Wybór\_exec – wyjście dla kontynuacji działania.
* wyj\_Wartość\_wyboru – zmienna wynikająca z wyboru gracza.

Na rycinie 4.2.22 zaprezentowano kod wewnętrzny funkcji.

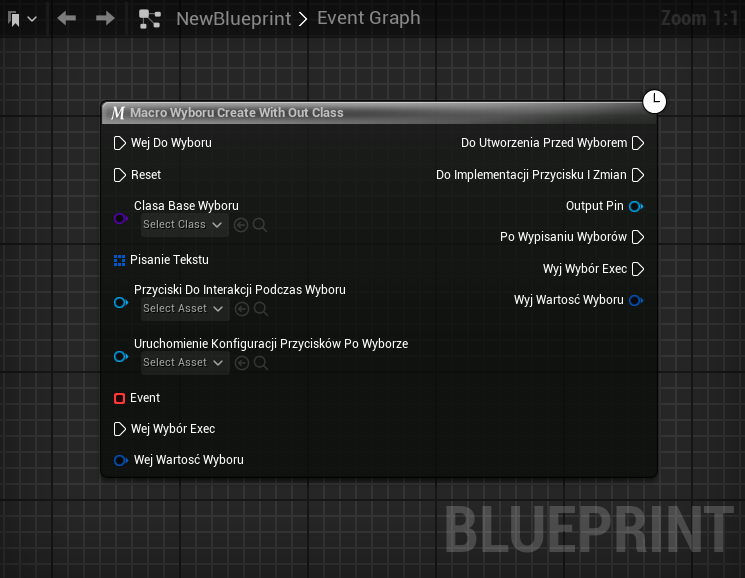


**Ryc. 4.2.22. Przedstawienie kodu funkcji Macro\_wyboru\_input\_button\_to\_macro**

Źródło: Opracowanie własne.

**Macro\_wyboru\_create\_with\_out\_class**

Tworzy nowe przyciski wyboru na podstawie klasy Base\_widget\_wybór.



**Ryc. 4.2.23. Przedstawienie zewnętrzne jako blok macro Macro\_wyboru\_create\_with\_out\_class**

Źródło: Opracowanie własne.

Na rycinie 4.2.23 zaprezentowano zewnętrzny kod funkcji, w której wejścia i wyjścia opisano w następujący sposób:

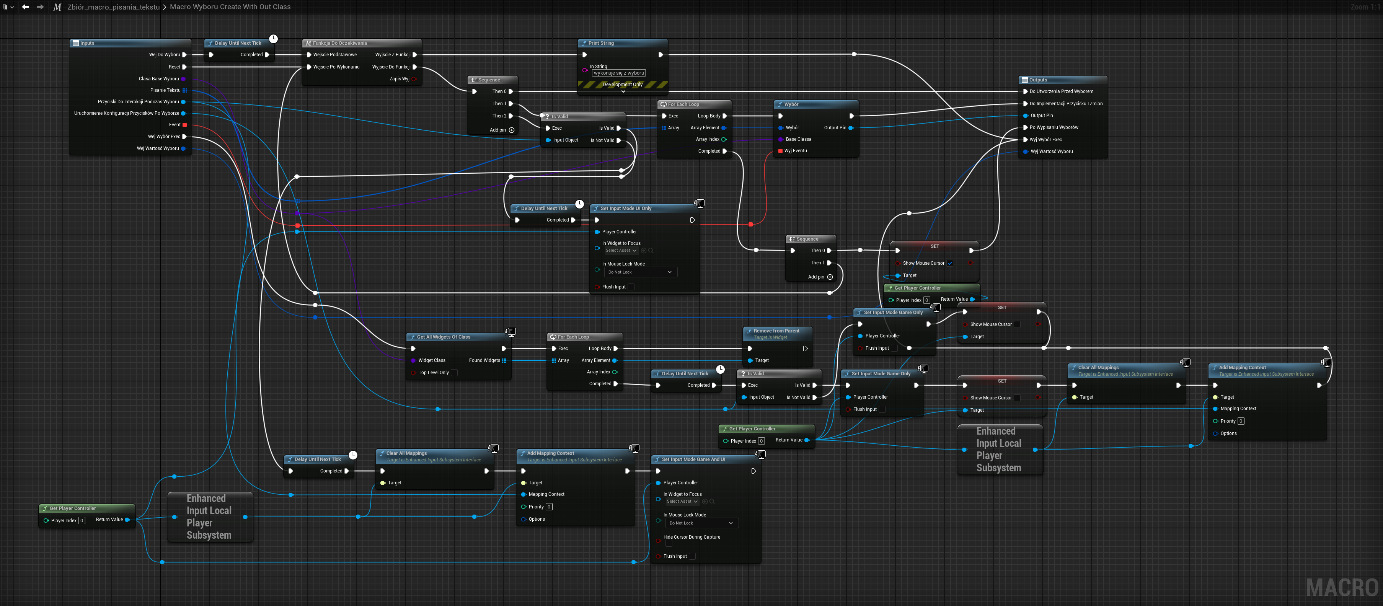
Wejścia

* Clasa\_base\_wyboru – klasa widgetu do utworzenia.
* Pisanie\_tekstu – tekst przypisywany do przycisku.

Wyjścia

* OutputPin – identyfikator utworzonego przycisku.

Z kolei na rycinie 4.2.24 przedstawiono wewnętrzny kod funkcji.

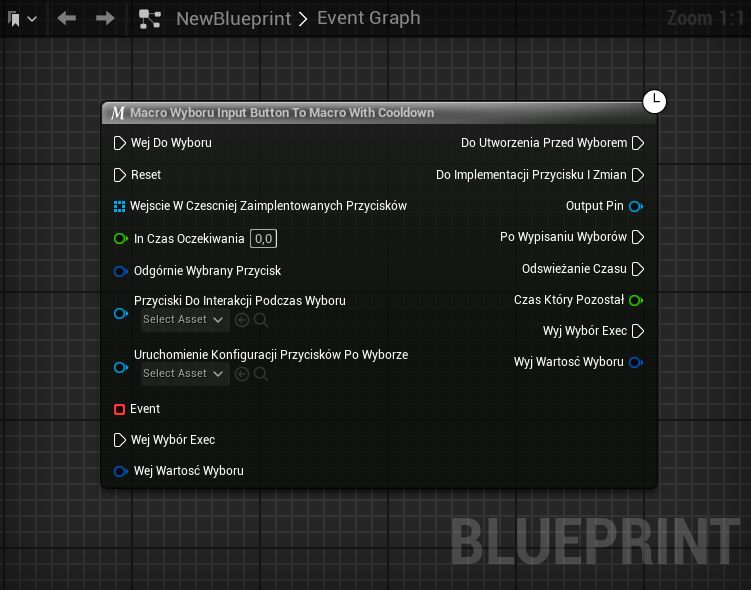


**Ryc. 4.2.24. Przedstawienie kodu funkcji Macro\_wyboru\_create\_with\_out\_class**

Źródło: Opracowanie własne.

**Macro\_wyboru\_input\_button\_to\_macro\_with\_cooldow**n

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



**Ryc. 4.2.25. Przedstawienie zewnętrzne jako blok macro Macro\_wyboru\_input\_button\_to\_macro\_with\_cooldown**

Źródło: Opracowanie własne.

Na rycinie 4.2.25 zaprezentowano zewnętrzny kod funkcji, w której wejścia i wyjścia opisano następująco:

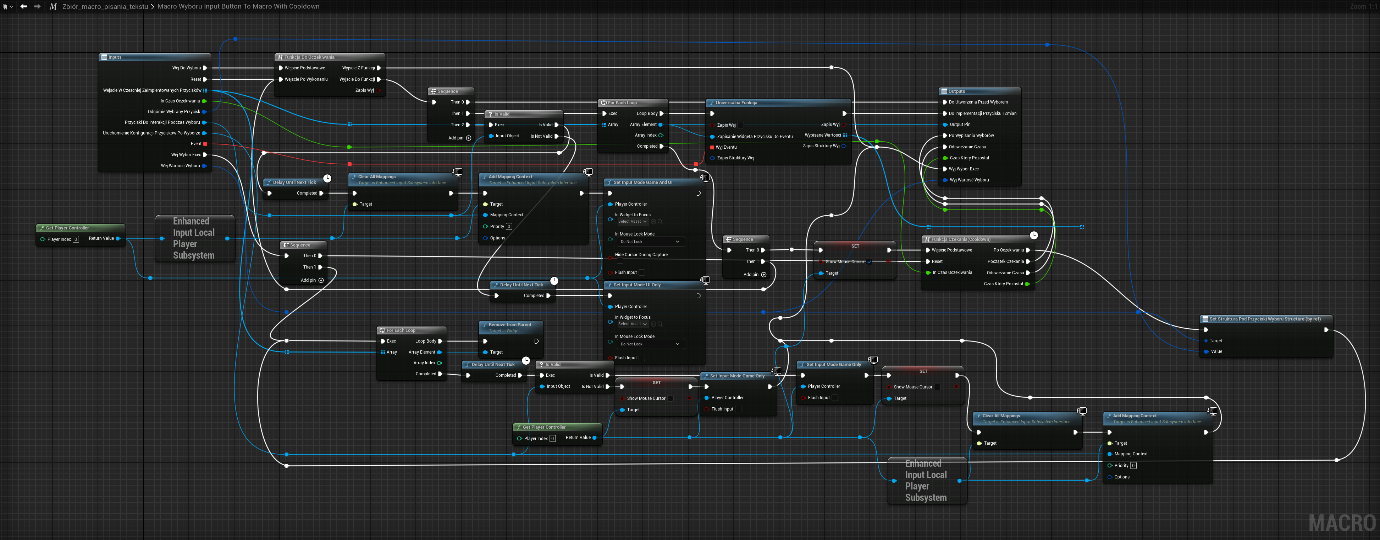
Wejścia

* Wszystkie z input\_button\_to\_macro.
* in\_czas\_oczekiwania – czas cooldownu.

Wyjścia

* Wszystkie standardowe wyjścia z dodatkiem cooldownu.

Natomiast na rycinie 4.2.26 pokazano wnętrze kodu funkcji.

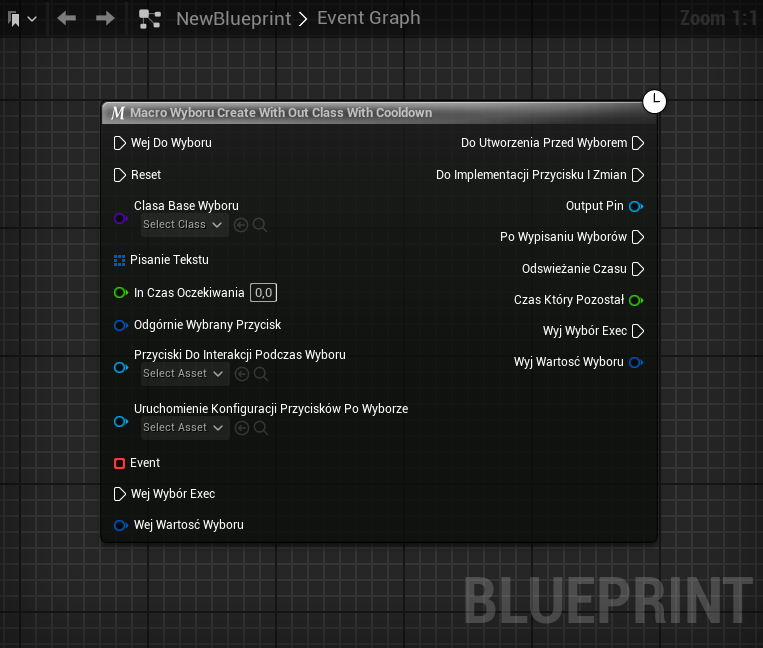


**Ryc. 4.2.26. Przedstawienie kodu funkcji Macro\_wyboru\_input\_button\_to\_macro\_with\_cooldown**

Źródło: Opracowanie własne.

**Macro\_wyboru\_create\_with\_out\_class\_with\_cooldown**

Łączy tworzenie nowych przycisków z mechanizmem cooldown.



**Ryc. 4.2.27. Przedstawienie zewnętrzne jako blok macro Macro\_wyboru\_create\_with\_out\_class\_with\_cooldown**

Źródło: Opracowanie własne.

Na rycinie 4.2.27 zaprezentowano, jak wygląda zewnętrzna część kodu funkcji. Wejścia i wyjścia oznaczono w następujący sposób:

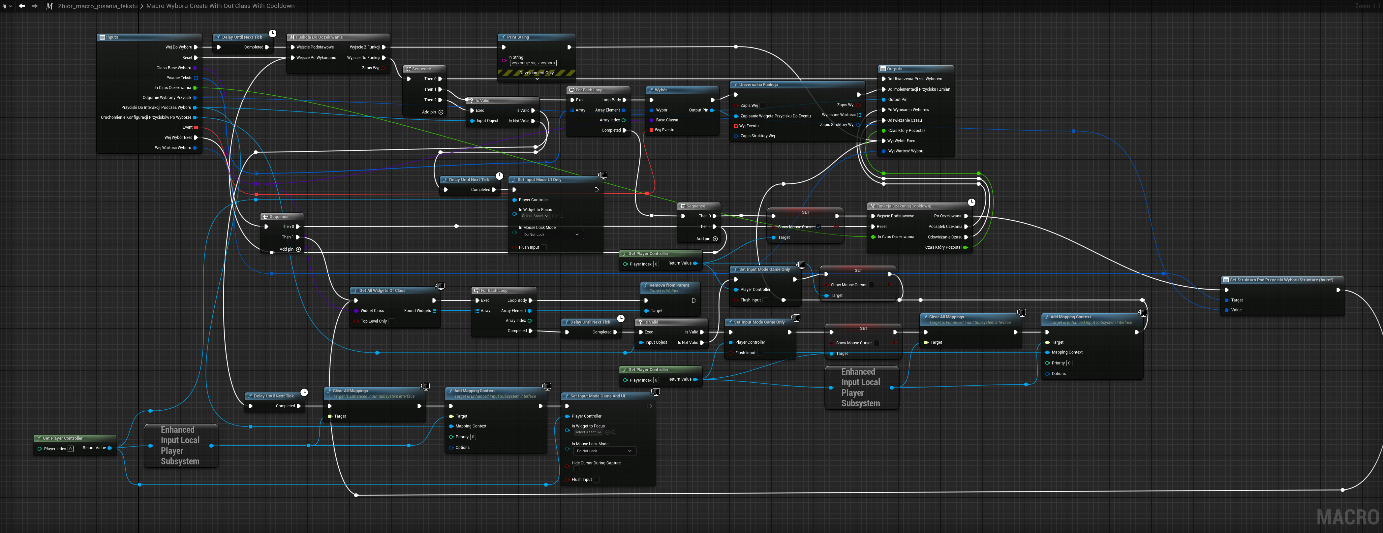
Wejścia

* Wszystkie z create\_with\_out\_class.
* Odgórnie\_wybrany\_przycisk – przydziela konkretny przycisk po upływie cooldownu.

Wyjścia

* Wszystkie standardowe wyjścia.

Z kolei na rycinie 4.2.28 pokazano wnętrze kodu funkcji.



**Ryc. 4.2.28. Przedstawienie kodu funkcji Macro\_wyboru\_create\_with\_out\_class\_with\_cooldown**

Źródło: Opracowanie własne.

# 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 – funkcję pisania zrealizowano jako pierwszą. Stanowiła ona 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 – każdą nową funkcjonalność testowano pod kątem poprawności działania. Błędy identyfikowane podczas testów eliminowano w kolejnych iteracjach lub pod koniec projektu w ramach pomniejszych testów. Stare funkcje i makra często zmieniano podczas implementacji nowych, ponieważ ich wcześniejsze wersje okazywały się niewystarczające dla potrzeb projektu.
3. Test zewnętrzny – narzędzie udostępniono 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. W trakcie testów zauważono, że funkcje cooldown w pewnych warunkach nie działały poprawnie, co wymagało wprowadzenia dodatkowych warunków zabezpieczających. Integracja makr wyboru z dynamicznym tworzeniem widgetów początkowo była problematyczna, jednak usprawniono ją 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 napisano w sposób przystępny, unikając zbyt technicznego języka, co miało ułatwić zrozumienie narzędzia przez osoby trzecie.

Wyniki testu zewnętrznego nie były zadowalające. Tester wskazał na brak intuicyjności narzędzia, co utrudniało jego obsługę dla osoby bez wcześniejszego doświadczenia z systemem. Zauważono niedostateczną szczegółowość dokumentacji, która nie wyjaśniała w pełni działania niektórych funkcji i mechanizmów. 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. Powstałe w ten sposób środowisko do testowania umożliwiło 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*, która zdaniem autora 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. Mechanikę gry oparto na prostych, ale znaczących wyborach, takich jak skręt w lewo lub w prawo, wchodzenie po schodach, korzystanie z windy czy zabranie ze sobą wiadra. Każda z tych decyzji prowadzi do innego zakończenia.

Gra w swojej formie spełniła swoją rolę, wyświetlała tekst i wykonywała dodatkowe akcje przed, podczas i po wywołaniu funkcji pisania. Funkcje wyboru również działały poprawnie. Jedynym problemem, który został zaobserwowany, było wywoływanie pochodnych Base\_Actor\_Component i implementacja logiki przycisku.

W pierwszych próbach stworzenia bardziej otwartej formy, bez konieczności tworzenia nowego obiektu tylko po to, by dodać odpowiednią pochodną Base\_Actor\_Component do nowego blueprintu obiektu, pojawił się pomysł na tworzenie Actor Componentów z klas. To rozwiązanie zadziałało po dodaniu funkcji „Add Actor Component”. Jednak komponent był przypisywany bezpośrednio do obiektu, co w krótkiej perspektywie nie stanowiło problemu. Z czasem zauważono, że nie działa kontroler gracza. Rozwiązaniem okazało się tworzenie Actor Controllerów w Playerze.

Takie podejście, choć skuteczne, nie było w pełni pożądane, ale zostało szybko naprawione poprzez dodanie do wartości „Target” w „Add Actor Component” odwołania do Playera. To rozwiązanie okazało się wystarczające dla tego projektu. Jednak funkcjonalność resetu nie była wykorzystywana, ponieważ każda akcja powodowała usunięcie tekstu, co może wymagać dalszego rozpatrzenia w przyszłości.

Pomimo tych wyzwań gra działała poprawnie dzięki prostocie i praktycznemu wykorzystaniu zaprojektowanych mechanik. Nie licząc systemu interakcji gracza z obiektami, który opracowano na podstawie tutoriala[[1]](#footnote-1), gra spełniła swoje zadanie jako platforma testowa.

Gra ukazała użyteczność zaprojektowanych mechanik w rozgrywce, w której kluczowe znaczenie miały podejmowanie decyzji i dokonywanie wyborów. Wykorzystanie Actor Componentów w wywoływaniu animacji lub dźwięków okazało się trafnym pomysłem, choć funkcje mogłyby być bardziej otwarte na implementację w różnych miejscach. W idealnym przypadku funkcje powinny być pozostawiane w obiektach, które je wywołują, i same wywoływać działania bez konieczności znajdowania się w Playerze.

Podczas testów zauważono, że każdy Actor Component jest tworzony od nowa. Konieczne będzie zbadanie, czy takie zrzucanie i tworzenie nowych komponentów nie wpływa negatywnie na stabilność gry. Pomimo tych zastrzeżeń gra spełniła swoją rolę jako narzędzie demonstracyjne, skutecznie pokazując potencjał zaprojektowanych mechanik.

## 5.3. Wnioski z testów

Testowanie narzędzia wspomagającego tworzenie dynamicznych kampanii w grach komputerowych pozwoliło na wyciągnięcie wniosków dotyczących jego funkcjonalności, użyteczności oraz potencjalnych obszarów wymagających usprawnienia. W rozdziale tym przedstawiono ogólne refleksje wynikające z przeprowadzonych testów jednostkowych oraz integracyjnych, a także ich wpływ na dalszy rozwój narzędzia.

Proces tworzenia, jak wspomniano w poprzednich rozdziałach, obejmował ciągłe testy iteracyjne kodu. W trakcie implementacji funkcji wielokrotnie je testowano, poprawiano i dostosowywano do nowych wymagań oraz doskonalono wraz z nabywanym doświadczeniem projektowym. Rozwiązania, które początkowo wydawały się bardzo efektywne, z czasem okazywały się problematyczne, co wymagało ich modyfikacji lub ponownego przepisania w celu dostosowania do zmieniającej się wizji projektu.

Po oddaniu narzędzia do otwartych testów, tester zgłosił uwagę dotyczącą niedopracowanej instrukcji. Mimo że stworzono ją z myślą o lepszym zrozumieniu funkcji, dostarczała zbyt mało informacji, aby potencjalny użytkownik mógł sprawnie zaplanować kodowanie projektów. Instrukcję poprawiano podczas testów, jednak nadal nie spełniała oczekiwań i nie przyczyniła się do pozytywnego odbioru projektu. Wyniki ankiety wskazały, że zarówno sam program, jak i dostarczone materiały pomocnicze oceniono na średnim poziomie. Kluczowym problemem był brak jasności w działaniu funkcji – użytkownicy skarżyli się, że musieli samodzielnie domyślać się ich działania i tworzyć własne rozwiązania, aby przetestować dostarczone mechanizmy. Z tego wynika, że narzędzie nie było intuicyjne dla nowych użytkowników, co wskazało na potrzebę lepszego nazewnictwa funkcji. Mimo poprawnego działania funkcji oraz ich użyteczności, brak odpowiedniego oznaczenia i dokumentacji sprawiał, że narzędzie zostało źle odebrane.

Testy wewnętrzne przyniosły inne rezultaty. Autor, korzystając z funkcji podczas tworzenia gry, wiedział, jakie mają zastosowania, co znacznie ułatwiało ich obsługę. Dodawanie tekstu było proste, a same rozwiązania okazały się intuicyjne dla osób zaznajomionych z ich działaniem. Problemy, takie jak brak widgetów czy niewprowadzenie tekstu, można było szybko naprawić bez konieczności ingerencji w kod. Testy wewnętrzne wykazały, że funkcje są przyjazne w użyciu dla osób znających ich konstrukcję i umożliwiają łatwe ograniczenie liczby tworzonych widgetów dzięki drzewiastej konstrukcji logiki.

# 6. Podsumowanie

Cały proces tworzenia projektu był dla autora wyjątkową przygodą. Podczas tworzenia projektu, procesy planowania, zmian decyzji, udoskonalania oraz poszukiwania kompromisów okazały się niezwykle pouczające, pokazując pełny obraz tego, jak w praktyce realizuje się tego rodzaju przedsięwzięcia. W poszczególnych rozdziałach pracy odwzorowano etapy tego procesu w formie opisowej – od wyboru silnika, na którym oparto makra pisania i wyboru, przez przemyślenie zasad, jakie powinny spełniać te funkcje, aż po ich implementację i końcowe testy, które ujawniły ograniczenia projektu. Spojrzenie na projekt na jego początku było zupełnie inne niż po zagłębieniu się w pracę z blueprintami w Unreal Engine 5.3.

Niemniej, projekt w końcowej formie nie w pełni odzwierciedlał pierwotne założenia. Funkcje miały być intuicyjne i łatwe w użyciu dla każdej osoby. Choć autor nie miał trudności z ich obsługą, użytkownicy zewnętrzni zgłaszali problemy z ich zrozumieniem. Pomimo tego, udało się stworzyć solidną bazę, która może być rozwijana, naprawiana i ulepszana w przyszłości. Choć blueprinty Unreal Engine mają pewne ograniczenia, szczególnie w kontekście edytowalnych sekwencji opcji, to podczas projektowania gry zauważono, że wiele z tych funkcji można zrealizować przy użyciu odpowiednich rozwiązań. Trudność polegała jednak na zbyt małej wiedzy autora w momencie rozpoczęcia projektu, co utrudniało osiągnięcie niektórych założeń.

Projekt sprawdził się bardziej jako badanie doświadczeń i odkrywanie możliwości niż jako kompletny program gotowy do komercyjnego wykorzystania. Biorąc pod uwagę rok pracy, jaki przeznaczono na jego realizację, osiągnięte efekty były zadowalające. Zgłębienie tematu wymagałoby znacznie więcej czasu, refleksji, modyfikacji oraz wprowadzenia dodatkowych mechanik. Gry z dynamiczną kampanią wymagają prostych i elastycznych rozwiązań, umożliwiających szybkie dostosowanie treści, systemów interakcji z otoczeniem, łatwiejszego tworzenia scenariuszy oraz wizualizacji relacji między elementami fabularnymi. Dodatkowo, mechanizmy śledzenia i zaznaczania zmian w strukturze historii oraz wpływu czynów gracza na fabułę mogłyby znacznie wzbogacić funkcjonalność narzędzia.

Autor ma nadzieję, że opracowany projekt będzie stanowił inspirację dla przyszłych programistów oraz podstawę dla bardziej zaawansowanych rozwiązań w tej dziedzinie. Być może stanie się on krokiem w stronę ustandaryzowania takich mechanik lub przyczyni się do stworzenia narzędzi, które wprowadzą nowe standardy w projektowaniu dynamicznych kampanii w grach komputerowych.

# Bibliografia

Użyta podczas pisania tekstowej części

Źródła internetowe

1. Unreal Engine Documentation.Epic Games, Inc. Oficjalna dokumentacja Unreal Engine, <https://docs.unrealengine.com>.
2. Reptile *Czym jest RPG Maker? (Co to jest, do czego służy?),* <https://www.rpgmaker.pl/?co=krypta&typ=kompendium&id=rm-cotojestrm>
3. Opracowanie z Wikipedii <https://pl.wikipedia.org/wiki/RPG_Maker>
4. Golenia, Aleksandra. (2023). *Unity – silnik graficzny nie tylko do gier*, <https://teamquest.pl/blog/2639_unity-silnik-graficzny-nie-tylko-do-gier>
5. Wikipedia poświęcona programowi Twine <https://twinery.org/cookbook/terms/terms_twine.html>
6. Ogólny kontekst czym jest program Twine <https://pl.gameofgdansk.eu/twine/>

Inne materiały

1. **Własna interpretacja wiadomości znanych wcześniej** Doświadczenie z rożnych miejsc oraz przesłuchanych opinii z różnych źródeł.
2. **Moja baza wiedzy i doświadczeń**. Wiedza oparta na doświadczeniach z zakresu tworzenia gier komputerowych i wykorzystania narzędzi deweloperskich (np. Unreal Engine).
3. **Własna implementacja funkcji i testy.** Doświadczenia z projektowania narzędzia wspomagającego dynamiczne kampanie, opisane szczegółowo w pracy inżynierskiej.
4. **Baza wiedzy ChatGPT** Użycie AI do poszukiwania dodatkowych nie konwencjonalnych propozycji oraz poprawy tekstu pod kątem gramatycznym i lingwistycznym.

Użyte podczas opracowywania funkcji i gry

Źródła internetowe

1. Własna interpretacja swoich doświadczeń z gier komputerowych z dynamiczną kampanią.
2. Unreal Engine Documentation. Epic Games, Inc. Oficjalna dokumentacja Unreal Engine. Dostępny pod adresem: <https://docs.unrealengine.com>.
3. The CG Essentials. (2021). *6 Ways to CREATE WALLS in Blender! (Architecture in Blender)*, <https://www.youtube.com/watch?v=Cunh57XjRv4&t=418s>
4. NiceShadow. (2023). *Interaction system* (seria filmów), <https://www.youtube.com/playlist?list=PL3TrrCsmmxlmNTR5FbExgWsK7p-DjBIZA>
5. Albeluhn, Christopher. (2024). *Paintings: Landscapes of the USA*, <https://www.fab.com/pl/listings/fc52be9e-ffaf-4bfb-8010-5a498fad6e7e>
6. Ashty. (2023). *How to make your own customizable PS1 Shader in 4 minutes (UE4/UE5)*, <https://www.youtube.com/watch?v=ST_JTXuGmv0>
7. Kanał z samouczkami Blendera. (2021). *Make a book case in blender. Quick and easy tutorial, blender how to make a bookshelf,,wow*, <https://www.youtube.com/watch?v=JSOLEnq02ag>
8. Deepak Graphics. (2022). *Modeling Table Under 2Min || Blender Beginners Tutorial*, <https://www.youtube.com/watch?v=8b7G0AWcV1k>
9. Visual Pixels CG. (2024). *How to make a Coffee cup in Blender Tutorial (Beginner friendly)*, <https://www.youtube.com/watch?v=anG15t4od80&t=208s&pp=ygULYmxlbmRlciBtdWc%3D>
10. BlenderVitals. (2024). *Create a Chair in Blender in 1 Minute!*, <https://www.youtube.com/watch?v=1k9gRrg8c9Q>
11. Grafika drzwi z windy <https://gotowi.org/jak-wydostac-sie-z-windy-w-razie-awarii/>
12. Grafika Kafelki <https://www.estile.pl/sklep/motyw/plytki-z-motywem-geometrycznym/wow-tesserae-play-blanc-28x28-mozaika/>
13. Grafika Drzwi zwykłe <https://www.freepik.com/vectors/metal-door-texture/3>
14. Grafika Ściana z poduszek ściągnięta <https://depositphotos.com/photo/quilted-velour-buttoned-ultimate-grey-color-fabric-wall-pattern-background-434099288.html>
15. Grafika Boazeria <https://www.vidaron.pl/poradnik/metamorfoza-starej-boazerii-krok-po-kroku>
16. Grafika ceglanej ściany pod obrazek: <https://t4.ftcdn.net/jpg/05/61/53/05/360_F_561530525_t98TqRdgHySsjdWRK86nHRS90XzbJNCt.jpg> pod portal: <https://stock.adobe.com/pl/search?k=white+grey+vintage+brick+wall+texture+background>
17. Grafika Podłoga z deskami <https://stock.adobe.com/ch_fr/images/floor-wood-parquet-flooring-wooden-seamless-pattern-design-laminate-parquet-rectangular-tessellation-floor-tile-parquetry-plank-hardwood-tiles-rectangles-slabs-brown-wooden-vector-background/396396535>

1. <https://www.youtube.com/playlist?list=PL3TrrCsmmxlmNTR5FbExgWsK7p-DjBIZA> [dostęp: 14.01.2025]. [↑](#footnote-ref-1)