



UNIWERSYTET ŚLĄSKI
W KATOWICACH

Symulacje Komputerowe

Projekt: Symulacja wyborów parlamentarnych
w Polsce

Wiktor Kalaga, Kamil Wyżgoł
Informatyka II stopnia (niestacjonarne)
Grupa 1
Rok 1

1. Opis projektu

Projekt został wykonany w środowisku .NET jako biblioteka programistyczna, do której przykładowa wizualizacja została zaimplementowana w silniku WPF (ang. *Windows Presentation Foundation*) z wykorzystaniem języka C# przy użyciu Visual Studio IDE (ang. *Integrated Development Environment*). Jest to stworzony od zera wieloagentowy symulator, mający na celu przeprowadzenie skonfigurowanej przez użytkownika symulacji wyborów parlamentarnych w Polsce. Dzięki temu, że najważniejsza część symulatora (model środowiska oraz wszystkie potrzebne klasy reprezentujące dane oraz ich przepływ) została zaimplementowana jako niezależna biblioteka środowiska .NET, można próbować wdrażać przedstawione rozwiązanie i/lub jego elementy w wielu różnych technologiach – w tym także webowych i mobilnych.

Agenci w symulatorze reprezentowani są przez wyborców, polityków i liderów partii politycznych. Symulator prezentuje aktualne poparcie danej partii w wybranym regionie oraz obsługuje dwa tryby widoku – sejm oraz senat. Symulator uwzględnia również aspekt losowości, co sprawia, że każda symulacja jest unikalna.

Po zakończeniu obliczeń symulacyjnych, symulator prezentuje użytkownikowi ostateczne wyniki poparcia danej partii (w danym regionie senackim lub sejmowym), które z kolei mogą zostać przedstawiane bezpośrednio na mapie Polski w postaci kolorów odpowiadających danej partii politycznej – ustalonej jest to na poziomie konfiguracji. W przypadku rozwiązań opartych o technologię WPF stworzona biblioteka zapewnia wysoce interaktywną mapę, która jest w pełni konfigurowalna oraz jest w stanie wyświetlać wyniki wygenerowane w symulacji. Jednym z istotnych elementów programu jest możliwość parametryzacji procesu symulacji. Każda symulacja na samym początku wymaga skonfigurowania, jednakże zaimplementowano również domyślne parametry bazowe możliwe do wykorzystania, takie jak np.: partie polityczne: nazwa, kolor, podstawowe poglądy oraz populacja każdego okręgu senackiego wraz z ich przykładowymi poglądami mieszkańców danego regionu.

W efekcie, stworzony symulator stanowi narzędzie umożliwiające lepsze zrozumienie dynamiki procesu wyborczego, wpływu różnych czynników na wyniki oraz eksperymentowanie z różnymi scenariuszami. Dzięki zastosowaniu technologii .NET i WPF, komponenty są wysoce reużywalne i stosunkowo łatwe w użyciu.

1.1. Agenci

Jak zostało to wspomniane w [Opisie projektu](#) symulator zakłada występowanie trzech grup agentów:

- **Wyborcy** – zamieszkują dany okręg wyborczy. Mogą przeglądać internet, oglądać telewizję, spać, brać udział w spotkaniach z politykami oraz pracować.
- **Politycy** – organizują spotkania, uczęszczają do telewizji, zamieszczają wpisy w Internecie. Są związani z konkretnym okręgiem wyborczym. Ich liczba dla danej partii to średnio liczba 2 razy większa niż liczba mandatów do sejmu.
- **Liderzy polityczni** – agenci globalni – tj. mogą organizować spotkania w dowolnym (losowym) okręgu wyborczym. Mają podobny zakres czynności co politycy lokalni, ale są bardziej efektywni w zjednywaniu sobie ludzi (oczywiście mowa o domyślnych ustawieniach). Ich liczba to jeden na partię polityczną.

Ponadto każdy z typów agentów posiada swoje poglądy w 4 kategoriach:

- Konserwentyzm – Progresywyzm (poglądy społeczne),
- Eurosceptyzyzm – Eurooptymizm (poglądy dotyczące Unii Europejskiej),

- Socjalizm – Kapitalizm (poglądy gospodarcze),
- Demokracja nieliberalna (demokracja ludowa) – Demokracja liberalna (poglądy dotyczące podejścia do praworządności).

1.2. Opis modelu

Działanie symulacji rozpoczyna się od sprecyzowania **parametrów politycznych** (partie polityczne oraz ich poglądy (na tej podstawie generowani są politycy – liderzy polityczni mogą się różnić tylko o maksymalnie 5% od poglądów partii, a zwykli politycy maksymalnie 10%) oraz **parametrów demograficznych**, w których zdefiniowana jest populacja danego okręgu senackiego (zdecydowano się na taki zabieg z uwagi na to, że okręgi senackie zawierają się w jednym okręgu sejmowym, więc pozwala to na uniknięcie redundancji) oraz średnia wartość poglądów ludności zamieszkującej dany region. Dane dotyczące liczby ludności danego okręgu zaczerpnięto z PKW.

Następnie obliczana jest symulacja. Można wymienić kluczowe etapy takie jak:

- Rozprowadzanie liczby ludności po danych regionach (poziom gminy, miasta, dzielnicy miasta, miasta na prawach powiatu).
 - Generowanie liderów politycznych.
 - Generowanie polityków w każdym okręgu.
 - Generowanie wyborców - początkowo przydzielane są losowe wartości poglądów, a następnie wybierani są losowi agenci, których poglądy są modyfikowane tak, aby zaspokoić określoną średnią dla okręgu (możliwy do ustalenia próg akceptowalnej różnicy pomiędzy tymi wartościami).
 - Następnie ma miejsce główna pętla symulacji (która się toczy przez określony w kodzie okres czasu, wartość startowa to 08.08.2023 20:00:00, czyli dzień, kiedy została ogłoszona prawdziwa kampania wyborcza):
 - Liderzy partii politycznej wykonują swoje działanie.
 - Przejście do pętli okręgów, gdzie:
 - Swoje akcje wykonują lokalni politycy.
 - Swoje akcje wykonują potencjalni wyborcy.
 - Powrót do pętli głównej, zapisanie wyników i przejście do następnej godziny.
2. Akcje agentów mają określone prawdopodobieństwa wylosowania się ze zbioru, oraz z różnym prawdopodobieństwem mogą wzmacniać i osłabiać poglądy danego agenta lub wpływać na poparcie danej partii u innych agentów.

3. Specyfikacja techniczna projektu

Cały projekt symulatora składa się z kilku różnych podprojektów, które są ze sobą zintegrowane dzięki czemu tworzą dość zaawansowane i realistyczne środowisko symulacyjne.

3.1. Biblioteka klas

Reprezentowana jest jako osobny projekt w Visual Studio jako *Class library*. Jest to zestaw klas i przestrzeni nazw, które w odróżnieniu do innych szablonów projektów w Visual Studio nie mają punktu wejścia, w którym bezpośrednio można pisać kod całego programu. Biblioteka

klas umożliwia korzystanie z zawartych w niej komponentów takich jak nowych typów danych (klas), czy funkcji bez konieczności definiowania całej logiki w miejscach, w których nie jest to konieczne.

W projekcie biblioteka klas nosi nazwę **ElectionSimulatorLibrary** i zawiera główną logikę symulatora. Zastosowanie tego podejścia umożliwiło stworzenie uniwersalnego zestawu narzędzi programistycznych, który może posłużyć do zaimplementowania symulatora w dowolnej technologii i platformie sprzętowej (kompatybilnej z technologią *.NET*). Do najważniejszych elementów zawartych w bibliotece należą:

- Folder *Core*
 - **DemographySettings** – klasa z ustawieniami dot. demografii. Dane związane z frekwencją w całym kraju, a także liczebności populacji w danym okręgu są wypełniane przez użytkownika z poziomu konfiguracji (tj. przed uruchomieniem symulacji),
 - **Environment** – klasa środowiska symulacji, czyli jej kluczowa część. To tutaj ma miejsce pełny cykl życia agentów oraz dzieje się obliczanie symulacji.
 - **PoliticalSettings** – klasa z ustawieniami politycznymi. Nazwa, kolor oraz poglądy są uzupełniane przez użytkownika z poziomu konfiguracji,
 - **Simulation** – klasa będąca pośrednikiem pomiędzy środowiskiem, a użytkownikiem. To do niej dostarczana jest konfiguracja niezbędna do obliczania symulacji oraz to tutaj można pozyskać wyniki.
 - **TimeManager** – klasa, której obiekty zarządzają czasem symulacji. Wchodzi w skład środowiska.
 - **Values** – klasa będąca repozytorium wiedzy środowiska. To tutaj trafiają dane konfiguracyjne oraz wyniki symulacji.
- Folder *Data*
 - **Action** – klasa odwołująca się do wykonywanych akcji przez agentów,
 - **Agent** – klasa abstrakcyjna będąca klasą bazową dla klasy **Voter**. Jest to klasa reprezentująca agenta,
 - **Attributes** – klasa zawierająca w sobie pola dot. poglądów agenta (lub partii politycznej). Wartości te są w zakresie od -100 do 100 dla każdego z 4 określonych zagadnień.
 - **Event** – klasa reprezentująca zdarzenia występujące podczas symulacji. Jest wynikiem wykonania akcji przez agenta. **Eventy** polityków mogą trafić to listy dostępnych wydarzeń, co umożliwia wzięcie w nich udziału przez wyborcę.
 - **PoliticalParty** – klasa reprezentująca partie polityczne, mające brać udział w symulacji. Parametry uzupełniane przez użytkownika z poziomu konfiguracji,
 - **Politician** – klasa reprezentująca polityka. Dziedziczy po klasie **Voter**. nadpisywany jest wybór najlepszej partii w rankingu agenta na partię do której polityk należy.
 - **PoliticianLeader** – klasa specjalna reprezentująca lidera politycznego, który przewodzi danej partii.
 - **Region** – klasa niezbędna do funkcjonowania interaktywnej mapy Polski oraz całości symulacji. Przechowuje m.in. nazwy, współrzędne geograficzne granic okręgów a także identyfikatory danych okręgów, również tych zawartych wewnątrz.
 - **Result** – klasa bazowa reprezentująca obiekt z wynikami.

- **Voter** – klasa reprezentująca wyborcę. Dziedziczy po klasie abstrakcyjnej **Agent**, ponieważ wyborcy są agentami.
- Klasa **BaseValues** – klasa zawierająca funkcje ‘bazowe’, z których wyróżnić można statyczną metodę **bool LoadDataFromFile (string mapFilePath)**, która pozwala na załadowanie danych do wygenerowania mapy (dane te zostały w ramach projektu wygenerowane przez autorskie narzędzie służące do WebScrapingu, a samo narzędzie zostało dołączone do projektu).
- Plik **Enums** – zawiera zestaw ‘specjalnych’ klas, które reprezentują pewne stałe symboliczne mapowane do typów liczbowych, dzięki którym kod staje się czytelniejszy. Wyróżniamy:
 - Typy regionów (**RegionType**) – powiązane z mapą,
 - Rodzaj wyborów (**ElectionType**) – sejm lub senat,
 - Tryb mapy (**MapMode**) – mapa w trybie normalnym lub wynikowym,
 - Rodzaj wykonywanej akcji (**ActionType**) – akcje wykonywane przez agentów.
- Klasa **GlobalUsings** – od wersji 10 języka C# globalne usingi umożliwiają zdefiniowanie wielu referencji w jednym miejscu, które będą dostępne dla wielu klas bez konieczności definiowania ich w każdej z nich. W naszym projekcie przechowują kilka referencji do wielu klas.

3.2. Interfejs użytkownika

Wspomniana w [Opisie projektu](#) technologia WPF pozwala na tworzenie aplikacji okienkowych z różnymi kontrolkami. Symulator wyborów parlamentarnych składa się z wielu okien (widoków), między którymi można się płynnie poruszać. Widoki napisane są przy użyciu języka znaczników XAML. W projekcie głównym z szablonu WPF można wyróżnić następujące m.in. następujące widoki:

- **MainWindow** – główny widok symulator. Miejsce startowe programu, z którego można zacząć proces tworzenia symulacji lub dowiedzenia się jak to zrobić,

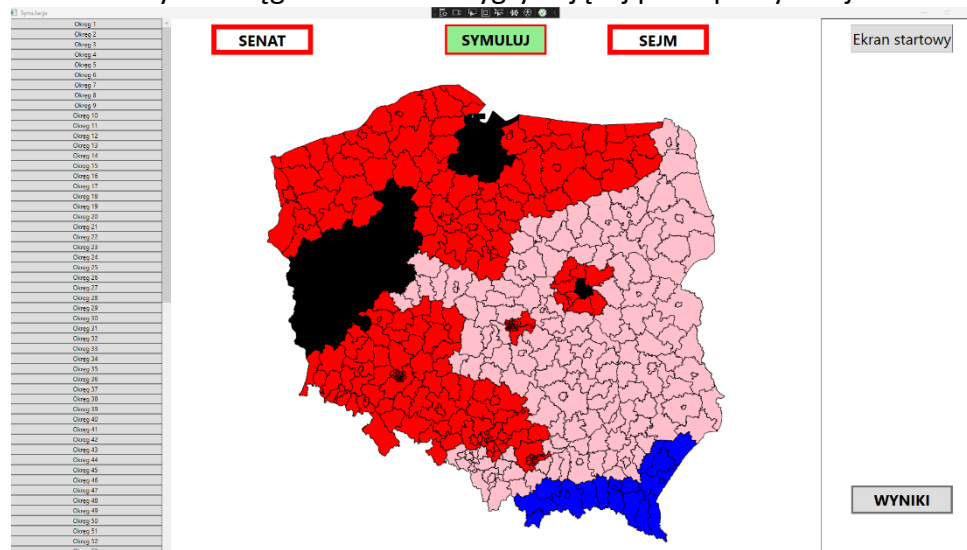
Symulator Wyborów Parlamentarnych

Rozpocznij symulację

Informacje

Repozytorium

- **MapSimulationWindow** – widok przedstawiający wygenerowaną mapę polski z zakreślonymi okręgami. W tym widoku również wyświetlone są wyniki symulacji z namalowanymi okręgami kolorem wygrywającej partii politycznej.



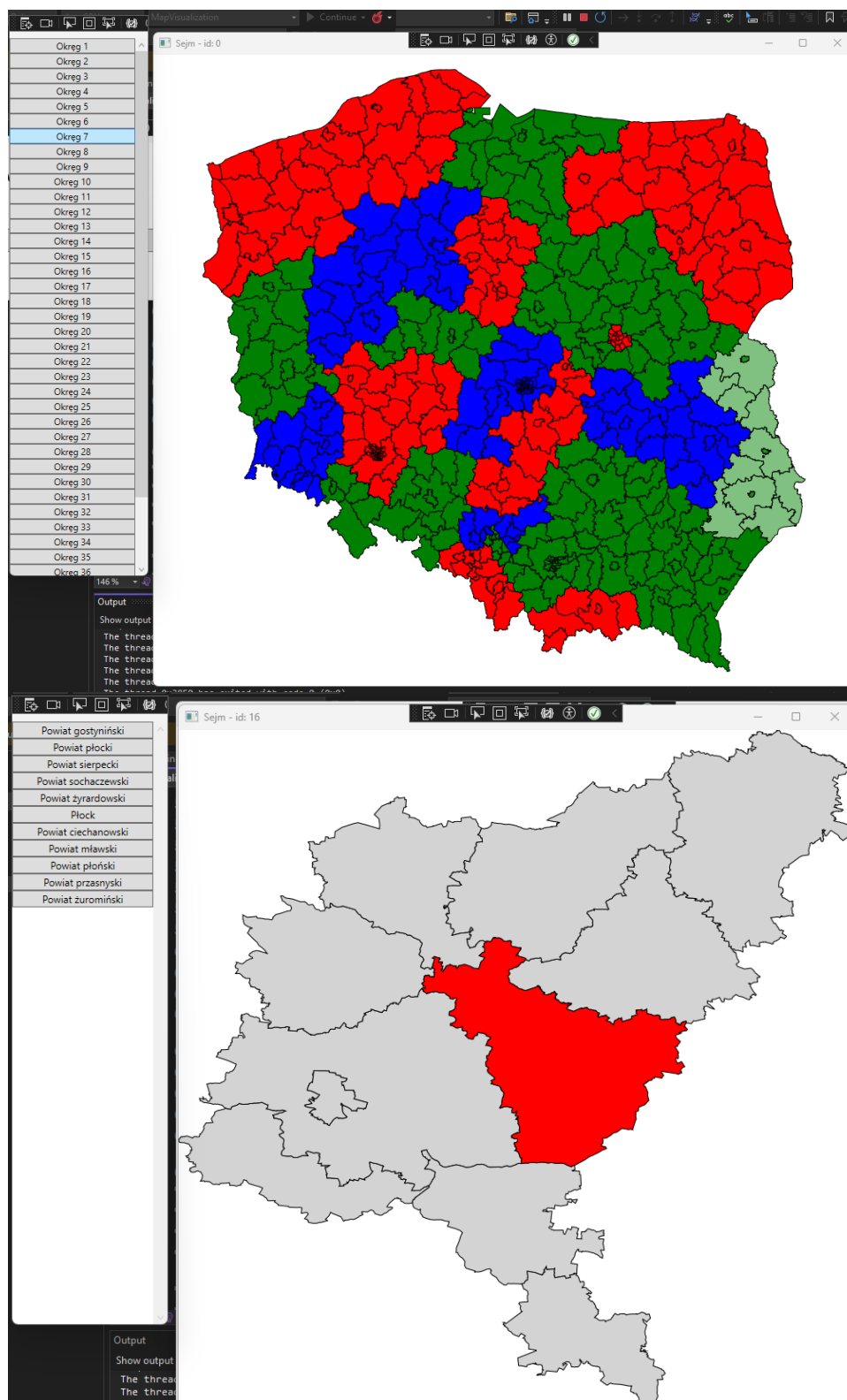
- **SummaryWindow** – widok podsumowania, posiadający w sobie zagnieźdżoną kontrolkę WebView, która przetrzymuje widok strony internetowej. W tym widoku wykorzystano technologię Blazor, dzięki której w kodzie HTML można pisać kod C#. Do wyświetlania wykresów z wynikami zastosowano bibliotekę JavaScript – Chart.js, która jest dostosowana właśnie pod technologię Blazor. Aby obsłużyć wyniki w tym widoku potrzeba jest ręczne ich dostosowanie (np. wyeksportowanie do pliku)



3.3. Interaktywna mapa

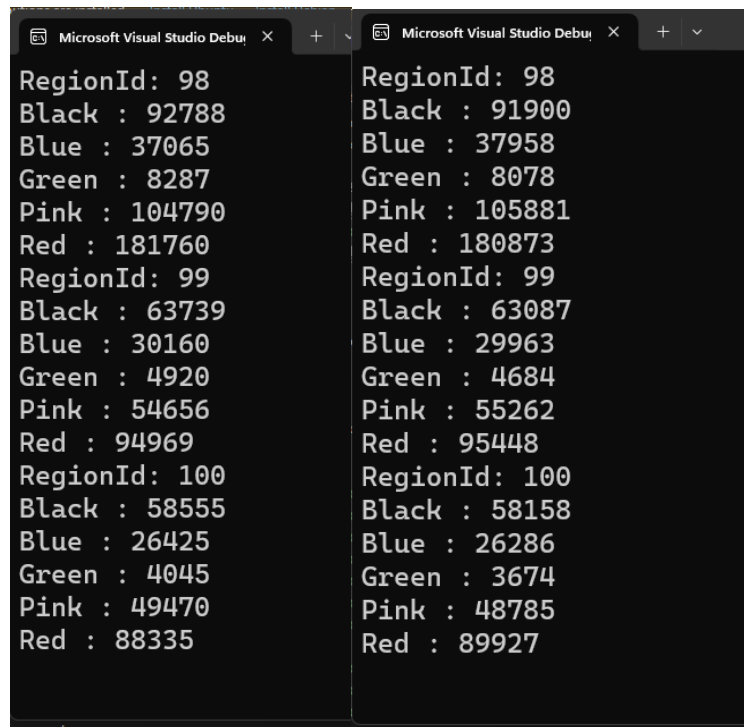
Jednym z ważniejszych elementów całego projektu symulatora jest interaktywna mapa Polski. Jest to komponent, który został zaprogramowany tak, aby był reużywalny i możliwy do użycia jako zwykły element w projekcie WPF.

Sama mapa została rzetelnie odwzorowana na podstawie danych udostępnionych przez Państwową Komisję Wyborczą, przy użyciu metody ekstrakcji danych ze stron internetowych z ich kodu HTML (ang. *Web scrapping*). Pozyskano dane wszystkich okręgów wyborczych zarówno do sejmu, jak i do senatu. Następnie użyto zdobytych informacji do określenia identyfikatora danego regionu w otwartej bazie OpenStreetMap, który następnie pozwolił na pobranie współrzędnych geograficznych składających się na granice danego regionu.



4. Przykładowe wyniki symulacji

Poniżej przedstawiono częściowe wyniki symulacji (wyświetlone w aplikacji konsolowej, która również jest częścią projektu i służy za przykład uniwersalności programistycznej opisywanego rozwiązania):



RegionId	Black	Blue	Green	Pink	Red
98	92788	37065	8287	104790	181760
99	63739	30160	4920	54656	94969
100	58555	26425	4045	49470	88335

RegionId	Black	Blue	Green	Pink	Red
98	91900	37958	8078	105881	180873
99	63087	29963	4684	55262	95448
100	58158	26286	3674	48785	89927

Są to dwa różne odpalenia aplikacji konsolowej dla tych samych parametrów. Jak można zauważyć wartości te są zbliżone, ale nie identyczne, więc zachowany został względny balans pomiędzy losowością, a sparametryzowanym odwzorowywaniem pewnego zjawiska zachodzącego w modelu symulacji. Stopień zróżnicowania wyników może w dużej mierze zależeć od ilości przebiegów czasowych symulacji (które można zmienić np. w kodzie źródłowym klasy `TimeManager`) oraz wykorzystanych parametrów.

5. Podsumowanie i wnioski

Podczas wykonywania projektu zdecydowaliśmy się stworzyć coś oryginalnego i własnego w tematyce przedmiotu, aby zaznajomić się z trudami i wyzwaniem, jakie mogą towarzyszyć podczas projektowania i wdrażania takiego oprogramowania, jakim jest symulator. Zaznajomiliśmy się z wieloma etapami, takimi jak np. proces zdobywania danych, tworzenie modelu oraz jego weryfikacja. Nie wszystkie elementy przedstawianego rozwiązania są pozbawione wad, ale jest to solidna baza pod jego dalszy rozwój. Dzięki skupieniu się na takich aspektach jak uniwersalność i reużywalność kodu jesteśmy w stanie skorzystać w przyszłości z wielu elementów projektu na różnych technologiach i platformach sprzętowych.

Zwróceniu uwagi wymagałby balans samej symulacji – tj. dopracowanie zależności pomiędzy parametrami i elementami środowiska. Zdecydowanie należy również zadbać o stabilność oprogramowania interfejsu graficznego – tak, aby był jak najbardziej przyjazny użytkownikowi końcowemu i odporny na błędy. Inną opcję rozwoju stanowi również rozbudowanie symulacji o nowe elementy, takie jak np. większy zasób akcji, poglądów, typów agentów, zwiększenie interakcji między agentami z różnych okręgów, wdrożenie sztucznej inteligencji zamiast prawdopodobieństwa (np. poprzez uczenie ze wzmocnieniem, ale wymagałoby to zdobycia sporej ilości danych i rozważenia wielu psychologicznych mechanizmów - np. heurystyk wydawania sądów - którymi kierują się ludzie podczas podejmowania decyzji).

Projekt był dla nas bardzo wymagający (ilość linii kodu można by liczyć w tysiącach), ale też bardzo rozwijający.

Wszystkie nasze zmagania i historia edycji projektu dostępne są w historii repozytorium w serwisie GitHub: <https://github.com/Rotkiw00/ComputerSimulationProject>

6. Źródła

- Państwowa Komisja Wyborcza; Krajowe Biuro Wyborcze <https://www.pkw.gov.pl/>
- Dokumentacja języka C# <https://learn.microsoft.com/pl-pl/dotnet/csharp/>
- Centrum Badań Opinii Społecznej <https://www.cbos.pl/PL/home/home.php>
- Serwis internetowy OpenStreetMap: <https://www.openstreetmap.org/>