Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie oraz nie korzystałem ze źródeł innych niż wymienione w pracy.

…….

(czytelny podpis)

Spis treści

1. Wstęp

Już od najdawniejszych czasów ludzie interesowali się różnego rodzaju zmaganiami sportowymi. Organizowano rozmaite zawody sportowe, igrzyska, turnieje a na początku XX wieku powstały cyklicznie rozgrywki ligowe. Do najpopularniejszych rozgrywek można zaliczyć: piłkę nożną, koszykówkę, hokej, baseball czy piłkę siatkową. Kibice obok ekscytacji związanej z oglądaniem spotkań sportowych zaczęli interesować się również statystykami prowadzonymi podczas rozgrywanych spotkań. Zaczęto odnotowywać liczbę zdobytych punktów, bramek czy asyst na mecz przez danego zawodnika. Wylicza się różnego rodzaju średnie, które pozwalają wybrać najlepszego zawodnika ligi, stworzenie najlepszej drużyny sezonu czy ustalenie która drużyna ma najlepsza obronę lub atak. Zaczęły też powstawać rożnego rodzaju gry planszowe czy karciane, które poprzez opis parametrów danego zawodnika czy drużyny umożliwiały rozegrać kilku osobom rozgrywki zbliżonej do tej rzeczywistej.

Pod koniec XX wieku rozwój komputerów i oprogramowania pozwolił na tworzenie aplikacji, które miały na celu symulować dane rozgrywki. Powstały rożnego rodzaju gry symulujące rozgrywki piłkarskie lub inne sporty drużynowe. Odkąd pamiętam zawsze interesowałem się piłką nożną. Oglądałem wszystkie turnieje Mistrzostw Świata i Europy w piłce nożnej, śledziłem różne ligi europejskie i puchary. Jednocześnie byłem miłośnikiem komputerowych symulatorów rozgrywek piłkarskich tak zwanych futbol menadżerów. Grałem w nie od czasów Commodore C-64 (lata 80-te) po dzień dzisiejszy. Zawsze zastanawiałem się jak działają i jak są skonstruowane. Tak narodziła się pasja do programowania, która w połączeniu z pasją do piłki nożnej była głównym czynnikiem, który pchnął mnie do stworzenia symulatora rzeczywistych rozgrywek ligowych.

Wyzwaniem jakim jest stworzenie takiego symulatora pozwoliła mi połączyć pasję z nauką programowania co było kolejnym krokiem na drodze osobistego rozwoju oraz pomogło mi na uzyskanie awansu zawodowego. Jednocześnie umożliwiło mi to pogłębienie wiedzy w zakresie programowania obiektowego i zastosowania jej w praktyce.

Podstawowe pojęcia wykorzystane w pracy:

- Klasa - typ złożony, tworzony przez programistę. Składa się z atrybutów (składowe, pola) opisujących daną klasę oraz z metod (funkcji) które mogą zmieniać wartości atrybutów. Atrybuty mogą posiadać specyfikator public – dostępne z każdego miejsca programu, private –dostępne tylko dla metod danej klasy oraz protected – dostępne dla metod klasy oraz klas które po niej dziedziczą. Oprócz zwykłych metod w jej skład wchodzi konstruktor służący do wytwarzania obiektów oraz destruktor do niszczenia.
- Obiekt – instancja klasy. Jest wytwarzany podobnie jak zmienna(C++), ponieważ w istocie jest nią lub przez dodanie operatora new (C++, Java, C#) służącego do utworzenia konkretnego egzemplarza klasy.
- Wzorce programowania -

2. Cel Pracy

Celem niniejszej pracy jest zaprojektowanie i stworzenie aplikacji, która umożliwiłaby symulowanie rozgrywek ligowych w tym przypadku rozgrywek piłki nożnej. Najważniejszym aspektem projektu a zarazem najtrudniejszym było odwzorowanie realistycznych zachowań drużyn i zawodników w odniesieniu do rzeczywistości. Na rynku istnieje kilka grup symulatorów (managerów). Pierwsza z nich realistycznie odwzorowuje rozgrywki ligowe różnych krajów oraz pucharów krajowych i zagranicznych. Druga grupa to symulatory typu fantazy w których uczestnicy mogą mieć te same drużyny z powtarzającymi się zawodnikami w ich składach.

W rzeczywistości istnieją rozgrywki piłkarskie pod nazwą „Champions League" skupiające najlepsze drużyny z Europy. Jednakże nie jest to liga w dosłownym słowa tego znaczeniu. System rozgrywek jest bardzo podobny do tego jaki jest stosowany w „Mistrzostwach Świata" w piłce nożnej drużyn narodowych. Polega on na tym, że wszystkie drużyny rozlosowywane są do grup składających się z czterech zespołów, następnie rozgrywają mecze każdy z każdym tak aby ostatecznie wyłonić dwie najlepsze drużyny z grupy, które następnie zmierzą się z pozostałymi drużynami w systemie pucharowym wyłonionymi z innych grup w ten sam sposób. Różnica polega na tym że w „Champions League" drużyny rozgrywają mecz i rewanż natomiast w „Mistrzostwach Świata" tylko jeden mecz z każdy z każdym. W rezultacie daje to 6 meczów w fazie grupowej plus 7 meczów w fazie pucharowej co daje łącznie 13 meczów pod warunkiem dotarcia drużyny do finału. Rezultatem takich rozgrywek jest to, że tylko dwie drużyny rozgrywają po 13 spotkań, połowa po 6 spotkań (przegrani w fazie grupowej) a pozostałe (drużyny które odpadły w fazie pucharowej) od 8 do 11.

Natomiast celem tego projektu jest utworzenie ligi składających się z 16 najlepszych drużyn europejskich tak aby umożliwić spotkanie każdy z każdym w systemie mecz rewanż. Pozwoliło by to na rozegranie każdej drużynie 30 spotkań i wyłonienie najlepszej z nich w możliwie najsprawiedliwszy sposób. Realizacja takiego symulatora pozwoli sprawdzić jak będą kształtowały się rozgrywki kiedy w jednej lidze zostaną zgromadzone najlepsze drużyny i najlepsi zawodnicy.

Z powodu wyżej wymienionych czynników projekt został napisany w technologii wykorzystującej obiektowość, która najbardziej nadaje się do takich rozwiązań. Ważnym celem jest także zaprojektowanie przyjaznego graficznego interfejsu użytkownika (GUI), który w łatwy i intuicyjny sposób umożliwi obsługę symulatora osobie z niego korzystającej.

3. Wybór technologii

3.1. Przegląd technologii

W tym rozdziale skupię się na przeglądzie dostępnych technologii na rynku, które umożliwiają wytwarzanie oprogramowania. Przedstawię podział języków ze względu na kryterium przydatności oraz uargumentuję dlaczego jedna grupa jest lepsza od innej do wykonania tego projektu. Dodatkowo postaram się opisać biblioteki graficzne dedykowane do poszczególnych języków, które umożliwiają w krótkim czasie wytwarzanie aplikacji okienkowych.

Języki programowania możemy podzielić na kilka grup. Najpopularniejsze to:

- funkcyjne
- skryptowe
- proceduralne
- obiektowe

3.1.1. Języki funkcyjne

Przedstawicielami języków funkcyjnych jest Haskell, Lisp, Erlang czy F#. Programy napisane w językach funkcyjnych składają się jedynie z funkcji. Główny program jest funkcją, która przyjmuje argumenty i zwraca obliczoną wartość. Może składać się on z innych funkcji, które mogą wywoływać jeszcze inne funkcje. Funkcja może przyjmować jako parametr funkcję jak również zwracać inna funkcję. W programowaniu funkcyjnym zamiast zmiennych występują stałe a zamiast pętli stosowana jest rekurencję. Najważniejszą cechą tych języków jest to, iż nie pozwalają na żadne efekty uboczne. W czystym programowaniu funkcyjnym, raz zdefiniowana funkcja dla tych samych danych wejściowych zawsze zwróci ten sam wynik. Języki funkcyjne zostały zaprojektowane z myślą o zastosowaniach współbieżnych (Erlang) oraz pod kątem tworzenia systemów rozproszonych wymagających długotrwałej pracy oraz odporności na awarie. Jak również często stosowane są do tworzenia sztucznej inteligencji (Lisp).

3.1.2. Języki skryptowe

Drugą grupą są języki skryptowe. Jak sama nazwa wskazuje są to języki, które obsługują skrypty. Najpopularniejsze to JavaScript, Pyton, PHP, Ruby on Rails, Perl czy Node.JS. Grupa ta odróżnia się od pozostałych tym, że kod napisany w tych językach może być tworzony nawet w notatniku i nie musi być kompilowany do pliku wykonywalnego - zrozumiałego dla komputera. Proces kompilacji w językach skryptowych odbywa się za każdym razem w momencie uruchomienia programu. Zaletą takiego programowania jest szybsze pisanie

aplikacji a programy tworzone w tych technologiach mają zwykle małą objętość. Języki skryptowe często stosuje się do automatyzowania różnego rodzaju zadań najczęściej administracyjnych. Skrypty wykorzystywane są również do sterownia przebiegiem gry, kontrolują wtedy fabułę, dialogi czy sterują zachowaniem wirtualnych postaci. Języki takie jak PHP czy JavaScript wykorzystywane są w technologiach internetowych (HTML) do dynamicznej interakcji z użytkownikiem poprzez reagowanie na zdarzenia, zapewnieniu walidacji danych z formularzy czy dynamicznym manipulowaniu obiektami na stronach internetowych.

3.1.3. Języki proceduralne

Kolejną grupą, której przedstawicielem jest język C, są języki proceduralne. Głównym założeniem tych technologii jest to, że program główny składa się z wielu procedur (podprogramów) lub funkcji realizujących określone zadania które mogą być wywoływane wile razy. Zadaniem procedur jest modyfikacja zmiennych globalnych, albo też modyfikacja wartości lub struktury przekazanej w parametrze, natomiast funkcja oprócz modyfikacji parametrów może je również zwracać. Możliwe jest również zagnieżdżanie odwołań do procedur dzięki czemu można korzystać z rekurencji. C jest językiem strukturalnym i nie wspiera programowania obiektowego, jednakże programowanie obiektowe jest w nim możliwe. Dodatkowo język C jest wśród języków wysokopoziomowych „najbliżej" maszyny co czyni go bardzo wygodnym narzędziem do tworzenia oprogramowania dla systemów czy mikrokontrolerów. Jeżyk C jest bardzo wydajny i lekki przez co chętnie jest wykorzystywany do programowania systemów wbudowanych stosowanych w urządzeniach domowych takich jak pralki, lodówki poprzez zegarki a kończąc na elementach samochodów takich jak: wyświetlacze, czujniki czy systemy zapewniające bezpieczeństwo. C został również do napisania jąder takich systemów jak Linux, Windows, Android czy biblioteki graficznej OpenGL.

3.1.4. Języki obiektowe

Ostatnią grupą stanowią języki obiektowe. Do najpopularniejszych technologii z tej grupy należą: C++, Java, C#. Najważniejszym pojęciem z wiązanym z programowaniem obiektowym jest klasa, która stanowi szablon na podstawie którego tworzone są obiekty (instancje klas). Można powiedzieć, że klasa jest to typ złożony zdefiniowany przez użytkownika. Dodatkowo klasa w odróżnieniu od struktury posiada metody(funkcje) umożliwiające wpływanie na zachowanie obiektu. Podstawowymi założeniami paradygmatu obiektowego są cztery cechy:

- Abstrakcja - polega na ukrywaniu lub pomijaniu mało istotnych informacji a skupieniu się na wydobyciu informacji, które są niezmienne i wspólne dla pewnej grupy obiektów.

- Dziedziczenie - Podstawowym zastosowaniem dziedziczenia jest ponowne wykorzystanie kodu. Jeśli dwie klasy wykonują podobne zadania, możemy utworzyć dla nich wspólną klasę bazową, do której przeniesiemy identyczne metody oraz atrybuty.
- Enkapsulacja – czyli hermetyzacja polega na ukrywaniu implementacji obiektu przed użytkownikiem i zapewnia , że obiekt nie może zmieniać w nieoczekiwany sposób stanu wewnętrznego innych obiektów.
- Polimorfizm – uwolnienie się od typów czyli traktowanie różnych danych w jednolity sposób. Rozróżniamy polimorfizm statyczny (czasu kompilacji) i polimorfizm dynamiczny (czasu wykonania)

Zastosowanie języków obiektowych jest bardzo szerokie. Wykorzystuje się je do pisania gier, aplikacji desktopowych, internetowych jak również mobilnych czy nawet systemów wbudowanych (C++).

Odnosząc się do powyższej charakterystyki technologii językowych stwierdzam, że najodpowiedniejszą grupą do stworzenia symulatora rzeczywistych rozgrywek ligowych w oparciu o graficzny interfejs użytkownika jest grupa języków obiektowych. Języki funkcyjne jak wcześniej napisałem mają główne zastosowanie w systemach rozproszonych czy projektowaniu sztucznej inteligencji. Języki skryptowe są w ostatnich czasach bardzo często wykorzystywane przy wytwarzaniu aplikacji internetowych. Języki proceduralne nadają się do programowania tego typu aplikacji, jednakże przewagą języków obiektowych nad językami proceduralnymi jest fakt, że taki sposób programowania jest bardziej zgodny rzeczywistością a ludzki mózg jest lepiej przystosowany do takiego podejścia podczas przetwarzania informacji. Podstawowe założenia paradygmatu obiektowego takie jak abstrakcja czy enkapsulacja przyczyniają się do zwiększenia czytelności natomiast dziedziczenie i polimorfizm są zwykle używane w celu redukcji zbędnego kodu. Dodatkowo w ostatnich latach języki obiektowe zostały wyposażone w potężne darmowe biblioteki graficzne , które ułatwiają i przyśpieszają znacząco tworzenie kodu.

3.2. Przegląd Języków Obiektowych

3.2.1. C++ i Qt

Język C++ to potężne narzędzie łączące w sobie trzy różne kategorie programowania: programowanie proceduralne charakterystyczne dla języka C, obiektowe wyrażone przez dodanie klas oraz programowanie uogólnione znajdujące wyraz w szablonach języka C++. Charakteryzuje się wysoką wydajnością kodu wynikowego, bezpośrednim dostępem do zasobów sprzętowych i funkcji systemowych, łatwością tworzenia i korzystania z bibliotek (napisanych w C++, C lub innych językach), niezależnością od konkretnej platformy sprzętowej lub systemowej (co gwarantuje wysoką przenośność kodów źródłowych) oraz niewielkim środowiskiem uruchomieniowym.

C++ został zaprojektowany przez Bjarne Stroustrupa jako rozszerzenie języka C o obiektowe mechanizmy abstrakcji danych i silną statyczną kontrolę typów. Zachowanie zgodności z językiem C na poziomie kodu źródłowego pozostaje jednym z podstawowych celów projektowych kolejnych standardów języka. W latach 90. XX wieku język C++ zdobył pozycję jednego z najpopularniejszych języków programowania ogólnego przeznaczenia.

Qt jest to wieloplatformowy framework służący do tworzenia aplikacji desktopowych, wbudowanych (embedded) i mobilnych które, mogą być uruchamiane na wszystkich podstawowych systemach (Windows, Linux, iOs, Android, BlackBerry i inne). Qt używa standardów C++ z rozszerzeniami takimi jak sygnały i sloty. Posiada również własny zestaw szablonów kontenerów (QVector, QList, QMap i inne). Qt wyposażony jest w Qt Creator narzędzie do szybkiego tworzenia graficznego interfejsu użytkownika (GUI). Ponadto Qt posiada bardzo dobrze napisaną dokumentację, oraz liczne przykłady i tutoriale ułatwiające pracę z tym środowiskiem. Biblioteki Qt, oprócz obsługi interfejsu użytkownika, zawierają także niezależne od platformy systemowej moduły obsługi procesów, plików, sieci, grafiki trójwymiarowej (OpenGL), baz danych (SQL), języka XML, lokalizacji, wielowątkowości, zaawansowanej obsługi napisów oraz wtyczek.

Bardzo łatwy sposób tworzenia GUI przy pomocy kreatora, rozbudowana dokumentacja z licznymi przykładami, jak również bardzo pomocny edytor powodują, że jest to najbardziej intuicyjne środowisko programistyczne z jakim do tej pory się spotkałem.

3.2.2. Java i Swing

Java jest językiem programowania który umożliwia wytwarzanie aplikacji na wiele platform. Każdy program napisany przez programistę kompilowany jest do kodu bajtowego i dzięki wirtualnej maszynie może działać na wielu systemach operacyjnych takich jak Windows, Linux i Mac OS. Java odziedziczyła wiele cech po C i C++ z których zapożyczono dużą cześć składni i słów kluczowych, natomiast zarządzanie pamięcią zajmuje się maszyna wirtualna (JVM) z języka Smalltalk. Java została stworzona przez Jamesa Goslinga z firmy Sun Microsystems.

Podobnie jak w C++ Java wykorzystuje cztery główne paradygmaty programowania obiektowego: abstrakcję, dziedziczenie, enkapsulację i polimorfizm. W odróżnieniu od języka C++ Java nie udostępnia wielodziedziczenia ale w zamian za to wprowadza interfejsy, których zadaniem jest wyeliminowanie konfliktów podczas przekazywania właściwości przez klasy nadrzędne (śmiertelny rąb). W Javie wszystkie obiekty oprócz typów prostych (int, float) dziedziczą po klasie nadrzędnej Object. Implementacja jej podstawowych zachowań i właściwości umożliwia ich porównywanie, identyfikację, kopiowanie oraz niszczenie. Kolejną różnicą miedzy tymi językami jest zarządzanie pamięcią (Memory Management). W C++ całość spoczywa w rękach programisty poprzez tworzenie i niszczenie wskaźników natomiast w Javie kontroluje to system przy pomocy narzędzia zwanego Garbage Collection. Java w przeciwieństwie do C++ który stosunkowo umożliwia programowanie niskopoziomowe dostarcza szeroki zakres klas dla różnych usług wysokiego poziomu.

Swing jest biblioteką graficzną służącą do wytwarzania aplikacji okienkowych w języku Java. Biblioteka powstała w 1997 roku i jest nową ulepszoną wersją biblioteki AWT. Najistotniejszą różnicą pomiędzy tymi bibliotekami jest to że AWT pobiera wszystkie

kontrolki z systemu co miało być zgodne z główną zasadą - niezależnością od platformy.. Natomiast Swing rysuje wszystkie komponenty od początku, dzięki czemu aplikacje wygląda identycznie na wszystkich platformach, na których jest uruchamiana. Minusem tego rozwiązania jest zmniejszenie wydajność aplikacji, jednakże przy obecnym rozwoju technologii (coraz szybsze komputery) jest praktycznie niewyczuwalne.

Budowa aplikacji z użyciem obu tych bibliotek polega na tworzeniu graficznego interfejsu użytkownika (GUI) z mniejszych składników takich jak przyciski, pola tekstowe, rozwijane listy czy etykiety. Całość umieszczana jest w panelach (JPanel), które umieszczane są w obiekcie nadrzędnym – ramce (JFrame). Do interakcji z użytkownikiem wykorzystywane są zdarzenia, które implementując interfejs ActionListener pozwalają na obsługę wciśniętego przycisku, odczytu położenia kursora na ekranie oraz wielu innych zdarzeń.

3.2.3. C# i WPF

C# to odpowiedź Microsoftu na Javę. Jest to obiektowy język programowania ogólnego przeznaczenia z bezpiecznymi typami. Język został zaprojektowany w latach 1998-2001 a jego głównym architektem trzymającym pieczę nad językiem jest Anders Hejlsberg(twórca Turbo Pascala). Projektanci tego języka starali zachować balans między prostotą, ekspresyjnością i wydajnością. Nazwa języka (podobnie jak nazwa C++ który poprzez wykorzystanie operatora inkrementacji ++ oznaczającego zwiększenie o jeden w odniesieniu do języka C) zawierająca symbol # przypominająca dwa operatory inkrementacji sugeruje, że C# jest następcą C++.

Podobnie jak Java wykorzystuję dużą część składni i słów kluczowych z C++. Również jeżeli chodzi o obiektowość i główne paradygmaty programowania to jest to bardzo zbliżone do Javy. (zamiast dziedziczenia wielokrotnego wprowadzono interfejsy).Dodatkowo C# wprowadza nowe elementy składowe klas takie jak: właściwości i indeksery. delegaty i zdarzenia to odpowiedniki i rozwinięcie wskaźników na funkcje z C++. Hierarchia dziedziczenia opiera się na istnieniu jednej klasy Object po której dziedziczą wszystkie inne klasy. W odróżnieniu od Javy również typy proste takie jak (int, double) są strukturami posiadające odpowiednie dla siebie metody takie jak: ToString, Equals czy GetType. W języku C# pamięcią automatycznie zarządza system wykonawczy. Środowisko uruchomieniowe CLR(Common Language Runtime) wyposażone jest w moduł usuwania nieużywanych obiektów(Garbage Collector) co zwalnia programistę (tak jak w Javie a w przeciwieństwie do C++) od pamiętania o własnoręcznej dezalokacji pamięci. Jednakże w celach optymalizacji wydajności kodu możliwe jest stosowanie wskaźników w specjalnych blokach oznaczonych jako niebezpieczne(unsafe).

W przeszłości język C# przeznaczony był w zasadzie do tworzenia aplikacji przeznaczonych na systemy Windows. Jednakże w ostatnich czasach poprzez rozwój takich narzędzi jak Xmarin czy ASP.NET możliwe jest tworzenie wieloplatformowych aplikacji mobilnych jak również aplikacji internetowych. C# znajduje również zastosowanie w popularnym silniku Unity, który jest wykorzystywany do tworzenia gier na PC, konsole i urządzenia mobilne.

Windows Presentation Foundation (WPF) znany wcześniej jako "Avalon" to graficzny podsystem (podobny do WinForms) stworzony do generowania interfejsu użytkownika

stworzony jako część .NET Framework #.0 w 2006 roku. WPF do renderownia elementów pulpitu w grafice wektorowej takich jak okna wykorzystuje bibliotekę Direc3D. Pozwala to na wyświetlanie bardziej złożonych elementów grafiki z wykorzystaniem procesorów graficznych (GPU) jednocześnie zmniejszając obciążenie procesora komputera (CPU). Rezultatem tego jest szybsze odświeżanie ekranu co przekłada się znacząco na poprawę wydajności działania aplikacji graficznych oraz animacji.

Aplikacje tworzone przy pomocy WPF wykorzystują wzorzec model-view-view-model (MVVM) w celu oddzielenia logiki biznesowej (tworzonej w c#) od widoku i składają się z trzech kluczowych elementów:

- warstwa widoku (plik z kodem XAML przypominającym strukturą HTML)

- warstwa kontrolera (code-behind – odpowiedzialnych za obsługę widoków)

- warstwa modelu (odpowiedzialna za realizację logiki biznesowej i przechowywanie danych)

Dzięki takiemu podejściu możliwe jest tworzenie aplikacji desktopowych przez projektantów nie będących ekspertami od C#. Tworzenie aplikacji w WPF polega podobnie jak w Swingu i Qt na budowaniu aplikacji z „cegiełek" tak zwanych kontrolek (Button, Label, CheckBox) i umieszczaniu ich w oknie (Window). Dodatkowo do dyspozycji mamy pojemniki służące do grupowania kontrolek (Border, Grid, StackPanel, DockPanel). Następnym udogodnieniem jest wprowadzenie styli, które pozwalają na definiowanie kolorów, czcionek, wyrównania i innych graficznych atrybutów dla wszystkich kontrolek co pozwala zaoszczędzić czas kiedy zaistnieje potrzeba zmiany wyglądu aplikacji.

3.3. Podsumowanie

Podsumowując wyżej wymienione technologie można dojść do wniosku, że każda z nich doskonale nadaję się do stworzenia symulatora rzeczywistych rozgrywek ligowych. Różnice pomiędzy samą składnią języków nie są duże (zwłaszcza między C# a Javą), jak również wytwarzanie samych aplikacji graficznych podlega podobnym zasadom. W pracy zawodowej jako programista do wytwarzania aplikacji używam języka C# i technologii .NET, która doskonale sprawdza się w rozwiązaniach biznesowych. Technologie z wiązane z Javą używałem sporadycznie w większości przypadków jako projekty uczelniane. Język C++ to mój pierwszy język obiektowy. Jest językiem najtrudniejszym z wszystkich trzech a jego główną wadą albo zaletą jest brak automatycznego zarządzania pamięcią. Sprawia to wiele trudności na początku jednakże opanowanie tej umiejętność pozwala na wytwarzanie aplikacji dużo wydajniejszych niż w Javie czy C#. Dodatkowo framework w postaci Qt z intuicyjnym QtCreatorem i znakomitą dokumentacją sprawił że został on wybrany do stworzenia tego projektu.

4. Projekt Systemu

W tym rozdziale przedstawię projekt systemu jego mechanikę działania oraz tworzenie struktury aplikacji. Zaprezentowane zostaną wszelkie wykorzystane mechanizmy, które umożliwiają działanie programu jak również funkcje i logika odpowiedzialna za przebieg meczów oraz całych rozgrywek.

4.1. Pomocne Narzędzia

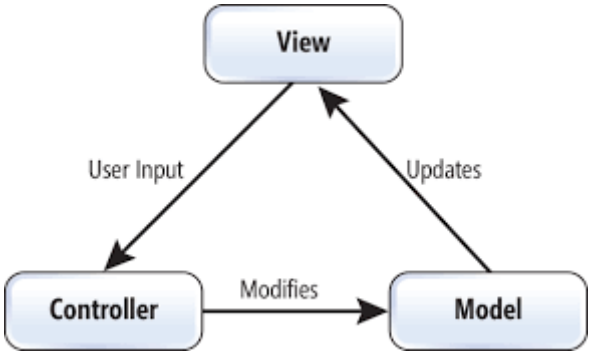
Zanim przejdę do omawiania funkcji sterujących aplikacją muszę objaśnić kilka zagadnień bez których poznania ciężko będzie zrozumieć strukturę oraz mechanizmy które są odpowiedzialne za działanie systemu.

Cały projekt został stworzony przy pomocy Qt Creatora – zintegrowanego środowiska programistycznego (IDE) działającego na różnych platformach, stworzonego z myślą o programistach piszących aplikacje w języku C++ korzystające z Qt. Składa się on z edytora

kodu w C++ z funkcją uzupełniania kodu, komunikatami o błędach oraz ostrzeżeniami podczas pisania i narzędziami do szybkiej nawigacji w strukturze projektu. Dodatkowo środowisko zostało wzbogacone o Qt Designer – narzędzie do tworzenia formularzy metodą „przeciągnij i upuść". Qt Creator oferuje wizualny debuger , narzędzia do budowania projektów i zarządzania nimi, a także możliwość generowania kodu.

4.1.1. Wykorzystanie Wzorca MVC

Framework Qt oparty jest na wzorcu „model-widok-kontroler" (MVC), który ułatwia stworzenie aplikacji, gdzie klasy danych(model) są odseparowane od klas związanych z prezentacją interfejsu użytkownika (widok). Kontroler jest odpowiedzialny za synchronizację danych z widoku do modelu. Kiedy użytkownik dokonując jakiejś zmiany na widoku np. naciska przycisk następuje wywołanie odpowiedniej metody w kontrolerze, która wprowadza zmiany w modelu. Kontroler pełni rolę pośrednika pomiędzy widokiem a modelem.



Rys. Wzorzec MVC

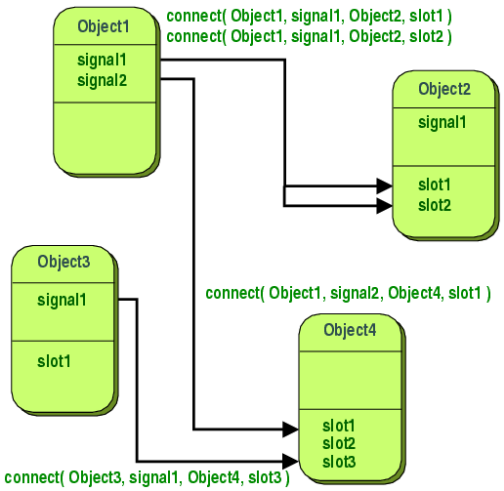
Takie podejście wymusza na programiście tworzeni osobnych klas odpowiedzialnych za model danych oraz klas odpowiedzialnych za prezentację widoku. Oddzielenie modelu od widoku zmniejsza złożoność każdego z tych elementów. Model i widok utrzymywane są w zupełnie inny sposób – zmiany są powodowane przez kompletnie inne czynniki dlatego łatwiej nad nimi panować, gdy nie są połączone. Kolejna zaleta separacji to możliwość stworzenia kilku różnych, spójnych widoków na te same dane.

„Architektura MVC obejmuje obiekty trzech rodzajów. Model to obiekt aplikacji, View odpowiada prezentacji widocznej na ekranie, a Controller określa, jak interfejs ma reagować na działanie użytkownika. Przed wprowadzeniem architektury MVC w interfejsach użytkownika wszystkie te obiekty były zwykle połączone ze sobą. MVC rozdziela je co, zwiększ elastyczność i możliwości powtórnego wykorzystania projektu." – Banda Czterech [Gamma95].

4.1.2. Sygnały i Sloty

Sygnały i sloty służą do komunikacji między obiektami. Mechanizm sygnałów i slotów (gniazd) jest centralną cechą Qt i prawdopodobnie częścią, która najbardziej różni się od funkcji zapewnianych przez inne frameworki. Sygnały i sloty są możliwe dzięki systemowi metaobiektu Qt.

Sygnał to komunikat widoczny w definicji klasy jako deklaracja pustej funkcji void. Ma on listę parametrów, ale nie ma ciała. Sygnał jest częścią interfejsu klasy. Wygląda jak funkcja, ale jest wywoływany w inny sposób: jest emitowany przez obiekt danej klasy.



Rys. Sygnały i Sloty

Slot również najczęściej jest pustą funkcją składową. Można ją wywołać w tradycyjny sposób lub niebezpośrednio przez system QMetaObject.

Sygnał jednego obiektu może zostać podłączony do slotów jednego lub więcej obiektów, o ile akie obiekty istnieją, a lista parametrów sygnału jest kompatybilna z listą slotu. Składnia instrukcji connect (podłącz) jest następująca:

```
connect(ui->down, SIGNAL(clicked(bool)), this, SLOT(volumeDown()));
```

Pierwszy parametr to wskaźnik na wysyłający obiekt, drugi nazwa sygnału z opcjonalną listą argumentów, trzeci wskaźnik na odbierający obiekt, czwarty nazwa slotu z opcjonalną listą argumentów.

4.1.3. Inteligentne wskaźniki

Jeżyk C++ nie posiada automatycznego mechanizmu odśmiecania pamięci (tzw. Garbage Collection), jednak istnieje kilka sposobów na automatyzację zarządzania pamięcią zajmowaną przez obiekty. Do najważniejszych takich sposobów należy zaliczyć inteligentne wskaźniki oraz zliczanie referencji. Qt posiada wiele typów inteligentnych wskaźników, stworzonych do realizacji różnych celów.

Klasę nazywamy inteligentnym wskaźnikiem, jeśli przesłania operator()* i operator ->(), czyli operatory wyłuskania wskaźnika. Pozwala to instancjom zachowywać się jak wskaźniki wbudowane. Klasy te prawie zawsze są szablonami, więc w definicjach musimy podawać odpowiednie typy w postaci argumentów szablonowych. Najłatwiej znaleźć przesłonięte wersje tych operatorów w iteratorach i właśnie w inteligentnych wskaźnikach. Nazwa „inteligentne”

wynika z niestandardowego zachowania się wskaźników podczas konstrukcji , destrukcji i przypisywaniu.

Klasa QScopedPointer to inteligentny wskaźnik, który automatycznie usuwa obiekt docelowy wskaźnika, gdy tylko wskaźnik wyjdzie poza zakres. Przypomina to std::auto_ptr. Jest to wskaźnik który zezwala na posiadanie tylko jednego egzemplarza wskaźnika na dany obiekt. Zakres wskaźnika wyraźnie zaznacza czas życia właściciela obiektu.

Klasa QSharedPointer również jest inteligentnym wskaźnikiem usuwający obiekt docelowy, jednak dozwolone jest tworzenie kopii, a QSharedPointer przechowuje licznik referencji. Współdzielony obiekt na stercie jest niszczony dopiero wtedy, gdy zniszczony zostanie ostatni wskaźnik na niego

4.1.4 QStackedWidget

Klasa QStackedWidget wykorzystywana jest do budowani stosu widżetów (widoków), w którym jednocześnie widoczny jest tylko jeden widżet. QStackedWidget może służyć do tworzenia interfejsu użytkownika podobnego do tego zapewnianego przez QTabWidget. Jest to bardzo wygody widget układu layoutów zbudowany na podstawie klasy QStackedLayout. Podobnie jak QStackedLayout, QStackedWidget można zbudować i wypełnić wieloma podrzędnymi widżetami „stronami”

Podczas wypełniania stosu widżetów widżety są dodawane do wewnętrznej listy. Funkcja indexOf () zwraca indeks widżetu na tej liście. Widżety można dodać na końcu listy za pomocą funkcji addWidget() lub wstawić pod dany indeks za pomocą funkcji insertWidget (). Funkcja removeWidget() usuwa widżet ze stosu widżetów. Liczbę widżetów zawartych w ułożonym stosie widżetów można uzyskać za pomocą funkcji count().

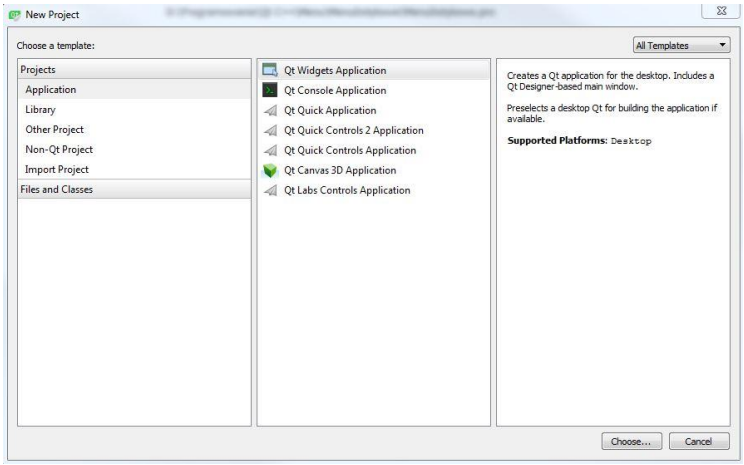
Funkcja widget () zwraca widget w danej pozycji indeksu. Indeks widżetu wyświetlany na ekranie jest podawany przez currentIndex () i można go zmienić za pomocą setCurrentIndex (). W podobny sposób aktualnie pokazany widget można pobrać za pomocą funkcji currentWidget () i zmienić za pomocą funkcji setCurrentWidget (). Ilekroć bieżący widget w widgecie ułożonym w stos zmienia się lub widżet jest usuwany ze stosu, emitowane są odpowiednio sygnały currentChanged () i widgetRemoved ().

4.2. Tworzenie Projektu

Jak wspomniana wcześniej framework Qt oparty jest na wzorcu „model-widok-kontroler” co w naturalny sposób wymusiło podział projektu na klasy danych(model) oraz klas związanych z prezentacją interfejsu użytkownika (widok).

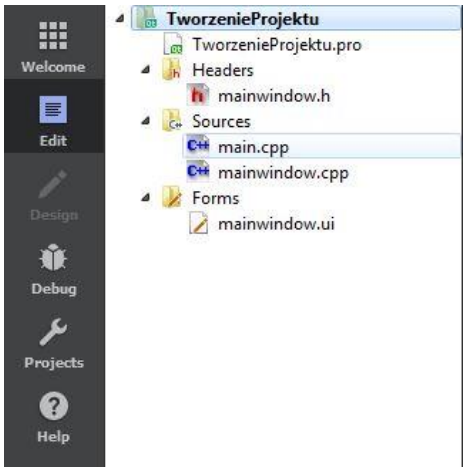
4.2.1. Nowy Projekt

Po uruchomieniu Qt Creatora i wybraniu opcji New Project dostajemy do dyspozycji okno dialogowe , w którym mamy do wyboru szablon projektu w jakim będziemy rozwijać naszą aplikację. W tum wypadku jest to Aplikations - Qt Widgets Application.



Rys. Nowy projekt

Następnie w celu konfiguracji projektu będziemy musieli przejść przez kilka okien dialogowych. Po nadaniu nazwy dla projektu, wybranie odpowiednich tzw. „Kit Selections", typu okna startowego, jego nazwy oraz skonfigurowaniu repozytorium dostajemy gotowy projekt startowy.



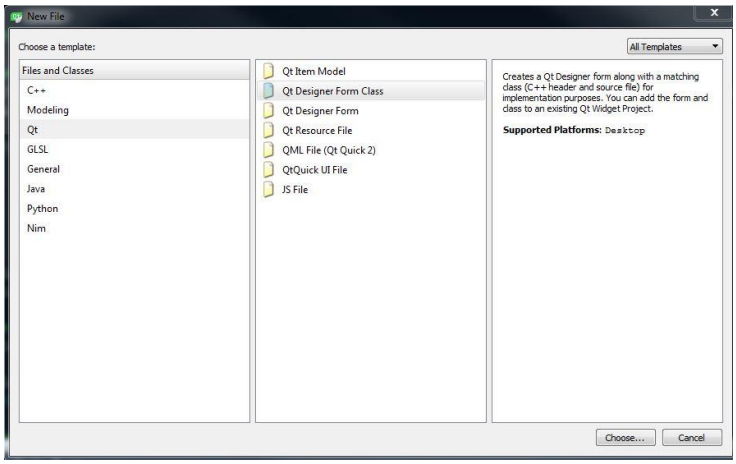
Rys. Struktura Projektu Startowego

Projekt składa się z pliku z rozszerzeniem .pro gdzie przechowywane są informacje o wersji Qt, jakie moduły zostały dołączone do projektu, spis plików źródłowych (.cpp), nagłówkowych (.h) oraz form (.ui). Poniżej znajdują się trzy katalogi, które służą do przechowywania trzech wyżej wymienionych plików. Plikiem startowym całej aplikacji jest main.cpp to w nim wywoływane jest okno startowe aplikacji MainWindow, które posiada wskaźnik na

niewidoczny w projekcie plik ui_mainwindow.h. Jest to plik generowany automatycznie przez Qt Creator i zawiera kod odpowiadający wszystkim elementom, które tworzymy na formie mainwindow.ui. Plik ui_mainwindow.h zawiera metodę setupUI(QMainWindow *MainWindow), która pobiera wskaźnik na obiekt mainwindow, żeby ustawić wszystkie elementy formy na layoucie okna głównego. Do tak skonfigurowanego projektu możemy dodawać kolejne okna dialogowe jak również klasy C++ odpowiedzialne za operacje na modelach danych.

4.2.2. Klasy Widoku

Klasy widoku nie są widoczne w samym projekcie, generowane są automatycznie i zapisywane w osobnym katalogu. Dostęp do widoków możemy uzyskać poprzez kliknięcie na odpowiedni plik w katalogu Forms. Uzyskujemy w ten sposób dostęp do narzędzia zwanego Qt Designer, który umożliwia dodawanie do formy layoutów, przycisków, kontenerów, kontrolek odpowiadających za wprowadzanie i wyświetlanie danych oraz ich dowolną konfigurację i stylizację.



Rys. Dodawanie Formy do projektu

Projekt można rozbudować o dowolną ilość form. Po wygenerowaniu nowej formy Qt Creator automatycznie generuje również plik nagłówkowy (.h) oraz plik źródłowy (.cpp) i zapisuje w odpowiednich katalogach.

4.2.3. Klasy Modelu

Klasy modelów odpowiedzialne za manipulowaniem na danych tworzone są w podobny sposób jak klasy widoków podczas dodawania formy do projektu. Różnica polega na tym, że

zamiast szablonu Qt wybieramy C++. W ten sposób tworzymy klasy modelów, które zapisywane są odpowiednio do katalogów Headers i Source.

4.3. Budowa i Struktura Aplikacji

W poprzednim podrozdziale zostało opisane ogólne tworzenie nowego projektu oraz podział plików i katalogów projektu w zależności od pełnionej roli. W tej części skupię się na pokazaniu jak zbudowany jest ten konkretny projekt. Jakie zależności i interakcje zachodzą pomiędzy klasami widoków i modelów.

4.3.1. Start Aplikacji

Plikiem startowym całej aplikacji jest plik main.cpp. Plik ten znajduje się w katalogu Sources nie posiada on pliku nagłówkowego i zawiera funkcjęint main(), która jest główną funkcją całej aplikacji. Funkcja main wywołuje obiekt klasy QApplication, która zarządza przepływem sterowania i głównymi ustawieniami aplikacji GUI. Dla każdej aplikacji GUI używającej Qt istnieje dokładnie jeden obiekt QApplication, bez względu na to czy aplikacja ma jedno czy nawet kilka okien otwartych w danym momencie.

```
#include "intro.h"
#include "container.h"
#include <QApplication>
#include <QSharedPointer>

QSharedPointer<Container>container(new Container());

int main(int argc, char *argv[])
{
    QCoreApplication::addLibraryPath(".");

    QTime time = QTime::currentTime();
    qsrand((uint)time.msec());

    QApplication a(argc, argv);
    container->loadDefaultData();

    a.setWindowIcon(QIcon("Button/ball.png"));
    Intro w;
    w.show();
    return a.exec();
}
```

Rys. 43 Plik main.cpp

Dodatkowo funkcja int main() zawiera klasę QTime, która zapewnia funkcję zegara Obiekt QTime zawiera czas zegarowy, tj. Liczbę godzin, minut, sekund i milisekund od północy. Może odczytywać aktualny czas z zegara systemowego i mierzyć przedział czasu, który upłynął. Zapewnia funkcje porównywania czasów i manipulowania czasem poprzez dodanie liczby milisekund. QCoreApplication::addLibraryPath(".") zwraca ścieżkę do pliku wykonywalnego aplikacji. Linia kodu a.setWindowIcon(QIcon("Button/ball.png")); ustawia ikonę okna głównego aplikacji. Następnie wywoływany jest obiekt klasy Intro poprzez metodę show() i następuje uruchomienie GUI widoku intro.ui.

Widok intro.ui jest widokiem początkowym, który po naciśnięciu przycisku przenosi nas do widoku mainmenu.ui, który jest z kolei głównym widokiem całej aplikacji. Przejście z widoku intro.ui do maimenu.ui odbywa się poprzez utworzenie obiektu klasy MainMenu, następnie wywoływana jest metoda statyczna klasy Intro, która zamyka widok intro.ui tak aby na koniec wywołać obiekt klasy MainMenu zawierający widok mainmenu.ui.

```
MainMenu mainmenu;
Intro::close();
mainmenu.exec();
```

.

Rys. Uruchomienie widoku MainMenu

Takie podejście w uruchomianiu okien nie jest najlepszym sposobem. Sprawdza się podczas uruchamiania okien modalnych, które pojawiają się na wierzchu okna macierzystego a następnie po wykonaniu potrzebnych operacji jest zamykane. Kiedy zależy nam na płynnym przemieszczaniu między pełnoekranowymi widokami lepszym sposobem jest wykorzystanie klasy QStackedWidget oraz sygnałów i slotów do przemieszczania się między widokami.

4.3.2. QStackedWidget, Sygnały i Sloty w praktyce.

MainMenu jako widok główny aplikacji zawiera w swojej strukturze odwołania pośrednie lub bezpośrednie do pozostałych widoków aplikacji. Bezpośrednio przy pomocy przycisków umieszczonych na górze ekranu możemy przejść do takich widoków jak „Wybór Drużyny”, „Menu Gry” , „Menu Opcji” czy „Abaut”. Koncepcja taka polega na tym, że w obiekcie klasy MainMenu (właściwie jego widoku Ui_mainMenu) tworzony jest obiekt klasy QstackedWidged do którego dodawany jest w indeksie zerowym layout widoku ‘MainMenu’. Następnie na kolejnych indeksach dodawane są kolejne widoki (‘About’, ‘Opcji’, ‘Wyboru Drużyny’, ‘Menu Gry’). W ten sposób otrzymujemy kontener, który przechowuje wskaźniki lub referencje do odpowiednich obiektów.

```
QStackedWidget *stackedWidget = new QStackedWidget(mainMenu);
stackedWidget->setCurrentIndex(0);
```

Rys. Utworzenie obiektu klasy QstackedWidged w Klasie Ui_MainMenu

```
ui->stackedWidget->insertWidget(1, &about);
ui->stackedWidget->insertWidget(2, &options);
ui->stackedWidget->insertWidget(3, choseTeam);
ui->stackedWidget->insertWidget(4, userDataDialog);
ui->stackedWidget->insertWidget(5, gameDialog);
```

Rys.Dodawanie obiektów do QstackedWidged

Przemieszczanie pomiędzy widokami następuję po naciśnięciu przycisku znajdującego się na górze ekranu. Przykładowo po wciśnięciu przycisku ‘Nowa Gra’ wywoływana jest metoda setCurrentIndex() obiektu stackedWidget – z parametrem 3, co oznacza ustawienie widoku ‘Wybór Drużyny ’ jako głównego layoutu. Dodatkowo wywoływana jest funkcja obiektu container, która zmienia tło widoku. (klasa Container zostanie opisana później) .

```
QPixmap bkgnd("Stadiony/realfuture.jpg");
container->functions->setBackground(this, bkgnd);
 ui->stackedWidget->setCurrentIndex(3);
```

Rys. Zmiana wyświetlanego layoutu oraz jego tła.

Możliwe jest również utworzenie kolejnego kontenera widoków na nowym layoucie. Takie rozwiązanie zastosowane jest w widoku ‘Opcje’, gdzie utworzony jest nowy niezależny kontener klasy QStackedWidget do którego dodawane są kolejne referencje do innych widoków. Takie rozwiązanie odciąża główny widok od zbytniej odpowiedzialności oraz porządkuje kod.

Kolejnym rozwiązaniem jest zastosowanie ‘Sygnałów i Slotów’ jest to mechanizm służący do komunikacji między obiektami. Przykładowym zastosowaniem tej funkcjonalności może być sytuacja kiedy znajdujemy się w widoku ‘Menu Gry’, który nie posiada przycisków widoku ‘Menu Głowne’ i chcemy dostać się do widoku ‘Opcji’. W takim wypadku wymagane jest kilka kroków które umożliwią komunikację między wyżej wymienionymi widokami.

```
connect(gameDialog, SIGNAL(dialogClicked()), this, SLOT(bakToMainMenu()));
```

Rys. Utworzenie połączenia pomiędzy widokami.

W pierwszym kroku tworzone jest w klasie ‘MainMenu’ połączenie pomiędzy widokiem ‘Menu Gry’ (gameDialog) a ‘Menu Głowne’ (this). Następnie tworzona jest metoda backToMainMenu() w której za pomocą metody setCurentIndex() klasy QstackedWidget ustawiany jest indeks na pozycje zero (‘Menu Główne’) . Jest to tak zwana funkcja powrotu.

```
signals:
    void dialogClicked();
```

Rys. Ustawienie sygnału w klasie ‘GameDialog’

W kolejnym kroku w klasie ‘GameDialog’ ustawiany jest sygnał dialogClicked(), tak aby po naciśnięciu przycisku ‘Menu Główne’ została wywołana metoda on_mainmenu_clicked(), która wyemituje sygnał dialogClicked(). Wówczas uruchamiana jest funkcja connect, która uruchamia sygnał wywołujący slot, który zawiera metodę backToMainMenu().

```
void GameDialog::on_mainmenu_clicked()
{
    emit dialogClicked();

    container->music_player->stop();
    container->loadDefaultData();
}
```

Rys. Wywołanie sygnału dialogClicked()

W tym podrozdziale zostało zaprezentowane działanie w praktyce wcześniej omówionych mechanizmów takich jak QStackedWidget oraz Sygnały i Sloty. Takie podejście zastosowane jest w całym projekcie zapewniając płynniejsze działanie aplikacji, lepsze zarządzanie pamięciom oraz poprawia czytelność kodu.

4.4. Sterowanie Aplikacją

Za sterownie logiką aplikacji odpowiadają klasy modelów, które stworzone zostały w odróżnieniu od klas kontrolerów jako czyste klasy języka C++. Są to klasy które kontrolują między innymi przebieg meczu, serializację danych czy wyliczają parametry zawodników oraz drużyn.

4.4.1. Klasy C++ vs Qt

4.4.2. Klasa Container

Klasa ‘Container’ inicjalizowana jest jako obiekt globalny przy pomocy inteligentnego wskaźnika w pliku main.cpp [Rys.77]. Zapewnia to szybki dostęp do jej składowej z każdego miejsca w programie wystarczy wywołac ją w pliku .cpp (extern Container *container). Klasa ‘Container’ spełnia w aplikacji dwie funkcje. Pierwszą z nich jest przechowywanie wskaźników na inne klasy modelów. Jest to bardzo wygodny sposób ułatwiający szybki dostęp do wszystkich funkcji aplikacji. Drugą funkcjonalność zapewniają trzy metody: loadDefaultData(), saveData(), loadData().

```
class Container
{

public:
    Container();
    ~Container();
    QVector<QVector<Player>> player;
    QList<Team> teams;
    League *league;
    MusicPlayer *musicPlayer;
    Functions *functions;
    MatchSimulation *matchSimulation;
    Serialization *serialization;
    Versus *versus;
    ShowTeamStats *showTeamStats;
    UserData *userData;
    ArtificialIntelligence *artificialInteligence;
    MatchAlgorithms *matchAlgoritm;
    Hovered *hovered;
    Formations *formations;
    SetMultimedia *setMultimedia;

    void loadDefaultData();
    void saveData();
    void loadData();
    void memeoryHarvester();

};
```

Rys. Klasa ‘Container’

Metoda loadDefaultData() wywoływana jest przy starcie aplikacji [Rys.77]. Jej głównym zadaniem jest utworzenie wszystkich obiektów wymaganych do prawidłowego działania systemu. Obiekty te odpowiadają za wypełnienie danymi początkowymi wszystkich drużyn oraz zawodników. Natomiast metody saveData() oraz loadData() wywoływane są już w trakcie działania programu przez użytkownika w celu zapisania stanu aplikacji lub przy ponownym uruchomieniu do przywrócenia zapisanego stanu.

4.8. Koncepcja Systemu

Głównymi aspektami, które były brane pod uwagę podczas opracowania systemu były przede wszystkim wady spotkane w innych podobnych projektach poddawanych analizie. Mówiąc wady nie należy tego rozumieć jako wady w samym oprogramowaniu i jego złym funkcjonowaniu. Należy to rozumieć jako inne założenia koncepcyjne systemu, które w tym przypadku dotyczą samego systemu rozgrywek.

4.8.1. Specyfikacja wymagań systemu

Specyfikacja wymagań jest niezbędnym elementem podczas tworzenia każdego systemu. Jest elementem pośrednim łączącym klienta zamawiającego system a jego wykonawcą. Dobrze napisana specyfikacja powinna zawierać możliwie jak najwięcej najważniejszych funkcjonalności systemu. Pozwoli to na uniknięcie nieporozumień pomiędzy klientem a wykonawcą wynikających z różnic jakie mogą pojawić się w finalnym produkcie. Ma też ona duży wpływ na czas w jakim zostanie wykonane zadanie co przekłada się na koszty projektu.

4.8.2. Wymagania funkcjonalne

Wymagania funkcjonalne powinny zawierać opis funkcji jakie zostały zastosowane do zbudowania systemu w celu szybkiego zidentyfikowania jego zachowania. Jest to istotny element, który umożliwia szybką reakcję w razie usterki poprzez szybkie odnalezienie źródła potencjalnego problemu i usunięcia go.

4.8.3. Wymagania niefunkcjonalne

4.8.3. Diagram przypadków użycia

4.8.5. Diagram klas UML

4.9. Wzorce Projektowe

4.9.1. Model-View-Controler (MVC)
4.9.2. Model Fabryka
4.9.3. Model tabel
4.9.4. Inteligentne wskaźniki

5. Prezentacja Projektu

Podczas projektowania aplikacji przyjęto jej zgodność z systemem *„Winows 7”* oraz rozdzielczością Full Hd (1920 x 1080). W przypadku wykorzystywania innych systemów oraz rozdzielczości mogą wystąpić różnice w sposobie wyświetlania różnych elementów oraz czcionek.

5.1. Ekran Powitalny

Po uruchomieniu aplikacji pojawia się *„Ekran Powitalny”* z jednym przyciskiem na środku w dolnej części ekranu, Po naciśnięciu przycisku aplikacja przechodzi do *„Menu Startowego”*.

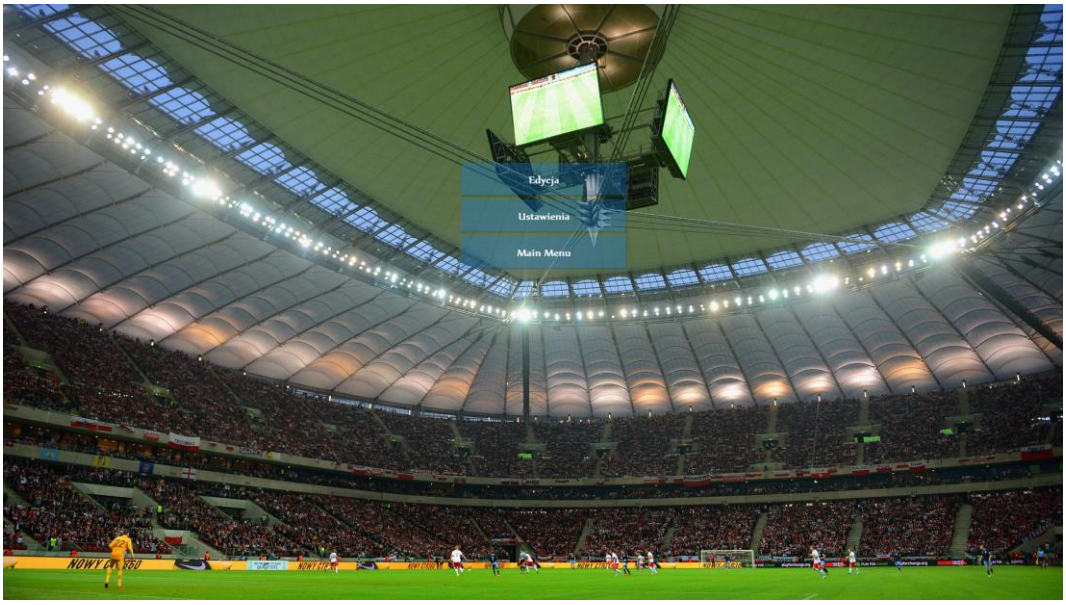
Rys. 15 Widok 'Ekran Powitalny'

5.2. Menu Startowe

„Menu Startowe", zawiera pięć przycisków: *„Nowa Gra"*, *„Kontynuacja"*, *„Opcje"*, *„About"* oraz „Koniec". Przycisk Nowa Gra uruchamia widok 'Wybór Drużyny'. „Kontynuacja" umożliwia wznowienia ostatniego zapisanego stanu gry. „Opcje' przechodzi do *„Menu Opcji"*. *„About'* wyświetla informacje na temat aplikacji oraz jej twórcy. *„Koniec"* kończy działanie programu.

Rys. 16 Widok 'Menu Startowe'

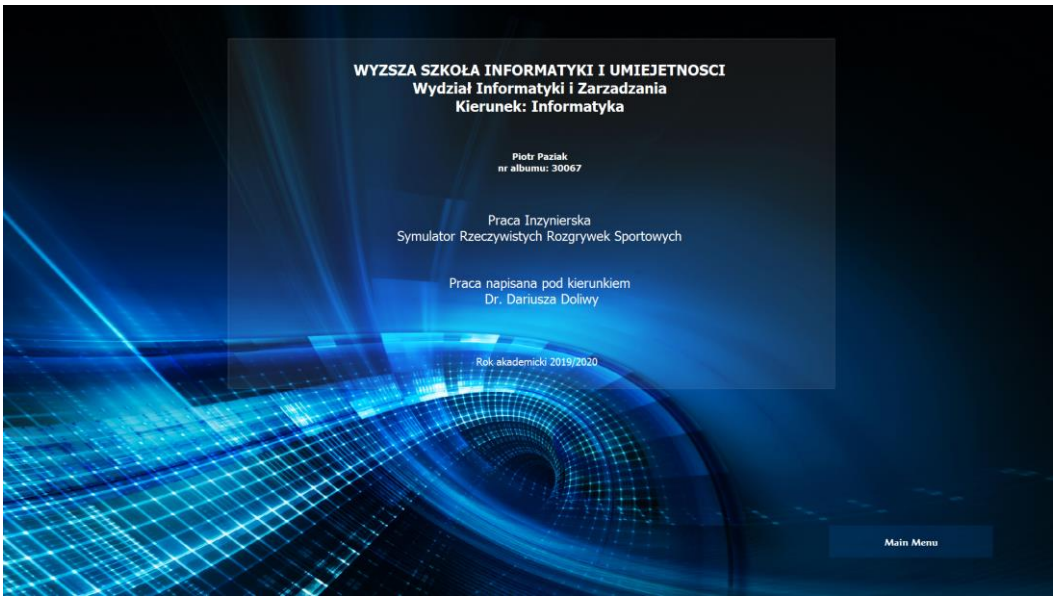
5.3. Menu Opcji



Rys. 17 'Menu Opcji'

5.4. About

Widok *„About"* jest widokiem informacyjnym zawierającym w centralnej części planszę informacyjną o autorze aplikacji, uczelni oraz promotorze. W prawym dolnym rogu znajduje się przycisk „Main Menu" umożliwiający cofnięcie się do „Menu Startowego".



Rys. 19 Widok 'About'

5.5. Wybór Drużyny

W centralnej części widoku *„Wybór drużyny”* znajduję się 16 herbów drużyn do wyboru. Po najechaniu kursorem na dany herb pojawiają się dwie tabelki. Tabela po lewej stronie zawiera listę zawodników wraz z ich średnią oceną umiejętności.

Rys. 20 Widok ‘Wybór Drużyny’

Tabela po prawej stronie zawiera herb drużyny oraz pięć elementów opisujących siłę drużyny: umiejętności bramkarza, obrony, pomocy, ataku oraz średnią ocenę całego zespołu. Po naciśnięciu herbu wybranej drużyny użytkownik przechodzi do ‘Menu Użytkownika’. W prawej dolnej części ekranu znajduje się przycisk *„Menu Główne”,* który umożliwia powrót do *„Menu Startowego”.*

5.5.1. Menu Użytkownika

„Menu Użytkownika” to okno modalne, które umożliwia użytkownikowi wprowadzenie informacji o sobie. Składa się ono z dwóch pól typu edit text umożliwiających podanie swojego imienia i nazwiska, combo box do ustalenia narodowości, spin box do podania wieku. Ponadto okno zawiera w centralnej części herb drużyny oraz dwa przyciski. Pierwszy *„Graj”* przenosi użytkownika do *„Menu Gry”*, drugi *„Wybór Drużyny”* cofa do widoku *„Wybór drużyny”*

Rys. 21 'Menu Użytkownika'

5.6. Menu Gry

5.7. Zarządzanie Drużyną

Rys. 21 'Zarządzanie Drużyną'