Milionerzy - Projekt Politechnika Lubelska

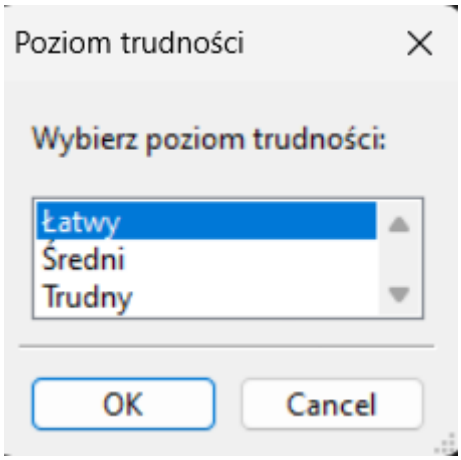
Mateusz Walo, Igor Kozak, Oskar Wilkos

Spis treści

1 Ogólny opis aplikacji

Aplikacja jest inspirowana popularną grą *„Milionerzy”* i umożliwia użytkownikowi odpowiadanie na pytania quizowe. Gra oferuje dwa koła ratunkowe: „50/50” i „Telefon do przyjaciela”. Rozgrywka kończy się, gdy użytkownik zdobędzie 1000 punktów, co oznacza zwycięstwo. Każda poprawna odpowiedź daje 100 punktów. Po zakończeniu gry, użytkownik może rozpocząć nową rundę. Aby zmienić poziom trudności, należy zamknąć grę i uruchomić ją ponownie.

2 Interfejs graficzny



Rysunek 1: Wybór poziomu trudności

Jak widać na rysunku 1, użytkownikowi prezentowane jest pierwsze okno aplikacji, w którym dokonuje się wyboru poziomu trudności gry. Interfejs został zaprojektowany tak, aby już na samym początku użytkownik mógł łatwo określić, jak wymagające mają być zadania, z jakimi będzie miał do czynienia. Każdy poziom trudności wiąże się z innym zestawem pytań, co umożliwia indywidualne dostosowanie rozgrywki do umiejętności gracza. Dzięki temu system wyboru poziomu, gra staje się bardziej dostępna – zarówno dla osób szukających mniej wymagającej zabawy, jak i dla tych, którzy pragną zmierzyć się z poważnym wyzwaniem.

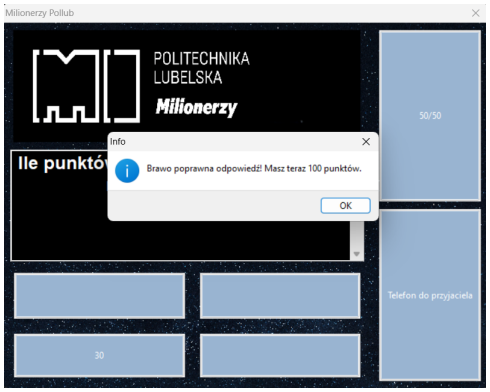
Rysunek 2: Interfejs startowy

Rysunek 2 ilustruje główne okno aplikacji, które pojawia się po wybraniu przez użytkownika odpowiedniego poziomu trudności. W tym widoku widoczne są cztery przyciski odpowiadające możliwym opcjom odpowiedzi, a także obszar z treścią aktualnie zadawanego pytania. Dodatkowo interfejs zawiera dwa elementy reprezentujące koła ratunkowe, które można wykorzystać w krytycznych momentach rozgrywki. Okno to nie tylko prezentuje estetyczny wygląd dzięki funkcji rysowania tła zaimplementowanej w metodzie `OnPaint`, ale również dynamicznie ładuje

i wyświetla pytania poprzez funkcję `DisplayQuestion`. Dzięki temu rozwiązaniu gra zyskuje na interaktywności i zapewnia płynne przechodzenie między kolejnymi pytaniami. Dodatkowo, dobrze dobrane elementy graficzne i przejrzysty układ pozwalają na intuicyjne poruszanie się po aplikacji, co pozytywnie wpływa na pierwsze wrażenie z użytkowania aplikacji.

Rysunek 3: Koło ratunkowe telefon do przyjaciela

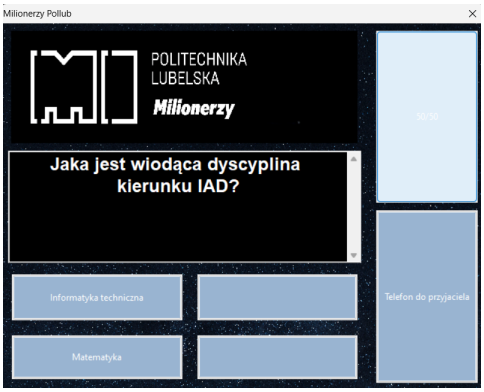
Na rysunku 3 przedstawiono moment, w którym użytkownik decyduje się na skorzystanie z koła ratunkowego „Telefon do przyjaciela". Po aktywacji tego elementu interfejs natychmiast reaguje, usuwając trzy niepoprawne odpowiedzi, co wyraźnie podkreśla istotę udzielanej pomocy. Dzięki takiemu rozwiązaniu, gracz może w krytycznej chwili otrzymać wskazówkę, która umożliwia mu podjęcie lepszej decyzji. Mechanizm ten został zaprojektowany z myślą o zwiększeniu interaktywności gry oraz jej atrakcyjności – możliwość skorzystania z „telefonu" dodaje element strategii, ponieważ gracz musi rozważnie wykorzystać tę szansę, wiedząc, że jest ona dostępna tylko raz w trakcie całej rozgrywki. Wizualne przedstawienie koła ratunkowego oraz natychmiastowa reakcja interfejsu podkreślają dynamikę i intuicyjność działania aplikacji.



Rysunek 4: Wybranie poprawnej odpowiedzi

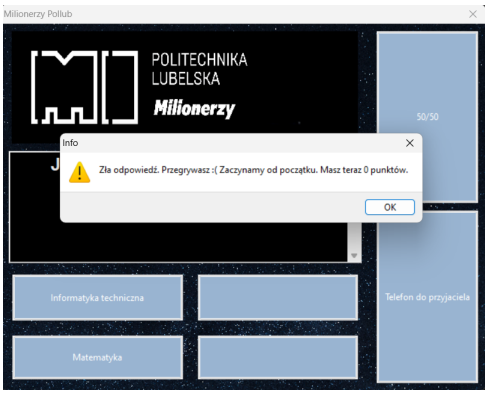
Rysunek 4 ukazuje kluczowy moment gry – wybór odpowiedzi przez użytkownika, która okazuje się być prawidłowa. Interfejs graficzny wyraźnie podkreśla zaznaczenie wybranej opcji oraz natychmiast informuje gracza o słuszności jego decyzji. Dodatkowo, system punktacji w czasie rzeczywistym dodaje 100 punktów za każdą poprawną odpowiedź, co jest wizualnie komuniko-

wane w formie okienka informacyjnego. Taki mechanizm nagradzania pozytywnie wpływa na motywację gracza, zachęcając go do kontynuacji rozgrywki. Dbałość o detale w interfejsie – takie jak wyraźne podświetlenie wybranej odpowiedzi i responsywność komunikatów – sprawia, że gra jest nie tylko interaktywna, ale również przyjazna dla użytkownika, co zwiększa ogólne zaangażowanie i satysfakcję z rozgrywki.



Rysunek 5: Koło ratunkowe 50/50

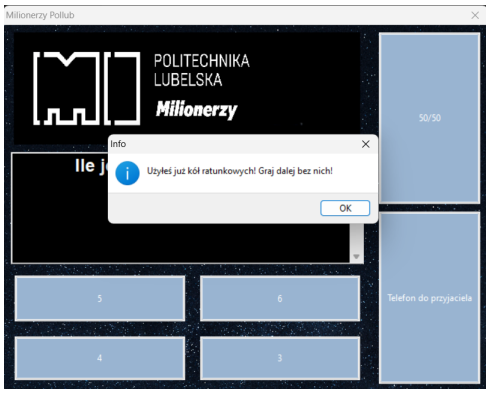
Na rysunku 5 widać zastosowanie drugiego z dostępnych kół ratunkowych – funkcji 50/50. Po aktywacji tej opcji, interfejs natychmiast usuwa dwie niepoprawne odpowiedzi, pozostawiając użytkownikowi jedynie jedną poprawną oraz jedną potencjalnie mylącą odpowiedź. Takie rozwiązanie znacząco zwiększa szansę na udzielenie właściwej odpowiedzi, szczególnie w sytuacjach, gdy gracz nie jest pewien swojej wiedzy. Wizualna reprezentacja efektu działania koła 50/50 jest intuicyjna – zmiana etykiet przycisków jasno komunikuje, które opcje zostały wyeliminowane. Ponadto, ograniczenie użycia tego koła do jednego razu na grę dodaje strategicznego wymiaru, zmuszając gracza do rozważnego podejmowania decyzji o jego użyciu.



Rysunek 6: Komunikat o złej odpowiedzi

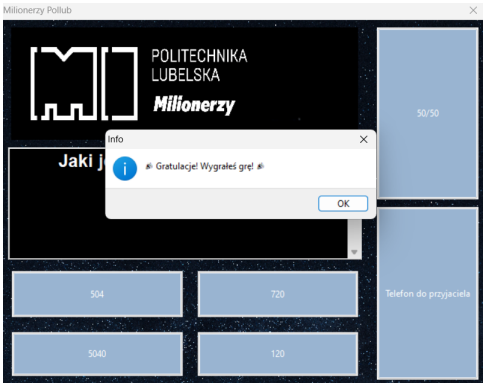
Rysunek 6 przedstawia sytuację, w której gracz udzielił błędnej odpowiedzi. Interfejs aplikacji reaguje natychmiast, wyświetlając komunikat ostrzegawczy, który informuje użytkownika o popełnionym błędzie. W wyniku niepoprawnej decyzji, liczba zdobytych punktów zostaje zresetowana do zera, a gra rozpoczyna się od nowa – ale na tym samym, wybranym wcześniej poziomie

trudności. Takie rozwiązanie ma na celu nie tylko zwiększenie realizmu gry, ale również zachęcenie użytkownika do ponownej analizy i lepszego przygotowania się do kolejnych rund. Jasny i czytelny komunikat sprawia, że gracz od razu wie, co poszło nie tak, a sam fakt resetu punktów motywuje do bardziej przemyślanych wyborów w przyszłości.



Rysunek 7: Komunikat o użyciu kół ratunkowych

Na rysunku 7 widzimy komunikat, który informuje gracza, iż wszystkie dostępne koła ratunkowe zostały już wykorzystane. Taki komunikat jest niezwykle ważny, ponieważ pełni funkcję informacyjną – użytkownik zostaje ostrzeżony, że nie będzie miał już możliwości skorzystania z dodatkowej pomocy przy rozwiązywaniu pytań. Dzięki temu gracz jest zmuszony do polegania wyłącznie na swojej wiedzy i intuicji w kolejnych etapach gry. Przejrzystość komunikatu oraz jego strategiczne umiejscowienie w interfejsie podkreślają dbałość o szczegóły, co dodatkowo zwiększa realizm i emocjonujący charakter rozgrywki. Informacja ta ma również walor motywacyjny – zmusza do bardziej przemyślanych decyzji, co wpływa na ogólne doświadczenie użytkownika.



Rysunek 8: Komunikat o wygranej

Rysunek 8 przedstawia finalny moment gry, kiedy gracz osiąga 1000 punktów dzięki serii poprawnych odpowiedzi. Na ekranie pojawia się komunikat gratulacyjny, który nie tylko informuje o zwycięstwie, ale także podkreśla satysfakcję i sukces osiągnięty przez użytkownika. Taki efekt końcowy jest kluczowy, gdyż stanowi nagrodę za wysiłek włożony w grę, a jednocześnie motywuje do powtórzenia rozgrywki lub podejmowania nowych wyzwań. Wizualne elementy komunikatu,

takie jak kolory, czcionki i układ, zostały starannie dobrane, aby podkreślić wyjątkowość tego momentu. Dzięki temu użytkownik doświadcza pozytywnych emocji, co wpływa na ogólną atrakcyjność aplikacji i zwiększa jej wartość rozrywkową.

3 Struktura kodu

3.1 Nagłówki i biblioteki

Kod rozpoczyna się od dołączenia wymaganych bibliotek oraz plików nagłówkowych. Główne użyte elementy to:

- `#include "ProjektMain.h"` - Plik nagłówkowy projektu, który zawiera deklaracje klas, funkcji oraz zmiennych specyficznych dla tego projektu. Dzięki temu możliwe jest oddzielenie interfejsu od implementacji oraz łatwiejsze zarządzanie kodem.
- `#include <wx/msgdlg.h>` - Biblioteka WX Widgets do obsługi okien dialogowych. Umożliwia wyświetlanie komunikatów, ostrzeżeń i informacji, co jest kluczowe dla interaktywności aplikacji oraz komunikacji z użytkownikiem.
- `#include <nlohmann/json.hpp>` - Biblioteka JSON, która umożliwia łatwe wczytywanie i przetwarzanie danych zapisanych w formacie JSON. Dzięki niej aplikacja może dynamicznie ładować bazę pytań z plików, co zwiększa elastyczność i skalowalność projektu.
- `#include <random>` - Biblioteka służąca do generowania losowych wartości. Jest wykorzystywana do losowania pytań oraz do mieszania kolejności odpowiedzi, co wprowadza element nieprzewidywalności i urozmaicenia w rozgrywce.
- `#include <algorithm>` - Biblioteka zawierająca metody i algorytmy, które są używane m.in. do losowego tasowania elementów, co jest kluczowe przy losowym wybieraniu pytań i odpowiedzi.

Struktura `Question` przechowuje dane pojedynczego pytania quizowego:

- `std::string tresc` - Zawiera treść pytania, którą użytkownik widzi na ekranie.
- `std::vector<std::string> odpowiedzi` - Lista możliwych odpowiedzi, z której użytkownik wybiera jedną.
- `std::string poprawna_odpowiedz` - Przechowuje poprawną odpowiedź, co jest wykorzystywane przy weryfikacji odpowiedzi udzielonej przez gracza.

3.2 Zmienne globalne

Następnie zdefiniowane są zmienne globalne używane w aplikacji:

- `std::vector<Question> questions` - Lista wszystkich pytań dostępnych w bazie. Pozwala na dynamiczne dodawanie lub usuwanie pytań, co ułatwia modyfikację zawartości quizu.
- `std::vector<bool> askedQuestions` - Tablica logiczna, która śledzi, które pytania zostały już zadane. Dzięki temu mechanizmowi unika się powtarzania pytań w jednej sesji gry.

- `size_t currentQuestionIndex` - Zmienna przechowująca indeks aktualnie wyświetlanego pytania. Umożliwia łatwe odwoływanie się do bieżącego elementu z listy pytań.
- `bool fiftyFiftyUsed, phoneAFriendUsed, fiftyFiftyUsedGlobal, phoneAFriendUsedGlobal` - Flagi służące do monitorowania użycia kół ratunkowych. Rozróżnienie na lokalne i globalne flagi pozwala na precyzyjne kontrolowanie dostępności opcji pomocy w trakcie rozgrywki.
- `int points` - Zmienna przechowująca aktualną liczbę punktów zdobytych przez użytkownika. Mechanizm punktacji jest kluczowy dla oceny postępów gracza oraz określania warunków zwycięstwa.

3.3 Funkcja `wxbuildinfo`

Funkcja `wxbuildinfo` zwraca informacje o kompilacji aplikacji:

- Argument `format` określa, czy zwracane informacje mają być w formacie krótkim, czy długim.
- Funkcja wykorzystuje makra preprocesora do dodania informacji o systemie operacyjnym i trybie kompilacji (Unicode/ANSI).

```
wxString wxbuildinfo(wxbuildinfoformat format)
{
 wxString wxbuild(wxVERSION_STRING);

 if (format == long_f)
 {
  #if defined(__WXMSW__)
  wxbuild << _T("-Windows");
  #elif defined(__UNIX__)
  wxbuild << _T("-Linux");
  #endif

  #if wxUSE_UNICODE
  wxbuild << _T("-Unicode build");
  #else
  wxbuild << _T("-ANSI build");
  #endif // wxUSE_UNICODE
 }

 return wxbuild;
}
```

3.4 Klasa `ProjektDialog`

Klasa `ProjektDialog` reprezentuje główne okno dialogowe aplikacji. To w niej zawarte są wszystkie elementy interfejsu użytkownika oraz logika sterująca rozgrywką.

3.4.1 Konstruktor i destruktor

Konstruktor klasy `ProjektDialog` odpowiada za inicjalizację interfejsu użytkownika oraz załadowanie pytań w oparciu o wybrany przez użytkownika poziom trudności. Inicjalizacja ta obejmuje ustawienie początkowych wartości zmiennych, konfigurację przycisków, pól tekstowych oraz innych elementów interfejsu. Destruktor natomiast jest pusty, co oznacza, że nie wykonuje dodatkowych operacji przy zamykaniu okna – wszystkie zasoby są zarządzane automatycznie lub przez system.

3.4.2 Metoda `OnPaint`

Metoda `OnPaint` odpowiada za rysowanie tła okna dialogowego. Wykorzystuje obiekt klasy `wxPaintDC`, który umożliwia rysowanie bezpośrednio na obszarze okna. Kluczowe etapy tej metody to:

- Inicjalizacja obiektu `wxPaintDC`, co pozwala na wykonywanie operacji graficznych.
- Wczytanie bitmapy tła z pliku (w tym przypadku `"resources/kosmos.jpg"`), co dodaje aplikacji atrakcyjny wygląd.
- Konwersja bitmapy na obiekt `wxImage` i przeskalowanie jej do aktualnego rozmiaru okna, dzięki czemu tło idealnie dopasowuje się do rozmiarów interfejsu.
- Rysowanie przeskalowanej bitmapy na oknie, a następnie wywołanie metody `event.Skip()`, aby umożliwić dalsze przetwarzanie zdarzenia.

```
void ProjektDialog::OnPaint(wxPaintEvent& event)
{
 wxPaintDC dc(this);

 wxBitmap backgroundBitmap(wxT("resources/kosmos.jpg"), wxBITMAP_TYPE_JPEG);

 wxImage scaledImage = backgroundBitmap.ConvertToImage();
 scaledImage = scaledImage.Scale(GetSize().GetWidth(), GetSize().GetHeight());

 dc.DrawBitmap(wxBitmap(scaledImage), 0, 0, true);

 event.Skip();
}
```

3.4.3 Metoda `DisplayQuestion`

Metoda `DisplayQuestion` jest odpowiedzialna za wyświetlenie aktualnego pytania oraz możliwych odpowiedzi na ekranie. W jej działaniu można wyróżnić kilka etapów:

- Sprawdzenie, czy lista pytań nie jest pusta. Jeśli nie ma żadnych pytań, zostaje wyświetlony komunikat błędu.
- Losowe wybranie pytania spośród dostępnych. Do tego celu wykorzystuje się generator liczb losowych, co pozwala na uniknięcie powtórzeń.

- Oznaczenie wybranego pytania jako zadane, co zapobiega jego ponownemu wyświetleniu w trakcie jednej rozgrywki.
- Losowe tasowanie kolejności odpowiedzi, aby gracz nie mógł zorientować się, która odpowiedź jest poprawna na podstawie pozycji.
- Ustawienie treści pytania oraz etykiet przycisków odpowiedzi przy użyciu metod odpowiednich dla kontrolki tekstowej oraz przycisków.

```
void ProjektDialog::DisplayQuestion()
{
 if (questions.empty())
 {
  wxMessageBox(_("Brak pytań! :( "), _("Error"), wxOK | wxICON_ERROR);
  return;
 }

 std::random_device rd;
 std::mt19937 gen(rd());
 std::uniform_int_distribution<> dis(0, questions.size() - 1);
 do {
  currentQuestionIndex = dis(gen);
 } while (askedQuestions[currentQuestionIndex]);

 askedQuestions[currentQuestionIndex] = true;
 Question& question = questions[currentQuestionIndex];

 std::vector<std::string> shuffledAnswers = question.odpowiedzi;

 std::shuffle(shuffledAnswers.begin(), shuffledAnswers.end(), gen);

 TextCtrl1->SetValue(wxString::FromUTF8(question.tresc));
 Button1->SetLabel(wxString::FromUTF8(shuffledAnswers[0]));
 Button2->SetLabel(wxString::FromUTF8(shuffledAnswers[1]));
 Button3->SetLabel(wxString::FromUTF8(shuffledAnswers[2]));
 Button4->SetLabel(wxString::FromUTF8(shuffledAnswers[3]));

 fiftyFiftyUsed = false;
 phoneAFriendUsed = false;
}
```

3.4.4 Metoda `LoadQuestionsFromFile`

Metoda `LoadQuestionsFromFile` odpowiada za wczytywanie pytań z pliku JSON. Umożliwia to dynamiczne ładowanie bazy pytań w zależności od wybranego poziomu trudności. Jej działanie obejmuje:

- Otwarcie pliku z bazą pytań przy użyciu `std::ifstream`.
- Wczytanie danych z pliku do obiektu typu `json`, co ułatwia późniejsze przetwarzanie danych.

- Wyczyść listę dotychczasowych pytań oraz stan zadanych pytań, aby przygotować przestrzeń na nowe dane.
- Iterację przez każdy element w sekcji "pytania" w pliku JSON, utworzenie obiektu `Question` oraz dodanie go do listy pytań.
- Jeśli plik nie może zostać otwarty, wyświetlenie komunikatu błędu, co informuje użytkownika o problemie z dostępem do bazy pytań.

```
void ProjektDialog::LoadQuestionsFromFile(const wxString& fileName)
{
 std::ifstream file(fileName.ToStdString());
 if (file.is_open())
 {
  json jsonData;
  file >> jsonData;
  questions.clear();
  askedQuestions.clear();
  for (const auto& item : jsonData["pytania"])
  {
   Question q;
   q.tresc = item["tresc"];
   q.odpowiedzi = item["odpowiedzi"].get<std::vector<std::string>>();
   q.poprawna_odpowiedz = item["poprawna_odpowiedz"];
   questions.push_back(q);
   askedQuestions.push_back(false);
  }
  file.close();
 }
 else
 {
  wxMessageBox(_("Nie można wczytać bazy pytań"), _("Error"), wxOK | wxICON_ERROR);
 }
}
```

3.4.5 Metoda `OnButtonClicked`

Metoda `OnButtonClicked` obsługuje interakcję użytkownika z przyciskami odpowiedzi. Odpowiada za weryfikację udzielonej odpowiedzi oraz aktualizację stanu gry:

- Pobiera identyfikator kliknięty przycisku oraz jego etykietę.
- Za pomocą funkcji `trimString` usuwa zbędne białe znaki, co zapewnia poprawne porównanie tekstów.
- Dokonuje porównania odpowiedzi użytkownika z poprawną odpowiedzią, przy czym ignorowana jest wielkość liter – dzięki temu system jest bardziej przyjazny.
- W przypadku udzielenia poprawnej odpowiedzi, metoda zwiększa liczbę punktów oraz wyświetla komunikat potwierdzający sukces. Po osiągnięciu 1000 punktów gra informuje o zwycięstwie, resetując jednocześnie stan gry.

- Jeśli odpowiedź jest błędna, liczba punktów jest resetowana do zera, a użytkownik otrzymuje odpowiedni komunikat ostrzegawczy. Dodatkowo resetowany jest stan kół ratunkowych, co umożliwia ponowne rozpoczęcie gry.

```
void ProjektDialog::OnButtonClicked(wxCommandEvent& event)
{
 int buttonId = event.GetId();
 std::string correctAnswer = trimString(questions[currentQuestionIndex].poprawna_odpowiedz);

 wxButton* clickedButton = dynamic_cast<wxButton*>(event.GetEventObject());
 if (clickedButton)
 {
  std::string userAnswer = trimString(clickedButton->GetLabel().ToStdString());
  std::transform(userAnswer.begin(), userAnswer.end(), userAnswer.begin(), ::tolower);
  std::transform(correctAnswer.begin(), correctAnswer.end(), correctAnswer.begin(), ::tolower);

  if (userAnswer == correctAnswer)
  {
   points += 100;
   wxMessageBox(wxString::Format(_("Brawo poprawna odpowiedź! Masz teraz %d punktów."), points)
   if (points >= 1000)
   {
    wxMessageBox(_("Gratulacje! Wygrałeś grę!"), _("Info"), wxOK | wxICON_INFORMATION);
    points = 0;
    std::fill(askedQuestions.begin(), askedQuestions.end(), false);
   }
   DisplayQuestion();
  }
  else
  {
   points = 0;
   wxMessageBox(wxString::Format(_("Zła odpowiedź. Przegrywasz :( Zaczynamy od początku. Masz t
   std::fill(askedQuestions.begin(), askedQuestions.end(), false);
   fiftyFiftyUsedGlobal = false;
   phoneAFriendUsedGlobal = false;
   DisplayQuestion();
  }
 }
}
```

3.4.6 Metoda `OnFiftyFifty`

Metoda `OnFiftyFifty` obsługuje aktywację koła ratunkowego 50/50, które ma na celu zwiększenie szans gracza na udzielenie poprawnej odpowiedzi poprzez eliminację dwóch błędnych opcji:

- Na początku metoda sprawdza, czy już użyto któregokolwiek z kół ratunkowych (50/50 lub Telefon do przyjaciela). Jeśli tak, wyświetlany jest komunikat informacyjny.
- Jeśli koło ratunkowe jest dostępne, ustawiane są flagi `fiftyFiftyUsed` oraz `fiftyFiftyUsedGlobal` na wartość `true`, aby zapobiec ponownemu użyciu tego wsparcia.

- Metoda pobiera poprawną odpowiedź oraz tworzy listę wszystkich przycisków reprezentujących opcje odpowiedzi.
- Następnie, przy użyciu pętli, wyłaniane są przyciski z niepoprawnymi odpowiedziami, które umieszczane są w osobnej liście.
- Lista niepoprawnych przycisków jest tasowana przy użyciu generatora liczb losowych, a następnie losowo usuwane są etykiety dwóch z nich, co wizualnie eliminuje te opcje.

```
void ProjektDialog::OnFiftyFifty(wxCommandEvent& event)
{
 if (fiftyFiftyUsed || fiftyFiftyUsedGlobal || phoneAFriendUsed)
 {
  wxMessageBox(_("Użyłeś już kół ratunkowych! Graj dalej bez nich!"), _("Info"), wxOK | wxICON_
  return;
 }

 fiftyFiftyUsed = true;
 fiftyFiftyUsedGlobal = true;

 std::string correctAnswer = questions[currentQuestionIndex].poprawna_odpowiedz;
 std::vector<wxButton*> buttons = {Button1, Button2, Button3, Button4};
 std::vector<wxButton*> incorrectButtons;

 for (auto button : buttons)
 {
  if (button->GetLabel().ToStdString() != correctAnswer)
  {
   incorrectButtons.push_back(button);
  }
 }

 std::random_device rd;
 std::mt19937 gen(rd());
 std::shuffle(incorrectButtons.begin(), incorrectButtons.end(), gen);
 incorrectButtons[0]->SetLabel("");
 incorrectButtons[1]->SetLabel("");
}
```

3.4.7 Metoda `OnPhoneAFriend`

Metoda `OnPhoneAFriend` obsługuje kliknięcie przycisku „Telefon do przyjaciela”, które polega na usunięciu trzech błędnych odpowiedzi z ekranu, zwiększając tym samym szansę na poprawne udzielenie odpowiedzi:

- Metoda najpierw sprawdza, czy już użyto któregokolwiek z kół ratunkowych – jeżeli tak, wyświetlany jest odpowiedni komunikat informacyjny.
- Jeżeli wsparcie jest dostępne, flagi `phoneAFriendUsed` i `phoneAFriendUsedGlobal` są ustawiane na `true`.

- Pobierana jest poprawna odpowiedź, a następnie tworzona jest lista przycisków zawierająca wszystkie możliwe opcje.
- Lista przycisków jest tasowana losowo, co zapobiega przewidywalności usuwania opcji.
- Następnie metoda przechodzi przez przyciski i usuwa etykiety trzech niepoprawnych opcji, co wizualnie eliminuje te odpowiedzi z interfejsu.

```
void ProjektDialog::OnPhoneAFriend(wxCommandEvent& event)
{
 if (phoneAFriendUsed || phoneAFriendUsedGlobal || fiftyFiftyUsed)
 {
  wxMessageBox(_("Użyłeś już kół ratunkowych! Graj dalej bez nich!"), _("Info"), wxOK | wxICON_
  return;
 }

 phoneAFriendUsed = true;
 phoneAFriendUsedGlobal = true;

 std::string correctAnswer = questions[currentQuestionIndex].poprawna_odpowiedz;
 std::vector<wxButton*> buttons = {Button1, Button2, Button3, Button4};

 std::random_device rd;
 std::mt19937 gen(rd());
 std::shuffle(buttons.begin(), buttons.end(), gen);

 int hiddenCount = 0;
 for (auto button : buttons)
 {
  if (button->GetLabel().ToStdString() != correctAnswer && hiddenCount < 3)
  {
   button->SetLabel("");
   hiddenCount++;
  }
 }
}
```

3.4.8 Metoda `OnQuit`

Metoda `OnQuit` odpowiada za zakończenie działania aplikacji. Jej zadanie polega na zamknięciu głównego okna dialogowego, co skutkuje zakończeniem całego procesu aplikacji:

```
void ProjektDialog::OnQuit(wxCommandEvent& event)
{
 Close();
}
```

- Wywołanie metody `Close()` zamyka okno dialogowe, co jest standardowym sposobem kończenia aplikacji w oparciu o WX Widgets.

3.4.9 Metoda `OnAbout`

Metoda `OnAbout` służy do wyświetlenia informacji o aplikacji, takich jak wersja, system operacyjny czy tryb kompilacji. Umożliwia to szybkie uzyskanie szczegółowych danych dotyczących środowiska uruchomieniowego:

```
void ProjektDialog::OnAbout(wxCommandEvent& event)
{
 wxString msg = wxbuildinfo(long_f);
 wxMessageBox(msg, _("Welcome to..."));
}
```

- Funkcja `wxbuildinfo(long_f)` pobiera szczegółowe informacje o kompilacji, a następnie te informacje są wyświetlane w oknie dialogowym za pomocą `wxMessageBox()`.

3.5 Funkcja `main`

Funkcja `main` jest punktem wejścia do aplikacji. Jej główne zadania to:

- Ustawienie instancji aplikacji WX Widgets, co jest niezbędne do zarządzania cyklem życia aplikacji.
- Uruchomienie głównej pętli aplikacji przy użyciu funkcji `wxEntry()`, która kontroluje wszystkie zdarzenia w aplikacji.

```
int main()
{
 wxApp::SetInstance(new ProjektApp);
 return wxEntry();
}
```

3.6 Dodatkowe metody

Oprócz głównych metod, klasa `ProjektDialog` zawiera również metody pomocnicze, które choć puste, mogą zostać wykorzystane w przyszłości do rozszerzenia funkcjonalności aplikacji:

- `OnInit` - Metoda wywoływana podczas tworzenia okna dialogowego. Obecnie jest pusta, ale może zostać rozszerzona o dodatkowe operacje inicjalizacyjne, które powinny zostać wykonane przy starcie aplikacji.

  ```
  void ProjektDialog::OnInit(wxInitDialogEvent& event)
  {
  }
  ```

- `OnTextCtrl1Text` - Metoda obsługująca zmiany tekstu w polu tekstowym. Chociaż aktualnie nie zawiera logiki, może być użyta w przyszłości do dynamicznej walidacji lub modyfikacji danych wprowadzanych przez użytkownika.

```
void ProjektDialog::OnTextCtrl1Text(wxCommandEvent& event)
{
}
```

4 Podział obowiązków

4.1 Mateusz Walo

- Implementacja logiki gry:
  - Opracowanie mechanizmu zadawania pytań i sprawdzania odpowiedzi.
  - Implementacja systemu punktacji oraz warunków wygranej i przegranej.
- Funkcje losujące pytania:
  - Implementacja funkcji losującej pytania z bazy danych.
  - Zapewnienie losowego wyboru pytań, z uwzględnieniem, aby nie powtarzały się w jednej grze.
- Projektowanie baz pytań na poziom średni i trudny:
  - Opracowanie pytań dostosowanych do średniego i trudnego poziomu trudności.
  - Weryfikacja poprawności pytań i odpowiedzi.
- Przygotowanie raportu LaTeXowego:
  - Opracowanie dokumentacji technicznej projektu w formacie LaTeX.
  - Redakcja i formatowanie raportu, w tym spis treści, sekcje techniczne i graficzne.
- Obsługa repozytorium na GitHub:
  - W celu ujednolicenia i kontroli wersji projektu.
  - Obsługa poprawek i wysyłanie commitów po aktualizacjach w celu spójności.

4.2 Igor Kozak

- Implementacja biblioteki do obsługi plików JSON:
  - Integracja biblioteki JSON do odczytu i zapisu danych.
  - Implementacja funkcji wczytujących pytania z plików JSON do aplikacji.
- Zaprojektowanie bazy danych na pierwszy poziom trudności:
  - Opracowanie pytań dla poziomu łatwego.
  - Weryfikacja poprawności pytań i odpowiedzi.
- Przygotowanie instalatora:
  - Opracowanie skryptu instalacyjnego dla aplikacji.
  - Testowanie procesu instalacji na różnych systemach operacyjnych.
- Zaktualizowanie ikonki:
  - Projektowanie nowej ikonki dla aplikacji.
  - Integracja ikonki z aplikacją.

4.3 Oskar Wilkos

- Zaprojektowanie interfejsu graficznego:
  - Opracowanie projektu interfejsu użytkownika (UI).
  - Implementacja interfejsu przy użyciu biblioteki WX Widgets.
- Implementacja okienka dialogowego do wyboru poziomu trudności:
  - Opracowanie okienka dialogowego umożliwiającego wybór poziomu trudności.
  - Integracja okienka z główną logiką gry.
- Przygotowanie plików graficznych i struktury projektowej:
  - Tworzenie i przygotowanie plików graficznych używanych w aplikacji.
  - Organizacja struktury katalogów projektu, w tym zasobów graficznych i plików konfiguracyjnych.

5 Podsumowanie

Aplikacja *Milionerzy* jest w pełni funkcjonalnym quizem inspirowanym popularnym teleturniejem. Dzięki możliwości wyboru poziomu trudności oraz użyciu kół ratunkowych, gra oferuje różnorodne wyzwania dla użytkowników. Implementacja opiera się na bibliotece WX Widgets, co pozwala na tworzenie interaktywnych i atrakcyjnych wizualnie interfejsów użytkownika. Kod jest dobrze zorganizowany i modularny, co ułatwia jego modyfikację i rozszerzenie o nowe funkcje w przyszłości.