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 Struktura kodu

2.1 Nagłówki i biblioteki

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

- `#include "ProjektMain.h"` - Plik nagłówkowy projektu.
- `#include <wx/msgdlg.h>` - Biblioteka WX Widgets do obsługi okien dialogowych.
- `#include <nlohmann/json.hpp>` - Biblioteka JSON do obsługi plików będących bazami pytań.
- `#include <random>` - Obsługa generowania losowych wartości.
- `#include <algorithm>` - Obsługa metod wykorzystanych do losowania pytań.

Struktura `Question` przechowuje dane pojedynczego pytania quizowego:

- `std::string tresc` - Treść pytania.
- `std::vector<std::string> odpowiedzi` - Lista możliwych odpowiedzi.
- `std::string poprawna_odpowiedz` - Poprawna odpowiedź.

2.2 Zmienne globalne

Następnie zdefiniowane są zmienne globalne używane w aplikacji:

- `std::vector<Question> questions` - Lista wszystkich pytań.
- `std::vector<bool> askedQuestions` - Informacja o tym, które pytania zostały już zadane.
- `size_t currentQuestionIndex` - Indeks bieżącego pytania.
- `bool fiftyFiftyUsed, phoneAFriendUsed, fiftyFiftyUsedGlobal, phoneAFriendUsedGlobal` - Flagi użycia kół ratunkowych.
- `int points` - Aktualna liczba punktów użytkownika.

2.3 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;
}
```

2.4 Klasa `ProjektDialog`

Klasa `ProjektDialog` reprezentuje główne okno dialogowe aplikacji:

2.4.1 Konstruktor i destruktor

Konstruktor inicjalizuje interfejs użytkownika oraz ładuje pytania na podstawie wybranego poziomu trudności. Destruktor jest pusty i nie zawiera żadnej dodatkowej logiki.

2.4.2 Metoda `OnPaint`

Metoda `OnPaint` odpowiada za rysowanie tła okna:

```
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();
}
```

2.4.3 Metoda `DisplayQuestion`

Metoda `DisplayQuestion` wyświetla bieżące pytanie:

```
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;
}
```

2.4.4 Metoda `LoadQuestionsFromFile`

Metoda `LoadQuestionsFromFile` ładuje pytania z pliku JSON. Jest to kluczowy element aplikacji, który pozwala na dynamiczne wczytywanie pytań na podstawie wybranego poziomu trudności.

```
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);
   }
  }
```

Opis:

- Otwiera plik JSON przy użyciu `std::ifstream`.
- Wczytuje dane z pliku do obiektu `json`.
- Czyści istniejącą listę pytań i stan zadanych pytań.
- Przechodzi przez każdy element w sekcji "pytania" pliku JSON, tworzy obiekt `Question` i dodaje go do listy pytań.
- Jeśli plik nie może zostać otwarty, wyświetla komunikat błędu.

2.4.5 Metoda `OnButtonClicked`

Metoda `OnButtonClicked` obsługuje kliknięcia przycisków odpowiedzi. Użytkownik wybiera odpowiedź, a metoda ta sprawdza jej poprawność.

```
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();
   }
  }
 }
```

Opis:

- Pobiera identyfikator kliknięty przycisku.
- Sprawdza, czy kliknięty przycisk jest poprawny i pobiera jego etykietę.
- Porównuje odpowiedź użytkownika z poprawną odpowiedzią (ignorując wielkość liter).
- Jeśli odpowiedź jest poprawna, zwiększa liczbę punktów i wyświetla komunikat o sukcesie. Jeśli użytkownik osiągnie 1000 punktów, resetuje grę.
- Jeśli odpowiedź jest błędna, resetuje liczbę punktów i wyświetla komunikat o błędzie. Resetuje również stan kół ratunkowych.

2.4.6 Metoda `OnFiftyFifty`

Metoda `OnFiftyFifty` obsługuje kliknięcie przycisku "50/50", który usuwa dwie niepoprawne odpowiedzi z ekranu.

```
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("");
}
```

Opis:

- Sprawdza, czy koło ratunkowe "50/50" lub "Telefon do przyjaciela" zostało już użyte.
- Ustawia flagi `fiftyFiftyUsed` i `fiftyFiftyUsedGlobal` na `true`.
- Pobiera poprawną odpowiedź i tworzy listę przycisków.
- Tworzy listę niepoprawnych przycisków i losowo usuwa etykiety dwóch z nich.

2.4.7 Metoda `OnPhoneAFriend`

Metoda `OnPhoneAFriend` obsługuje kliknięcie przycisku "Telefon do przyjaciela", który usuwa trzy niepoprawne odpowiedzi z ekranu.

```
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++;
  }
 }
}
```

Opis:

- Sprawdza, czy koło ratunkowe "Telefon do przyjaciela" lub "50/50" zostało już użyte.
- Ustawia flagi `phoneAFriendUsed` i `phoneAFriendUsedGlobal` na `true`.
- Pobiera poprawną odpowiedź i tworzy listę przycisków.
- Losowo usuwa etykiety trzech niepoprawnych przycisków.

2.4.8 Metoda `OnQuit`

Metoda `OnQuit` zamyka aplikację.

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

Opis:

- Wywołuje metodę `Close()`, która zamyka główne okno dialogowe i kończy działanie aplikacji.

2.4.9 Metoda `OnAbout`

Metoda `OnAbout` wyświetla informacje o aplikacji.

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

Opis:

- Wywołuje funkcję `wxbuildinfo()` z argumentem `long_f`, aby uzyskać szczegółowe informacje o kompilacji.
- Wyświetla te informacje w oknie dialogowym przy użyciu `wxMessageBox()`.

2.5 Funkcja `main`

Funkcja `main` tworzy instancję aplikacji WX Widgets i uruchamia ją.

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

Opis:

- Ustawia instancję aplikacji na nowy obiekt `ProjektApp`.
- Wywołuje funkcję `wxEntry()`, która uruchamia główną pętlę aplikacji WX Widgets.

2.6 Dodatkowe metody

- `OnInit` - Inicjalizuje dialog. Jest to metoda, która jest wywoływana podczas tworzenia okna dialogowego, ale w tym przypadku jest pusta i nie zawiera żadnej dodatkowej logiki.

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

- `OnTextCtrl1Text` - Obsługuje zmiany tekstu w polu tekstowym. Jest to metoda, która jest wywoływana, gdy zawartość pola tekstowego zostanie zmieniona, ale w tym przypadku jest pusta i nie zawiera żadnej dodatkowej logiki.

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

3 Podział obowiązków

3.1 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.

3.2 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.

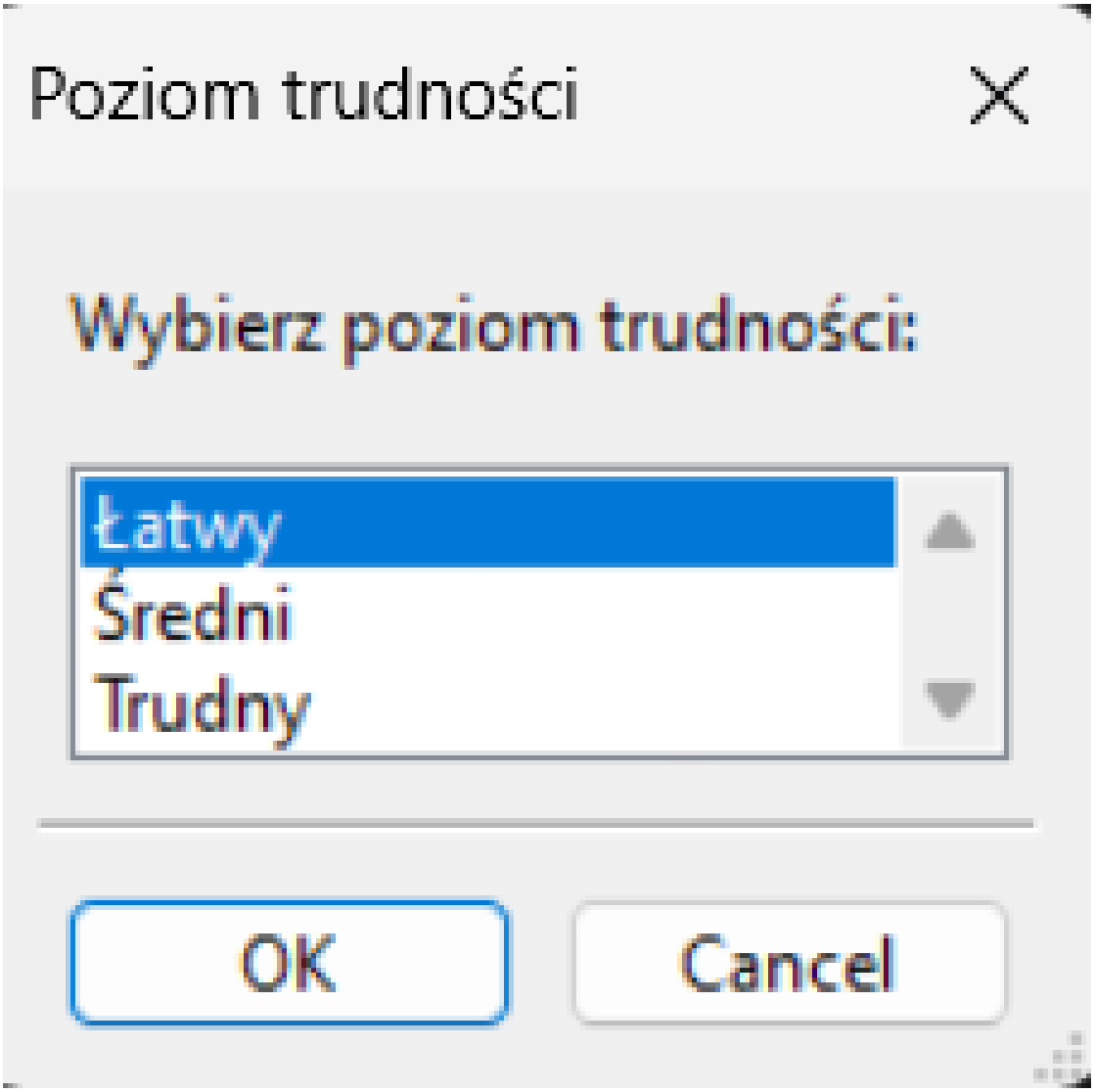
3.3 Oskar Wilkos

- 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.

4 Podsumowanie

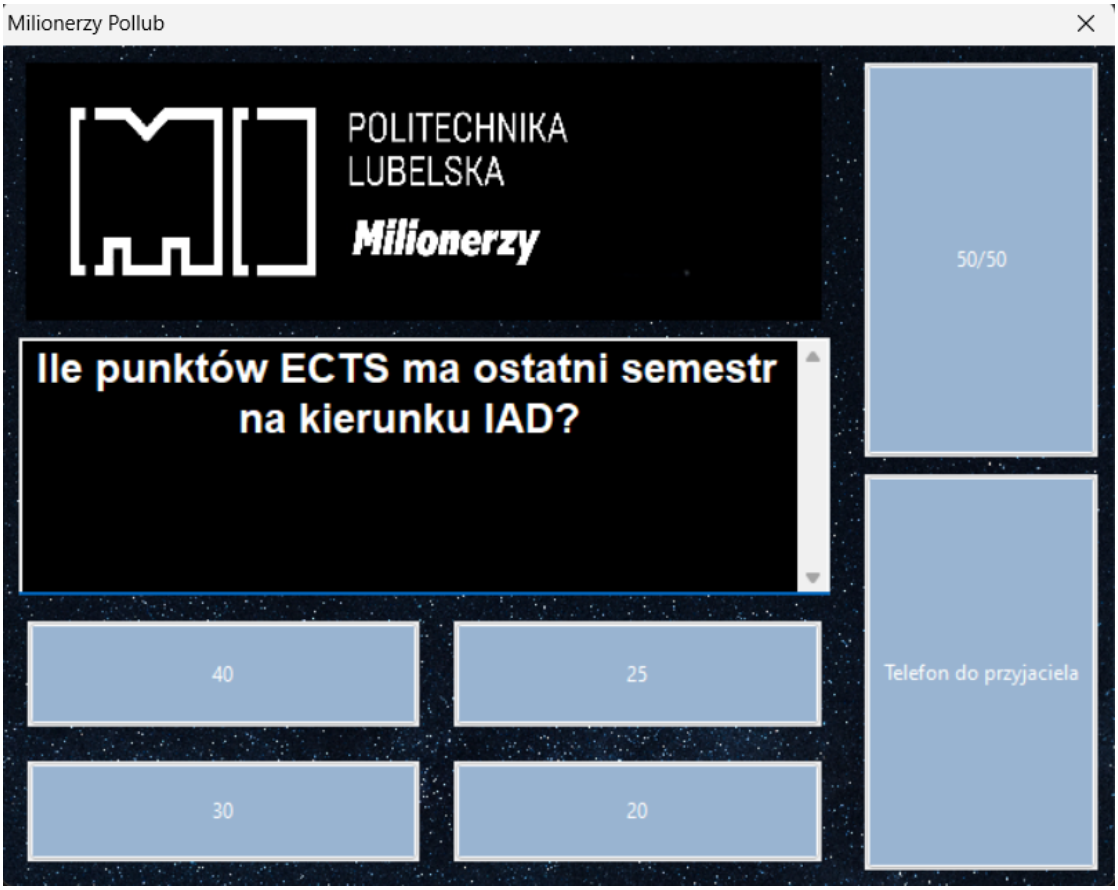
Aplikacja *Milionerzy* jest w pełni funkcjonalnym quizem inspirowanym popularnym teleturniejem. Dzięki możliwości wyboru poziomu trudności oraz użyciu kół ratunkowych, gra oferuje różnorodne wyzwania dla użytkowników. Implementacja opiera się na bibliotece WX Widgets, co pozwala na tworzenie interaktywnych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

5 Interfejs graficzny



Rysunek 1: Wybór poziomu trudności

Jak widać na rysunku 1, jest to wstępne okienko pojawiające się przed rozpoczęciem gry, aby dostosować poziom trudności, tym samym wybór bazy pytań. Gracz ma możliwość wyboru między różnymi poziomami trudności, co wpływa na zestaw pytań, z którymi będzie musiał się zmierzyć podczas gry. Opcje te pozwalają na dostosowanie gry do umiejętności i preferencji gracza, co zwiększa jej atrakcyjność i dostosowanie do indywidualnych potrzeb.

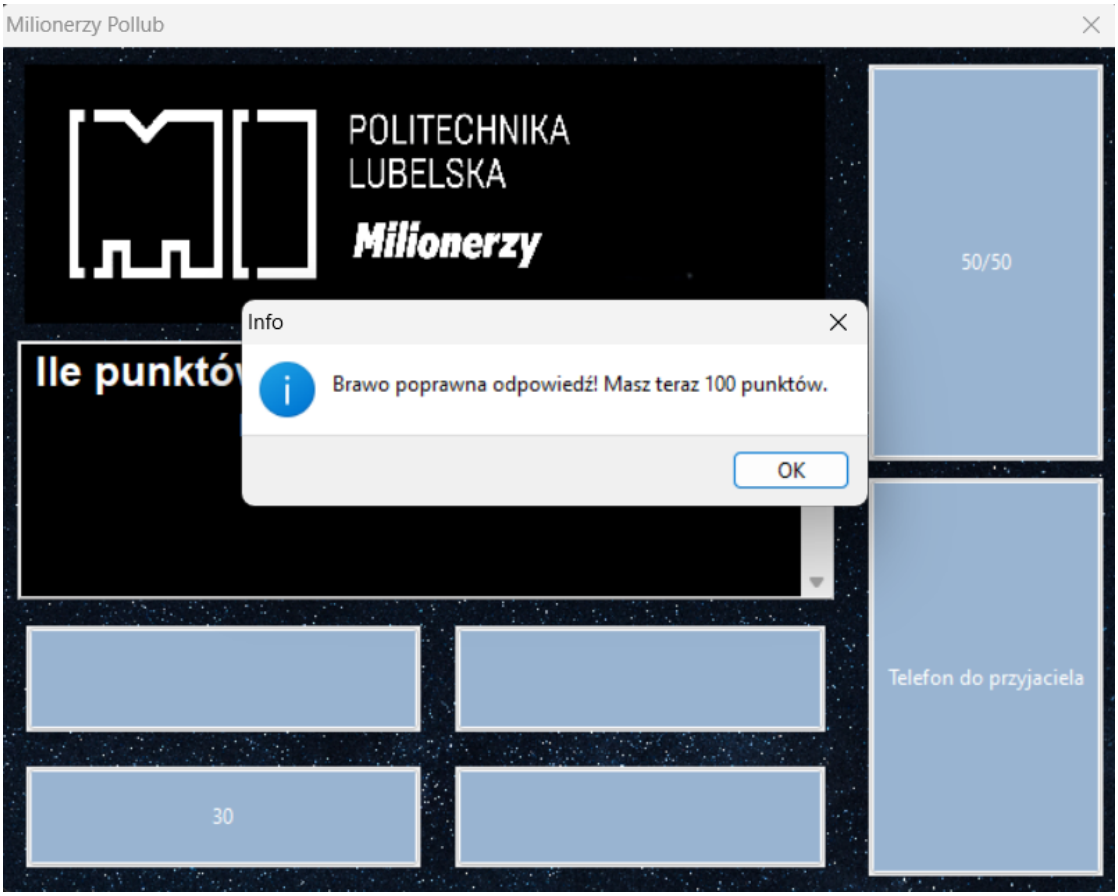


Rysunek 2: Interfejs startowy

Rysunek 2 przedstawia okno po uruchomieniu aplikacji w momencie startowym po wybraniu poziomu trudności. W tym wypadku został wybrany poziom trudności "łatwy", co powoduje, że widzimy skutek działania funkcji rysowania tła i losowania pytań.

Rysunek 3: Koło ratunkowe telefon do przyjaciela

Na rysunku 3 pokazano sytuację, w której gracz używa koła ratunkowego "telefon do przyjaciela" w celu uzyskania pomocy przy odpowiedzi na pytanie. Jest to jedno z dostępnych kół ratunkowych, które gracz może wykorzystać w trudnych momentach gry. Wybór tego koła ratunkowego pozwala "skontaktować się z wirtualnym przyjacielem", który raz w całej grze może nam wskazać poprawną odpowiedź.

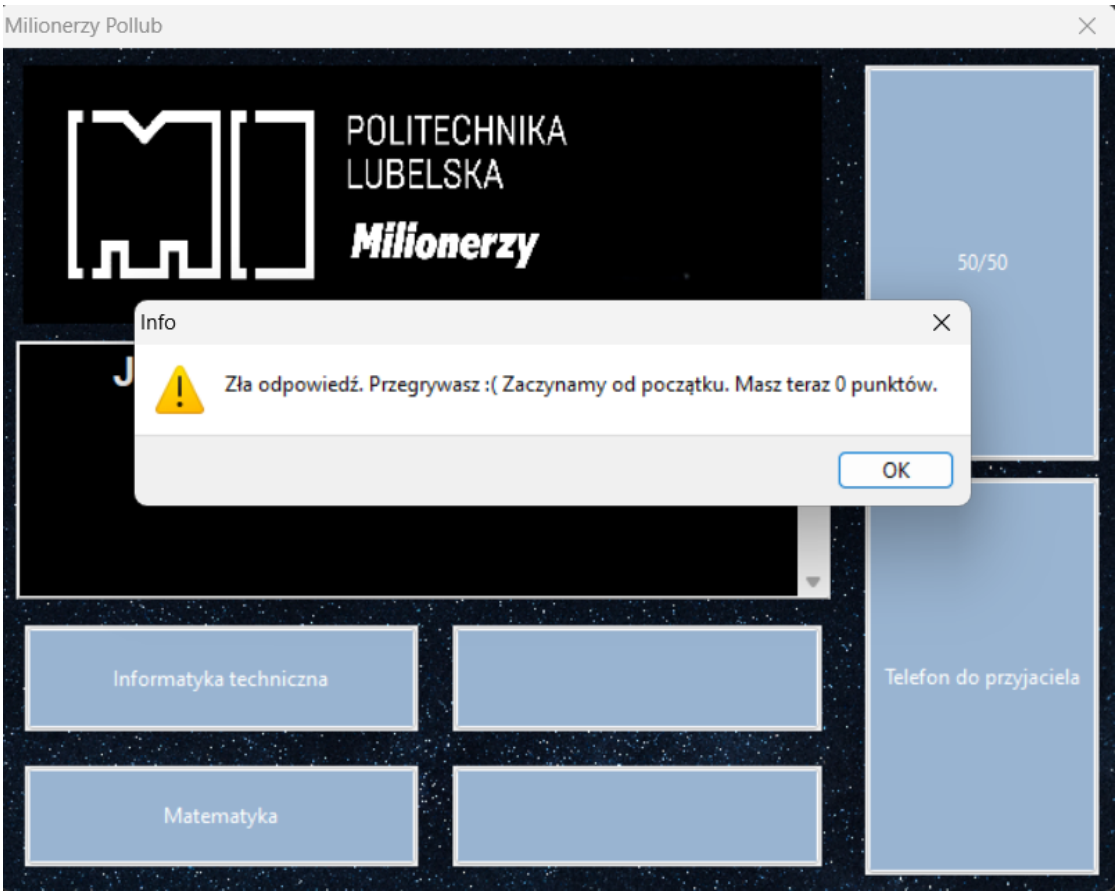


Rysunek 4: Wybranie poprawnej odpowiedzi

Rysunek 4 przedstawia moment, w którym gracz wybiera poprawną odpowiedź na zadane pytanie. Interfejs wyraźnie wskazuje, która odpowiedź została wybrana, a także informuje gracza o jej poprawności. Poprawne odpowiedzi są nagradzane i gracz zyskuje 100 pkt, co motywuje gracza do dalszej rozgrywki i zwiększa jego zaangażowanie w grę.

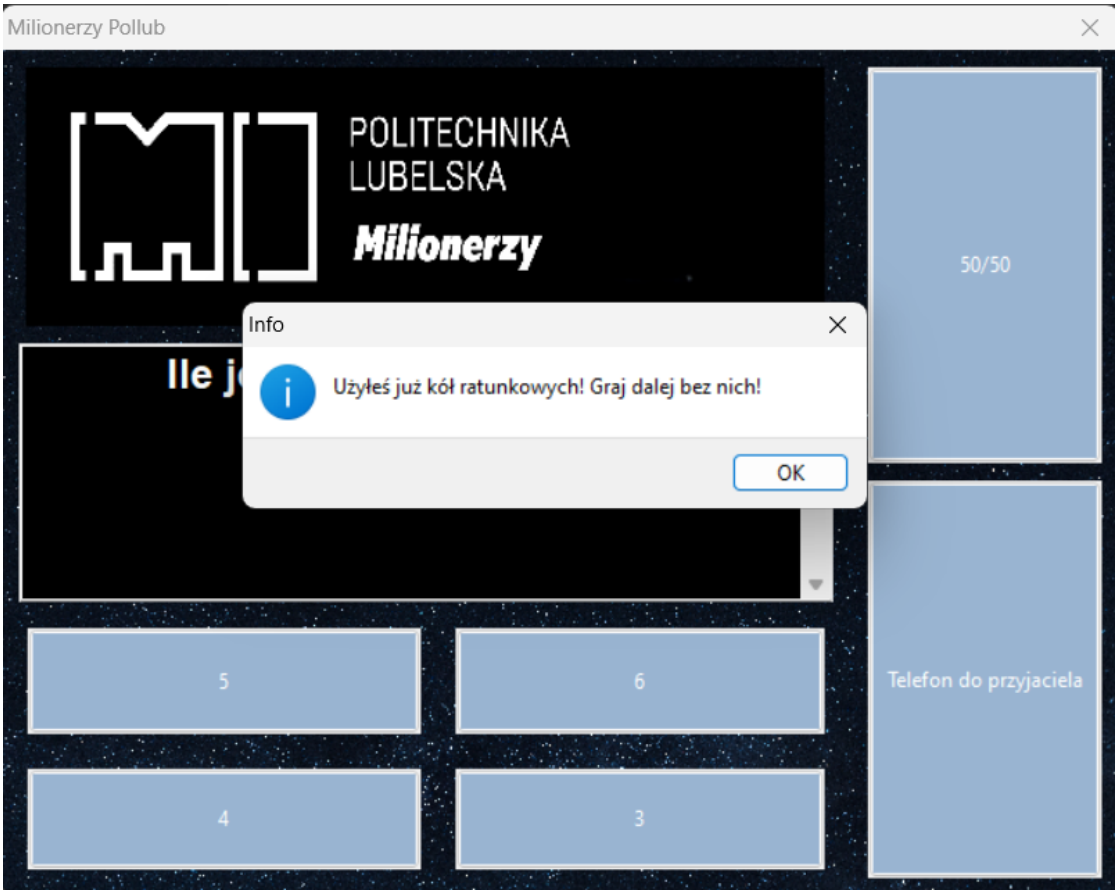
Rysunek 5: Koło ratunkowe 50/50

Na rysunku 5 widać użycie koła ratunkowego 50/50, które usuwa dwie błędne odpowiedzi, pozostawiając jedną poprawną i jedną niepoprawną odpowiedź. Jest to koło które znacząco zwiększa szanse gracza na udzielenie poprawnej odpowiedzi, zwłaszcza w trudniejszych pytaniach, gdzie pewność odpowiedzi jest niska. Te koło również możemy użyć tylko raz w ciągu całej gry.



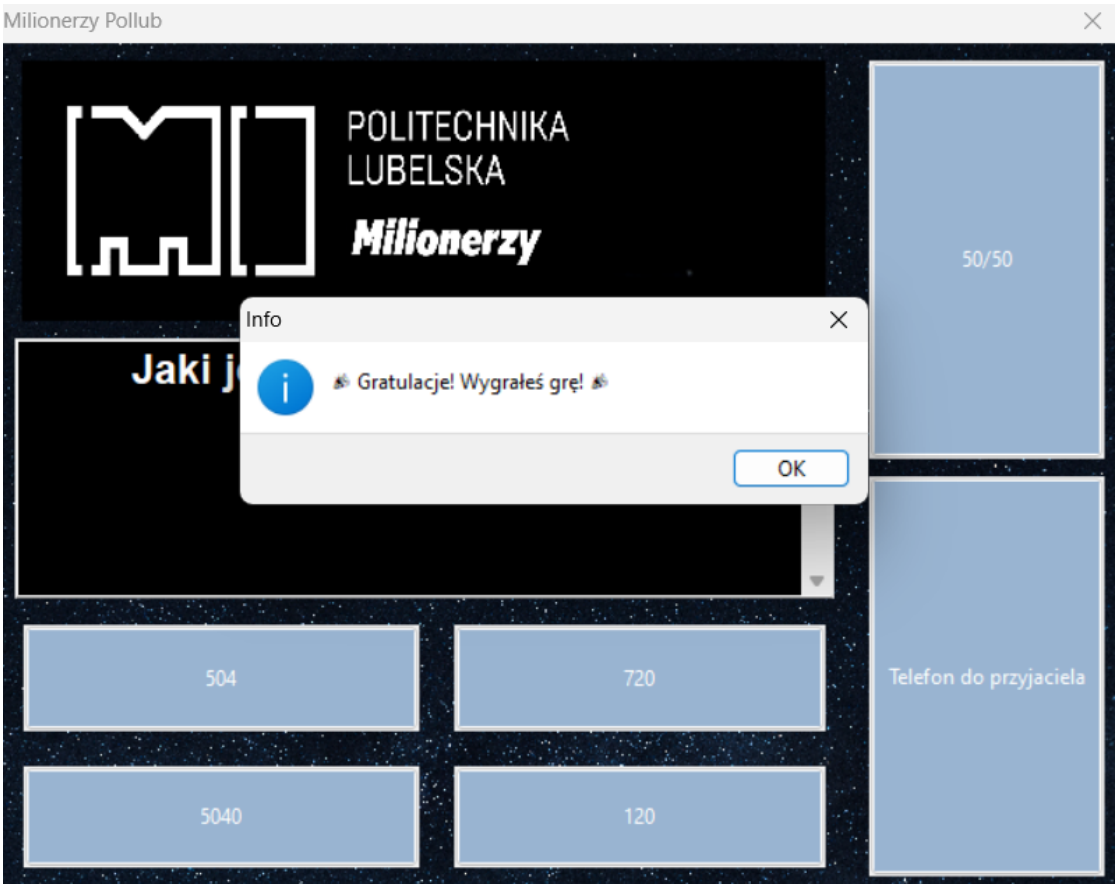
Rysunek 6: Komunikat o złej odpowiedzi

Rysunek 6 przedstawia komunikat, który pojawia się, gdy gracz udzieli niepoprawnej odpowiedzi na pytanie. Interfejs informuje gracza o błędnej odpowiedzi i przekazuje odpowiednie informacje zwrotne, o resetwaniu liczny punktów do 0 i ropzpoczęcie gry od początku na tym samym poziomie gry. Komunikat ten jest kluczowym elementem gry, który wpływa na doświadczenie użytkownika i jego zaangażowanie.



Rysunek 7: Komunikat o użyciu kół ratunkowych

Rysunek 7 pokazuje komunikat, który informuje gracza o użyciu wszystkich dostępnych kół ratunkowych. Jest to ważna informacja, ponieważ gracz musi być świadomy, że nie może już skorzystać z dodatkowej pomocy i musi polegać wyłącznie na swojej wiedzy w dalszej części gry. Komunikat ten motywuje do ostrożniejszego podejmowania decyzji w kolejnych rundach.



Rysunek 8: Komunikat o wygranej

Na rysunku 8 przedstawiono komunikat, który pojawia się, gdy gracz wygra grę tzn. uzyska 1000 pkt, udzielając poprawnej odpowiedzi na ostatnie pytanie. Komunikat ten informuje o zakończeniu gry i gratuluje graczowi osiągnięcia sukcesu. Jest to kluczowy moment gry, który nagradza gracza za jego wysiłek i wiedzę, zapewniając satysfakcję i pozytywne wrażenia z rozgrywki.