AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Operatsiya 'Kooperatsiya'(Operacja kooperacja)

Autor:
Maciej Śmierciak
Michał Jonak
Konrad Szczurek

Prowadzący:
mgr inż. Dawid Kotlarski

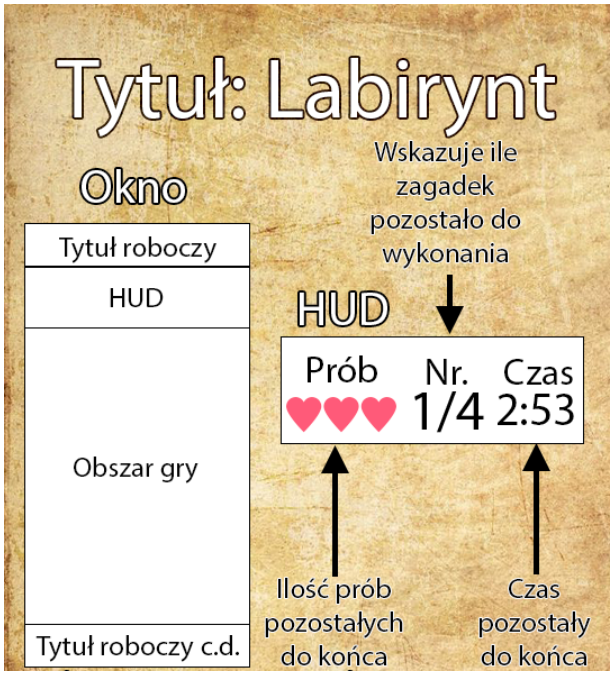
Nowy Sącz 2022

Spis treści

1. Ogólne określenie wymagań

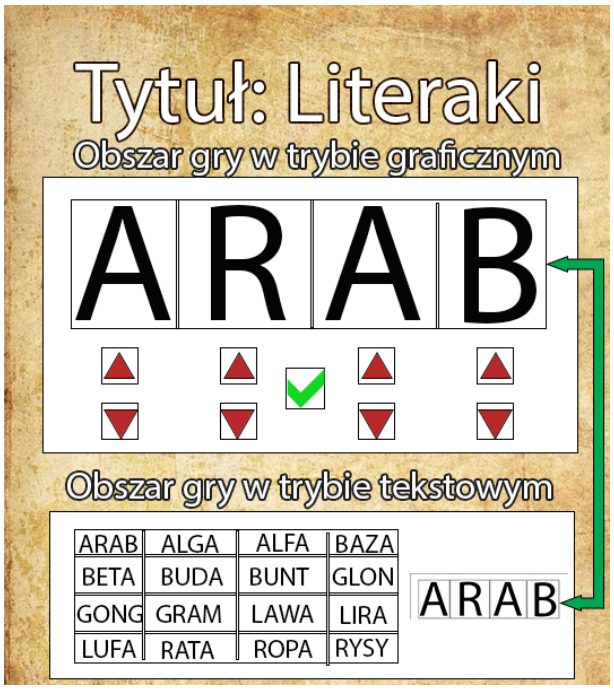
1.1. Gra logiczna

Projektem jest gra logiczna możliwa do zagrania tylko w trybie kooperacji. Gra będzie opierać się na stosunkowo łatwych zagadkach, które będzie można rozwiązać tylko współpracując.
Ogólnym konceptem jest podzielenie gry na 2 główne części:
-Tryb graficzny
-Tryb tekstowy



Rys. 1.1. Labirynt

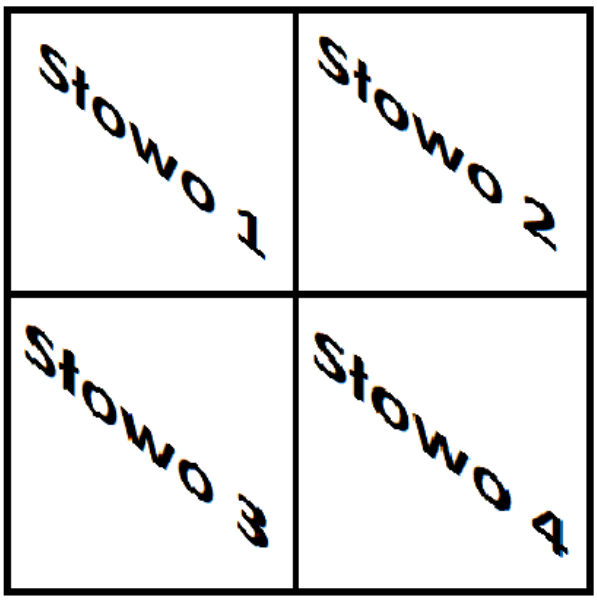
Ja możemy zauważyć na rysunku będziemy mieli określoną liczbę żyć na rozwiązanie określonej liczby zagadek w określonym czasie. W tym trybie zagadki będą polegały na wyjściu z labiryntu.



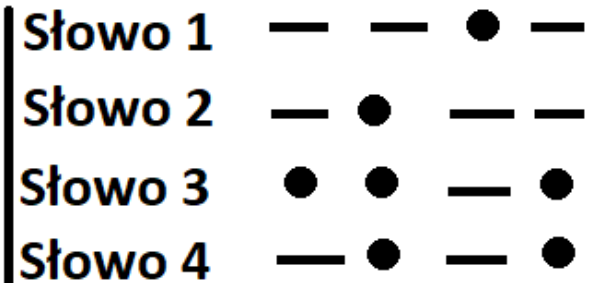
Rys. 1.2. Literaki

W tym trybie gracze mają za zadanie ułożyć czteroliterowy wyraz, który nakłada się z wyrazem w bazie. Dostęp do bazy wyrazów ma gracz w trybie tekstowym. Gracz obsługujący tryb graficzny za pomocą strzałek zmienia litery na danej pozycji. Ze względu na to, ze nie wszystkie litery są na wszystkich polach możliwość będzie tylko jedna. Błędna kombinacja oznacza utratę jednego z żyć. Podobnie jak w trybie labiryntu po utracie 3 żyć gracze przegrywają.

Trzeci tryb gry, który zostanie zaimplementowany do gry będzie oparty na latarce zawartej w telefonie. Telefon osoby obsługującej tryb graficzny włączy i wyłączy latarkę określoną liczbę razy tworząc przy tym "kod morsa"opisany w trybie tekstowym. Gracz obsługujący tryb graficzny będzie miał za zadanie zapamiętać stosunkowo krótki kod i podać go osobie będącej w trybie graficznym. Następnie osoba zarządzająca trybem graficznym musi dopasować go do jednego ze słów po czym podaje słowo kluczowe wspólnikowi. Wybór złego słowa pozbawia nas jednego życia i losuje nowy sygnał.



Rys. 1.3. Kod - Tryb graficzny

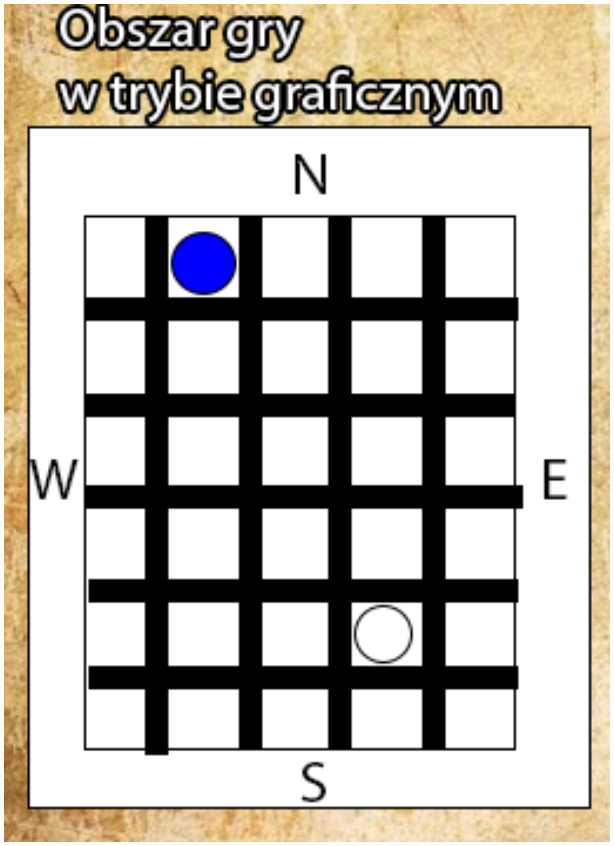


Rys. 1.4. Kod - Tryb tekstowy

1.1.1. Tryb Graficzny

Będzie opierał się na rozwiązywaniu zagadek. Gracz sam nie będzie w stanie rozwiązać zagadki, ponieważ podpowiedzi czy też cała solucja danej zagadki będą zawarte w trybie tekstowym.

W tym trybie będziemy widzieć plansze rozgrywki i będziemy mogli sterować naszą postacią.



Rys. 1.5. Labirynt

W trybie graficznym, jak widać na powyższym rysunku, widzimy naszą postać, niebieską kulkę, i nasz cel, białą kulkę. Natomiast nie widzimy drogi do mety i w tym celu musimy komunikować się z partnerem.

Za każdym razem jak wykonamy zły ruch czyli wejdziemy w ścianę nasza kulka będzie wracać na początek trasy a my tracimy jedno z naszych żyć. Po utracie wszystkich żyć kończymy rozgrywkę.

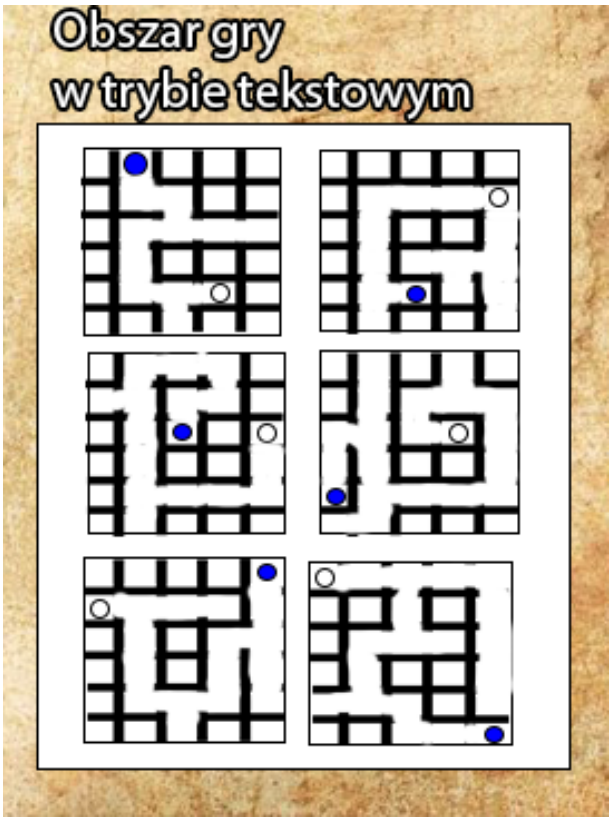
Innym trybem gry będą literaki polegające na układaniu słów. Gracz w tym trybie będzie za pomocą strzałek zmieniał litery na określonych pozycjach. Błędna kombinacja prowadzi do utraty życia i jest równoznaczna z wejściem w ścianę w trybie labiryntu.

Tryb graficzny trzeciej gry będzie oparty na prostej tabeli z paroma różnymi opcjami do wyboru. W zależności od otrzymanych instrukcji od naszego wspólnika będziemy musieli wybrać jedną z nich. Telefon gracza obsługującego ten tryb będzie za pomocą latarki wyświetlał jeden z dwóch rodzajów sygnału. Bedzie to kod oparty o kod morsa ale ze zmienionymi słowami. Dla przykładu jeżeli latarka włączy się raz na długo a potem 3 razy mrugnie będzie to oznaczało jeden sygnał długi(oznaczenie w trybie tekstowym - kreska) i 3 krótkie(w trybie tekstowym - kropka).

1.1.2. Tryb Tekstowy

Będzie opierał się na znajdowaniu podpowiedzi czy też solucji do aktualnie wykonywanej zagadki przez osobę obsługującą tryb graficzny.

Naszym zadaniem będzie współpraca z osobą, która steruje postacią w trybie graficznym w celu jak najefektywniejszego ukończenia zagadek przed końcem ustalonego czasu.



Rys. 1.6. Labirynt

Jak można zobaczyć na powyższym rysunku w trybie tekstowym będziemy widzieć dostępne mapy rozgrywki. Zadaniem gracza w trybie teksowym będzie ta-

kie poprowadzenie partnera w trybie graficznym, żeby niebieska kula dotarła do mety(białej kuli) unikając wchodzenia w ściany.

W trybie gry "Literaki"gracz w tym trybie będzie miał dostępną bazę ze słowami i będzie musiał dzięki komunikacji z graczem operującym interfejsem graficzym pomóc mu ułożyć pasujące słowo. Gracz w trybie graficznym będzie miał opcję ułożenia tylko jednego słowa z bazy.

W trzecim trybie gry będziemy musieli za pomocą komunikacji z naszym partnerem rozszyfrować o jaki kod chodzi w danym momencie. W trybie tekstowym będziemy otrzymywać informacje dotyczące "wyglądu kodu". W rzeczywistości ostrzymamy informację ile było długich sygnałów a ile krótkich i w jakiej kolejności. Po otrzymaniu takowych informacji gracz będzie stawał przed wyborem jednego ze słów, które w tym przypadku będzie oznaczone przed chwilą otrzymanym kodem. Jeżeli żadne słowo się nie będzie zgadzać to najprawdopodobniej otrzymaliśmy złe podpowiedzi od naszego partnera. Jeżeli jednak znajdujemy pasujący kod to podajemy słowo, które ten kod opisuje do operatora trybu graficznego.

2. Określenie wymagań szczegółowych

2.1. Założenia główne

- **Utrzymanie modułowości projektu**
- **Łatwość implementacji**
- **Prostota w testowaniu i ewentualnym debuggingu**
- **Użycie technologii Bluetooth**
- **Użycie żyroskopu i czujnika oświetlenia**
- **Użycie latarki**

2.1.1. Utrzymanie modułowości projektu

Pozwoli to na pracę nad wieloma "poziomami"jednocześnie co przełoży się na lepsze rozłożenie pracy pomiędzy członków grupy.

2.1.2. Latwość implementacji

Pozwoli to na testowanie każdego modułu osobno. Dzięki temu rozwiązaniu będziemy mogli lepiej wyeliminować błędy. A co za tym idzie lepiej dopracować nasz projekt.

2.1.3. Prostota w testowaniu i ewentualnym debuggingu

Chcemy dążyć do jak najłatwiejszego i jednocześnie najbardziej efektywnego sposobu testowania aplikacji. Pozwoli nam to zaoszczędzić cenny czas, który będziemy mogli poświęcić na lepsze dopracowanie szczegółów.

2.1.4. Użycie technologii Bluetooth

Jedna z zagadek w naszej grze będzie oparta o technologię bluetooth. Zadaniem gracza będzie wyłączenie i włączenie bluetooth odpowiednią ilość razy.

Bluetooth to standard bezprzewodowej komunikacji krótkiego zasięgu pomiędzy różnymi urządzeniami elektronicznymi.Jest opisany w specyfikacji IEEE 802.15.1. Jego specyfikacja techniczna obejmuje trzy klasy mocy nadawczej ERP 1-3 o zasięgu 100,

10 oraz 1 metra w otwartej przestrzeni. Najczęściej spotykaną klasą jest klasa druga. Standard korzysta z fal radiowych w paśmie częstotliwości ISM 2,4 GHz.

Urządzenie umożliwiające wykorzystanie tego standardu to adapter Bluetooth.

2.1.5. Użycie żyroskopu i czujnika oświetlenia

Żyroskop jak i czujnik oświetlenia docelowo mają posłużyć do rozwiązywania zagadek. Dla przykładu niektóre zagadki mogą być wykonane tylko w nocy czy też po prostu przy słabym oświetleniu.

Żyroskop to urządzenie służące do pomiaru lub utrzymywania położenia kątowego. Żyroskop w smartfonie to bowiem mikroskopijne płytki, które w odpowiednim momencie wibrują przekazując informacje do czujników. Dzięki temu możliwe jest ustalenie położenia smartfona oraz to czy i wokół której osi został obrócony.

Czujnik oświetlenia mierzy intensywność barwy białej, czerwonej, zielonej i niebieskiej w otoczeniu. Używany jest do zarządzania kolorystyką ekranu oraz zmiany intensywności jego podświetlenia. Dzięki temu wyświetlany obraz jest lepiej dostosowany do warunków w jakich przebywamy. Pozwala też zaoszczędzić nieco energii i dzięki temu wydłużyć czas pracy na baterii.

2.1.6. Użycie latarki

Jedna z zagadek będzie oparta na wysyłaniu sygnałów w formnie kodu morsa za pomocą latarki. Latarka będzie uruchamiana na określony odstęp czasu po czym zostanie wyłączona i włączona ponownie jeżeli ostatni sygnał nie został pokazany. Kod będzie oparty na 2 sygnałach: krótkim i długim.

2.2. Struktura aplikacji

W aplikacji będzie dostępne:

- **Menu główne**
- **Menu ustawień**
- **Dwa tryby gry**
  - **Tryb graficzny**

– **Tryb tekstowy**

2.2.1. Menu główne

W tym panelu będziemy mieli dostęp zarówno do wyboru trybu gry jak i do ustawień Ten panel będzie przejrzysty i łatwy w obsłudze, wszyskie opcje będą podpisane lub będą zawierały adekwatną do nazwy ikonę. Dla przykładu ustawienia zostaną oznaczone zębatką z podpisem ustawienia.

2.2.2. Menu ustawień

Ten panel umożliwi użytkownikom, jak sama nazwa wskazuje, ustawienia rozgrywki takie jak głośność muzyki.

2.2.3. Dwa tryby gry

Użytkownicy będą mieli do wyboru tryb gry. Jeżeli użytkownik zdecyduje się na wybór trybu graficznego jedyne co będzie musiał zrobić to kliknąć w odpowieni przycisk oznaczony jako tryb graficzny.

3. Projektowanie

3.1. Wykorzystane narzędzia

Podczas tworzenia aplikacji, której docelowym środowiskiem będzie Android wykorzystaliśmy platformę open source Xamarin i zestaw narzędzi Xamarin Community Toolkit, który ułatwił wykonywanie niektórych zadań. Projekt jest tworzony w języku C# co umożliwia nam wykorzystanie Microsoft Visual Studio Community Edition 2022. IDE (z ang. Integrated Development Environment), czyli zintegrowane środowisko programistyczne posłuży nam do łatwiejszego modułowania aplikacji. Wszystkie wersje kodów aplikacji znajdują się na platformie GitHub dzięki czemu łatwiej będzie wrócić do poprzednich wersji aplikacji w przypadku wystąpienia błędów w działaniu aktualnej. Wszystko początkowo miało zostać połączone za pomocą bazdy danych Firebase. Baza ta umożliwiałaby przesyłanie danych między graczami i ułatwiała przejście do następnych poziomów.

3.1.1. Xamarin

Xamarin[1] (logo - rys. 3.1) to platforma typu open source do tworzenia nowoczesnych i wydajnych aplikacji dla systemów iOS, Android i Windows za pomocą platformy .NET. Xamarin to warstwa abstrakcji, która zarządza komunikacją udostępnionego kodu z bazowym kodem platformy. Środowisko Xamarin działa w środowisku zarządzanym, które zapewnia wygody, takie jak alokacja pamięci i odzyskiwanie pamięci.

Rys. 3.1. Xamarin

[1] http://xamarinlab.pl/

3.1.2. Xamarin Community Toolkit

Zestaw narzędzi Xamarin Community Toolkit[2] (logo - rys 3.2)to zbiór elementów wielokrotnego użytku na potrzeby tworzenia aplikacji mobilnych za pomocą zestawu narzędzi Xamarin.Forms, w tym animacji, zachowań, konwerterów, efektów i pomocników. Upraszcza i demonstruje typowe zadania deweloperskie podczas kompilowania aplikacji dla systemów iOS, Android, macOS, WPF i platforma uniwersalna systemu Windows (UWP) przy użyciu platformy Xamarin.Forms.

Rys. 3.2. Xamarin Community ToolKit

3.1.3. Microsoft Visual Studio

Microsoft Visual Studio[3] (logo - rys 3.3) to zintegrowane środowisko programistyczne, służące do tworzenia, edytowania i debugowania kodu. IDE (z ang. Integrated Development Environment), czyli zintegrowane środowisko programistyczne to software oferujący szereg funkcji przydatnych podczas tworzenia oprogramowania dla systemów Windows, Android, iOS, rozwiązań internetowych oraz opartych o chmurę. Poza standardowym edytorem oraz debugerem, które zapewnia większość aplikacji IDE, program Microsoft Visual Studio zawiera jeszcze kompilatory, narzędzia do uzupełniania kodu i wiele innych funkcji usprawniających pracę.

[2]https://learn.microsoft.com/pl-pl/xamarin/community-toolkit/
[3]https://visualstudio.microsoft.com/pl

Rys. 3.3. Visual

3.1.4. Git, GitHub

Git jak i GitHub[4] (logo - rys 3.4) to najczęściej używany nowoczesny system kontroli wersji. Za pomocą usługi Git możesz śledzić zmiany kodu wprowadzane w czasie i przywrócić określone wersje. Ponadto pozwala kontrolować dostęp do danych, wspiera zarządzanie wieloma repozytoriami. Jest to rozwiązanie, które pozwala zaoszczędzić sporo czasu i nerwów. Kolejnym plusem jest to, że jest on w miare łatwy w użyciu i jest przejrzysty.

[4]https://github.com/

Rys. 3.4. GitHub

3.1.5. Język C#

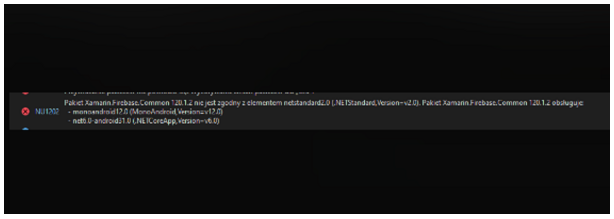
Język C#[5] (logo - rys 3.5) to zorientowany obiektowo język programowania, który umożliwia programostom tworzenie różnych bezpiecznych i niezawodnych aplikacji w środowisku .NET. Język ten udostępnia konstrukcje językowe, które bezpośrednio obsługują te koncepcje, dzięki czemu język C# jest językiem naturalnym, w którym można tworzyć składniki oprogramowania i ich używać.

[5]https://learn.microsoft.com/pl-pl/dotnet/csharp/

Rys. 3.5. C#

3.1.6. Baza danych Firebase

Firebase to platforma do tworzenia aplikacji, która pomaga tworzyć i rozwijać aplikacje i gry, które użytkownicy uwielbiają. Wspierany przez Google. W zamyśle nasza gra miała obsługiwać bazę danych lecz podczas implementacji i łączenia bazy z grą wystąpiły liczne komplikacje. Jednym z głównych błędów była niekompatybilność aktualnej wersji Firebase z naszą wersją Xamarina (rys 3.6). Po wielu próbach postanowiliśy zrezygnować z implementacji bazy danych i zdecydowaliśmy się na inne rozwiązanie.



Rys. 3.6. Komunikat błędu

Treść komunikatu: *Pakiet.Xamarin.Firebase.Common.120.1.2 nie jest zgodny z elementem netstandard2.0 (.NETStandard,Version=v2.0). Pakiet Xamarin.Firebase.Common 120.1.2 obsługuje:*
-monoandroid12.0(MonoAndroid,Version=v12.0)
-net6.0-android31.0(.NETCoreApp,Version=v6.0)

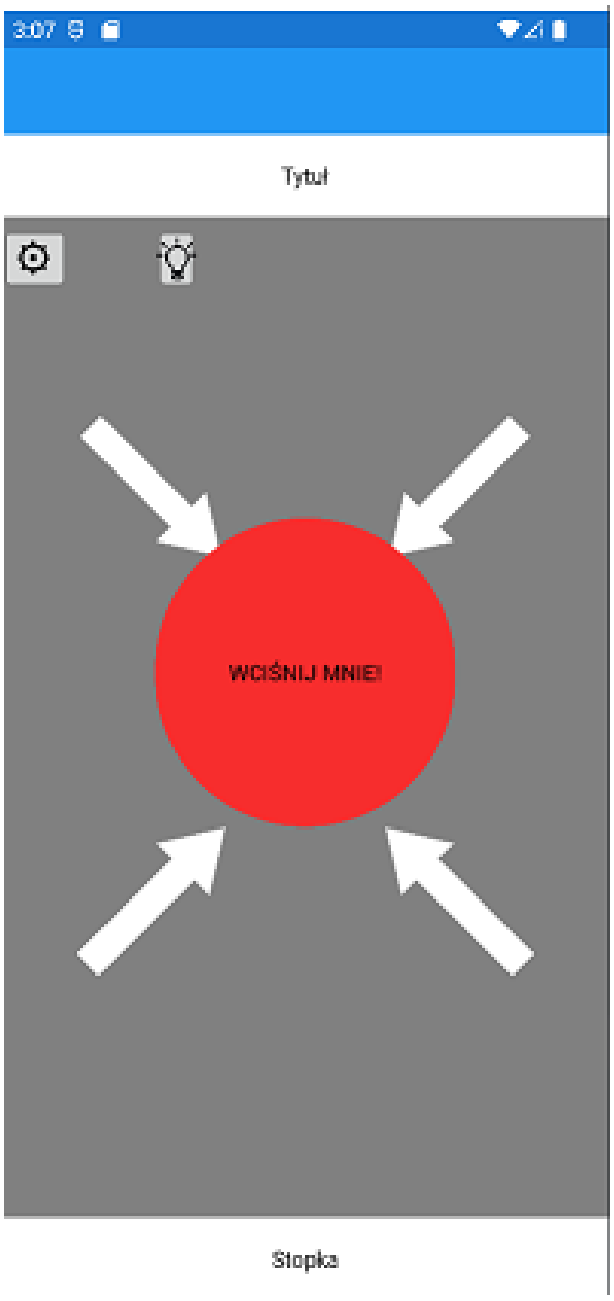
3.2. Działanie aplikacji

Gra zawiera kilka paneli:

- **Menu główne**
- **Menu ustawień**
- **Dwa tryby gry**
  - **Tryb graficzny**
  - **Tryb tekstowy**

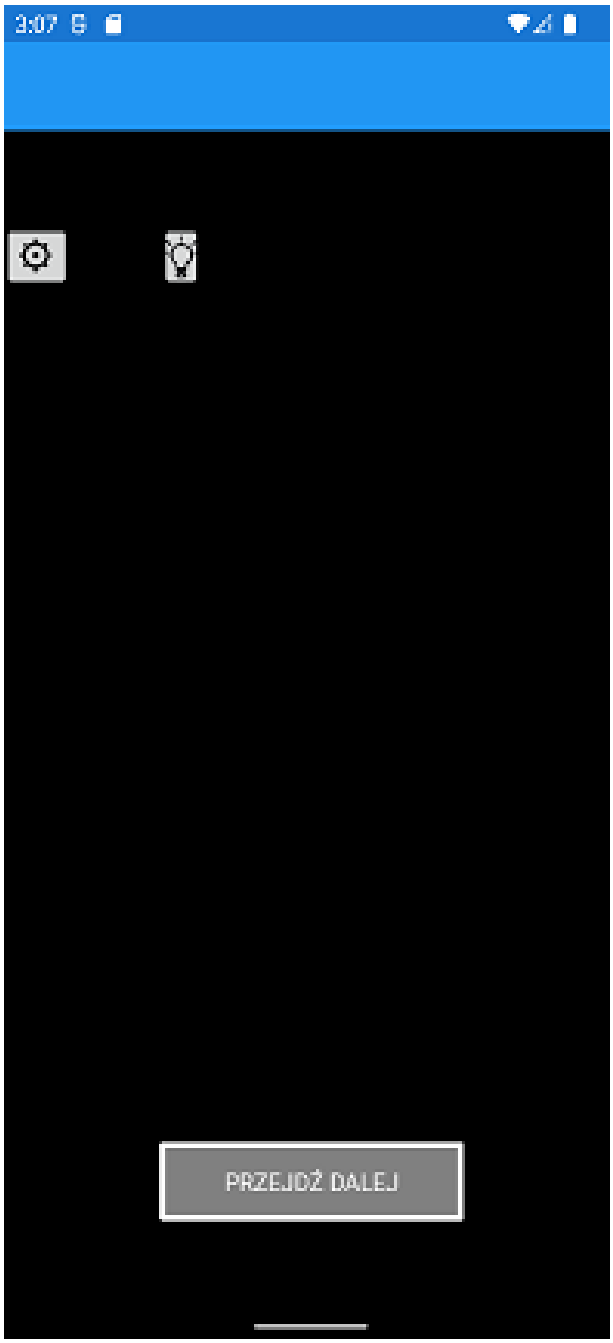
3.2.1. Menu główne

Po uruchomieniu aplikacji powita nas już pierwsza zagadka(rys 3.7), która polega na znalezienu przycisku pozwalającego przejść dalej. Tym przyciskiem jest przycisk oznaczony żarówką. Jeśli jednak zostanie wciśnięty duży czerwony przycisk na środku ekranu to gra zostanie wyłączona.
Jak widać mamy dostępny jeszcze przycisk oznaczony zębatką. Prowadzi on do ustawień rozgrywki.



Rys. 3.7. Menu Główne

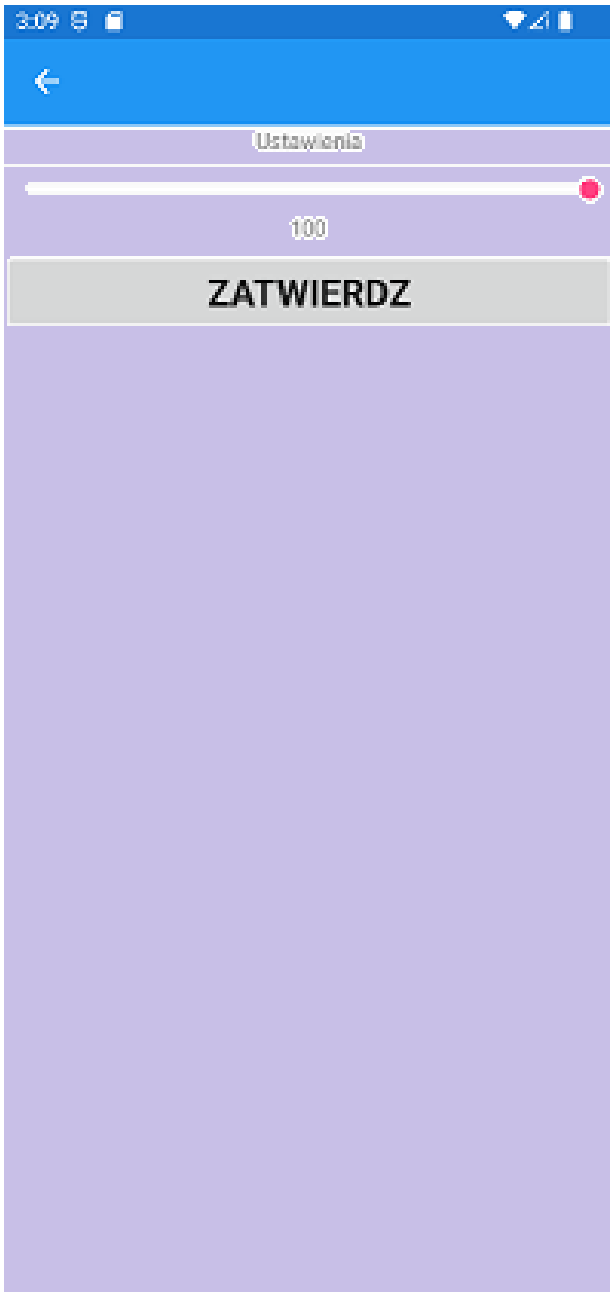
Po wciśnięciu dobrego przycisku przechodzimy do właściwego menu głównego(rys 3.8), w którym możemy przejść dalej



Rys. 3.8. Menu Główne v2

3.2.2. Ustawienia

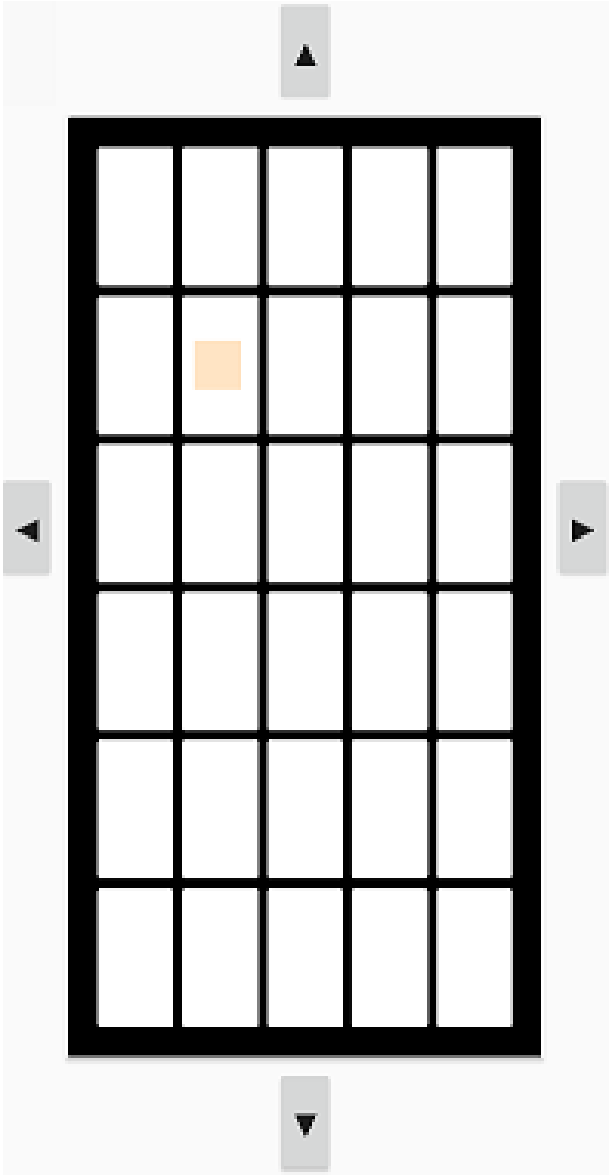
Ten panel oferuje nam zmianę ustawień gry. Dla przykładu głośności muzyki. Po zmienieniu głośności w panelu ustawień wartość ustawiona zostaje zapisana i przenoszona na inne panele aplikacji. Przejście do menu ustawień będzie możliwe zarówno z menu głównego jak i z poziomu gry.



Rys. 3.9. Ustawienia

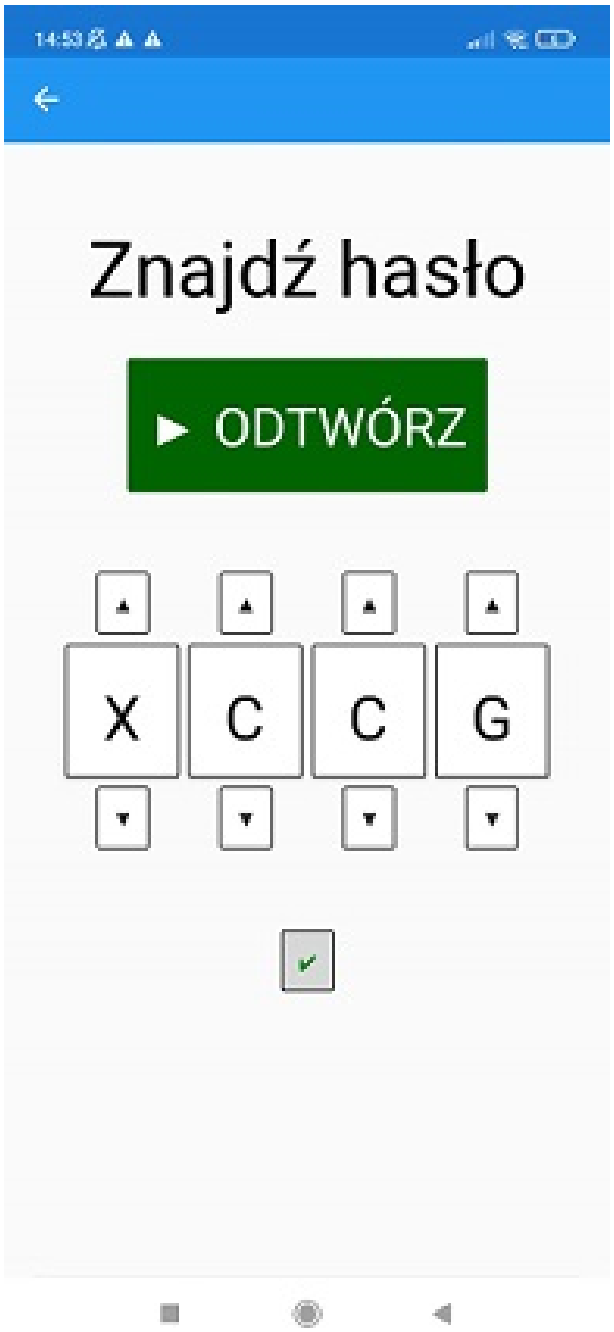
3.2.3. Tryb graficzny

Pierwsza zagadka w trybie graficznym będzie wyglądała tak jak na rys 3.10. Pomarańczowy kwdrat jest naszą postacią, którą poruszamy za pomocą strzałek umieszczonych na ekranie. Każda strzałka odpowiada za ruch w inną stronę.



Rys. 3.10. Labirynt

Kolejna zagadka będzie polegała na wyborze odpowiedniego słowa, będziemy zamieniać litery za pomocą strzałek jak na rys 3.11. Jeżeli ułożymy odpowiednie słowo zatwierdzamy przyciskem oznaczonym "ptaszkiem". Za pomocą przycisku odtwórz możemy odtworzyć kod morsa z latarki od nowa.



Rys. 3.11. Literaki

4. Implementacja

4.1. Przycisk

Przyciski to podstawowy i najprostszy element większości aplikacji. Ten element poprzez klikanie w niego wykonuje określoną akcje. Kod przykładowego przycisku w naszej grze można znaleźć na listungu 1:

```
<Button Grid.Row="0" Grid.Column="0" Text="&#x2699;" x:Name="
 SettingsButton" Padding="0,0,2,2" FontAttributes="Bold"
 FontSize="24" Clicked="GoToSettings"></Button>
```

Listing 1. Button

4.1.1. Użyte atrybuty przycisku

- **Text - napis wewnątrz przycisku. W tym wypadku kod szesnastkowy (hexcode) zębatki**
- **Padding - wewnętrzny margines**
- **Clicked - przycisk po kliknięciu odwołuje się do metody o tej nazwie**
- **CornerRadius - zaokrągla rogi przycisku, przy odpowiednio dużej wartości, przicisk może stać się okrągły**
- **FontSize - rozmiar czcionki**
- **Grid - ustawia element na ”siatce”**
  - **Grid.Row - odpowiada za oś Y czyli rząd**
  - **Grid.Column - odpowiada za oś X czyli kolumnę**

4.1.2. Inne użyteczne atrybuty

Można znaleźć na listingu 2. Opisują one polecenie label.

- **X:Name - nazwa elementu używana potem w C#**
- **Vertical/HorizontalTextAlignment - ustawia test na osi X/Y w zależności od wybranej opcji**
- **BackgroundColor -pozwala na zmianę koloru tła**

```
<Label Grid.Row="0" Grid.Column="0" x:Name="Tit" Text="TytuL"
 Grid.ColumnSpan="4" VerticalTextAlignment="Center"
 HorizontalTextAlignment="Center" BackgroundColor="White"/>

```

Listing 2. Label

4.1.3. Przyciski w formie zdjęcia

Inna forma wstawiania przyciusku, w którym zamiast tekstu znajdziemy wstawione zdjęcie jest pokazana na listingu 3.

```
<ImageButton Grid.Column="0" Grid.Row="0" x:Name="BulbButton"
    Clicked="ChangeLight" ClassId="0" Source="https://cdn-icons-png.
    flaticon.com/512/74/74072.png" Aspect="AspectFit" IsVisible="
    true" >
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualState x:Name="Normal">
        <VisualState.Setters>
          <Setter Property="Scale" Value="1" />
        </VisualState.Setters>
      </VisualState>
      <VisualState x:Name="Pressed">
        <VisualState.Setters>
          <Setter Property="Scale" Value="0.8" />
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</ImageButton>
```

Listing 3. ImageButton

4.1.4. Użyte atrybuty

- **VisualState - pozwala na zmianę wyglądu w zależności od stanu przycisku. W naszym przypadku state "normal"odnosi się do domyślnego ustawienia. State "Pressed"sprawia, że skala obiektu zmienia się do 0.8 (zmniejsza się) symulując efekt naciśnięcia po czym wraca do state "normal"**
- **Source - źródło obrazka, w tym wypadku link ale możliwe jest też użycie ścieżki**

4.2. MainPage

Funkcja pokazana na listingu 4 służy do sprawdzenia czy jest ustalona wartość głośności muzyki, a następnie ustawia wartość Volume soundtracku właśnie jako tą wartość.

```
public MainPage(double dalej)
{
  InitializeComponent();
  if (dalej == 0)
  dalej = 1;
  soundtrack.Volume = dalej;
}
```

Listing 4. MainPage

4.3. OnAppearing

Na listingu 5 nadpisujemy metodę OnAppearing(), która wykonuje się na starcie. Metoda FindByName znajduje elementy o podanych nazwach, na następnie wykonuje dla nich metodę Animate_Pulse().
Do zmiennej status przypisany zostaje stan "Uprawnienia do używania aparatu"przez aplikację (potrzebne do użycia lampy błyskowej).
Następnie w linii 16 sprawdzany jest status. Jeżeli status jest nieznany (unknown) lub odmówiono dostępu aplikacja prosi o przyznanie takiego uprawnienia

```
async protected override void OnAppearing()
{
  base.OnAppearing();
  arrow_1 = (Label)FindByName("arrow_1");
  arrow_2 = (Label)FindByName("arrow_2");
  arrow_3 = (Label)FindByName("arrow_3");
  arrow_4 = (Label)FindByName("arrow_4");
  bigRed = (Button)FindByName("bigRed");
  BulbButton = (ImageButton)FindByName("BulbButton");
  Animate_pulse(arrow_1);
  Animate_pulse(arrow_2);
  Animate_pulse(arrow_3);
  Animate_pulse(arrow_4);
  animate_button(bigRed);
  var status= await Permissions.CheckStatusAsync<Permissions.Camera
   >();
  if (status==PermissionStatus.Unknown || status ==
   PermissionStatus.Denied)
  {
    await Permissions.RequestAsync<Permissions.Camera>();
  }
}
```

Listing 5. OnAppearing

4.4. Animate_pulse

Animate_pulse zawarte na listingu 6 włącza animacje, w której to skala strzałek zmienia się pomiędzy 1 a 1.2
W tym przypadku używane jest ClassId po to, by nazywać animacje różnymi nazwami (te same nazwy nadpisywałby się)
W tym przypadku, przez połowę czasu animacji skala rośnie, by przez drugą połowę maleć

- **This – Animacja, która będzie animowana**
- **Objname – nazwa służąca jako uchwyt do animacji**
- **16 – Czas w milisekundach pomiędzy elementami animacji**
- **950 – czas w milisekundach trwania animacji**
- **Easing.Linear – funkcja używana do opisu animacji**
- **(v,c) =»obj.Scale=1 – co ma zostać wykonane po skończeniu animacji**
- **()=»true – czy funkcja ma być zapętlona**

```
1 void Animate_pulse(Label obj)
2 {
3   var Objname = obj.ClassId;
4   var a = new Animation {
5     {0,0.5, new Animation(v => obj.Scale = v, 1, 1.2) },
6     {0.5,1, new Animation(v => obj.Scale = v, 1.2, 1) }
7   };
8   a.Commit(this,Objname,16,950,Easing.Linear, (v, c) => obj.Scale =
      1,()=> true);
9 }
```

Listing 6. Animate_pulse

4.5. FirstTask

Funkcja zawarta na listingu 7 to funkcja, przyjmująca jako parametry obiekt, który został wciśnięty i argumenty zdarzenia, pozwala na przemieszczanie się pomiędzy stronami aplikacji.
Metoda soundtrack.Stop() sprawia, że soundtrack przestaje grać,

Do zmiennej dalej przekazywana jest aktualna wartość głośności muzyki granej w tle. Następnie asynchronicznie przechodzimy do nowej strony, przekazując jako parametr głośność soundtracku, by ten mógł grać z taką samą głośnością jaka była ustawiona

```
private async void FirstTask(object sender, EventArgs e)
{
  soundtrack.Stop();
  double dalej1 = soundtrack.Volume;
  await Navigation.PushAsync(new LabirynthPage(dalej1));
}
```

Listing 7. FirstTask

4.6. Wciśnięcie złego przycisku

Funkcja na listingu 8 odpowiada za wciśnięciu złego przycisku. Po wciśnięciu go zmienia się widoczność wszystkich elementów menu, następnie wyświetla się obrazek świadczący o porażce, a po odczekaniu 3,5s proces aplikacji kończy się i zwraca wartość 0, co świadczy o poprawnym zamknięciu aplikacji.

```
async void WrongButtonClicked(object sender, EventArgs e)
{
  ChangeLight(sender,e);
  var ButtonDalej = (Button)FindByName("LabirynthButton");
  var bulbButton = (ImageButton)FindByName("BulbButton");
  var settingsButton= (Button)FindByName("SettingsButton");
  var deadAnimation = (Image)FindByName("deadAnimation");
  var ButtonLatarka = (Button)FindByName("FlashlightButton");
  var ButtonTekstowy = (Button)FindByName("TextModeButton");
  var ButtonMorse = (Button)FindByName("ButtonMorse");
  ButtonDalej.IsVisible = false;
  ButtonTekstowy.IsVisible = false;
  ButtonLatarka.IsVisible = false;
  ButtonMorse.IsVisible = false;
  bulbButton.IsVisible = false;
  settingsButton.IsVisible = false;
  deadAnimation.IsVisible = true;
  deadAnimation.IsAnimationPlaying = true;
  await Task.Delay(3500);
  System.Environment.Exit(0);
}
```

Listing 8. WrongButton

4.7. HUD

HUD (ang. heads-up display) – Ważna część gier komputerowych. Jest to sekcja zawierająca informacje na temat rozgrywki, pozwalająca graczowi na odnalezienie informacji o niej. Nazwa wywodzi się od tzw. wyświetlacza przeziernego HUD stosowanego np. przez pilotów samolotów bojowych.
W linach 5-7 są opisane życia w postaci serc.
Linie 8-9 są odpowiedzialne za timer, jego widoczność, przezroczystość i cień jaki rzuca.

```
1             <Frame Grid.Row="0" Grid.ColumnSpan="20" Grid.RowSpan
    ="2" BackgroundColor="Gray">
2 <Grid>
3 <Label Grid.Column="0"></Label>
4
5 <Label Grid.Column="0" TextColor="White"  FontSize="30"
    VerticalTextAlignment="Center" x:Name="HealthBar1" Text
    ="&#10084;"></Label>
6 <Label Grid.Column="1" TextColor="White"  FontSize="30"
    VerticalTextAlignment="Center" x:Name="HealthBar2" Text
    ="&#10084;"></Label>
7 <Label Grid.Column="2" TextColor="White"  FontSize="30"
    VerticalTextAlignment="Center" x:Name="HealthBar3" Text
    ="&#10084;"></Label>
8
9 <Label x:Name="Timer" Margin="0,0,0,15"  Grid.Column="3" Grid.
    ColumnSpan="3"  TextColor="White" VerticalTextAlignment="Center"
     xct:ShadowEffect.Color="Black" xct:ShadowEffect.Opacity="0.7"
    xct:ShadowEffect.OffsetX="5" xct:ShadowEffect.Radius="5"
    FontSize="25" FontAttributes="Bold"></Label>
10 <Label x:Name="TimerCount" xct:ShadowEffect.Color="Black" xct:
    ShadowEffect.Opacity="0.7" xct:ShadowEffect.OffsetX="5" xct:
    ShadowEffect.Radius="5" Margin="0" TextColor="White" Grid.Column
    ="5" VerticalOptions="Center" FontSize="25"></Label>
11 <Label Grid.Column="5" Grid.ColumnSpan="3"  TextColor="White"
    VerticalTextAlignment="Center" x:Name="CodeNumber"></Label>
12
13 <Label Grid.Column="1"></Label>
14 </Grid>
15 </Frame>
```

Listing 9. HUD

4.8. Literaki

Linia 1 - zmienna zawierająca ilość ”żyć”[podejść]
Linia 2 - tablica przechowująca indeks każdego z bloków
Linia 3 - tablica przechowująca słowa mogące wystąpić jako rozwiązanie
Linia 4 - tablica przechowująca wszystkie wartości indeksów każdego z bloków
Linia 11 - zmienna zawierająca indeks wartości z TextTable[], która została wybrana jako rozwiązanie
Linia 12 - zmienna zawierająca pozostały czas na rozwiązanie zagadki
Linia 13 - zmienna zawierająca obiekt klasy Random, służący m.in. do losowania liczby z przedziału

```
1  public int HealthCount = 3;
2  int[] BlockCounters = { 0, 0, 0, 0 };
3  readonly string[] TextTable = { "ARAB", "ALGA", "ALFA", "BAZA", "
       BETA", "BUDA", "BUNT", "GLON", "GONG", "GRAM", "LAWA", "LIRA", "
       LUFA", "RATA", "ROPA", "RYSY" };
4  public char[,] WordTable =
5  {
6    {'A','A','A','A'},
7    {'A','A','A','A'},
8    {'A','A','A','A'},
9    {'A','A','A','A'},
10 };
11 public int ChosenWord;
12 public int TimeLeft = 60;
13 Random RandomCharCount = new Random();
```

Listing 10. Literaki

4.9. SetTime

W Linii 4 wyświetla się okno, które należy zamknąć, zanim ruszy zegar. Później uruchamiana jest metoda StartTimer(TimeSpan). TimeSpan przyjmuje trzy wartości - godziny, minuty i sekundy.
W Liniach 8-9 Odejmowane jest 1 od pozostałego czasu, a następnie uzyskana wartość jest wpisywana jako text do obiektu TimeCount.
W linii 10 widoczny jest warunek, który zmienia kolor tekstu na czerwony jeśli mamy mniej czasu niż 15 sekund, a w wypadku gdy się skończy czas wyświetla odpowiedni komunikat

```
1  async void SetTime()
2  {
```

```
3   await DisplayAlert("Rozpocznij zagadke", "Wcisnij OK, aby
     rozpoczac", "OK");
4   Label TimeCount = (Label)FindByName("TimerCount");
5   Label TimeBar = (Label)FindByName("Timer");
6   Device.StartTimer(new TimeSpan(0, 0, 1), () =>
7   {
8     TimeLeft--;
9     TimeCount.Text = TimeLeft.ToString();
10    if (TimeLeft < 15)
11    {
12      TimeCount.TextColor = Color.Red;
13      if (TimeLeft == 0)
14      {
15        DisplayAlert("Przegrales", "):", "No nie");
16        TimeCount.Text = "";
17        TimeBar.Text = TimeCount.Text;
18        return false;
19      }
20    }
21    return true;
22  });
23 }
```

Listing 11. SetTime

4.10. SetColumns

W każdej iteracji pętli do zmiennej name wpisywana jest nazwa Bloku, po czym w zmiennej Obj umieszczany jest obiekt o tej nazwie.
W Linii 6 Losowany jest liczba stałoprzecinkowa z zakresu 0 do rozmiaru tablicy WordTable.
W linii 7 jako wartość atrybutu text umieszczamy wylosowany znak.
Ta funkcja służy początkowemu ustawieniu tekstu w blokach.

```
1 setColumns()
2 {
3   for (int i = 0; i < 4; i++)
4   {
5     var Name = "Block" + i;
6     var Obj = (Label)FindByName(Name);
7     var Los = RandomCharCount.Next(0, (WordTable.Length / 4) - 1);
8     Obj.Text = WordTable[Los, i].ToString();
9   }
```

```
10 }
```

Listing 12. SetColums

4.11. RandomizeWordTable

W Linii 7 do zmiennej nextChar wpisywana jest losowa liczba z zakresu ¡65,90), która następnie zostaje przekonwertowana na char [są to znaki z zakresu A-Z] i wpisana jako wartość tablicy WordTable - w ten sposób za każdym razem losowana jest cała tablica ze znakami.

```
1  RandomizeWordTable()
2  {
3    for (int i = 0; i < (WordTable.Length/4); i++)
4    {
5      for (int j = 0; j < 4; j++)
6      {
7        var nextChar = RandomCharCount.Next(65, 90);
8        WordTable[i, j] = Convert.ToChar(nextChar);
9      }
10   }
11   char[] chosenWordChars = TextTable[ChosenWord].ToCharArray();
12   for (int i = 0; i < 4; i++)
13   {
14     int random = RandomCharCount.Next(1, (TextTable.Length/4) - 1);
15     WordTable[random, i] = Convert.ToChar(chosenWordChars[i]);
16   }
17 }
```

Listing 13. RandomizeWordTable

4.12. RandomizeText

Sprawia, że do zmiennej ChosenWord wpisywana jest losowa liczba z zakresu od zera do długości tablicy ze słowami. Dzięki temu wylosowane zostało jedno słowo, które zostanie potem użyte jako hasło. RandomizeWordTable()

```
1    RandomizeWordTable()
2    {
3      for (int i = 0; i < (WordTable.Length/4); i++)
4      {
5        for (int j = 0; j < 4; j++)
6        {
7          var nextChar = RandomCharCount.Next(65, 90);
```

```
8         WordTable[i, j] = Convert.ToChar(nextChar);
9       }
10    }
11    char[] chosenWordChars = TextTable[ChosenWord].ToCharArray();
12    for (int i = 0; i < 4; i++)
13    {
14      int random = RandomCharCount.Next(1, (TextTable.Length/4) -
    1);
15      WordTable[random, i] = Convert.ToChar(chosenWordChars[i]);
16    }
17  }
```

Listing 14. RandomizeText

4.13. MoveUp

Ta funkcja zbiera informacje na temat bloku o danym ClassId ¡każdy blok ma swój classId 0-3, classId jest takie same dla przycisku i bloku¿ i wysyła je do funkcji ChangeBlockText() z parametrem up.

```
1  void MoveUp(object sender, EventArgs e)
2  {
3    var Button = (Button)sender;
4    var Name = "Block" + Button.ClassId.ToString();
5    var whichBlock = (Label)FindByName(Name);
6    int ClassId = Convert.ToInt32(Button.ClassId);
7    ChangeBlockText(whichBlock, ClassId, "up");
8  }
```

Listing 15. MoveUp

4.14. ChangeBlockText

W zależności od podanego parametru, wartość indeksu BlockCounters[blockNumber] zwiększa się (dla parametru down) lub zmniejsza się (dla parametru up) sprawiając efekt ”przewijania się”po liście znaków.
Jeżeli będziemy chcieli wyjść poza zakres znaków w tablicy, to wartość indeksu zmieni się na pierwszy (dla parametru dow) lub ostatni (dla parametru up)

```
1          void ChangeBlockText(Label block, int blockNumber, string
    where)
2  {
3    if (where == "up")
4    {
```

```
5       if (BlockCounters[blockNumber] == 0)
6       {
7         BlockCounters[blockNumber] = (WordTable.Length / 4) - 1;
8       }
9       else BlockCounters[blockNumber]--;
10
11    }
12    else if (where == "down")
13    {
14      if (BlockCounters[blockNumber] == (WordTable.Length / 4) - 1)
15      {
16        BlockCounters[blockNumber] = 0;
17      }
18      else BlockCounters[blockNumber]++;
19    }
20    block.Text = WordTable[BlockCounters[blockNumber], blockNumber
     ].ToString();
21  }
```

Listing 16. ChangeBlockText

4.15. CheckIfWin

Ta funkcja sprawdza warunek zwycięstwa (ciąg znaków zgadza się z elementem z tablicy WordTable).
W linii 4 ustawiana jest flaga WinFlag jako false (false oznacza brak zgodności, true zgodność).
W liniach 5-10 odczytywana jest wartość z elementów Block0 - Block3, po czym jest to sklejane w jeden ciąg, umieszczony w zmiennej userGuess.
W liniach 11-17 iterujemy po wszystkich elementach tablicy WordTable (tam są słowa możliwe do uzyskania).
W linii 13 umieszczony jest warunek, który sprawdza, czy wybrany przez użytkownika ciąg znaków jest zgodny z elementem z tablicy. Jeśli tak to flaga WinFlag przestawiana jest na ”true”.
W linii 18 sprawdzamy czy flaga ma wartość True. Jeśli tak, oznacza to wygraną. W przeciwnym razie zmniejszana jest liczba serc o 1.

```
1   async void CheckIfWin(object sender, EventArgs e)
2  {
3    var userGuess = "";
4    var WinFlag = false;
5    for (int i = 0; i < 4; i++)
6    {
```

```
7       var Name = "Block" + i;
8       var Block = (Label)FindByName(Name);
9       userGuess += Block.Text;
10    }
11    for (int i = 0; i < TextTable.Length; i++)
12    {
13      if (TextTable[i] == userGuess)
14      {
15        WinFlag = true;
16      }
17    }
18    if (WinFlag)
19    {
20      DisplayAlert("Kod do nastepnej zagadki to: ", "2115", "Ok");
21      await Navigation.PushAsync(new FlashlightPage());
22    }
23    else
24    {
25      setHeartStatus();
26      LoverHeartCount();
27    }
28  }
```

Listing 17. CheckIfWin

4.16. MoveDown

Ta funkcja zbiera informacje na temat bloku o danym ClassId (każdy blok ma swój classId 0-3, classId jest takie same dla przycisku i bloku) i wysyła je do funkcji ChangeBlockText() z parametrem down.

```
1         void MoveDown(object sender, EventArgs e)
2 {
3   var Button = (Button)sender;
4   var Name = "Block" + Button.ClassId.ToString();
5   var whichBlock = (Label)FindByName(Name);
6   int ClassId = Convert.ToInt32(Button.ClassId);
7   ChangeBlockText(whichBlock, ClassId, "down");
8 }
```

Listing 18. MoveDown

5. Testowanie

Tab. 5.1. Opis testów menu głównego i ustawień

Obiekt	Opis działania	Poprawność działania
Duzy czerwony przycisk "wciśnij mnie"	Wyłącza aplikacje	Działa poprawnie
Przycisk ustawień	Włącza menu ustawień	Działa poprawnie
Przycisk przejścia dalej (żarówka)	Włacza prawdziwe menu główne	Działa poprawnie
Przycisk "Przejdź dalej"	Przechodzi dalej	Działa poprawnie
Pasek zmiany głośności (Menu ustawień)	Podczas przesuwania zmienia wartość	Działa poprawnie
Przycisk zatwierdź (Menu ustawień)	Zapisuje wartość ustawioną na pasku i przypisuje ją dla każdej z kart	Działa poprawnie

Tab. 5.2. Opis testów labiryntu

Obiekt	Opis działania	Poprawność działania
Strzałka w lewo (Labirynt)	Postać przemieszcza się w lewo	Działa poprawnie
Strzałka w prawo (Labirynt)	Postać przemieszcza się w prawo	Działa poprawnie
Strzałka w górę (Labirynt)	Postać przemieszcza się w górę	Działa poprawnie
Strzałka w dół (Labirynt)	Postać przemieszcza się w dół	Działa poprawnie
Przejście labiryntu (Wejście na odpowiednie pole)	Przenosi do innej zagadki	Działa poprawnie
Wejście w ścianę	Resetuje pozycję postaci i odbiera jedno z żyć	Działa poprawnie

Tab. 5.3. Opis testów literaków

Obiekt	Opis działania	Poprawność działania
Strzałka w górę (Pierwsza)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w górę (Druga)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w górę (Trzecia)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w górę (Czwarta)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w dół (Pierwsza)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w dół (Druga)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w dół (Trzecia)	Zmienia literę na odpowiednim polu	Działa poprawnie
Strzałka w dół (Czwarta)	Zmienia literę na odpowiednim polu	Działa poprawnie
Przycisk odtwórz	Odtwarza kod morsa za pomocą latarki	Działa poprawnie
Zatwierdzenie hasła	W przypadku podania błędnego hasła zabiera życie natomiast w przypadku podania dobrego hasła przechodzi dalej	Działa poprawnie

6. Podręcznik użytkownika

Spis rysunków

Spis tabel

Spis listingów