WYŻSZA SZKOŁA BANKOWA W GDAŃSKU  
WYDZIAŁ FINANSÓW I ZARZĄDZANIA

Marta Przyborska

43289

**Aplikacja do nauki połączona z grą na platformę mobilną Android**

Praca inżynierska

na kierunku informatyka

Praca napisana pod kierunkiem

dr inż. Mariusza Mola

Gdańsk 2021

Spis treści

[1. Wstęp 5](#_Toc64862194)

[2. Cel i zakres pracy 7](#_Toc64862195)

[3. Część teoretyczna 8](#_Toc64862196)

[3.1. Pokolenie iGen 8](#_Toc64862197)

[3.2. Psychologia nawyku 9](#_Toc64862198)

[4. Część teoretyczna techniczna 11](#_Toc64862199)

[4.1. Android 11](#_Toc64862200)

[4.2. Pliki XML 11](#_Toc64862201)

[4.2.1 TableLayout 11](#_Toc64862202)

[4.2.2 ConstraintLayout 12](#_Toc64862203)

[4.2.3 RelativeLayout 12](#_Toc64862204)

[4.2.4 LinearLayout 12](#_Toc64862205)

[4.2.5 TextView 13](#_Toc64862206)

[4.2.6 EditText 13](#_Toc64862207)

[4.2.7 Spinner 13](#_Toc64862208)

[4.2.8 Button 13](#_Toc64862209)

[4.3. Kotlin 13](#_Toc64862210)

[4.3.1 Klasy 13](#_Toc64862211)

[4.3.2 Typ zmienny (var) 13](#_Toc64862212)

[4.3.3 Typ tylko do odczytu (val) 14](#_Toc64862213)

[4.3.4 Instrukcje warunkowe (if, else) 14](#_Toc64862214)

[4.3.5 Konstrukcja when 14](#_Toc64862215)

[4.3.6 Generowanie liczb pseudolosowych (random) 14](#_Toc64862216)

[*4.4.* *Model-View-Controller* 14](#_Toc64862217)

[5. Część praktyczna 15](#_Toc64862219)

[5.1. Mechanika gry 15](#_Toc64862220)

[5.2. Menu 16](#_Toc64862221)

[5.2.1 Model 16](#_Toc64862222)

[5.2.2 Widok 17](#_Toc64862223)

[5.2.3 Kontroler 20](#_Toc64862224)

[5.3. Wyprawa 24](#_Toc64862225)

[5.3.1 Model 24](#_Toc64862226)

[5.3.2 Widok 26](#_Toc64862227)

[5.3.3 Kontroler 29](#_Toc64862228)

[5.4. Zadania 33](#_Toc64862229)

[5.4.1 Model 33](#_Toc64862230)

[5.4.2 Widok 35](#_Toc64862231)

[5.4.3 Kontroler 37](#_Toc64862232)

[5.5. Postać 41](#_Toc64862233)

[5.5.1 Model 41](#_Toc64862234)

[5.5.2 Widok 43](#_Toc64862235)

[5.5.3 Kontroler 44](#_Toc64862236)

[5.6. Dungeon 45](#_Toc64862237)

[5.6.1 Model 45](#_Toc64862238)

[5.6.2 Widok 48](#_Toc64862239)

[5.6.3 Kontroler 49](#_Toc64862240)

[5.7. Klasa dodatkowa *SharedPreference.kt* 50](#_Toc64862241)

[6. Podsumowanie i wnioski 54](#_Toc64862242)

[6.1. Realizacja założeń 54](#_Toc64862243)

[6.2. Wnioski 54](#_Toc64862244)

[6.3. Możliwości rozwoju aplikacji 54](#_Toc64862245)

[6.3.1 Sklep 54](#_Toc64862246)

[6.3.2 Inne statystyki 54](#_Toc64862247)

[6.3.3 Klasy i umiejętności 55](#_Toc64862248)

[6.3.4 Przedmioty 55](#_Toc64862249)

[6.3.5 Grafika 55](#_Toc64862250)

[6.3.6 Wyższe poziomy 55](#_Toc64862251)

[6.3.7 Serwer gry 55](#_Toc64862252)

[7. Bibliografia 57](#_Toc64862253)

[Spis ilustracji 58](#_Toc64862254)

# Wstęp

Praca dotyczy możliwości łączenia aplikacji rozrywkowych z naukowymi oraz budowania dobrych nawyków przy ich użyciu. Aplikacje z elementami gier, czyli tak zwaną grywalizacją stają się coraz popularniejsze, wspomagając tworzenie dobrych nawyków.

Celem niniejszej pracy jest stworzenie aplikacji będącej połączeniem gry typu RPG oraz aplikacji do nauki matematyki. Aplikacja jest dedykowana dla dzieci z przedziału wiekowego 6 do 10 lat czyli tym w trzech pierwszych klasach szkoły podstawowej. Rozpatrywany jest zakres materiału dydaktycznego ustalony przez odpowiednie instytucje w Polsce.

W części teoretycznej zostało omówione pokolenie iGen dla którego dedykowana jest aplikacja oraz psychologia nawyków.

W części teoretycznej technicznej krótko zostały omówione narzędzia użyte do stworzenia aplikacji.

W części praktycznej została opisana aplikacja oraz szczegółowo omówione jej elementy.

W ostatniej części znajduje się podsumowanie i wnioski.

Na końcu pracy znajduje się bibliografia.

# Cel i zakres pracy

Celem niniejszej pracy jest stworzenie aplikacji będącej połączeniem gry typu RPG oraz aplikacji do nauki matematyki. Aplikacja jest dedykowana dla dzieci z przedziału wiekowego 6 do 10 lat czyli tym w trzech pierwszych klasach szkoły podstawowej. Rozpatrywany jest zakres materiału dydaktycznego ustalony przez odpowiednie instytucje w Polsce.

Aplikacja ma mieć mechanikę wspierającą codzienne rozwiązywanie zadań, a co za tym idzie utrwalanie wiedzy. Model przyznawania punktów za rozwiązane zadanie sprawdził się na korepetycjach z matematyki (doświadczenie autora) a pomysł z rozwijaniem postaci przy użyciu tychże daje uczniowi poczucie sprawczości. Często jest to wystarczający bodziec aby zmienić nastawienie do przedmiotu.

Ze względu na różnice w indywidualnych możliwościach, aplikacja nie będzie wspierać rozwiązywania zadań na czas. Zniechęcenie jest pierwszym podstawowym powodem późniejszych „trudności w nauce” i tworzenia zaległości. Celem aplikacji jest odwrócenie tych wektorów i pokazanie, że nawet mały wysiłek punktuje, a zadania trudniejsze z czasem wydają się łatwe.

Grupa docelowa dla pierwszej wersji aplikacji została wybrana na podstawie doświadczeń autora. Dzieci przechodzące przez próg między trzecią a czwartą klasą szkoły podstawowej mają już ugruntowany stosunek emocjonalny do matematyki i w tym momencie pojawiają się pierwsze osoby uznające się za „słabe” z matematyki. Zakres materiału w polskiej szkole w pierwszych trzech klasach odejmuje dodawanie i odejmowanie w zakresie do 1000, mnożenie i dzielenie w zakresie do 100, podstawy geometrii oraz rozpoznawanie miar i jednostek. Nawet przy tak podstawowych informacjach zaczyna tworzyć się podział na uczniów „dobrych” i „słabych”, który na tym etapie zależy tylko i wyłącznie od poziomu utrwalenia materiału, a więc praktyki.

Aplikacja ma za zadanie wspomóc nawyk regularnego powtarzania oraz wzmocnić pozytywny odbiór matematyki.

# Część teoretyczna

## Pokolenie iGen

Pokolenie iGen (inaczej pokolenie Internetu) jest definiowane jako dzieci urodzone po 1995 roku (niektóre źródła podają po roku 2000).[[1]](#footnote-1) Ich dzieciństwo oraz dorastanie jest nierozerwalnie związane z dostępem do sieci. Dla przedstawicieli tego pokolenia życie wirtualne jest równie realne jak to codzienne, offline. Dodatkowo jako jednostki dorastające „razem z Internetem”, nie wyobrażają sobie życia bez dostępu do sieci, preferowanie cały czas.[[2]](#footnote-2) Technologia jest ich mocną stroną i środowiskiem naturalnym, nieraz od najmłodszych lat. Według badań, 80% dzieci w wieku 7-15 lat posiada własny telefon.[[3]](#footnote-3)

Polska szkoła niewiele zmieniła się po wprowadzeniu Internetu. Główną zauważalną zmianą jest e-dziennik (zazwyczaj Librus) i zmiana polecenia „poszukajcie informacji” na „poszukajcie informacji w Internecie”. Pokolenie iGen jako „urodzone z technologią” często jest też dużo bardziej obeznane w obsłudze komputera czy smartfona niż nauczyciele. Dodatkowo polska podstawa programowa zmienia się powoli i nadal zakłada uczenie się wielu rzeczy na pamięć lub procesowanie danych według określonego wzoru.

Dzieci aktualnie chodzące do szkoły podstawowej często narzekają na poziom lekcji informatyki, na której chciałyby się uczyć rzeczy bardziej zaawansowanych niż oczywiste dla nich zapisywanie danych lub podstawowa obsługa komputera. W systemie, w którym technologia nie jest obecna na lekcjach i wykorzystywana do nauki uczniom daje się sygnał o jej rozrywkowym charakterze. Dodatkowym problemem jest uczenie obsługi konkretnych programów, które zmieniają się w ogromnym tempie, z wersji na wersję, co oznacza, że wiedza tego typu będzie bezużyteczna jeszcze zanim uczeń opuści szkołę.[[4]](#footnote-4)

W aktualnych pandemiczych warunkach szczególnie dobrze widać, że dzieci i młodzież nie są przyzwyczajone do używania technologii i Internetu jako narzędzia do innych niż rozrywkowe celów. Szczególnie dotkliwie odczuwają to najmłodsze dzieci, które wcześniej miały ograniczony czas na „zabawę przed komputerem” a teraz siedzą przed nim po kilka godzin dziennie na zdalnych lekcjach. Kolejnym tematem jest przystosowanie zdalnych lekcji do możliwości dzieci, biorąc pod uwagę, że większość odbywa się w formie wykładu na jakimś komunikatorze, co nie jest najlepszą metodą na tym etapie rozwoju.[[5]](#footnote-5)

## Psychologia nawyku

Zgodnie z książką „Siła Nawyku” Charlesa Duhigga, proces nabywania nawyku można podzielić na cztery etapy:

1. Sygnał
2. Pragnienie
3. Reakcja
4. Nagroda

Ilustrując to przykładem, po zobaczeniu wyskakującego powiadomienia na telefonie (sygnał), użytkownik chce się dowiedzieć czego dotyczy (pragnienie), więc sięga po telefon i sprawdza czego dotyczy (reakcja) czym zaspokaja swoją ciekawość (nagroda).

Jeżeli któryś z elementów zostanie usunięty, nawyk się nie wykształci, lub jeżeli użytkownik już go ma, zacznie zanikać.[[6]](#footnote-6)

Uczenie się jest nawykiem, więc aplikacje do nauki muszą wspierać proces tworzenia nawyku, żeby miały szansę działać zgodnie z założeniami. Idąc krok dalej, James Clear w książce „Atomowe Nawyki” podaje cztery prawa nabywania dobrego nawyku:[[7]](#footnote-7)

1. Uczyń to oczywistym
2. Uczyń to atrakcyjnym
3. Uczyń to łatwym
4. Uczyń to satysfakcjonującym

Podążając za tymi prawami, aplikacja wspierająca tworzenie nawyku powinna być oczywista, atrakcyjna, łatwa i satysfakcjonująca. Dla pokolenia iGen korzystanie z technologii jest oczywiste. Dzięki połączeniu z grą aplikacja jest atrakcyjna, a dzięki prostym zadaniom łatwa. Częścią dającą satysfakcję jest zdobywanie złota i poziomów jako wymiernych znaczników postępu.

# Część teoretyczna techniczna

## Android

Android to system operacyjny na urządzenia mobilne z linuksowym jądrem. Pierwsza wersja Android SDK została opublikowana w 2007 roku i jest rozwijana do teraz.

Android Studio to oficjalne zintegrowane środowisko informatyczne (*IDE*) dla Androida. Od 2019 roku preferowanym językiem do pisania aplikacji na platformę Android i zarazem wspieranym przez Android Studio jest Kotlin.[[8]](#footnote-8)

## Pliki XML

Obraz zawierający tekst, wewnątrz

Opis wygenerowany automatycznie

Dokumentacja 1: XML[[9]](#footnote-9)

Pliki XML umożliwiają tworzenie interfejsu graficznego aplikacji i projektowania układu danych na wyświetlaczu urządzenia.

### *TableLayout*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Dokumentacja 2: TableLayout[[10]](#footnote-10)

Zgodnie z definicją *TableLayout* umożliwia zorganizowane rozmieszczenie elementów widoku w wiersze i kolumny. Zastosowanie tego układu jest najczęściej spowodowane potrzebą podzielenia elementów na wiersze przy pomocy *TableRow.* Tabela wygenerowana przy użyciu tego układu przyjmuje ilość kolumn zależną od ilości elementów w jednym wierszu. Dość typowo dla języków programowania indeksy kolumn zaczynają się od zera. Ilość elementów w wierszach może być różna.

### *ConstraintLayout*

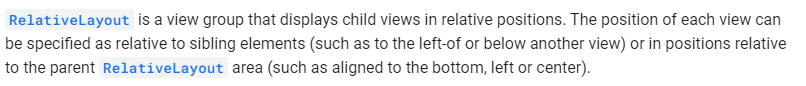
Obraz zawierający tekst

Opis wygenerowany automatycznie

Dokumentacja 3: *ConstraintLayout*[[11]](#footnote-11)

Wzorzec *ConstraintLayout* umożliwia rozmieszczenie elementów używając jako punktów odniesienia brzegów wyświetlacza oraz innych elementów. Jest to domyślny układ przy tworzeniu nowego widoku.

### *RelativeLayout*



Dokumentacja 4: *RelativeLayout[[12]](#footnote-12)*

Zgodnie z definicją *RelativeLayout* to widok, wyświetlający elementy w zależnych od siebie pozycjach. Jego zaletą i podstawowym zastosowaniem jest grupowanie elementów.

### *LinearLayout*



Dokumentacja 5: *LinearLayout[[13]](#footnote-13)*

Wzorzec *LinearLayout* wyrównuje zawartość w jednym kierunku (wertykalnie lub horyzontalnie). Możliwość wybrania dokonuje się przez użycie atrybutu *orientation*.

### *TextView*

*Widget TextView* umożliwia wyświetlanie tekstu nieedytowalnego dla użytkownika. W aplikacjach jest wykorzystywany do wszelkiego rodzaju komunikatów, napisów oraz informacji.

### *EditText*

*Widget EditText* to pole tekstowe do którego użytkownik może wprowadzić dane. Przy ustaleniu typu danych podczas wybrania pola wyświetla się odpowiednia klawiatura. Jest możliwe także wyświetlenie podpowiedzi dla użytkownika.

### *Spinner*

*Widget Spinner* to lista rozwijana. Najbardziej typowe zastosowanie dotyczy wyboru jednej z wielu opcji przy ograniczonym miejscu na wyświetlaczu.

### *Button*

Przyciski umożliwiają obsługę zdarzeń przy kliknięciu. Oprócz pól tekstowych są najbardziej niezastąpionym i intuicyjnym elementem rozmaitych aplikacji.

## Kotlin

### Klasy

Klasy w Kotlinie deklarujemy za pomocą słowa kluczowego *class,* przy czym deklaracja musi zawierać nazwę klasy, nagłówek i ciało (otoczone nawiasami klamrowymi).[[14]](#footnote-14)

### Typ zmienny (*var*)

Zmienne w Kotlinie możemy zadeklarować za pomocą słów kluczowych *var* oraz *val*. Ta pierwsza deklaracja umożliwia późniejszą zmianę wartości, dlatego jest nazywana typem zmiennym.

### Typ tylko do odczytu (*val*)

Po deklaracji przy użyciu słowa kluczowego *val* otrzymujemy stałą, a więc typ tylko do odczytu.

### Instrukcje warunkowe (*if, else*)

Instrukcje warunkowe jak sama nazwa wskazuje służą do sprawdzania czy zachodzą warunki, aby wykonać dany fragment kodu. Jeżeli zadany warunek jest prawdziwy, kod zostanie wykonany, jeżeli nie, pominięty.

### Konstrukcja *when*

Konstrukcja *when* jest odpowiednikiem konstrukcji *switch* występującej w językach z grupy C, a więc rozgałęzioną instrukcją warunkową.

### Generowanie liczb pseudolosowych (*random*)

Klasa abstrakcyjna *Random* służy do generowania pseudolosowych wartości. Z punktu widzenia aplikacji najistotniejsza jest metoda *Random.nextInt* z tej klasy, losująca liczbę z zadanego przedziału.

## *Model-View-Controller*

Model-Widok-Kontroler to wzorzec architektoniczny zaprojektowany przez Trygve Reenskaug w 1979. Model to pewien fragment logiki aplikacji bądź problemu; Widok jak sama nazwa wskazuje to część interfejsu użytkownika wyświetlająca fragment modelu; Kontroler przyjmuje dane od użytkownika i zarządza zmiany w modelu i widoku[[15]](#footnote-15). Android Studio umożliwia tworzenie aktywności złożonych z pliku XML i połączonej z nim klasy napisanej w Javie bądź Kotlinie, więc dość intuicyjnym jest założenie, że plik lub pliki XML odpowiadają za Widok, klasy napisane w Kotlinie za Kontroler, a Modelem w tym przypadku jest część logiki aplikacji którą obsługują dane aktywności.

# Część praktyczna

## Mechanika gry

Ze względu na wiek grupy docelowej aplikacja limituje zadania poprzez konieczność ich zdobywania. W wersji *proof of concept* zadania można zdobyć tylko na wyprawie. Użytkownik wybiera na jaki czas chce iść na wyprawę i po jej ukończeniu dostaje losowe zadania. Z samej wyprawy nie ma żadnego innego zysku, co oznacza, że nie warto jej używać w trakcie korzystania z aplikacji. Zupełnie inaczej wygląda sytuacja, kiedy telefon będzie zostawiony sam sobie, na przykład na noc. Użycie wyprawy na 20 godzin o godzinie 20 gwarantuje, że następnego dnia o godzinie 16 (potencjalnie po lekcjach) można się cieszyć dużą ilością zadań. Za każde zadanie zdobywane są punkty (odpowiednio jeden za zadanie pierwszego poziomu, dwa za zadanie drugiego poziomu itd.). Punkty przydziela się do statystyk postaci aby zwiększyć jej możliwości. Istnieje możliwość przydzielenia punktów w siłę (wzrośnie atak) lub w witalność (wzrosną punkty życia). Te dwie statystyki zwiększają nasze szanse na przeżycie i łupy w Dungeonie. Dungeon to już typowo gra RPG starego typu. W każdym punkcie można zadecydować w jakim kierunku chce się iść, po rozpatrzeniu zdarzenia. Czasem znajduje się skarby, a czasem trzeba się zmierzyć z przeciwnikiem. Na potrzeby PoC, jedynymi zyskami z zamku potworów jest złoto oraz doświadczenie, które po osiągnięciu określonych progów zwiększa poziom.

## Menu

Obraz zawierający tekst, zrzut ekranu, sprzęt elektroniczny, wyświetlanie

Opis wygenerowany automatycznie

Rysunek 1: Menu aplikacji

### Model

Menu pozwala na przejście do pozostałych aktywności w aplikacji, więc jest łącznikiem pomiędzy modelami. Podstawowym celem tego modelu jest podanie informacji o innych modelach oraz możliwość przełączenia się do nich. Z racji tego, że aplikacja jest w całości umieszczona na urządzeniu i nie wymaga dostępu do sieci, menu jest też przestrzenią startową, która wyświetla się zaraz po uruchomieniu aplikacji.

### Widok

Całość widoku została stworzona przy użyciu *TableLayout*. Ten typ wzorca umożliwia tworzenie wierszy w których można zagnieżdżać kolejne schematy.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 1: *TableLayout* w *activity\_main.xml*

Podstawowymi parametrami tego widoku są *layout\_width* oraz *layout\_height,* oba ustawione na wartość *„match\_parent”,* dzięki czemu bez względu na szerokość i wysokość ekranu urządzenia na którym zostanie uruchomiona, aplikacja zajmie cały ekran. *Id* jest podstawowym identyfikatorem elementów pozwalającym się do nich odwoływać. Dzięki zastosowaniu parametru *stretchColumns* kolumny w tabeli mają równą szerokość.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 2: *goToJourneyButton* w *activity\_main.xml*

Parametr *text* pozwala wyświetlić tekst na przycisku a *layout\_margin* ustalić szerokość marginesu – odległości od krawędzi ramki. Użycie parametru *layout\_column* pozwala ustalić w której części wiersza (*TableRow*) znajdzie się przycisk. Pole *backgroundTint* zmienia kolor przycisku, w tym przypadku na zielony. Efektem powyższego kodu jest zielony przycisk z napisem „Wyprawa” zajmujący lewą górną ćwiartkę ekranu przy jednoczesnym jej wypełnieniu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 3: *goToProfile* w *activity\_main.xml*

Analogicznie do poprzedniego przycisku parametry ustalone dla tego elementu powodują wyświetlenie w prawej górnej ćwiartce niebieskiego przycisku z napisem „Postać”. Ze względu na ujednolicenie menu w tym przypadku ponownie margines wynosi 6dp.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 4: *TableRow* w *activity\_main.xml*

Parametr *layout\_weight* użyty w *TableRow* służy do ustalenia proporcji pomiędzy elementami w tabeli. Ponieważ oba wiersze mają równie dużą wagę, ekran jest podzielony równo.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 5: *TableRow* w *acticity\_main.xml*

Analogicznie drugi wiersz wzorca zawiera następne dwa przyciski: żółty „Zadania” i czerwony „Dungeon”, umieszczone odpowiednio w lewej i prawej ćwiartce ekranu.

### Kontroler

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 6: Odczyt danych z zapisu w *MainActivity.kt*

Pierwsza część kontrolera obejmuje odczyt danych z *Shared Preferences*, to jest udostępnionych preferecji, które są miejscem zapisu danych dostępnym z każdej części aplikacji, przechowywanej również po zamknięciu aplikacji. Mechanizm ten został stworzony celowo do przechowywania prostych typów danych, dokładnie takich jak te, na których zostały zadeklarowane zmienne istotne dla funkcjonowania gry (*integer*, *long* oraz *boolean*). Ze względu na wielokrotne użycie w obrębie całej aplikacji obsługa zapisu i odczytu z *Shared Preferences* została przeniesiona do osobnej klasy, opisanej w podpunkcie 5.7.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 7: Obsługa przycisku *gotoJourney* w *MainActivity.kt*

Powyższy kod służy głównie obsłudze kliknięcia we wcześniej opisany zielony przycisk „Wyprawa”. Podczas obsługi zdarzenia oprócz przejścia do następnej aktywności jest też wykonywany zapis wszystkich kluczowych zmiennych.

Analogicznie został napisany kod do obsługi pozostałych trzech przycisków, co obejmują poniższe fragmenty kodu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 8: Obsługa przycisku *goToProfile* w *MainActivity.kt*Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 9: Obsługa przycisku *goToTasks* w *MainActivity.xml*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 10:Obsługa przycisku goToDungeon w MainActivity.xml

Ostatni fragment odpowiada za przywrócenie danych po ponownym otwarciu aplikacji oraz sprawdzenie, czy postać nie jest aktualnie na wyprawie, co jest istotne z punktu widzenia mechaniki gry. Aby umożliwić identyfikację czasu wyprawy, nawet jeżeli proces aplikacji zostanie zakończony, została użyta wbudowana komenda *System.currentTimeMillis()* zwracająca aktualny czas w milisekundach. Względny czas zakończenia wyprawy zostaje przeliczony na punkt zakończenia w czasie realnym i przechowywany jest w zmiennej *journeyEnd.* Po przywróceniudziałania aplikacji (dzięki przeciążonej klasie zagnieżdżonej *onResume()*) starczy porównać aktualny czas oraz wartość przechowywaną w zmiennej, aby dowiedzieć się, czy wyprawa nadal trwa.

Poniższy fragment kodu znajduje się w każdej aktywności poza *OnJourneyActivity.kt,* do której odsyła.

Obraz zawierający tekst

Opis wygenerowany automatycznie

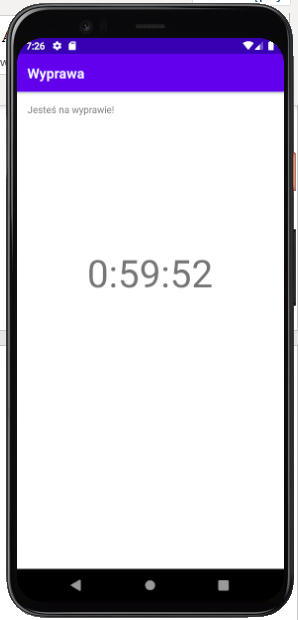
Kod źródłowy 11: *onResume()* w aktywnościach

## Wyprawa

### Model



Rysunek 2: Pierwszy widok Wyprawy



Rysunek 3: Drugi widok Wyprawy

Wyprawa została podzielona na dwie aktywności: pierwszą umożliwiającą wybranie czasu wyprawy z listy rozwijanej i wyruszenie na nią oraz drugą, odliczającą czas pozostały do końca wyprawy. Podstawowa mechanika aplikacji zakłada, że kiedy postać jest na wyprawie, nie da się dostać do żadnej innej części aplikacji, więc tak długo, aż czas nie minie, użytkownik będzie odsyłany z każdej innej części aplikacji do drugiego ekranu wyprawy. Im dłuższa wyprawa, tym więcej zadań dostaje postać.

### Widok

Pierwszy widok składa się z pola tekstowego, listy rozwijanej oraz dwóch przycisków. Pole tekstowe służy informacji o możliwości wyboru czasu wyprawy.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 12: *TextView* w *activity\_journey.xml*

Pole zostało dopasowane do szerokości ekranu i wysokości tekstu za pomocą parametrów *layout\_width* oraz *layout\_height*. Komplet marginesów został ustawiony na *16dp* w celu zwiększenia czytelności. Ze względu na to, że tekst umieszczony w polu nie powtarza się nigdzie w aplikacji, został tam umieszczony bezpośrednio, bez używania przestrzeni *strings.xml*.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 13: *Spinner* w *activity\_journey.xml*

Kolejnym elementem jest lista rozwijana. Tak samo jak pole tekstowe została dopasowana na szerokość do ekranu i dla większej czytelności otoczona marginesami *16dp*. Ze względu na dużą ilość tekstu źródło listy znajduje się w pliku *strings.xml*.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 14: Źródło listy rozwijanej w pliku *strings.xml*

Pierwszy z przycisków otwiera drugi widok Wyprawy, zaś drugi pozwala użytkownikowi wrócić do menu. Oba również zostały dopasowane do szerokości ekranu i mają margines szerokości *16dp.*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 15: Przyciski w *activity\_journey.xml*

Drugi widok składa się z dwóch pól tekstowych. Pierwsze pole, umieszczone w górnej części widoku, informuje o wyprawie, zaś drugie, wyśrodkowane, służy jako wyświetlać dla odliczania czasu do końca wyprawy.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 16: Pola tekstowe w *activity\_on\_journey.xml*

Oba widoki zostały utworzone za pomocą wzorca *ConstraintLayout.*

### Kontroler

Ze względu na podział funkcjonalności na dwie aktywności, dla tej funkcjonalności aplikacji istnieją dwie klasy źródłowe: *JourneyActivity.kt* oraz *OnJourneyActivity.kt*. Pierwsza zajmuje się ustalaniem parametrów, takich jak czas trwania oraz ilość zadań, które postać dostanie w nagrodę. Druga pełni funkcję swoistego minutnika odliczającego czas do końca wyprawy oraz ekranu blokady uniemożliwiającego korzystanie z innych części aplikacji.

Najważniejszą częścią pierwszej aktywności jest obsługa listy rozwijanej, wykonana za pomocą instrukcji warunkowej *when().* Sama lista rozwijana do prawidłowego działania wymaga adaptera, który zwraca pozycję wybranego elementu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 17: Adapter listy rozwijanej w *JourneyActivity.kt*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 18: Obsługa listy rozwijanej w *JourneyActivity.kt*

Warunek wielokrotnego wyboru *when()* obsługuje komplet dwudziestu przypadków w analogiczny do powyższego sposób, wykorzystując przyjmowanie przez zmienną *position* wartości od 0 do 19, zgodnie z ilością elementów we wcześniej opisanej liście. Zmienna *timeInMilliSeconds* wyznacza długość trwania wyprawy a *tasksNr* podbija ilość możliwych do wykonania zadań.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 19: Wczytywanie danych oraz deklaracja przycisków w *JourneyActivity.kt*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 20: Obsługa przycisków w *JourneyActivity.kt*

Pierwszy z przycisków, obsługiwany przez zmienną *buttonGo*, przenosi użytkownika do odliczania czasu na wyprawie, a drugi umożliwia powrót do menu. Obsługa kliknięcia na pierwszy z przycisków obejmuje kalkulację końca wyprawy, zmianę statusu *onJourney* na *true*, dodanie ilości zadań oraz przejście do aktywności *OnJourneyActivity.kt*. Zwiększenie tego ostatniego parametru już na tym etapie nie jest błędem, ze względu na brak możliwości użycia innych funkcjonalności aplikacji tak długo, aż nie upłynie czas wyprawy.

Przycisk powrotu do menu zapisuje dane oraz przenosi użytkownika do MainActivity.kt, która została opisana powyżej.

Druga aktywność to dostosowany minutnik z komentarzem. Tak jak poprzednie aktywności zaczyna od wczytania wszystkich zapisanych danych, po czym zaczyna odliczanie czasu za pomocą obiektu *CountDownTimer*.

Obraz zawierający tekst

Opis wygenerowany automatycznie

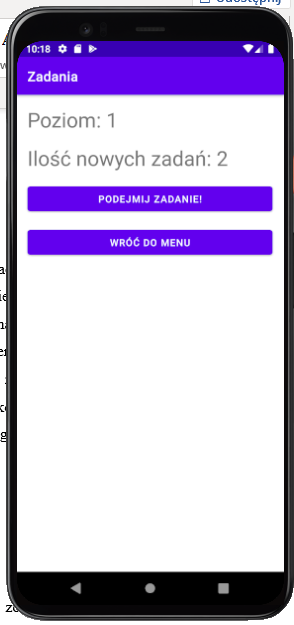
Kod źródłowy 21: Odliczanie czasu w *OnJourneyActivity.kt*

Każda z pozostałych aktywności w aplikacji tak długo jak zmienna *onJourney* ma wartość *true* i czas podróży wciąż nie upłynął, odsyła do tej aktywności.

## Zadania

### Model

Początkowy pomysł dotyczący wykonania aplikacji zakładał losowanie zadania z puli, ale bardzo szybko okazało się, że nawet niewielka ilość zadań niepotrzebnie wydłuża kod oraz zajmuje pamięć RAM. Ze względu na fakt, że zakres materiału dla grupy docelowej obejmuje tylko cztery typy zadania, kierując się metodą prób i błędów udało się utworzyć generator zadań, który uwzględnia zakresy wiedzy adekwatne do poziomów. Zamiast losować gotowe zadania aktualny kod losuje dwie liczby oraz typ działania jakie należy na nich wykonać. Zakres z jakiego zostanie wylosowana liczba jest uzależniony od poziomu (zmienna *lvl*).



Rysunek 4: Pierwszy widok Zadań

Obraz zawierający tekst, monitor, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek 5: Drugi widok zadań

### Widok

Tak samo jak Wyprawa, Zadania również zostały podzielone na dwie aktywności, a co za tym idzie dwa widoki. Pierwszy zawiera dwa pola tekstowe, odpowiednio informujące o poziomie postaci oraz ilości dostępnych zadań, a także dwa przyciski, w tym jeden obsługujący powrót do Menu.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 22: Pole tekstowe *lvlMessage* w *activity\_journey.xml*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 23: Pole tekstowe *tasksMessage* w *activity\_journey.xml*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 24: Przyciski w *activity\_tasks.xml*

### Kontroler

Po naciśnięciu przycisku „Podejmij zadanie” wyświetla się ekran z wylosowanym zadaniem. W tym celu najpierw jest losowany typ zadania z czterech dostępnych (dodawanie, odejmowanie, mnożenie dzielenie) a następnie w zależności od poziomu przypisywane są maksymalne wartości liczb.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 25: Przypisywanie maksymalnych wartości liczb na podstawie poziomu

Poziom pierwszy obejmujący zakres matematyki w klasie pierwszej szkoły podstawowej nie zna jeszcze mnożenia i dzielenia, więc jeżeli wartość jest większa lub równa dwa (czyli zostało wylosowane mnożenie albo dzielenie), trzeba ją skorygować. Na potrzeby aplikacji wykorzystano odejmowanie, ale równie dobrze sprawiła by się operacja modulo. Po przypisaniu maksymalnych wartości są losowane i wyświetlane liczby. Ze względu na możliwe komplikacje liczby przed wyświetleniem są walidowane.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 26: Generator zadań - dodawanie i odejmowanie

Z racji tego, że klasa *random* losuje liczby naturalne, dodawanie nie wymaga walidacji. Przy odejmowaniu należy się upewnić, że rozwiązaniem nie jest liczba ujemna.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 27: Generator zadań - mnożenie i dzielenie

Mnożenie tak jak dodawanie nie wymaga dodatkowych warunków, za to przy dzieleniu trzeba wykluczyć zero jako dzielnik i nie całkowity wynik dzielenia.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 28: Obsługa kliknięcia *solutionButton* w *NewTaskActivity.kt*

Po wpisaniu odpowiedzi przez użytkownika i kliknięciu potwierdzenia, program weryfikuje poprawność odpowiedzi. Jeżeli była poprawna, następuje zwiększenie ilości punktów postaci a zmniejszenie ilości możliwych do podjęcia zadań, jeżeli błędna, nie ma żadnych skutków.

## Postać

### Model

Ekran postaci wyświetla dane o niej, oraz pozwala zwiększać siłę i wytrzymałość przy użyciu punktów za zadania.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 6: Ekran postaci bez możliwości rozdania punktów

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 7: Ekran postaci z możliwością rozdania punktów

### Widok

Ekran Postaci obejmuje tylko jedna aktywność, zbudowana przy użyciu *TableLayout.* Pola tekstowe w tym widoku zostały stworzone tak samo jak reszta pól w aplikacji.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 29: *vitalityButton* w *activity\_profile.xml*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 30: *strengthButton* w *activity\_profile.xml*

Dzięki parametrowi *visibility* z wartością *invisible*, przyciski są ukryte. Są to jedyne dwa przyciski tego typu w aplikacji.

### Kontroler

Punkty postaci mogą być dodawane do siły lub witalności, które pomagają w przejściu Labiryntu. Żeby nie wprowadzać dezorientacji, jeżeli ilość punktów postaci jest zerowa, przyciski nie są widoczne.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 31: Obsługa przypadku, kiedy postać ma punkty i może je rozdać

Wcześniej ustawiony parametr *visibility* umożliwia zmianę przycisków na widoczne kiedy postać zdobywa punkty za zadania. Po wydaniu wszystkich dostępnych punktów przyciski ponownie znikają.

## Dungeon

### Model

Tak samo jak Podróż i Zadania, Dungeon zawiera dwie aktywności. Pierwszą, potwierdzającą, że użytkownik chce wejść do labiryntu i drugą, będącą odwzorowaniem kolejnych pomieszczeń w labiryncie.

Obraz zawierający tekst, zrzut ekranu, monitor

Opis wygenerowany automatycznie

Rysunek 8: Dungeon - pierwsza aktywność

Obraz zawierający tekst, monitor, zrzut ekranu, sprzęt elektroniczny

Opis wygenerowany automatycznie

Rysunek 9: Dungeon - druga aktywność

Biorąc pod uwagę, że Labirynt jest magiczny, mapy się na nic nie zdadzą, a pójście cztery razy pod rząd w prawo nie gwarantuje powrotu do tego samego pomieszczenia. Każde pomieszczenie jest losowane z trzech możliwych przypadków: smoka, skarbu i pustego pomieszczenia. Puste pomieszczenie jest tylko łącznikiem przed następnym, skarb zwiększa złoto postaci, a smok po pokonaniu doświadczenie. Z każdym zdarzeniem danego typu rośnie jego parametr, więc im dalej postać zagłębia się w labirynt, tym więcej zdobywa złota i doświadczenia, ale też spotyka coraz silniejsze smoki.

### Widok

Widok pomieszczeń labiryntu składa się z pola tekstowego, czterech przycisków i kolejnego pola tekstowego.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 32: Przyciski wyboru kierunku w Labiryncie

### Kontroler

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 33: Obsługa przejścia do nowego pokoju w Labiryncie cz. 1

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 34: Obsługa przejścia do nowego pokoju w Labiryncie cz. 2

Każdy z przycisków ma taką samą obsługę zdarzenia, obejmującą losowanie następnego zdarzenia i wynikających z niego nagród. Jeżeli witalność spadnie do zera albo smok okaże się za silny, zdobywanie Labiryntu zostaje przerwane i aplikacja wraca do menu startowego, bez utraty nagród.

## Klasa dodatkowa *SharedPreference.kt*

Ze względu na częste zapisywanie i odczytywanie danych (praktycznie przy każdej zmianie aktywności) z udostępnionych preferencji*,* najpierw zostały utworzone metody odsługujące te dwa zadania, a następnie cała klasa. Zaletami tego rozwiązania są zwiększona czytelność kodu oraz jego mniejsza objętość.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 35: Klasa *SharedPreference.kt*

Klasa zawiera metody: przeciążoną *save()*, *getValueInt()*, *getValueLong()* oraz *getValueBoolean().* W aplikacji nie ma potrzeby użycia innego typu prostego, więc te cztery są wystarczające, żeby zapewnić przetrwanie istotnych danych podczas i poza użytkowaniem aplikacji.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 36: Metoda *save()* w klasie *SharedPreference.kt*

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod źródłowy 37: Metody *getValueInt(), getValueLong()* oraz *getValueBoolean()* w klasie *SharedPreference.kt*

# Podsumowanie i wnioski

## Realizacja założeń

W pracy udało się osiągnąć podstawowe założenia odnośnie utworzenia aplikacji. Ze względu na aktualne warunki nie udało się przetestować aplikacji na większej ilości użytkowników z grupy docelowej, w związku z czym wyniki ankiety są niemiarodajne i nie zostały umieszczone w pracy.

## Wnioski

W trakcie pracy zostały zmienione i udoskonalone początkowe pomysły utworzenia działającej wersji. Najważniejszą zmianą było utworzenie generatora zadań zamiast losowania ich z puli gotowych zadań, która okazała się niezbędna w trakcie pisania kodu. Dzięki temu doświadczeniu zaplanowana metoda działania Dungeonu została podmieniona na analogiczny odpowiednik.

## Możliwości rozwoju aplikacji

### Sklep

W aktualnym stanie aplikacji gracze zdobywają złoto, ale nie są w stanie go wydać. Sklep w którym można sprzedać niepotrzebne łupy, zakupić mikstury leczące czy kupić podstawowe wyposażenie jest nieodłącznym elementem gier typu RPG.

### Inne statystyki

Wprowadzenie mocy, zręczności, szczęścia i wytrzymałości pozwoliło by na stworzenie innych typów postaci (mag, łowca), rozwinięcie mechanizmu zdobywania zadań i łupów (za pomocą szczęścia) oraz na utworzenie drugiego limitu (oprócz punktów zdrowia) czasu spędzonego w zamku potworów. Dodatkowe statystyki postaci wiążą się też z potrzebą myślenia analitycznego i opierania się na rachunku prawdopodobieństwa, więc są dobrą metodą na zwiększenie wartości edukacyjnych aplikacji.

### Klasy i umiejętności

Po wprowadzeniu dodatkowych statystyk istnieje szansa na wprowadzenie klas, które będą się różnić bazową statystyką oraz umiejętności typowych dla klasy, np. mag może leczyć, łowca zadanie dodatkowe obrażenia, wojownik ma premię do punktów życia. Klasy zwiększają atrakcyjność gry.

### Przedmioty

Broń, talizmany, uzbrojenie to duża część gier typu RPG. Oprócz poziomu jest to podstawowy wyznacznik statusu gracza, więc dołożenie możliwości zdobywania bądź tworzenia przedmiotów jest praktycznie nieuniknione podczas rozwoju aplikacji.

### Grafika

W stanie aktualnym aplikacja nie ma atrakcyjnej warstwy graficznej. Jeżeli pojawią się przedmioty część wizualna będzie musiała zostać zmieniona i dostosowana pod nowe wymagania.

### Wyższe poziomy

Aktualnie postać może osiągnąć maksymalnie 3 poziom, co odpowiada III klasie szkoły podstawowej. Docelowo ma obejmować cały zakres szkoły podstawowej. Ze względu na różnice w rozwoju jest też możliwość stworzenia analogicznej aplikacji dla starszych klas, do której będzie można przenieść dane z obecnej wersji.

### Serwer gry

Aplikacja nie ma aktualnie możliwości zapisania na serwerze, z czego wynika kilka kwestii. Po pierwsze, jeżeli coś się stanie wersji na telefonie, nie ma możliwości odzyskania postaci i jej poziomu. Po drugie, nie ma możliwości grania z innymi graczami ani porównywania wyników. Po trzecie, brakuje możliwości zdobycia informacji ile osób korzysta z aplikacji oraz możliwości zgłaszania błędów przez użytkowników. Po czwarte, aplikacja zajmuje dużo miejsca, ponieważ wszystkie zadania i potencjalne zdarzenia z Dungeonu znajdują się w pamięci urządzenia zamiast na serwerze. Ponadto nie ma możliwości migracji wyników na inne urządzenie, czy grania na kilku różnych. Jest to najistotniejszy brak i w miarę rozwoju aplikacji będzie pierwszym celem.

# Bibliografia

1. **Elektronicznej Urząd Komunikacji.** Badanie konsumenckie dzieci i rodziców 2019. *https://www.uke.gov.pl/.* [Online] 13 styczenia 2020. https://www.uke.gov.pl/akt/badanie-konsumenckie-dzieci-i-rodzicow-2019,277.html.

2. **Igielska Beata.** Najmłodsi uczniowie nie radzą sobie w zdalnej nauce, rozwiązań brakuje. *prawo.pl.* [Online] 27 listopada 2020. https://www.prawo.pl/oswiata/zdalne-nauczanie-uczniow-klas-i-iii-problemy-i-skutki,504775.html.

3. **Olejak Karolina.** Szybki Internet nie wystarczy. Czy szkoła powinna być cyfrowa? *klubjagiellonski.pl.* [Online] 3 września 2018. https://klubjagiellonski.pl/2018/09/03/szybki-internet-nie-wystarczy-czy-szkola-powinna-byc-cyfrowa/.

4. **Olszak Teresa.** Pokolenie iGen: jakie jest i co mu zagraża. *polityka.pl.* [Online] 25 maja 2019. https://www.polityka.pl/tygodnikpolityka/spoleczenstwo/1792812,1,pokolenie-igen-jakie-jest-i-co-mu-zagraza.read.

5. **Wilczyńska Agnieszka i Baran Natasza.** Pokolenie iGEN – dorastanie ze smartfonem w plecaku. *https://psychologiawpraktyce.pl/.* [Online] 4 września 2020. https://psychologiawpraktyce.pl/artykul/pokolenie-igen-dorastanie-ze-smartfonem-w-plecaku.

6. **Wikipedia.** Wzmocnienie pozytywne. *wikipedia.pl.* [Online] 25 sierpnia 2018. https://pl.wikipedia.org/wiki/Wzmocnienie\_pozytywne.

7. —. Afiliacja. *wikipedia.pl.* [Online] 15 maja 2018. https://pl.wikipedia.org/wiki/Afiliacja.

8. **developer.android.com.** TableLayout. *https://developer.android.com/.* [Online] 27 grudnia 2019. https://developer.android.com/guide/topics/ui/layout/grid.

9. **developer.android.com/.** ConstraintLayout. *https://developer.android.com/.* [Online] 30 września 2020. https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout.

10. —. Relative Layout. *https://developer.android.com/.* [Online] 18 listopada 2020. https://developer.android.com/guide/topics/ui/layout/relative.

11. **developer.android.com.** Linear Layout. *https://developer.android.com/.* [Online] 18 listopada 2020. https://developer.android.com/guide/topics/ui/layout/linear.

12. **Wikipedia.** XML. *pl.wikipedia.org.* [Online] 9 lutego 2021. https://pl.wikipedia.org/wiki/XML.

13. —. Model-View-Controller. *Wikipedia.pl.* [Online] 9 maja 2020. https://pl.wikipedia.org/wiki/Model-View-Controlle.

14. **Wikipedia.com.** Android Studio. *en.wikipedia.org.* [Online] 13 stycznia 2021. https://en.wikipedia.org/wiki/Android\_Studio.

15. **Kotlin.** Classes. *kotlinlang.org.* [Online] 20 luty 2021. https://kotlinlang.org/docs/classes.html.

16. **Duhigg Charles.** *Siła Nawyku.* brak miejsca : Dom Wydawniczy PWN, 2013.

17. **Clear James.** *Atomowe Nawyki.* brak miejsca : Galaktyka Sp. z o.o., 2018.

## Spis ilustracji

[Kod źródłowy 1: *TableLayout* w *activity\_main.xml* 17](#_Toc64861475)

[Kod źródłowy 2: *goToJourneyButton* w *activity\_main.xml* 17](#_Toc64861476)

[Kod źródłowy 3: *goToProfile* w *activity\_main.xml* 18](#_Toc64861477)

[Kod źródłowy 4: *TableRow* w *activity\_main.xml* 19](#_Toc64861478)

[Kod źródłowy 5: *TableRow* w *acticity\_main.xml* 19](#_Toc64861479)

[Kod źródłowy 6: Odczyt danych z zapisu w *MainActivity.kt* 20](#_Toc64861480)

[Kod źródłowy 7: Obsługa przycisku *gotoJourney* w *MainActivity.kt* 21](#_Toc64861481)

[Kod źródłowy 8: Obsługa przycisku *goToProfile* w *MainActivity.kt* 22](#_Toc64861482)

[Kod źródłowy 9: Obsługa przycisku *goToTasks* w *MainActivity.xml* 22](#_Toc64861483)

[Kod źródłowy 10:Obsługa przycisku goToDungeon w MainActivity.xml 22](#_Toc64861484)

[Kod źródłowy 11: *onResume()* w aktywnościach 23](#_Toc64861485)

[Kod źródłowy 12: *TextView* w *activity\_journey.xml* 26](#_Toc64861486)

[Kod źródłowy 13: *Spinner* w *activity\_journey.xml* 27](#_Toc64861487)

[Kod źródłowy 14: Źródło listy rozwijanej w pliku *strings.xml* 27](#_Toc64861488)

[Kod źródłowy 15: Przyciski w *activity\_journey.xml* 28](#_Toc64861489)

[Kod źródłowy 16: Pola tekstowe w *activity\_on\_journey.xml* 29](#_Toc64861490)

[Kod źródłowy 17: Adapter listy rozwijanej w *JourneyActivity.kt* 30](#_Toc64861491)

[Kod źródłowy 18: Obsługa listy rozwijanej w *JourneyActivity.kt* 30](#_Toc64861492)

[Kod źródłowy 19: Wczytywanie danych oraz deklaracja przycisków w *JourneyActivity.kt* 31](#_Toc64861493)

[Kod źródłowy 20: Obsługa przycisków w *JourneyActivity.kt* 31](#_Toc64861494)

[Kod źródłowy 21: Odliczanie czasu w *OnJourneyActivity.kt* 32](#_Toc64861495)

[Kod źródłowy 22: Pole tekstowe *lvlMessage* w *activity\_journey.xml* 36](#_Toc64861496)

[Kod źródłowy 23: Pole tekstowe *tasksMessage* w *activity\_journey.xml* 36](#_Toc64861497)

[Kod źródłowy 24: Przyciski w *activity\_tasks.xml* 37](#_Toc64861498)

[Kod źródłowy 25: Przypisywanie maksymalnych wartości liczb na podstawie poziomu 38](#_Toc64861499)

[Kod źródłowy 26: Generator zadań - dodawanie i odejmowanie 39](#_Toc64861500)

[Kod źródłowy 27: Generator zadań - mnożenie i dzielenie 40](#_Toc64861501)

[Kod źródłowy 28: Obsługa kliknięcia *solutionButton* w *NewTaskActivity.kt* 41](#_Toc64861502)

[Kod źródłowy 29: *vitalityButton* w *activity\_profile.xml* 44](#_Toc64861503)

[Kod źródłowy 30: *strengthButton* w *activity\_profile.xml* 44](#_Toc64861504)

[Kod źródłowy 31: Obsługa przypadku, kiedy postać ma punkty i może je rozdać 45](#_Toc64861505)

[Kod źródłowy 32: Przyciski wyboru kierunku w Labiryncie 48](#_Toc64861506)

[Kod źródłowy 33: Obsługa przejścia do nowego pokoju w Labiryncie cz. 1 49](#_Toc64861507)

[Kod źródłowy 34: Obsługa przejścia do nowego pokoju w Labiryncie cz. 2 50](#_Toc64861508)

[Kod źródłowy 35: Klasa *SharedPreference.kt* 51](#_Toc64861509)

[Kod źródłowy 36: Metoda *save()* w klasie *SharedPreference.kt* 52](#_Toc64861510)

[Kod źródłowy 37: Metody *getValueInt(), getValueLong()* oraz *getValueBoolean()* w klasie *SharedPreference.kt* 52](#_Toc64861511)

1. (5) Pokolenie iGEN – dorastanie ze smartfonem w plecaku [↑](#footnote-ref-1)
2. (5) Pokolenie iGen: jakie jest i co mu zagraża [↑](#footnote-ref-2)
3. (2) Badanie konsumenckie dzieci i rodziców 2019 [↑](#footnote-ref-3)
4. (4) Szybki Internet nie wystarczy. Czy szkoła powinna być cyfrowa? [↑](#footnote-ref-4)
5. (3) Najmłodsi uczniowie nie radzą sobie w zdalnej nauce, rozwiązań brakuje [↑](#footnote-ref-5)
6. (16) Siła Nawyku [↑](#footnote-ref-6)
7. (17) Atomowe Nawyki [↑](#footnote-ref-7)
8. (14) Android Studio [↑](#footnote-ref-8)
9. (12) XML [↑](#footnote-ref-9)
10. (8) TableLayout [↑](#footnote-ref-10)
11. (9) ConstraintLayout [↑](#footnote-ref-11)
12. (10) RelativeLayout [↑](#footnote-ref-12)
13. (11) LinearLayout [↑](#footnote-ref-13)
14. (15) Classes [↑](#footnote-ref-14)
15. (13)Model-View-Controller [↑](#footnote-ref-15)