STWÓRZ GRĘ Aplikacje Mobilne

ES6+ JavaScript
React Native - Hooks

na podstawie kodów gry: Falling Jumping Shapes

Wydawnictwo: poswojsku.pl

Wydawnictwo poswojsku.pl

1. dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne, poprawne oraz rzetelne,
2. nie ponosi żadnej odpowiedzialności za ewentualne szkody, które mogą wyniknąć z wykorzystania informacji zawartych w tej książce.

Wydawnictwo poswojsku.pl – kontakt:

Strona firmowa: www.poswojsku.pl (tutaj znajdziesz także zasoby graficzne do gry - tła)

e–mail: marketing@poswojsku.pl

ul. Paprocka 86, 98–220 Zduńska Wola

ISBN: 978-83-964647-2-9

Recenzja: Gołębiowski Dariusz

Część I:

Stwórz Grę

ES6+ JavaScript React Native

SPIS TREŚCI

Część II Dodatek:

Aplikacje Mobilne Wprowadzenie

SPIS TREŚCI

WSTĘP

Witaj w poradniku wydawnictwa poswojsku.pl, dzięki któremu wspólnie zaprogramujemy grę mobilną Falling Jumping Shapes. Stworzymy projekt od pomysłu przez grafikę, aż do kompleksowego zaprogramowania w najwspanialszym języku programowania: JavaScript. Aby jednak to osiągnąć, najpierw nauczę Ciebie podstaw kilku niesamowitych technologii. Większość z nich należy do tak zwanych aplikacji open source, czyli jak podaje Wikipedia:

„Open source to kod źródłowy, który jest swobodnie udostępniany do ewentualnej modyfikacji i redystrybucji. Produkty obejmują pozwolenie na użycie kodu źródłowego, dokumentów projektowych, lub zawartości produktu. Model open source to zdecentralizowany model rozwoju oprogramowania, który zachęca do otwartej współpracy. Główną zasadą rozwoju oprogramowania open source jest produkcja równorzędna, z produktami takimi jak kod źródłowy, plany i dokumentacja swobodnie dostępnymi publicznie. „

Źródło powyższego cytatu oraz dużo więcej na temat zagadnień związanych ze światem open source znajdziesz pod adresem: https://en.wikipedia.org/wiki/Open_source . Serdecznie polecam dostępną tam wiedzę.

Zgodnie z tym, co zostało powyżej napisane, mamy pełen dostęp do kodów aplikacji typu open source oraz – zwykle – możemy je używać bezpłatnie w realizowanych projektach deweloperskich. Ale pamiętaj, że wyrażenie „zwykle” nie oznacza „zawsze” i że trzeba pamiętać o ograniczeniach (o ile takie są) w użytkowaniu cudzego oprogramowania. Najprostszym sposobem, aby sprostać wymogom prawnym, jest czytanie licencji oraz opieranie się na znajomości poszczególnych typów licencji. Dla przykładu, przy realizacji projektów IT jednymi z najbardziej przyjaznych są m.in. licencje: GNU General Public License (GPL), BSD (Berkeley Software Distribution Licenses, BSDL) czy MIT.

W trakcie nauki technologii do tworzenia aplikacji na urządzenia mobilne, zrealizujemy proste aplikacje, celem zrozumienia podstawowych zasad związanych z rozwojem programów na urządzenia przenośne. Skupimy się tylko na programowaniu aplikacji z przeznaczeniem na system operacyjny Android.

Technologie, które będziemy poznawali i używali do programowania aplikacji mobilnych, to przede wszystkim:

1. JavaScript wraz z jej najnowszymi wersjami, czyli tzw. ES6+ (więcej znajdziesz na stronach projektu: https://www.ecma–international.org/), gdzie „+” oznacza także kolejne nowsze – ES7, ES8, itd.,
2. React Native – framework do programowania aplikacji mobilnych (więcej znajdziesz na stronach projektu: https://reactnative.dev/)
3. React – framework do programowania aplikacji internetowych (więcej znajdziesz na stronach projektu: https://reactjs.org/)

4. CSS (więcej znajdziesz na stronach projektu: https://www.w3.org/standards/webdesign/htmlcss)

Rozwój tworzonych aplikacji dokonamy wykorzystując między innymi oprogramowanie:

- VSCodium – IDE (Integrated Development Environment czyli Zintegrowane środowisko programistyczne) służące do rozwoju aplikacji (więcej znajdziesz na stronach projektu: https://vscodium.com/) lub możesz użyć inny edytor kodu, np.
  - Visual Studio Code (więcej znajdziesz na stronach projektu: https://code.visualstudio.com/) ,
  - ATOM (więcej znajdziesz na stronach projektu: https://atom.io/),
- Android Studio, które jak można znaleźć na jego oficjalnej stronie „zapewnia najszybsze narzędzia do tworzenia aplikacji na każdym typie urządzenia z Androidem” (więcej znajdziesz na stronach projektu: https://developer.android.com/studio) ,
- System operacyjny Windows (więcej znajdziesz na stronach projektu: https://www.microsoft.com/pl–pl/windows)

Aby bez większego problemu zrozumieć zagadnienia omawiane w tym poradniku, powinnaś/nieneś posiadać pewną wiedzę startową związaną z obszarem IT, w szczególności Co najmniej podstawową wiedzę na tematy:

1. technologie internetowe: HTML, CSS, JavaScript, ES6+,
2. obsługa systemów operacyjnych: Android oraz Windows.

Zakładam, że w obecnych czasach wyżej wymienione systemy operacyjne raczej znasz (przynajmniej w zakresie ich codziennej obsługi), a co do zagadnień z punktu 1., to celem przyswojenia, czy też odświeżenia wiedzy mogę polecić portal:

MDN – https://developer.mozilla.org/pl/docs/Learn – znajdziesz tam mnóstwo przydatnych informacji na temat tzw. technologii webowych.

UWAGA! Teraz powinnaś/nieneś zainstalować czystą instalkę React Native i dalej już na niej pracować, czyli tworzyć w niej kolejne pliki oraz katalogi projektu gry.

Stworzenie pustego, czystego projektu React Native, to polecenie w terminalu:

npx react–native init czystaAplikacja

Uruchomienie testowe czystej instalacji:

npx react–native run–android

UWAGA!

dla osób zupełnie początkujących w programowaniu

Na kilkudziesięciu końcowych stronach tej książki znajduje się poradnik, będący wstępem teoretycznym do budowy aplikacji mobilnych z wykorzystaniem: React Native, JavaScript oraz ES6+. Tak, może to nieco nietypowe, ale zakładam, że być może podstawy developmentu już znasz. Jeżeli jednak tak nie jest, rozpocznij proszę od Część II Aplikacje Mobilne Wprowadzenie, który jak już wspomniałem znajdziesz w drugiej części tej książki.

ROZDZIAŁ 1

OPIS ZAŁOŻEŃ GRY FALLING JUMPING SHAPES

OPIS GRY

WYBRANA TECHNOLOGIA WYKONANIA

ZASOBY: GRAFIKA, DŹWIĘKI

OPIS GRY

Planujemy zbudować bardzo prostą grę, posiadającą trzy poziomy, a właściwie warianty:

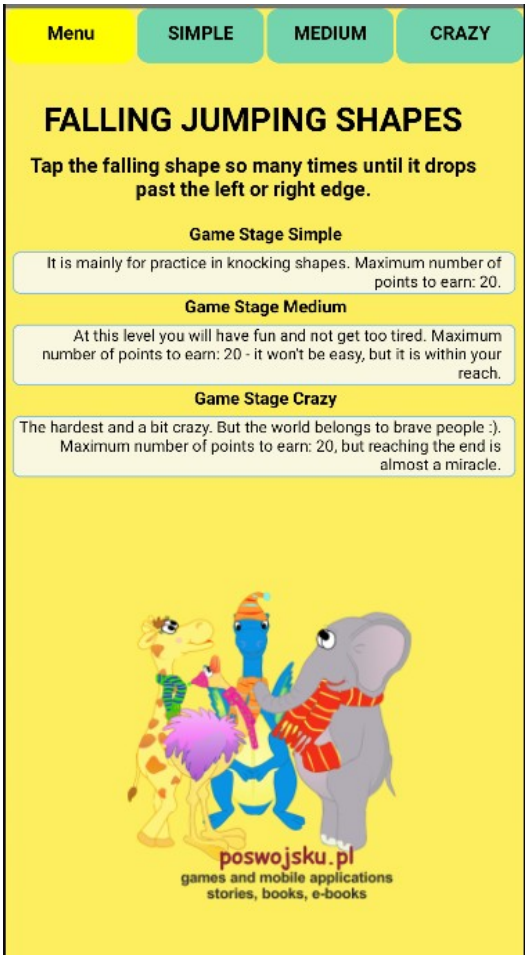
prosty – nazwijmy go SIMPLE oraz przypiszmy do niego oznaczenie ‘zero’

trudniejszy – nazwijmy go MEDIUM oraz przypiszmy do niego oznaczenie ‘one’

nieco szalony – nazwijmy go CRAZY oraz przypiszmy do niego oznaczenie ‘two’

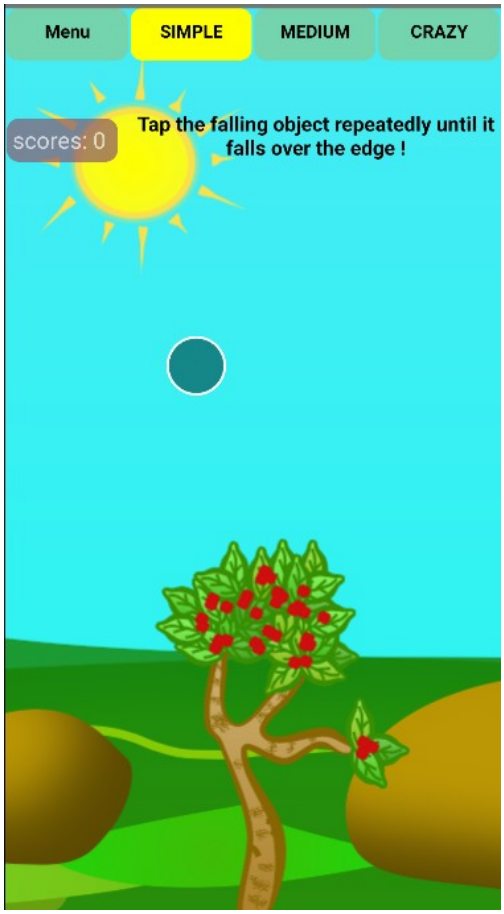
Użyłem określenia ‘warianty’ a nie poziomy, gdyż można uruchomić każdy z nich bez względu na przejście przez poprzedni. Technicznie rzecz ujmując będą to trzy zupełnie niezależne od siebie gry, ale oparte na podobnym wyglądzie oraz grafice.

Oto oczekiwany wygląd:



Na górze strony powinien znaleźć się pasek nawigacyjny składający się z czterech elementów. Oprócz trzech wyżej wymienionych, znajdzie się na nim miejsce także dla pozycji o nazwie Menu, która będzie prowadziła do strony głównej.

Elementy: 2, 3, 4 paska nawigacyjnego będą umożliwiały uruchomienie trzech poziomów, a raczej wersji gry. Zobaczmy przykład na zdjęciu:



Zasady gry są bardzo proste:

A) Wybierając z menu nawigacyjnego jeden z wariantów, Gracz rozpocznie grę,

B) Po kliknięciu w przycisk START GAME pojawia się na górnej krawędzi kulka o losowym kolorze,

C) Kulka opada powoli w dół,

D) Gracz ma za zadanie kliknąć kulkę, a ona za każdym kliknięciem zmienia swoją pozycję w poziomie (w sposób

losowy w lewo lub prawo – zgodnie z logiką zapisaną w kodzie JavaScript),

E) Gdy kulka wyjedzie za lewą lub prawą krawędź – Gracz uzyskuje punkt – trzeba:

- uruchomić Zliczanie Punktów,
- kulka ponownie musi wrócić powyżej górnej krawędzi, aby rozpocząć swobodne spadanie, ale już w innym kolorze oraz z większą szybkością niż poprzednio,
- Gracza trzeba poinformować o liczbie punktów,
- Gracza trzeba poinformować o tym, że idzie mu dobrze,

F) Gdy kulka spadnie poniżej dolnej krawędzi, Gracz nie uzyskuje punktu:

- kulka ponownie musi wrócić powyżej górnej krawędzi, aby rozpocząć spadanie, ale już w innym kolorze,
- Gracza trzeba poinformować o tym, że nie wykonał poprawnie kliknięć w spadający obiekt,

G) Ogranicznik – maksymalna liczba punktów – po uzyskaniu określonej, założonej przez programistę liczby punktów, kończy się gra:

- Gracza trzeba poinformować, że gra się zakończyła bo uzyskał określoną liczbę punktów,
- na ekranie powinien znowu pojawić się napis START GAME, gdyby chciał Gracz zagrać ponownie.

H) INNE ?? tutaj jest miejsce na dalszy rozwój gry – dla Ciebie oraz Twojej kreatywności :).

WYBRANA TECHNOLOGIA WYKONANIA

Gra zostanie wykonana w technologii:

- JavaScript/ES6+/CSS,
- framework React Native z biblioteką React.

ZASOBY: GRAFIKA, DŹWIĘKI

Grafika potrzebna będzie jedynie na tło, gdyż kulka powinna zostać zaprogramowana w taki sposób, aby za każdym razem pojawiała się w innym kolorze. W ten sposób Gracz będzie miał wrażenie, że istnieje więcej niż jedna kulka.

Do gry wykorzystana została grafika z serii bajek dla dzieci: Smok Krolus i Przyjaciele (więcej na temat w.w. serii bajek możesz znaleźć na stronie wydawnictwa www.poswojsku.pl).

ROZDZIAŁ 2

PRZYGOTOWANIE STRUKTURY PROJEKTU

KATALOGI

PLIKI

GRAFIKA

HAKI W REACT WPROWADZENIE

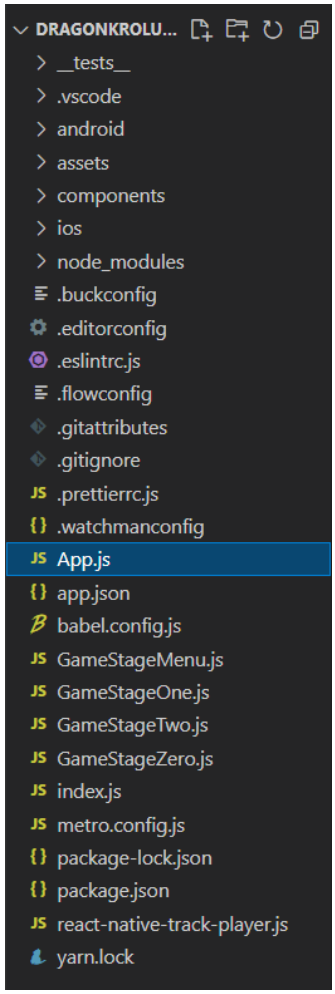
KATALOGI

Po wykonaniu polecenia: npx react–native init nazwaapki

w katalogu głównym aplikacji znajdą się główne pliki planowanej gry mobilnej. Teraz utwórz dwa dodatkowe katalogi:

assets – przechowywał będzie grafiki oraz plik dźwiękowy

components – umieścimy w nim dodatkowe pliki JavaScript.



Zdjęcie: widok plików świeżo zainstalowanej aplikacji

PLIKI

Cztery pliki główne, w których będzie zapisany kod odpowiadający pozycjom z paska nawigacji, znajdą się w katalogu głównym naszej apki, czyli:

App.js – plik główny,

GameStageZero.js – plik zawierający kod poziomu SIMPLE

GameStageOne.js – plik zawierający kod poziomu MEDIUM

GameStageTwo.js – plik zawierający kod poziomu CRAZY

Do każdego z czterech plików będziemy potrzebowali oddzielnego zdjęcia, które będzie tłem. Co do dźwięku, to skorzystamy z dwóch plików, ale opowiem o tym nieco później.

Do każdego z trzech poziomów wykorzystamy ten sam komponent z informacjami tekstowymi – plik TekstInfo.js. Natomiast komponent tworzący spadającą kulkę, stworzymy oddzielenie dla każdego z poziomów, czyli będą to trzy pliki odpowiednio o nazwach:

SquareZero.js

SquareOne.js

SquareTwo.js

GRAFIKA

Jak zapewne już zauważyłeś/aś, gra znajduje się w katalogu DRAGONKROLUSFUN. Nazwa ta wzięła się od serii bajek dla dzieci Dragon Krolus And Friends. Jestem ich twórcą lub jak wolisz autorem :), a firma poswojsku.pl jest ich wydawcą, podobnie jak tego poradnika. Wskazana firma posiada prawa do grafik z wymienionej książeczki i najłatwiej było po prostu skorzystać z pracy, która już została wykonana. Dzięki temu w tle gry znajdują się bajkowe postacie wraz z innymi elementami wspomnianej serii bajek.

Co do spadającej kulki, to generowana jest przez kod napisany w JavaScript, a znajdujący się odpowiednio w:

SquareZero.js

SquareOne.js

SquareTwo.js

a za jej kolor odpowiada kod z plików:

GameStageZero.js

GameStageOne.js

GameStageTwo.js.

Oczywiście powyższe informacje zostaną szczegółowo opisane na kolejnych stronach tego poradnika. Ale tak dla przykładu w pliku GameStageZero.js – kod generujący kolor to:

```
// EFFECT DLA ZMIANY KOLORU DLA KOLEJNYCH POZIOMÓW
```

```
useEffect(() => {
  let tabForBackgroundColor = [];
  for (let j = 0; j < 6; j++) {
    tabForBackgroundColor.push(Math.floor(Math.random() *
10));
  }

  const backgroundSquareNew = `#$
{tabForBackgroundColor[0]}${tabForBackgroundColor[1]}$
{tabForBackgroundColor[2]}${tabForBackgroundColor[3]}$
{tabForBackgroundColor[4]}${tabForBackgroundColor[5]}`;
  // const backgroundSquareNew = 'red';

  setBackgroundSquare(
    backgroundSquare => (backgroundSquare =
backgroundSquareNew),
  );
  // return () => {
  // cleanup
  // }
}, [scoreGameStage0]);
```

Jak widzisz powyżej wewnątrz haka efektów (React) useEffect(), wykorzystując język JavaScript:

1. tworzymy pustą tablicę o nazwie tabForBackgroundColor
2. losujemy 6 cyfr zapełniając nimi stworzoną wcześniej tablicę tabForBackgroundColor
3. przypisujemy elementy wyżej wymienionej tablicy do nowej zmiennej o nazwie: backgroundSquareNew – powstał string, taki jaki się wykorzystuje w CSS do tworzenia stylów
4. ustawiamy zadeklarowany wcześniej reactowy stan o nazwie: setBackgroundSquare na zmienną backgroundSquareNew

Pozostaje powiedzieć kilka słów na temat reactowego haka useEffect(), ale najpierw – ogólnie co to są haki w React.

HAKI W REACT – WPROWADZENIE

W React,js istnieją dwa rodzaje komponentów:

funkcyjne

klasowe.

W początkowych wersjach React, jedynie komponenty klasowe zawierały tak zwany stan. Komponenty funkcyjne były bezstanowe. Ale klasy posiadają wiele problemów, które mogą stanowić bardzo dużą przeszkodę w nauce Reacta. Dla przykładu: trzeba zrozumieć jak działa this w JavaScripcie, pamiętać o wiązaniu (ang. bind) funkcji obsługi zdarzeń (ang. event handlers). Wiele osób (programistów) miało i ciągle ma problem ze zrozumieniem właściwości, stanu oraz kierunku przepływu danych z góry do dołu. A może odwrotnie :)? Tak, klasy stanowią wyzwanie. Zapewne dlatego od wersji React 16.8 zostały wprowadzone tzw. HOOKI (haki), które będąc komponentami funkcyjnymi, dają możliwość przechowywania tak zwanego stanu. Wspomniałem: stanu – czyli właściwie czego? Wyobraź sobie kulkę, która jest zielona, albo czerwona. Czyli ma dwa stany, a hak React'a pomaga w dokonywaniu zmiany z jednego koloru na drugi.

Dwoma najpopularniejszymi hakami są:

hak stanu – useState()

hak efektów – useEffect()

O pierwszym spośród wyżej wymienionych, powiem nieco później, gdy pojawi się on w omawianym kodzie.

Teraz, gdy już wiemy czym są haki w React, przejdźmy do useEffect(). Jak można znaleźć w dokumentacji:

„Hook efektów pozwala na przeprowadzanie efektów ubocznych w komponentach funkcyjnych. …

Pobieranie danych, tworzenie subskrypcji czy ręczna ingerencja w drzewo DOM z wewnątrz komponentów – to wszystko przykłady efektów ubocznych.” – źródło: https://pl.reactjs.org/docs/hooks–effect.html. Wskazana dokumentacja daje nam jeszcze jedną cenną wskazówkę:

„.. metody cyklu życia (ang. lifecycle methods) Reacta, możesz myśleć o hooku useEffect jako o połączeniu metod componentDidMount, componentDidUpdate i componentWillUnmount w jedną.”

W Cyklu Życia Komponentu React, poszczególne metody odpowiadają:

componentDidMount – Montowanie

componentDidUpdate – Aktualizacja

componentWillUnmount – Odmontowanie

Podsumowując:

wywołanie haka useEffect powoduje przekazanie informacji do React, że dany komponent musi wykonać jakąś czynność po jego wyrenderowaniu. Wspaniały React zapamiętuje funkcję przekazaną do haka, a następnie, gdy już zaktualizuje drzewo DOM – wywołuje tę funkcję, czyli wskazanego haka. Powtórzmy to sobie: drzewo DOM zostanie zaktualizowane zanim React wywoła efekty

Pewną wadą jest, że useEffect() zadziała przy każdym renderze, co nie zawsze jest wskazane. Czasami przeprowadzanie efektów przy każdym renderze może stworzyć problemy z wydajnością. Jeżeli pewne wartości pomiędzy renderowaniami się nie zmieniły, to nie ma sensu „odpalać" danego efektu. Aby osiągnąć opisane założenie, możemy do efektu dodać drugi argument:

```
useEffect(() => {
  document.title = `Klknięto ${licznik} razy`;
}, [licznik]);
```

Przy takim zapisie efekt uruchomi się jedynie, gdy pomiędzy poszczególnymi renderowaniami dokonała się zmiana wartości licznik.

W przypadku omawianej gry mamy zmienną scoreGameStage0, która obrazuje nam ilość punktów, czyli efekt zmiany koloru spadającego elementu wykona się, gdy naliczmy kolejny punkt. Gdyby nie dodać ograniczenia liczby renderowań w postaci dodatkowego argumentu do efektu, to otrzymalibyśmy komunikat:

„Warning: Maximum update depth exceeded. This can happen when a component calls setState inside useEffect, but useEffect either doesn't have a dependency array, or one of the dependencies changes on every render." czyli:

„Ostrzeżenie: przekroczono maksymalną głębokość aktualizacji. Może się to zdarzyć, gdy komponent wywołuje setState wewnątrz useEffect, ale useEffect albo nie ma tablicy zależności, albo jedna z zależności zmienia się przy każdym renderowaniu”.

Jak zapewne się domyślasz, taki brak optymalizacji spowodowałby zawieszenie się gry, a tego przecież nie chcemy :). Dlatego właśnie użycie drugiego argumentu w omawianym haku useEffect() jest w przypadku tworzonej gry elementem niezbędnym.

ROZDZIAŁ 3

EKRAN STARTOWY

WYGLĄD STRONY Menu

MENU APLIKACJI

DODATKOWE USTAWIENIA

WYGLĄD STRONY Menu

Główna strona aplikacji, to plik App.js. Kod w nim zawarty rozpoczyna się od importów niezbędnych elementów. Ale zanim opowiemy o importach najpierw spostrzeżenie, że planowana gra powinna działać tylko w ustawieniu pionowym urządzenia mobilnego, gdyż w poziomie, opadająca kulka miałaby zbyt mało miejsca do opadanie, a w związku z tym Graczowi trudno byłoby w nią skutecznie kliknać. Zrobimy blokadę na poziomie Androida – ustawień Landscape – Portrait.

ZABLOKOWANIE ORIENTACJI EKRANU NA TRYB PIONOWY W ANDROID

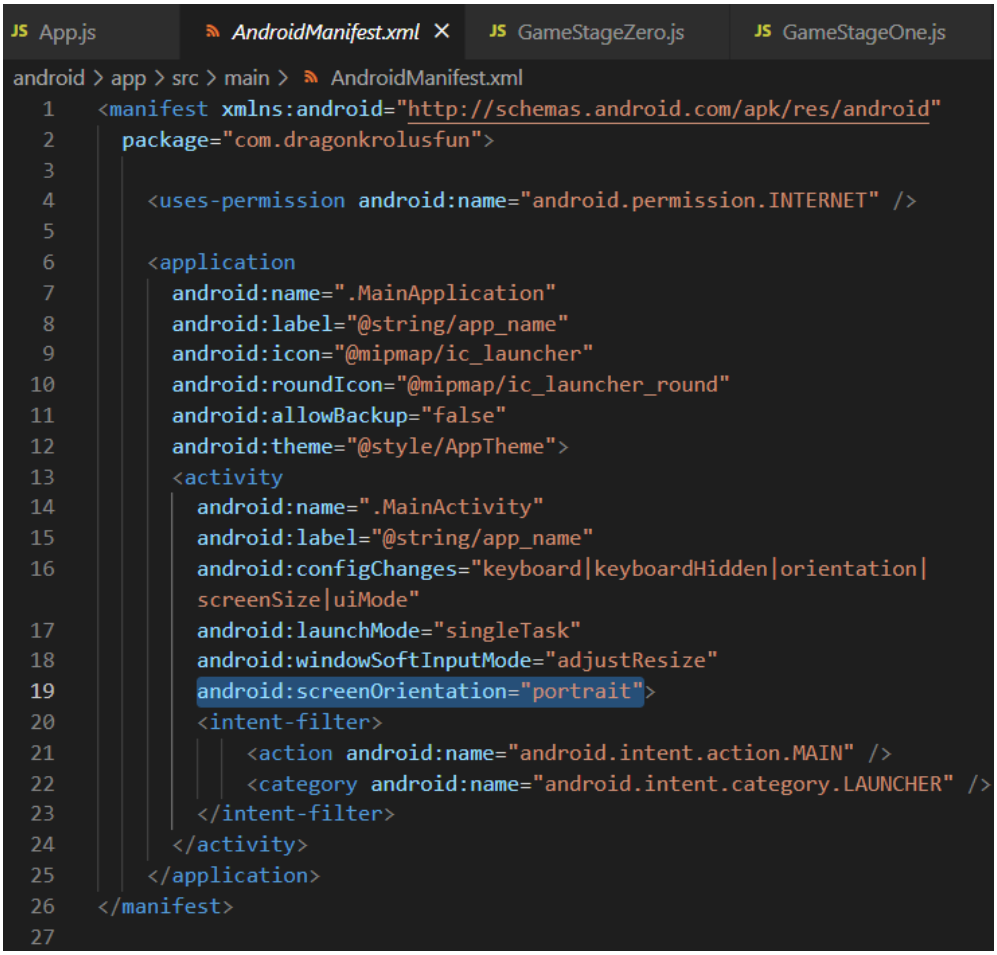
Jednym z najłatwiejszych sposobów ustawienia trybu orientacji pionowej jest dodanie tego fragmentu kodu:
android:screenOrientation="portrait" do tagu aktywności w pliku manifestu Androida: C:\projekty\reactnative\dragonkrolusfun\ android\app\src\main\AndroidManifest.xml , tak jak poniżej

W manifeście ustaw dla wszystkich swoich działań – w elemencie aktywności dodaj:

„android:screenOrientation="portrait"', aby zablokować w orientacji pionowej lub

„android:screenOrientation="landscape"”, aby zablokować w orientacji poziomej.

Zerknij proszę na poniższe zdjęcie, przedstawiające omawiany kawałek kodu aplikacji, a zapewne łatwiej zrozumiesz o co mi chodzi.



Zdjęcie: ustawienie orientacji pionowej w systemie Android

Wróćmy do podstawowego kodu tworzonej wplikacji.

Najpierw musimy zaimportować React oraz z niego dwa haki – useEfect oraz useState:

import React, {useEffect, useState} from 'react';

Następnie importujemy z React Native niezbędne komponenty, czyli to co znajduje się poniżej – wewnątrz nawiasu klamrowego:

```
import {
  StyleSheet,
  Text,
  View,
  Dimensions,
  Pressable,
  ImageBackground,
  SafeAreaView,
} from 'react–native';
```

Ciekawy element to *Dimensions* – pozwala on na pobranie wielkości okna aplikacji.

W kolejnym kroku importujemy komponenty, które zbudują trzy poziomy/wersje gry, czyli – jak poniżej:

```
import GameStageZero from './GameStageZero';
import GameStageOne from './GameStageOne';
import GameStageTwo from './GameStageTwo';
```

Następnym elementem kodu jest stworzenie komponentu głównego, funkcyjnego wraz z jego eksportem – zgodnie z zasadami ES6+:

```
export default function App() {
```

// jakieś dane, np. hooki

return (

<ZwracanyKomponent> – tylko jeden główny element – a wewnątrz niego inne komponenty

)

}

Wewnątrz nawiasów klamrowych znajdzie się ciało głównej funkcji tworzącej komponent o nazwie App. Poniżej funkcji, na końcu pliku znajdzie się jeszcze:

const styles = StyleSheet.create({

}

czyli sposób na uzyskanie odpowiedniego wyglądu.

StyleSheet – jest to abstrakcja podobna do arkuszy stylów CSS. Pozwala na odsunięcie stylu od funkcji render, co z kolei ułatwia zrozumienie tworzonego kodu.

Wykorzystanie StyleSheet do tworzenia wyglądu, to trzy proste kroki.

KROK 1

Import StyleSheet do projektu – zrobiliśmy to na początku pliku w importach, np.:

import {StyleSheet} from 'react–native'

KROK 2

Przerzucenie styli do zmiennej.

Wynikiem tego etapu będzie dla przykładu zapis:

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginBottom: 0,
    paddingBottom: 0,
  },
});
```

KROK 3

Dodanie stylu do komponentu React Native np. do <Text></Text> – czyli zapis:

```
<Text
            style={{
              textAlign: 'center',
              fontWeight: 'bold',
              color: 'black',
              paddingTop: 8,
            }}>Tutaj będzie np. zwykły tekstowym
</Text>
```

Jak widzisz, właściwości styli zapisywane są zgodnie z tak zwanym Camel Case (jak w JavaScript), np.: textAlign (w CSS byłoby: text–align). Warto wspomnieć, że wykonując design w JavaScript napisalibyśmy, np.:

element.style.fontWeight = 'bold';

czyli bardzo podobnie do:

fontWeight: 'bold', .

Hmm, czy już wcześniej nie wspominaliśmy sobie istotny fakt, że w React i React Native wystarczy znajomość JavaScript :). Fajnie, prawda? Jeden, ale za to potężny język programowania.

MENU APLIKACJI

Zarządzanie poszczególnymi komponentami aplikacji zostało napisane z wykorzystaniem haka: useState:

```
// ZARZĄDZANIE POZIOMAMI GRY – MENU


// Zmienne – STANY do zarządzania POZIOMAMI GRY – MENU
  const [stageMenu, setStageMenu] = useState(true);
  const [stageZero, setStageZero] = useState(false);
  const [stageOne, setStageOne] = useState(false);
  const [stageTwo, setStageTwo] = useState(false);
```

W takim razie pora na poznanie prawdopodobnie najważniejszego i najczęściej używanego reactowego haka, czyli useState.

Hak USESTATE()

Wywołanie useState() deklaruje „zmienną stanu”. Jest to sposób na przechowanie wartości pomiędzy wywołaniami funkcji. Hak useState jest innym sposobem wykorzystania tych samych możliwości, jakie daje this.state w klasach. Zwykle w React zmienne „znikają” kiedy funkcja kończy działanie. Tymczasem zmienne stanu pozostają i są przechowywane, celem późniejszego ich wykorzystania.

Hook useState przyjmuje jeden argument, którym jest początkowa wartość stanu. Może nim być na przykład liczba lub ciąg znaków. I co ciekawe – w przeciwieństwie do reactowych klas – stan nie musi być obiektem.

Wywołanie useState() zwraca parę wartości: aktualną wartość stanu i funkcję, która pozwala stan zaktualizować. Dlatego stosujemy zapis:

const [stageMenu, setStageMenu] = useState(true);

Jeżeli znasz klasy React, to możesz zauważyć podobieństwo do właściwości this.state.nazwa i metody this.setState w klasie. Ale my tutaj nie będziemy sięgali do klas, poprzestaniemy na hakach.

Haki ustawiają nam stany w grze, ale jeszcze potrzebujemy funkcje, które będą odpowiadały za dobieranie właściwego menu do danego poziomu gry. W tym celu stworzone zostały zmienne const:

onClickMenu, onClickZero, onClickOne, onClickTwo.

Wykorzystując wartości logiczne: true lub false, ustawiane są właściwości charakterystyczne dla danego poziomu. Jak zobaczysz w poniżej zapisanym kodzie, wykorzystując haki, zmieniane są stany z true na false i odwrotnie, w zależności od poziomu, na którym się znajdujemy.

Zwróć uwagę także na haka ustawiającego stan zmiennej setSomeBackgrImg, czyli zmianę tła dla danego poziomu. Proste prawda?

```
        // Obsługa MENU – POZIOM Ogólnego Menu
        const onClickMenu = () => {
          setStageMenu(stageMenu => true);
          setStageZero(stageZero => false);
          setStageOne(stageOne => false);
          setStageTwo(stageTwo => false);
          // Zmiana tła dla poziomu
          setSomeBackgrImg(someBackgrImg =>
require('./assets/tloAllFriendsLogo387x745.png'));
        };
        // Obsługa MENU – POZIOM 0 SIMPLE
        const onClickZero = () => {
          setStageMenu(stageMenu => false);
          setStageZero(stageZero => true);
          setStageOne(stageOne => false);
          setStageTwo(stageTwo => false);
          setSomeBackgrImg(someBackgrImg =>
require('./assets/backgroundaa374x720.png'));
        };
        // Obsługa MENU – POZIOM 1 MEDIUM
        const onClickOne = () => {
          setStageMenu(stageMenu => false);
          setStageZero(stageZero => false);
```

```
        setStageOne(stageOne => true);
        setStageTwo(stageTwo => false);
        setSomeBackgrImg(someBackgrImg =>
require('./assets/tloRoom2–388x745.png'));
      };
      // Obsługa MENU – POZIOM 2 CRAZY
      const onClickTwo = () => {
        setStageMenu(stageMenu => false);
        setStageZero(stageZero => false);
        setStageOne(stageOne => false);
        setStageTwo(stageTwo => true);
        setSomeBackgrImg(someBackgrImg =>
require('./assets/background374x720.png'));
      };
```

Powyższe kody nie są jakieś zaawansowane. Oczywiście można by je poddać ulepszeniu. Ale krótszy zapis byłby trudniejszy do zrozumienia, a to jest poradnik dla początkujących programistów React Native i JavaScript/ES6+, więc pozostaniemy przy obecnych zapisach kodów.

DODATKOWE USTAWIENIA

Gdy zaczyna się gra, to na poziomie wejściowym ustawiony jest obrazek jako tło. Obsługuje to poniższy kod.

```
// Zmienna ustawiająca tło poziomu gry
const [someBackgrImg, setSomeBackgrImg] = useState(
  require('./assets/tloAllFriendsLogo387x745.png'),
);
```

Podobnie jak w poprzednich przykładach, hak useState pozwala nam na ustawienie stanu początkowego, na wyświetlenie obrazka: ‘tloAllFriendsLogo387x745.png’. Dzięki temu we wspomnianych powyżej ustawieniach dla poszczególnych poziomów gry, hak setState zmienia obrazek na innych.

ROZDZIAŁ 4

EKRAN SIMPLE

plik: GameStageZero.js

WYGLĄD

KULKA – BUDOWA

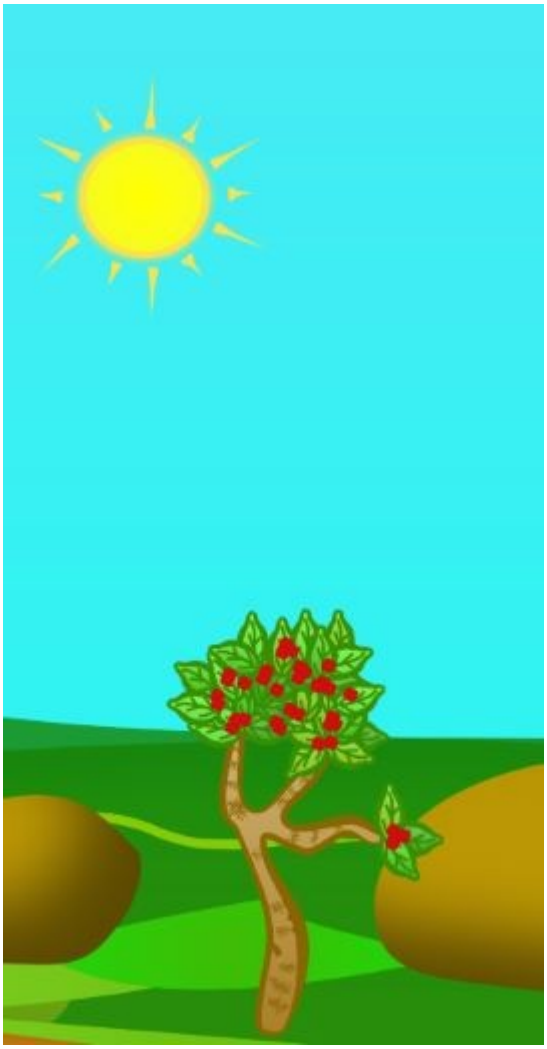
KULKA – ANIMACJA

START GAME

KOMUNIKATY

WYGLĄD

W kodowaniu aplikacji za pomocą technologii React trzeba zwykle na początku danego pliku zaimportować niezbędne elementy.



Zdjęcie: widok tła dla poziomu Zero

TO JEST WERSJA TESTOWA KSIĄŻKI - ZAPRASZAMY DO ZAKUPU PEŁNEJ WERSJI NASZEGO PORADNIKA

PODSUMOWANIE

W tym krótkim poradniku nauczyliśmy się wspólnie:

1. podstaw tworzenia aplikacji mobilnej z wykorzystaniem frameworka React Native (oraz React),
2. dodawania podstawowych elementów aplikacji mobilnej <View> oraz <Text>,
3. zmian w wyglądzie aplikacji mobilnej wykonywanych bezpośrednio w elemencie frameworka React Native – tutaj w <Text></Text>
4. przypisywania stylów do zmiennej, co powoduje, że tworzony kod jest bardziej zrozumiały, elastyczny oraz łatwiejszy do dalszego rozwoju.

PROPOZYCJE DOTYCZĄCE KONTYNUOWANIA NAUKI REACT NATIVE

Szukasz bardziej zaawansowanych materiałów edukacyjnych dotyczących tworzenia aplikacji mobilnych, w tym gier, za pomocą frameworka React Native? Skorzystaj z naszej oferty:

1. książki poswojsku.pl – obecnie dostępny jest: Stwórz Grę Mobilną – eporadnik książkowy z pełnym kodem gry wykonanej w React Native i JavaScript – dostępne w sklepach on-line, m.in.: Google Books,
2. szkolenia indywidualne oraz grupowe – szczegóły znajdziesz na poswojsku.pl lub też wyślij maila z zapytaniem na adres: biuro@poswojsku.pl