Wydział Informatyki i Telekomunikacji
Politechnika Wrocławska

Aplikacja do rozwiązywania nonogramów

Mateusz Wałejko
Nr indeksu: 250335

Praca magisterska napisana
pod kierunkiem
dr. Marcina Michalskiego

Politechnika
Wrocławska

Wrocław 2021

Spis treści

Wstęp

Praca dyplomowa inżynierska jest dokumentem opisującym zrealizowany system techniczny. Praca powinna być napisana poprawnym językiem odzwierciedlającym aspekty techniczne (informatyczne) omawianego zagadnienia. Praca powinna być napisana w trybie bezosobowym (w szczególności należy unikać trybu pierwszej osoby liczby pojedynczej i mnogiej). Zdania opisujące konstrukcję systemu informatycznego powinny być tworzone w stronie biernej. W poniższym dokumencie przykłady sformułowań oznaczono kolorem niebieskim. W opisie elementów systemu, komponentów, elementów kodów źródłowych, nazw plików, wejść i wyjść konsoli należy stosować czcionkę stałej szerokości, np: zmienna `wynik` przyjmuje wartość zwracaną przez funkcję `dodaj(a,b)`, dla argumentów `a` oraz `b` przekazywanych ....

Układ poniższego dokumentu przedstawia wymaganą strukturę pracy, z rozdziałami zawierającymi analizę zagadnienia, opis projektu systemu oraz implementację (dobór podrozdziałów jest przykładowy i należy go dostosować do własnej tematyki pracy).

Wstęp pracy (nie numerowany) powinien składać się z czterech części (które nie są wydzielane jako osobne podrozdziały): zakresu pracy, celu, analizy i porównania istniejących rozwiązań oraz przeglądu literatury, oraz opisu zawartości pracy.

Każdy rozdział powinien rozpoczynać się od akapitu wprowadzającego, w którym zostaje w skrócie omówiona zawartość tego rozdziału.

Praca poświęcona jest wielowarstwowym rozproszonym systemom informatycznym typu „B2B", wspierającym wymianę danych pomiędzy przedsiębiorstwami. Systemy tego typu, konstruowane dla dużych korporacji, charakteryzują się złożoną strukturą poziomą i pionową, w której dokumenty ...

Celem zrealizowanego w ramach pracy projektu było zaprojektowanie i wykonanie aplikacji o następujących założeniach funkcjonalnych:

- wspieranie zarządzania obiegiem dokumentów wewnątrz korporacji z uwzględnieniem ...,
- wspieranie zarządzania zasobami ludzkimi z uwzględnieniem modułów kadrowych oraz ...,
- gotowość do uzyskania certyfikatu ISO ...,
- ....

Istnieje szereg aplikacji o zbliżonej funkcjonalności: ..., przy czym ....

Praca składa się z czterech rozdziałów. W rozdziale pierwszym omówiono strukturę przedsiębiorstwa ..., scharakteryzowano grupy użytkowników oraz przedstawiono procedury związane z obiegiem dokumentów. Szczegółowo opisano mechanizmy .... Przedstawiono uwarunkowania prawne udostępniania informacji ..., w ramach ....

W rozdziale drugim przedstawiono szczegółowy projekt systemy w notacji UML. Wykorzystano diagramy .... Zawarto w niej w pseudokod algorytmów generowania oraz omówiono jego właściwości. ....

W rozdziale trzecim opisano technologie implementacji projektu: wybrany język programowania, biblioteki, system zarządzania bazą danych, itp. Przedstawiono dokumentację techniczną kodów źródłowych interfejsów poszczególnych modułów systemu. Przedstawiono sygnatury metod publicznych oraz ....

W rozdziale czwartym przedstawiono sposób instalacji i wdrożenia systemu w środowisku docelowym.

Końcowy rozdział zawiera podsumowanie uzyskanych wyników.

Analiza problemu

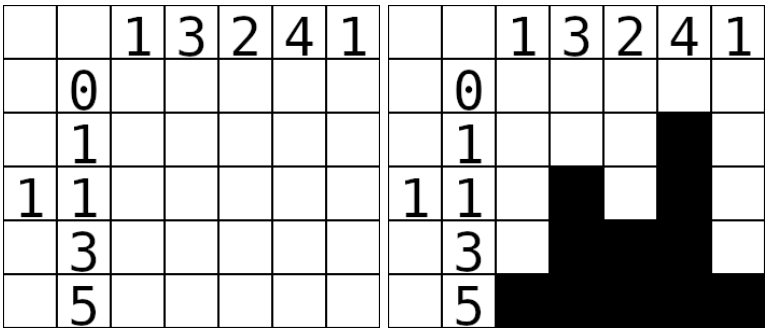
W tym rozdziale przedstawiona jest definicja nonogramów oraz sposób ich rozwiązywania. W kolejnej części rozdziału wypisane są także definicje potrzebne do zrozumienia złożoności problemu, jakim jest rozwiązywanie nonogramów.

2.1 Przedstawienie nonogramów

2.1.1 Opis

Nonogramy (znane także jako *Paint by Number* oraz *Picross*) to łamigłówki, w których celem jest odpowiednie wypełnienie komórek na siatce tak, by uzyskać określony wzór (np. obrazek). W tym celu należy kolorować pola zgodnie ze wskazówkami umieszczonymi obok każdego wiersza oraz kolumny na siatce. Wskazówki mają postać ciągu liczb - każda z liczb oznacza ilość wypełnionych komórek z rzędu, a pomiędzy grupami wypełnionych komórek znajduje się przynajmniej jedna pusta komórka.

Uściślając powyższą, nieformalną definicję, nonogram to łamigłówka na siatce wielkości $w \times h$, gdzie w oznacza szerokość planszy wyrażoną w ilości komórek, a h wysokość planszy wyrażoną w ilości komórek. Dla każdego wiersza i kolumny mamy przedstawiony ciąg liczb H_n będący wskazówką dla danej linii. Dany element h_i opisuje blok stworzony z h_i wypełnionych komórek z rzędu, i jeśli h_i nie jest pierwszym elementem ciągu, to blok opisany przez h_i jest oddzielony przynajmniej jedną pustą komórką od bloku opisanego przez h_{i-1}, oraz jeśli h_i nie jest ostatnim elementem ciągu, to blok opisany przez h_i jest oddzielony przynajmniej jedną pustą komórką od bloku opisanego przez h_{i+1}. Linię spełniającą zadaną wskazówkę opisuje wyrażenie regularne: $l = \hat{}0^*1^{h_1}0^+1^{h_2}0^+ \ldots 0^+1^{h_n}0^*\$$, gdzie 0, 1 oznaczają odpowiednio pustą i wypełnioną komórkę, h_i jest i-tym elementem ciągu H_n, będącego wskazówką dla wiersza/kolumny, a $|l| = n$, $n = w$ dla wiersza i $n = h$ dla kolumny. Rozwiązaniem nonogramu jest wypełnienie zadanej planszy w taki sposób, by dla każdego wiersza oraz każdej kolumny wskazówki dla nich były spełnione.



(a) przed rozwiązaniem (b) po rozwiązaniu

Rysunek 2.1: Przykładowa plansza

2.1.2 Definicje

Definicja 2.1 *Wskazówką nazywamy ciąg H_i liczb, opisujący ułożenie wypełnionych komórek w danej linii.*

Uwaga. Dla uproszczenia opisu algorytmów następuje założenie, że pusta linia jest opisywana przez wskazówkę będącą ciągiem pustym.

Definicja 2.2 *Instancją problemu nonogramu jest czwórka $N = (w, h, R_n, C_n)$, gdzie w, h to odpowiednio szerokość i wysokość planszy, a R_n i C_n są ciągami wskazówek dla wierszy oraz kolumn.*

2.2 Wymagane zagadnienia matematyczne

2.2.1 Problem decyzyjny

Definicja 2.3 *Problem decyzyjny to problem, na który odpowiedź stanowi 'tak' lub 'nie'.*

Problem decyzyjny to pojęcie kluczowe dla klasyfikacji problemu do wybranej klasy złożoności. By móc zaklasyfikować wybrany problem (np. rozwiązanie nonogramu) do jakiejś klasy złożoności, należy przedstawić go w postaci problemu decyzyjnego.

Przykład 2.1 *Czy zadany nonogram $N = (w, h, R_n, C_n)$ ma rozwiązanie?*

2.2.2 Klasa złożoności P

Definicja 2.4 *Klasa złożoności P zawiera wszystkie problemy decyzyjne, których rozwiązanie można znaleźć w czasie wielomianowym.*

Przykład 2.2 *Znalezienie najkrótszej ścieżki między dwoma punktami w grafie należy do klasy P, ponieważ algorytm Dijkstry znajduje najkrótszą ścieżkę w czasie wielomianowym. Sformułowanie tego problemu w postaci problemu decyzyjnego mogłoby brzmieć następująco: Czy dla danego wejściowego grafu $G = (V, E)$ istnieje ścieżka z punktu $v_1 \in V$ do punktu $v_2 \in V$ o długości niewiększej niż x?*

2.2.3 Klasa złożoności NP

Definicja 2.5 *Klasa złożoności NP zawiera wszystkie problemy decyzyjne, których rozwiązanie dla odpowiedzi pozytywnej można zweryfikować w czasie wielomianowym.*

Przykład 2.3 *Mając zbiór I przedmiotów, gdzie przedmiot i_n to dwójka (v_n, w_n), gdzie v_n to wartość, a w_n to waga, oraz ograniczenie górne na sumę wag wybranych przedmiotów w_{max}, czy można wybrać przedmioty w taki sposób by nie przekroczyć limitu wagi w_{max}, a by suma wartości wybranych przedmiotów była większa lub równa c?*

Tak zadany problem to wersja decyzyjna problemu plecakowego. Być może nie istnieje algorytm znajdujący przydział przedmiotów w czasie wielomianowym, ale mając przedstawione rozwiązanie $S \subseteq I$ można zsumować wartości przedmiotów z S i sprawdzić czy jest to poprawne rozwiązanie.

Należy zauważyć, że każdy problem z klasy P należy także do klasy NP, ponieważ rozwiązanie problemu decyzyjnego jest jednym ze sposobów weryfikacji poprawności jego rozwiązania. To czy $P = NP$ jest jak do tej pory nierozwiązanym problemem.

2.2.4 Redukcja wielomianowa

Definicja 2.6 *Problem A jest redukowalny do problemu B w czasie wielomianowym, jeśli wejścia dla problemu A można przekształcić na wejścia dla problemu B w czasie wielomianowym, a następnie rozwiązać problem A wywołując procedurę rozwiązującą problem B wielomianową ilość razy.*

Przykład 2.4 *Można zdefiniować mnożenie liczb $a \cdot b$ za pomocą operacji dodawania w następujący sposób:*

$$a \cdot b = \underbrace{a + a + \ldots + a}_{b}$$

Należy zauważyć, że jeśli problem A jest redukowalny do problemu B w czasie wielomianowym, a problem B należy do klasy P, to także problem A należy do klasy P, jako że sposobem na jego rozwiązanie jest użycie redukcji wielomianowej by traktować go jako instancję problemu B, a następnie rozwiązanie go za pomocą algorytmu działającego w czasie wielomianowym.

Wniosek 2.1 *Jeśli istnieje redukcja z A w B w czasie wielomianowym, to B jest co najmniej tak złożony jak A.*

2.2.5 Klasa problemów NP-trudnych

Definicja 2.7 *Problem H należy do klasy problemów NP-trudnych, jeśli każdy problem w klasie NP jest redukowalny do H w czasie wielomianowym.*

W przypadku klasy problemów NP-trudnych nie ma wymogu, by należące do niej problemy były problemami decyzyjnymi.

Przykład 2.5 *Przykładem problemu NP-trudnego jest problem spełnialności (SAT): 'Czy dla danej formuły logicznej istnieje wartościowanie, dla którego zadana formuła jest spełniona?'. Przynależność tego problemu do tej klasy została udowodniona w 1971 roku przez Stephena Cooka i Leonida Levina w dowodzie twierdzenia Cooka-Levina [1].*

Dla udowadniania przynależności problemu do tej klasy kluczowa jest obserwacja, że istnienie redukcji wielomianowej z A w B implikuje przynależność B do tej klasy, o ile A także do niej należy.

2.2.6 Klasa problemów NP-zupełnych

Definicja 2.8 *Problem decyzyjny C należy do klasy problemów NP-zupełnych, jeśli należy do klas problemów NP-trudnych oraz NP.*

Wniosek 2.2 *Pokazanie, że problem decyzyjny A jest NP-zupełny sprowadza się do pokazania, że istnieje redukcja wielomianowa z problemu H z klasy problemów NP-trudnych, oraz że rozwiązanie problemu A można zweryfikować w czasie wielomianowym.*

2.3 Przypisanie problemu rozwiązania nonogramów do odpowiedniej klasy złożoności

Mając zdefiniowane pojęcia potrzebne do klasyfikacji problemu do odpowiedniej klasy złożoności, należy znaleźć klasę do jakiej należy rozwiązywanie nonogramów. Z uwagi na specyfikę klas, klasyfikacji poddana zostanie decyzyjna wersja problemu, tj. '*Czy zadany nonogram $N = (w, h, R_n, C_n)$ ma rozwiązanie?*'.

2.3.1 Problem rozwiązania nonogramu jest w NP

Niech $M_{h,w}$ będzie macierzą oznaczająca rozwiązanie zadanego nonogramu $N = (w, h, R_n, C_n)$, gdzie $m_{i,j}$ oznacza stan komórki w wierszu i i kolumnie j, oraz $m_{i,j} = 1$, jeśli komórka jest wypełniona, a $m_{i,j} = 0$ jeśli pusta. Macierz $M_{h,w}$ jest mapowana na $h + w$ list, będących ciągami stanów komórek w kolejnych wierszach, i kolumnach.

Do weryfikacji rozwiązania użyjemy następującej procedury:

Pseudokod 2.1: Poprawność rozwiązania w osi

```
Input: Lista linii L, lista wskazówek LH, długość linii n
Output: Poprawność rozwiązania w osi (true/false)
1  for i ← 1 to |L| do
2      Li ← L[i];
3      Hi ← LH[i];
4      a ← 0;
5      A ← [];
6      for c ∈ Li do
7          if c = 1 then
8              a ← a + 1;
9          else
10             if a > 0 then
11                 A.push(a);
12                 a ← 0;
13     if a > 0 then
14         A.push(a);
15     for j ← 1to|Hi| do
16         if A[j] ≠ Hi[j] then
17             return false;
18 return true;
```

W procedurze 2.1 następuje weryfikacja rozwiązania w danej osi. Przykładowo, wywołując procedurę 2.1 dla listy wierszy i ich wskazówek, weryfikujemy Poprawność rozwiązania w poziomie. Weryfikacja rozwiązania następuje przez wywołanie procedury dwukrotnie, dla wierszy oraz kolumn. Jeśli w obu przypadkach procedura zwróci `true`, to rozwiązanie jest poprawne.

Czas wykonania procedury jest zależny od wielkości planszy. Zewnętrzna pętla wykonuje się tyle razy, ile jest linii w osi (h w przypadku wierszy, w w przypadku kolumn). Na początku pętli dochodzi do ekstrakcji pewnych danych do lokalnych zmiennych oraz inicjalizacji tablicy - w zależności od jezyka użytego do implementacji, ta grupa operacji zajmuje czas stały bądź liniowy. Następnie uruchamiana jest pierwsza wewnętrzna pętla. W tej pętli analizowane są dane w danej linii, by zmapować układ jej komórek do wskazówki jaką reprezentuje. Złożoność operacji w każdej iteracji jest stała, jeśli założymy że powiększenie tablicy o dodatkowy element wymaga stałego czasu - w p.p. czas wykonania iteracji może być liniowy. Ilość wykonań tej pętli zależy od długości linii. Po wykonaniu pierwszej pętli, w zależności od układu stanu komórek w linii, może dojść do kolejnego powiększenia tablicy o dodatkowy element - złożoność nie przekracza liniowej. Na końcu zewnętrznej pętli wykonywana jest druga pętla, która iteruje po elementach wskazówki zadanej w rozwiązaniu, i porównuje ich watość do analogicznych elementów we wskazówce odtworzonej z układu linii. Rozbieżność oznacza, że rozwiązanie nie jest prawidłowe, i procedura przedwcześnie zakańcza wykonanie. Długość wskazówki można z góry ograniczyć przez $\lceil \frac{x}{2} \rceil$, gdzie x jest długością linii.

Zadana procedura sprawdza poprawność rozwiązania nonogramów, a jej złożoność, w zależności od implementacji operacji na tablicach, może wynosić $\mathcal{O}(n^2)$ bądź $\mathcal{O}(n^3)$. Zaproponowana procedura ma złożoność wielomianową, zatem problem decyzyjny rozwiązywania nonogramów należy do klasy *NP*.

2.3.2 Problem rozwiązania nonogramu jest NP-trudny

Dowód NP-trudności rozwiązywania nonogramów jest obszerny i wykracza poza zakres tej pracy. Przykładowy dowód jest opisany w pracy [2] i jego zarys jest następujący. Autor rozpoczyna dowód od powołania się na NP-trudność gry na grafach, nazwanej jako *Bounded Nondeterministic Constraint Logic*. Następnie, poprzez redukcję, autor udowadnia NP-trudność zmodyfikowanej wersji gry, określonej na grafach planarnych. Po udowodnieniu tego faktu, autor konstruuje redukcję wielomianową z planarnej *Bounded Nondeterministic Constraint Logic* w rozwiązywanie nonogramów, tym samym udowadniając ich przynależność do tej klasy

problemów.

2.3.3 Problem rozwiązania nonogramu jest NP-zupełny

Pokazawszy, że zadany problem jest w NP, oraz jest NP-trudny, pokazane zostało że problem ten jest NP-zupełny.

Projekt systemu

W tej części opisane zostały wymagania dla aplikacji w kontekście możliwości interakcji przez użytkownika.

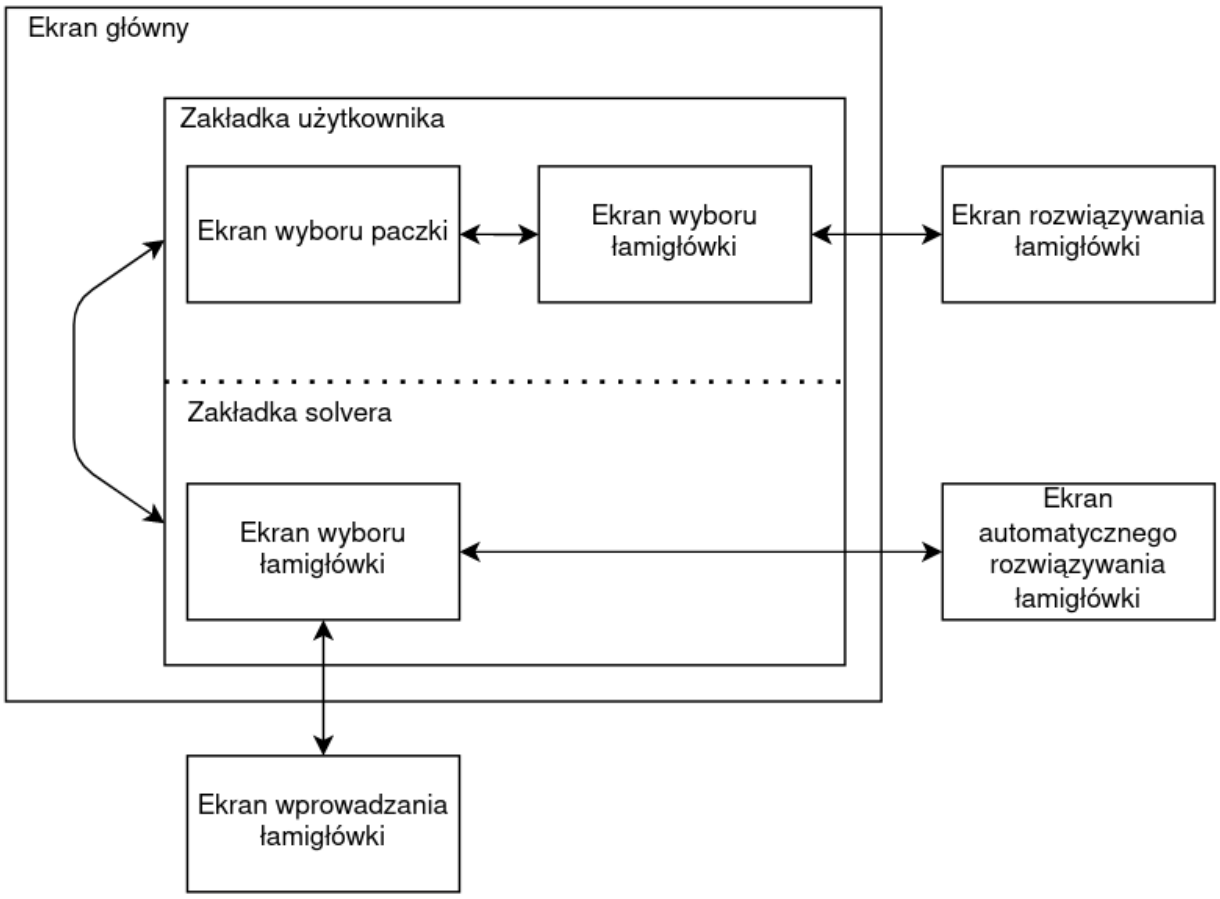
3.1 Wymagania aplikacji

1. Użytkownik może wybrać jedną z predefiniowanych paczek łamigłówek by przejść do wyboru łamigłówki
2. Użytkownik może wybrać jedną z predefiniowanych łamigłówek w paczce by przejść do jej rozwiązywania
3. Użytkownik może rozwiązać łamigłówkę przy użyciu wyświetlanej planszy
4. Użytkownik może oznaczać pola, co do których ma pewność, że są puste
5. Aplikacja śledzi błędy użytkownika w trakcie rozwiązywania łamigłówki i przerywa jej rozwiązywanie w przypadku popełnienia zbyt wielu błędów
6. Postęp rozwiązywania łamigłówki jest zapisywany przy wyjściu do poprzedniego ekranu
7. W menu wyboru łamigłówki wyświetlany jest stan łamigłówki tak, by można było ocenić:
   - czy została rozpoczęta
   - czy została ukończona bez przegranej
   - czy została ukończona z przegraną
8. Użytkownik może wprowadzić łamigłówkę dla solvera za pomocą ekranu wprowadzania łamigłówki
9. Użytkownik może wybrać łamigłówkę do rozwiązania dla solvera
10. Użytkownik może nakazać rozwiązanie wprowadzonej łamigłówki

3.2 Nawigacja pomiędzy aktywnościami

Aplikacja otwiera się w menu głównym. W menu głównym dostępne są dwie zakładki: zakłada użytkownika i zakładka w solvera. W zakładce użytkownika, użytkownik najpierw wybiera paczkę łamigłówek, a następnie łamigłówkę do rozwiązania, po czym przechodzi do ekranu rozwiązywania. W zakładce solvera dostępna jest lista wprowadzonych i predefiniowanych łamigłówek dla solvera. Użytkownik może przejść do ekranu wprowadzania łamigłówki, by dodać kolejną łamigłówkę, bądź do ekranu auto-rozwiązywania, gdzie nakazuje solverowi rozwiązanie wprowadzonej łamigłówki.

Zależności między opisanymi aktywnościami są ukazane na diagramie.



Rysunek 3.1: Diagram zależności między aktywnościami

Implementacja systemu

W tym rozdziale opisane są technologie i biblioteki użyte do stworzenia aplikacji, jak i schemat użytej bazy danych.

4.1 Opis technologii

4.1.1 React Native

Aplikacja została napisana w bibliotece React Native [3]. Jest to framework umożliwiający tworzenie aplikacji mobilnych, komputerowych oraz internetowych. React Native jest oparty na bibliotece React.js [4], stworzonej przez Jordana Walke, a rozwijanej przez Meta Platforms Inc. oraz społeczność, i jej rozwój także jest nadzorowany przez tę firmę. Framework jest dostępny na licencji MIT.

Głównymi pojęciami potrzebnymi do tworzenia aplikacji w React Native są komponenty oraz stany. Komponenty reprezentują nie tylko podstawowe składniki interfejsu graficznego, jak pola tekstowe czy przyciski, ale także zestawy składników realizujące określoną funkcję, np. plansza do gry lub lista gier. Komponenty opierają się o stan, czyli zestaw informacji przechowywany w komponencie i komunikowany do użytkownika za pomocą interfejsu. Framework odświeża wygląd interfejsu przy zmianie stanu, która najczęściej następuje w wyniku interakcji użytkownika z interfejsem. Wtedy to dochodzi do aktualizacji komponentów korzystających z danego stanu.

Programowanie w React Native odbywa się za pomocą języka skryptowego JavaScript. Znak towarowy należy do Oracle [5], standard jest utrzymywany przez ECMA [6], a uruchamiany jest na różnych silnikach rozwijanych zgodnie ze standardem (np. V8 [7] od Google, czy SpiderMonkey [8] od Mozilli).

4.1.2 Dodatkowe biblioteki

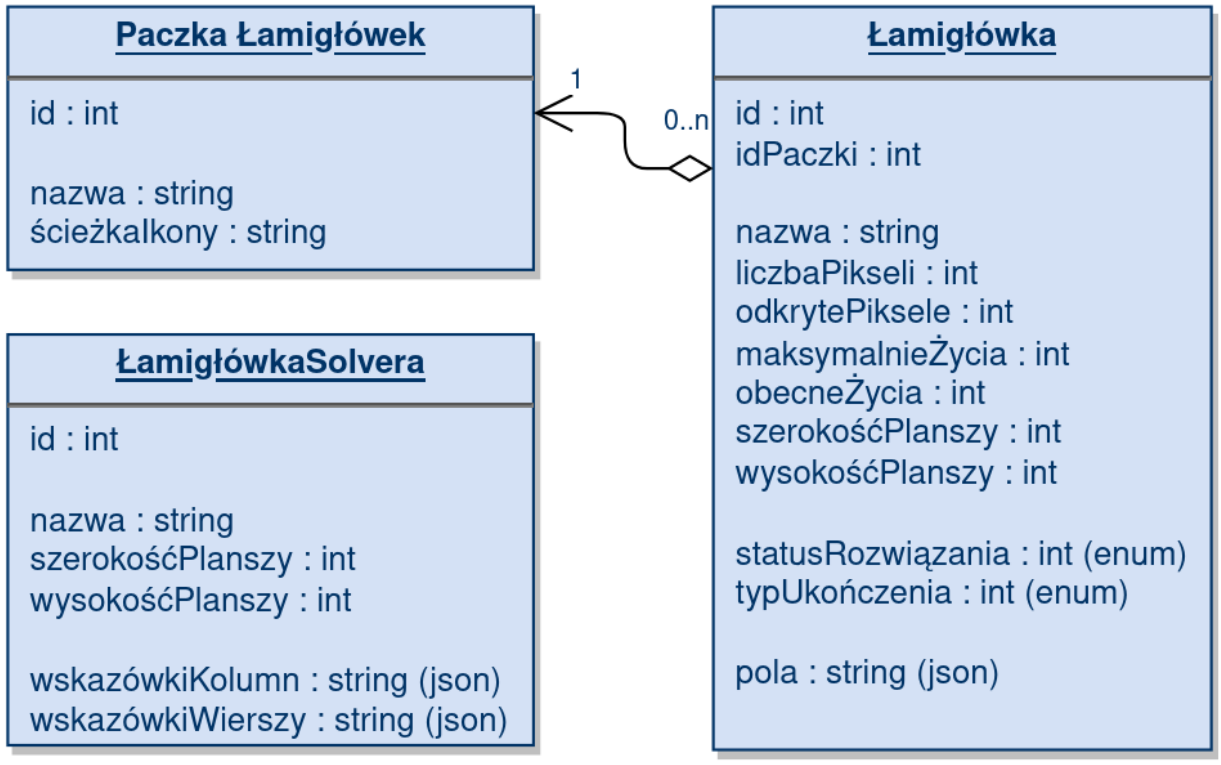
Do stworzenia aplikacji zostały wykorzystane biblioteki tworzone przez społeczność. Są to między innymi:

- React Native Elements [9] - biblioteka zawierająca podstawowe komponenty zgodne z Material Design
- React Native Navigation [10] - biblioteka obsługująca nawigację między aktywnościami
- React Native SQLite Storage [11] - biblioteka pozwalająca korzystać z lokalnej bazy SQLite
- React Native Zoomable View [12] - biblioteka dodająca komponent obsługujący przybliżanie ekranu

4.2 Baza danych

Dane aplikacji są przechowywane w bazie danych opartej na systemie SQLite. Schemat bazy jest przedstawiony na diagramie poniżej.

Baza składa się z 3 tabel: paczek, łamigłówek i łamigłówek solvera. Paczki zawierają jedynie niezbędne dane do reprezentacji grupy łamigłówek, czyli nazwa i ikona. W tabeli łamigłówek zawarte są proste dane takie jak nazwa, maksymalna ilość żyć (ilość błędów, po których się przegrywa). Przechowywane są tam również informacje o stanie gry, tj. `statusRozwiązania` - łamigłówka nierozpoczęta i nierozwiązana, łamigłówka rozpoczęta, łamigłówka rozwiązana - oraz `typUkończenia` - łamigłówka nieukończona bez przegranej,



Rysunek 4.1: Diagram bazy danych

łamigłówka nieukończona z przegraną, łamigłówka ukończona bez przegranej, łamigłówka ukończona z przegraną. Pod zmienną `pola` przechowywana jest lista pól wraz z ich stanami, pod postacią stringa w formacie JSON. Tabela łamigłówek solvera także zawiera łamigłówki, ale w innej formie - zapisane jedynie jako listy wskazówek w formacie JSON oraz dane do identyfikacji, np. nazwa czy wielkość planszy.

Solver nonogramów

W tym rozdziale opisany jest rozwój solvera. Opisane są kolejne główne wersje solverów, zbadany jest także wpływ zastosowanych heurystyk na wydajność w rozwiązywaniu wybranych łamigłówek.

5.1 Wersje solverów

5.1.1 Solver całościowy

Solver całościowy jest najprostszym z solverów implementowanych w toku pisania aplikacji. Jego implementacja opiera się na założeniu, że obrazek ukryty w łamigłówce jest ciągiem pustych i pełnych pikseli. Solver sprawdza wszystkie możliwe kombinacje pól, aż do wykrycia rozwiązania, bądź stwierdzenia jego braku. Wskazówki umieszczone obok planszy służą jedynie do walidacji rozwiązania, i nie są wykorzystywane w trakcie rozwiązywania nonogramu.

Solver ten zaczyna od pustej planszy. Następnie, dla pierwszego pola wywoływana jest rekursyjna metoda: jeśli indeks pola mieści się w zakresie planszy, to najpierw jego status ustawiany jest na pusty, i następuje wywołanie metody dla nastepnego indeksu, a jeśli rozwiązanie nie zostanie znalezione, to pole jest wypełniane i ponownie dochodzi do wywołania metody na następnym polu. Jeśli indeks wykracza poza zakres planszy, to znaczy że wszystkie pola mają ustawiony status i wywoływana jest metoda sprawdzająca poprawność rozwiązania, podobna do tej opisanej w 2.1. Jeśli solver zakończy działanie zwracając `true`, to w przekazanej mu macierzy pól (równoznaczne z listą wierszy) znajdzie się rozwiązanie łamigłówki.

Pseudokod 5.1: SolverCałościowy

Input: Lista wierszy R, indeks pola i, lista wskazówek wierszy Hr i kolumn Hc, szerokość w i wysokość h planszy
Output: Czy znaleziono rozwiązanie `true/false`

```
1  if i ⩾ w · h then
2      return Verify(R, Hr, Hc, w, h);
3  else
4      iWiersza ← ⌈i/w⌉;
5      iKolumny ← i mod w;
6      R[iWiersza][iKolumny] ← 0;
7      if SolverCałościowy(R, i + 1, Hr, Hc, w, h) then
8          return true;
9      R[iWiersza][iKolumny] ← 1;
10     return SolverCałościowy(R, i + 1, Hr, Hc, w, h);
```

Złożoność czasowa tego solvera jest bardzo duża. Procedura sprawdzająca poprawność rozwiązania może zostać wywołana 2^{w*h} razy, czyli inaczej 2^n, gdzie n to ilość pól na planszy. Wynika to z faktu, że kazde kolejne pole wymaga sprawdzenia pól poprzednich, a samo może znajdować się w dwóch stanach, więc podwaja ilość wywołań procedury sprawdzającej.

5.1.2 Solver osiowy

W odróżnieniu od solvera całościowego, solver osiowy korzysta ze wskazówek przy szukaniu rozwiązań. Opiera się on na fakcie, że każda linia (wiersz lub kolumna) może znajdować się w jednym z możliwych stanów,

których liczba nigdy nie dojdzie do 2^x, gdzie x jest długością linii. Sprawdzając rozwiązanie w tym solverze, gwarantowana jest poprawność w jednej z osi, co dodatkowo znacząco skraca czas szukania rozwiązania.

Solver zaczyna od pustej planszy. Przed rozpoczęciem rozwiązywania sprawdzana jest ilość wszystkich kombinacji w danej osi (iloczyn możliwości każdej z linii), i wybierana jest oś z mniejszą liczbą możliwości. Następnie generowane są kombinacje dla każdej z linii. Solver korzysta z rekursyjnej metody, i ustawia pierwszą kombinację dla pierwszej linii. Następnie wywołuje metodę dla kolejnej linii, aż do ostatniej, i wtedy weryfikuje rozwiązanie. Jeśli dla danego ustawienia w linii łamigłówka nie ma rozwiązania, to solver przechodzi do kolejnego ustawienia i wywołuje metodę w kolejnej linii.

Pseudokod 5.2: SolverOsiowy

```
Input: Lista linii L, indeks linii i, lista wskazówek linii prostopadłych H, ilość linii n
Output: Czy znaleziono rozwiązanie true/false
1 if i = n then
2     Verify(L, H, n);
3 else
4     line ← L[i];
5     foreach comb ∈ line.combinations do
6         ApplyComb(line, comb);
7         if SolverOsiowy(L, i + 1, H, n) then
8             return true;
9     return false;
```

Dzięki eliminacji kombinacji sprzecznych ze wskazówkami w danej osi, procedura sprawdzania poprawności rozwiązania jest wywoływana o wiele rzadziej niż w przypadku solvera całościowego. O ile ilość kombinacji w linii przy rozpatrywaniu każdej komórki z osobna to 2^n, gdzie n to długość linii, tak w przypadku rozważania poprawnych kombinacji dla linii jest ona zależna od długości i zawartości wskazówki, i można ją ograniczyć z góry przez $\binom{n+1-h}{h}$, a h to ilość wskazówek w linii. To przybliżenie jest zawyżone, ponieważ zakłada występowanie jedynie bloków długości jeden we wskazówce. W przeciętnym przypadku, bloki wypełnionych komórek będą dłuższe, oraz będzie ich mniej. Dodatkowo, jak zostało wspomniane na początku, weryfikacja jest wymagana jedynie w jednej z dwóch osi, jako że konstrukcja potencjalnych rozwiązań opiera się o zestawianie poprawnych kombinacji z linii.

Instalacja i wdrożenie

W tym rozdziale należy omówić zawartość pakietu instalacyjnego oraz założenia co do środowiska, w którym realizowany system będzie instalowany. Należy przedstawić procedurę instalacji i wdrożenia systemu. Czynności instalacyjne powinny być szczegółowo rozpisane na kroki. Procedura wdrożenia powinna obejmować konfigurację platformy sprzętowej, OS (np. konfiguracje niezbędnych sterowników) oraz konfigurację wdrażanego systemu, m.in. tworzenia niezbędnych kont użytkowników. Procedura instalacji powinna prowadzić od stanu, w którym nie są zainstalowane żadne składniki systemu, do stanu w którym system jest gotowy do pracy i oczekuje na akcje typowego użytkownika.

Podsumowanie

W podsumowanie należy określić stan zakończonych prac projektowych i implementacyjnych. Zaznaczyć, które z zakładanych funkcjonalności systemu udało się zrealizować. Omówić aspekty pielęgnacji systemu w środowisku wdrożeniowym. Wskazać dalsze możliwe kierunki rozwoju systemu, np. dodawanie nowych komponentów realizujących nowe funkcje.

W podsumowaniu należy podkreślić nowatorskie rozwiązania zastosowane w projekcie i implementacji (niebanalne algorytmy, nowe technologie, itp.).

Bibliografia

[1] L. L. Stephen Cook, "The complexity of theorem-proving procedures," 1971.

[2] J. N. van Rijn, "Playing games. the complexity of klondike, mahjong, nonograms and animal chess." https://liacs.leidenuniv.nl/assets/2012-01JanvanRijn.pdf, 2012.

[3] "React native." https://reactnative.dev/.

[4] "React.js." https://reactjs.org/.

[5] "Javascript oracle." https://developer.oracle.com/javascript/.

[6] "Javascript ecma." https://www.ecma-international.org/.

[7] "V8 javascript engine." https://v8.dev/.

[8] "Spidermonkey javascript engine." https://spidermonkey.dev/.

[9] "React native elements." https://reactnativeelements.com/.

[10] "React native navigation." https://reactnavigation.org/.

[11] "React native sqlite storage." https://github.com/andpor/react-native-sqlite-storage.

[12] "React native zoomable view." https://github.com/openspacelabs/react-native-zoomable-view.

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.