

# Dokumentacja projektu Kalkulator punktów Riichi Mahjong

Dariusz Piwowarski

Mateusz Łopaciński

Hieronim Koc

Styczeń 2026

## Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>2</b>
<b>2</b>	<b>Cel projektu</b>	<b>2</b>
<b>3</b>	<b>Technologie i działanie aplikacji</b>	<b>2</b>
3.1	Wykorzystane biblioteki . . . . .	2
3.2	Model uczenia maszynowego . . . . .	3
<b>4</b>	<b>Działanie aplikacji krok po kroku</b>	<b>3</b>
<b>5</b>	<b>Funkcjonalności</b>	<b>3</b>
5.1	Skanowanie i rozpoznawanie . . . . .	3
5.2	Kalkulacja punktów . . . . .	4
5.3	Ręczna edycja . . . . .	4
5.4	Historia gier . . . . .	4
5.5	Lista Yaku . . . . .	4
5.6	Ekran kalibracji (planowane) . . . . .	4
<b>6</b>	<b>Opis ekranów i widoków</b>	<b>4</b>
6.1	Ekran główny kalkulatora ( <i>Calculate Home</i> ) . . . . .	5
6.2	Ekran skanera ( <i>Scanner Screen</i> ) . . . . .	5
6.3	Ekran potwierdzenia ręki ( <i>Confirm Screen</i> ) . . . . .	5
6.4	Ekran kalkulatora / budowania ręki ( <i>Calculator Screen</i> ) . . . . .	5
6.5	Ekran wyniku ( <i>Result Screen</i> ) . . . . .	6
6.6	Ekrany historii ( <i>History List &amp; Detail</i> ) . . . . .	6
6.7	Ekrany listy yaku ( <i>Yaku List &amp; Detail</i> ) . . . . .	6
<b>7</b>	<b>Opis komponentów</b>	<b>7</b>
7.1	Scanner . . . . .	7
7.2	HandBuilder . . . . .	7
7.3	Calculator . . . . .	7
<b>8</b>	<b>Rozpoznawanie obrazu</b>	<b>7</b>
8.1	Model TensorFlow Lite . . . . .	7
8.2	Struktura detekcji . . . . .	8
8.3	Optymalizacja . . . . .	8

<b>9 Kalkulacja punktów</b>	<b>8</b>
9.1 Biblioteka riichi-ts . . . . .	8
9.2 Przykładowy wynik . . . . .	8
<b>10 Historia i przechowywanie danych</b>	<b>9</b>
10.1 AsyncStorage . . . . .	9
10.2 Hook useHistory . . . . .	9
<b>11 Testowanie</b>	<b>9</b>
11.1 Framework testowy . . . . .	9
11.2 Uruchomienie testów . . . . .	9
11.3 Sprawdzanie jakości kodu . . . . .	9
<b>12 Podsumowanie</b>	<b>10</b>
12.1 Główne osiągnięcia . . . . .	10
12.2 Możliwe rozszerzenia . . . . .	10

# 1 Wprowadzenie

Riichi Mahjong to japońska odmiana klasycznej gry Mahjong, w której gracze układają ręce z płytek (tiles) w celu osiągnięcia zwycięskich kombinacji. Podliczanie punktów po zakończeniu rozgrywki jest skomplikowane i wymaga znajomości wielu reguł, yaku (specjalnych kombinacji) oraz systemu punktacji opartego na Han i Fu.

Niniejsza aplikacja mobilna automatyzuje proces obliczania punktów poprzez wykorzystanie aparatu telefonu do rozpoznawania układu płytek na stole i automatycznego obliczania wyniku.

## 2 Cel projektu

Głównym celem projektu jest stworzenie mobilnej aplikacji wspomagającej podliczanie punktów w grze Riichi Mahjong poprzez:

- **Automatyzację procesu liczenia** — eliminacja potrzeby ręcznego liczenia punktów,
- **Wykorzystanie rozpoznawania obrazu** — analiza układu klocków na stole za pomocą kamery,
- **Dokładność obliczeń** — zapewnienie poprawności wyników zgodnie z oficjalnymi regułami,
- **Ułatwienie gry** — szybsze i wygodniejsze podliczanie wyników partii.

## 3 Technologie i działanie aplikacji

### 3.1 Wykorzystane biblioteki

- **React Native** — wieloplatformowy framework do budowy aplikacji mobilnych (Android, iOS),
- **react-navigation** — obsługa nawigacji pomiędzy ekranami (stack + dolne zakładki); definiuje przechodzenie między kalkulatorem, listą yaku i historią gier,
- **react-native-vision-camera** — dostęp do kamery i strumienia wideo; dostarcza surowe klatki do modułu rozpoznawania płytek,
- **react-native-fast-tf-lite** — integracja z modelem TensorFlow Lite; wykonuje inferencję YOLO na klatkach z kamery i zwraca detekcje płytek,
- **@shopify/react-native-skia** — renderowanie graficzne na żywo; służy do rysowania ramek wokół wykrytych płytek na podglądzie z kamery,
- **riichi-ts** — biblioteka odpowiedzialna za logikę liczenia punktów w Riichi Mahjong (wykrywanie yaku, liczenie Han/Fu, wynik końcowy),
- **@react-native-async-storage/async-storage** — lokalne przechowywanie danych; dzięki niej zapisywana jest historia rozegranych partii,
- **fuse.js** — wyszukiwanie z tolerancją błędów w liście yaku (np. filtrowanie po nazwie lub opisie),
- **react-native-reanimated** — biblioteka do tworzenia płynnych animacji i interakcji (korzysta z Worklets),
- **react-native-unistyles** — nowoczesny silnik stylów oparty na C++ zapewniający wysoką wydajność i obsługę motywów.

### 3.2 Model uczenia maszynowego

Aplikacja wykorzystuje model TensorFlow Lite (`model.tflite`) wytrenowany w podejściu YOLO do rozpoznawania płytek Mahjong.

- wejściem modelu są klatki z kamery przeskalowane do kwadratu (np. 640x640),
- wyjściem jest lista detekcji (klasa płytki, współrzędne, pewność),
- po stronie aplikacji wykonywany jest post-processing (m.in. Non-Maximum Suppression) oraz mapowanie detekcji na strukturę ręki używaną przez moduł liczący punkty.

## 4 Działanie aplikacji krok po kroku

Poniżej opisano typowy scenariusz użycia aplikacji od uruchomienia do zapisania wyniku.

1. **Ekran startowy / zakładka *Calculate*** — użytkownik widzi główny ekran kalkulatora z możliwością przejścia do skanera lub ręcznego budowania ręki.
2. **Skanowanie stołu** — użytkownik przechodzi do ekranu skanera, kieruje kamerę na stół; aplikacja w czasie rzeczywistym próbuje wykryć komplet ręki zwycięzcy oraz istotne elementy (meldy, Dora, itp.).
3. **Potwierdzenie ręki** — po udanym wykryciu użytkownik przechodzi do ekranu potwierdzenia, gdzie może poprawić rękę (np. zmienić pojedyncze płytki, ustawić wiatry, zaznaczyć Riichi).
4. **Obliczenie punktów** — po potwierdzeniu ręki aplikacja wywołuje moduł `riichi-ts`, który zwraca wynik (Han, Fu, punktacja, lista yaku).
5. **Prezentacja wyniku** — na ekranie wyniku użytkownik widzi czytelne podsumowanie: kto wygrał, jaką ręką, z jakich yaku składa się wynik oraz jak rozdzielono punkty między graczy.
6. **Zapis do historii** — wynik może zostać zapisany do lokalnej historii; dane trafiają do pamięci przy użyciu `AsyncStorage`.
7. **Przeglądanie historii i yaku** — z dolnych zakładek użytkownik może przejść do listy historii (przegląd poprzednich gier) lub do listy yaku (edukacyjny przegląd możliwych kombinacji wraz z filtrowaniem).

Rysunek 1: Przykładowy diagram działania aplikacji.

## 5 Funkcjonalności

### 5.1 Skanowanie i rozpoznawanie

- **Skanowanie stołu kamerą** — aplikacja wykorzystuje kamerę telefonu do analizy układu płytek,
- **Rozpoznawanie elementów ręki:**
  - układ płytek w ręce zwycięzcy (closed part),
  - wyłożone meldy (pon, chi, kan) — open part,
  - płytka wygrywająca (ron/tsumo),

- wskaźniki Dora i Uradora,
- wiatry (round wind, seat wind),
- deklaracje Riichi,
- specjalne zdarzenia (Ippatsu, Under the Sea, Blessing of Heaven).

## 5.2 Kalkulacja punktów

- **Automatyczne obliczanie** — aplikacja oblicza punkty na podstawie rozpoznanej ręki,
- **Szczegółowy opis wyniku** — wyświetlanie wszystkich czynników wpływających na wynik:
  - lista yaku (kombinacji specjalnych),
  - liczba Han i Fu,
  - końcowa wartość punktowa,
  - informacja o tym, kto wygrał i od kogo punkty zostały pobrane.

## 5.3 Ręczna edycja

- **Uzupełnianie danych** — możliwość ręcznego uzupełnienia informacji, które nie zostały poprawnie rozpoznane,
- **Korekta rozpoznania** — edycja wykrytych płytek i meldów,
- **Ustawienia dodatkowe** — konfiguracja wiatrów, typu wygranej, deklaracji Riichi.

## 5.4 Historia gier

- **Zapisywanie wyników** — automatyczne zapisywanie obliczonych wyników,
- **Przeglądanie historii** — lista wszystkich zapisanych gier,
- **Szczegóły ręki** — możliwość przejrzania szczegółów każdej zapisanej ręki.

## 5.5 Lista Yaku

- **Katalog yaku** — kompletna lista wszystkich dostępnych yaku w Riichi Mahjong,
- **Szczegóły yaku** — opis każdego yaku, jego wartość w Han, warunki spełnienia,
- **Filtrowanie** — możliwość filtrowania yaku według różnych kryteriów (rzadkość, kategoria).

## 5.6 Ekran kalibracji (planowane)

- **Dopasowanie do zestawu klocków** — możliwość kalibracji systemu rozpoznawania do konkretnego zestawu płytek,
- **Uczenie wyglądu** — aplikacja uczy się wyglądu poszczególnych płytek,
- **Rozpoznawanie znaczników** — kalibracja sticków Riichi i znaczników wiatrów.

# 6 Opis ekranów i widoków

Ta sekcja służy do szczegółowego opisanie kolejnych ekranów aplikacji oraz powiązanych z nimi funkcjonalności. Dla każdego ekranu przewidziano także miejsce na zrzut ekranu.

## 6.1 Ekran główny kalkulatora (*Calculate Home*)

### Opis:

- pierwszy ekran widoczny po uruchomieniu aplikacji (w zakładce *Calculate*),
- prezentuje skrót dostępnych opcji: skanowanie ręki kamerą lub ręczne budowanie ręki,
- może zawierać podsumowanie ostatnio policzonej ręki lub skróty do historii.

Miejsce na zrzut ekranu:

Rysunek 2: Ekran główny kalkulatora (do uzupełnienia).

## 6.2 Ekran skanera (*Scanner Screen*)

### Opis:

- wyświetla podgląd z kamery z nałożonymi ramkami na wykryte płytki,
- w tle działa model TensorFlow Lite wykonujący rozpoznawanie płytek i budujący wstępną reprezentację ręki,
- po wykryciu kompletnej ręki pozwala przejść dalej do etapu potwierdzania.

Miejsce na zrzut ekranu:

Rysunek 3: Ekran skanera z kamerą (do uzupełnienia).

## 6.3 Ekran potwierdzenia ręki (*Confirm Screen*)

### Opis:

- prezentuje rękę wykrytą przez model: płytki na ręce, meldy, Dora itp.,
- umożliwia ręczne poprawki (zmiana pojedynczych płytek, dodanie/usunięcie meldów),
- pozwala ustawić opcje wpływające na punktację: wiatry stołu i gracza, typ wygranej (Ron/T-sumo), deklaracje Riichi, Ippatsu itp.

Miejsce na zrzut ekranu:

Rysunek 4: Ekran potwierdzenia wykrytej ręki (do uzupełnienia).

## 6.4 Ekran kalkulatora / budowania ręki (*Calculator Screen*)

### Opis:

- alternatywa dla skanowania — ręczne budowanie ręki za pomocą selektora płytek,
- użytkownik wybiera płytki do części zamkniętej i otwartej (meldy) oraz ustawia parametry rozdania,
- po zbudowaniu poprawnej ręki może przejść do obliczenia punktów.

Miejsce na zrzut ekranu:

Rysunek 5: Ekran ręcznego budowania ręki (do uzupełnienia).

## 6.5 Ekran wyniku (*Result Screen*)

### Opis:

- wyświetla szczegółowy wynik obliczeń: Han, Fu, końcową liczbę punktów,
- prezentuje listę wykrytych yaku wraz z liczbą Han przypisaną każdemu z nich,
- pokazuje, jak punkty zostały rozdzielone pomiędzy graczy (kto od kogo płaci, różnica punktów),
- umożliwia zapisanie wyniku do historii.

### Miejsce na zrzut ekranu:

Rysunek 6: Ekran prezentujący wynik obliczeń (do uzupełnienia).

## 6.6 Ekran historii (*History List & Detail*)

### Lista historii:

- prezentuje listę zapisanych rozdań (np. z datą, liczbą punktów, skrótem yaku),
- umożliwia szybkie wyszukiwanie / filtrowanie (np. po dacie lub wyniku),
- po wybraniu pozycji otwierany jest ekran szczegółów.

### Szczegóły rozdania:

- pokazują pełną rękę, listę yaku, parametry rozdania oraz dokładny rozkład punktów,
- mogą zawierać dodatkowe dane kontekstowe (np. kto był dealerem, który to był stół/runda).

### Miejsce na zrzuty ekranu:

Rysunek 7: Lista zapisanych rozdań (do uzupełnienia).

Rysunek 8: Szczegóły pojedynczego rozdania (do uzupełnienia).

## 6.7 Ekran listy yaku (*Yaku List & Detail*)

### Lista yaku:

- zawiera wszystkie obsługiwane yaku wraz z krótkim opisem i liczbą Han,
- umożliwia filtrowanie i wyszukiwanie (np. wg kategorii, liczby Han, popularności).

### Szczegóły yaku:

- opisują dokładne warunki spełnienia danego yaku,
- mogą zawierać przykładowe układy płytek ilustrujące dane yaku.

### Miejsce na zrzuty ekranu:

Rysunek 9: Lista yaku z możliwością filtrowania (do uzupełnienia).

Rysunek 10: Ekran szczegółów pojedynczego yaku (do uzupełnienia).

## 7 Opis komponentów

### 7.1 Scanner

Komponent odpowiedzialny za skanowanie i rozpoznawanie płytek. Wykorzystuje:

- **react-native-vision-camera** — dostęp do kamery,
- **TensorFlow Lite** — model rozpoznawania,
- **Skia** — renderowanie detekcji na obrazie.

#### Główne funkcje:

- przetwarzanie klatek wideo w czasie rzeczywistym,
- wykrywanie płytek na obrazie,
- wizualizacja detekcji (prostokąty na wykrytych obiektach),
- automatyczne wykrywanie kompletnej ręki.

### 7.2 HandBuilder

Komponent do ręcznego budowania i edycji ręki. Pozwala na:

- dodawanie/usuwanie płytek,
- tworzenie meldów (pon, chi, kan),
- ustawianie opcji (wiatry, typ wygranej, Riichi),
- weryfikację poprawności ręki.

### 7.3 Calculator

Moduł obliczający punkty na podstawie ręki. Wykorzystuje bibliotekę **riichi-ts** do:

- walidacji ręki,
- identyfikacji yaku,
- obliczania Han i Fu,
- obliczania końcowej wartości punktowej.

## 8 Rozpoznawanie obrazu

### 8.1 Model TensorFlow Lite

Aplikacja wykorzystuje model YOLO w formacie TensorFlow Lite do rozpoznawania płytek Mahjong.

#### Proces przetwarzania:

1. **Przechwytywanie klatki** — kamera przechwytuje obraz w rozdzielczości 1920x1080,
2. **Przygotowanie danych** — obraz jest przeskalowywany do 640x640 i konwertowany do formatu float32,
3. **Inferencja** — model przetwarza obraz i zwraca detekcje,
4. **Post-processing** — zastosowanie Non-Maximum Suppression do eliminacji duplikatów,
5. **Interpretacja** — konwersja detekcji na strukturę ręki (**Hand**).

## 8.2 Struktura detekcji

Każda detekcja zawiera:

- pozycję (x, y),
- rozmiar (width, height),
- klasę (typ płytki),
- pewność (confidence score).

## 8.3 Optymalizacja

- **Worklets** — przetwarzanie odbywa się w wątku natywnym, nie blokując UI,
- **Core ML** — na iOS wykorzystywana jest delegacja Core ML dla lepszej wydajności,
- **Resize plugin** — efektywne przeskalowywanie obrazów.

# 9 Kalkulacja punktów

## 9.1 Biblioteka riichi-ts

Aplikacja wykorzystuje bibliotekę **riichi-ts** do obliczania punktów. Biblioteka implementuje oficjalne reguły Riichi Mahjong.

**Wejście:**

- zamknięta część ręki (closed part),
- otwarte meldy (open part),
- wiatry (round wind, seat wind),
- płytki wygrywająca (tile taken),
- opcje specjalne (Riichi, Ippatsu, Double Riichi, itd.).

**Wyjście:**

- liczba Han,
- liczba Fu,
- końcowa wartość punktowa (ten),
- lista wykrytych yaku,
- informacja o błędach (jeśli ręka jest niepoprawna).

## 9.2 Przykładowy wynik

```
{
  error: false,
  fu: 30,
  han: 6,
  isAgari: true,
  ten: 12000,
  yaku: {
```

```
    chinitzu: 5,  
    tanyao: 1  
  },  
  yakuman: 0  
}
```

## 10 Historia i przechowywanie danych

### 10.1 AsyncStorage

Aplikacja wykorzystuje `@react-native-async-storage/async-storage` do lokalnego przechowywania danych.

**Przechowywane dane:**

- historia obliczonych rąk,
- szczegóły każdej ręki (płytki, meldy, opcje, wynik).

### 10.2 Hook useHistory

Custom hook `useHistory` zarządza operacjami na historii:

- zapisywanie nowych wyników,
- pobieranie listy zapisanych gier,
- pobieranie szczegółów konkretnej gry,
- usuwanie zapisanych gier.

## 11 Testowanie

### 11.1 Framework testowy

Projekt wykorzystuje **Jest** jako framework testowy.

### 11.2 Uruchomienie testów

```
yarn test
```

### 11.3 Sprawdzanie jakości kodu

```
# Linting  
yarn lint  
  
# Sprawdzanie typów  
yarn typecheck  
  
# Sprawdzanie zależności cyklicznych  
yarn circular-dependency-check
```

## 12 Podsumowanie

Aplikacja **Riichi Mahjong Calculator** to kompleksowe rozwiązanie do automatyzacji obliczania punktów w grze Riichi Mahjong. Wykorzystuje zaawansowane technologie rozpoznawania obrazu (TensorFlow Lite) oraz sprawdzoną bibliotekę do obliczania punktów (**riichi-ts**), zapewniając graczom szybkie i dokładne podliczanie wyników.

### 12.1 Główne osiągnięcia

- Wieloplatformowa aplikacja mobilna (iOS, Android),
- Real-time rozpoznawanie płytek z kamery,
- Automatyczne obliczanie punktów zgodnie z oficjalnymi regułami,
- Intuicyjny interfejs użytkownika,
- Historia zapisanych gier,
- Kompletna lista yaku z opisami.

### 12.2 Możliwe rozszerzenia

- Ekran kalibracji dla różnych zestawów płytek,
- Eksport wyników do PDF,
- Tryb offline z pełną funkcjonalnością,
- Wsparcie dla różnych wariantów reguł,
- Statystyki i analityka gier.