Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **inżynieria systemów (INS)**

PRACA DYPLOMOWA INŻYNIERSKA

Narzędzie wspomagające trening układania kostki Rubika

Bartosz Piotr Krzysztoszek

Opiekun pracy
Dr Inż. Dariusz Gąsior

Słowa kluczowe: aplikacja, kostka Rubika, speedcubing, React.js

WROCŁAW (2024)

Streszczenie

Praca ma na celu opracowanie projektu oraz zaimplementowanie narzędzia wspomagającego trening układania kostki Rubika. Narzędzie jest przeznaczone głównie dla osób potrafiących już układać kostkę Rubika i chcących zwiększyć swoje umiejętności układania kostki metodą CFOP. Do wytworzenia narzędzia wykorzystano technologię React.js. Na początku zostało omówione wprowadzenie do dziedziny kostki Rubika oraz dyscypliny „speedcubing” w celu lepszego zrozumienia zamysłu narzędzia, a także zostały omówione istniejące rozwiązania dotyczące tej dziedziny. Projekt narzędzia został podzielony na wymagania, przypadki użycia, omówienie wykorzystanych technologii i architekturę. Po omówieniu projektu został umieszczony opis samego narzędzia wraz z opisem implementacji, a także kroki niezbędne do jego uruchomienia. W zakończeniu zostały wypisane wnioski oraz potencjalne kierunki dalszego rozwoju narzędzia.

Abstract

The aim of the work is to develop a project and implement a tool supporting Rubik's Cube solving training. The tool is intended mainly for people who already know how to solve a Rubik's Cube and want to improve their skills in solving the cube using the CFOP method. React.js technology was used to create the tool. At the beginning, an introduction to the Rubik's Cube and the "speedcubing" discipline was discussed in order to better understand the idea of the tool, and existing solutions in this field were also discussed. The design was divided into requirements, use cases, discussion of the technologies used and architecture. After discussing the design, a description of the tool itself was included along with a description of the implementation, as well as the steps necessary to run it. The conclusion includes conclusions and potential directions for further development of the tool.

Spis treści

1. Wstęp

Kostka Rubika to jedna z najbardziej ikonicznych łamigłówek świata, która od wielu lat fascynuje i inspiruje miliony ludzi na całym świecie. Niezwykłe połączenie kolorów oraz elementów geometrycznych tej układanki stwarza nietypowe jak na zabawkę zależności, w tym potencjał do ogromnej liczby kombinacji, co stanowi wyzwanie będące pasją dla wielu entuzjastów. To co wyjątkowe w kostce Rubika w połączeniu z naturalną u ludzi chęcią rywalizacji sprawiło, iż od 2003 roku regularnie na całym świecie odbywają się zawody w dyscyplinie zwanej „speedcubing". Obecnie coraz więcej osób, szczególnie młodych ale i nie tylko, zaczyna interesować się dziedziną speedcubingu. Jest to spowodowane rozwojem społeczności speedcuberów, która aktywnie dzieli się wiedzą i doświadczeniem poprzez różnorodne platformy internetowe. Youtube czy różne fora dyskusyjne pełne są poradników, analiz, a także relacji z zawodów, co sprzyja łatwiejszemu wejściu w świat speedcubingu dla nowych osób. Ponadto, organizacja międzynarodowych i lokalnych zawodów speedcubingowych dostarcza uczestnikom okazji do spotkania się, wymiany doświadczeń czy zawarcia nowych znajomości.

Z powodu coraz bardziej rosnącej popularności dyscypliny speedcubingu naturalną koleją rzeczy jest zwiększanie się zapotrzebowania na różnego rodzaju rozwiązania wspomagające. Tak jak każda sfera ludzkiego życia, speedcubing również ma swoje problemy wynikające ze specyfiki poszczególnych jego aspektów, z którymi muszą zmagać się osoby chcące rozwijać się w tej dziedzinie. Istotą speedcubingu jest bowiem stałe dążenie do lepszych wyników poprzez zarówno naukę efektywnych metod rozwiązywania kostki jak i zwiększanie szybkości w poruszaniu ściankami układanki poprzez praktykę. Z tego właśnie powodu, w niniejszej pracy zostanie podjęta próba stworzenia narzędzia wspomagającego proces treningu układania kostki Rubika na czas.

1.1. Cel i zakres pracy

Celem pracy jest stworzenie projektu oraz implementacja narzędzia w formie aplikacji webowej mającej wspomagać trening układania kostki Rubika na czas. Narzędzie będzie dotyczyło jedynie oryginalnej kostki Rubika bez uwzględniania innych układanek podobnych. Narzędzie będzie tworzone dla osób, które potrafią już ułożyć kostkę Rubika oraz chcą trenować w celu osiągania lepszych rezultatów czyli w celu skrócenia czasu układania kostki. W szczególności narzędzie jest tworzone głównie dla osób układających lub chcących układać kostkę Rubika metodą CFOP czyli najpopularniejszą metodą zaawansowaną.

W pracy zostanie zawarty opis problemu treningu układania kostki Rubika na czas wraz z istotnymi aspektami dziedziny aby umożliwić pełne zrozumienie celu pracy. Zostaną również przedstawione istniejące rozwiązania warz z ich analizą. W pracy znajdzie się opis procesu powstawania narzędzia zawierający informacje na temat jego projektu, zastosowanych metod rozwiązania problemu, implementacji oraz opis sposobu jego użytkowania. Projekt narzędzia zostanie podzielony na wymagania funkcjonalne oraz niefunkcjonalne, przypadki użycia, omówienie wykorzystanych technologii oraz opis architektury narzędzia.

2. Wprowadzenie do dziedziny

W tym rozdziale zostaną wyjaśnione zagadnienia oraz terminy związane z kostką Rubika, dyscypliną speedcubingu oraz oficjalnymi zawodami tej dyscypliny. Zostanie również szczegółowo omówiony problem wynikający ze specyfiki dziedziny, którego próba rozwiązania zostanie podjęta w dalszej części pracy. Należy zaznaczyć, że zostaną

przedstawione tylko zagadnienia, które będą istotne z poziomu opisywanego problemu, natomiast pozostałe szczegóły dotyczące kostki Rubika oraz oficjalnych zawodów zostaną pominięte.

2.1. Kostka Rubika

Kostka Rubika (ang. Rubik's cube) to trójwymiarowa układanka wynaleziona pierwotnie w 1974 roku.

2.1.1. Mechanika kostki Rubika

Kostka Rubika ma kształt sześcianu składającego się z 26 unikalnych miniaturowych sześcianów. Każdy z miniaturowych sześcianów zawiera ukryte przedłużenie znajdujące się wewnątrz kostki, które umożliwia połączenie się z innymi sześcianami oraz z rdzeniem, dzięki czemu sześciany mogą poruszać się jednocześnie. Środkowy miniaturowy sześcian każdej z sześciu ścian to jedynie pojedyncze kwadratowe pole przymocowane do rdzenia. Rdzeń zapewnia strukturę układanki, do której miniaturowe sześciany mogą się dopasować dzięki czemu mogą się one swobodnie poruszać bez możliwości odpadnięcia od reszty mechanizmu.

Miniaturowe sześciany można podzielić ze względu na liczbę ich zewnętrznych ścian. Są to:

- narożniki, czyli elementy o trzech zewnętrznych ścianach
- krawędzie, czyli elementy o dwóch zewnętrznych ścianach
- środkowe elementy, tak zwane centry, o jednej zewnętrznej ścianie

Każdy z miniaturowych sześcianów jest unikalny ze względu na kolory znajdujące się na jego ścianach. Wartą zauważenia właściwością kostki Rubika jest to, iż ze względu na przymocowanie centrów do rdzenia, centry pozostają względem siebie w tej samej pozycji, co dodatkowo sprawia, iż centry definiują jaki kolor powinien znajdować się na danej ścianie kostki Rubika

2.1.2. Schemat kolorów kostki Rubika

Każda z sześciu ścian kostki Rubika ma unikalny kolor, przy czym warto wspomnieć , że większość istniejących kostek zachowuje następujący schemat:

- Ściany kostki mają kolory biały, żółty, niebieski, zielony, czerwony oraz pomarańczowy,
- pary kolorów biały i żółty, niebieski i zielony, czerwony i pomarańczowy umieszczone są na przeciwnych ścianach kostki,
- trzymając kostkę ścianką z kolorem białym do góry oraz ścianką z kolorem zielonym do siebie, ścianka znajdująca się po prawej stronie kostki jest czerwona.

2.1.3. Układanie kostki Rubika

Wewnętrzny mechanizm kostki umożliwia niezależny obrót warstw sześcianów z każdej strony, mieszając w ten sposób kolory znajdujące się na poszczególnych ścianach. Stan kostki, w którym na każdej ścianie znajduje się tylko jeden kolor jest stanem docelowym, który oznacza rozwiązanie lub inaczej mówiąc ułożenie układanki. Trudność kostki Rubika polega na bardzo dużej liczbie możliwych do uzyskania kombinacji kolorów na ścianach przy tylko jednej kombinacji będącej rozwiązaniem. Liczba możliwych kombinacji jest określana na ponad 43 tryliony co oznacza, że losowe wykonywanie obrotów na kostce daje znikome prawdopodobieństwo na rozwiązanie jej. Z tego powodu do rozwiązywania pomieszanej kostki Rubika stosowane są metody o różnym podejściu oraz stopniu zaawansowania.

W kontekście metod układania kostki Rubika istotne jest pojęcie tak zwanego „algorytmu”. W żargonie osób układających kostki Rubika algorytmem określa się stałą sekwencję obrotów wykonywanych na kostce, które mają pożądany wpływ. Terminologia ta wywodzi się z matematycznego zastosowania algorytmu, czyli listy instrukcji wykonania zadania od danego stanu początkowego, poprzez dobrze określone kolejne stany, aż do pożądanego stanu końcowego. W tym momencie chciałbym zaznaczyć, że w dalszej części pracy do określenia sekwencji ruchów wykonywanych na kostce Rubika będzie stosowane określenie „algorytm” lub „sekwencja rozwiązująca”. Każda metoda układania kostki wykorzystuje własny zestaw algorytmów wraz z opisem, jaki efekt ma algorytm i kiedy można go zastosować, aby przybliżyć kostkę do rozwiązania. Szczególną cechą wielu algorytmów jest przeznaczenie do przekształcania tylko małej części kostki bez ingerencji w pozostałe części. Niektóre metody układania kostki, które polegają na stopniowym układaniu kolejnych części kostki mają również algorytmy, które wpływają na dużą liczbę elementów, jednak algorytmy te są stosowane w początkowych etapach układania, kiedy należy skupić się na ułożeniu mniejszej liczby elementów bez zwracania uwagi na położenie pozostałych elementów. Ważną kwestią dotyczącą algorytmów, o której należy wspomnieć jest to, iż może istnieć kilka algorytmów mających ten sam efekt. W praktyce algorytmy takie mogą różnić się pod względem liczby obrotów ale i wygody wykonywania co może przekładać się na szybkość ich wykonywania.

2.1.4. Notacja

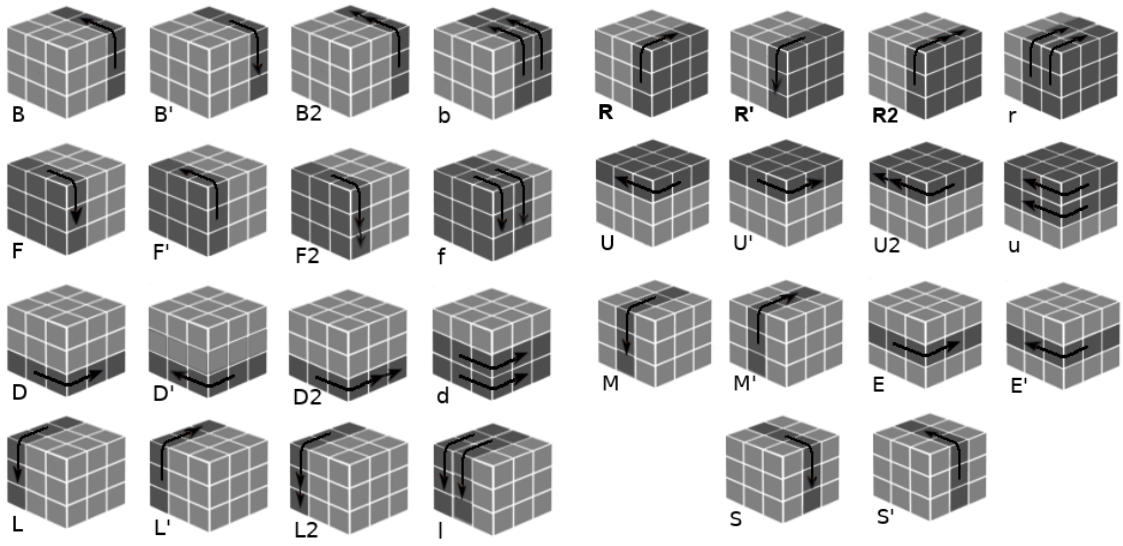
Najpopularniejszym sposobem opisu sekwencji obrotów kostki Rubika jest opracowana przez matematyka Davida Singmastera tak zwana „notacja Singmastera”. Notacja pozwala na zapis algorytmów w taki sposób, że możliwe jest ich stosowanie niezależnie od orientacji w jakiej trzymana jest kostka. Każdej ze ścian kostki przypisana jest duża litera:

- F – ściana przednia
- B – ściana tylna
- U – ściana górna
- D – ściana dolna
- L – ściana po lewej stronie
- R – ściana po prawej stronie

Litery użyte do oznaczeń ścian pochodzą od pierwszych liter angielskich słów oznaczających położenie (front, back, up, down, left, right). Zapis danej litery z notacji oznacza obrót jedną warstwą danej ściany o dziewięćdziesiąt stopni zgodnie z ruchem wskazówek zegara. Zapisanie symbolu apostrofu (′) po danej literze z notacji oznacza obrót jedną warstwą danej ściany o dziewięćdziesiąt stopni przeciwnie do ruchu wskazówek zegara. Zapisanie cyfry „2” po danej literze z notacji oznacza obrót jedną warstwą danej ściany o sto osiemdziesiąt stopni. Występują również obroty dwoma warstwami ścian zamiast jedną warstwą oznaczane małymi odpowiednikami liter notacji, dla których stosują się takie same reguły dotyczące kierunku obrotu. Dodatkowo w notacji występują obroty całą kostką wokół jednej z jej osi, czyli zmiany orientacji w jakiej trzymana jest kostka lub ujmując inaczej, obroty trzema warstwami naraz. Są one oznaczane małymi literami „x”, „y” oraz „z” i odpowiadają odpowiednio obrotom ściany prawej (R), obrotom warstwą ściany górnej (U) oraz obrotom warstwą ściany przedniej (F). Dla obrotów całą kostką stosuje się takie same reguły dotyczące kierunku obrotu co do obrotów jedną warstwą [1].

Istnieje również rozszerzenie notacji Singmastera, które zostało ogólnie przyjęte przez osoby układające kostki Rubika. Polega ono na dodaniu opisów dla obrotów środkowymi warstwami kostki. Każdej warstwie środkowej przypisana jest duża litera:

- M – warstwa pomiędzy warstwami ściany prawej (R) i lewej (L)
- E – warstwa pomiędzy warstwami ściany górnej (U) i dolnej (D)



Rysunek 2.1 Graficzne przedstawienie notacji kostki Rubika (źródło: [2])

- S – warstwa pomiędzy warstwami ściany przedniej (F) i tylnej (B)

Litery użyte do oznaczeń warstw pochodzą od pierwszych liter angielskich słów „middle", „equator" oraz „standing". Zapis litery „M" oznacza obrót warstwą o dziewięćdziesiąt stopni z góry na dół. Zapis litery „E" oznacza obrót warstwą o dziewięćdziesiąt stopni w prawo. Zapis litery „S" oznacza obrót warstwą o dziewięćdziesiąt stopni zgodnie z ruchem ściany przedniej (F). Zapisanie symbolu apostrofu (′) po danej literze z notacji oznacza obrót warstwą o dziewięćdziesiąt stopni w przeciwnym kierunku. Zapisanie cyfry „2" po danej literze z notacji oznacza obrót warstwą o sto osiemdziesiąt stopni [2].

Notacja Singmastera oraz jej rozszerzenie są zaprezentowane w formie graficznej na rysunku 2.1.

2.1.5. Metody układania kostki Rubika

Najpopularniejszą metodą układania kostki Rubika dla początkujących jest metoda „LBL". W skrócie polega ona na ułożeniu każdej warstwy kostki po kolei zaczynając od dolnej warstwy i kończąc na górnej warstwie. Metoda ta wymaga nauczenia się zaledwie kilku prostych algorytmów oraz dobrze opanowana pozwala zrozumieć ogólne podstawowe aspekty układania kostki. Pomimo że metoda nie jest efektywna pod względem liczby wymaganych do ułożenia kostki obrotów, pozwala ona na stopniowe wdrażanie większej liczby algorytmów co przekłada się na skrócenie liczby wymaganych obrotów [3].

Najpopularniejszym rozwinięciem metody LBL jest metoda zaawansowana o nazwie „CFOP" (można się również spotkać z nazwą „metoda Fridrich"). Nazwa metody bierze się z czterech etapów wymaganych do ułożenia kostki. Pierwszym etapem jest ułożenie krzyża dolnej warstwy. Drugi etap, tak zwany „F2L" (First Two Layers), polega na ułożeniu dolnej i środkowej warstwy układając w tym samym momencie narożnik dolnej warstwy i krawędź należącą do środkowej warstwy. Trzeci etap, tak zwany „OLL" (Orientation of the Last Layer) polega na zorientowaniu górnej warstwy za pomocą jednego algorytmu, tak aby górna ściana kostki była pokryta tylko jednym kolorem poprzez odpowiednie obrócenie wszystkich miniaturowych sześcianów. Czwarty etap, tak zwany „PLL" (Permutation of the Last Layer) polega na permutowaniu górnej warstwy za pomocą jednego algorytmu, tak aby górna warstwa została w całości ułożona. Metoda ta jest bardziej efektywna od metody podstawowej, jednakże ze względu na liczbę możliwych sytuacji, które mogą wystąpić w kolejnych etapach układania, wymaga nauczenia się ponad stu algorytmów. Dobra

znajomość wszystkich wymaganych algorytmów oraz umiejętność szybkiego stosowania ich podczas układania pozwala na uzyskiwanie czasów ułożenia kostki nawet poniżej dziesięciu sekund [4].

Istotnym aspektem metod układania kostki Rubika, o którym warto wspomnieć jest orientacja kostki Rubika. Kostka Rubika może być bowiem trzymana w jednej z 24 orientacji, które można określać za pomocą koloru znajdującego się na górnej ścianie ułożonej kostki Rubika oraz koloru znajdującego się na przedniej ścianie kostki Rubika. Znając kolory górnej i przedniej ściany kostki Rubika jesteśmy w stanie określić jej orientację. Podczas rozpoczynania ułożenia kostki Rubika pierwszym krokiem jest zdecydowanie w jakiej orientacji lub grupie orientacji będziemy układać kostkę Rubika. Dla przykładu, używając metody LBL albo CFOP należy zdecydować, na której ścianie zostanie ułożony początkowy krzyż. W celu przyzwyczajenia się do kostki Rubika, poradniki dla osób początkujących zazwyczaj tłumaczą aby rozpoczynać ułożenia od białego krzyża co określa, że kostka Rubika będzie układana w grupie czterech orientacji, których kolor górnej ściany kostki Rubika to żółty. Jednakże możliwe jest ułożenie kostki Rubika za pomocą każdej metody w dowolnej orientacji, co sprawia, że wybór orientacji może zależeć od preferencji danej osoby.

2.2. Zawody

„Speedcubing” to dyscyplina polegającą na jak najszybszym układaniu kostki Rubika i innych układanek logicznych z jej rodziny. Istotą tej dziedziny jest to, że aby układać układanki w najkrótszym czasie trzeba umieć szybko reagować na losowo natrafiane sytuacje. Każde pomieszanie układanki jest bowiem unikalne i liczba możliwych kombinacji jest na tyle duża, że nie jest możliwym nauczenie się każdej kombinacji na pamięć.

Organizowaniem oficjalnych zawodów w tej dyscyplinie zajmuje się organizacja międzynarodowa WCA. Oficjalne zawody przebiegają według zasad oraz regulacji ustalanych przez WCA.

2.2.1. Przebieg zawodów

Na każde oficjalne zawody są określane układanki w jakich odbywa się rywalizacja, przy czym obecnie mogą być to tylko układanki wybrane z siedemnastu oficjalnych konkurencji. Warto wspomnieć, że standardowa kostka Rubika nie jest obowiązkową układanką, w której odbywa się rywalizacja na każdych zawodach, ale jest ona zdecydowanie najczęściej pojawiającą się układanką na zawodach. Zawody są podzielone na rundy. W każdej rundzie odbywa się rywalizacja w danej układance, przy czym rywalizacja w jednej układance może obejmować więcej niż jedną rundę.

Przebieg każdej rundy rozpoczyna się od zaniesienia swoich układanek przez zawodników do miejsca mieszania. Następnie zawodnicy udają się do poczekalni. Układanki są mieszane według ustalonych sekwencji mieszających, przy czym każdy zawodnik otrzymuje swoją układankę pomieszaną w ten sam sposób, jednakże każda próba rundy ma swoją sekwencje mieszającą. Pomieszane układanki są zabierane na bieżąco przez wyznaczonych do tego sędziów. Sędzia z układanką wywołuje odpowiedniego zawodnika z poczekalni oraz udaje się z nim do stanowiska startowego składającego się ze stołu na którym rozłożona jest mata wraz z profesjonalnym miernikiem czasu określanym jako „timer”. Następuje procedura układania układanki, po której zawodnik wraca do poczekalni i oczekuje na kolejne wywołanie, a sędzia zabiera układankę z powrotem do miejsca mieszania. Proces powtarza się do momentu, kiedy zawodnik wykona wszystkie próby.

Zajęte przez zawodnika miejsce w danej rundzie na oficjalnych zawodach dla większości układanek jest ustalane na podstawie średniego czasu trzech środkowych ułożeń kostki z pięciu prób.

2.2.2. Procedura układania

Cała procedura układania jest nadzorowana przez sędziego, tak aby uniemożliwić potencjalne oszustwa. Układanka zakryta pudełkiem ustawiana jest przed zawodnikiem na macie przez sędziego. W momencie kiedy zawodnik zgłosi gotowość, sędzia odkrywa układankę. Rozpoczyna się tak zwana „inspekcja", w której zawodnik ma piętnaście sekund na obejrzenie pomieszanej układanki przed rozpoczęciem układania. Czas inspekcji jest sprawdzany przez sędziego za pomocą osobnego stopera. Dodatkowo sędzia informuje zawodnika w momencie upływu ośmiu oraz dwunastu sekund czasu inspekcji. Zawodnik powinien rozpocząć układanie poprzez uruchomienie miernika czasu przed upływem czasu przeznaczonego na inspekcję. W przypadku, gdy zawodnik przekroczy limit piętnastu sekund inspekcji, zostaje nałożona kara w postaci dwóch sekund dodanych do końcowego wyniku ułożenia. Jeżeli zawodnik po przekroczeniu limitu piętnastu sekund inspekcji nie rozpocznie ułożenia przez kolejne dwie sekundy, ułożenie zostaje uznane za nieważne co jest określane poprzez skrót „DNF", który rozwija się do angielskiej frazy „did not finished". Zawodnik, który rozpoczął ułożenie powinien rozwiązać układankę, a następnie zatrzymać miernik czasu. Po ułożeniu sędzia zapisuje czas z miernika czasu. Jeżeli miernik czasu został zatrzymany, ale układanka nie została rozwiązana poprawnie, ułożenie zostaje uznane za nieważne co ponownie jest określane poprzez skrót „DNF". Jeżeli miernik czasu został zatrzymany, a układanka znajduje się w stanie, w którym brakuje wykonania jednego ruchu aby została ona rozwiązana, zostaje nałożona kara w postaci dwóch sekund dodanych do końcowego wyniku. Warto wspomnieć, że kary dwóch sekund są kumulowane.

2.2.3. Sekwencje mieszające

Sekwencje mieszające to sekwencje ruchów wykonywanych na układance w celu jej pomieszania zapisane za pomocą odpowiedniej dla danej układanki notacji. Dla standardowej kostki Rubika sekwencje mieszające wykorzystują notację Singmastera. Sekwencje mieszające są stosowane na zawodach w celu mieszania układanki w sposób obiektywny i całkowicie losowy oraz w celu uzyskania tego samego pomieszania układanki dla każdego zawodnika, tak by wyniki zawodników były porównywane sprawiedliwie. Według organizacji WCA aby układanka była pomieszana sprawiedliwie, powinna ona reprezentować jeden z wszystkich możliwych do uzyskania na układance stanów z wyłączeniem stanów, które wymagają liczby posunięć do ułożenia mniejszej niż dwa ruchy. Należy zaznaczyć, że aby pomieszane za pomocą sekwencji mieszającej układanki były identyczne, sekwencje mieszającą rozpoczyna się trzymając układankę w określonej orientacji. Dla standardowej kostki Rubika jest to orientacja trzymając białą ścianę skierowaną do góry oraz zieloną ścianę skierowaną do siebie.

Sekwencje mieszające są generowane losowo przed zawodami za pomocą oprogramowania komputerowego przez osobę odpowiedzialną za poprawny przebieg oficjalnych zawodów, czyli przez delegata WCA. Delegat WCA odpowiada za bezpieczne przechowanie sekwencji mieszających do momentu rozpoczęcia danej rundy zawodów, tak aby żaden z zawodników nie miał możliwości wcześniejszego zapoznania się ze sposobem pomieszania układanki.

2.3. Opis problemu

W celu przygotowania się na zawody zawodnicy muszą trenować układanie kostki Rubika poza zawodami. Tworzy to potrzebę narzędzia umożliwiającego mierzenie czasu układania kostki Rubika. Profesjonalne mierniki czasu używane na zawodach są powszechnie dostępne do zakupu, jednakże ich ceny są często całkiem wysokie. Urządzenia te są bardzo dokładne w mierzeniu czasu, aczkolwiek jest to ich jedyna funkcja. Ze względu na koszt oraz ograniczoną funkcjonalność rodzi się potrzeba alternatywnego rozwiązania.

Istotnym aspektem treningu w każdej dziedzinie są uzyskiwane wyniki. W celu zwiększania efektywności treningu niezbędne jest podejmowanie odpowiednich decyzji o jego dalszym przebiegu. Aby decyzje były odpowiednie konieczne jest branie pod uwagę uzyskiwanych do tej pory wyników. Uzyskiwane wyniki bowiem obrazują to jak przebiegał dotychczasowy trening i stanowią jego ocenę. Tak samo w dziedzinie speedcubingu jest ważnym to aby dostosowywać trening do zawodnika odpowiednio, a informacja o tym jak to zrobić znajduje się w jego wynikach. Rodzi to potrzebę analizy uzyskiwanych przez zawodników czasów układania kostki Rubika.

Tak jak zostało to opisane w poprzednim podrozdziale, kostki Rubika są mieszane na zawodach za pomocą sekwencji mieszających. Powstaje jednak tutaj problem sprawiedliwego mieszania układanki do celów treningu. Mieszanie układanki samodzielnie przez zawodnika może prowadzić do pewnych tendencji lub powtarzających się schematów uzyskiwanych na kostce Rubika. Tego typu podejścia nie można uznać za obiektywne, a wyniki zawodnika nie byłby porównywalne względem wyników innych zawodników.

Wcześniej opisana metoda CFOP jest obecnie zdecydowanie najpopularniejszą metodą układania kostki Rubika wśród zawodników zawodów speedcubingu. Dwa ostatnie etapy metody CFOP, czyli OLL oraz PLL obejmują łącznie 78 przypadków, z czego każdy przypadek to inna kombinacja kolorów na ostatniej górnej warstwie. Nauczenie się algorytmów OLL oraz PLL pozwala na znaczne skrócenie czasów układania kostki Rubika, jednakże uczenie się ich nie jest proste. Każdy algorytm to sekwencja kilkunastu ruchów co sprawia, że informacji do przyswojenia jest dużo. Ważnym aspektem jest również to, iż nie wystarczy zapamiętać algorytmu zapisanego za pomocą notacji. Aby móc wykorzystać pełny potencjał i szybkość metody CFOP niezbędne jest wykorzystanie pamięci mięśniowej podczas nauki algorytmów. Oznacza to, że osoba jest nauczona danego algorytmu dopiero wtedy kiedy jest w stanie wykonać go automatycznie lub spontanicznie bez zastanawiania się nad nim, co wymaga pewnej ilości praktyki w stosowaniu algorytmu. Dodatkowo ważne jest też poprawne rozpoznawanie dla którego przypadku należy użyć którego algorytmu.

Biorąc pod uwagę wyżej rozpisane problemy można uświadomić sobie jak złożonym procesem może być trening układania kostki Rubika na czas. Rodzi się potrzeba narzędzia wspomagającego trening układania kostki Rubika, które umożliwiałoby generowanie sekwencji mieszających, mierzenie i zapisywanie rezultatów, podsumowywanie i analizę treningu, ale i również narzędzia skupionego na wspomaganiu nauki i ćwiczenia algorytmów.

3. Przegląd istniejących rozwiązań

W tym rozdziale zostały przedstawione najpopularniejsze rodzaje istniejących rozwiązań w formie narzędzi informatycznych dotyczących wspomagania ogólnie pojętego treningu układania kostki Rubika oraz w szczególności treningu metody CFOP. Zostały przedstawione zarówno typowe cechy tych narzędzi wynikające ze specyfiki oraz zwyczajów dyscypliny jak i cechy najpopularniejszych przykładów narzędzi, dzięki którym się wyróżniają.

Obecnie w Internecie można znaleźć dużą liczbę aplikacji służących do wspomagania treningu układania kostki Rubika w różny sposób. Są to aplikacje w formie zarówno aplikacji desktopowych, stron internetowych jak i aplikacji mobilnych. Aplikacje te w zróżnicowany sposób podchodzą do samego procesu trenowania układania kostki Rubika oraz oferują różnego rodzaju funkcje.

3.1. Aplikacje typu „timer”

Najbardziej powszechnym typem narzędzia wspomagającego trening układania kostki Rubika są aplikacje typu „timer”. Powszechnym założeniem tego typu aplikacji jest odwzorowanie funkcjonalności oraz doświadczeń jakie oferują fizyczne mierniki przeznaczone do mierzenia czasu układania układanek, w szczególności mierniki czasu używane na zawodach. Do głównych funkcjonalności aplikacji typu „timer” należą:

- uruchamianie i zatrzymywanie mierzenia za pomocą klawiszy klawiatury lub ekranu dotykowego smartphone’a,
- zapisywanie czasów ułożeń układanki,
- przedstawianie prostych danych statystycznych w celu podsumowania uzyskiwanych wyników,

Dodatkowo często pojawiającą się funkcją jest możliwość edycji i usuwania uzyskiwanych wyników.

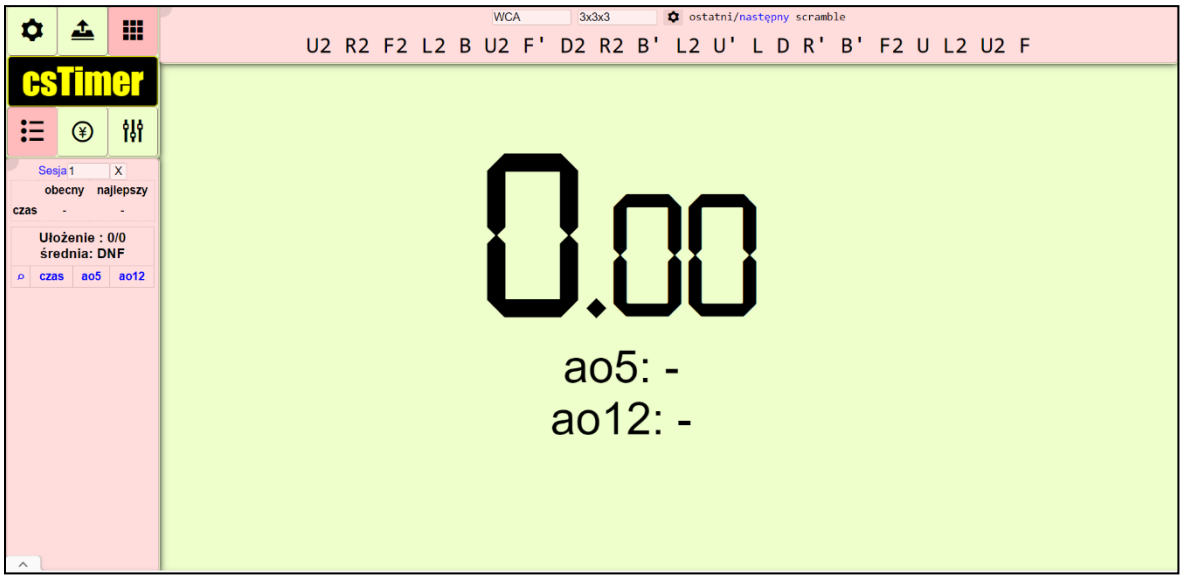
W wielu aplikacjach typu „timer” znajdują się również funkcjonalności mające na celu odwzorowanie procedury układania układanek na oficjalnych zawodach. Chodzi tutaj o funkcjonalności takie jak:

- wyświetlanie czasu inspekcji wraz z komunikatami o upływie ośmiu i dwunastu sekund,
- kary dwóch sekund dodanych do wyniku w przypadku przedłużenia czasu inspekcji lub w przypadku brakującego jednego ruchu do ułożenia układanki,
- oznaczenie ułożenia jako nieważne poprzez skrót „DNF”,
- generowanie i wyświetlanie sekwencji mieszających.

Mimo że wiele aplikacji typu „timer” skupia się głównie na wspomaganiu treningu zwykłej kostki Rubika to jednak wiele aplikacji jest również przeznaczona do treningu układania innych układanek, w szczególności układanek, w których odbywa się rywalizacja na oficjalnych zawodach, jednakże istnieją również aplikacje obsługujące szersze pule różnych układanek. Działa to w ten sposób, iż użytkownik wybiera, którą układankę chce trenować, a aplikacja generuje odpowiednią dla wybranej układanki sekwencję mieszającą oraz zapisuje uzyskiwane wyniki oddzielnie.

3.1.1. Aplikacja „csTimer”

Najpopularniejszą aplikacją typu „timer” jest „csTimer” (rys. 3.1). Mimo, że aplikacja jest w formie aplikacji webowej, możliwe jest używanie jej również na urządzeniach mobilnych jako aplikacji natywnej. Aplikacja zawiera wszystkie wymienione wcześniej główne funkcjonalności aplikacji typu „timer” dotyczące zarówno mierzenia oraz zapisywania czasu z wyświetlaniem szeregu informacji statystycznych na temat uzyskiwanych wyników jak i funkcjonalności dotyczące procesu układania układanek na oficjalnych zawodach. Aplikacja obsługuje generowanie sekwencji mieszających dla kostki Rubika, dla układanek, w których odbywa się rywalizacja na oficjalnych zawodach, a także dla wielu innych mniej znanych oraz nietypowych układanek. Istotnym jest fakt tego, że aplikacja daje możliwość dostosowywania każdego aspektu jej działania. Oglądając interfejs graficzny aplikacji po raz pierwszy można odnieść wrażenie, że nie spełnia on dzisiejszych standardów o czym świadczy chociażby domyślny dobór kolorystyki, jednakże pomimo, że interfejs jest dość surowy, aplikacja daje użytkownikowi szereg opcji dostosowywania interfejsu takich jak wybieranie kolorów poszczególnych komponentów i dostosowywanie ich rozmiarów czy możliwość wybrania, które sekcje mają być wyświetlane w danym momencie. Dodatkowo aplikacja posiada dużą liczbę wbudowanych dodatkowych narzędzi wspomagających trening pod różnymi względami, takich jak wyświetlanie docelowego stanu układanki po pomieszaniu, możliwość połączenia z własnym fizycznym urządzeniem



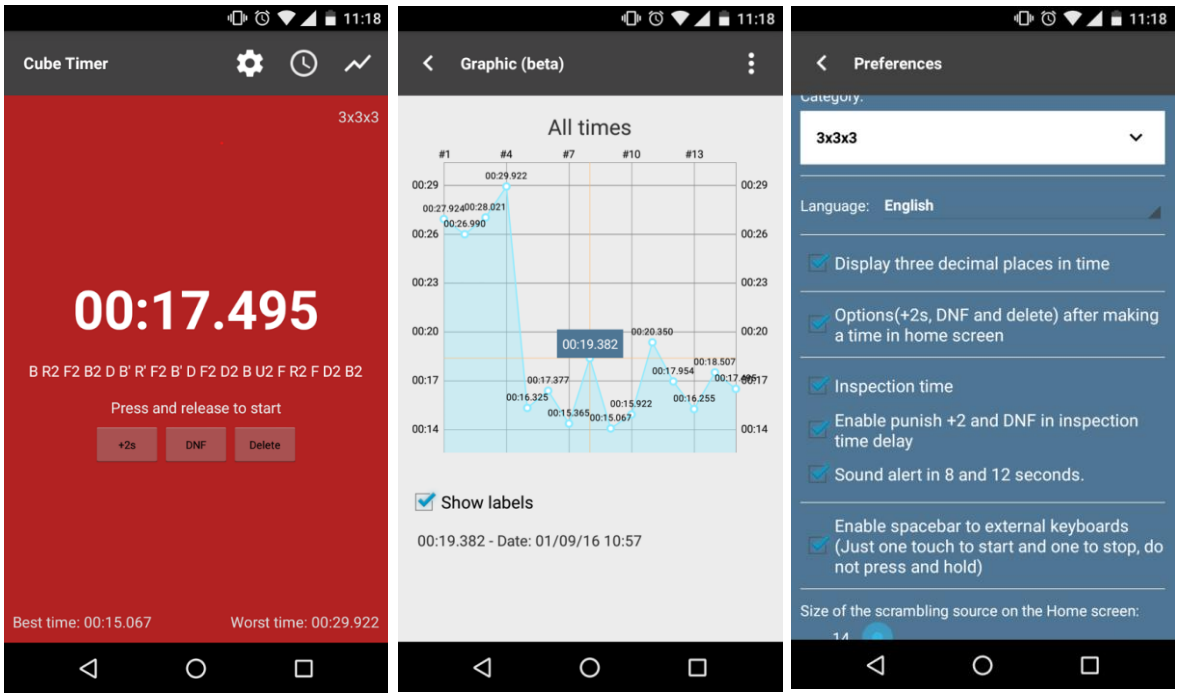
Rysunek 3.1 Aplikacja „csTimer” (źródło: [5])

mierzącym czas, możliwość połączenia z elektroniczną kostką bluetooth, metronom i wiele więcej. Można śmiało stwierdzić, że aplikacja jest prosta ale jednocześnie pozwala na wysokie spersonalizowanie oraz dostarcza wiele dodatkowych funkcjonalności pomagających w treningu układania.

Warto zwrócić dodatkową uwagę na rozmieszczenie elementów interfejsu na ekranie. Na samym środku znajduje się duże miejsce na uzyskiwany czas. Pozostałe elementy jak przyciski ustawień, sekwencja mieszająca czy lista wyników znajdują się dookoła centralnego obszaru. Tego typu układ pozytywnie wpływa na przejrzystość interfejsu oraz ułatwia szybkie i wygodne odczytanie rezultatu, a jednocześnie użytkownik ma łatwy dostęp do wszystkich dodatkowych opcji w każdym momencie. Dodatkowo, podczas mierzenia czasu ułożenia, wszystkie elementy poza wyświetlanym czasem są chowane. Jest to pomocne w minimalizowaniu zakłóceń i dodatkowym skupianiu uwagi na ułożeniu. Ogólnie rzecz biorąc, rozmieszczenie elementów w interfejsie zostało zaplanowane tak, aby dostarczyć intuicyjne i przejrzyste środowisko do pomiaru czasu, co pozytywnie wpływa na doświadczenie użytkowania z aplikacji [5] [6].

3.1.2. Aplikacja „Cube Timer”

Inną, jedną z popularnych aplikacji mobilnych typu „timer” jest aplikacja „Cube Timer” (rys. 3.2). Aplikacja jest dostępna do pobrania w sklepie Google Play. Tak samo jak poprzednia omawiana aplikacja, zawiera ona wszystkie główne funkcjonalności aplikacji typu „timer”. Aplikacja jednak nie jest tak rozbudowana i nie zawiera dodatkowych narzędzi. Działanie aplikacji ogranicza się do mierzenia oraz zapisywania czasu, generowania sekwencji mieszających tylko dla układanek w których odbywa się rywalizacja na oficjalnych zawodach oraz wyświetlania prostych statystyk wraz z wykresem przedstawiającym uzyskane kolejno wyniki. Aplikacja dodatkowo oferuje panel opcji, gdzie użytkownik może dokonać niewielkiej personalizacji. Wyróżniającym się elementem aplikacji jest panel, na którym wyświetlone są dane pobierane ze strony organizacji WCA na temat zbliżających się oficjalnych zawodów. Panel ten jednak ogranicza się tylko do wyświetlania informacji. Można stwierdzić, że aplikacja jest prosta oraz nie daje wielu możliwości użytkownikowi, natomiast dobrze zapewnia podstawowe funkcje [7].



Rysunek 3.2 Aplikacja „Cube Timer” (źródło: [7])

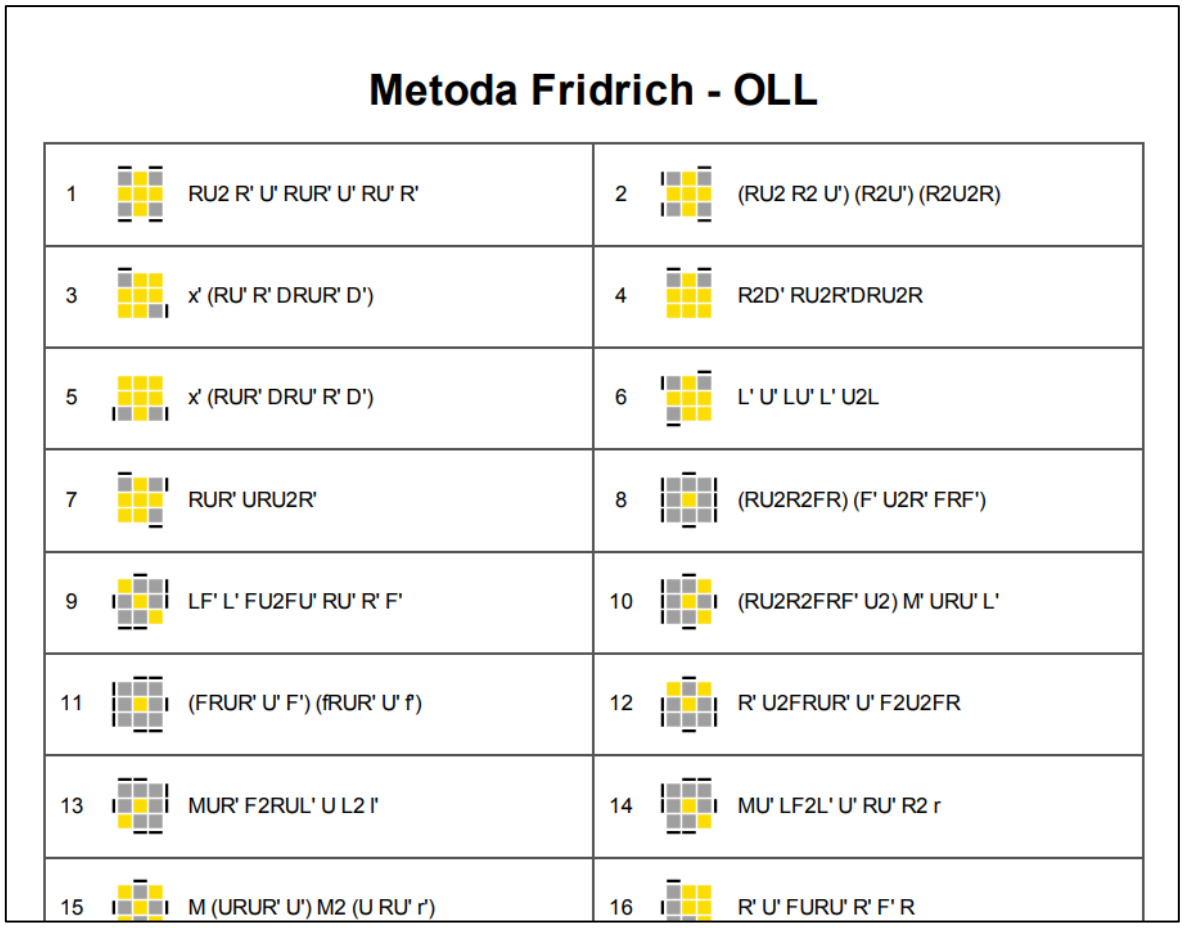
3.2. Rozwiązania wspomagające trening metodą CFOP

Próbując znaleźć rozwiązanie wspomagające naukę lub trening metody CFOP w Internecie, bardzo łatwo natrafić w pierwszej kolejności na strony internetowe zawierające listy algorytmów do poszczególnych etapów tej metody (rys. 3.3). Takie listy zazwyczaj ograniczone są jedynie do wyświetlania przedstawionej sytuacji na kostce Rubika, nazwy danego przypadku oraz samego algorytmu czyli sekwencji ruchów. Niektóre z takich stron służą także jako poradnik poszczególnych etapów metody CFOP. Rozwiązanie takie jest jednak interaktywne w bardzo niewielkim stopniu [8]. Dodatkowo można też natrafić na analogiczne rozwiązania w formie aplikacji mobilnych.

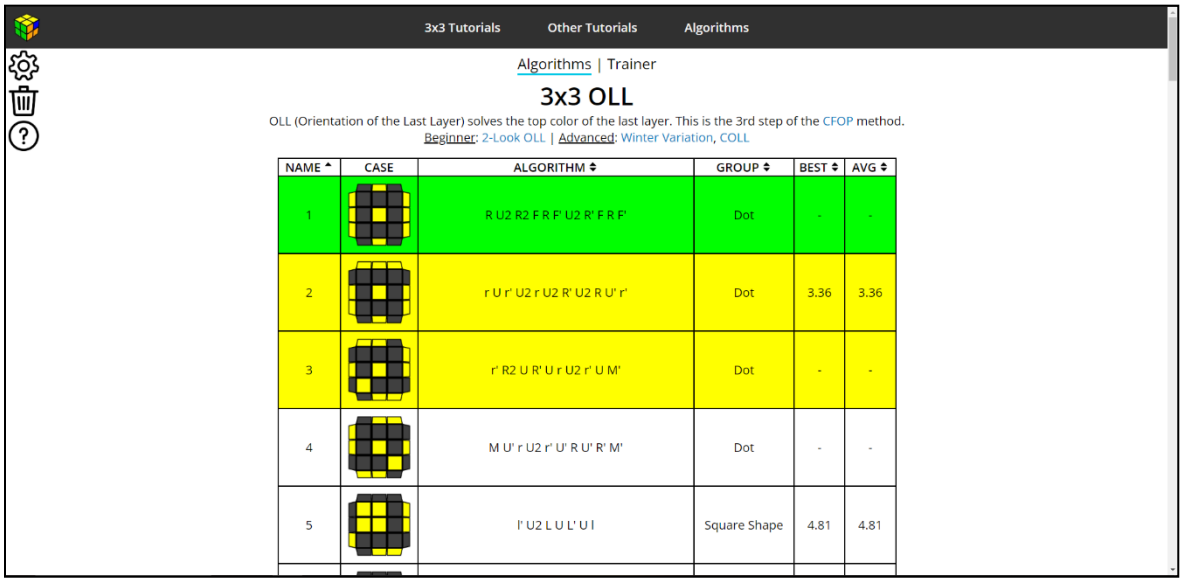
3.2.1. Strona „jperm.net”

Bardziej interaktywne podejście oferuje strona „jperm.net”. Na stronie można znaleźć zarówno poradniki do różnych metod układania układanek, w tym poradnik do metody CFOP, jednak bardziej istotnym elementem są przedstawione tutaj listy algorytmów (rys. 3.4). Listy algorytmów znajdujące się na stronie dotyczą etapów metody CFOP ale również innych metod oraz układanek. Listy te pozwalają na kilka szczególnych interakcji. Użytkownik może oznaczać wybrane algorytmy jako „nieznane”, „obecnie uczone” lub „nauczone” poprzez kliknięcie i zmianę koloru wiersza listy z danym przypadkiem, co pozwala na śledzenie postępów w ich nauce. Użytkownik może również decydować jakiej sekwencji ruchów używać dla danego przypadku poprzez wybór sekwencji ruchów z proponowanych sekwencji lub wprowadzenie własnej sekwencji, co umożliwia personalizacje listy i możliwość pełnej modyfikacji, zamiast przedstawiania „gotowej” listy algorytmów bez możliwości zmian.

Oprócz samej listy algorytmów jest również zakładka „Trainer” (rys. 3.5), w której użytkownik może ćwiczyć wybrane przypadki na takiej samej zasadzie jak odbywa się to w aplikacjach typu „timer”. Narzędzie losuje jeden z przypadków, który został oznaczony jako „obecnie uczony”, generuje i wyświetla sekwencję mieszającą mającą doprowadzić kostkę Rubika do stanu z wylosowaną sytuacją. Użytkownik uruchamia i zatrzymuje mierzenie

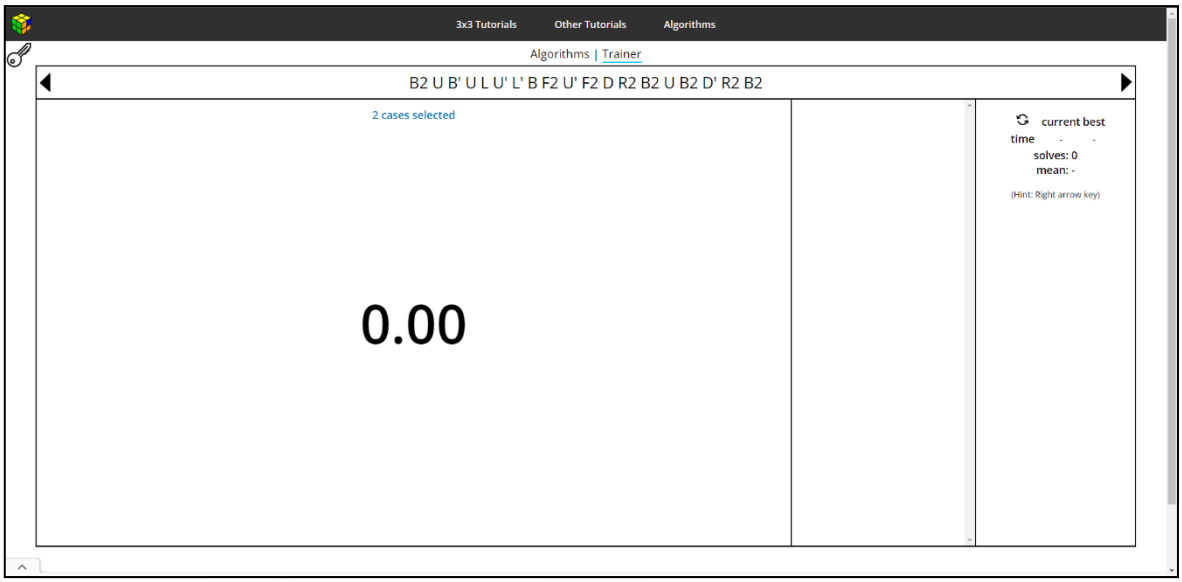


Rysunek 3.3 Lista algorytmów OLL (źródło: [8])



Rysunek 3.4 Strona jperm.net, lista algorytmów (źródło: [9])

czasu za pomocą klawiszy klawiatury w celu zmierzenia czasu wykonywania algorytmu dla wylosowanej sytuacji, oraz wyświetlane są dane statystyczne w celu podsumowania dotychczasowego treningu. W ten sposób użytkownik może trenować wykonywanie wybranych algorytmów pod względem szybkości ich wykonywania, a dostarczane użytkownikowi dane statystyczne stanowią podsumowanie uzyskiwanych wyników



Rysunek 3.5 Strona jperm.net, zakładka "Trainer" (źródło: [9])

treningu, dzięki któremu użytkownik może bardziej efektywnie dostosować dalszy trening [9].

3.2.2. Aplikacja „csTimer”

Inne interaktywne podejścia do uczenia się i trenowania metody CFOP można znaleźć również we wspomnianej już aplikacji „csTimer”. W tej aplikacji możliwe jest generowanie sekwencji mieszających, które mają doprowadzić kostkę Rubika do jednej z możliwych sytuacji, które mogą pojawić się w poszczególnych etapach metody CFOP, jednakże w tej aplikacji nie ma żadnej możliwości wyboru, które sytuacje użytkownik chce w danym momencie trenować. Poza tym ta aplikacja nie zawiera funkcjonalności dotyczących listy algorytmów [5].

3.3. Spostrzeżenia

Analizując przegląd istniejących rozwiązań można dojść do kilku wniosków. Przede wszystkim, do tej pory powstało bardzo niewiele narzędzi, które są przeznaczone do wspomagania nauki metody CFOP, a żadne z nich nie obejmuje jednocześnie wspomagania klasycznego treningu układania kostki Rubika.

Innym aspektem jest to, iż każda lista algorytmów metody CFOP, którą można znaleźć w Internecie wykorzystuje wizualizacje sytuacji na kostce Rubika dla algorytmów w celu pokazania tego, kiedy należy stosować dany algorytm. Istotny jest fakt, że owe wizualizacje znajdują się w formie grafik co sprawia, że nie ma możliwości na dostosowanie orientacji wizualizowanych kostek Rubika, a wyświetlane są jedynie stany kostki Rubika tylko w jednej wybranej wcześniej orientacji. Umożliwienie zmiany orientacji kostek Rubika będących wizualizacjami przypadków mogłoby pozytywnie wpłynąć na efektywność poznawania i uczenia się kolejnych algorytmów przez użytkowników. Każdy użytkownik mógłby bowiem wybrać orientację w jakiej aktualnie znajduje się jego kostka w celu łatwego znalezienia wymaganego algorytmu do rozwiązania natrafionej sytuacji. Dodatkowo każdy użytkownik mógłby wybrać orientację pasującą do jego preferowanego koloru.

Kolejnym wnioskiem, który należy wziąć pod uwagę jest charakterystyczne rozmieszczenie elementów interfejsu na ekranie. Ze względu na specyfikę dziedziny speedcubingu, której istotą jest osiąganie jak najkrótszych czasów, interfejsy

rozpatrywanych narzędzi poświęcają dużo przestrzeni środkowej części ekranu właśnie na liczony czas, podczas gdy reszta narzędzi znajduje się na obrzeżach ekrany dookoła czasu.

4. Projekt narzędzia

W tym rozdziale zostanie przedstawiony projekt narzędzia tak, by możliwe było sprecyzowanie struktury jego działania. Dzięki temu, proces implementacji i rozwoju narzędzia będzie mógł przebiegać w konkretny sposób, a żaden istotny z poziomu planu narzędzia wcześniej określony aspekt nie zostanie pominięty. Zostaną sprecyzowane zarówno wymagania funkcjonalne jak i wymagania niefunkcjonalne w celu określenia najważniejszych elementów narzędzia oraz najważniejszych aspektów [10]. Zostaną przedstawione przypadki użycia narzędzia, tak aby tworzenie go mogło odbywać się w kierunku zapewnienia praktycznej użyteczności [11]. Określone zostaną technologie niezbędne do implementacji zarówno poszczególnych elementów narzędzia jak i jego całości, a także technologie, które będą wykorzystane jedynie w procesie implementacji jako rozwiązania wspierające sam proces realizacji. Zostanie również przedstawiona architektura narzędzia.

4.1. Wymagania funkcjonalne

Poniżej znajdują się wymagania funkcjonalne. Każdemu wymaganiu został nadany priorytet określony liczbą w skali od 1 do 3, gdzie liczba 3 oznacza najwyższy priorytet.

1. Mierzenie czasu rozwiązywania kostki Rubika przez użytkownika z dokładnością do 10 milisekund – 3
2. Wyświetlanie na bieżąco czasu rozwiązywania kostki Rubika przez użytkownika – 2
3. Wyświetlenie uzyskanego czasu rozwiązywania kostki Rubika przez użytkownika – 3
4. Zapisywanie danych dotyczących każdego rezultatu rozwiązywania kostki Rubika przez użytkownika – 3
5. Możliwość edycji oraz usuwania danych dotyczących każdego rezultatu przez użytkownika – 3
6. Obsługiwanie mierzenia czasu za pomocą klawiszy klawiatury przez użytkownika – 3
7. Możliwość zatwierdzenia lub odrzucenia rezultatu przez użytkownika po wykonaniu ułożenia – 2
8. Generowanie sekwencji mieszającej dla kostki Rubika – 3
9. Generowanie sekwencji mieszającej dla losowo wybranego „algorytmu” - 3
10. Wyświetlanie sekwencji mieszającej – 3
11. Wyświetlanie prostych danych statystycznych dotyczących uzyskiwanych rezultatów użytkownika – 3
12. Możliwość dodawania kar dwóch sekund dla rezultatów przez użytkownika – 2
13. Możliwość oznaczenia rezultatu jako „dnf” – 2
14. Możliwość włączenia czasu piętnastu sekund „inspekcji” przez użytkownika – 3
15. Wyświetlanie ostrzeżeń podczas „inspekcji” dla ośmiu oraz dwunastu sekund – 2
16. Sygnał dźwiękowy podczas „inspekcji” dla ośmiu oraz dwunastu sekund – 2
17. Wyświetlanie listy „algorytmów” metody „CFOP” dla etapów „OLL” i „PLL” wraz z wizualizacją stanu kostki Rubika – 3
18. Możliwość dostosowania orientacji kostki Rubika dla jakiej są wizualizowane stany dla każdego „algorytmu” – 1
19. Możliwość zmiany i zapisywanie stanu każdego „algorytmu” dotyczącego nauki przez użytkownika – 2
20. Możliwość wyboru trenowanych „algorytmów” – 3
21. Możliwość dostosowania wyświetlanych informacji dla użytkownika – 1

22. Sekcja pomocy wyjaśniająca podstawy korzystania z narzędzia – 1
23. Zapisywanie wybranych przez użytkownika ustawień narzędzia - 1

4.2. Wymagania niefunkcjonalne

Poniżej znajdują się wymagania niefunkcjonalne określone ogólnie wraz z opisem wyjaśniającym wymaganie w odniesieniu do narzędzia.

1. Użyteczność

Interfejs graficzny narzędzia powinien być intuicyjny i pozwalający na łatwe nawigowanie oraz korzystanie. Podczas korzystania z narzędzia powinno być jasne, które elementy tylko wyświetlają informacje, a które pozwalają na interakcję. Funkcje narzędzia powinny być nietrudne w zrozumieniu dla osób zaznajomionych z dziedziną układania kostki Rubika na czas, a zapoznawanie się z nimi nie powinno wymagać większego wysiłku. Nieoczywiste interakcje, takie jak korzystanie z klawiszy klawiatury podczas korzystania z narzędzia powinny być dodatkowo opisane w narzędziu, tak by użytkownik mógł łatwo się o nich dowiedzieć i przyswoić ich używanie.

2. Dostępność

Narzędzie powinno być możliwe do wykorzystania poprzez wszystkie najpopularniejsze obecnie przeglądarki internetowe.

3. Niezawodność

Narzędzie powinno być odporne na błędy.

4. Wydajność

Narzędzie powinno być wydajne. Wszystkie operacje odbywające się w narzędziu powinny natychmiastowo zwracać wynik, a użytkownik nie powinien oczekiwać w żadnym momencie korzystania z narzędzia po jego uruchomieniu.

5. Łatwość utrzymania

Struktura narzędzia powinna pozwalać na łatwe dodawanie nowych funkcji i rozwijanie istniejących. Kod narzędzia powinien być czytelny oraz podzielony na odpowiednie części.

4.3. Przypadki użycia

Poniżej zostały przedstawione przypadki użycia narzędzia przez użytkownika w celu określenia możliwości wykorzystania narzędzia. Dla każdego przypadku została podana jego nazwa, warunek wstępny, scenariusz oraz warunek końcowy.

1. Wykonanie ułożenia kostki Rubika

Aktor: Użytkownik
Warunki wstępne: Użytkownik ma uruchomione narzędzie oraz ma ze sobą ułożoną kostkę Rubika. Narzędzie wyświetla ekran stopera w zakładce „3x3”.
Scenariusz główny:

1. Użytkownik miesza kostkę Rubika stosując sekwencję mieszającą wyświetloną na ekranie narzędzia.
2. Użytkownik przytrzymuje klawisz spacji na klawiaturze.
3. Czas stopera zmienia kolor na zielony oraz pozostałe elementy interfejsu zostają ukryte.
4. Użytkownik puszcza klawisz spacji na klawiaturze i rozpoczyna układanie kostki Rubika.
5. Mierzenie czasu zostaje uruchomione.
6. Użytkownik kończy układanie kostki Rubika po czym naciska klawisz spacji na klawiaturze.
7. Mierzenie czasu zostaje zatrzymane oraz zostają wyświetlone przyciski odpowiadające za określenie poprawności ułożenia.

8. Użytkownik ponownie naciska klawisz spacji na klawiaturze lub wybiera opcję „OK”.
9. Rezultat zostaje zapisany do bazy danych jako ułożenie poprawne.
10. Przyciski odpowiadające za określenie poprawności ułożenia zostają ukryte, a pozostałe elementy interfejsu ponownie się pojawiają

Alternatywny przebieg:

3.A.1. Czas stopera zmienia kolor na żółty oraz pozostałe elementy interfejsu zostają ukryte.

3.A.2. Użytkownik puszcza klawisz spacji na klawiaturze.

3.A.3. Czas stopera zmienia kolor na czerwony, pozostałe elementy interfejsu zostają ukryte oraz rozpoczyna się odliczanie 15 sekund czasu inspekcji.

3.A.4. Użytkownik przytrzymuje klawisz spacji na klawiaturze.

3.A.5. Czas stopera zmienia kolor na zielony.

3.A.6. Użytkownik puszcza klawisz spacji na klawiaturze i rozpoczyna układanie kostki Rubika.

3.A.4.A.1 Użytkownik naciska klawisz „esc” na klawiaturze.

3.A.4.A.2 Elementy interfejsu ponownie się pojawiają oraz inspekcja zostaje anulowana.

6.A.1. Użytkownik naciska klawisz „esc” na klawiaturze.

6.A.2. Rezultat zostaje zapisany do bazy danych jako ułożenie nieukończone.

6.A.3. Elementy interfejsu ponownie się pojawiają.

8.A.1. Użytkownik wybiera opcję „+2”.

8.A.2. Rezultat zostaje zapisany do bazy danych jako ułożenie z karą dwóch sekund.

8.A.3. Przyciski odpowiadające za określenie poprawności ułożenia zostają ukryte, a pozostałe elementy interfejsu ponownie się pojawiają

8.B.1. Użytkownik naciska klawisz „esc” na klawiaturze lub wybiera opcję „DNF”.

8.B.2. Rezultat zostaje zapisany do bazy danych jako ułożenie nieukończone.

8.B.3. Przyciski odpowiadające za określenie poprawności ułożenia zostają ukryte, a pozostałe elementy interfejsu ponownie się pojawiają

Warunek końcowy: Rezultat jest zapisany i jest widoczny na liście wyników.

2. Wykonanie treningu algorytmów

Aktor: Użytkownik

Warunki wstępne: Użytkownik ma uruchomione narzędzie oraz ma ze sobą ułożoną kostkę Rubika. Narzędzie wyświetla ekran stopera w zakładce „PLL” lub „OLL”.

Scenariusz główny:

1. Użytkownik miesza kostkę Rubika stosując sekwencję mieszającą wyświetloną na ekranie narzędzia.
2. Użytkownik przytrzymuje klawisz spacji na klawiaturze.
3. Czas stopera zmienia kolor na zielony oraz pozostałe elementy interfejsu zostają ukryte.
4. Użytkownik puszcza klawisz spacji na klawiaturze i rozpoczyna wykonywanie algorytmu.
5. Mierzenie czasu zostaje uruchomione.
6. Użytkownik kończy wykonywanie algorytmu na kostce Rubika po czym naciska klawisz spacji na klawiaturze.
7. Mierzenie czasu zostaje zatrzymane, a rezultat zostaje zapisany do bazy danych.
8. Pozostałe elementy interfejsu ponownie się pojawiają

Alternatywny przebieg:

6.A.1. Użytkownik naciska klawisz „esc” na klawiaturze.

6.A.2. Mierzenie czasu zostaje anulowane.

6.A.3. Elementy interfejsu ponownie się pojawiają.

Warunek końcowy: Rezultat jest zapisany i jest widoczny na liście wyników.

3. Usunięcie wszystkich wyników

Aktor: Użytkownik

Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran stopera w jednej z zakładek. Opcja wyświetlania okna wyników jest włączona.

Scenariusz główny:

1. Użytkownik wybiera opcję „Delete all results” w oknie wyników.
2. Zostaje wyświetlone okienko z komunikatem z prośbą o potwierdzenie.
3. Użytkownik wybiera opcję „delete all results”.
4. Okienko znika, a wszystkie wyniki dla danej zakładki stopera zostają usunięte z bazy danych.

Alternatywny przebieg:

3.A.1. Użytkownik wybiera opcję „close” dla okienka.

3.A.2. Okienko znika.

Warunek końcowy: Wszystkie wyniki dla danej zakładki stopera są usunięte i nie są widoczne na liście wyników.

4. Usunięcie jednego wyniku

Aktor: Użytkownik

Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran stopera w jednej z zakładek. Opcja wyświetlania okna wyników jest włączona.

Scenariusz główny:

1. Użytkownik wybiera jeden z wyników w oknie wyników.
2. Zostaje wyświetlone okienko z informacjami na temat wyniku.
3. Użytkownik wybiera opcję „delete result”.
4. Okienko znika, a wybrany wynik zostaje usunięty z bazy danych.

Alternatywny przebieg:

3.A.1. Użytkownik wybiera opcję „close” dla okienka.

3.A.2. Okienko znika.

Warunek końcowy: Wybrany wynik jest usunięty i nie jest widoczny na liście wyników.

5. Zmiana szczegółów wyniku

Aktor: Użytkownik

Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran stopera w jednej z zakładek. Opcja wyświetlania okna wyników jest włączona.

Scenariusz główny:

1. Użytkownik wybiera jeden z wyników w oknie wyników.
2. Zostaje wyświetlone okienko z informacjami na temat wyniku.
3. Użytkownik zaznacza lub odznacza jedną z opcji „+2 (inspection)”, „+2 (one turn)” lub „DNF”.
4. Zmiana dotycząca wyniku zostaje zapisana w bazie danych.
5. Użytkownik wybiera opcję „close” dla okienka.
6. Okienko znika.

Alternatywny przebieg:

3.A.1. Użytkownik wybiera opcję „close” dla okienka bez dokonywania zmiany.

3.A.2. Okienko znika.

Warunek końcowy: Zmiana dotycząca wyniku jest zapisana oraz widoczna na liście wyników.

6. Wyświetlenie informacji na temat wyniku

Aktor: Użytkownik

Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran stopera w jednej z zakładek. Opcja wyświetlania okna wyników jest włączona.
Scenariusz główny:

1. Użytkownik wybiera jeden z wyników w oknie wyników.
2. Zostaje wyświetlone okienko z informacjami na temat wyniku.

Warunek końcowy: Okienko z informacjami na temat wyniku jest wyświetlone.

7. Zmiana stanu uczenia się algorytmu

Aktor: Użytkownik
Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran algorytmów w jednej z zakładek „PLL" lub „OLL".
Scenariusz główny:

1. Użytkownik wybiera jeden z algorytmów na liście algorytmów.
2. Użytkownik klika w wizualizację kostki Rubika dla wybranego algorytmu.
3. Stan uczenia się algorytmu zostaje zmieniony na kolejny stan zaawansowania oznaczany przez kolor tła algorytmu, czyli z „nieznane", na „obecnie uczone", z „obecnie uczone" na „nauczone" oraz z „nauczone" z powrotem na „nieznane".
4. Zmiana stanu uczenia się algorytmu zostaje zapisana w bazie danych.

Warunek końcowy: Zmiana dotycząca stanu uczenia się algorytmu jest zapisana oraz widoczna na liście algorytmów.

8. Zmiana ustawień

Aktor: Użytkownik
Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran stopera lub algorytmów.
Scenariusz główny:

1. Użytkownik wybiera opcję „settings".
2. Zostaje otworzone okienko z listą dostępnych ustawień.
3. Użytkownik zaznacza lub odznacza wybrane ustawienia.
4. Zmiana dotycząca ustawień zostaje zapisana w bazie danych.
5. Użytkownik wybiera opcję „close" dla okienka.
6. Okienko znika, a wybrany wynik zostaje usunięty z bazy danych.

Alternatywny przebieg:

3.A.1. Użytkownik wybiera opcję „close" dla okienka bez dokonywania zmian.
3.A.2. Okienko znika.

Warunek końcowy: Zmiana dotycząca ustawień jest zapisana oraz zastosowana.

9. Wyświetlenie sekcji pomocy

Aktor: Użytkownik
Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran stopera lub algorytmów.
Scenariusz główny:

1. Użytkownik wybiera opcję „help".
2. Zostaje wyświetlone okienko z sekcją pomocy.

Warunek końcowy: Okienko z sekcją pomocy jest wyświetlone.

10. Zmiana orientacji wyświetlanych kostkek Rubika na liście algorytmów

Aktor: Użytkownik
Warunki wstępne: Użytkownik ma uruchomione narzędzie. Narzędzie wyświetla ekran algorytmów.
Scenariusz główny:

1. Użytkownik wybiera jeden z sześciu kolorowych przycisków pod sekcją „top color".

2. Orientacja wyświetlanych kostek Rubika na liście algorytmów zostaje dostosowana do wybranego koloru górnej ściany, a kolorowe przyciski pod sekcją „front color” zostają ograniczone do czterech dostępnych.
3. Użytkownik wybiera jeden z czterech kolorowych przycisków pod sekcją „front color”.
4. Orientacja wyświetlanych kostek na liście algorytmów zostaje dostosowana do wybranego koloru górnej i przedniej ściany.

Warunek końcowy: Orientacja wyświetlanych kostek Rubika na liście algorytmów jest dostosowana do wybranych kolorów górnej i przedniej ściany.

4.4. Wybór technologii

4.4.1. Główne technologie

W trakcie procesu tworzenia projektu narzędzia kluczowym jest wybór głównych technologii, na których oparte będzie narzędzie. Docelowo narzędzie ma funkcjonować jako aplikacja webowa tak by możliwe było uruchomienie jej w szybki sposób jedynie poprzez przejście pod odpowiedni adres url strony. Biorąc pod uwagę dotychczasowe założenia na temat narzędzia, sensownym wydaje się być wybranie technologii służącej do tworzenia tak zwanych „Single Page Applications” czyli aplikacji jednostronnicowych, opartych tylko o jeden plik HTML i wymagających tylko jednego załadowania podczas uruchamiania aplikacji. Z tych powodów zdecydowano w pracy na wybranie technologii React.js.

React.js jest darmową biblioteką języka Java Script rozwijaną głównie przez firmę Meta. Biblioteka ta służy do budowania dynamicznych i interaktywnych interfejsów, w tym między innymi właśnie do budowy jednostronicowych aplikacji. React.js jest obecnie uważany za najpopularniejszy framework front-ednowy wśród deweloperów co zawdzięcza swojej wydajności, modularności oraz deklaratywnej składni. Istotnym aspektem tego frameworku jest koncepcja komponentów. Komponent jest odizolowanym elementem interfejsu, w którym mogą być umieszczane inne komponenty co pozwala na łatwe zarządzanie strukturą interfejsu. Każdy komponent może być wykorzystywany wielokrotnie, a przepływ danych między komponentami odbywa się tylko od komponentów nadrzędnych do podrzędnych co ułatwia kontrolowanie przepływu danych i ich śledzenie. Komponenty można podzielić na komponenty stanowe i bezstanowe. Komponenty bezstanowe służą do renderowania interfejsu użytkownika na podstawie przekazanych im informacji, natomiast komponenty stanowe posiadają stan i są odpowiedzialne za zarządzanie nim. Każdy komponent jest definiowany w kodzie w postaci funkcji języka Java Script, która przyjmuje pojedynczy argument „props” oraz zwraca kod komponentu, zazwyczaj w postaci kodu JSX. JSX to rozszerzenie języka Java Script pozwalające na definiowanie znaczników HTML w kodzie Java Script co znacznie upraszcza proces tworzenia elementów interfejsu. Kolejnym istotnym aspektem jest wykorzystanie przez React.js wirtualnej wersji obiektowego modelu dokumentu (ang. Document Object Model, DOM) do skutecznego zarządzania aktualizowaniem interfejsu. Zamiast bezpośrednio zarządzać rzeczywistym drzewem DOM, React.js porównuje wirtualny DOM utworzony w swojej pamięci z rzeczywistym i wprowadza zmiany tylko w niezbędnych miejscach, co w efekcie daje zwiększoną wydajność [12] [13].

Ze względu na użycie frameworka React.js należy również wspomnieć o konieczności wykorzystania języka HTML, dzięki któremu możliwe będzie zdefiniowanie struktury strony dla aplikacji [14].

W celu zapewnienia obecnych standardów w narzędziu zostanie wykorzystany język arkuszy stylów CSS. Jest to język pozwalający na określanie właściwości znaczników HTML, takich jak kolor, rozmiar, położenie i wiele innych. Zastosowanie go umożliwia oddzielenie treści od prezentacji i zdefiniowanie wyglądu strony w jednym miejscu [15].

4.4.2. Baza danych

Biorąc pod uwagę przyjęte założenia na temat narzędzia można stwierdzić, że nie jest koniecznym aby dane były zbierane na serwerze aplikacji webowej, a dobrym wyjściem jest zbieranie ich lokalnie na urządzeniu użytkownika bez konieczności ciągłego przesyłania, co dodatkowo może wpłynąć na zwiększenie wydajności oraz uniezależni działanie aplikacji od połączenia z siecią. Istnieje kilka rozwiązań stosowanych powszechnie w różnych aplikacjach w takich właśnie sytuacjach. Jednym z nich jest „local storage".

Local storage to określenie pamięci używanej przez przeglądarki internetowe wykorzystywanej przez Java Script. Zaletą local storage jest przechowywanie pamięci nawet po zamknięciu przeglądarki w formie par „klucz - wartość". Dzięki temu możliwe jest zapisywanie różnych informacji dotyczących aplikacji webowych. Local storage jest dosyć prostym narzędziem, jednakże posiada ono kilka znaczących ograniczeń. Rozmiar pamięci jaka może być przechowana za pomocą local storage dla większości współczesnych przeglądarek wynosi 5 MB na domenę. Dodatkowo jedynym typem danych jaki może być przechowywany przez local storage jest string, co wymaga dodatkowych konwersji typów danych za każdym razem kiedy chcemy zapisać dane innego typu niż tekstowe, co w pewnym stopniu może tworzyć utrudnienia. Ze względu na swoje cechy local storage jest idealnym wyborem na zapisywanie informacji, których liczba jest stała i nieduża, takich jak ustawienia [16].

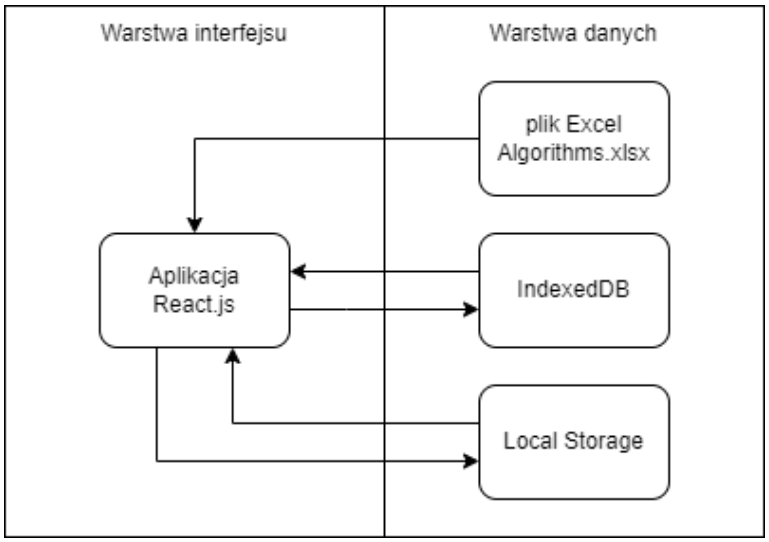
Z perspektywy tworzonego w pracy narzędzia, w którym zapisywane rekordy uzyskiwanych rezultatów będą się stale zwiększać, limit rozmiaru pamięci może stanowić barierę. Z tego powodu kolejnym wybranym rozwiązaniem będzie IndexedDB. Jest to przeglądarkowa baza danych NoSQL. Pozwala ona na przechowywanie większych ilości danych i umożliwia bardziej zaawansowane operacje niż local storage [17].

4.4.3. Narzędzia wspomagające

W celu zapewnienia wydajności pracy nad tworzeniem narzędzia konieczne jest zastosowanie rozwiązań wspomagających, które usprawnią pracę z kodem.

Jako zintegrowane środowisko programistyczne (and. Integrated Development Environment, IDE) zostanie wykorzystane narzędzie WebStorm 2023 od firmy JetBrains. Narzędzie to ułatwia pracę z kodem poprzez między innymi dynamiczne oznaczanie poszczególnych części kodu odpowiednim kolorem, dynamiczne sugerowanie wpisywanych nazw wraz z autouzupełnianiem, zaawansowane narzędzia do debugowania czy integrację z popularnymi systemami kontroli wersji. Jest to jedno z wielu IDE firmy JetBrains przeznaczone głównie do pracy w języku Java Script oraz w innych powiązanych z Java Scipt technologii, w tym w React.js.

Jako system kontroli wersji zostanie zastosowany Git. Jest to obecnie najpopularniejsze rozwiązanie umożliwiające zapisywanie postępów pracy tworzenia oprogramowania. Dzięki Git'owi praca nad tworzeniem narzędzia będzie przebiegać bardziej efektywnie. Git pozwoli na bardziej bezpieczne wprowadzanie zmian oraz kolejnych elementów aplikacji, bez obawy o utratę działającej wersji aplikacji. Możliwe również będzie łatwe zapisywanie postępów pracy w zdalnym repozytorium GitHub co da możliwość pracy nad aplikacją z dowolnego urządzenia [18].



Rysunek 4.1 Architektura narzędzia (opracowanie własne)

4.5. Architektura narzędzia

4.5.1. Struktura aplikacji

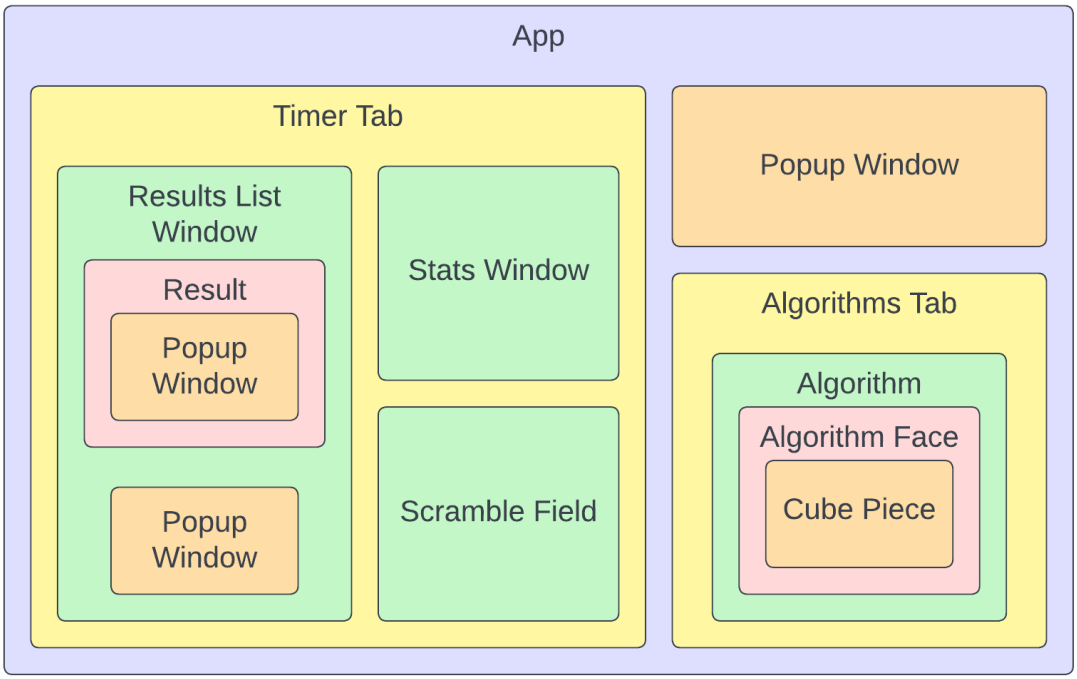
Elementy składające się na narzędzie można podzielić na dwie warstwy struktury: warstwę interfejsu oraz warstwę danych. Diagram obrazujący architekturę narzędzia został przedstawiony na rysunku 4.1. Poniżej znajduje się opis każdego przepływu danych między warstwami narzędzia.

1. Dane z pliku arkusza kalkulacyjnego Algorithms.xlsx są importowane do aplikacji w momencie jej uruchamiania.
2. W momencie uruchamiania aplikacji dane dotyczące uzyskiwanych wyników z bazy danych IndexedDB są przesyłane do aplikacji, a także są one przesyłane później w każdym momencie wystąpienia zmiany w bazie danych.
3. Kiedy użytkownik zatrzymuje czas ułożenia kostki Rubika lub wykonania algorytmu, wynik jest zapisywany do bazy danych IndexedDB
4. W momencie uruchamiania aplikacji dane dotyczące ustawień, a także dane dotyczące stanu listy algorytmów są przesyłane do aplikacji z narzędzia local storage
5. Kiedy użytkownik dokona zmian ustawień lub dokona zmiany dotyczącej listy algorytmów, zmiana jest zapisywana w narzędziu local storage

4.5.2. Architektura komponentów React

Na rysunku 4.2 został przedstawiony diagram komponentów React. Głównym komponentem jest komponent aplikacji App. Ten komponent będzie pozostawał wyrenderowany przez cały czas działania aplikacji od momentu jej uruchomienia co sprawi, iż będzie mógł on pełnić funkcję miejsca zarządzania globalnym stanem aplikacji. Dzięki temu, zewnętrzne dane będą mogły być w łatwy sposób udostępniane do innych komponentów z jednego miejsca. Wpływa to na poprawę elastyczności aplikacji w zakresie wprowadzania zmian. Komponentem podrzędnym do komponentu App jest Popup Window. Komponent ten jest wykorzystany wielokrotnie. Jego działanie będzie polegało na wyświetlaniu podanej mu zawartości w formie wyskakującego okna. To użycie komponentu jako komponent podrzędny do komponentu App będzie odpowiedzialne za wyświetlanie okna ustawień oraz za wyświetlanie okna pomocy.

Kolejnymi komponentami są Timer Tab oraz Algorithms Tab. Będą to dwie oddzielne zakładki w formie ekranów aplikacji.



Rysunek 4.2 Diagram komponentów React (opracowanie własne)

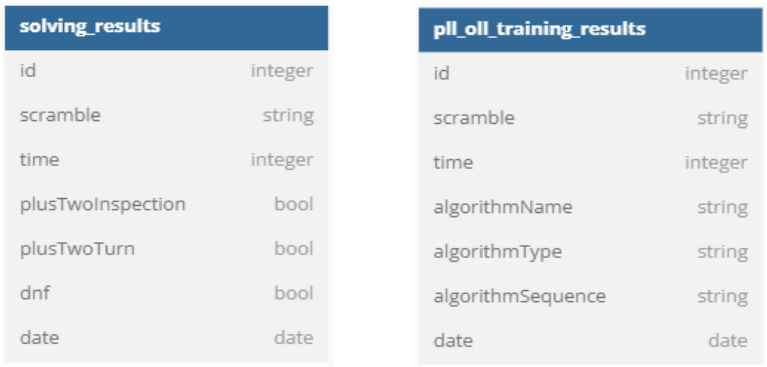
W zakładce Timer Tab użytkownik będzie mógł trenować układanie oraz wykonywanie algorytmów. Komponent Timer Tab zawiera podrzędne komponenty takie jak Stats Window, który będzie wyświetlał statystyki dotyczące uzyskiwanych wyników, Scramble Field, który będzie wyświetlał sekwencje mieszającą oraz Results List Window, który będzie wyświetlał listę uzyskanych wyników. Komponent Results List Window będzie zawierał komponent Popup Window przeznaczone do zatwierdzania operacji usuwania wszystkich uzyskanych wyników, tak by użytkownik nie mógł wykonać tej operacji przypadkowo. Lista uzyskanych wyników będzie tak naprawdę składała się z wielu kopii komponentu Result. Każdy komponent Result będzie jednym wynikiem na liście, który po kliknięciu otworzy komponent Popup Window zawierający szczegóły danego wyniku.

W zakładce Algorithms Tab będzie wyświetlona lista algorytmów metody CFOP. Każdy algorytm na liście będzie kopią komponentu Algorithm. Komponent Algorithm będzie wyświetlał więc dane na temat algorytmu. Za wyświetlanie wizualizacji stanu kostki będzie odpowiadał komponent Algorithm Face, a ten będzie złożony z wielu podrzędnych komponentów Cube Piece.

4.5.3. Struktura bazy danych

Baza danych IndexedDB będzie przeznaczona do przechowywania danych dotyczących uzyskiwanych przez użytkownika rezultatów. Rezultaty jakie będzie uzyskiwał użytkownik można podzielić na dwa typy: rezultaty zwykłego trenowania układania kostki Rubika oraz rezultaty trenowania algorytmów. Te typy rezultatów różnią się pod względem informacji jakie mogą być istotne, dlatego struktura bazy danych będzie oparta o dwie różne tabele faktów. Atrybuty obu tabel są widoczne na rysunku 4.3.

Tabela solving_results zawiera informacje na temat klasycznych ułożeń kostki Rubika. W każdym istniejącym rozwiązaniu kary nakładane na ułożenie były ograniczone jedynie do kary dwóch sekund oraz kary dnf, jednakże podjąłem decyzję aby umieścić w narzędziu możliwość dodawania oddzielnie kary dwóch sekund za przedłużenie czasu inspekcji oraz



Rysunek 4.3 Schemat bazy danych IndexedDB (opracowanie własne)

kary dwóch sekund z powodu brakującego ruchu do ukończenia ułożenia. Możliwe będzie nałożenie obu kar na jeden wynik, a wtedy dodatkowe sekundy będą się kumulowały. Dzięki temu narzędzie będzie w większym stopniu odzwierciedlać procedurę układania kostki Rubika na zawodach. Atrybuty tabeli:

- id – identyfikator rekordu
- scramble – wygenerowana sekwencja mieszająca
- time – uzyskany czas w jednostce 10 milisekund
- plusTwoInspection – wartość logiczna oznaczająca czy na wynik została nałożona kara dwóch sekund z powodu przedłużenia czasu inspekcji
- plusTwoTurn – wartość logiczna oznaczająca czy na wynik została nałożona kara dwóch sekund z powodu brakującego ruchu do ukończenia ułożenia
- dnf – wartość logiczna oznaczająca czy wynik został uznany za nieważny z powodu nieułożenia kostki Rubika
- date – data uzyskania wyniku

Tabela pll_oll_training_results zawiera informacje na temat wyników wykonanych algorytmów. Atrybuty algorithmName, algorithmType oraz algorithmSequence zawierają informacje na temat algorytmu, który był ćwiczony. Jednym z rozpatrywanych przeze mnie rozwiązań było zamienienie tych atrybutów na pojedynczy identyfikator algorytmu, dzięki któremu można by uzyskać te same informacje z danych otrzymanych z pliku excel jednocześnie nie powielając ich niepotrzebnie, jednakże zdecydowałem pozostawić ową tabelę w obecnej formie. Jest to spowodowane chęcią oddzielenia uzyskanych już wyników od innych danych. Dzięki temu, możliwa będzie modyfikacja danych na temat algorytmów, na przykład w przyszłości przy dalszym rozwijaniu narzędzia, bez obawy o naruszenie danych uzyskanych wyników. Atrybuty tabeli pll_oll_training_results:

- id – identyfikator rekordu
- scramble – wygenerowana sekwencja mieszająca
- time – uzyskany czas w jednostce 10 milisekund
- algorithmName – nazwa wykonanego algorytmu
- algorithmType – typ wykonanego algorytmu, pll lub oll
- algorithmSequence – sekwencja rozwiązująca algorytmu
- date – data uzyskania wyniku

5. Implementacja

W tym rozdziale zostaną opisane szczegóły dotyczące procesu implementowania narzędzia na podstawie wcześniej stworzonego projektu narzędzia. Zostaną przedstawione szczególne problemy napotkane podczas implementowania oraz zastosowane rozwiązania.

Narzędzie zostało zaimplementowane zgodnie z projektem oraz z zastosowaniem dobrych praktyk programowania [19].

5.1. Struktura plików

Struktura plików projektu aplikacji jest pokazana na rysunku 5.1. W katalogu głównym znajdują się: katalog „public”, katalog „src” oraz pliki .gitignore, package.json i package-lock.json.

5.1.1. Pliki package.json oraz package-lock.json

Pliki package.json oraz package-lock.json to kluczowe elementy projektów React.js a także projektów Node.js. Format JSON (JavaScript Object Notation) jest prostym typem danych opartym o pary klucz-wartość w formacie tekstowym [20].

Plik package.json służy do zapisywania podstawowych informacji o projekcie, takich jak nazwa, wersja, opis oraz inne metadane. Dodatkowo są w nim zapisane listy skryptów służących do uruchamiania różnych poleceń, oraz listy zależności czyli dodatkowych bibliotek wraz z określeniem wersji każdej zależności lub podaniem przedziału wersji.

Plik package-lock.json służy do zapisywania szczegółowych informacji na temat wersji wszystkich wykorzystywanych zależności i pełni on rolę zabezpieczenia zgodności wersji zależności projektu. Oba pliki są generowane automatycznie w momencie tworzenia projektu React.js oraz aktualizowane w momencie instalowania nowej zależności do projektu. Informacje o konfiguracji projektu w pliku package.json powinny być wpisane ręcznie, jednakże nie wpływa to na działanie aplikacji.

5.1.2. Plik .gitignore

Plik .gitignore to plik, w którym definiowane są pliki oraz katalogi, które powinny być ignorowane przez narzędzie git. Oznacza to, że te pliki nie będą się wyświetlały w narzędziu git jako potencjalne pliki repozytorium. W owym projekcie plik ten pozwala na ignorowanie katalogów .idea, czyli katalogu ustawień używanego IDE WebStorm oraz katalogu node_modules, w którym zainstalowane są zależności projektu. Oba ignorowane katalogi nie powinny być dodawane do repozytorium git projektu.

5.1.3. Katalog „public”

W katalogu „public” znajdują się zasoby statyczne. Należą do nich index.html, icon.png oraz Algorithms.xlsx w podkatalogu „excel”.

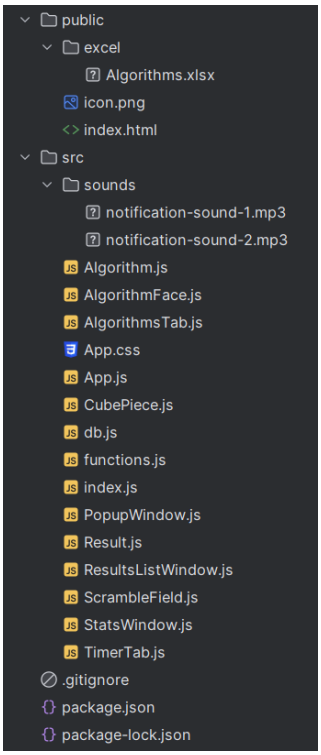
Plik index.html to główny plik HTML aplikacji, który stanowi podstawę dla całego interfejsu. Znajduje się w nim root drzewa DOM, czyli element HTML, do którego są dołączane pozostałe elementy. Dodatkowo w pliku znajdują się metadane, takie jak tytuł strony, opis czy kolory motywów.

Plik icon.png to obraz będący ikoną strony. Dla celów pracy została stworzona prosta ikona.

Plik Algorithms.xlsx znajdujący się w podkatalogu „excel” to dane dotyczące listy algorytmów, które są wczytywane do aplikacji w momencie jej uruchamiania. Dokładny opis tego pliku znajduje się w podrozdziale 5.2.

5.1.4. Katalog „src”

W katalogu „src” znajduje się cały kod źródłowy aplikacji React. Kod źródłowy jest podzielony na 14 plików kodu JavaScript. Każdy plik, którego nazwa rozpoczyna się z dużej litery jest oddzielnym komponentem React. Taki podział pozwala na lepszą organizację kodu co zwiększa wygodę pracy z kodem. Pozostałymi plikami JavaScript są db.js, w którym znajduje się definicja bazy danych IndexedDB, functions.js, który zawiera funkcje



Rysunek 5.1 Struktura plików

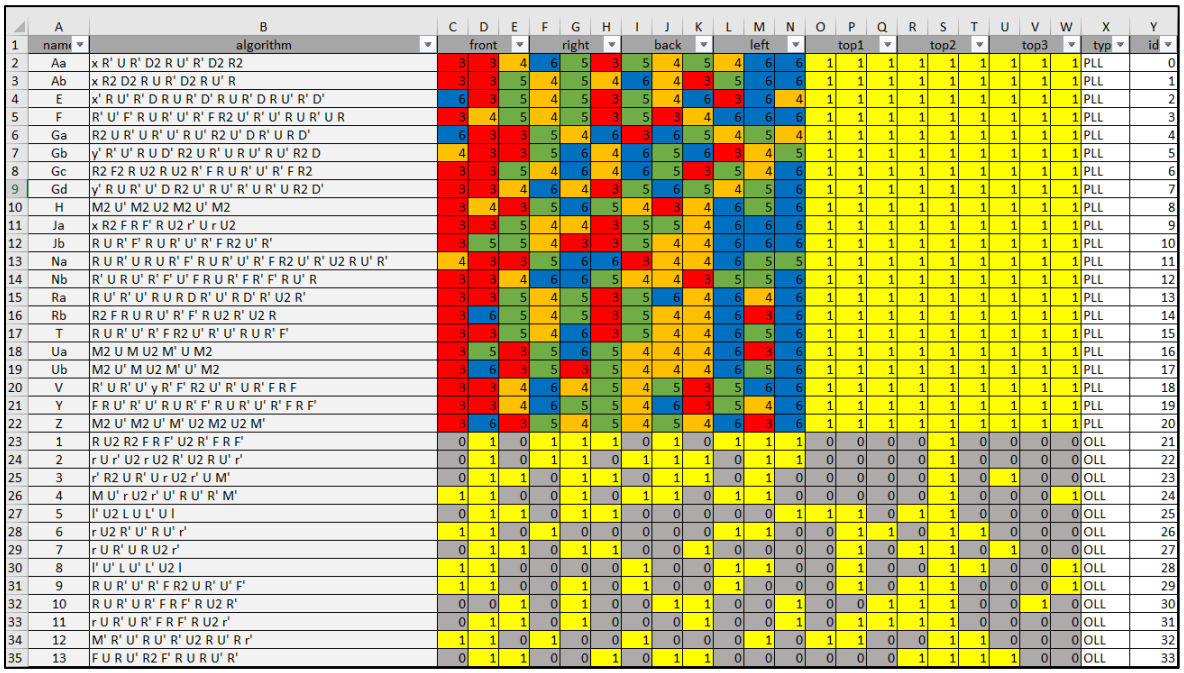
wykorzystywane wielokrotnie w więcej niż jednym komponencie oraz index.js, który inicjalizuje aplikację React.

Dodatkowo w katalogu „src" znajduje się plik App.css, który zawiera definicję wszystkich stylów dla komponentów oraz podkatalog „sounds" z dwoma plikami dźwiękowymi w formacie mp3 będące sygnałami odtwarzanymi w trakcie inspekcji w momencie minięcia ośmiu i dwunastu sekund.

5.2. Wczytywanie danych z pliku Excel

W celu umożliwienia w narzędziu zmiany orientacji wizualizowanej kostki Rubika, dane dotyczące stanu kostki Rubika dla którego algorytm jest przeznaczony muszą znaleźć się w formie liczbowej, tak by każda ścianka kostki Rubika była reprezentowana przez liczbę. Dzięki temu możliwe będzie odpowiednie zmienienie przypisania kolorów do poszczególnych ścianek co da efekt zmiany orientacji kostki Rubika. Ręczne wpisywanie kolejnych liczb reprezentujących wygląd kostki dla każdego z 78 przypadków może zostać uproszczone dzięki wykorzystaniu funkcji formatowania warunkowego programu Excel. Komórki w tabeli zmieniają kolor swojego tła w zależności od liczby jaka zostanie w nich wpisana, co ułatwia weryfikację poprawności wpisanych danych oraz znajdywanie błędów. Kolory komórek odpowiadają wybranej jednej z możliwych orientacji na kostce Rubika. Wykorzystane formatowanie warunkowe jest następujące:

- 0 – ściana dowolna (wykorzystany dla algorytmów oll, dla których istotny jest tylko kolor górnej ściany)
- 1 – ściana górna
- 2 – ściana dolna
- 3 – ściana przednia
- 4 – ściana tylna
- 5 – ściana prawa
- 6 – ściana lewa



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	nam	algorithm	front			right			back			left			top1			top2			top3			typ	id
2	Aa	x R' U R' D2 R U' R' D2 R2	3	3	4	6	5	3	5	4	5	4	6	6	1	1	1	1	1	1	1	1	1	PLL	0
3	Ab	x R2 D2 R U R' D2 R U' R	3	3	5	4	5	4	6	4	3	5	6	6	1	1	1	1	1	1	1	1	1	PLL	1
4	E	x' R U' R' D R U R' D' R U R' D R U' R' D'	6	3	5	4	5	3	5	4	6	3	6	4	1	1	1	1	1	1	1	1	1	PLL	2
5	F	R' U' F' R U R' U' R' F R2 U' R' U' R U R' U R	3	4	5	4	5	3	5	3	4	6	6	6	1	1	1	1	1	1	1	1	1	PLL	3
6	Ga	R2 U R' U R' U' R U' R2 U' D R' U R D'	6	3	3	5	4	6	3	6	5	4	5	4	1	1	1	1	1	1	1	1	1	PLL	4
7	Gb	y' R' U' R U D' R2 U R' U R U' R U' R2 D	4	3	3	5	6	4	6	5	6	3	4	5	1	1	1	1	1	1	1	1	1	PLL	5
8	Gc	R2 F2 R U2 R U2 R' F R U R' U' R' F R2	3	3	5	4	6	4	6	5	3	5	4	6	1	1	1	1	1	1	1	1	1	PLL	6
9	Gd	y' R U R' U' D R2 U' R U' R' U R' U R2 D'	3	3	4	6	4	3	5	6	5	4	5	6	1	1	1	1	1	1	1	1	1	PLL	7
10	H	M2 U' M2 U2 M2 U' M2	3	4	3	5	6	5	4	3	4	6	5	6	1	1	1	1	1	1	1	1	1	PLL	8
11	Ja	x R2 F R F' R U2 r' U r U2	3	3	5	4	4	3	5	5	4	6	6	6	1	1	1	1	1	1	1	1	1	PLL	9
12	Jb	R U R' F' R U R' U' R' F R2 U' R'	3	5	5	4	3	3	5	4	4	6	6	6	1	1	1	1	1	1	1	1	1	PLL	10
13	Na	R U R' U R U R' F' R U R' U' R' F R2 U' R' U2 R U' R'	4	3	3	5	6	6	3	4	4	6	5	5	1	1	1	1	1	1	1	1	1	PLL	11
14	Nb	R' U R U' R' F' U' F R U R' F R' F' R U' R	3	3	4	6	6	5	4	4	3	5	5	6	1	1	1	1	1	1	1	1	1	PLL	12
15	Ra	R U' R' U' R U R D R' U' R D' R' U2 R'	3	3	5	4	5	3	5	6	4	6	4	6	1	1	1	1	1	1	1	1	1	PLL	13
16	Rb	R2 F R U R U' R' F' R U2 R' U2 R	3	6	5	4	5	3	5	4	4	6	3	6	1	1	1	1	1	1	1	1	1	PLL	14
17	T	R U R' U' R' F R2 U' R' U' R U R' F'	3	3	5	4	6	3	5	4	4	6	5	6	1	1	1	1	1	1	1	1	1	PLL	15
18	Ua	M2 U M U2 M' U M2	3	5	3	5	6	5	4	4	4	6	3	6	1	1	1	1	1	1	1	1	1	PLL	16
19	Ub	M2 U' M U2 M' U' M2	3	6	3	5	3	5	4	4	4	6	5	6	1	1	1	1	1	1	1	1	1	PLL	17
20	V	R' U R' U' y R' F' R2 U' R' U R' F R F	3	3	4	6	4	5	4	5	3	5	6	6	1	1	1	1	1	1	1	1	1	PLL	18
21	Y	F R U' R' U' R U R' F' R U R' U' R' F R F'	3	3	4	6	5	5	4	6	3	5	4	6	1	1	1	1	1	1	1	1	1	PLL	19
22	Z	M2 U' M2 U' M' U2 M2 U2 M'	3	6	3	5	4	5	4	5	4	6	3	6	1	1	1	1	1	1	1	1	1	PLL	20
23	1	R U2 R2 F R F' U2 R' F R F'	0	1	0	1	1	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	OLL	21
24	2	r U r' U2 r U2 R' U2 R U' r'	0	1	0	1	1	0	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0	OLL	22
25	3	r' R2 U R' U r U2 r' U M'	0	1	1	0	1	1	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	OLL	23
26	4	M U' r U2 r' U' R U' R' M'	1	1	0	0	1	0	1	1	0	1	1	0	0	0	0	0	1	0	0	0	1	OLL	24
27	5	l' U2 L U L' U l	0	1	1	0	1	1	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	OLL	25
28	6	r U2 R' U' R U' r'	1	1	0	1	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0	0	0	OLL	26
29	7	r U R' U R U2 r'	0	1	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	1	0	0	OLL	27
30	8	l' U' L U' L' U2 l	1	1	0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1	0	0	1	OLL	28
31	9	R U R' U' R' F R2 U R' U' F'	1	1	0	0	1	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0	1	OLL	29
32	10	R U R' U R' F R F' R U2 R'	0	0	1	0	1	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0	1	OLL	30
33	11	r U R' U R' F R F' R U2 r'	0	1	1	0	1	0	0	0	1	0	0	1	0	1	1	1	1	0	0	0	0	OLL	31
34	12	M' R' U' R U' R' U2 R U' R r'	1	1	0	1	0	0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	0	OLL	32
35	13	F U R U' R2 F' R U R U' R'	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	OLL	33

Rysunek 5.2 Plik Algorithms.xlsx

Kolumny tabeli „front”, „right”, „back” oraz „left” odpowiadają kolorom kwadratów górnego rzędu bocznych ścian kostki licząc kwadraty od lewej do prawej strony przechodząc dookoła kostki. Kolumny „top1”, „top2” oraz „top3” odpowiadają kwadratom górnej ściany, gdzie każda kolejna kolumna to kolejny rząd od góry licząc kwadraty od lewej do prawej strony. Takie zorganizowanie pliku Excel pozwoliło na wygodne i sprawne wprowadzanie danych dla całej listy algorytmów. Wygląd pliku arkusza kalkulacyjnego programu Excel Algorithms.xlsx został przedstawiony na rysunku 5.2.

Podczas uruchamiania aplikacji, dane z pliku Excel są przesyłane do głównego komponentu App oraz przekształcone na typ danych JSON.

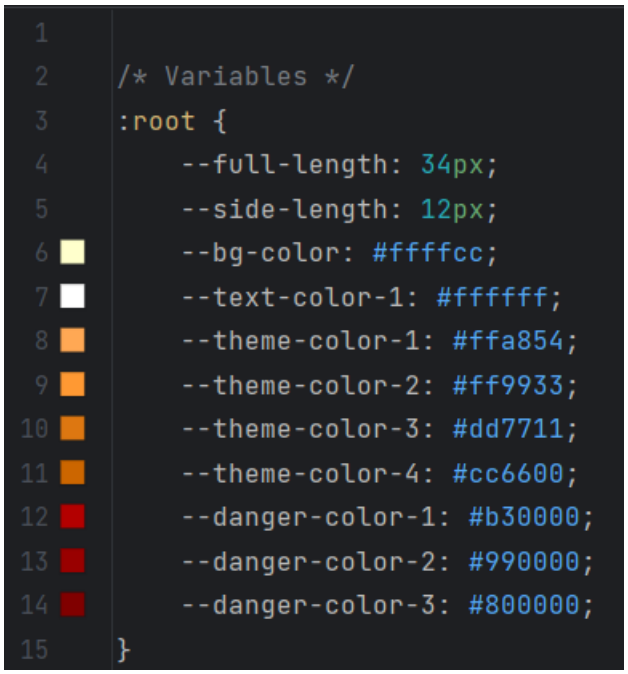
5.3. Baza danych IndexedDB

Wykorzystanie przeglądarkowej bazy danych IndexedDB odbyło się z pomocą biblioteki Dexie.js. Dexie.js zapewnia interfejs dla IndexedDB. Dzięki niemu korzystanie z bazy danych IndexedDB staje się proste, a wszelkie operacje dotyczące baz danych odbywają się w sposób asynchroniczny.

Baza danych IndexedDB została zdefiniowana w pliku db.js w formie dwóch oddzielnych tabel, tak jak przedstawiane to było w projekcie narzędzia. Za pomocą tak zwanego hook’a „useLiveQuery” zaimportowanego z biblioteki dexie-react-hooks, jest tworzone zapytanie do bazy danych, którego wynik jest automatycznie śledzony przez hook’a. Umożliwia to automatyczne ponowne renderowanie się komponentów w momencie wystąpienia zmiany w bazie danych. Dodatkowo, w celu uproszczenia, tworzone są dwa osobne zapytania dla wyników algorytmów PLL oraz OLL co wymaga zastosowania filtrów po wartości pola algorithmType.

5.4. Wygląd interfejsu

W celu zdefiniowania wyglądu poszczególnych elementów interfejsu została wykorzystana technologia CSS. W pliku App.css zostały umieszczone definicje wszystkich stylów dla komponentów z zastosowaniem praktyk upraszczających pracę z technologią CSS takimi jak zmienne czy komentarze. Wykorzystanie zmiennych wraz z wybraną paletą kolorów dla interfejsu jest widoczne na rysunku 5.3 [15].



Rysunek 5.3 Definicja zmiennych w pliku App.css

Wygląd interfejsu można zaobserwować na rysunkach 5.6 oraz 5.15. Ogólny motyw kolorystyczny narzędzia został wybrany na motyw jasny. Tło narzędzia jest w kremowym kolorze, a czcionka tekstu znajdującego się bezpośrednio na tle jest koloru czarnego w celu wysokiego kontrastu i zapewnienia czytelności. Kolor elementów interfejsu takich jak okna czy pasek nawigacyjny został ustawiony na różne odcienie koloru pomarańczowego w celu budowania skojarzeń z optymizmem oraz energią, natomiast czcionka na tych elementach jest koloru białego w celu uzyskania czytelności. Odcienie koloru czerwonego są stosowane w celu oznaczenia przycisków dotyczących nieodwracalnych operacji takich jak usunięcie danego lub wszystkich wyników. Kolory oznaczające stany poszczególnych algorytmów na liście algorytmów (rys. 5.15) mają następujące znaczenie:

- kolor szary – algorytm nieznany
- kolor żółty – algorytm obecnie uczony
- kolor zielony – algorytm nauczony

5.5. Pasek nawigacyjny

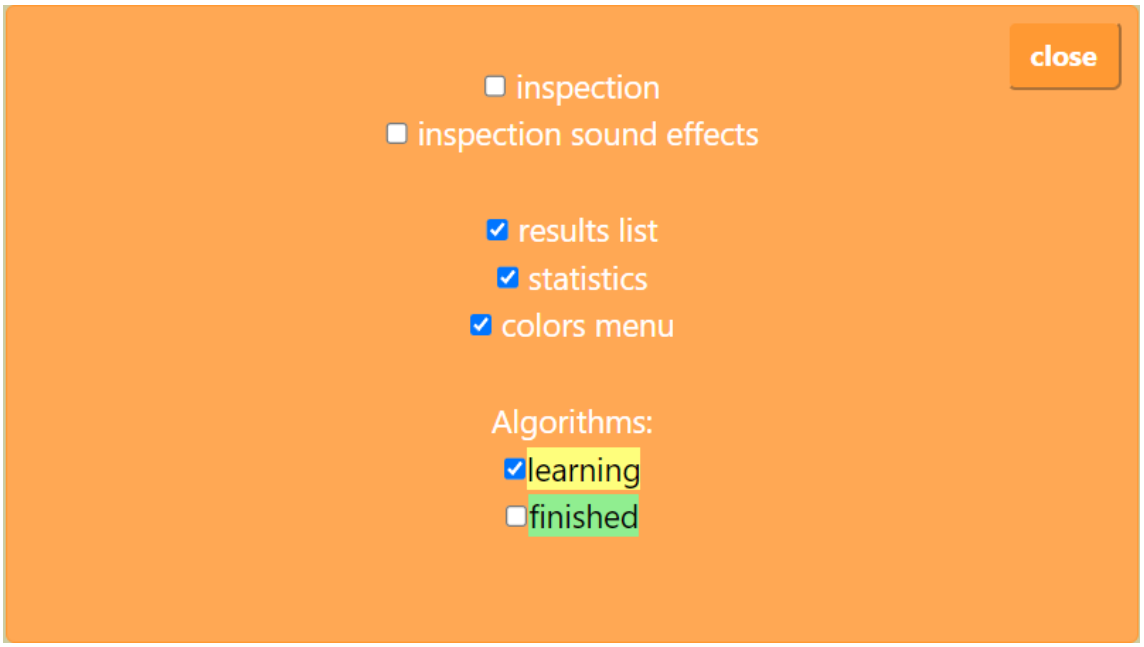
Z poziomu głównego komponentu aplikacji App został zdefiniowany element interfejsu, który można określić jako pasek nawigacyjny. Jest on widoczny w każdym momencie korzystania z narzędzia przy górnej krawędzi ekranu. Znajdują się na nim przyciski przenoszące do zakładki stopera lub zakładki listy algorytmów aplikacji oraz przyciski otwierające okno ustawień i okno pomocy.

5.5.1. Okno ustawień

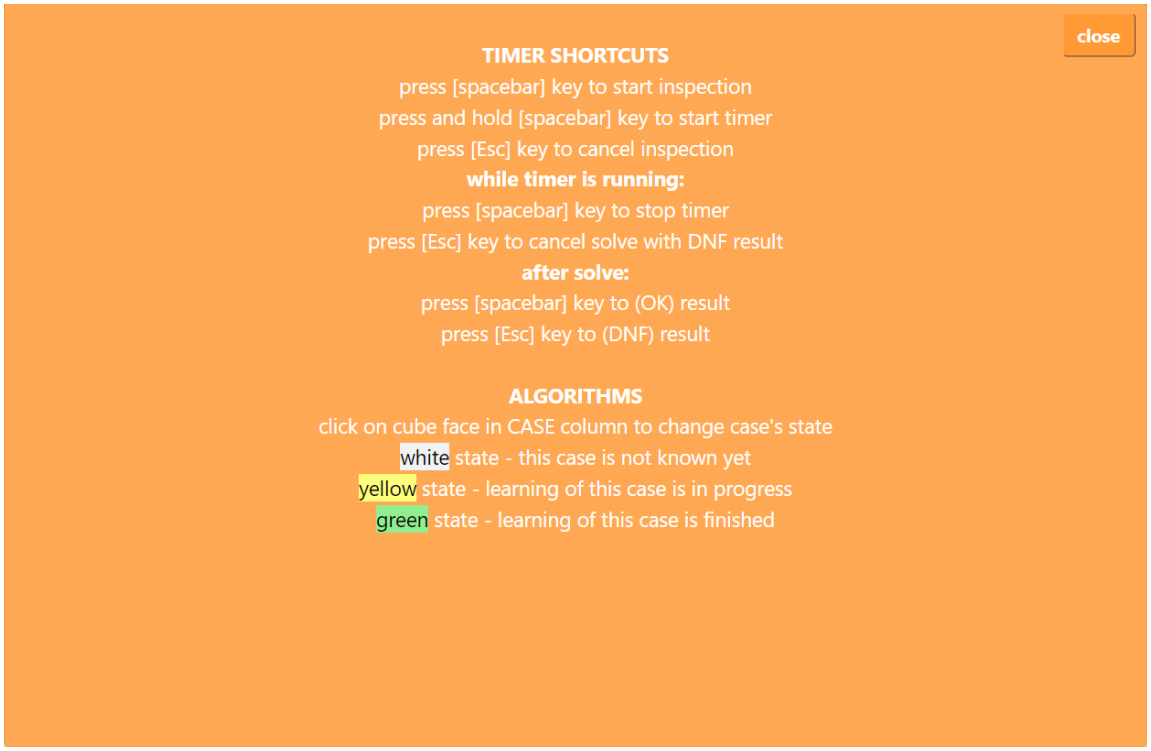
Po kliknięciu w przycisk „Settings” otwiera się wyskakujące okno zawierające dostępne ustawienia narzędzia w formie checkbox’ów, które użytkownik może zmieniać (rys. 5.4).

Pierwsze dwie pozycje ustawień dotyczą procedury inspekcji. Użytkownik może zdecydować czy chce aby procedura inspekcji była włączona oraz czy chce włączyć sygnały dźwiękowe dla momentów upływu ośmiu i dwunastu sekund inspekcji.

Kolejne trzy pozycje ustawień dotyczą wyświetlania okien narzędzia: okna uzyskanych wyników, okna statystyk oraz okna zmiany orientacji kostki Rubika. To rozwiązanie daje



Rysunek 5.4 Okno ustawień



Rysunek 5.5 Okno pomocy

użytkownikowi możliwość dostosowania interfejsu do własnych potrzeb. Domyślnie wyświetlanie tych okien jest włączone.

Następne dwie pozycje dotyczą wyboru grup, z których losowane są algorytmy do trenowania. Użytkownik może zdecydować czy chce trenować tylko algorytmy oznaczone jako „learning”, czy tylko te oznaczone jako „finished”, lub jednocześnie obie te grupy, albo w przypadku odznaczenia obu grup, ćwiczyć wszystkie dostępne algorytmy bez względu na oznaczenie. To rozwiązanie daje użytkownikowi wybór w postacie dodatkowych prostych sposobów dostosowania swojego treningu algorytmów.

Wszystkie wymienione ustawienia są zapisywane w narzędziu local storage w postaci jednej wartości o formacie JSON pod kluczem „settings”. Ze względu na ograniczenie local storage jakim jest możliwość przechowywania jedynie danych tekstowych, konieczne było zastosowanie operacji zamiany formatu danych JSON właśnie na dane tekstowe. Analogicznie, w momencie uruchamiania narzędzia, zapisane wcześniej dane w local storage są przesyłane, konwertowane z danych tekstowych na format JSON i zapisywane w stanie komponentu App.

5.5.2. Okno pomocy

Po kliknięciu w przycisk „Help” otwiera się wyskakujące okno zawierające część informacji, które mogą być pomocne w korzystaniu z narzędzia (rys. 5.5). Informacje jakie są tu widoczne dotyczą skrótów klawiszowych używanych do obsługi mierzenia czasu oraz krótkiego wyjaśnienia kolorów oznaczających stan algorytmu na liście algorytmów. Warto zaznaczyć, że priorytet tego elementu narzędzia był niski, dlatego na ten moment informacje znajdujące się w oknie pomocy nie wyjaśniają dokładnie do czego służy narzędzie oraz wszystkich jego funkcji.

5.6. Zakładka stopera

Jedną z dwóch głównych zakładek narzędzia jest zakładka stopera. Ekran aplikacji na tej zakładce widoczny jest na rysunku 5.6. Interfejs tej części narzędzia został zaimplementowany zgodnie z omawianymi praktykami dotyczącymi rozmieszczenia elementów w istniejących rozwiązaniach, tak aby mierzony czas wyświetlał się na samym środku a pozostałe elementy były rozłożone na bokach ekranu dookoła czasu. W górnej części zakładki stopera znajdują się przyciski, za pomocą których użytkownik może zmieniać tryb stopera, czyli wybrać trening zwykłego układania kostki Rubika, trening algorytmów PLL lub trening algorytmów OLL.

5.6.1. Generowanie sekwencji mieszającej

Na dole ekranu w zakładce stopera znajduje się wyświetlona sekwencja mieszająca. Nowa sekwencja mieszająca jest generowana automatycznie po zakończeniu każdego ułożenia. Dodatkowo obok sekwencji znajduje się przycisk „next scramble” pozwalający na wygenerowanie nowej sekwencji mieszającej przez użytkownika.

Generowanie sekwencji mieszającej dla trenowania zwykłego układania kostki Rubika odbywa się za pomocą funkcji generateClassicScrambleSequence. Funkcja ta przyjmuje jako argument długość sekwencji.

W celu zapewnienia tego, że kostka Rubika będzie pomieszana poprawnie, długość sekwencji jest ustawiona w narzędziu na 20 ruchów oczywiście traktując ruchy podwójne czyli o 180 stopni jako jeden ruch. Wynika to z przeprowadzonego dowodu przez grupę matematyków, iż każdy możliwy stan kostki Rubika może być rozwiązany w 20 lub mniej ruchów [21]. Oznacza to, że wykorzystując 20 ruchów do pomieszania kostki Rubika istnieje szansa na natrafienie każdego możliwego stanu kostki. Należy jednak zrozumieć, że nie każdy z wykonanych do pomieszania kostki Rubika 20 ruchów „oddala” stan kostki od stanu ułożonego. Szacuje się, że stanów kostki Rubika, które wymagają przynajmniej 20 ruchów do rozwiązania jest bardzo mało w porównaniu do stanów wymagających przynajmniej 18 lub 17 ruchów.

Generowanie klasycznej sekwencji odbywa się poprzez wielokrotne powtarzanie operacji losowego wyboru jednej ze ścian oraz losowego kierunku wykonanego ruchu, czyli zgodnie z ruchem wskazówek zegara, przeciwnie do ruchu wskazówek zegara lub o 180 stopni. Dodatkowo zdefiniowane są dwa warunki, dla których losowanie w danym kroku jest powtarzane. Pierwszym warunkiem jest wylosowanie tej samej ściany co w poprzednim kroku. Wynika to z braku uzasadnienia dla wykonywania dwóch ruchów pod rząd tą samą

Rysunek 5.6 Interfejs zakładki stopera

ścianą. Dwa ruchy pod rząd tą samą ścianą bowiem można zastąpić jednym ruchem. Drugim warunkiem jest wylosowanie tej samej ściany co dwa kroki temu jeżeli jeden krok temu została wylosowana ściana po przeciwnej stronie. Wynika to ze specyfiki przeciwnych ścian kostki Rubika. Wykonanie ruchu daną ścianą nie zmienia stanu części kostki Rubika zależnej od ściany po przeciwnej stronie. Z tego powodu kolejność wykonywania ruchów ścianami po przeciwnych stronach nie jest ważna. Dla przykładu, można wykonać ruch górną ścianą kostki Rubika, a następnie dolną ścianą, po czym cofnąć te ruchy i wykonać je ponownie w odwrotnej kolejności. Uzyskany stan kostki Rubika będzie identyczny w obu próbach.

Generowanie sekwencji mieszającej dla trenowania wykonywania algorytmów PLL lub OLL odbywa się za pomocą funkcji generateAlgorithmScrambleSequence. Proces ten różni się od generowania sekwencji mieszającej dla zwykłego układania kostki Rubika, gdyż chcemy uzyskać stan kostki umożliwiający wykonanie ćwiczenia danego algorytmu, nie zaś pomieszać całą kostkę Rubika.

Na początku funkcja ta losuje jeden z możliwych do wylosowania algorytmów z listy algorytmów. Możliwe do wylosowania algorytmy zależą od wybrania grup algorytmów przez użytkownika w ustawieniach oraz od oznaczenia stanu uczenia się każdego algorytmu przez użytkownika na liście algorytmów.

Kolejnym krokiem jest zamiana ruchów dwoma warstwami, warstwami środkowymi oraz obrotów całą kostką Rubika na ruchy analogiczne, które nie wpływają na zmianę orientacji kostki Rubika. Proces ten jest wykonywany w celu ograniczenia sekwencji mieszającej do podstawowych ruchów notacji. Sekwencje rozwiązujące znajdujące się na liście algorytmów często zawierają ruchy dwoma warstwami, warstwami środkowymi lub obroty całą kostką, pomimo że mogą one być uproszczone do ruchów podstawowej notacji. Wynika to z lepszych możliwości pokazania wygodnego wykonywania danego algorytmu, co może wpływać na polepszenie szybkości jego wykonywania. Przykładowo, możliwe jest zamienienie ruchu środkową warstwą „M” na ruchy „ r’ ” oraz „R”, następnie możliwe jest zamienienie ruchu dwoma warstwami „ r’ ” na ruchy „ L’ ” oraz „ x’ ”, w końcu możliwe jest usunięcie obrotu całą kostką „ x’ ” poprzez zamianę wszystkich ruchów występujących w sekwencji po obrocie całą kostką, dla których pozycja ściany została zmieniona. W celu wyjaśnienia, przykładowo wykonując obrót „ x’ ” ściana górna staje się dla nas ścianą przednią oraz analogiczna zamiana przebiega dla ściany przedniej, dolnej i tylnej, natomiast

ściany prawa i lewa pozostają w tym samym miejscu lub ujmując inaczej, pozostają tymi samymi ścianami.

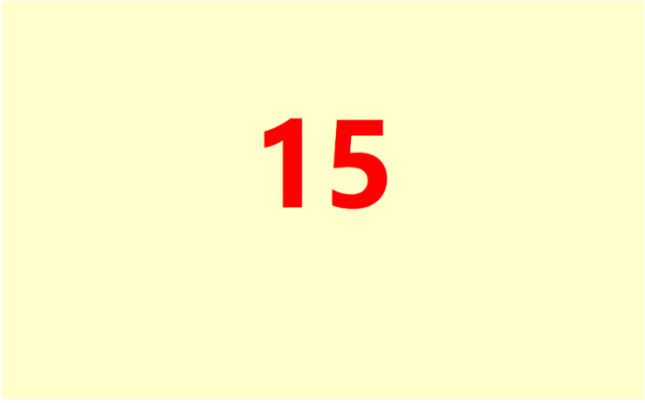
Następnie uproszczona sekwencja rozwiązująca ulega procesowi odwrócenia co oznacza, że kolejność ruchów sekwencji jest odwrócona oraz kierunek każdego ruchu jest zmieniony na przeciwny. Oznacza to że ruchy ścianami zgodnie z ruchem wskazówek zegara są zamieniane na ruchy przeciwne do ruchu wskazówek zegara i odwrotnie, za to ruchy podwójne nie ulegają odwróceniu.

Na koniec jest dodawany ruch górną ścianą w losowym kierunku na początek oraz na koniec uzyskanej sekwencji. Wynika to z potrzeby minimalnego modyfikowania sytuacji na kostce poprzez obrócenie jej, co może pozytywnie wpłynąć na umiejętność odpowiedniego rozpoznawania przypadku przez użytkownika.

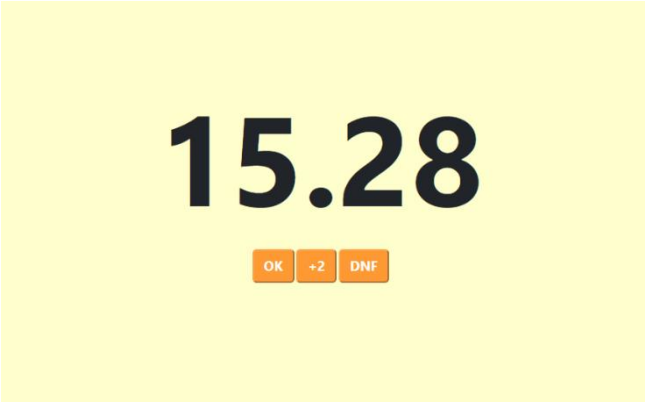
5.6.2. Stoper

W celu odwzorowania procedury układania kostki Rubika na oficjalnych zawodach, stoper służący do mierzenia czasu układania jest obsługiwany za pomocą klawiszy klawiatury. Aby rozpocząć mierzenie czasu znajdując się na ekranie zakładki stopera należy nacisnąć i przytrzymać klawisz spacji na klawiaturze przez pół sekundy. Następnie, po puszczeniu klawisza spacji, mierzenie czasu się uruchamia, mierzony czas jest wyświetlany na bieżąco, a użytkownik powinien zacząć układać kostkę. Dodatkowo pozostałe elementy interfejsu zostają schowane co widać na rysunku 5.7. W przypadku włączonej procedury inspekcji, na początku należy nacisnąć i puścić klawisz spacji, wtedy rozpoczyna się odliczanie piętnastu sekund inspekcji, dodatkowo wyświetlane na ekranie, w trakcie których użytkownik powinien rozpocząć mierzenie czasu w taki sam sposób jak bez procedury inspekcji. Dodatkowo po upłynięciu 8 i 12 sekund inspekcji na ekranie zostają wyświetlone niewielkie komunikaty oraz w przypadku włączonej opcji dźwięku można dodatkowo usłyszeć sygnały dźwiękowe. W momencie kiedy czas inspekcji zostanie przekroczony lub minie 17 sekund, użytkownik nadal ma możliwość uruchomienia mierzenia czasu, jednak po wykonaniu ułożenia zostaje automatycznie nałożona odpowiednia kara. W momencie kiedy użytkownik ułoży kostkę Rubika, powinien nacisnąć klawisz spacji ponownie w celu zatrzymania mierzenia czasu. Po zakończeniu mierzenia czasu zostają wyświetlone przyciski pozwalające na nałożenie odpowiedniej kary na wynik lub określenie go jako poprawny (rys. 5.8). Określenie wyniku jako poprawny dodatkowo może się odbyć poprzez ponowne naciśnięcie klawisza spacji, a nałożenie kary „dnf” na wynik dodatkowo może się odbyć poprzez naciśnięcie klawisza „Esc” na klawiaturze. W celu uniknięcia przypadkowych uruchomień stopera możliwe jest anulowanie przytrzymania klawisza spacji lub procedury inspekcji poprzez wciśnięcie klawisza „Esc”. Dodatkowo poprzez wciśnięcie klawisza „Esc” możliwe jest zatrzymanie mierzenia czasu z automatycznie dodaną karą „dnf”.

Istotną cechą obsługi stopera za pomocą klawiatury jest zmiana koloru czcionki wyświetlanego czasu co ma odzwierciedlać świecenie diody obecnej w fizycznych timer’ach. Podczas procedury inspekcji wyświetlany czas ma kolor czerwony. W momencie chwilowego naciśnięcia klawisza spacji czas zmienia kolor na żółty, natomiast kiedy klawisz spacji jest odpowiednio długo wciśnięty i stoper jest już gotowy do uruchomienia czas zmienia kolor na zielony.



Rysunek 5.7 Interfejs w trakcie procedury inspekcji



Rysunek 5.8 Interfejs po zakończeniu mierzenia czasu

5.6.3. Okno statystyk

Okno statystyk umieszczone jest w górnym prawym rogu ekranu zakładki stopera Wyświetlane w nim informacje są zależne od wybranego trybu stopera.

Dla zwykłego treningu układania kostki Rubika wyświetlane statystyki to (rys. 5.9):

- najlepszy wynik
- średnia z ostatnich pięciu wyników,
- średnia z ostatnich dwunastu wyników
- średnia ze wszystkich wyników

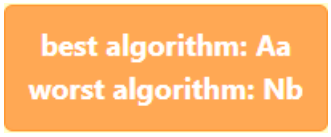
Istotnym faktem to, iż średnie z ostatnich pięciu i dwunastu wyników są obliczane z wyłączeniem najlepszego i najgorszego wyniku w celu odwzorowania liczenia wyników na oficjalnych zawodach, natomiast nie dotyczy to średniej ze wszystkich wyników. Dodatkowo, jeżeli w ostatnich pięciu wynikach znajdą się dwa lub więcej wyników

oznaczonych jako „dnf”, nie jest możliwe obliczenie średniej. Sprawia to, że w serii pięciu ułożeń możliwe jest nieukończenie tylko jednego ułożenia aby móc uzyskać średnią. Taka sama zasada jest stosowana dla średniej z dwunastu wyników, jednakże ponownie nie dotyczy to średniej ze wszystkich wyników.

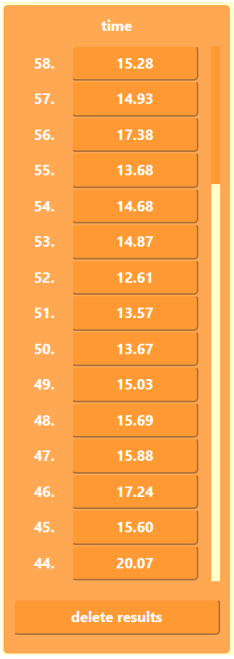
Dla treningu wykonywania algorytmów metody CFOP wyświetlane statystyki to (rys. 5.10):

- nazwa najlepiej wykonywanego algorytmu
- nazwa najgorzej wykonywanego algorytmu

Rysunek 5.9 Okno statystyk dla zwykłego treningu układania kostki Rubika



Rysunek 5.10 Okno statystyk dla treingu wykonywania algorytmów



Rysunek 5.11 Okno listy uzyskanych wyników

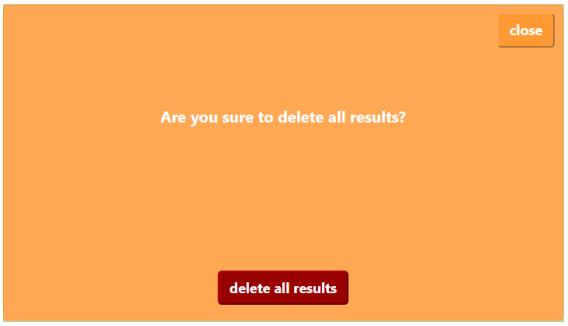
Do określania najlepiej i najgorzej wykonywanego algorytmu są wykorzystywane średnie wszystkich wyników dla każdego algorytmu. Pomimo że wykonywane algorytmy metody CFOP mogą różnić się długością wykonywanych ruchów, nie jest to brane pod uwagę podczas określania najlepiej i najgorzej wykonywanego algorytmu. Wynika to z faktu przeznaczenia algorytmów do układania kostki Rubika jak najszybciej, dlatego zawodnicy powinni dążyć do wykonywania algorytmów jak najszybciej bez względu na ich liczbę ruchów. Poza tym szybkość wykonywania algorytmu nie jest bezpośrednio związana z jego liczbą ruchów, a bardziej zależy od specyfiki algorytmu wymagającej odpowiedniego używania palców w danej części algorytmu.

5.6.4. Okno listy uzyskanych wyników

Okno listy uzyskanych wyników jest umieszczone po lewej stronie ekranu w zakładce stopera (rys 5.11). Wyświetlana jest w nim lista wyników uzyskanych dla danego trybu stopera. Możliwe jest przewijanie listy za pomocą paska bocznego w momencie kiedy wyników robi się odpowiednio dużo. Wyniki te znajdują się w formie przycisków z uzyskanym czasem. Kliknięcie na dany wynik otwiera wyskakujące okno ze szczegółowymi

Rysunek 5.12 Okno wyniku treningu zwykłego układania kostki Rubika

Rysunek 5.13 Okno wyniku treningu wykonywania algorytmów metody CFOP



Rysunek 5.14 Okno z pytaniem o potwierdzenie operacji usunięcia wyników

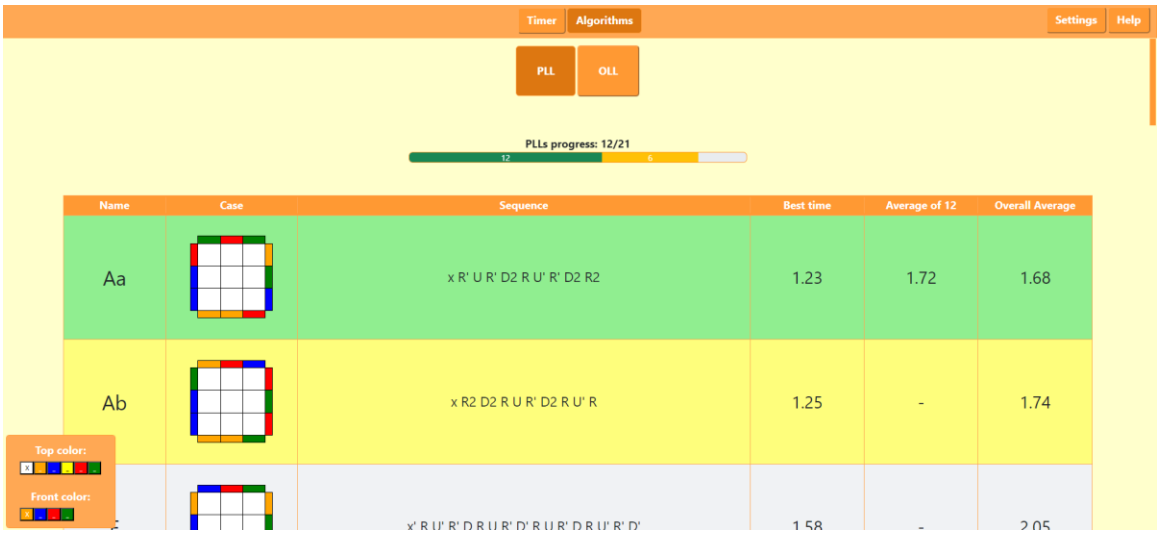
informacjami na temat wyniku (rys. 5.12, 5.13). Informacje te odpowiadają informacją zapisywanym w bazie danych narzędzia.

Dodatkowo na dole listy znajduje się przycisk pozwalający na usunięcie wszystkich wyników. Po jego klikni ęciu otwiera się wyskakujące okno z pytaniem o potwierdzenie chęci wykonania operacji usunięcia wyników (rys. 5.14). Takie rozwiązanie pozwala na uniknięcie przypadkowego usunięcia wyników.

5.7. Zakładka listy algorytmów

Drugą zakładką narzędzia jest zakładka listy algorytmów. Ekran aplikacji na tej zakładce widoczny jest na rysunku 5.15. W górnej części zakładki stopera znajdują się przyciski, za pomocą których użytkownik może zmieniać typ algorytmów, czyli wybrać algorytmy PLL lub OLL. Zaraz pod przyciskami widoczny jest pasek postępu pokazujący sumę liczby algorytmów należących do każdej z grup, dzięki któremu możliwe jest wygodne śledzenie postępów w nauce algorytmów.

Główną częścią tej zakładki jest lista algorytmów wyświetlona w formie tabelki. Kolumnami tej tabelki są kolejno:



Rysunek 5.15 Interfejs zakładki listy algorytmów

- „Name” – nazwa algorytmu
- „Case” – wizualizacja stanu kostki Rubika
- „Sequence” – sekwencja rozwiązująca
- „Best time” – najlepszy czas wykonania algorytmu
- „Average of 12” – średnia z ostatnich dwunastu wyników wykonania algorytmu
- „Overall Average” – średnia wszystkich wyników wykonania algorytmu

Zmiana stanu uczenia się danego algorytmu odbywa się poprzez kliknięcie w wizualizację kostki Rubika i jest zapisywana w narzędziu local storage. W celu poinformowania o możliwości kliknięcia tego pola, jego kolor przyciemnia się w momencie najechania kursorem myszki. Wizualizacja stanu kostki Rubika w kolumnie „Case” dotyczy stanu górnej warstwy kostki. Widoczne kwadraty wizualizacji odpowiadają górnej ścianie kostki, natomiast prostokąty znajdujące się dookoła kwadratów odpowiadają górnemu rzędowi bocznych ścian kostki.

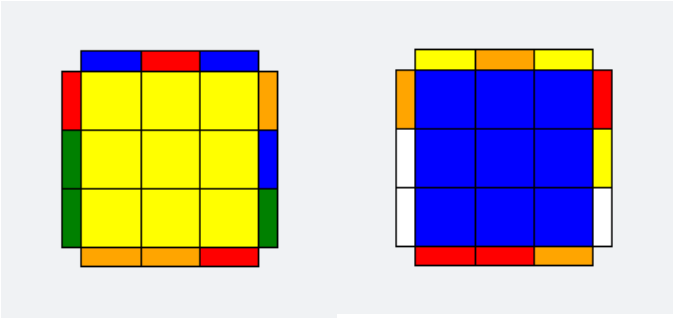
5.7.1. Zmienianie orientacji wizualizacji kostki Rubika

W lewym dolnym rogu ekranu zakładki listy algorytmów znajduje się okno pozwalające na zmianę orientacji wizualizowanych kostek Rubika na liście (rys. 5.16). Wybór orientacji sprowadza się do wyboru jednego z sześciu kolorów górnej ściany kostki oraz następnie jednego z czterech kolorów przedniej ściany kostki. Znając kolory tych ścian jesteśmy w stanie określić kolory wszystkich pozostałych ścian. Możliwe do wybrania kolory ściany przedniej dostosowują się od wybranego koloru ściany górnej, gdyż nie jest możliwe wybranie każdej kombinacji z sześciu kolorów co wynika z budowy kostki Rubika. Wybrane kolory zostają zapisane w narzędziu local storage. Przykład zmieniania orientacji widać na rysunku 5.17.

Działanie zmiany orientacji jest oparte o przypisanie liczbą od 1 do 6 odpowiedniego koloru. Wykorzystane przypisania są następujące:

- 1 – kolor biały
- 2 – kolor pomarańczowy
- 3 – kolor niebieski
- 4 – kolor żółty
- 5 – kolor czerwony
- 6 – kolor zielony

Rysunek 5.16 Okno zmiany orientacji kostki Rubika



Rysunek 5.17 Porównanie wizualizacji kostki w różnych orientacjach

Taki system oznaczeń nie jest przypadkowy a polega on na umieszczeniu kolorów kostki Rubika „po kolei". Wyjaśniając, mając przed sobą kostkę Rubika kolorem białym do przodu i kolorem zielonym do dołu, chcąc przejść od koloru białego po wszystkich kolorach zgodnie z kolejnością przypisanych im numerów wystarczy obracać kostkę w lewo i do góry na zmianę. Dzięki takiemu systemowi oznaczeń możliwe było stworzenie prostych funkcji zwracających odpowiednie kolory. Odpowiednia zmiana każdego koloru jest możliwa dzięki dwóm funkcją. Pierwsza z nich to giveOppositeColor. Jako argument przyjmuje ona numer oznaczający kolor i zwraca ona numer koloru znajdującego się na ścianie przeciwnej. Działanie funkcji można przedstawić za pomocą wzoru (1), gdzie x to dany numer koloru a y to numer koloru na przeciwnej ścianie kostki.

$$y = (x + 2)\ mod\ 6 + 1 \qquad (1)$$

Druga funkcja to giveColorOnLeft. Jako argument przyjmuje ona numer oznaczający kolor ściany górnej oraz numer oznaczający kolor ściany przedniej i zwraca numer koloru znajdującego się na ścianie lewej. Działanie funkcji polega na przejściu od koloru ściany przedniej do następnego koloru zgodnie z kolejnością numerów, z wyłączeniem numerów koloru ściany górnej oraz koloru ściany dolnej. Kierunek w jakim należy przejść zależy od parzystości numeru koloru ściany górnej. Jeżeli numer ściany górnej jest parzysty, należy przejść od numeru ściany przedniej o jeden numer w tył, natomiast jeżeli numer ściany górnej jest nieparzysty, należy przejść o jeden numer w przód. Przykładowo, trzymając kostkę kolorem białym do góry, czyli numer 1 i zielonym do przodu, czyli numer 6, przechodzimy od numeru 6 o jeden numer do przodu z pominięciem numerów 1 i 2 co daje numer 2 więc kolorem lewej ściany będzie kolor pomarańczowy.

Dzięki dwóm funkcją określającym kolor na podstawie oznaczenia numerowego możliwe jest określanie kolorów wszystkich ścian na kostce. Na rysunku 5.18 przedstawiony jest fragment kodu odpowiadający za definiowanie kolorów w obiekcie colorForFace. Numery będące atrybutami tego obiektu odpowiadają ścianą na kostce Rubika w ten sam sposób jak zostało to wykorzystane w pliku Excel.

```
const colorForFace = {
    0: 0,
    1: colorOnTop,
    2: giveOppositeColor(colorOnTop),
    3: colorOnFront,
    4: giveOppositeColor(colorOnFront),
    5: giveOppositeColor(giveColorOnLeft(colorOnTop, colorOnFront)),
    6: giveColorOnLeft(colorOnTop, colorOnFront),
};
```

Rysunek 5.18 Obiekt colorForFace

6. Uruchamianie narzędzia

W tym rozdziale zostaną przedstawione kroki, które należy podjąć w celu uruchomienia narzędzia na urządzeniu z systemem Windows. Do uruchomienia narzędzia wymagane jest połączenie z Internetem. Warto zaznaczyć, że poniższe kroki były pisane z myślą o uruchomieniu narzędzia na systemie Windows 10, jednakże w przypadku systemów starszych lub nowszych jak Windows 7 czy Windows 11 procedura ta nie powinna się bardzo różnić. Dodatkowo warto wspomnieć, iż w przyszłości narzędzie mogłoby być stale uruchomione na serwerze, a dostęp do niego byłby możliwy poprzez zaledwie adres url, jednak na ten moment narzędzie wymaga samodzielnego uruchamiania na swoim urządzeniu.

Oto kroki niezbędne do uruchomienia narzędzia:

1. Pobranie oraz zainstalowanie środowiska wykonawczego Node.js
   Ze strony https://nodejs.org/en należy pobrać rekomendowaną wersję pliku instalacyjnego Node.js klikając w przycisk „Recommended For Most Users” (numer wersji w momencie pisania - 20.11.0). Następnie należy uruchomić plik instalacyjny. Podczas instalacji należy postępować zgodnie z wyświetlanymi poleceniami. Proces instalacji wymaga zaakceptowania licencji oraz określenia miejsca zainstalowania. Wszystkie ustawienia instalacji mogą pozostać domyślne. Po wykonanej instalacji należy otworzyć wiersz polecenia (wpisując frazę „wiersz polecenia” lub „cmd” w wyszukiwarce menu Start) oraz wpisać komendę „npm -v”. Jeżeli po wpisaniu komendy zostaje wyświetlony numer wersji np. „9.6.7” oznacza to, że instalacja Node.js się powiodła.
2. Przejście do katalogu projektu
   Należy otworzyć wiersz polecenia (wpisując frazę „wiersz polecenia” lub „cmd” w wyszukiwarce menu Start) oraz przejść do katalogu zawierającego pliki projektu. Katalog projektu jest nazwany „” oraz został dołączony jako plik źródłowy w formie załącznika do owej pracy. Przejście do lokalizacji katalogu projektu odbywa się za pomocą komendy „cd <ścieżka do katalogu projektu>”, w której należy wskazać ścieżkę do katalogu projektu.
3. Zainstalowanie zależności
   Po przejściu do katalogu projektu w wierszu polecenia należy wpisać komendę „npm install”. Dzięki temu zostaną pobrane oraz zainstalowane zależności niezbędne do uruchomienia projektu. Instalacja zależności może trwać kilka minut.
4. Uruchomienie aplikacji
   Po zainstalowaniu zależności należy wpisać komendę „npm start”. Dzięki temu projekt rozpocznie uruchamianie. Uruchamianie może trwać kilka minut.

Zakończenie uruchamiania powinno zostać zakomunikowane poprzez wypisanie wiadomości „webpack compiled successfully”.

5. Otworzenie adresu aplikacji w przeglądarce
   W przypadku gdyby okno przeglądarki nie zostało otworzone automatycznie wraz ze stroną z uruchomionym narzędziem, należy otworzyć przeglądarkę internetową i przejść pod adres http://localhost:3000/. Narzędzie powinno być uruchomione w widocznym oknie przeglądarki.

7. Testy

W tym rozdziale zostaną przeprowadzone testy narzędzia. Pierwsza część rozdziału zostanie poświęcona testom manualnym w celu sprawdzenia ogólnej sprawności aplikacji oraz zweryfikowania zgodności zaimplementowanych funkcjonalności z wymaganiami. Druga część zostanie poświęcona testom użyteczności w celu sprawdzenia czy narzędzie jest odpowiednio dostosowane do oczekiwań i potrzeb użytkowników końcowych.

7.1. Testy manualne

Testy manualne to proces, w którym testerzy ręcznie sprawdzają funkcje, interakcje oraz ogólną jakość oprogramowania. Polegają one na przeprowadzaniu scenariuszy w celu zasymulowania praktycznego użytkowania oprogramowania przez użytkowników końcowych. Testy manualne służą wykrywaniu błędów oraz niedoskonałości oprogramowania, a także pozwalają zweryfikować zgodność oprogramowania z założeniami projektowymi [22].

Podczas testów manualnych narzędzie było uruchomione w przeglądarce Google Chrome, jednakże ogólne działanie narzędzia zostało dodatkowo przetestowane w przeglądarce Microsoft Edge.

Poprawność działania zaimplementowanego narzędzia została sprawdzona za pomocą następujących testów manualnych.

1. Test listy algorytmów
   Celem tego testu było sprawdzenie poprawności danych wprowadzonych za pomocą pliku Excel, w szczególności chodziło o poprawność sekwencji rozwiązującej algorytmu oraz wizualizację stanu kostki Rubika. Test polegał procedurze mieszania fizycznej kostki Rubika za pomocą sekwencji mieszającej wygenerowanej dla danego algorytmu, porównaniu stanu kostki z wizualizacją stanu kostki z listy algorytmów oraz rozwiązaniu przypadku na kostce za pomocą sekwencji rozwiązującej algorytmu. Poprawność wizualizacji kostki Rubika oraz sekwencji rozwiązującej algorytmu była potwierdzona, jeżeli stan kostki Rubika po pomieszaniu zgadzał się z wizualizacją stanu kostki z listy algorytmów oraz jeżeli po całej procedurze kostka Rubika była ułożona.
2. Test zmiany orientacji wizualizowanych stanów kostki Rubika
   Celem tego testu było sprawdzenie poprawności działania funkcji zmiany orientacji wizualizowanych stanów kostki Rubika. Test polegał na sprawdzaniu poprawności wizualizacji stanów kostki Rubika dla kilku wybranych algorytmów po każdej zmianie orientacji. W tym teście zostały zbadane wszystkie z 24 możliwych orientacji.
3. Test stopera
   Celem tego testu było sprawdzenie poprawności działania stopera oraz jego obsługi za pomocą klawiszy klawiatury. Test polegał na wielokrotnym użyciu wszystkich zaimplementowanych działań klawiszy klawiatury podczas

używania stopera. Dodatkowo zostało sprawdzone wykonanie nieprzewidzianych naciśnięć klawiszy klawiatury oraz ich kombinacji.

4. Test listy wyników oraz statystyk
   Celem tego testu było sprawdzenie poprawności działania okna statystyk oraz statystyk na liście algorytmów. Test polegał na zapisaniu do bazy danych odpowiedniej liczby wyników oraz policzeniu na ich podstawie odpowiednich statystyk w celu porównania ich ze statystykami obliczonymi przez narzędzie.
5. Test losowania algorytmu
   Celem tego testu było sprawdzenie poprawności działania losowania algorytmu z odpowiedniej grupy algorytmu na liście algorytmów. Test polegał na zmianie stanu kilku algorytmów na liście algorytmów oraz generowaniu sekwencji mieszającej. Owa procedura była powtarzana dla różnych ustawień grup algorytmów.

Wykonanie wymienionych testów manualnych pozwoliło na upewnienie się, iż określone wcześniej wymagania zostały spełnione. Dodatkowo testy manualne wykazały brak bardziej istotnych błędów narzędzia, jednakże dzięki testom manualnym udało się wykryć drobne niespójności danych niektórych algorytmów na liście algorytmów. Niespójności te zostały wyeliminowane zaraz po ich wykryciu.

7.2. Testy użyteczności

Testy użyteczności to badania mające na celu ocenę doświadczeń użytkowników podczas faktycznej interakcji z oprogramowaniem. Dzięki nim, możliwe staje się poprawienie nieintuicyjnych elementów oprogramowania czy zidentyfikowanie potencjalnych problemów, a to z kolei może prowadzić do zwiększenia satysfakcji korzystających z oprogramowania użytkowników oraz dostosowanie narzędzia do ich potrzeb [23].

Testy użyteczności narzędzia wspomagającego trening układania kostki Rubika zostały przeprowadzone na grupie ośmiu osób potrafiących ułożyć kostkę Rubika w dowolnym czasie. Badanie polegało na przeprowadzeniu z osoba badaną wywiadu dotyczącego korzystania z narzędzia. W ramach badania osobie badanej zostały zadane pytania wstępne, po czym osoba badana została poproszona o wykonanie kilku zadań dotyczących wykorzystania z narzędzia. Scenariusz przeprowadzonego badania był następujący:

1. Osobie badanej zostają zadane pytania dotyczące jej wieku, przybliżonego średniego czasu układania kostki Rubika, doświadczenia dotyczącego oficjalnych zawodów w układaniu kostki Rubika na czas oraz doświadczenia w korzystaniu z innych narzędzi wspomagających trening układania kostki Rubika.
2. Osoba badana zostaje poproszona aby zapoznać się z narzędziem i dowiedzieć się jak się z niego korzysta.
3. Osoba badana ma za zadanie pomieszać kostkę Rubika oraz zmierzyć czas układania kostki.
4. Osoba badana ma za zadanie wybrać jeden lub kilka algorytmów PLL lub OLL z listy algorytmów, po czym spróbować wykonać jedno ćwiczenie za pomocą narzędzia.
5. Osoba badana ma za zadanie dodać do jednego uzyskanego wyniku karę dwóch sekund, a następnie usunąć jeden z uzyskanych wyników.
6. Osobie badanej zostaje zadane pytanie o ogólne przemyślenia na temat korzystania z narzędzia

Podczas badania osoby badane były proszone o mówienie na głos na temat swoich przemyśleń związanych z narzędziem oraz wykonywanymi za pomocą narzędzia czynnościami. Osoby badane były również informowane o celach przeprowadzania badania.

Zadania były przedstawiane w taki sposób, aby nie informować osób badanych o sposobie ich wykonania. Dodatkowo treść zadań była formułowana w taki sposób, aby nakłonić osobę badaną do wyobrażenia sobie rzeczywistej sytuacji, w której korzysta ona z narzędzia. Kostka Rubika na potrzeby badań została osobą badanym zapewniona.

Wśród osób badanych znalazły się osoby w przedziale wiekowym od 14 do 50 lat, zarówno osoby osiągające średnie czasy ułożenia kostki Rubika poniżej 10 sekund, które regularnie jeżdżą na oficjalne zawody oraz korzystają z urządzeń typu „timer" na co dzień jak i osoby układające kostkę średnio w kilka minut, które nigdy na zawodach nie były oraz nigdy nie trenowały układania kostki Rubika za pomocą narzędzi wspomagających. Osoby osiągające lepsze czasy ułożenia kostki względem pozostałych osób badanych wykazywały się niemalże całkowitym zrozumieniem wszystkich podstawowych funkcji narzędzia. W przeciwieństwie do nich, osoby osiągające gorsze czasy ułożenia kostki miewały trudności w zrozumieniu działania niektórych funkcji, a najwięcej problemów sprawiało zrozumienie przeznaczenia listy algorytmów. Fakt ten wynika z braku wiedzy osób układających kostkę wolniej na temat metody CFOP. Z tego samego powodu osoby układające kostkę wolniej miały trudności przy wykonywaniu drugiego zadania. Ogólne przemyślenia badanych osób na temat narzędzia były w większości pozytywne. Szczególnym elementem, który zwrócił uwagę oraz pozytywne zaskoczenie kilku badanych osób była możliwość zmiany orientacji wizualizowanego stanu kostki Rubika. Dodatkowo podczas badań nie zostały wykryte żadne dodatkowe błędy narzędzia.

Wyniki przeprowadzonych badań pozwoliły na sformułowanie kilku wniosków. Narzędzie w obecnej formie jest dobrze dostosowane dla osób obeznanych z tego typu narzędziami oraz innymi aspektami dziedziny speedcubing'u, jednakże jednocześnie narzędzie nie jest dostatecznie dobrze dostosowane dla osób układających kostkę Rubika wolniej, które nie miały wcześniej styczności z tego typu narzędziami oraz nie uczęszczają na oficjalne zawody a w szczególności nie znają zaawansowanych metod układania kostki. Oznacza to, iż w celu poprawienia użyteczności narzędzia, dobrym rozwiązaniem mogłoby okazać się poszerzenie sekcji pomocy o dodatkowe wyjaśnienia czym jest narzędzie oraz jakie jest jego przeznaczenie, a także o innego rodzaju pomoce czy poradniki.

8. Zakończenie

Cel pracy można uznać za osiągnięty. Pomimo natrafiania na różnego rodzaju problemy podczas procesu implementacji, zaimplementowane narzędzie spełnia wszystkie wcześniej postawione założenia i wymagania. Dodatkowo podczas tworzenia pracy zdołałem poszerzyć swoją wiedzę oraz umiejętności tworzenia aplikacji webowych w technologii React.js.

Wykonane narzędzie zostało przygotowane w sposób umożliwiający dalszą modyfikacją oraz dodawanie nowych funkcjonalności. Z tego powodu warto wspomnieć o konceptach, które ze względu na brak czasu nie zostały wykonane, jednakże możliwe będzie dodanie owych konceptów w przyszłości. Pierwszym z nich jest możliwość połączenia narzędzia z fizycznymi urządzeniami wspomagającymi układanie kostki Rubika. Mowa tutaj o fizycznych miernikach czasu zwanych timerami, o których wspominane było już w pracy. Istnieją narzędzia pozwalające na połączenie timerów stosowanych na oficjalnych zawodach z komputerem. Dzięki temu startowanie oraz zatrzymywanie czasu odbywa się nie przez klawisze klawiatury a poprzez właśnie fizyczne urządzenie co zwiększa wygodę korzystania z narzędzia. Innym fizycznym urządzeniem mogą być inteligentne kostki Rubika, których obecnie na rynku pojawia się coraz więcej, a ich ceny stają się coraz bardziej przystępne dla wielu osób. Połączenie inteligentnej kostki Rubika z owym narzędziem pozwalałoby na zautomatyzowanie wielu czynności takich jak na przykład nakładanie kar

czy rozpoczynanie układania. Dodatkowo, możliwa byłaby weryfikacja poprawnego pomieszania układanki, co mogłoby pomóc szczególnie mniej doświadczonym w układaniu kostki Rubika osobom. Kolejnym konceptem wartym rozpatrzenia w przyszłości jest możliwość exportowania i importowania zapisanych danych dotyczących zarówno wyników jak i ustawień użytkownika. Dzięki tej funkcjonalności możliwe byłoby korzystanie z narzędzia jednocześnie na różnych urządzeniach z zachowaniem tych samych danych. Innym pomysłem wartym rozpatrzenia jest dostosowanie narzędzia do użytku na urządzeniach mobilnych. Obecnie praktycznie każda osoba posiada smart phone'a. Pomimo, że trenowanie układania kostki Rubika z pomocą narzędzia uruchomionego na komputerze lub laptopie w dużym stopniu może odwzorować warunki układania kostki Rubika na oficjalnych zawodach to możliwość uruchomienia narzędzia na urządzeniu mobilnym pozwoliłaby na wykonywanie treningu nawet w momentach kiedy dostęp do urządzeń stacjonarnych jest niemożliwy.

Bibliografia

[1] D. Singmaster, Notes on Rubik's Magic Cube, Enslow Publishers, 1981.

[2] „Pełna notacja dla kostki 3x3x3," [Online]. Available: https://rcube.pl/pelna-notacja. [Data uzyskania dostępu: 20 styczeń 2024].

[3] „How to solve the Rubik's Cube?," [Online]. Available: https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/. [Data uzyskania dostępu: 20 styczeń 2024].

[4] „Rubik's Cube solution with advanced Fridrich (CFOP) method," [Online]. Available: https://ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/. [Data uzyskania dostępu: 20 styczeń 2024].

[5] „csTimer," [Online]. Available: https://cstimer.net/. [Data uzyskania dostępu: 29 styczeń 2024].

[6] A. H. Allam, A. R. C. Hussin i H. M. Dahlan, „User experience: challenges and opportunities," *Journal of Information Systems Research and Innovation,* 2013.

[7] „Cube Timer," [Online]. Available: https://play.google.com/store/apps/details?id=br.com.mateusfiereck.cubetimer&hl=en_SG. [Data uzyskania dostępu: 20 styczeń 2024].

[8] „Metoda Fridrich - OLL," [Online]. Available: https://www.speedcubing.pl/algorytmy-do-druku/fridrich-oll-do-druku.pdf. [Data uzyskania dostępu: 20 styczeń 2024].

[9] „J Perm," [Online]. Available: https://jperm.net/. [Data uzyskania dostępu: 20 styczeń 2024].

[10] H. F. Hofmann i F. Lehner, „Requirements engineering as a success factor in software projects," *IEEE Software,* tom 18, nr 4, pp. 58-66, sierpień 2001.

[11] G. Schneider i J. P. Winters, Applying Use Cases: A Practical Guide, Pearson Education, 2001.

[12] A. Fedosejev, React.js Essentials, Packt Publishing, 2015.

[13] M. Haverbeke, Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming, No Starch Press, 2018.

[14] C. Musciano i B. Kennedy, HTML & XHTML: The Definitive Guide, O'Reilly, 2002.

[15] E. A. Meyer i E. Weyl, Cascading Style Sheets: The Definitive Guide, O'Reilly, 2019.

[16] „Introduction to Web Storage," [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/legacy/bg142799(v=vs.85). [Data uzyskania dostępu: 22 styczeń 2024].

[17] S. Kimak i J. Ellman, „The role of HTML5 IndexedDB, the past, present and future," *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST),* pp. 379-383, 2015.

[18] D. Spinellis, „Git,” *IEEE Software*, tom 29, nr 3, pp. 100-101, 2012.

[19] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2009.

[20] L. Bassett, Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON, O'Reilly Media, 2015.

[21] „God's Number is 20,” [Online]. Available: https://www.cube20.org/. [Data uzyskania dostępu: 20 styczeń 2024].

[22] M. V. Mantyla, C. Lassenius i J. Itkonen, „How Do Testers Do It? An Exploratory Study on Manual Testing Practices,” *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 494-497, 2009.

[23] J. Nielsen i H. Loanger, Optymalizacja funkcjonalności serwisów Internetowych, Gliwice: Helion, 2007.

Spis rysunków