

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno-Decyzyjne

Wieloagentowy system wyboru miejsca spotkań na urządzenia mobilne

Jakub Kierejsza

Numer albumu 261423

promotor
dr inż. Piotr Pałka

Warszawa 2017

Streszczenie

Wraz z rozwojem technologii mobilnych i ogólnej dostępności do urządzeń przenośnych, telefony zaczęły przejmować funkcjonalność dedykowanych urządzeń. Jedną z takich funkcjonalności jest nawigacja i wspomaganie decyzji. Powstał pomysł, aby wykorzystać system wieloagentowy na smartfonie do takiego rozwiązania, które pomaga podjąć decyzję grupie osób, do jakiego miejsca chcą się udać, a następnie nawigacja poprowadzi ich wszystkich do przegłosowanego lokalu.

Niniejsza praca prezentuje wspomnianą aplikację mobilną oraz przedstawia wykorzystane przez nią technologie. Została ona opracowana na system Android. Używa ona narzędzi wspomagającego rozwój aplikacji wieloagentowych JADE ze specjalną nakładką LEAP. Do wizualizacji mapy użyte zostało API Google Maps, a do wyboru miejsc API Google Places.

Jako dodatek zamieszczono instrukcję obsługi aplikacji.

Słowa kluczowe:

- smartfon
- system wieloagentowy
- Android
- nawigacja
- system wspomaganie decyzji

Abstract

Multi-Agent meeting place selection system for mobile devices

With the development of mobile technologies and general access to mobile devices phones started to take over functionalities of previously dedicated devices. One such functionality is navigation and decision support. An idea appeared to use multi-agent system on a smartphone device to implement such solution, which aids a group of people in choosing points of interest and then voting on them. Finally, this application should lead everyone in the group to selected place.

The result of this thesis is mentioned application and description of used technologies. It has been developed for Android operating system. It uses JADE software framework with LEAP add-on. Google Maps API was used for map visualization and Google Places API for place picker.

The application instruction guide has been added as an appendix.

Keywords:

- smartphone
- multi-agent system
- Android
- navigation
- decision support system



Politechnika Warszawska

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1 Wstęp.....	6
1.1 Cele.....	6
2 Aplikacje o podobnej funkcjonalności.....	7
3 Wymagania funkcjonalne i niefunkcjonalne.....	9
4 Wybór technologii.....	10
4.1 System operacyjny.....	10
4.2 Struktura wieloagentowa.....	13
4.2.1 System wieloagentowy.....	13
4.2.2 Oprogramowania wspierające implementację systemu wieloagentowego.....	13
4.2.3 JADE.....	14
4.2.4 JADE-LEAP.....	14
4.3 System nawigacji i mapa.....	14
5 Projekt aplikacji.....	16
5.1 Projekt systemu wieloagentowego.....	16
5.1.1 Metodologia Gaia.....	16
5.1.2 Role.....	16
5.1.3 Protokoły.....	19
5.2 Projekt aplikacji mobilnej.....	24
5.2.1 Stany.....	24
5.2.2 Widoki.....	25
6 Aplikacja mobilna.....	27
6.1 Wykorzystane algorytmy.....	28
6.2 Klasa agenta.....	29
6.2.1 Zachowania.....	29
6.3 Aktywność menu.....	32
6.4 Aktywność mapy.....	32
6.4 Dodatkowe klasy.....	32
7 Testy.....	34
8 Wnioski i podsumowanie.....	38
Dodatek A Obsługa aplikacji.....	40
A.1 Instrukcja dla gospodarza.....	43
A.1.1 Faza GATHER.....	43
A.1.2 Faza CHOOSE.....	44
A.1.3 Faza VOTE.....	46
A.1.4 Faza LEAD.....	47
A.2 Instrukcja dla klienta.....	48
A.2.1 Faza Gather.....	48
A.1.2 Faza CHOOSE.....	49
A.1.3 Faza VOTE.....	51

A.1.4 Faza LEAD.....	52
Bibliografia.....	53
Wykaz symboli i skrótów.....	54
Skróty.....	54
Spis rysunków.....	55
Spis tabel.....	56

1 Wstęp

Wraz z rozwojem technologii mobilnych zwiększyła się powszechna dostępność do tego typu urządzeń. Dzięki temu powstały „smartfony” - urządzenia będące telefonami, ale coraz częściej posiadające funkcjonalności osobistego komputera przenośnego. Znają one zastosowanie w wielu dziedzinach, od rozrywki po zastosowania biznesowe lub nawet jako inne narzędzia takie jak latarka, lub aparat.

Od samego początku widać było potencjał w aplikacjach mobilnych, lecz dopiero stosunkowo niedawno zaczął się ich gwałtowny rozwój, wraz z łatwiejszym dostępem do internetu przenośnego, przez co stają się coraz popularniejsze. Na rynku dostępnych jest wiele aplikacji nawigacyjnych czy aplikacji wspomagających decyzję, jednakże podczas swoich badań nie zetknąłem się dotychczas z aplikacją, która łączyłaby oba te aspekty. Z powodu braku dostępności tego typu programów uważam, że wśród młodych ludzi istnieje zapotrzebowanie na aplikację, która pomogłaby w prosty i przejrzysty sposób umówić się z grupą znajomych i ustalić za pomocą głosowania miejsce, gdzie chcieliby się spotkać.

1.1 Cele

Celem tej pracy jest zaprojektowanie i realizacja aplikacji na urządzenia mobilne, która ma za zadanie doprowadzić grupę ludzi do wybranego wspólnie celu. Aplikacja to powinna dać możliwość stworzenia grupy i wybrania interesujących punktów użyteczności publicznej. Następnie ma nastąpić głosowanie na wybrane wcześniej punkty. Na końcu aplikacja ma poprowadzić każdego z grupy do przegłosowanego miejsca.

2 Aplikacje o podobnej funkcjonalności

Istnieje wiele narzędzi służących do nawigacji, które posiadają także bazę punktów użyteczności publicznej. Jednym z nich jest narzędzie Google Maps [1], którego API zostało użyte w aplikacji, opisywanej w tej pracy. Narzędzie to zdominowało cały rynek map i różnych systemów nawigacji. Ważne jest również to, że jest ciągle rozwijane i rozbudowywane przez ogromny zespół programistów. Dzięki takiemu zarządzaniu aplikacją, na bieżąco opracowywane są nowe funkcjonalności i dodatki. Google Maps także posiada bardzo dużą liczbę opcji pozwalających zobaczyć między innymi natężenie ruchu ulicznego, trasy autobusowe lub rowerowe czy nawet zdjęcia okolicy wybranego miejsca.

Drugim przykładem takiego narzędzia służącego do nawigacji jest OpenStreetMap [2]. Biorąc pod uwagę jego funkcjonalność, w działaniu przypomina wcześniej opisywane Google Maps. Mimo wielu podobieństw, OpenStreetMap charakteryzuje się jawnym dostępem do danych mapy i możliwością ich samodzielnego edytowania, czego w Google Maps nie znajdziemy. Dzięki takim opcjom, narzędzie to pozwala na szybką reakcję użytkowników na gwałtowne oraz niespodziewane zmiany, które można w każdej chwili umieścić w OpenStreetMap. Mimo że dostęp do samodzielnego edytowania map niesie również ryzyko pojawienia się nieprawidłowych danych, to społeczność tego narzędzia regularnie sprawdza oraz kontroluje każdą wprowadzoną zmianę. OpenStreetMap nie posiada jednak dodatkowych funkcjonalności, takich jak natężenie ruchu ulicznego czy zrobić zdjęcia okolicy wybranego miejsca, z których popularny jest Google Maps.

Trzecią aplikacją służącą do nawigacji jest aplikacja Jakdojade [3]. Korzysta ona z danych OpenStreetMaps. W przeciwieństwie do wymienionych powyżej aplikacji obejmuje ona obszar tylko w większych miastach Polski. Baza punktów użyteczności publicznej jest ograniczona wyłącznie do przystanków autobusowych i tramwajowych oraz stacji metra i stacji kolejowych. Nawigacja, która zaprogramowana jest, aby optymalizować czas podróży, odbywa się tylko przy pomocy środków komunikacji miejskiej oraz pieszo, która optymalizuje czas podróży.

Powyższe aplikacje nie spełniają postawionych wymagań funkcjonalnych. Pozwalają na nawigację tylko dla jednej osoby, bez żadnej formy interakcji pomiędzy użytkownikami, którzy chcą się spotkać. Nie wspomagają też podejmowania decyzji co do wyboru miejsca docelowego ani żadnej formy głosowania na wybrane miejsca.

Mimo że systemy wspomagania decyzji są bardzo popularne, to istnieje bardzo niewiele rozwiązań bazujących na systemach wieloagentowych. Jednym z nich jest struktura MASST (Multi-Agent Decision Support System for Stock Trading) zaproponowana przez Luo *et al.*[4].

Została ona stworzona głównie do wspomagania handlu akcjami, jednak może zostać użyta też w innych dziedzinach. MASST jest to system o ścisłe współpracujących agentach, z których każdy ma wyspecjalizowane wiedzę i funkcjonalności. Stanowi on fazę pośrednią pomiędzy stroną żądającą informacji (np. inwestorem na Giełdzie Papierów Wartościowych) a stroną dostarczającą informację (np. internet).

Struktura MASST została stworzona jako system wspomagania decyzji, ale jest zbyt rozbudowana, aby użyć jej przy aplikacji, będącej tematem tej pracy. Duża ilość agentów, jakiej wymaga, mocno obciążałaby urządzenie mobilne, lub wymagałaby dodatkowych agentów znajdujących się na serwerze. Funkcjonalność, jaką spełnia ten system, nie wymaga również opcji głosowania, więc nie zostało ono zaimplementowane.

3 Wymagania funkcjonalne i niefunkcjonalne

Podstawowym celem niniejszej pracy było zaprojektowanie aplikacji mobilnej. Podczas pracy nad projektem, okazało się, że struktura wieloagentowa Jade posiada parę błędów i ograniczeń, które nie były opisane na oficjalnej stronie. Sprawiło to, że obejście napotkanych trudności stało się drugorzędnym celem niniejszej pracy. Jednym z większych problemów sprawiło, że zaistniała potrzeba stworzenia zewnętrznego serwera, który służyłby aplikacji jedynie do komunikacji pomiędzy agentami.

Wymagania funkcjonalne. Aplikacja powinna mieć:

- wygodny i prosty interfejs użytkownika,
- tryb założyciela grupy i klienta
- możliwość nawiązywania połączenia z innymi użytkownikami z grupy poprzez serwer,
- możliwość rozwiązywania połączenia z grupą i serwerem,
- możliwość wyświetlania położenia pozostałych członków grupy,
- możliwość wyznaczania proponowanej okolicy spotkania,
- możliwość wyświetlania i dodawania punktów użyteczności publicznej,
- funkcję głosowania na punkty użyteczności publicznej,
- funkcję nawigacji do wybranego punktu.

Wymagania niefunkcjonalne:

- wykorzystanie systemu wieloagentowego spełniającego specyfikacje FIPA [3]

4 Wybór technologii

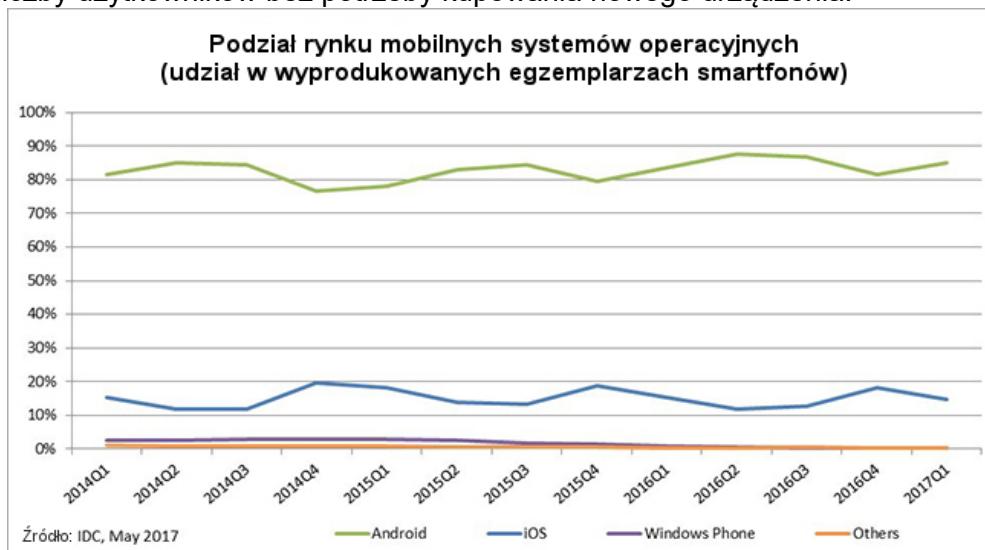
Na samym początku prac związanych z tworzeniem projektu konieczny był wybór technologii, z których użyciem napisana będzie aplikacja. Najważniejszą decyzją był system operacyjny. Następnie należało się zastanowić nad strukturą wieloagentową, w której aplikacja będzie powstawała, jak i nad systemem nawigacji, mapą oraz wyświetlaniem punktów użyteczności publicznej.

4.1 System operacyjny

W ramach niniejszej pracy aplikacja miała zostać na jeden z wybranych przez mnie systemów operacyjnych. Głównym kryterium, którym się kierowałem przy wyborze systemu, stanowiła jego popularność wśród użytkowników. Wybór został przeze mnie zawężony do dwóch najpopularniejszych obecnie systemów operacyjnych: Android oraz iOS. Obydwa spełniają podstawowe wymagania funkcjonalne, jak i niefunkcjonalne.

Poniżej przedstawione będą argumenty wykorzystane przy wyborze systemu operacyjnego:

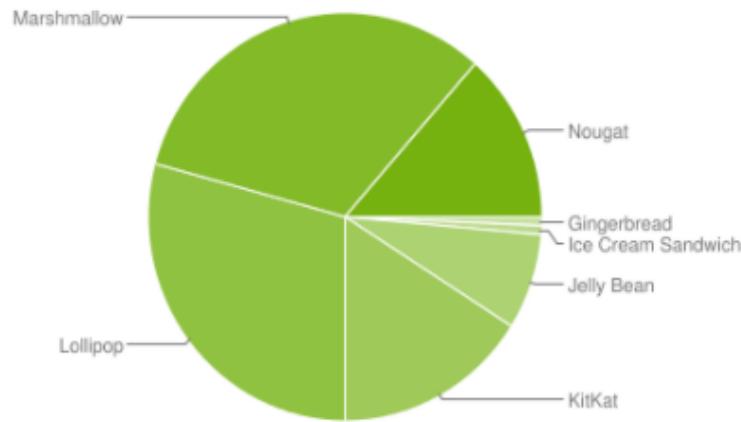
1. Popularność systemu Android (na rysunku 1 zaznaczona zieloną linią) znacznie przewyższa popularność pozostałych systemów operacyjnych. Różnica ta sięga aż do 75% całego rynku smartfonów. Dzięki temu aplikacja ma szansę trafić do większej liczby użytkowników bez potrzeby kupowania nowego urządzenia.



Rysunek 1. Podział rynku smartfonów ze względu na zainstalowany system operacyjny. Oś pozioma przedstawia umieszczone kolejno kwartały.

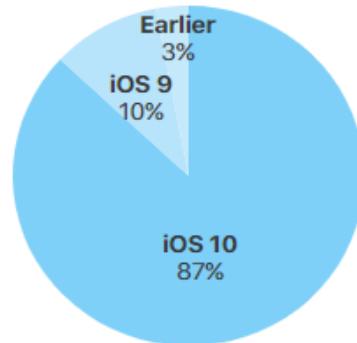
Źródło elementów graficznych: [4].

2. Oba systemy operacyjne mają swój dedykowany język programowania (Java dla Androida, a Swift[5] dla iOS), lecz wspierają też inne języki takie jak Python, Javascript i wiele więcej. Posiadają też dedykowane zintegrowane środowiska programistyczne, które ułatwiają rozwój aplikacji.
3. Fragmentacja systemu Android (rysunek 2) jest znacznie wyższa w porównaniu z iOS (rysunek 3). Może to stwarzać problemy z dostępnością aplikacji, jeśli zostałyby wykorzystane funkcjonalności, które nie są kompatybilne wstecz lub w przód.



Rysunek 2. Podział urządzeń z systemem Android ze względu na zastosowane w nich wersje tego systemu. Dane z 8 sierpnia 2017 r.

Źródło elementów graficznych: [6]



Rysunek 3. Podział urządzeń z systemem iOS ze względu na zastosowane w nich wersje tego systemu. Dane z 28 czerwca 2017 r.

Źródło elementów graficznych: [7]

Na podstawie powyższych argumentów wybrałem system operacyjny Android. Należało też zdecydować jaką najstarszą wersję systemu będzie wspierała aplikacja. Wybierając

starszą wersję systemu, powiększa się potencjalną liczbę użytkowników, ale traci się funkcjonalności dodane w nowszych wersjach. Dlatego też jako najstarszą wspieraną wersję systemu Android wybrałem 4.4. Zapewnia ona działanie na około 91% urządzeń z tym systemem. Nie wybrano poprzednich wersji, ponieważ stanowią one już bardzo małą część całego rynku, a w przyszłości liczba ta ciągle będzie się zmniejszać.

4.2 Struktura wieloagentowa

4.2.1 System wieloagentowy

System wieloagentowy jest to system składający się z autonomicznych agentów komunikujących się ze sobą. Każdy agent posiada cele oraz informacje, które nie zawsze muszą być identyczne. Cały system ma swój nadzędny cel, który realizuje poprzez indywidualne działania i interakcje agentów.

4.2.2 Oprogramowania wspierające implementację systemu wieloagentowego

Struktura wieloagentowa jest niezbędna, aby spełnić wymagania niefunkcjonalne projektu. Postanowiłem wybrać jedno z gotowych narzędzi wspomagających implementację systemów wieloagentowych, aby ułatwić powstawanie projektu. Głównym zamierzeniem było to, aby spełniał wszystkie specyfikacje i standardy FIPA (tabela 2 strona 9).

Oprogramowanie	Licencja	Język programowania	System operacyjny	Wsparcie techniczne
Agent Service [8]	Nie podano	.NET, Mono	Windows	Dokumentacja, API, przykłady.
DALI [9]	Open Source, darmowe dla aplikacji uczelnianych, Apache Public License 2.0	Język DALI	Windows XP; Windows 7; Windows 8; Linux; Mac OS X	FAQ, samouczki, prezentacje, dokumentacja, przykłady, publiczne forum.
JADE [10]	GPLv2	Java	Każda platforma obsługująca Java	FAQ, lista mailingowa, samouczki, API, dokumentacja.
JIAC [11]	Licencja Apache 2.0	JAVA	Każda platforma obsługująca Java	Lista mailingowa, samouczki, API, dokumentacja.

Tabela 1. Porównanie oprogramowań do programowania agentowego zgodnych z FIPA

Spośród rozpatrywanych w Tabeli 2 oprogramowań służące do programowania agentowego DALI i Agent Service nie są wspierane na systemie operacyjnym Android. Z tego powodu oba te oprogramowania zostały wykluczone i tym samym nieużyte w projekcie. Następnie wybór został zawężony do JADE'a i JIAC'a, gdzie obie opcje oferowały bardzo podobne funkcjonalności. Okazało się jednak, że spośród nich to JADE był częściej aktualizowany, jednocześnie posiadając o wiele bardziej przejrzystą i kompletną dokumentację. Miał też o wiele więcej przykładów oraz specjalną nakładkę na oprogramowanie, które dodaje funkcje potrzebne do komunikacji z Androidem. Dostępność takiej nakładki znacząco zaważyła na wyborze JADE.

4.2.3 JADE

Jade jest to oprogramowanie napisane w pełni w Javie, które znacznie upraszcza implementację systemów wieloagentowych poprzez pośrednią warstwę oprogramowania zgodną ze wszystkimi specyfikacjami FIPA. Posiada również graficzne narzędzia pomagające w debugowaniu aplikacji oraz pozwalające na śledzenie komunikacji pomiędzy agentami.

4.2.4 JADE-LEAP

JADE-LEAP jest nakładką do platformy JADE, która pozwala na umieszczanie JADE'owych agentów na różnych platformach obsługujących Java. Należy również zaznaczyć, że wspomniana nakładka udostępnia API, które są identyczne dla JADE, co umożliwia użytkownikom bezproblemowe uruchomienie agenta w tej platformie. Dzięki temu nie trzeba w żaden sposób na nowo dostosowywać JADE-LEAP.

4.3 System nawigacji i mapa

System nawigacji w tego typu aplikacji jest bardzo istotny, ponieważ stanowi jej najbardziej widoczną część. W sytuacji, w której nawigacja nie funkcjonowałaby w odpowiedni sposób, okazałoby się, że reszta aplikacji nie miałaby sensu. Z tego powodu postanowiłem wybrać spośród gotowych i sprawdzonych narzędzi używanych przez bardzo dużą liczbę aplikacji. Dwa najpopularniejsze narzędzia, które zawierają w sobie mapę i system nawigacji to Google Maps i OpenStreetMap [12]. Oba te narzędzia posiadają swoje API pozwalające w łatwy sposób użyć ich na telefonach z systemem Android.

Największą różnicą pomiędzy tymi dwoma rozwiązaniami jest dostęp do nieprzetworzonych danych mapy. Google Maps daje dostęp tylko do serwisów, które korzystają z nieprzetworzonych danych, podczas gdy OpenStreetMap (w skrócie OSM)

umożliwia tworzenie własnych serwisów na podstawie właśnie tych danych. OSM pozwala także na wgrywanie zmian, które później są weryfikowane. Taka polityka umożliwia bardzo szybką reakcję społeczności OSM na jakiekolwiek zmiany w sklepach, przepustowości ruchu na niektórych odcinkach dróg z powodu korków lub nawet większych wypadkach drogowych. Szczególnie zauważalne jest to w miejscowościach na świecie z mniej rozwiniętą infrastrukturą aglomeracyjnymi, gdzie oznaczane są ścieżki i szlaki. Niestety takie rozwiązania niosą ze sobą ryzyko pojawienia się błędnych danych, nawet jeśli są one sprawdzane regularnie. W przeciwieństwie do OSM, Google Maps nie daje dostępu do nieprzetworzonych danych mapy. Z tego powodu, mimo że dane na mapie są poprawne, to w miejscowościach o mniejszej populacji aktualizacja miejsc następuje o wiele rzadziej.

Oba wymienione wcześniej narzędzia są darmowe. OpenStreetMap jest na licencji „Open Data Commons Open Database License” co pozwala na nielimitowany dostęp do danych i pozostałych serwisów OSM. Google Maps, poza podstawową funkcjonalnością oferuje jeszcze wiele rozmaitych gotowych serwisów, gdzie większość posiada płatną licencję. Jednym z dodatkowych serwisów, który ma okrojoną darmową wersję jest serwis Google Places, który pozwala na 150 tysięcy zapytań do bazy.

Zarówno OSM, jak i Google Maps ma bardzo dobrze udokumentowane API, jednak Google Maps ma o wiele więcej przykładów i samouczków dostępnych w internecie. Istnieje też o wiele większa społeczność, która odpowiada na nurtujące pytania.

Na podstawie wymienionych wyżej porównań wywnioskowałem, że Google Maps będzie lepszym rozwiązaniem dla mojej aplikacji. To narzędzie posiada serwis Google Places, który ma wystarczający limit dla tego rodzaju aplikacji. W przypadku OSM taki serwis musiałbym stworzyć sam od nowa, co zajęłoby dużo czasu. Ilość przykładów i samouczków też była mocnym argumentem za Google Maps, ponieważ znacznie przyspieszyła rozwiązywanie napotkanych po drodze problemów.

5 Projekt aplikacji

Przed stworzeniem samej aplikacji konieczny był dobrze przemyślany projekt. W niniejszej pracy potrzebne były dwa projekty: systemu wieloagentowego i aplikacji.

5.1 Projekt systemu wieloagentowego

5.1.1 Metodologia Gaia

Do zaprojektowania systemu wieloagentowego użyłem metodologii Gaia [13]. Pozwala ona na utrzymanie abstrakcji oraz pokazanie zależności i interakcji między rolami. Cechuje się ona brakiem fazy formułowania wymagań, brakiem szczególnych technik modelowania i notacji. Również nie odnosi się ona do realizacji, ponieważ jest ona zależna od implementacji.

5.1.2 Role

Rolą w metodologii Gaia nazywamy opis abstrakcyjnej funkcjonalności, który w prawdziwym życiu przypomina stanowisko w firmie. Opis każdej roli składa się z nazwy, krótkiego opisu działania, wymienionych protokołów i aktywności, listy pozwoleń oraz obowiązków.

Zaproponowany przeze mnie system wieloagentowy składa się z pięciu ról: nadawcy, odbiorcy, założyciela, nawigatora i mediatora. Poniżej zamieściłem opis poszczególnych ról.

Rola nadawcy odpowiada za poinformowanie reszty grupy lub gospodarza grupy o zmianie pozycji urządzenia. Taka aktualizacja dzieje się nie rzadziej niż raz na dziesięć sekund.

Rola: Nadawca
Opis: Przesyła grupie swoją aktualną pozycję co 10 sekund
Protokoły i aktywności: <u>OdczytajPozycję</u> , <u>OdczytajGrupę</u> , <u>WyślijPozycję</u>
Pozwolenia:
reads ObecnaPozycja // pozycja użytkownika na mapie
Obowiązki:
Żywotne:
Nadawca = <u>OdczytajPozycję</u> . <u>OdczytajGrupę</u> . <u>WyślijPozycję</u>
Bezpieczeństwa: true

Tabela 2. Opis roli Nadawcy

Rola odbiorcy odpowiada za odebranie i przekazanie przychodzących informacji do urządzenia, aby mogły one zostać wyświetlane.

Rola: Odbiorca
Opis: Odbiera informacje
Protokoły i aktywności: <u>OdbierzNMS</u> , <u>OdbierzLokal</u> , <u>OdbierzPozycje</u> , <u>PrzekażInformacje</u>
Pozwolenia:
Obowiązki:
Żywotne: Odbiorca = (<u>OdbierzNMS</u> <u>OdbierzLokal</u> <u>OdbierzPozycje</u>) . <u>PrzekażInformacje</u>
Bezpieczeństwa: true

Tabela 3. Opis roli Odbiorcy

Rola założyciela odpowiada za stworzenie i zarejestrowanie grupy. Tak stworzona grupa pozwala innym użytkownikom do niej dołączyć.

Rola: Założyciel
Opis: Rejestruje grupę
Protokoły i aktywności: <u>ZarejestrujGrupę</u>
Pozwolenia:
Obowiązki:
Żywotne: Założyciel = <u>ZarejestrujGrupę</u>
Bezpieczeństwa: true

Tabela 4. Opis roli Założyciela

Rola nawigatora jest odpowiedzialna za wyznaczenie najlepszego miejsca spotkania. Odczytuje ona pozycje wszystkich członków grupy i oblicza proponowany punkt spotkania.

Rola: Nawigator
Opis: Znajduje najlepsze miejsce spotkania (NMS)
Protokoły i aktywności: <u>OdczytajGrupę</u> , <u>OdbierzPozycję</u> , <u>OdczytajPozycję</u> , WyznaczNMS, <u>WyślijNMS</u>
Pozwolenia:
Obowiązki:
Żywotne: Nawigator = <u>OdczytajGrupę</u> . (<u>OdczytajPozycję</u> . <u>OdbierzPozycję+</u>)+ . WyznaczNMS . <u>WyślijNMS</u>
Bezpieczeństwa: true

Tabela 5. Opis roli Nawigatora

Rola mediatora odpowiada za głosowanie. Odbiera wszystkie głosy wysłane przez pozostałych członków grupy, a następnie wybiera przegłosowany lokal i rozsyła jego pozycje do wszystkich.

Rola: Mediator
Opis: Odpowiada za głosowanie na lokale
Protokoły i aktywności: <u>OdczytajGrupę</u> , <u>OdbierzGłosy</u> , <u>OdczytajGłosy</u> , WybierzLokal, <u>WyślijLokal</u>
Pozwolenia:
Obowiązki:
Żywotne: Mediator = <u>OdczytajGrupę</u> . (<u>OdczytajGłosy</u> . <u>OdbierzGłosy+</u>)+ . WybierzLokal . <u>WyślijLokal</u>
Bezpieczeństwa: true

Tabela 6. Opis roli Mediatory

5.1.3 Protokoły

Protokoły są to aktywności, które wymagają interakcji z innymi agentami.

Nadawca

Protokół „Odczytaj grupę” służy do uzyskania listy osób aktualnie będących w grupie. Jedyne co ten protokół potrzebuje to nazwę założyciela grupy.

Protokół: Odczytaj grupę		
Inicjator: Nadawca	Respondent: Założyciel	Dane wejściowe: Nazwa założyciela
Opis: Zapytaj i odczytaj listę osób w grupie.		Dane wyjściowe: Lista osób z grupy

Tabela 7. Opis protokołu „Odczytaj grupę” dla roli nadawcy

Protokół „Wyślij pozycję” służy do powiadomienia pozostałych członków grupy o zmianie położenia urządzenia.

Protokół: Wyślij pozycję		
Inicjator: Nadawca	Respondent: Odbiorcy	Dane wejściowe: Grupa, Pozycja
Opis: Wyślij swoją pozycję do wszystkich osób z grupy		Dane wyjściowe: Wysłanie swojej pozycji do wszystkich z grupy

Tabela 8. Opis protokołu „Wyślij pozycję” dla roli nadawcy

Protokół „Odczytaj pozycję” komunikuje się z urządzeniem, na którym agent działa, abytrzymać informację o aktualnym położeniu urządzenia.

Protokół: Odczytaj pozycję		
Inicjator: Nadawca	Respondent: Telefon	Dane wejściowe: -
Opis: Odczytuje pozycję GPS telefonu		Dane wyjściowe: Pozycja

Tabela 9. Opis protokołu „Odczytaj pozycję” dla roli nadawcy

Nawigator

Protokół „Odczytaj grupę” dla roli nawigatora działa identycznie jak protokół o tej samej nazwie dla roli nadawcy.

Protokół: Odczytaj grupę		
Iinicjator: Nawigator	Respondent: Założyciel	Dane wejściowe: Nazwa założyciela
Opis: Zapytaj i odczytaj listę osób w grupie.		Dane wyjściowe: Lista osób z grupy

Tabela 10. Opis protokołu „Odczytaj grupę” dla roli nawigatora

Protokół „Odczytaj pozycję” działa analogicznie do protokołu o tej samej nazwie dla roli nadawcy.

Protokół: Odczytaj pozycję		
Iinicjator: Nawigator	Respondent: Telefon	Dane wejściowe: -
Opis: Odczytuje pozycję GPS telefonu		Dane wyjściowe: Pozycja

Tabela 11. Opis protokołu „Odczytaj pozycję” dla roli nawigatora

Protokół „Odbierz pozycję” czeka i odczytuje informacje o pozycji innych urządzeń przesłanej przez rolę nadawcy.

Protokół: Odbierz pozycję		
Iinicjator: Nawigator	Respondent: Nadawca	Dane wejściowe: -
Opis: Odczytuje pozycję GPS przysłane przez nadawców		Dane wyjściowe: Lista pozycji pozostałych osób z grupy

Tabela 12. Opis protokołu „Odbierz pozycję” dla roli nawigatora

Protokół „WyślijNMS” rozsyła wyliczone przez rolę nawigatora najlepsze miejsce spotkania do wszystkich osób z grupy.

Protokół: WyślijNMS		
Inicjator: Nawigator	Respondent: Odbiorcy	Dane wejściowe: Lista osób z grupy
Opis: Wysyła informacje o najlepszym miejscu spotkania do wszystkich osób z grupy		Dane wyjściowe: -

Tabela 13. Opis protokołu „WyślijNMS” dla roli nawigatora

Mediator

Protokół „Odczytaj grupę” działa analogicznie do protokołu o tej samej nazwie opisany w roli nadawcy.

Protokół: Odczytaj grupę		
Inicjator: Mediator	Respondent: Założyciel	Dane wejściowe: Nazwa założyciel
Opis: Zapytaj i odczytaj listę osób w grupie.		Dane wyjściowe: Lista osób z grupy

Tabela 14. Opis protokołu „Odczytaj grupę” dla roli mediatora

Protokół „OdbierzGłosy” odbiera informacje o głosach oddanych na pozostałych urządzeniach. Ten protokół nie musi czekać aż wszyscy oddadzą głosy, wystarczy odpowiednia liczba głosów potrzebna do wyznaczenia zwycięskiego miejsca

Protokół: OdbierzGłosy		
Inicjator: Mediator	Respondent: Nadawca	Dane wejściowe: Lista osób z grupy
Opis: Odczytuje głosy przesypane przez nadawców		Dane wyjściowe: Lista głosów

Tabela. 15 Opis protokołu „OdbierzGłosy” dla roli mediatora

Protokół „OdczytajGłosy” odbiera głosy wybrane na aktualnym urządzeniu, na którym działa rola mediatora.

Protokół: OdczytajGłosy		
Inicjator: Mediator	Respondent: Telefon	Dane wejściowe: -
Opis: Odczytuje głosy użytkownika telefonu		Dane wyjściowe: Lista głosów

Tabela 16. Opis protokołu „OdczytajGłosy” dla roli mediatora

Protokół „WyślijLokal” odpowiada za rozesłanie informacji o zwycięskim punkcie do wszystkich członków grupy.

Protokół: WyślijLokal		
Inicjator: Mediator	Respondent: Odbiorcy	Dane wejściowe: Lista osób z grupy
Opis: Wysyła lokal, który został wybrany przez mediatora na podstawie głosów		Dane wyjściowe: -

Tabela 17. Opis protokołu „WyślijLoka” dla roli mediatora

Odbiorca

Protokół „OdbierzNMS” odpowiada za odczytanie wiadomości o wybranym najlepszym miejscu spotkania.

Protokół: OdbierzNMS		
Inicjator: Odbiorca	Respondent: Nawigator	Dane wejściowe: -
Opis: Odbiera najlepsze miejsce spotkania		Dane wyjściowe: NMS

Tabela 18. Opis protokołu „OdbierzNMS” dla roli odbiorcy

Protokół „OdbierzLokal” odpowiada za odczytanie wiadomości o przegłosowanym miejscu.

Protokół: OdbierzLokal		
Inicjator: Odbiorca	Respondent: Mediator	Dane wejściowe: -
Opis: Odbiera wybrany lokal		Dane wyjściowe: Wybrany lokal

Tabela 19. Opis protokołu „OdbierzLokal” dla roli odbiorcy

Protokół „OdbierzPozycję” odpowiada za odczytanie wiadomości zawierającej pozycję urządzenia zgłaszającego.

Protokół: OdbierzPozycje		
Inicjator: Odbiorca	Respondent: Nadawca	Dane wejściowe: -
Opis: Odbiera pozycje pozostałych osób		Dane wyjściowe: Lista pozycji pozostałych osób z grupy

Tabela 20. Opis protokołu „OdbierzPozycje” dla roli odbiorcy

Protokół „PrzekażInformacje” przekazuje odebrane informacje do urządzenia

Protokół: PrzekażInformacje		
Inicjator: Odbiorca	Respondent: Telefon	Dane wejściowe: -
Opis: Przekazuje wszystkie informacje do telefonu		Dane wyjściowe: -

Tabela 21. Opis protokołu „PrzekażInformacje” dla roli odbiorcy

Założyciel

Protokół „ZarejestrujGrupę” komunikuje się z serwerem i rejestruje tam grupę, pozwalając tym samym pozostałym roliom na dołączenie do niej.

Protokół: ZarejestrujGrupę		
Inicjator: Założyciel	Respondent: Serwer	Dane wejściowe: -
Opis: Rejestruje grupę na serwerze		Dane wyjściowe: -

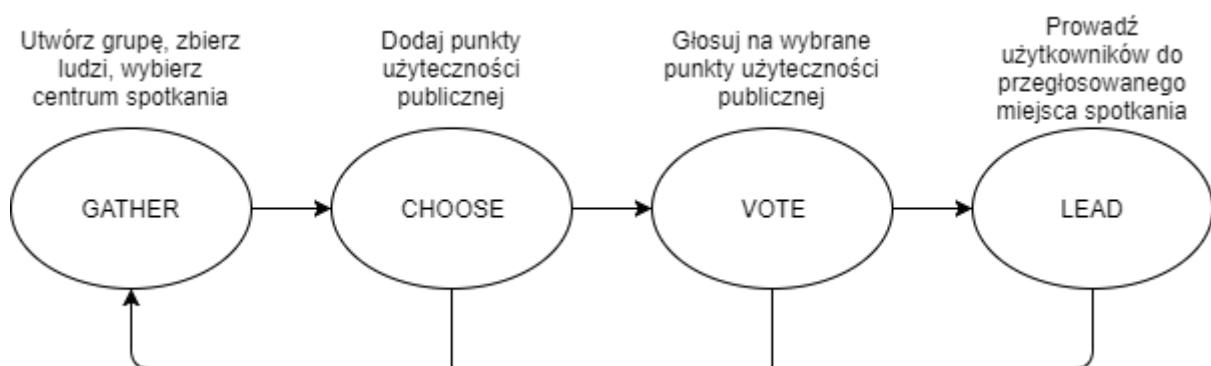
Tabela 22. Opis protokołu „ZarejestrujGrupę” dla roli założyciela

5.2 Projekt aplikacji mobilnej

Po zaprojektowaniu systemu wieloagentowego zająłem się projektowaniem aplikacji mobilnej. Na samym początku postanowiłem zaprojektować stany aplikacji, a następnie liczbę widoków potrzebnych do wyświetlenia tych stanów.

5.2.1 Stany

Po przeanalizowaniu potrzeb aplikacji zdecydowałem się na cztery stany, w których aplikacja może się znajdować: GATHER, CHOOSE, VOTE, LEAD¹.



Rysunek 4. Schemat blokowy stanów aplikacji

Przejście do każdego kolejnego stanu kontroluje założyciel grupy. Oczywiście każdy może opuścić grupę w każdym momencie.

Stan GATHER (zbierania grupy) pozwala na stworzenie grupy lub dołączenie do już istniejącej. W tym stanie można również przesuwać proponowane miejsce spotkania.

W stanie CHOOSE (wybierania miejsc) każdy w grupie może wyszukać interesujące go miejsce spotkania i dodać to miejsce do mapy, aby wszyscy mogli je zobaczyć. Nie ma limitu, ile miejsc można dodać.

Po przejściu do stanu VOTE (głosowania) każda osoba z grupy ma ograniczoną liczbę głosów, którą może wydać na wybrane miejsce. Na każdy punkt można głosować wielokrotnie, aby oddać swoje preferencje. Rezultat głosowania widzi tylko założyciel grupy i to on ma decydujący głos w podjęciu decyzji, dokąd grupa będzie prowadzona.

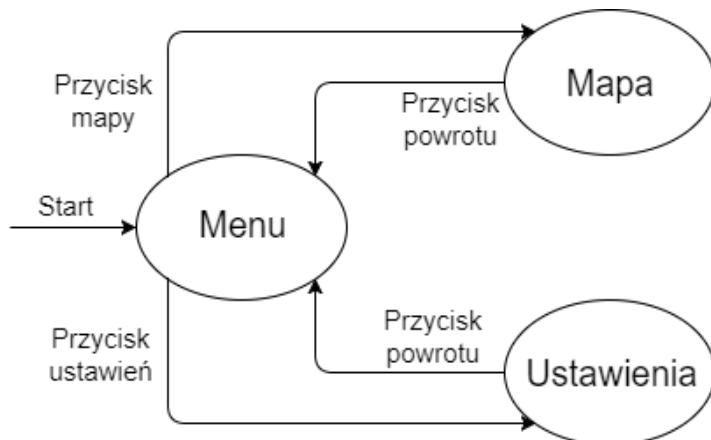
Stan LEAD (nawigacji) jest odpowiedzialny za doprowadzenie użytkownika urządzenia do miejsca spotkania. Jego zadaniem jest wyświetlenie na mapie najlepszej ścieżki, po której można dojść do wybranego celu. Po dotarciu użytkownika do celu stan się nie zmienia już dalej, ponieważ użytkownik może chcieć dalej obserwować pozostalych członków

¹ Podane nazwy są zgodne z nazwami w aplikacji

grupy. W momencie, gdy już wszyscy dotrą na miejsce, nie ma powodu, dlaczego aplikacja miałaby zmienić stan na inny, ponieważ spełniła już swoją funkcję i może zostać zamknięta.

5.2.2 Widoki

Pierwszym instynktem były trzy widoki: widok menu, widok ustawień i widok mapy. Jest to standardowy wybór w wielu aplikacjach, gdzie jest jedna główna aktywność, która zawiera w sobie większość funkcjonalności aplikacji.



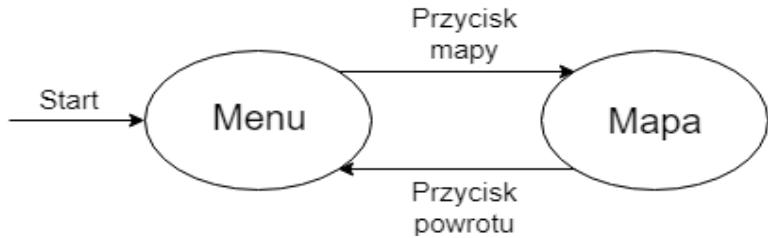
Rysunek 5. Pierwotny diagram widoków

Widok menu miał zawierać podstawowe dwa przyciski pozwalające przejść do pozostałych widoków oraz guzik uruchamiający agenta. Jedyną rolą tej aktywności jest nawigacja użytkownika po aplikacji.

Widok ustawień miał zawierać pole na wpisanie nazwy użytkownika oraz pole na adres sieciowy założyciela grupy. Niewypełniony adres założyciela oznacza intencję bycia gospodarzem. W innym przypadku należy wypełnić to pole odpowiednimi danymi, co ustawi aplikacje w tryb klienta.

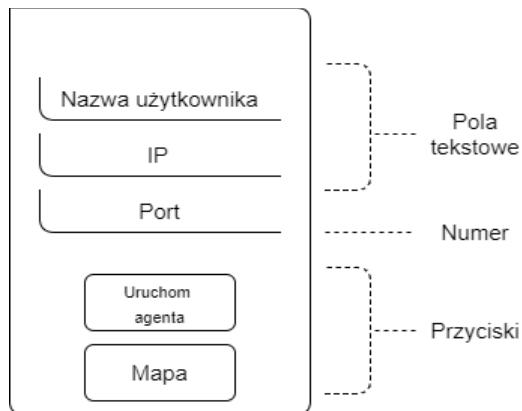
Widok mapy, który jest głównym widokiem aplikacji, miał zawierać mapę, położenie wszystkich osób z grupy oraz pinezki pokazujące wybrane miejsca. Dodatkowo dla gospodarza miały być widoczne guziki pozwalające przechodzić do kolejnych stanów aplikacji. Jest to widok, w którym użytkownicy spędzą większość czasu podczas korzystania z aplikacji.

Po głębszym przeanalizowaniu postanowiłem połączyć widok menu i ustawień (rysunek 4), ponieważ do tak małej liczby opcji do ustawienia nie był potrzebny dedykowany widok.



Rysunek 6. Końcowy diagram widoków

Poza zaprojektowaniem co w widokach ma być i ich interakcji między sobą zaprojektowałem jeszcze wstępny szkic wyglądu poszczególnych widoków.



Rysunek 7. Model widoku menu



Rysunek 8. Model widoku mapy

Źródło obrazka: Zrzut ekranu z Google Maps

6 Aplikacja mobilna

Wybrany systemu operacyjnego nie narzuca konkretnej technologii, w jakiej trzeba stworzyć program. Android pozwala na pełną implementację aplikacji w językach takich jak:

- Java,
- C,
- C++,
- języki skryptowe (Perl, Jruby, Python, lua, javascript).

Java była pierwszym językiem oficjalnie wspieranym do budowania i dystrybuowania aplikacji na Androidzie. Posiada ona najwięcej materiałów, jak i bibliotek, których można użyć podczas rozwijania programu.

C i C++ są to języki, których przy dzięki Android Native Development Kit (Android NDK) [14]. Te języki nie są zalecane do tworzenia całych aplikacji, ale można ich użyć do stworzenia własnych bibliotek.

Języki skryptowe też są wspierane, ale większość wersji systemu android jeszcze nie posiada wbudowanej biblioteki, która je obsługuje.

Biorąc pod uwagę powyższe argumenty Java wydaje się najlepszym rozwiązaniem. Dodatkowo jako jedyny z tych języków zapewnia kompatybilność z oprogramowaniem wspomagającym implementację systemów wieloagentowych JADE co znacznie ułatwi proces implementacji.

Metodologia tworzenia aplikacji na system Android wykorzystuje dwie podstawowe klasy: *Activity* i *Service*. Poniżej krótko przedstawię cechy charakteryzujące te klasy:

- Obiekty klasy *Activity* odpowiadają warstwie prezentacji aplikacji. Każdy widok ma swój oddzielnny obiekt dziedziczący po tej klasie. Wygląd każdego z nich zdefiniowany jest w pliku XML o nazwie odpowiadającej nazwie widoku. Głównym zadaniem tej klasy jest odbieranie zdarzeń przychodzących od użytkownika, jak i innej części aplikacji. Zdarzenia pochodzące z aplikacji są opisywane przez obiekty klasy *Intent*. Po odebraniu takiego zdarzenia jest ono obsługiwane przez funkcję z obiektu aktywności.
- Obiekty klasy *Service* służą do wykonywania zadań w tle. Taki serwis może działać nawet w momencie, gdy żadna aktywność aplikacji nie jest uruchomiona, czyli gdy

aplikacja jest uśpiona, nieaktywna lub czeka na wznowienie. Może też być wykorzystany jako obiekt pośredniczący w komunikacji między aplikacjami.

Ostatecznie aplikacja składa się z trzech podstawowych obiektów: klasy agenta, aktywności menu i aktywności mapy. Poza nimi powstało jeszcze kilka pomniejszych dodatkowych klas, które opiszę później.

Jedyną zewnętrzną bibliotekę, jaką użyłem w mojej aplikacji jest biblioteka JADE-LEAP oraz potrzebnych do niej kodeków. Pozwala ona uprościć implementację systemu wieloagentowego zgodnego ze specyfikacjami FIPA.

W trakcie pisania aplikacji okazało się, że JADE-LEAP nie pozwala na uruchomienie serwera JADE, do którego mogłyby się łączyć inne urządzenia. Rozwiązałem to poprzez postawienie zewnętrznego serwera, do którego aplikacja będzie się mogła łączyć. Na samym serwerze nie wykonują się żadne operacje. Służy on jedynie do komunikacji między agentami. Rozwiązało to też niedogodność z potrzebą znania adresu IP telefonu gospodarza, który nie jest łatwy do przekazania i zapamiętania. Zamiast tego można wyszukać gospodarza po jego nazwie, która jest łatwiejsza. Serwer zapewnia też unikalność nazw, więc nie ma problemu z niejednoznacznością.

6.1 Wykorzystane algorytmy

W niniejszej aplikacji występują tylko dwa algorytmy: znajdowania NMS oraz głosowania. Poniżej opiszę szczegółowo każdego z nich.

Algorytm znajdowania NMS znajduje środek na mapie pomiędzy wszystkimi osobami w grupie. Miałem do wyboru dwie opcje: średnia geograficzną i średnią arytmetyczną. Średnia geograficzna jest o wiele dokładniejsza przy dużych odległościach pomiędzy użytkownikami, bo bierze pod uwagę krzywiznę Ziemi. Nie ma też problemu w przypadkach skrajnych takich jak spotkanie dwóch ludzi na dwóch różnych biegunach. GPS ma dwie wartości oznaczające długość (od -180 do 180 stopni) i szerokość (od -90 do 90 stopni) geograficzną. W przypadku długości geograficznej występuje problem ze średnią arytmetyczną, ponieważ w przypadku gdy jedna osoba jest na -179 stopniu długości, a druga na 179 stopniu długości to średnia arytmetyczna pokaże miejsce spotkania po drugiej stronie kuli Ziemskiej zamiast na 180 południku. Jednak w przypadku mojej aplikacji, która ma służyć wspomaganiu decyzji o miejscu spotkania w mieście, średnia arytmetyczna w zupełności wystarcza. Przy małych obszarach takich jak miasto jest wystarczająco dokładna, a wcześniej wspomniany przypadek na lądzie może wystąpić tylko w górach

Czukockich w Rosji, lub na Fidżi. Zatem uznałem, że ten problem dotyczy tak małej części świata, że mogę go pominąć.

$$dl_{NMS} = \frac{\sum_{i=1}^N dl_i}{N}$$

$$szer_{NMS} = \frac{\sum_{i=0}^N szer_i}{N}$$

Dl_{NMS} i $Szer_{NMS}$ oznaczają długość i szerokość wyliczonego najlepszego miejsca spotkania. N w powyższych równaniach reprezentuje liczbę osób w grupie, a Dl i $szer$ odpowiednio długość i szerokość geograficzną każdego użytkownika.

Algorytm głosowania ma za zadanie pomóc gospodarzowi grupy z podjęciem decyzji, które miejsce wybrać. Każda osoba w grupie dostaje 5 głosów, które może wydać na dowolny wybrany punkt. Można głosować na ten sam punkt wiele razy, aby zaznaczyć swoje preferencje. Głosowanie jest anonimowe, a jego wyniki może zobaczyć wyłącznie gospodarz grupy. Ostateczna decyzja jednak należy do gospodarza grupy.

6.2 Klasa agenta

Klasa agenta nazywa się *MobileAgent*. Jej podstawowym zadaniem jest stworzenie agenta i dodanie mu odpowiednich zachowań, aby spełniał swoją funkcję. Stworzyłem tylko jedną taką klasę, która dostosowuje się i dodaje odpowiednia zachowania w zależności od trybu, w jakim została uruchomiona. Do tej klasy potrzebny był abstrakcyjny interfejs, który nazwałem *MobileAgentInterface*. Dzięki niemu aplikacja może komunikować się agentem, natomiast komunikacja w drugą stronę odbywa się za pomocą już wcześniej wspomnianych *Intent'ów*.

Kod zawarty w tej klasie można podzielić na trzy kategorie: funkcje dodające zachowania, metody interfejsu i definicje zachowań.

6.2.1 Zachowania

Akcje agenta są zdefiniowane w zachowaniach, które można dowolnie dodawać i usuwać w trakcie działania programu. W mojej aplikacji agent może być uruchomiony w dwóch trybach: gospodarza i gościa. Jednak nie różnią się one niczym poza kilkoma innymi zachowaniami. Przez to postanowiłem, że wystarczy jedna instancja agenta, do której zaaplikuje odpowiednie zachowanie.

Zachowania agenta gospodarza:

- *registerBehaviour* – jest to zachowanie cykliczne, które służy do odbierania komunikatów związanych z dołączaniem do grupy. Czeka ono na w uśpieniu na wiadomość spełniające te kryteria i jeśli przyjdzie taka informacja, to jest aktualizowana lista osób w grupie, a następnie dodawane jest jednorazowe zachowanie *UpdateGroupBehaviour*. Po spełnieniu swojej funkcji zachowanie zostaje uśpione aż do przyjścia następnej wiadomości.
- *UpdateGroupBehaviour* – jest to zachowanie jednorazowe, które rozsyła do wszystkich członków grupy informacje o dodaniu nowej osoby. Wysyła także *Intent* do aktualnego widoku aplikacji sygnalizując potrzeba zaktualizowania listy osób (w przypadku widoku menu) lub dodania punktu na mapie (w przypadku u widoku mapy)
- *deRegisterBehaviour* – jest to zachowanie cykliczne, które odpowiada za odbieranie i obsługiwanie komunikatów związanych z opuszczaniem grupy. Po odebraniu takiej wiadomości rozsyła komunikat do wszystkich pozostałych członków grupy i do aktualnego widoku. Gdy żadna wiadomość nie przychodzi, przechodzi w stan uśpienia.
- *changeState* – jest to jednorazowe zachowanie, uruchamiane w momencie przejścia do kolejnej fazy. Jego zadaniem jest zmiana fazy w aktualnym agencie i poinformowanie reszty agentów o zmianie stanu.
- *receiveLocationUpdateBehaviour* – jest to cykliczne zachowanie, które oczekuje w uśpieniu na wiadomość z aktualizacją położenia klienta. Po otrzymaniu takiej informacji aktualizuje aktualne położenie dla zgłaszającego urządzenia.
- *updateLocation* – jednorazowe zachowanie, które aktualizuje obecną pozycję urządzenia. Jest uruchamiane co każdą zmianę pozycji GPS.
- *locationBroadcast* – zachowanie cykliczne, które wybudza się co dziesięć sekund. Jego zadaniem jest wysłanie informacji o aktualnej pozycji wszystkich osób z grupy do klientów.
- *broadcastBehaviour* – cykliczne zachowanie, które co dziesięć sekund rozsyła do wszystkich pozostałych członków grupy wiadomość z aktualnymi pozycjami pozostałych użytkowników

- *groupMapUpdate* – zachowanie cykliczne, które co dziesięć sekund wysyła *Intent*, do aktywnego widoku, powiadamiając o potrzebie przerysowania pozycji pozostałych osób z grupy.
- *calculateCenter* – jest to cykliczne zachowanie, dodane po przejściu do widoku mapy, które co dziesięć sekund oblicza NMS opisanym powyżej algorytmem. Następnie rozsyła tę pozycję do wszystkich osób z grupy. Po przejściu do fazy *CHOOSE* to zachowanie jest usuwane.
- *sendSelectedPlaceBehaviour* – jednorazowe zachowanie, które jest wykonywane w momencie dodania nowego miejsca na mapie. Rozsyła wiadomość do pozostałych osób w grupie z informacją o nazwie miejsca i jego położeniu
- *getPlaceBehaviour* – zachowanie cykliczne dodawane w fazie *CHOOSE*, które odbiera wiadomości z informacją o nowym dodanym punkcie zawierającą jego nazwę i położenie. Po odebraniu takiej wiadomości dodaje zachowanie *sendSelectedPlaceBehaviour*.
- *SendDestinationBehaviour* – jest to jednorazowe zachowanie, którego zadaniem jest rozesłanie wybranego przez gospodarza miejsca jako miejsca spotkania do wszystkich uczestników grupy.

Zachowanie agenta klienta:

- *register* – jest to jednorazowe zachowanie, które wysyła wiadomość z prośbą o dodanie do grupy.
- *updateGroup* – cykliczne zachowanie, które w uśpieniu oczekuje na wiadomość od gospodarza z informacją o członkach grupy i ich pozycji GPS. Po odebraniu takiego komunikatu aktualizuje informacje i wysyła *Intent* do aktywnego widoku.
- *updateLocation* – cykliczne zachowanie, które co dziesięć sekund wysyła informacje do gospodarza zawierającą aktualną lokację urządzenia.
- *groupMapUpdate* – zachowanie cykliczne, które co dziesięć sekund wysyła Intent, do aktywnego widoku, powiadamiając o potrzebie przerysowania pozycji pozostałych osób z grupy.

- *getCenter* – zachowanie cykliczne uruchamiane po włączeniu widoku mapy. Odpowiada za odbieranie wiadomości o obliczonym NMS i informuje aktualny widok o tym fakcie.
- *getSelectedPlaceBehaviour* – jest to cykliczne zachowanie dodawane po przejściu do fazy *CHOOSE* odpowiedzialne za odbieranie informacji o dodanych miejscach przez innych użytkowników. Po przejściu do fazy *VOTE* to zachowanie jest usuwane.
- *getDestinationBehaviour* – jest to zachowanie cykliczne, które oczekuje na wiadomość z informacją o docelowym miejscu. To zachowanie jest dodawane w momencie, jak faza *LEAD* się zaczyna, a usuwane w momencie, gdy otrzyma informację o celu podróży.

6.3 Aktywność menu

Klasa aktywności menu nazywa się *Menu*. Jest powiązana z odpowiadającym jej plikiem XML *activity_menu*, który definiuje jej *wygląd*. Poszczególne pola i przyciski są zdefiniowane w pliku *content_menu*. Dokładny opis widoku znajduje się w dodatku A. W klasie *Menu* zawarta jest logika obsługująca zdarzenia zarówno pochodzące od użytkownika, jak i od agenta. Jest tutaj też odpalany serwis, który odpowiada za uruchomienie agenta, przekazując mu podane przez użytkownika informacje i połączenie go z serwerem.

6.4 Aktywność mapy

Klasa aktywności mapy nazywa się *MapsActivity* i jest to główny widok całej aplikacji. Zawartość tego widoku zawarta jest w pliku *activity_maps*. Dokładny opis widoku znajduje się w dodatku A. Ta aktywność obsługuje wszystkie zdarzenia związane z mapą i jej nawigacją. Odbiera też *Intent'y* od agenta aktywnego na urządzeniu i na ich podstawie aktualizuje widok.

6.4 Dodatkowe klasy

Poza powyżej wymienionymi klasami, w trakcie projektu powstały też dodatkowe, pomocnicze funkcjonalności.

BitMapResize jest klasą zawierającą pojedynczą publiczną i statyczną metodę, która służy do przeskalowywania bitmap, które wykorzystuje do reprezentowania punktów na mapie.

EncodedPolylineDecoder jest to klasa, która tak jak *BitMapResize* posiada tylko jedną publiczną i statyczną metodę. Odpowiada ona za zamianę odpowiedzi od API Google Maps,

która przychodzi w formacie JSON, w obiekt listy, który można już w prosty sposób wyświetlić na mapie.

PopUpWindow jest to klasa, która definiuje wyskakujące okienko. Wykorzystuje ją do wyświetlania powiadomień związanych z błędami lub opuszczeniem przez kogoś grupy.

7 Testy

Testowanie opisywanej aplikacji odbyło się przy pomocy testowania funkcjonalnego. Wykorzystane zostało sześć telefonów z różnymi wersjami systemu operacyjnego: dwa telefony z wersją *Marshmallow*, trzy telefony z wersją *Lollipop* i jeden telefon z systemem *Jelly Bean*. Poza tymi urządzeniami aplikacja była jeszcze testowana na emulatorze zainstalowanym systemem w wersji 4.4 – *KitKat*. Poniżej opiszę, w jaki sposób każda funkcjonalność była testowana.

Wygodny i prosty interfejs użytkownika – do przetestowania tej funkcjonalności potrzebne były dwa testy: test sprawdzający prostotę i czytelność interfejsu oraz test sprawdzający wygodę użytkowania.

W pierwszym teście osoby testujące dostały aplikację bez wcześniejszego pokazania im jak jej używać. Każdy z testerów wiedział, co aplikacja ma robić i ich zadaniem było sprawdzenie, czy sam interfejs wystarczy, aby użytkownik mógł przejść bezproblemowo od początku do końca działania aplikacji. Następnie pytałem się, który moment był najmniej oczywisty i sprawdzałem po kolejni czy wszystkie funkcjonalności zostały użyte. Taki test ma swoją wadę, ponieważ jedna osoba może go przeprowadzić tylko raz. Potem już osoba testująca korzysta z wiedzy zdobytej w poprzednim teście, co powoduje, że wyniki mogą być zakłamane.

Wyniki: Z sześciu osób tylko cztery osoby dotarły do końca programu, ale żadna z nich nie używa możliwości przesunięcia wyznaczonego najlepszego miejsca spotkania. Spośród tych czterech osób tylko trzy użyły możliwości głosowania więcej niż jeden raz. Dwie osoby, którym nie udało się dojść do końca, zatrzymały się w menu, na samym początku aplikacji.

Drugi test dotyczący wygody interfejsu polegał na prostym używaniu przez testerów aplikacji i spisywaniu rzeczy, które według nich by się jeszcze przydały lub należałyby coś zmienić.

Wyniki: Większość otrzymanych uwag dotyczyła wyglądu aplikacji. Testerzy zgłosili, że nie ma nigdzie informacji o aktualnej fazie, w której się znajduje aplikacja, a liczba głosów do wydania jest za mało widoczna. W widoku menu też nie dla wszystkich było jasne, w jaki sposób można założyć grupę, a w jaki sposób się dołączyć do już stworzonej grupy.

Tryb założyciela grupy i klienta – do przetestowania trybu założyciela wystarczy tylko jedna osoba, ale aby przetestować tryb klienta, potrzebny jest też założyciel grupy. Z tego powodu najpierw został przetestowany tryb założyciela.

Testując tryb założyciela, każdy z testerów miał za zadanie stworzyć swoją własną grupę i w pierwszej iteracji testów przejść przez całą aplikację bez żadnego innego członka. Następna iteracja miała przebiec z dwoma członkami grupy.

Wyniki: Bardzo szybko został wyłapany problem z resetowaniem się wyznaczanego najlepszego miejsca spotkania. Spowodowane było to nie usunięciem odpowiedzialnego za to zachowania po pierwszym obliczeniu. Innym problemem było resetowanie się całej aplikacji po zminimalizowaniu jej. Ten problem został rozwiązany poprzez zmienienie trybu, w jakim była uruchamiana aktywność mapy.

Testy trybu klienta przebiegały w podobny sposób jak druga iteracja testów trybu założyciela. Stworzony był jeden gospodarz i wszyscy testerzy mieli się dołączyć do jego grupy, a potem wspólnie przejść przez działanie aplikacji.

Wyniki: Poza błędami opisanymi w wynikach testów trybu założyciela testerzy zauważyli, że nic nie informuje klientów grupy o tym, na jaki punkt wydali swój głos. Innym zgłoszonym problemem był brak informacji o momencie przejścia przez gospodarza do kolejnej fazy.

Nawiązywania połączenia z innymi użytkownikami z grupy poprzez serwer – testowanie tej funkcjonalności polegało na wykonywaniu takich czynności w aplikacji w trybie gospodarza lub klienta, które powinny wysyłać wiadomości do pozostałych użytkowników. Następnie na pozostałych urządzeniach sprawdzało się, czy taka wiadomość faktycznie przyszła i czy została poprawnie obsłużona.

Wyniki: Podczas tych testów nie został znaleziony żaden błąd.

Rozwiązywanie połączenia z grupą i serwerem – aby przetestować tę funkcjonalność, jeden tester tworzył grupę, a reszta się dołączała. Następnie paru klientów rozłączało się, a następnie sam gospodarz się rozłączał.

Wyniki: Znaleziono błąd w momencie, gdy klient opuści grupę w trakcie działania programu. Był on spowodowany błędym wyrejestrowywaniem się agenta. Innym zgłoszonym błędem był całkowity brak wyrejestrowania w momencie zabicia aplikacji. Niestety jest to spowodowane tym, że aktywność w momencie zabicia może, ale nie musi

wywołać metody `onDestroy`. Skutkuje to w braku wysłanej wiadomości z prośbą o rozłączenie. Serwer JADE też aktywnie nie rozpoznaje i nie zabija nieodpowiadających agentów. Można to było rozwiązać, tworząc agenta na serwerze, który odpytuje wszystkich agentów po kolej i zabija te nieodpowiadające, ale rozwiązanie serwera nie znajduje się w zakresie tej pracy.

Wyświetlania położenia pozostałych członków grupy – do sprawdzenia tej funkcjonalności testerzy dołączali do wcześniej stworzonej grupy i w widoku mapy obserwowali czy zmiana położenia pozostałych członków grupy jest zgodna z rzeczywistością.

Wyniki: Jedyna zgłoszoną przez dwóch testerów wątpliwością był fakt, że po zminimalizowaniu aplikacji położenie takiego urządzenia nie jest aktualizowane na mapie pozostałych użytkowników. Jest to jednak zamierzone, bo jeśli ktoś nie chce ujawniać swojej pozycji, to ma taką możliwość.

Wyznaczania proponowanej okolicy spotkania – do przetestowania tej funkcjonalności osoby testujące w pierwszej iteracji uruchamiały aplikację trybie gospodarza. W tym momencie środek wyznaczonego najlepszego miejsca spotkania powinien znajdować się w tym samym miejscu na mapie co gospodarz. W drugiej iteracji testerzy łączą się w grupy i sprawdzają, czy wyznaczone najlepsze miejsce spotkania znajduje się na środku pomiędzy wszystkimi członkami grupy.

Wyniki: W pierwszej iteracji nie wykryto żadnych błędów. W drugiej iteracji wykryto jeden błąd. Aplikacja nie oczekiwała na zgłoszenie pozycji wszystkich członków grupy, tylko wyznaczała środek spotkania po określonym czasie. Zostało to zmienione w taki sposób, aby aplikacja czekała na informacje o lokacji wszystkich członków, którzy dołączyli przed tą fazą.

Wyświetlania i dodawania punktów użyteczności publicznej – testowanie tej funkcjonalności polegało na dodawaniu nowych lub tych samych punktów w grupie zarówno przez gospodarza, jak i przez pozostałe osoby w grupie. Sprawdzane było dodawanie wiele razy tego samego punktu, jak i dodawanie ich bardzo dużej ilości.

Wyniki: Pierwszym znalezionym błędem była możliwość dodania wielokrotnie tego samego punktu, który pojawiał się jako oddzielny obiekt na mapie. Drugim znalezionym

problemem było spowolnione działanie aplikacji po dodaniu dużej ilości miejsc. Po przekroczeniu 20 dodanych miejsc aplikacja spowalnia.

Głosowania na punkty użyteczności publicznej – aby przetestować tę funkcjonalność, testerzy stworzyli grupę, dodali kilkanaście punktów na mapie, a następnie głosowali na te punkty.

Wyniki: Podczas tych testów nie znaleziono błędów funkcjonalnych. Testerzy mieli jedynie uwagę co do reprezentacji oddanych głosów na mapie gospodarza. Zostało to rozwiązane poprzez powiększenie punktu w zależności od ilości głosów. To daje łatwą do zrozumienia graficzną prezentację stanu głosowania.

Nawigacji do wybranego punktu – ta funkcjonalność była testowana w obu trybach. Testerzy wyłącznie skupiali się na aspekcie nawigacji do punktów w różnych miejscach na całej Ziemi.

Wyniki: Już po pierwszych testach okazało się, że w momencie wyznaczania trasy do celu, na mapie gospodarza wyznaczane są dwie trasy: prawidłowa i w linii prostej. Spowodowane to było podwójnym zapytaniem wysyłanym do API Goole Maps.

8 Wnioski i podsumowanie

Zaprojektowana aplikacja spełnia wszystkie wymagania funkcjonalne przedstawione wcześniej. System pozwala na tworzenie grup, które później mają okazję dodać wybrane przez siebie proponowane miejsca spotkania i mogą głosować na nie. Po wybraniu już miejsca docelowego wszyscy są do niego prowadzeni. Udało się też zaimplementować tak architekturę aplikacji, że łatwo ją rozbudowywać. Dodanie nowego zachowania lub wiadomości z natury systemu agentowego jest bardzo proste.

Pokazano również, że system wieloagentowy bardzo dobrze wspiera tego typu rozwiązania. Smartfony posiadają wszystkie charakterystyki, które powinien posiadać agent: są autonomiczne, nie widzą całości systemu oraz są zdecentralizowane. Ten fakt znacznie ułatwił projektowanie systemu.

Testy wykazały, że interfejs może być zbyt nieczytelny i niejasny dla nowych użytkowników. Jednak prace nad tym wymagają dużej ilości czasu i wielu iteracji. Mimo to jest to bardzo ważny element aplikacji, ponieważ potencjalny sukces na rynku zależy od niego.

Aktualnie serwer, przez który komunikuje się aplikacja, stoi na mojej prywatnej maszynie. Niestety posiada ona zmienne IP, przez co wymagane jest w aplikacji pole na adres IP serwera. Jest to bardzo mylące dla użytkowników, którzy mylą to z adresem IP gospodarza. Najlepszym rozwiązaniem byłoby postawienie serwera ze statycznym adresem. W taki sposób można by całkowicie usunąć ple wypełniane przez użytkownika, a adres serwera przenieść do kodu programu. Ułatwiłoby to korzystanie z aplikacji, ponieważ wtedy wystarczy wpisać tylko swoją nazwę i ewentualną nazwę gospodarza. Wymaga to jednak posiadanie lub wykupienia takiego serwera, co na ten moment rozwoju aplikacji nie jest konieczne.

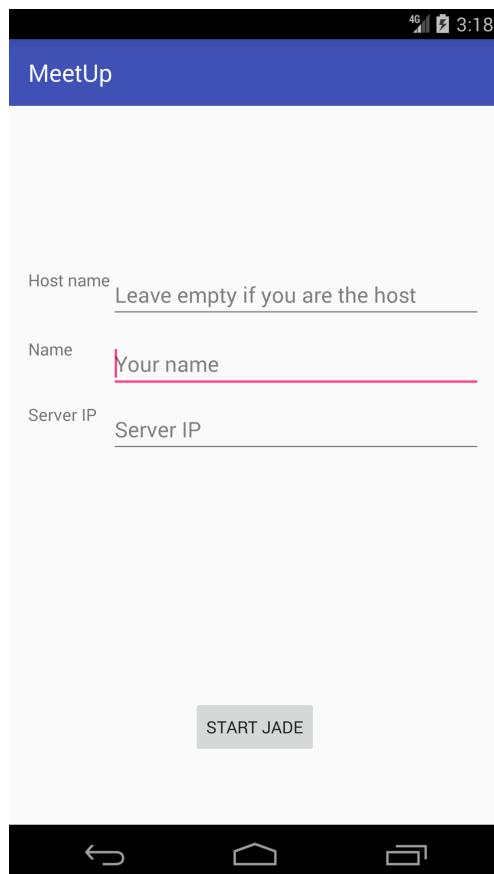
Najważniejszym elementem dalszych prac przy rozwoju aplikacji jest poprawa interfejsu graficznego aplikacji, aby była przyjaźniejsza i bardziej atrakcyjna dla użytkownika. Do tego jednak potrzebna jest informacja zwrotna od większej liczby użytkowników, szczególnie takich, którzy pierwszy raz korzystaliby z aplikacji.

Równolegle do poprawek w interfejsie można też prowadzić pracę nad wersjami tego rozwiązania na inne systemy operacyjne. Fakt użycia narzędzia JADE bardzo ułatwia przeniesienie tego systemu na inne systemy operacyjne, które obsługują Java. W aplikacji można też dodać nowe funkcjonalności, które zostały zaproponowane przez osoby testujące. Przykładami dodatkowych funkcjonalności są:

- lista z dostępnymi grupami z rozdziałem na grupy publiczne (bez hasła) i grupy prywatne (z hasłem),
- czat w obrębie grupy,
- możliwość zmiany miejsca docelowego, jeśli okaże się, że we wcześniej wybranym lokalu nie ma już miejsc,
- powiadomienia o momencie zmiany fazy lub jeśli ktoś z grupy dotrze do celu,
- zawężanie wyboru lokalów do kategorii takich jak: restauracja czy bar i inne miejsca o podobnym charakterze.

Dodatek A Obsługa aplikacji

Poniżej zamieszczam instrukcję obsługi aplikacji, która powstała w ramach pracy inżynierskiej. Pierwszym widocznym interfejsem, identycznym dla obu trybów, jest ekran menu pokazany na rysunku A.1.

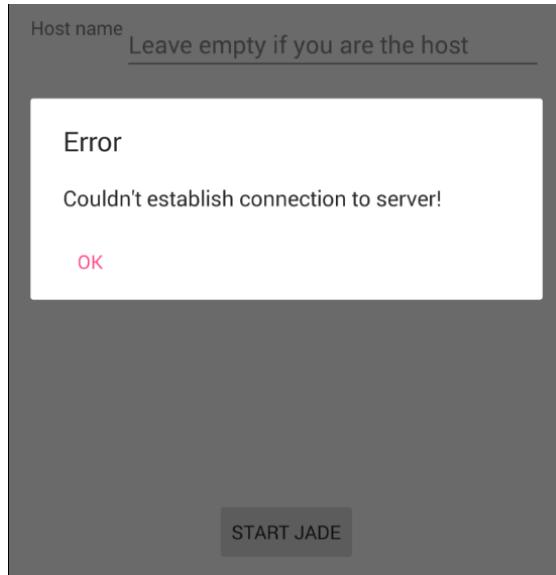


Rysunek A.1. Interfejs graficzny widoku menu aplikacji mobilnej

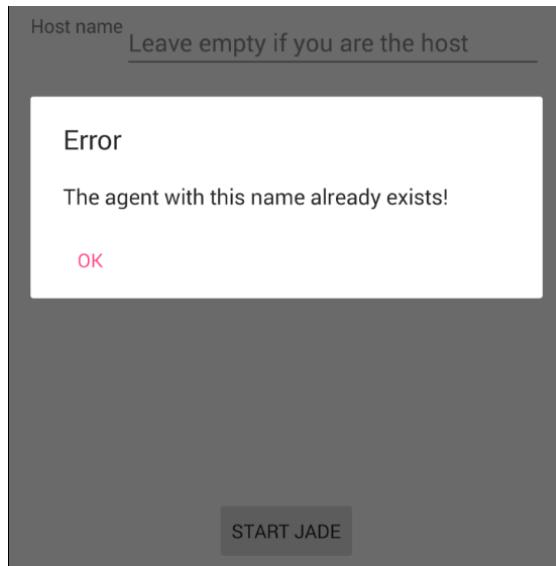
Interfejs menu składa się z czterech elementów:

- *Host name* – jest to pole tekstowe, które należy wypełnić nazwą założyciela grupy, do którego użytkownik chce się przyłączyć. Aby samemu zostać gospodarzem grupy, należy zostawić to pole puste.
- *Name* – pole tekstowe, w które należy wpisać własną nazwę. Inni użytkownicy aplikacji mogą się dołączyć do grupy używając właśnie tej nazwy. Musi być ona unikalna.
- *Server IP* – jest to pole tekstowe, w które należy wpisać adres IP serwera, przez który będzie się komunikowała aplikacja
- Przycisk **START JADE** - jest to przycisk uruchamiający agenta. Pobierze on adres IP serwera, nazwę użytkownika i opcjonalną nazwę założyciela grupy. W zależności od

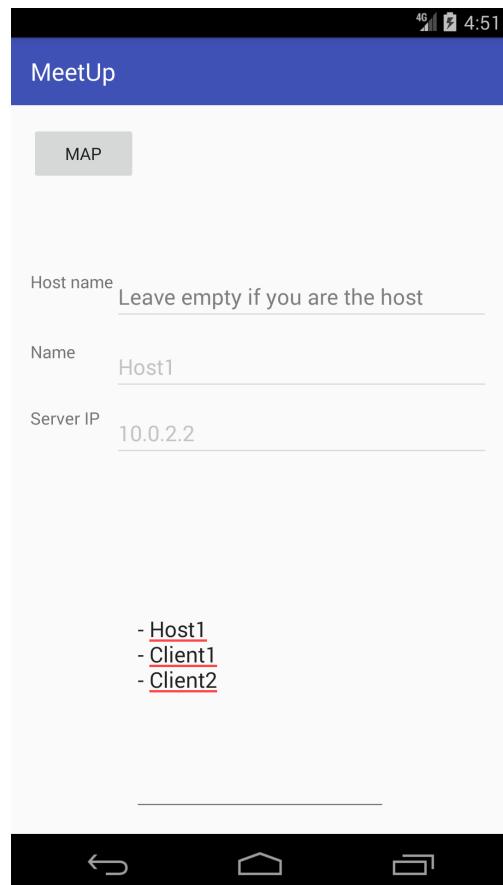
tego, czy pole *Host name* jest wypełnione, czy nie to aplikacja przejdzie w odpowiedni tryb (gospodarza, jeśli puste, klienta, jeśli wypełnione). Jeśli połączenie z serwerem się nie uda, użytkownik zostanie powiadomiony wyskakującym okienkiem jak na rysunku A.2. Jeśli nazwa jest już zajęta, to pojawi się okienko przedstawione na rysunku A.3.



Rysunek A.2. Okienko powiadamiające o niemożliwości nawiązania połączenia z serwerem



Rysunek A.3. Okienko powiadamiające o zajętej nazwie użytkownika



Po uruchomieniu agenta w trybie gospodarza lub po podłączeniu się w trybie klienta pojawi się nowy przycisk i niemodyfikowalne pole tekstowe widoczne na rysunku A.4. Rysunek A.4. Interfejs graficzny widoku menu po uruchomieniu agenta

Nowe elementy to:

- Przycisk *MAP* - jest to przycisk, który służy do przejścia do widoku mapy.
- Lista podłączonych użytkowników – jest to niemodyfikowalne pole tekstowe, które można w razie potrzeby przewijać, zawierające nazwy wszystkich użytkowników w grupie. Na samej górze zawsze jest własna nazwa.

A.1 Instrukcja dla gospodarza

A.1.1 Faza GATHER

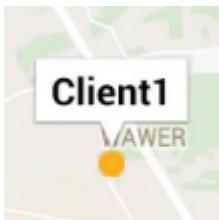
Po uruchomieniu widoku mapy w trybie gospodarza pokaże się mapa. Po krótkim czasie użytkownikowi pokaże się ekran widoczny na rysunku A.5.



Rysunek A.5. Widok mapy w trybie gospodarza w fazie GATHER

W tym widoku można wyróżnić cztery nowe elementy:

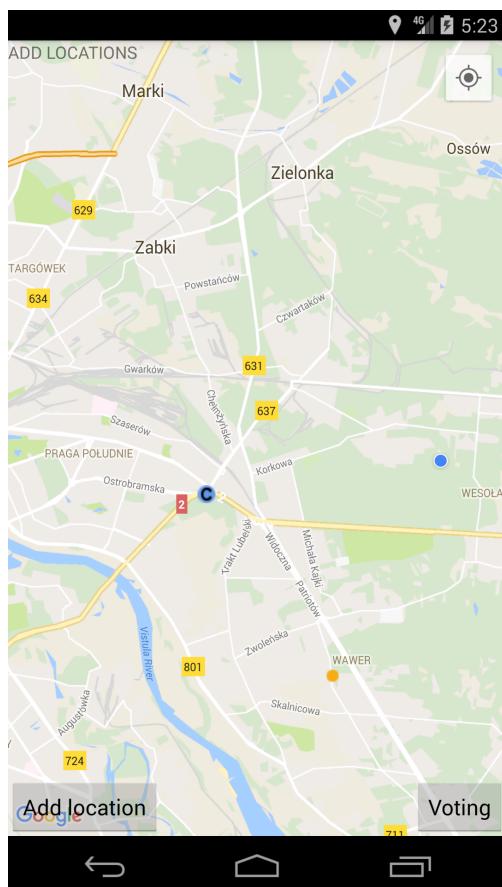
- jednokolorowy niebieski punkt – oznacza on aktualne położenie użytkownika aplikacji
- jednokolorowy pomarańczowy punkt – oznacza aktualne położenie innej osoby z grupy. Pojawia się po otrzymaniu informacji od urządzenia tej osoby. Można też nacisnąć dany punkt, aby wyświetlić nazwę członka grupy (rysunek A.6).
- Niebieski punkt z literą „C” w środku – oznacza obliczone najlepsze miejsce spotkania. Może zostać przesunięte poprzez przytrzymanie i przeniesienie go w wybrane miejsce.
- Przycisk *Start* – przycisk, który pokazuje się dopiero po wyznaczeniu najlepszego miejsca spotkania. Wciśnięcie go spowoduje przejście do kolejnej fazy aplikacji.



Rysunek A.6. Wyświetlona nazwa użytkownika nad jego pozycją w widoku mapy

A.1.2 Faza CHOOSE

Po przejściu do kolejnej fazy aplikacji (*CHOOSE*), gospodarzowi pokaże się ekran jak na rysunku A.7.

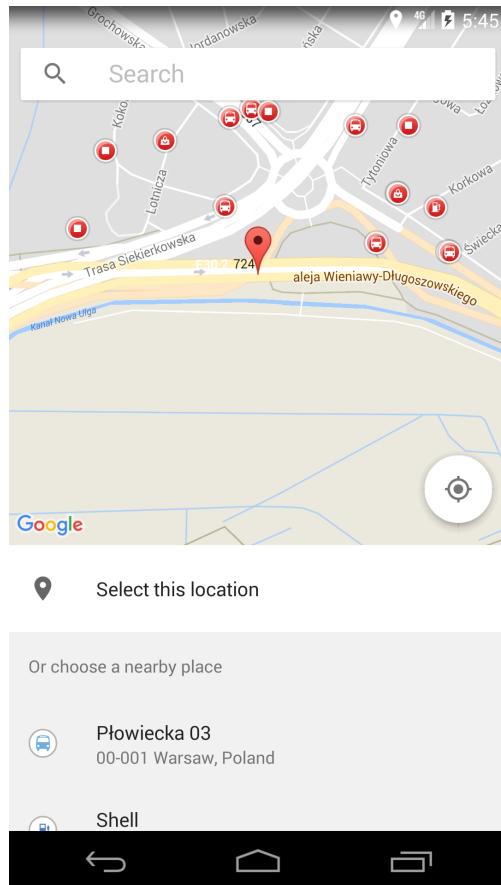


Rysunek A.7. Widok mapy w trybie gospodarza w fazie CHOOSE

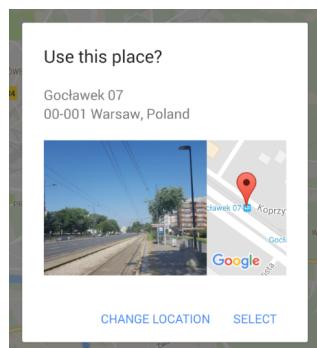
W tym widoku przycisk *Start* zamienia się w przycisk *Voting*. Jego działanie jest identyczne co poprzednio – przejście do kolejnej fazy, tym razem *VOTE*.

Poza tym guzikiem pojawia się też nowy przycisk, który odpowiada za uruchomienie widżetu *Place Picker* (rysunek A.8). Uruchamia się on wycentrowany na wybranej okolicy spotkania (niebieski punkt z literą „C”) i pozwala na wybranie punktów użyteczności publicznej lub po prostu dodanie własnego punktu. Akcję dodania punktu należy potwierdzić

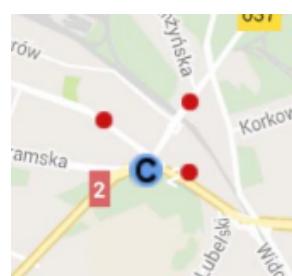
na wyskakującym okienku widocznym na rysunku A.9, które następnie pokażą się na głównym ekranie mapy jako ciemnoczerwone punkty (rysunek A.10).



Rysunek A.8. Widżet Place Picker



Rysunek A.9 Okno potwierdzające wybór miejsca

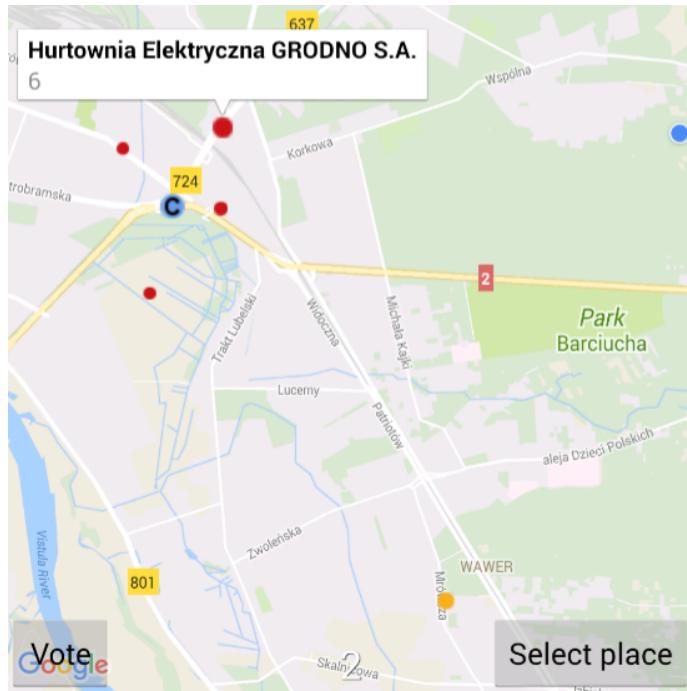


Rysunek A.10 Wybrane punkty oznaczone ciemno czerwoną kropką

Po dodaniu przez wszystkich w grupie zadowalającej liczby proponowanych miejsc do spotkania gospodarz może wcisnąć guzik *Voting*, aby przejść do kolejnej fazy.

A.1.3 Faza VOTE

W tej fazie gospodarz i pozostałe osoby z grupy mogą głosować na wybrane wcześniej punkty. Wyniki głosowania, w postaci powiększonego punktu, widzi tylko założyciel grupy. Ekran gospodarza w tej fazie będzie wyglądał podobnie jak na rysunku A.11.



Rysunek A.11. Widok mapy w trybie gospodarza w fazie VOTE

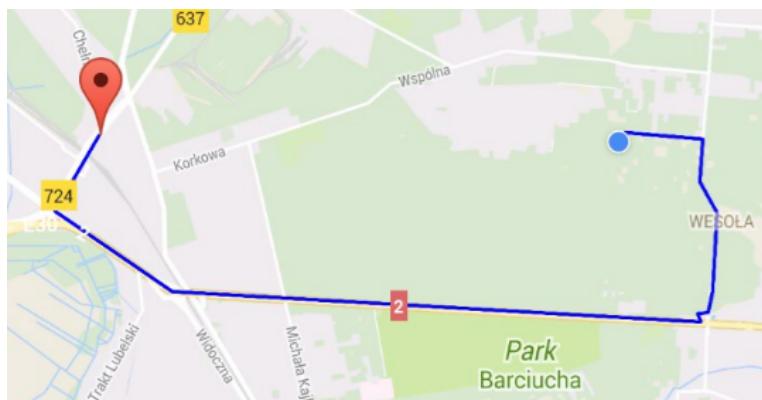
Na powyższym rysunku można zauważać następujące zmiany:

- Przycisk *Voting* zamienił się w guzik *Select place*, który odpowiada za przejście aplikacji do kolejnej fazy – *LEAD*. Aby móc przejść do kolejnej fazy, gospodarz musi mieć zaznaczony któryś z dodanych wcześniej punktów. Wybrany punkt nie musi posiadać największej liczby głosów – głosowanie tylko wspomaga założyciela przy podjęciu decyzji.
- Przycisk *Add location* został zastąpiony przyciskiem *Vote*. Służy on do oddawania głosu na aktualnie wybrany punkt.
- Pomiędzy przyciskami, na dole ekranu pojawił się numer oznaczający aktualną liczbę głosów, które może wydać gospodarz.

- Czerwone punkty, oznaczające wybrane wcześniej proponowane miejsca spotkań powiększają się proporcjonalnie do ilości oddanych na nie głosów. Można też nacisnąć na punkt, aby zobaczyć, ile dokładnie głosów zostało na niego oddanych.

A.1.4 Faza LEAD

Po przejściu do fazy *LEAD* zostanie wyznaczona droga dojazdu do wybranego miejsca, która wygląda jak na rysunku A.12. Podczas tej fazy też widać pozycję pozostałych członków grupy tak długo, jak ich aplikacja nie jest uśpiona.



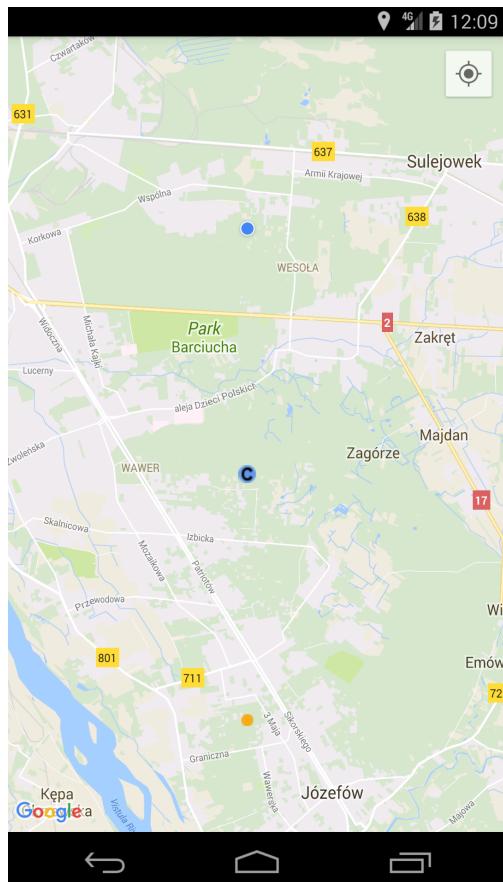
Rysunek A.12. Wyznaczona trasa do wybranego punktu docelowego

W momencie dotarcia wszystkich osób na wybrane miejsce aplikacja zakończyła swoje zadanie i można ją wyłączyć.

A.2 Instrukcja dla klienta

A.2.1 Faza Gather

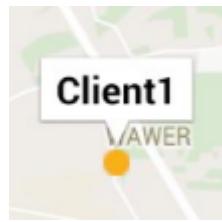
Po uruchomieniu widoku mapy w trybie Klienta pokaże się mapa. Po krótkim czasie użytkownikowi pokaże się ekran widoczny na rysunku A.13.



Rysunek A.13. Widok mapy w trybie klienta w fazie GATHER

W tym widoku można wyróżnić trzy nowe elementy:

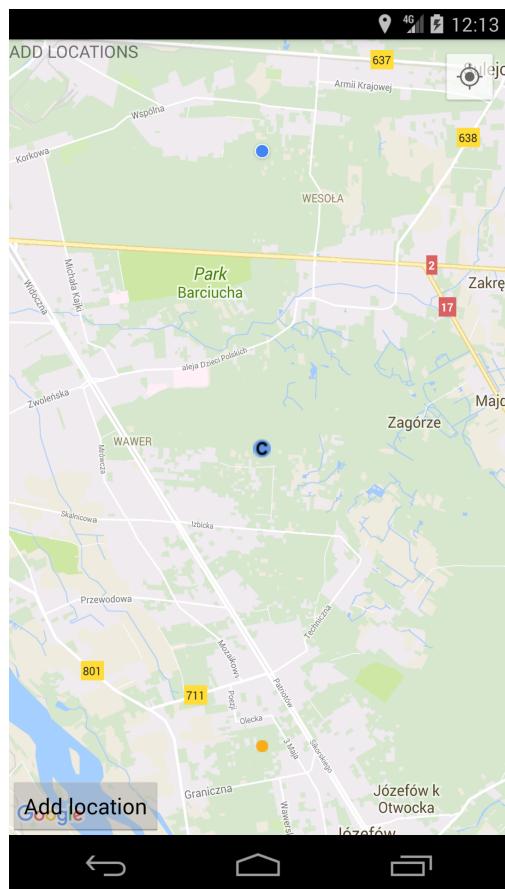
- jednokolorowy niebieski punkt – oznacza on aktualne położenie użytkownika aplikacji
- jednokolorowy pomarańczowy punkt – oznacza aktualne położenie innej osoby z grupy. Pojawia się po otrzymaniu informacji od urządzenia tej osoby. Można też nacisnąć dany punkt, aby wyświetlić nazwę członka grupy (rysunek A.14).
- Niebieski punkt z literą „C” w środku – oznacza obliczone najlepsze miejsce spotkania. Klient nie ma możliwości przesunąć tego znacznika.



Rysunek A.14. Wyświetlona nazwa użytkownika nad jego pozycją w widoku mapy

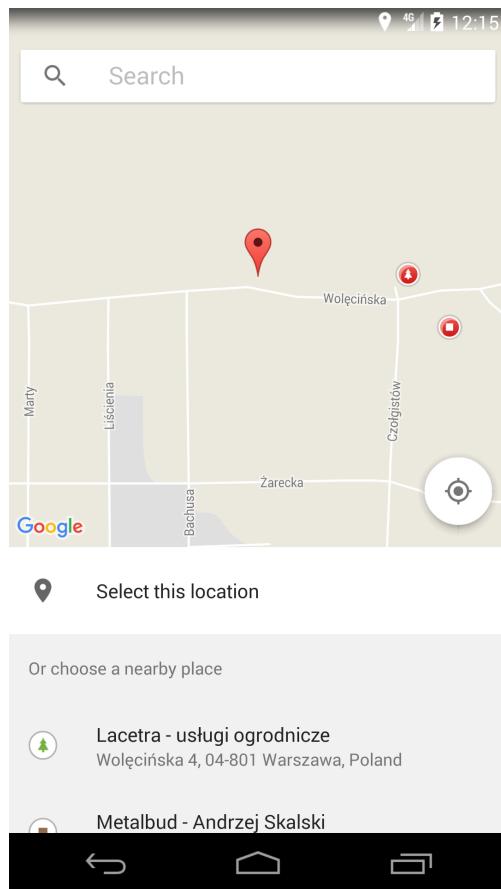
A.1.2 Faza CHOOSE

Po przejściu przez gospodarza grupy do kolejnej fazy aplikacji (*CHOOSE*), każdemu członkowi grupy poza gospodarzem pokaże się ekran jak na rysunku A.15.

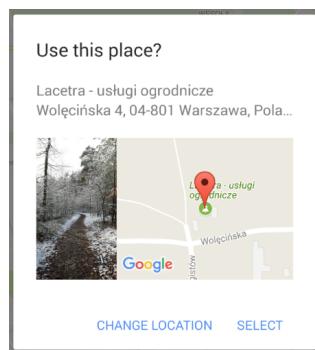


Rysunek A.15. Widok mapy w trybie klienta w fazie CHOOSE

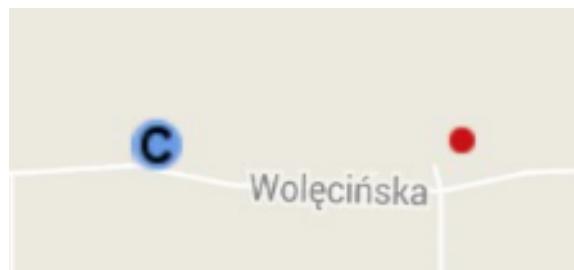
Jedynym przyciskiem występującym w tym widoku jest przycisk *Add location*, który odpowiada za uruchomienie widżetu *Place Picker* (rysunek A.16). Uruchamia się on wycentrowany na wybranej okolicy spotkania (niebieski punkt z literą „C”) i pozwala na wybór punktów użyteczności publicznej lub po prostu dodanie własnego punktu. Akcję dodania punktu należy potwierdzić na wyskakującym okienku widocznym na rysunku A.17, które następnie pokażą się na głównym ekranie mapy jako ciemnoczerwone punkty (rysunek A.10).



Rysunek A.16. Widżet Place Picker



Rysunek A.17 Okno potwierdzające wybór miejsca

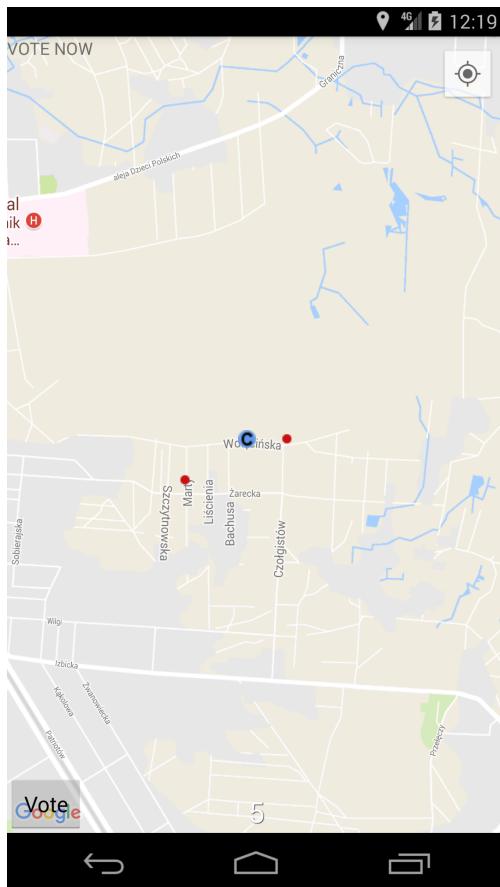


Rysunek A.18 Wybrane punkty oznaczone ciemnoniebieską kropką

Po dodaniu przez innych członków grupy kolejnych proponowanych miejsc do spotkania, będą się one wyświetlały w taki sam sposób.

A.1.3 Faza VOTE

W tej fazie wszyscy mogą głosować na wybrane wcześniej punkty. Wyniki głosowania, w postaci powiększonego punktu, widzi tylko założyciel grupy. Ekran klienta w tej fazie będzie wyglądał podobnie jak na rysunku A.19.



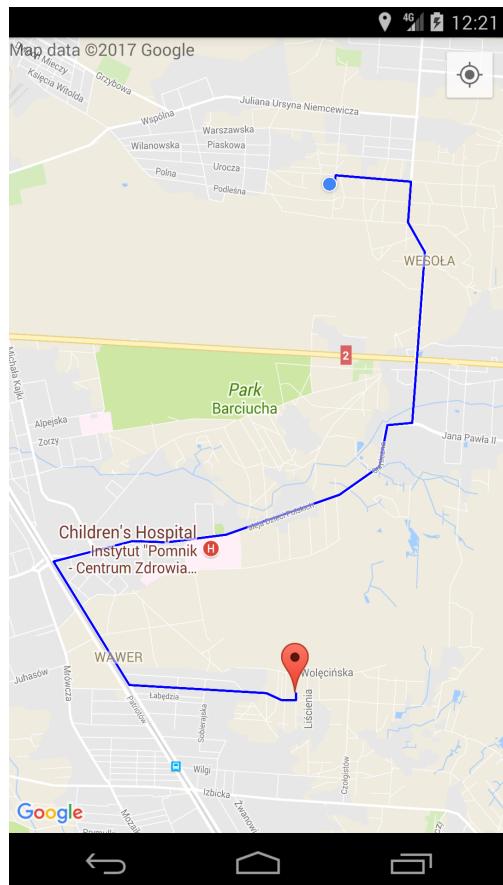
Rysunek A.19. Widok mapy w trybie klienta w fazie VOTE

Na powyższym rysunku można zauważać następujące zmiany:

- Przycisk *Add location* został zastąpiony przyciskiem *Vote*. Służy on do oddawania głosu na aktualnie wybrany punkt.
- Pomiędzy przyciskami, na dole ekranu pojawił się numer oznaczający aktualną liczbę głosów, które może wydać gospodarz.
- Czerwone punkty, oznaczające wybrane wcześniej proponowane miejsca spotkań pozwalają na wybranie miejsca, na które chce się głosować.

A.1.4 Faza *LEAD*

Po przejściu do fazy *LEAD* zostanie wyznaczona droga dojazdu do wybranego miejsca, która wygląda jak na rysunku A.20. Podczas tej fazy też widać pozycję pozostałych członków grupy tak długo, jak ich aplikacja nie jest uśpiona.



Rysunek A.20. Wyznaczona trasa do wybranego punktu docelowego

W momencie dotarcia wszystkich osób na wybrane miejsce aplikacja zakończyła swoje zadanie i można ją wyłączyć.

Bibliografia

- [1] Google Inc. *Mapa Google* 2017 Available at: <https://www.google.com/maps/> (Accessed: 19 August 2017)
- [2] OpenStreetMap. *Mapa OpenStreetMap* 2017 Available at: <https://www.openstreetmap.org/> (Accessed: 19 August 2017)
- [3] CITY-NAV. *Jakdojade* 2017 Available at: <https://jakdojade.pl> (Accessed: 5 September 2017)
- [4] Yuan Luo, Kechang Liu and D.N. Davis. *A Multi-Agent Decision Support System for Stock Trading*. [w:] „IEEE Network”. Tom: 16, Numer: 1, 01/02 2002 Available at: <http://www.reading.ac.uk/AcaDepts/si/sisweb13/ais/papers/journal12-A%20multi-agent%20Framework.pdf> (Accessed: 5 September 2017)
- [5] FIPA *IEEE Foundation for Intelligent Physical Agents* 2017 Available at: <http://www.fipa.org/> (Accessed: 17 August 2017)
- [6] IDC. *IDC: Smartphone OS market share* 2017 Available at: <http://www.idc.com/promo/smartphone-market-share/os> (Accessed: 17 August 2017).
- [7] Apple Inc. *Swift – Apple developer* 2017 Available at: <https://developer.apple.com/swift/> (Accessed: 17 August 2017)
- [8] Google Inc. *Dashboards | Android Developers*. 2017 Available at: <https://developer.android.com/about/dashboards/index.html> (Accessed: 22 August 2017).
- [9] Apple Inc. *App Store – Support – Apple Developers*. 2017 Available at: <https://developer.android.com/about/dashboards/index.html> (Accessed: 22 August 2017)
- [10] Agent Service *Agent Service official site* 2005 Available at: <http://www.agentservice.it/index.aspx> (Accessed: 17 August 2017)
- [11] DALI *Dali Multi Agent Systems Framework* 2016 Available at: <https://github.com/AAAI-DISIM-UnivAQ/DALI> (Accessed: 17 August 2017)
- [12] JADE *Telcom Italia: JAVA Agent Development Framework* 2017 Available at: <http://jade.tilab.com/> (Accessed: 17 August 2017)
- [13] JIAC *Java-based Intelligent Agent Componentware* 2016 Available at: <http://www.jiac.de/> (Accessed: 17 August 2017)
- [14] M. Wooldridge, N. R. Jennings, and D. Kinny. *The Gaia Methodology for Agent-Oriented Analysis and Design*. [w:] „Journal of Autonomous Agents and Multi-Agent Systems”. 3(3):285-312. 2000 Available at: <https://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/jaamas2000b.pdf> (Accessed: 19 August 2017)
- [15] Android. *Android Native Development Kit* 2017 Available at: <https://developer.android.com/ndk/index.html> (Accessed: 21 August 2017)

Wykaz symboli i skrótów

Skróty

Oznaczenie	Opis
FIPA	Foundation for Intelligent Physical Agents
OSM	OpenStreetMap

Spis rysunków

1	Podział rynku smartfonów ze względu na zainstalowany system operacyjny. Oś pozioma przedstawia umieszczone kolejno kwartały.....	9
2	Podział urządzeń z systemem Android ze względu na zastosowane w nich wersje tego systemu. Dane z 8 sierpnia 2017r.....	10
3	Podział urządzeń z systemem iOS ze względu na zastosowane w nich wersje tego systemu. Dane z 28 czerwca 2017r.....	10
4	Schemat blokowy stanów aplikacji.....	23
5	Pierwotny diagram widoków.....	24
6	Końcowy diagram widoków.....	25
7	Model widoku menu.....	25
8	Model widoku mapy.....	25
A.1	Interfejs graficzny widoku menu aplikacji mobilnej.....	39
A.2	Okienko powiadamiające o niemożliwości nawiązania połączenia z serwerem.....	40
A.3	Okienko powiadamiające o zajętej nazwie użytkownika.....	40
A.4	Interfejs graficzny widoku menu po uruchomieniu agenta.....	41
A.5	Widok mapy w trybie gospodarza w fazie <i>GATHER</i>	42
A.6	Wyświetlona nazwa użytkownika nad jego pozycją w widoku mapy.....	43
A.7	Widok mapy w trybie gospodarza w fazie <i>CHOOSE</i>	43
A.8	Widżet <i>Place Picker</i>	44
A.9	Okno potwierdzające wybór miejsca.....	44
A.10	Wybrane punkty oznaczone ciemno czerwoną kropką.....	44
A.11	Widok mapy w trybie gospodarza w fazie <i>VOTE</i>	45
A.12	Wyznaczona trasa do wybranego punktu docelowego.....	46
A.13	Widok mapy w trybie klienta w fazie <i>GATHER</i>	47
A.14	Wyświetlona nazwa użytkownika nad jego pozycją w widoku mapy.....	48
A.15	Widok mapy w trybie klienta w fazie <i>CHOOSE</i>	48
A.16	Widżet <i>Place Picker</i>	49
A.17	Okno potwierdzające wybór miejsca.....	49
A.18	Wybrane punkty oznaczone ciemno czerwoną kropką.....	49
A.19	Widok mapy w trybie klienta w fazie <i>VOTE</i>	50
A.20	Wyznaczona trasa do wybranego punktu docelowego.....	51

Spis tabel

1	Porównanie oprogramowań do programowania agentowego zgodnych z FIPA[1].	12
2	Opis roli Nadawcy.....	15
3	Opis roli Odbiorcy.....	16
4	Opis roli Założyciela.....	16
5	Opis roli Nawigatora.....	17
6	Opis roli Mediatory.....	17
7	Opis protokołu „Odczytaj grupę” dla roli nadawcy.....	18
8	Opis protokołu „Wyślij pozycję” dla roli nadawcy.....	18
9	Opis protokołu „Odczytaj pozycję” dla roli nadawcy.....	18
10	Opis protokołu „Odczytaj grupę” dla roli nawigatora.....	19
11	Opis protokołu „Odczytaj pozycję” dla roli nawigatora.....	19
12	Opis protokołu „Odbierz pozycję” dla roli nawigatora.....	19
13	Opis protokołu „WyślijNMS” dla roli nawigatora.....	20
14	Opis protokołu „Odczytaj grupę” dla roli mediatora.....	20
15	Opis protokołu „OdbierzGłosy” dla roli mediatora.....	20
16	Opis protokołu „OdczytajGłosy” dla roli mediatora.....	21
17	Opis protokołu „WyślijLokal” dla roli mediatora.....	21
18	Opis protokołu „OdbierzNMS” dla roli odbiorcy.....	21
19	Opis protokołu „OdbierzLokal” dla roli odbiorcy.....	22
20	Opis protokołu „OdbierzPozycje” dla roli odbiorcy.....	22
21	Opis protokołu „PrzekażInformacje” dla roli odbiorcy.....	22
22	Opis protokołu „ZarejestrujGrupę” dla roli założyciela.....	22