AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PRACA DYPLOMOWA INŻYNIERSKA

pt.

„Implementacja aplikacji na urządzenia mobilne do projektowania oraz wizualizacji lotu Quadcoptera z wykorzystaniem usługi Google Maps”

Imię i nazwisko dyplomanta: **Kamil Dudek**

Kierunek studiów: **Informatyka Stosowana**

Nr albumu: **268935**

Promotor: prof. dr hab. inż. Łukasz Rauch

Recenzent: prof. dr inż. Krzysztof Regulski

Podpis dyplomanta: Podpis promotora:

Kraków 2016

„Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej "sądem koleżeńskim"”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.”

Kraków, dnia …… Podpis dyplomanta…………….

**Spis treści**

1. **Wprowadzenie**
2. **Problem znalezienia optymalnej drogi**
3. **Cel pracy**
4. **Rozwiązania obecnie istniejące na rynku**
   1. **Google Maps**
   2. **OMPL – The Open Motion Planning Library**
   3. **OSMBuildings**
   4. **Ogólne zestawienie**
5. **Rozwiązanie problemu**
   1. **Definicja mapy**
   2. **Przekształcenie mapy w strukturę grafu**
   3. **Algorytm A\***
   4. **Wizualizacja**
   5. **Integracja z usługą Google Maps**
6. **Wyniki**
7. **Wnioski**
8. **Wprowadzenie**

Wraz z rozwojem technologicznym jedynie kwestią czasu było stworzenie małych, latających urządzeń, sterowanych zdalnie. Dziś są one znane wszystkim jako drony, tudzież Quadcoptery. Ich cechą jest stosunkowo niewielki koszt produkcji oraz szeroki wachlarz możliwości, dzięki którym w łatwy sposób mogą zastąpić tradycyjne rozwiązania. Już teraz słyszy się o wdrażaniu bezzałogowych statków powietrznych jako wsparcie w akcjach ratunkowych GOPR. Mówi się także o wykorzystaniu ich do rozsyłania paczek w obrębie danej aglomeracji, co już zaczyna testować firma Amazon w swoim programie Amazon Prime Air. Dostarczanie pizzy z wykorzystaniem Quadcoptera rozważa również amerykańska sieć gastronomiczna Domino’s Pizza. Wszystkie te działania jednoznacznie wskazują, iż próby upowszechnienia rozwiązań opartych o drony są sukcesywnie dokonywane, co świadczy o atrakcyjności tego rozwiązania.

W związku z dynamicznym rozwojem strefy bezzałogowych statków powietrznych, w 2014 roku w Kole Naukowym Metalsoft powstał projekt pt. „Quadcopter – Inteligentna poczta”, którego celem było zbudowanie drona oraz stworzenie dedykowanego systemu, który pozwoliłby na autonomiczne sterowanie Quadcopterem. System ten ma samodzielnie planować trasę lotu w oparciu o zbudowaną mapę przestrzeni. Dodatkowo musi on reagować na pojawiające się wokół niego przeszkody, by uniknąć nieprzewidzianych kolizji. Docelowo projekt ten miał za zadanie zbadać użyteczność dronów, które pełniłby funkcję inteligentnej poczty – byłyby w stanie samodzielnie dostarczyć przesyłkę w obrębie zadanego, znanego obszaru. Projekt ten, poza oczywistym zbudowaniem samego drona, musiał się skupić na trzech podstawowych aspektach. Pierwszym jest odpowiedni sposób komunikacji w celu osiągnięcia zdalnego sterowania dronem z poziomu aplikacji mobilnej. Drugim jest odpowiednie rozpoznawanie przeszkód znajdujących się wokół Quadcoptera na podstawie danych z kamer oraz dalmierzy, a także wczytanie ich do mapy przestrzeni systemu. Trzecim, ostatnim aspektem, jest stworzenie odpowiedniej reprezentacji wcześniej wspomnianej mapy przestrzeni oraz zaprojektowanie optymalnej trasy lotu drona, wraz z jej odpowiednią wizualizacją i integracją z usługą Google Maps. Zadaniem tej pracy inżynierskiej jest opisanie właśnie tego ostatniego, trzeciego aspektu wspomnianego już projektu.

1. **Problem znalezienia optymalnej drogi**

Pojawienie się na rynku pierwszych robotów wszelkiej maści, poczynając od tych poruszających się drogą lądową i kończąc na tych manewrujących w przestrzeni powietrznej, od razu zainspirowało programistów do nadania im funkcjonalności automatycznego planowania trasy do wyznaczonego celu. Inspiracja ta zrodziła problem, który uogólniając możemy nazwać problemem znalezienia optymalnej drogi. Aby go rozwiązać, należało przede wszystkim stworzyć odpowiednią reprezentację samej przestrzeni, w której mógłby działać odpowiednio przygotowany algorytm znajdujący pożądaną trasę lotu. Istnieje wiele podejść do reprezentacji przestrzeni znajdującej się wokół danego urządzenia.

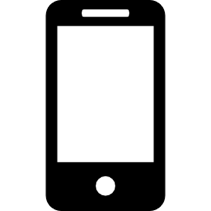
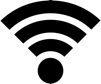
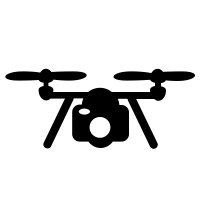
Pierwszym z nich jest zbudowanie modelu wektorowego. Polega on na interpretacji każdego potencjalnego elementu za pomocą uporządkowanych zbiorów współrzędnych, które jednocześnie definiują jego położenie oraz kształt. Model ten traktuje każdy możliwy element mapy jako byt samodzielny, opisując go za pomocą punktów, linii i powierzchni. Model ten nie jest zbytnio czytelny dla człowieka, gdyż jego zapis jest bardzo skomplikowany, szczególnie w przypadku bardziej złożonych obiektów. Wymaga on również bardzo dużej analizy przestrzennej, by móc odpowiednio definiować zbiory opisujące poszczególne elementy znajdujące się na mapie. Wady te zdecydowanie rekompensuje zdecydowanie mały rozmiar danych reprezentujących mapę, co umożliwia utworzenie bardzo szczegółowej mapy przy jednocześnie stosunkowo jej małym rozmiarze. Narzut obliczeniowy wymagany jest jedynie podczas generacji mapy, co związane jest z wspomnianą już koniecznością zaawansowanej analizy przestrzeni.

Drugim, konkurencyjnym podejściem do reprezentacji przestrzeni, jest zbudowanie modelu komórkowego. Model ten mając cechy obrazu cyfrowego często nazywany jest także Pixel-Map’ą. Polega on na odpowiednim podziale obszaru na komórki zazwyczaj o kwadratowym kształcie, tworząc tym samym siatkę najczęściej przypominającą rozległą szachownicę. Każda z takich komórek przechowuję informację o konkretnym obszarze, który obejmuje swoją wielkością. Komórki te tworzą swego rodzaju macierz, do której elementów możemy odwoływać się za pomocą zwykłych indeksów. Podejście takie jest zdecydowanie bardziej czytelne dla człowieka, gdyż wynik końcowy można w łatwy sposób zwizualizować. Nie wymaga on również specjalnie zaawansowanej analizy przestrzeni, co generuje korzyści w postaci mniejszego narzutu obliczeniowego. Niestety zalety te generują swojego rodzaju koszty, do których zalicza się zdecydowanie większy rozmiar tak zaprezentowanej mapy. Również jej szczegółowość jest dużym stopniu uzależniona od tak zwanej rozdzielczości mapy, czyli wielkości tworzącej jej komórek.

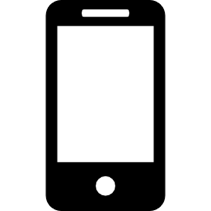
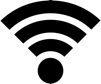
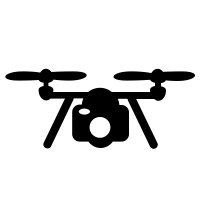
Jak można łatwo zauważyć, sam wybór odpowiedniej reprezentacji terenu jest stosunkowo trudnym wyborem. Należy sobie uświadomić, iż wybór ten będzie odzwierciedlał się również na poziom trudności w rozwiązaniu problemu znalezienia optymalnej drogi. Zauważmy, że mimo zalet znacznie optymalniejszego pod kątem pamięciowym modelu wektorowego, który pozwala z dużą dokładnością przechowywać informację o terenie, narzut obliczeniowy związany z zaawansowanymi technikami analizy przestrzeni znacząco utrudnia znalezienie pożądanej ścieżki. Postać wektorowa reprezentowanych danych powoduje nie trywialność rozwiązania problemu. Nie oznacza to jednak, iż rozwiązanie takie jest niemożliwe do uzyskania, wymaga on jednak zdecydowanie znacznie większych zasobów obliczeniowych niż tymi, którymi dysponuje sprzęt kierujący danym robotem. Sytuacja jest całkowicie odwrotna w przypadku modelu komórkowego. Model ten dzięki bardzo prostym analizom przestrzeni oraz bardzo przystępnej reprezentacji danych sprawia, iż liczba podejść do rozwiązania takiego problemu jest nieograniczona. Dzięki temu możemy tak wybrać algorytmy, aby ich złożoność obliczeniowa była adekwatna do sprzętu, na którym mają pracować.

1. **Cel pracy**

Celem omawianej pracy jest stworzenie odpowiedniej reprezentacji mapy, która byłaby przystępna dla systemu działającego na Quadcopterze. Na tak zdefiniowanej mapie potrzeba znaleźć optymalną ścieżkę lotu, którą podążałby dron. Należy zwrócić uwagę, iż system ten jest już obecnie zajęty takimi działaniami jak stabilizacja oraz kontrola lotu. Fakt ten wymusza na nas znalezienie rozwiązania które nie wymaga zbyt wielu zasobów obliczeniowych, dlatego też zdecydowano o tym, aby większość obliczeń związanych ze znajdywaniem drogi była przeniesiona na aplikację mobilną, która jedynie wysyłałby do systemu znajdującego się na Quadcopterze wyliczoną już ścieżkę.



a)



b)

*Rys. 1. Możliwe scenariusze komunikacji pomiędzy dronem a aplikacją mobilną: a) scenariusz, w którym aplikacja mobilna wyznacza drogę i przesyła ją do Quadcoptera, b) scenariusz, w którym dron napotyka przeszkodę i prosi o aktualizację trasy.*

Jak widzimy na Rys. 1, istnieje wiele możliwych scenariuszy komunikacji pomiędzy dronem a aplikacją mobilną. Na rys. 1. a) widzimy podstawowy scenariusz, w którym aplikacja mobilna znajduje optymalną drogę i wysyła ją do Quadcoptera. Należy jednak pamiętać, że równie często może pojawić się drugi scenariusz, przedstawiony na rys. 1. b). Przedstawia on bowiem scenariusz, w której to dron po napotkaniu przeszkody potrzebuje natychmiastowej aktualizacji trasy. Pytanie które samoistnie nasuwa się w tym momencie, dotyczy sposobu jej aktualizacji. Czy w takim przypadku dron powinien wysłać prośbę do aplikacji mobilnej? A co w przypadku, gdy aplikacja jest nieosiągalna z powodu kłopotów z siecią lub po prostu jest wyłączona? Możliwe, że znacznie bezpieczniejszym scenariuszem byłoby wykonanie aktualizacji trasy samodzielnie przez drona. Rozwiązanie to jednak wymagałoby utrzymywania dwóch wersji implementacji, co może być trudne w późniejszym etapie implementacji. Jakkolwiek są to realne scenariusze które należy rozpatrzyć, praca ta w głównej mierze skupia się na pierwszym scenariuszu, przedstawionym na rys. 1. a).

Podejście ścisłej współpracy pomiędzy dronem a aplikacją mobilną umożliwia nam dodatkowo odpowiednią wizualizację mapy, która pomogłaby końcowemu użytkownikowi wybrać początek oraz koniec trasy, a także zobrazowała mu wyznaczoną ścieżkę lotu. Dodatkowo zyskujemy możliwość łatwej integracji z dodatkowymi usługami takimi jak Google Maps, którą można wykorzystać w celu zainicjalizowania początkowego stanu mapy budynkami otrzymanymi za pośrednictwem tej usługi, co odciąża również samego drona od konieczności początkowego rozpoznawania całego terenu. Jako urządzenia, dla której przeznaczona jest wspomniana aplikacja, wybrano te z systemem Android.

1. **Rozwiązania obecnie istniejące na rynku**

Przed przystąpieniem do realizacji założonego celu, należało dokonać odpowiedniego

przeglądu literatury. Obecnie dostęp do map nie jest już utrudniony, przez co na rynku istnieje spora liczba gotowych i darmowych rozwiązań, z których każdy może skorzystać. Należy jednak przejrzeć je pod kątem integracji z urządzeniami autonomicznie się poruszających, a także urządzeń latających. W tym celu wybrano 3 najpopularniejsze serwisy udostępniających tego rodzaju usługę.

* 1. **Google Maps**

Serwis Google Maps obecnie jest najpopularniejszą usługą tego typu na rynku. Aplikacja z tą usługą jest dostarczana na każdym smartfonie z zainstalowanym system Android. Jej szeroki wachlarz możliwości zrzesza coraz większą liczbę zadowolonych użytkowników. Możemy z jej pomocą poza standardowym przeglądaniu mapy, znaleźć interesującą nas trasę do konkretnego celu. Pod tym względem Google Maps na tle konkurencji wyróżnia się tym, iż podczas wyznaczania trasy bierze pod uwagę informacje o obecnie panującym ruchu na drodze, co umożliwia wielu kierowcom skutecznie omijać korki. Poza możliwością planowania trasy samochodowej, usługa ta umożliwia również zaplanowanie trasy rowerowej czy też pieszej. Ponadto Google Maps posiada wiele autorskich podsystemów. Jednym z nich jest Street View – usługa umożliwiająca tak zwane „wirtualne spacery” po konkretnym fragmencie mapy. Bardzo korzystnym aspektem jest też ostatnio dodana możliwość pracy w trybie „offline”. Poza oczywistymi udogodnieniami warto też zwrócić uwagę na same mapy. Są one bardzo dobrej jakości, co jest spowodowane poprzez bieżące aktualizację. Również i same zdjęcia satelitarne są wysokiej rozdzielczości. Ponadto w obrębie większych aglomeracji coraz częściej można spotkać budynki, które możemy oglądnąć w 3D, co oznacza iż mają już wczytaną odpowiednią wysokość budynków. Funkcjonalność ta występuje jeszcze na niewielkich obszarach – przykładowo w Krakowie budynki 3D możemy zaobserwować jedynie w okolicach Rynku Głównego, pozostała część miasta nie posiada już takiej możliwości.

Od strony technicznej, Google Maps udostępnia szeroki wachlarz wspieranych platform. Usługę tą z łatwością możemy zintegrować nie tylko na systemach z systemem Android, ale również na konkurencyjnych systemach iOS. Integracja nie stanowi także problemu dla programistów webowych – tutaj mamy przystępne biblioteki dla języka JavaScript. Warto w tym miejscu zaznaczyć, iż Google Maps tak naprawdę można postrzegać jako tak zwany Web-Service. Wszystkie operacje na mapach w każdej aplikacji wymagają połączenia z serwerem Google Maps, który to przetwarza otrzymane zapytania. Dlatego też teoretyczna integracją z tym systemem jest możliwa z praktycznie każdego obecnie dostępnego języka programowania – każdy współczesny język umożliwia wysyłanie zapytań typu REST do zewnętrznych serwisów. Należy jednak wspomnieć, iż nie jest to usługa w pełni darmowa. Do momentu, gdy nie przekroczymy pewnego progu limitu zapytań, jest to usługa bezpłatna, jednak gdy przekroczymy ustalone limity, musimy uiścić odpowiednią kwotę. Limity te definiowane są w postaci „liczba zapytań na dzień”, gdzie liczba zapytań dotyczy konkretnej aplikacji, a nie konkretnego użytkownika. Aby skorzystać z usługi Google Maps, każda aplikacja musi skorzystać z wcześniej uzyskanego klucza dostępu, co oznacza że wszystkie egzemplarze tej aplikacji korzystają z tego samego klucza. Już tylko przy 25 000 użytkownikach, każdy z nich będzie mógł wykonać średnio jedno zapytanie dziennie, co jest bardzo małym limitem, który wymusza na właścicielach aplikacji zakupu dodatkowych pakietów.

* 1. **OMPL – The Open Motion Planning Library**

Następną tego rodzaju usługą, tym razem już znacznie mniej znaną, jest OMPL. Skrót ten pochodzi od „Open Motion Planning Library”. Jak sama nazwa wskazuje, jest to biblioteka przeznaczona do planowania drogi. Biblioteka ta jest jedynie ograniczona do algorytmów planowania drogi, co oznacza że nie uwzględnia ona żadnych specyfikacji dotyczących środowiska, wykrywania kolizji czy też wizualizacji. Jest to jednak zabieg zamierzony, gdyż biblioteka ta jest zaprojektowana do łatwej integracji z systemami które już zapewniają dodatkowe komponenty [1]. Biblioteka ta jest w całości zaimplementowana w języku C++ i oferuje bogate zaplecze algorytmiczne. Algorytmy wykorzystywane w tej bilbiotece to algorytmy oparte o próbkowanie przestrzeni. Fundamentalną ideą takich algorytmów jest aproksymacja połączeń w przeszukiwanej przestrzenii za pomocą strukry grafu. Przestrzeń może być próbkowana w różny, zależny od algorytmu sposób, a wybrane próbki tworzą wierchołki grafu aproksymującego [2]. Algorytmy te zaliczają się do klasy algorytmów probabilistycznych, co oznacza, że działają z bardzo wysokim prawdobodobieństwem znalezienia rozwiązania, jednak rozwiązanie to nie jest optymalne. Cechuje je wysoka wydajność obliczeniowa oraz szybkość działania.

Bliblioteka ta zyskała powodzenie w dziedzienie robotyki. Jest ona dostarczana wraz z systemem ROS – Robot Operating System, który jest często instalowany na różnego rodzaju robotach. Ważną zaletą tej biblioteki jest stosunkowo łatwa możliwość rozszerzania i dodwania własnych algorytmów. Jendakże biblioteka ta nie wizualizuje w żaden sposób znalezionej drogi. Nie integruje się ona również ze standardowymi mapami, co zrzuca obowiązek integracji z istniejącymi mapami na programistę.

* 1. **OSMBuildings**

OSMBuildings jest usługą oferującą dostęp do map. Skrót OSM oznacza „Open Street Map”, co już na wstępie sugeruje, że system ten bazuje na mapach pochodzących z projektu „OpenStreetMap”. Projekt ten jest w pełni otwarty, tworzony przez dużą społeczność, która codziennie aktualizuje mapy w oparciu nie tylko o zdjęcia satelitarne i dane GPS, ale także o własne, indywidualne doświadczenia. OSMBuildings oferują dokładną reprezentację 3D budynków pobranych z projektu OSM. Budynki znajdujące się w tym systemie cechują się wysoką dokładnością, zarówno w aspekcie wysokości budynków, ich kształtu jak i nawet rozmiarem i kształtem okien. Podstawową wadą tej usługi jest utrudniona integracja z innymi systemami. Cały system jest stworzony jako projekt webowy, co byłoby bardzo ciężko zintegrować z urządzeniem mobilnym. Mimo utrudnionej integracji usługa ta udostępnia nam Web-Service, który, podobnie jak w przypadku Google Maps, jest w stanie zwrócić nam interesujące nas obiekty za pomocą zapytania REST, jednak takie rozwiązanie zrzuca na programistę obowiązek ręcznej wizualizacji mapy. Największą wadą tego systemu jest kompletny brak jakiegokolwiek mechanizmu planowania drogi – udostępnia on jedynie ogromną bazę danych budynków, które są opisywane bardzo szczegółowo.

* 1. **Ogólne zestawienie**

Jak widzimy, obecnie na rynku znajduje się wiele rozwiązań dostarczających skuteczną reprezentację mapy wraz z mechanizmami planowania trasy. Jednakże patrząc w perspektywie omawianej pracy, nie można znaleźć systemu który spełniałby postawione wymagania. Rozwiązanie, które jest poszukiwane, to system integrujący się z istniejącymi już mapami, który byłby w stanie wyznaczyć optymalną trasę przelotu Quadcoptera. Zauważmy, iż usługa Google Maps pomimo zaawansowanych technik planowania trasy, nie posiada opcji planowania drogi dla dronów. Opcja taka mogłaby uwzględniać fakt, iż urządzenie nie musi poruszać się wyłącznie wzdłuż ulic, ale również nad budynkami i innymi obiektami. Wymagałoby to natomiast informacji o wysokościach budynków, które Google Maps na chwilę obecną nie ma skompletowanej – jedynie niektóre budynki mają informację o swojej wysokości. Z kolei rozwiązanie zaproponowane przez bibliotekę OMPL jest bardzo atrakcyjne pod względem algorytmów znajdowania drogi. Zauważmy jednak, iż biblioteka ta to sam zestaw algorytmów. Nie oferuje ona żadnej integracji z istniejącymi mapami czy też wizualizacji znalezionej drogi. Ponadto jest to biblioteka całościowo stworzona w języku C++, co utrudniałoby jej wykorzystanie na platformie Android, w której naturalnym językiem programowania jest Java. Natomiast OSMBuildings wraz ze swoją bardzo bogatą bazą danych na temat budynków nie posiada żadnego mechanizmu planowania trasy. Wszystkie te aspekty powodują, że przy obecnie istniejących rozwiązaniach nie jesteśmy w stanie wybrać żadnego z nich. Pójście na kompromis w którąkolwiek ze stron jest nieakceptowalne z punktu widzenia projektu „Quadcopter – Inteligentna poczta”. Projekt ten bowiem narzuca na takie oprogramowanie wymagania w postaci wizualizacji mapy oraz wyznaczonej ścieżki dla użytkownika, dobrej integracji z zewnętrznymi usługami oraz urządzeniami mobilnymi a także wyznaczanie trasy lotu Quadcoptera również nad budynkami, co wymaga znajomości ich wysokości. W związku z tym zdecydowano się na implementację własnego rozwiązania, które byłoby pewnego rodzaju hybrydą stojącą pośrodku wyżej wymienionych usług.

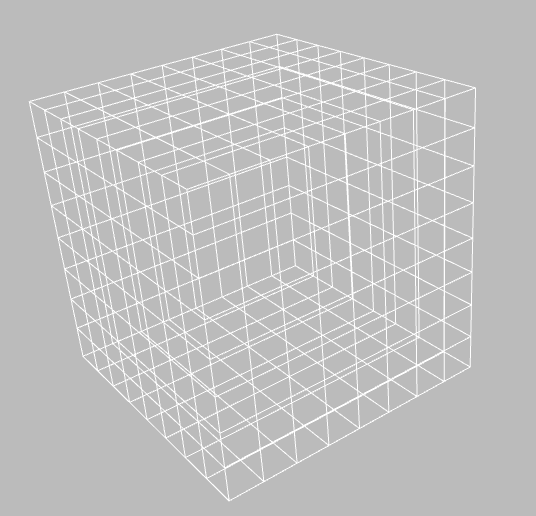
1. **Rozwiązanie problemu**

Rozwiązaniem problemu zdefiniowanego przez wymagania projektu „Quadcopter -

Inteligentna poczta” jest aplikacja mobilna zaprojektowana na urządzenia z systemem Android. Jej implementacja wymagała rozwiązania szeregu wyżej przedstawionych problemów w taki sposób, aby jej użyteczność była wystarczająca z punktu widzenia wyżej wymienionego projektu. Językiem programowania, który wykorzystano do implementacji była Java, która jest językiem naturalnym dla wybranej platformy mobilnej. Oprogramowanie zostało tworzone przy użyciu framework-u Android SDK w wersji 24, co oznacza, iż poziom kompilacji jest zgodny z systemami Android Nougat (inaczej Android 7.0). Jako iż w momencie implementacji był to system dopiero wdrażany do produkcji, jedynie niewielka liczba urządzeń na rynku posiadała najnowszy system. Aby zachować więc zgodność ze większością urządzeń działających z tym systemem, zdecydowano się na kompatybilność wsteczną aż do wersji 16 framework-u Android SDK. Oznacza to, że aplikacja będzie w pełni funkcjonować na systemach Android począwszy od wersji Android Jelly Bean (inaczej Android 4.1), co łącznie na dzień 7 listopada 2016 roku stanowiło aż 97.3% wspieranych urządzeń z systemem Android znajdujących się na rynku [3]. Aplikacja była tworzona w całości w IDE o nazwie „Android Studio”. Po zdefiniowaniu wymagań systemowych przystąpiono do implementacji.

* 1. **Definicja mapy**

Kluczowym elementem implementacji był dobór odpowiedniej reprezentacji mapy. Mapa ta musi przede wszystkim musi mieć postać łatwo czytelną zarówno dla drona, jak i dla człowieka. Dodawanie nowych elementów do tej mapy nie może być skomplikowane obliczeniowo, ponieważ dron może wykonywać takie operacje na bieżąco, podczas lotu. Jest to związane z tym, że ewentualne wykrycie nowej przeszkody musi zostać szybko uwzględnione w logicznej reprezentacji mapy. Z tego powodu zdecydowano się wzorować na tak zwanej PixelMap-ie. Opiera się ona w głównej mierze na rastrowej reprezentacji obrazu znanej z grafiki komputerowej. Podejście to przede wszystkim opiera się na założeniu, że teren pozbawiony jest naturalnych wypukłości i jest płaski. Teren ten następnie jest dzielony na małe kwadraty o ustalonym rozmiarze, którym przyporządkowane są liczby naturalne. Kwadraty te można porównać do pixel-i w rastrowym obrazie. Jeżeli do kwadratu przypisana jest liczba „zero”, oznacza to, że obszar ten jest wolny, czyli nie ma w tym miejscu żadnej przeszkody, tak więc dron może przelecieć przez ten kawałek przestrzeni. Jeżeli przypisana wartość jest większa od zera, oznacza to, że przez ten obszar dron nie jest w stanie się przemieścić, gdyż znajduje się tam jakaś przeszkoda. Jako iż głównymi przeszkodami drona będą budynki, przyjęto, iż liczba ta będzie prezentować wysokość budynku, do którego przynależy dany fragment obszaru. Jednakże w odróżnieniu od PixelMap-y (a także od obrazu rastrowego), których reprezentację można logicznie zapisać w formie liczbowej macierzy dwuwymiarowej, z naturalnych potrzeb tego projektu mapa jest prezentowana w formie trójwymiarowej macierzy. Forma taka jest wymagana, aby uwzględnić możliwość przelotu nad danym budynkiem. W efekcie powstała mapa przypomina nałożenie na siebie kilkunastu dwuwymiarowych PixelMap, które razem składają się w sześcian ułożony z wielu, mniejszych sześcianów o takich samych wymiarach.



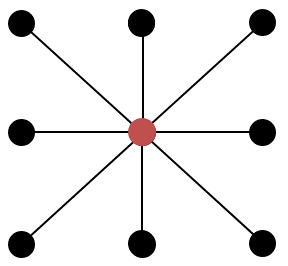
*Rys. 2. Przykładowa wizualizacja trójwymiarowej PixelMap-y. Podejście trójwymiarowe w dużym stopniu przypomina podział przestrzeni na szereg małych sześcianów, z których każdy posiada określony stan.*

Każdy pojedynczy element jest identyfikowany za pomocą trzech współrzędnych, przy czym początek układu współrzędnych znajduje się w prawym, dolnym rogu mapy. W implementacji mapa ta jest przedstawiona jako trójwymiarowa tablica typu Integer, zawierających informację o stanie danej komórki. Ponadto stworzono dodatkową warstwę abstrakcji reprezentującej budynek. Budynek ten jest opisywany poprzez podstawowe parametry, jakie jak jego wysokość, szerokość oraz długość. Ponadto posiada informację o współrzędnych punktach, które go tworzą, dzięki czemu można go w łatwy sposób odnieść do mapy, na której został zdefiniowany. Dzięki takiej strukturze Quadcopter jest w stanie w łatwy i przede wszystkim szybki sposób dodać nowy budynek w momencie rozpoznania oraz zidentyfikowania go. Wystarczy bowiem, iż zdefiniuje on jedynie ciąg współrzędnych, na których znajduje się przeszkoda, co już w zupełności wystarczy do zaaktualizowania mapy.

* 1. **Przekształcenie mapy w struturę grafu**

Kolejnym krokiem w implementacji aplikacji było stworzenie mechanizmu, który byłby w stanie odnaleźć w wyżej przedstawionej reprezentacji terenu optymalną ścieżkę. Aby tego dokonać należało odpowiednio zinterpretować wcześniej stworzoną mapę, tak aby mechanizm znajdywania drogi opowiednio mógł na niej operować. Zdecydowano się na konwersję trójwymiarowej przestrzenii na graf ważony. W ten sposób można z łatwością wykorzystać całą gamę algorytmów operujących na grafach, które wyznaczają najkrótszą ścieżkę w grafie. Przekształcenie to odbywa się w bardzo prosty sposób – każdy element mapy konwertowany jest na wierzchołek grafu, natomiast wagi połączeń między wierzchołkami stanowi koszt przelotu do sąsiedniego wierzchołka. Kluczowym elementem jest struktura takiego wierzchołka. Każdy z tych wierzchołków w swojej strukturze przechowuje następujące informacje:

* Lista sąsiedztwa: każdy wierzchołek przechowuje w liście referencje do najbliższych, otaczających go wierzchołkach
* Wagi połączeń: każdy wierzchołek jest w stanie określić wagę połęcznia (czyli koszt przejścia) do wierchołka sąsiedniego.



*Rys. 3. Sąsiedztwo dla wierzchołka na poziomie 2D. Dla danego przykładu, wierzchołek posiada jeszcze 9-ciu sąsiadów nad sobą oraz 9-ciu sąsiadów pod sobą, co łącznie daje 26-ciu sąsiadów.*

* Referncję do rodzica: w przypadku, gdy wierzchołek ten będzie należał do ścieżki, będzie on przechowywał w sobie referncję do swojego rodzica. Rodzic określa wierzchołek poprzedzający w wyznaczonej ścieżce. Przykładowo, rodzicem końcowego wierzchołka będzie przedostatni wierzchołek ścieżki, którego rodzicem będzie poprzedni wierzchołek, aż do wierzchołka startowego. Wierzchołek startowy, jako jedyny wierzchołek w ścieżce, nie posiada ustawionej referencji do rodzica.

*Rys. 4. Przykładowa strukutra ścieżki w grafie. Wierzchołek zielony jest punktem początkowym, wierzchołek czerwony jest punktem końcowym, czarne wierzchołki reprezentują punkty pośrednie tworzącę ścieżkę. Kierunek strzałki wskazuje rodzica wierzchołka, od którego strzałka wychodzi.*

* Przechodniość wierzchołka – informuje o tym, czy przez dany wierzchołek można przejść, czy nie (bo przykładowo wierzchołek ten należy do budynku lub innej przeszkody, przez którą nie można przejść)

Taka strukutra reprezentująca wierzchołek pozwala w łatwy sposób przechowywanie informacji o sąsiadach konkretnego wierzchołka oraz koszcie połączeń pomiędzy nimi. Dzięki takiej strukutrze jesteśmy także w łatwy sposób ocenić, czy dany wierzchołek należy do ścieżki, poprzez sprawdzenie referncji do jego rodzica. W zaproponowanym podejściu koszty połączeń pomiędzy wierzchołkami dobierany jest na podstawie energii potrzebnej na pokonanie danej odległości. Przykładowo, na podstawie testów doświadczalnych wywnioskowano, że znacznie większa energia potrzebna jest na podniesienie Quadcoptera w górę niż na przelot w lini prostej, nie zmianiając jego wysokości, stąd też waga połączenia pomiędzy wierzchołkiem znajdującym się ponad obecnie rozpatrywanym punktem będzie większa od wagi połączenia pomiędzy wierzchołkiem znajdującym się na tej samej wysokości.

* 1. **Algorytm A\***

Kluczowym etapem całego projektu jest odpowiednie znajdywanie optymalnej drogi przelotu Quadcoptera, co w gruncie rzeczy, dzięki wyżej opisanej konwersji, sprowadziło się do znalezienia najkrótszej ścieżki w grafie. Początkowo polanowano wykorzystanie do tego celu popularnego algorytmu Dijkstry. Zapewnia on, że znaleziona ścieżka jest zawsze ścieżką najoptymalniejszą, co gwarantuje poprawność rozwiązania. Jednakże dalsza analiza kosztu obliczeniowego wykazała, iż metoda ta ma zbyt dużą złożoną obliczeniową dla grafów gęstych, do których zalicza się również graf wygenerowany na podstawie mapy terenu. Działanie tego algorytmu opiera się bowiem na przetwarzaniu każdego możliwego wierzchołka grafu w celu znalezienia optymalnej ścieżki. Powoduje to, iż w takim przypadku zyskujemy kwadratową złożoność obliczeniową, co onzacza, iż liczba operacji potrzebnych do znalezienia ścieżki jest wprost proporcjonalna do kwadratu liczby danyh wejściowych. Tak duża złożoność z punktu widzenia projektu jest niedopuszczalna. Algorytm znajdywania drogi powinien zostać tak dobrany, aby jego złożoność obliczeniowa była jak najmniejsza. Nie oznacza to, że złożoność takiego algorytmu jest nieakceptowalna dla urządzeń mobilnych – czterordzeniowe procesory w smartfonach dzisiaj nie są już dla nikogo zaskoczeniem, więc jego odpowiednie zrównoleglenie powinno zapewnić satysfakcjonujące wyniki. Jednakże w perspektywie, w której sam Quadcopter zmuszony by był do samodzielnego przeliczenia drogi (na przykład w przypadku utraty komunikacji z kontrolerem), taka złożność mogłaby się okazać krytyczną dla mikrokomputera kontrolującego cały system lotu drona. W związku z tym zaczęto poszukiwanie innej metody znajdywania najkrótszej ścieżki w grafie, czego wynikiem okazał się algorytm A\* (A-Star). Jest to powszechnie wykorzystywana metoda we wszelkiego rodzaju grach komputerowych, w których należy w szybki sposób znaleźć drogę w tak zwanych mapach kafelkowych. Algorytm ten zalicza się do algorytmów heurystycznych, którego wynik jest zupełny i optymalny, co daje nam gwarancję znalezienia ścieżki jeśli tylko taka istnieje, przy czym ścieżka ta jest optymlana. Przy odpowiednio zdefiniowanej heurystyce, algorytm ten jest dopuszczalny (tzn. zawsze dostarcza prawidłowe rozwiązanie), ponadto przeszukuje najmniej węzłów spośród pozostałych algorytmów dopuszczalnych. Przy odpowiednio zdefiniowanej funkcji heurystycznej (tzn. w taki sposób, by odchylenie jej wartości od faktycznej odległości od celu było nie większe niż logarytm z liczby określającej wartość tej odległości), algorytm ten charakteryzuje się również optymalną złożonością obliczeniową klasy wielomianowej. Dzięki odpowiednio przyjętej heurystyce, złożoność tego algorytmu znacząco się zmniejsza.

Algorytm A\* podczas obliczania kosztu ścieżki bazuje na prostej funkcji kosztu przebycia drogi, która przyjmuje następującą wartość:

Przy czym:

* – funkcja opisująca koszt przebycia z wierzchołka startowego do wierzchołka aktualnie rozpatrywanego
* – funkcja heurystyczna (tzw. heurystyka), przedstawiającą szacunkową odległość od wierzchołka aktualnie rozpatrywanego do wierzchołka docelowego. Dla potrzeb aplikacji do wyznaczenia odległości przyjęto metrykę euklidesową.

Funkcja ta jest musi być wyznaczana dla każdego wierzchołka osobno, co w implementacji rozwiązano poprzez przeniesienie logiki obliczania wartości tej funkcji do klasy reprezentującej wierzchołek. Dzięki temu jesteśmy w stanie określić koszt ścieżki dla każdego wierzchołka podając jedynie referencję do punktu początkowego oraz końcowego.

Podstawowa zasada alogrytmu A\* opiera się na minimializacji wyżej przedstawionej funkcji reprezentującej koszt przebycia drogi. Ponadto, algorytm ten podczas działania przechowuje informację o dwóch listach – otwartej oraz zamkniętej liście wierzchołków. Otwarta lista wierzchołków zawiera wszystkie wierzchołki, które mogą (ale nie muszą) być dołączone do szukanej ścieżki w następnym kroku. Zamknięta lista wierzchołków reprezentuje wierzchołki, które już zostałe przeszukane, co pozwala uniknąć wielokrotnego rozpatrywania pojedynczego wierzchołka. Na samym początku punktu startowy jest dodawny do listy otwartych wierzchołków. Pojedynczy przebieg algorytmu składa się z następujących kroków:

1. Z listy otwartych wierzchołków wybierz ten, którego wartość funkcji jest najmniejsza. Usuń go z listy otwartych wierzchołków i dodaj go do listy zamkniętych wierzchołków. Oznacz go jako „obecnie rozpatrywany”.
2. Sprawdź sąsiedztwo wierzchołka „obecnie rozpatrywanego”.
   1. Jeżeli sąsiad nie należy do listy zamkniętych wierzchołków, jest przechodni oraz nie należy jeszcze do listy otwartych wierzchołków, dodaj go do tej listy. Ustaw rodzica dodanego sąsiada na obecnie rozpatrywany wierzchołek.
   2. Jeżeli sąsiad obecnie znajduje się na liście otwartych wierzchołków, sprawdź czy ścieżka od punktu początkowego do tego wierzchołka nie jest korzystniejsza. Aby tego dokonać, sprawdź wartość funkcji dla tego wierzchołka, reprezentującej koszt dotarcia do niego od wierzchołka startowego. Jeżeli wartość dla tego wierzchołka będzie mniejsza od wartości dla wierzchołka obecnie rozpatrywanego, ustaw jego rodzica na wierzchołek obecnie rozpatrywany oraz przelicz ponownie wartości funkcji .

Alogrytm kontynuuje swoje poszukiwanie do momentu, aż wierzchołek reprezentujący punkt końcowy nie zostanie oznaczony jako „obecnie rozpatrywany” oraz na lista wierzchołków otwartych nie jest pusta. W momencie osiągnięcia wierzchołka końcowego, ścieżka do niego od punktu początkowego zostaje znaleziona. Aby ją wyznaczyć wystarczy jedynie „wrócić się” do punktu początkowego, zaczynając od punktu końcowego, przechodząc poprzez referencję do rodzica wierzchołka, aż wierzchołek startowy nie zostanie osiągnięty.

* 1. **Wizualizacja**

Implementacja algorytmu szukającego drogi było kluczowym elementem pracy, jednak nie był to ostatni jej etap. Aby umożliwić użytkownikom przystępny sposób reprezentacji znalezionego rozwiązania, należało zająć się wizualizacją zarówno stworzonej mapy, jaki i znalezionej ścieżki. Wizualizacja powinna w łatwy dla użytkownika sposób przedstawić trójwymiarową przestrzeń, w której porusza się Quadcopter. Użytkownik powinien być w stanie dowolnie nią obracać, przybliżać lub oddalać. Jako iż aplikacja służy do znajdywania drogi, użytkownik musi mieć także możliwość do wprowadzania współrzędnych punktu startowego oraz końcowego.

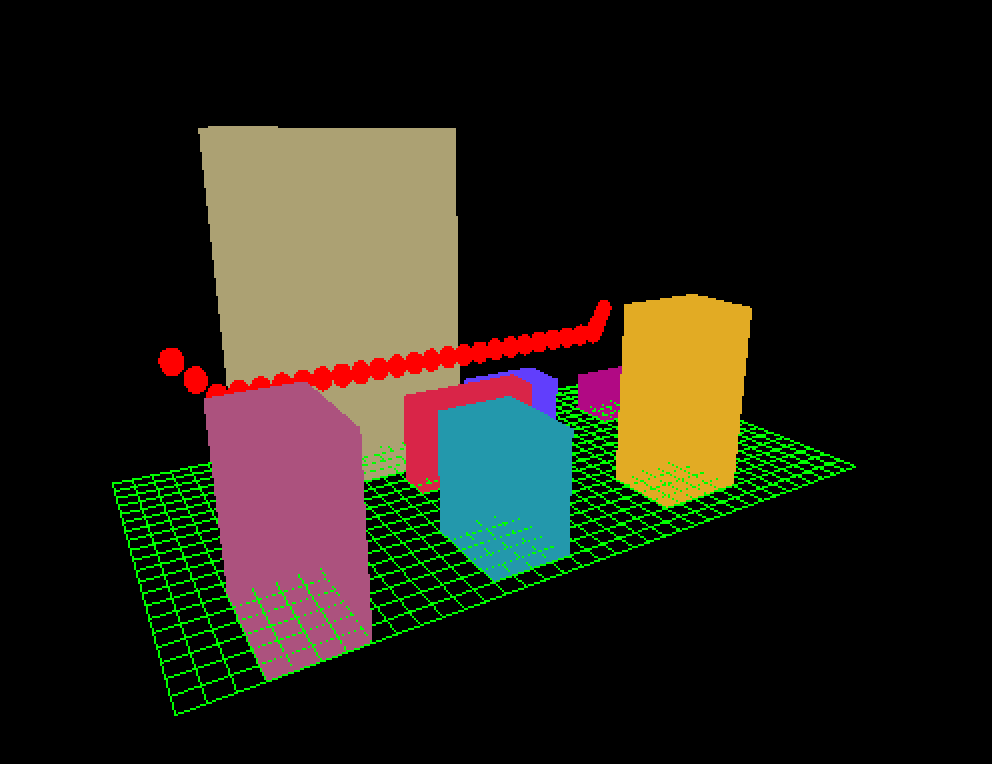
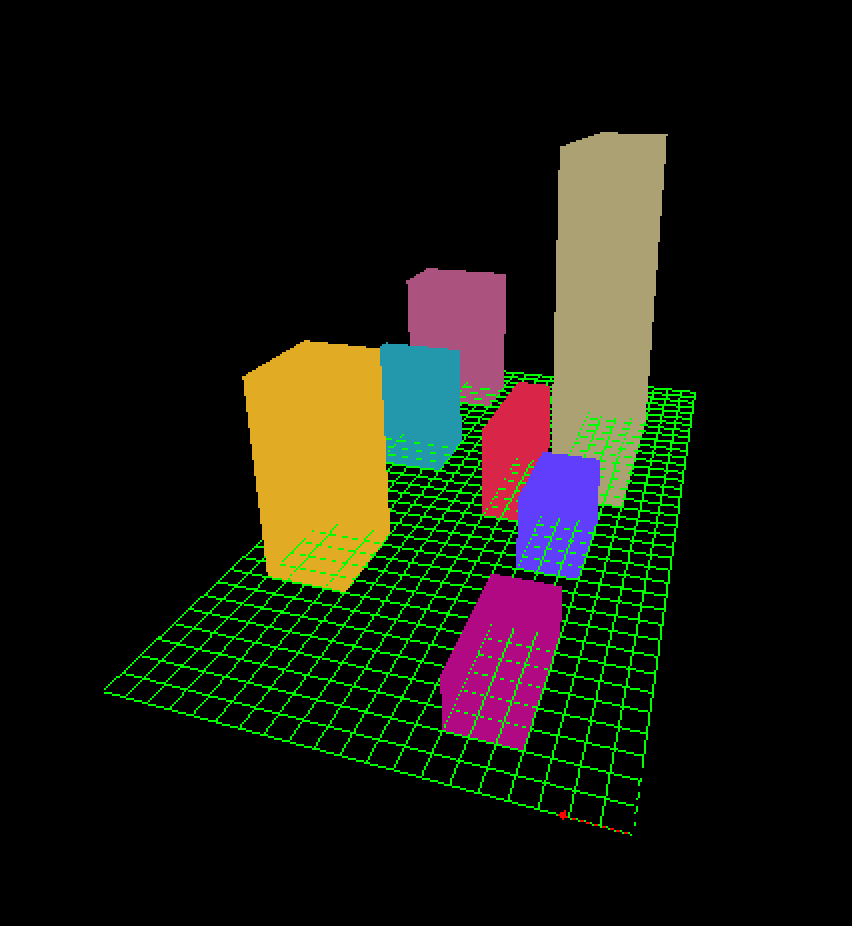
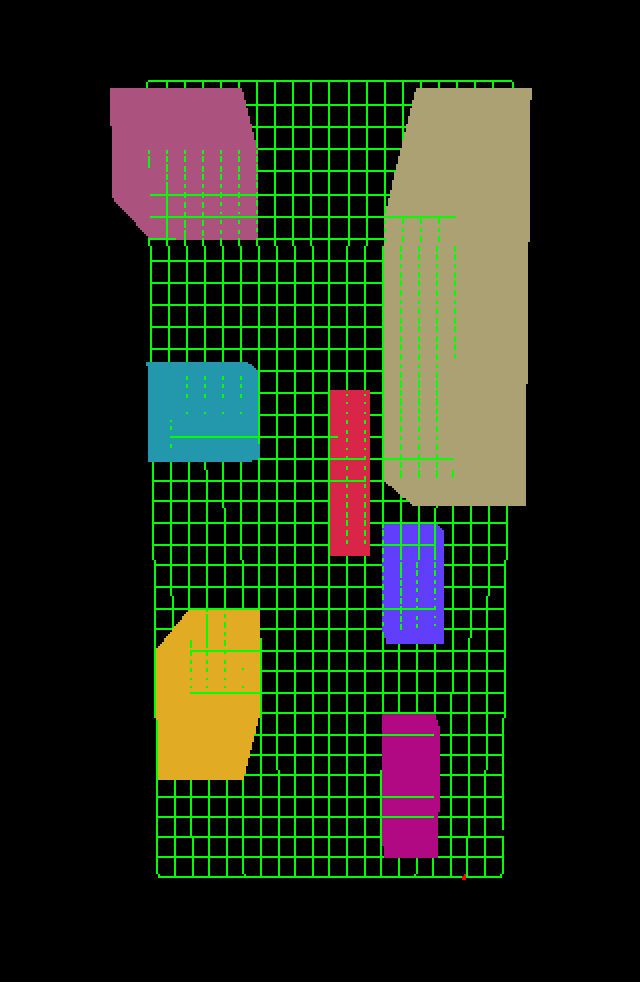
Ponieważ wizualiacja wymaga używania elementów trójwymiarowej grafiki komputerowej, nie można w tym celu użyć standardowych, wysokopoziomowych kontrolek oferowanych przez framework Android-a. Aby móc wyświetlać interaktywne elementy 3D, należy użyć w tym celu wbudowanej we framework biblioteki OpenGL. Aby przyśpieszyć implementację, postanowiono o skorzystaniu z dodatkowej biblioteki LibGDX. Biblioteka ta dostarcza programiście dodatkowych narzędzi pozwalających uniknąć niskopoziomowego programowania w OpenGL. Narzędzia te umożliwiają posługiwanie się nieco wyższym poziomem abstrakcji, skutecznie i optymalnie zamieniając ją w szereg wektorów i macierzy reprezentujących typy proste wymagane przez OpenGL. Ponadto biblioteka LibGDX umożliwia stworzenie aplikacji hybrydowych, działających na wielu platformach. Poza platfromą Android, biblioteka umożliwia uruchomienie tworzonego oprogramowania jako zwykłej aplikacji desktopowej lub webowej. Aby zapewnić hybrydowość aplikacji, należy podzielić kod całego projektu na moduły. Modułowość aplikacji łatwo można osiągnać w systemie Gradle, który jest naturalnym system budowania projektu dla aplikacji na platformy Android. Gradle jest konkurencyjnym systemem do Maven, który jest popularnie stosowany w projektach pisanych w języku Java. Za pomocą systemu Gradle, stworzono 3 moduły w projekcie, jak przedstawiono na poniższym rysunku:

*Rys. 5. Graf przedstawiający strukturę projektu wykorzystanego w implementacji*

Jak widzimy, cały projekt jest teraz podzielony na 3 moduły. Pierwszy z nich słusznie zawdzięcza swoją nazwę od angielskiego słowa rdzeń, gdyż można go przyrównać do silnika całej aplikacji. W module tym powinna się znaleźć cały kod odpowiadający za logikę oraz renderowanie obrazu. Pozostałe moduły, czyli „android” oraz „desktop” służą za punkty startowe dla poszczególnych platform. Przykładowo, gdy aplikacja będzie uruchamiana na telefonie z systemem Android, moduł „android” będzie odpowiadał za jej inicjalizację, by następnie móc uruchochomić zawartość modułu „core”. Dlatego też moduł ten zawiera kod, który realizuje funkcjonalności specyficzne dla platformy. W przypadku platformy Android będzie to stworzenie odpowiedniej struktury aplikacji, w której znajdą się zarówno definicje uprawnień aplikacji (AndroidManifest.xml), katalogi z zasobami aplikacji (przykładowo ikony aplikacji) oraz same źródła aplikacji, korzystające z podstawowych komponentów frameworku. Przykładem takiego podstawowego komponentu na platformie jest Activity. Komponent ten jest ściśle związany z systemem – stanowi punkt startowy aplikacji oraz posiada swój własny cykl życia, uwarunkowany w dużej mierze od samego systemu. Tak samo wygląda sprawa w przypadku modułu „desktop”, który będzie odpowiadał za inicjalizację okna, w której ma się wyrenderować aplikacja. Pozostała część logiki aplikacji – nie tylko ta mówiąca o tym co ma zostać wyrenderowane, ale także jak aplikacja ma reagować na polecenia użytkownika oraz jaką ma spełniać funkcjonalność, znajduje się w module „core”, który jest wspólny dla wszystkich innych modułów reprezentujących poszczególne platformy. Taka hybrydowość jest bardzo wydajnym podejściem, bowiem umożliwia ona programiście stworzenie jednolitego, wspólnego modelu dla wszystkich możliwych platfrom, odciążając go tym samym od konieczności implementacji różnych sposobów interakcji z użytkownikiem. Biblioteka LibGDX dostarcza spójną warstwę abstrakcji, która przykładowo umożliwia obłsugę aplikacji zarówno za pomocą myszki na komputerze jak i za pomocą gestów wykonywanych na dotykowym ekranie telefonu komórkowego, bez potrzeby implementacji osobnych metod specyficznych dla konkretnego rodzaju sprzętu. Nie oznacza to jednak, że rezygnujemy całkowicie z możliwości konkretnej platformy – LibGDX umożliwia tworzenie konkretnych wersji poszczególnych klas, które będą wykorzystane tylko w momencie uruchomienia aplikacji na platformie, na którą te klasy są przeznaczone. Podejście takie jednak nie jest zalecane, gdyż tworzenie implementacji wyspecjalizowanych pod kątem konkretnej platformy pozbawia aplikację hybrydową wszystkich jej zalet.

W związku z powyżej przedstawioną koncepcją aplikacji hybrydowej, cały kod odpowiedzialny za reprezentację mapy, przekształcanie jej w graf a także wyszukiwanie drogi został przeniesiony do wspólnego modułu „core”. Dodatkowo, w module tym pojawiły się klasy odpowiedzialne za samo renderowanie tejże mapy. Bibliteka LibGDX pozwala uninkąć skomplikowanego tworzenia trójwymiarowych elementów używając specyficznych dla OpenGL typów prymitywnych. Zostały one zastąpione pojęciem Modelu. Model w ujęciu LibGDX reprezentuje pewnego rodzaju wzór, według którego ma powstać konkretny element na scenie. Model nie jest w żaden sposób związany ze sceną – nie ma on informacji o tym, w którym miejscu na scenie ma on zostać umieszczony. Posiada on tylko i wyłącznie informacje o swoim wyglądzie, m. in. o materiale, z którego jest wykonany oraz o swoim kształcie. Kształt ten może być prosty, które tworzą podstawowe elementy geometryczne, takie jak prostopadłościany, sześciany, ostrosłupy, walce czy też kule. Model może też składać się z tak zwanych części („Parts”), co umożliwia tworzenie bardziej skomplikowanych kształtów. Nie zagłebiając się w szczegóły techniczne, Model posiada informacje typu „jak obiekt powinien wyglądać”. Informację na temat „jak dany obiekt wyrenderować” posiada Instancja Modelu („ModelInstance”). Instancja Modelu posiada informację o tym, z jakich typów prymitywnych obiekt powinien być zbudowany, a także w którym miejscu na scenie powinien on zostać wyrenderowany. Mówiąc prościej, Instancja Modelu zajmuje się stworzeniem odpowiedniej siatki Modelu, która silnik OpenGL będzie w stanie poprawnie zinterpretować oraz wyświetlić. W celach optymalizacyjnych, wiele Instancji Modelu można łączyć w jedną siatkę, która umożliwi szybsze renderowanie obiektów. W tym celu należy stworzyć specjalny bufor („ModelCache”), który jest w stanie scalać wszystkie obiekty typu ModelInstance w jedną, spójną siatkę, co umożliwi jej efektywne wyrenderowanie silnikowi OpenGL.

Podczas implementacji powstała osobna klasa „MapRenderer”, odpowiedzialna ze samo renderowanie obiektu mapy. Renderuje ona regularną, prostokątną siatkę reprezentujacą wymiary mapy, a także wszystkie zawarte w niej budynki. Ponieważ każdy budynek posiada listę punktów, reprezentującą punkty mapy które zajmuje ten budynek, na podstawie tej listy tworzona jest taka sama liczba Instancji Modelu. Instancje Modelu tworzone są z Modelu, który jest zwykłym sześcianem o wymiarach pojedynczej kratki na wcześniej wyrenderowanej siatce. Instancje te następnie łączone są w jedną siatkę za pomocą bufora ModelCache, który jest już optymlanie renderowany na scenie. Ponadto klasa MapRenderer zajmuje się także wyświetlaniem znalezionej ścieżki, która złożona jest z szeregu Instancji Modelu kuli o średnicy pojedynczej kratki tworzącej siatkę.



a)

b)

c)

*Rys. 6. Wizualizacja mapy za pomocą biblioteki LibGDX. a) widok prostopoadły, b) widok boczny, c) widok boczny z wyrenderowanym przykładem znalezionej ścieżki*

Warto w tym miejscu zauważyć, iż mapa ma taki sam wygląd niezależnie od urządzenia, na której została wyrenderowana. Na rys. nr 5 a) widzimy mapę wyrenderowaną na smartfonie z systemem Android, natomiast na rys. nr 5 b) i c) znajduje się ta sama mapa, jednak wyrenderowana za pomocą aplikacji desktopowej działającej na systemie OSX.

* 1. **Integracja z usługą Google Maps**

Ostatnim elementem omawianego projektu jest integracja z usługą Google Maps. Obecna modułowa struktura projektu uwzględnia działanie aplikacji również w trybie desktopowym. Ponieważ praca w jako cel przewiduje działanie aplikacji jedynie na urządzeniach z systemem Android, integracja ta została wykonana tylko w zakresie modułu „android”. Nie oznacza to jednak, że jest ona niemożliwa do przeprowadzenia w zakresie aplikacji desktopowej.

Na obecnym etapie aplikacja mobilna posiada tylko jedną aktywność (Acitvity), której zadaniem jest uruchomienie kodu znajdującego się w module „core”, czego efektem jest natychmiastowe pojawienie się mapy. Ponieważ celem integracji jest przekształcenie mapy zaczerpniętej z usługi Google Maps, postanowiono aby dodać dodatkowe Activity, które by było nowym punktem startowym aplikacji mobilnej. Activity to ma za zadanie otworzyć mapy z usługi Google Maps.

----- to poniżej jeszcze do poprawy ----

Użyte źródła - do poprawy

<http://home.agh.edu.pl/~krisfoto/lib/exe/fetch.php?media=sit:4_wektor_raster.pdf>

# Bibliography

1. **Şucan, Ioan A. , Moll, Mark and Kavraki, Lydia.** *The Open Motion Planning Library.* s.l. : IEEE Robotics & Automation Magazine, 2011.

2. **Mark, Moll, et al.** *Teaching Motion Planning Concepts to Undergraduate Students.* s.l. : IEEE Workshop on Advanced Robotics and Its Social Impacts (ARSO), 2011.

3. <https://developer.android.com/about/dashboards/index.html#Platform>