



**Politechnika Krakowska
im. Tadeusza Kościuszki**

Wydział Fizyki, Matematyki i Informatyki



Robert Matejczuk

Numer albumu: 98045

**Adaptacyjny system strategii gry w statki na
urządzenia mobilne, w oparciu o wcześniejsze
rozgrywki graczy**

**Adaptive strategy system for Battleships game
on mobile devices based on earlier player's
matches**

**Praca magisterska
na kierunku Informatyka Stosowana**

Praca wykonana pod kierunkiem:
dr inż. Jerzy Białas

Uzgodniona ocena:.....

.....

podpisy promotora i recenzenta

Kraków 2016

Rodzicom, za ich nieocenione wsparcie i troskę.

Przyjaciołom, którzy zawsze przy mnie byli i motywowali.

Nauczycielom i Profesorom, za ich cierpliwość i wskazanie drogi.

Spis treści

1.	Wstęp	4
1.1.	Cele i założenia pracy	5
1.2.	Przegląd zawartości pracy	5
2.	Zasady gry w statki	7
3.	Analiza wymagań.....	10
3.1.	Słownik użytych skrótów i pojęć	10
3.2.	Wymagania ogólne.....	11
3.3.	Wymagania funkcjonalne	12
3.3.1.	Lista i diagramy przypadków użycia	12
3.4.	Wymagania niefunkcjonalne	18
3.5.	Wymagania systemowe.....	19
4.	Analiza popularności systemów mobilnych	20
4.1.	Android	21
4.2.	iOS.....	25
4.3.	Podsumowanie	26
5.	Projekt systemu	28
5.1.	Architektura systemu	28
5.2.	Wybór technologii implementacji	29
6.	Implementacja systemu.....	36
6.1.	Diagram klas i opis działania algorytmu.....	37
6.2.	Baza danych SQLite	41
7.	Adaptacyjny system strategii Androida	42
8.	Wyniki przeprowadzonych badań.....	48
9.	Fragmenty kodu i przedstawienie rozwiązań niektórych funkcjonalności	52
10.	Podsumowanie	55
11.	Bibliografia	56

1. Wstęp

W dzisiejszych czasach niemal każda osoba posiada smartfona. Zastępuje on nam wiele różnych rzeczy jak np. kalendarze, kalkulatory, notesy które to trzeba by było nosić ze sobą. Smartfon również ułatwia nam życie. Przypominając nam o ważnych, zapisanych wydarzeniach, pozwalając na dostęp do Internetu gdziekolwiek jesteśmy, jak również i umożliwia nam komunikowanie się z innymi osobami. Dodatkowo istnieje wiele aplikacji które rozwijają możliwości i funkcjonalności naszego smartfona.

Dodatkowo też, coraz większym zainteresowaniem i popularnością cieszą się tablety, jak również i smartfony z większym ekranem. Pozwalają nam na wygodniejsze korzystanie z wielu funkcjonalności. Dla wielu ludzi tablet bądź smartfon to nie tylko potrzebne narzędzia do pracy bądź ułatwiające życie urządzenie. Wiele osób używa tych urządzeń w celach rozrywkowych. Oglądają filmy, czytają książki, bądź grają w gry. Ta ostatnia forma spędzania czasu nie jest wcale zadziwiająca, gdyż przy dzisiejszej technologii na tablecie dobrej klasy, użytkownik jest w stanie uruchomić pięknie wyglądające gry 3D w stałych 30 kl./s.

Niestety przez to, że wiele aplikacji można pozyskać w bardzo prosty sposób, gdyż każdy system mobilny ma swój własny sklep poprzez który użytkownik może pobierać aplikacje (np. Android: Google Play, iOS: App Store) dostępność różnych aplikacji jak i konkurencja jest spora. Dlatego nie lada wyzwaniem jest wymyślenie czegoś co jeszcze zupełnie nie jest dostępne. Dlatego wiele firm konkurencyjnych stara się robić własne produkcje, zbliżone do tych które już istnieją. Przeważnie z założeniem, że zostaną zrobione lepiej i ładniej. Tak też jest i z moją aplikacją. Gra w statki jest grą bardzo popularną i istnieje już wiele aplikacji z różnymi odmianami tej gry. Niestety implementacja sztucznej inteligencji przeciwnika nie jest prostym zadaniem i wiele razy może się okazać, że poziom reprezentowanej inteligencji przez przeciwnika jest zbyt niski aby stanowić wyzwanie dla gracza. Dlatego moja aplikacja powstała z myślą o przeprowadzaniu badań nad tym problemem i sprawdzeniu czy adaptacyjny system, który uczy się na podstawie wcześniejszych gier gracza może stanowić dla niego ciągłe wyzwanie i zmuszać go do ciągłego myślenia a nie stosowania sprawdzonych schematów które zapewniają mu proste zwycięstwo.

1.1. Cele i założenia pracy

Celem pracy jest stworzenie aplikacji mobilnej, będącej odwzorowaniem klasycznej gry w statki, zawierającej sztuczną inteligencję wraz z adaptacyjnym systemem strategii bazującym na wcześniejszych rozgrywkach gracza (użytkownika). A następnie przeprowadzenie badań czy owo powstały system spełnia swoje założenia.

Założeniem pracy jest to, że sztuczna inteligencja (SI) będzie rozwijać swoje umiejętności i uczyć się podejmowania trafnych decyzji na podstawie analizy danych zebranych z wcześniejszych gier z tym samym graczem. Zmuszając tym samym gracza do myślenia. Sztuczna inteligencja ma być na tyle wymagająca aby gracz odczuwał satysfakcję z wygranej, ale też aby nie był zniechęcony tym, że nie może wygrać z SI.

1.2. Przegląd zawartości pracy

1. Wstęp – Na początku tego rozdziału znajduje się wstęp który opisuje skąd się wziął pomysł na przeprowadzenie badań związanych ze sztuczną inteligencją. W tym rozdziale zostają również przedstawione założenia i cele pracy.
2. Zasady Gry w statki – Rozdział poświęcony zasadom gry w statki. Znajduje się tutaj dokładny opis zasad gry w statki i warunki zwycięstwa.
3. Analiza wymagań – W tym rozdziale znajdują się informacje o wymaganiach jakie aplikacja musi spełniać do prawidłowego działania. Zawiera informacje jak i diagram przypadków użycia oraz sytuacji alternatywnych.
4. Analiza popularności systemów mobilnych – Rozdział poświęcony analizie systemów mobilnych. Ma on na celu sprawdzić na jaki system mobilny najbardziej opłaca się stworzyć aplikację aby dotrzeć do potencjalnie jak największej ilości odbiorców. Porównanie dotyczy systemu Android oraz iOS, gdyż są to jedyne dwa systemy które na obecną chwilę mają jakieś znaczenie na rynku.
5. Projekt systemu – W tym rozdziale znajduje się projekt systemu. Opisuje on architekturę systemu, wybrane technologie wraz z ich zaletami oraz zawiera informacje jak działają aplikacje na Androidzie jak również jak wygląda proces komplikacji pliku APK.

6. Implementacja systemu – Rozdział ten przedstawia diagram użycia aplikacji, diagramy klas oraz przedstawia wygląd bazy danych SQLite. Znajduje się tutaj również opis działania aplikacji i sposób aktualizacji bazy danych.
7. Adaptacyjny system strategii Androida – Rozdział ten opisuje sposób działania algorytmu adaptacyjnego zaimplementowanego dla Androida. Jest to algorytm strategii gry który pozwala Androidowi na podejmowanie decyzji jak rozmieścić swoje statki podczas przygotowania oraz w które pole najlepiej jest oddać strzał podczas bitwy wraz z przykładami. Obrazuje jak android widzi planszę do umieszczania statków oraz do oddawania strzałów.
8. Wyniki przeprowadzonych badań – W tym rozdziale znajduje się opis sposobu oraz wyniki przeprowadzonych badań. Zawiera informacje o taktykach i sytuacji w jakiej algorytm może się znaleźć (rozkład normalny).
9. Fragmenty kodu i przedstawienie rozwiązań niektórych funkcjonalności – Przedstawienie kilku fragmentów kodu w celu pokazania sposobu rozwiązania niektórych funkcjonalności. Zawiera fragmenty kodu przedstawiające tworzenie bazy danych przy użyciu pomocnika oraz przekazywania intencji z aktywności odpowiedzialnej za przygotowanie statków gracza do aktywności odpowiadającej za bitwę wraz z parserem przetwarzającym odebraną intencję.
10. Podsumowanie – Rozdział podsumowujący przeprowadzone badania. Zawiera wnioski i obserwacje oraz propozycje przyszłej rozbudowy i ulepszeń jakie można wprowadzić do algorytmu.
11. Bibliografia – Zbiór informacji z których korzystałem i czerpałem swoją wiedzę aby zrealizować postawione cele i założenia pracy.

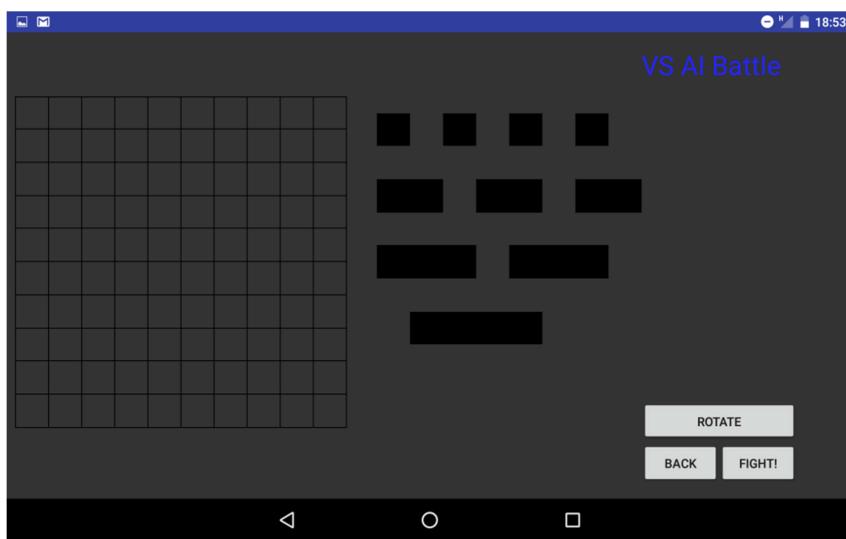
2. Zasady gry w statki

Gra w statki, jest to popularna gra zwana również Bitwa Morska lub Okręty (ang. Battleship). Jest to gra strategiczno-planszowa dla dwóch osób którą znam i gram od dzieciństwa. Wynaleziona została przez Clifforda Von Vicklera na początku XX wieku, a opatentowana przez Milton Bradley Company w 1943 roku [1].

Rozgrywkę można prowadzić na kartce papieru bądź na specjalnych planszach. Każdy z graczy posiada dwie plansze o wielkości 10x10 pól numerowane u góry od 1 do 10 i od boku literami od A do J. Na lewym kwadracie gracz umieszcza swoje statki. Do swojej dyspozycji w klasycznej¹ wersji ma 10 statków do rozmieszczenia:

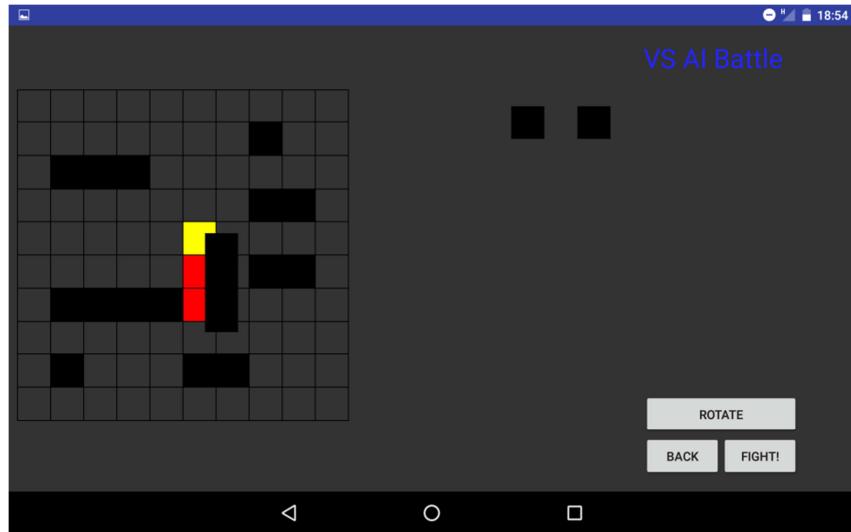
- 1x 4-masztowiec (zajmuje 4 pola)
- 2x 3-masztowce (zajmują po 3 pola)
- 3x 2-masztowce (zajmują po 2 pola)
- 4x 1-masztowce (zajmują po 1 polu)

Statki mogą być rozmieszczane pionowo bądź poziomo. Należy pamiętać aby między statkami było przynajmniej zawsze jedno pole odstępu.



Rysunek 1: Pole i statki gracza

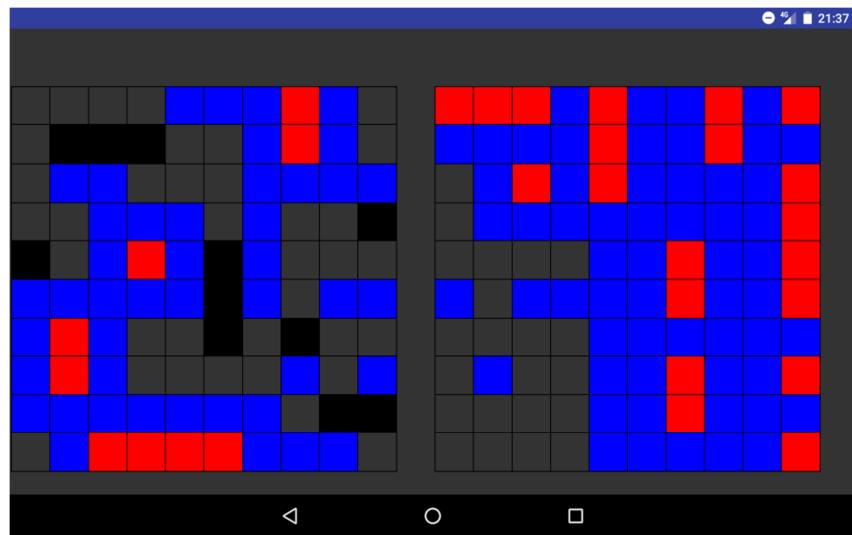
¹ Istnieje kilka różnych odmian gry. W niektórych grach gracz ma do dyspozycji 5-masztowiec ale za to nie ma czterech 1-masztowców.



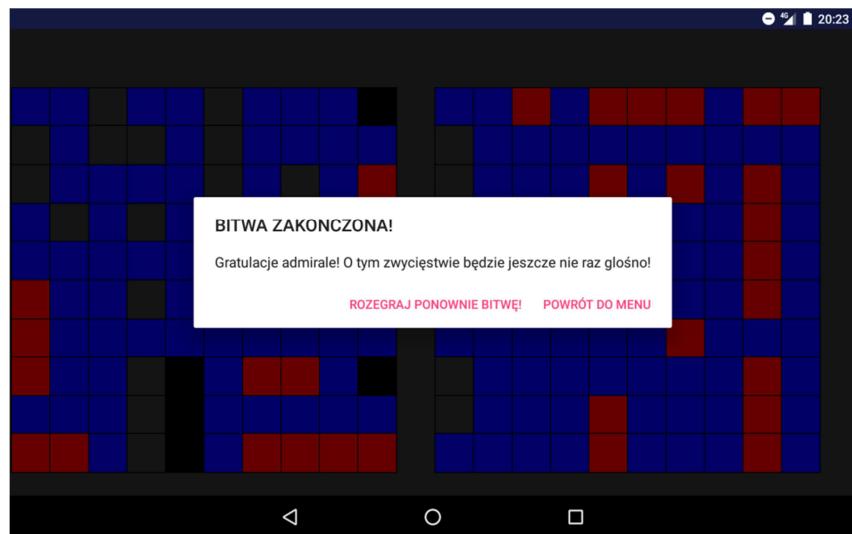
Rysunek 2: Przykładowe rozkładanie statków

Gdy statki zostały już rozmieszczone osoba rozpoczynająca mówi współrzędne pola (w naszym przypadku klika na pole) np. B8. Następnie otrzymuje informacje czy na tym polu znajdował się statek czy też nie. Jeśli był tam statek, to strzał został zaliczony jako trafienie co pozwala graczowi na oddanie kolejnego strzału. Jeśli natomiast nie było tam statku to strzał uznawany jest za tak zwane „pudło”. Wiąże się to ze stratą możliwości dalszego strzelania i przekazanie ruchu przeciwnikowi. W momencie gdy gracz trafi i zatopi statek to wszystkie pola wokół statku zostają automatycznie odsłonięte. Nazywam to wybuchem. Dlatego też trzeba zachowywać jedno pole odstępu między statkami. Gdy statek zostaje zniszczony gracz który trafił dalej kontynuuje strzelanie.

Gra się kończy w momencie gdy któryś z graczy straci wszystkie statki. Zwycięzcą zostaje osoba która zniszczyła wszystkie statki przeciwnika.



Rysunek 3: Przykładowa rozgrywka



Rysunek 4: Komunikat o zwycięstwie gracza

3. Analiza wymagań

Poniższy opis wymagań dla projektowanej aplikacji został opracowany ze szczególną uwagą na zabezpieczenia przed dokonywaniem potencjalnych błędnych ruchów przez gracza jak również skupieniu się nad prawidłowym algorytmem sztucznej inteligencji tak aby gracz czerpał jak największą satysfakcję z wygranej i nie zniechęcał się po porażce.

3.1. Słownik użytych skrótów i pojęć

W celu precyzyjnego opisania wymagań niezbędne jest wyjaśnienie używanych w specyfikacji terminów oraz skrótów:

- W – wymaganie aplikacji, gdzie WF oznacza wymaganie funkcjonalne, a WNF wymaganie niefunkcjonalne.
- UC – Przypadek użycia (ang. *use case*).
- Gracz – użytkownik korzystający z aplikacji mobilnej.
- Statek – Obiekt o różnej długości (ilość masztów) na planszy który umieszcza gracz.
- „Pudło” – odnosi się do strzału który nie trafił w statek.
- „Trafiony” – odnosi się do strzału który trafił w statek.
- „Trafiony zatopiony” – odnosi się do strzału który spowodował zniszczenie statku i odsłonięcie otoczki statku.
- Otoczka statku – są to pola przyległe do statku które w momencie zniszczenia statku i jego wybuchu zostają odsłonięte.
- Wybuch – moment zniszczenia statku i odsłonięcie pól stanowiących otoczkę statku.
- Android – sztuczna inteligencja z adaptacyjnym systemem strategii. Jest przeciwnikiem gracza.

3.2. Wymagania ogólne

Głównym celem w tworzonej przezem mnie aplikacji jest stworzenie i przeprowadzenie badań nad sztuczną inteligencją z adaptacyjnym systemem strategii gry w statki.

Wymaganiem pośrednim który łączy się z wyżej wymienionym wymaganiem jest stworzenie gry w statki z zachowaniem wszystkich zasad i umożliwienie prowadzenia rozgrywki graczowi z Androidem w celu przeprowadzenia badania. Dodatkowym głównym wymaganiem jest stworzenie bazy danych która pozwoli przechowywać informacje o rozgrywkach z której będzie korzystał Android jak również do przechowywania informacji o statystykach na temat rozgrywek.

Użytkownik będzie miał graficznie przedstawiony interfejs i za jego pomocą będzie mógł wchodzić w interakcje ze statkami (przesuwanie, umieszczenie na planszy, obracanie) oraz polami gry w celu oddania strzału. Gra będzie kontrolować ruchy gracza i blokować je gdy będzie on chciał dokonać zakazanego ruchu np. Ustawić statki bez zachowania pola odstępu między nimi. Gdy któryś ze statków zostanie zniszczony i nastąpi wybuch to system automatycznie odsłoni otoczkę statku. Zakończenie gry i komunikat o zwycięstwie bądź porażce zostanie wyświetlony w momencie, gdy któraś ze stron zniszczy ostatni statek przeciwnika. Po skończonej bitwie gra będzie zapisywała jej przebieg do bazy danych jak również zwycięzcę oraz czas gry w liczbie tur.

Aplikacja powstała na urządzenia mobilne ze względu na łatwą dostępność (użytkownik ma urządzenie przeważnie zawsze ze sobą) i możliwość dostarczenia rozrywki użytkownikowi gdziekolwiek będzie się znajdował.

3.3. Wymagania funkcjonalne

Funkcjonalności aplikacji gry w statki zostały podzielone na Ustawienia Rozgrywki oraz Rozegranie Bitwy:

- **W1 – Ustawienia Rozgrywki i przeprowadzenie bitwy** – Gracz ma możliwość wybrania różnych trybów rozgrywki². Ta funkcjonalność zawiera również proces przygotowania statków do gry jak i mechanizm zapobiegający dokonania niedozwolonych ustawień statków przez użytkownika. Do tej funkcjonalności zaliczam również sprawdzenie statystyk odnośnie wcześniejszych gier.

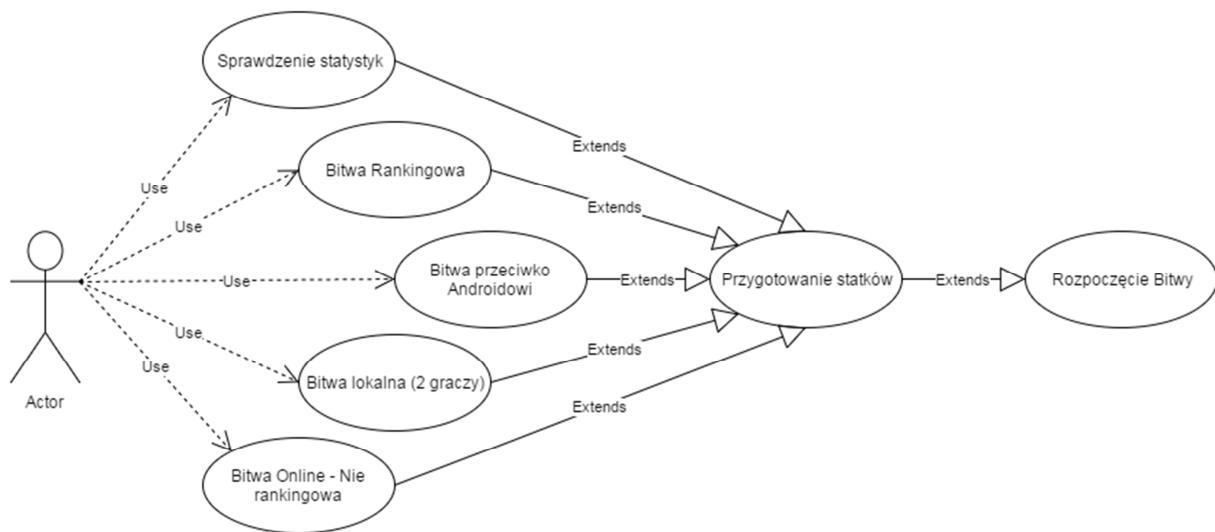
Funkcjonalność ta zawiera zbiór mechanizmów kontrolujących przebieg rozgrywki. Przekazuje turę graczom, wywołuje wybuch gdy statek zostanie zniszczony, informuje o zakończeniu gry i wyświetla wynik bitwy. Funkcjonalność ta zawiera adaptacyjny algorytm Androida (dokładny opis w rozdziale 7) który jest używany jeśli gracz wybierze tryb gry przeciwko Androidowi oraz mechanizm aktualizacji bazy danych na temat przeprowadzonej bitwy z Androidem.

3.3.1. Lista i diagramy przypadków użycia

Poniżej zostały przedstawione oraz szczegółowo opisane przypadki użycia wraz z scenariuszem użytkowania oraz scenariuszem alternatywnym jeśli takowy istnieje, obrazującego zachowanie systemu w przypadku błędu np. w sytuacji gdy gracz chce dokonać błędnego ustawienia statków.

² My skupimy się na rozgrywce prowadzonej przeciwko Androidowi

WF1 – Ustawienia Rozgrywki



Rysunek 5: Diagram przypadków użycia dla Ustawień Rozgrywki

Identyfikator i nazwa: UC.1.1 – Sprawdzenie statystyk

Opis: Sprawdzenie historii rozgrywek

Scenariusz:

1. Gracz przechodzi z aktywności głównego menu do aktywności statystyk.
2. Aktywność pobiera informacje z bazy danych i wywołuje układ przedstawiający pobrane statystyki pod postacią listy ukazującej ID rozgrywki, zwycięzcę oraz długość rozgrywki w postaci ilości rozegranych rund.

Scenariusz alternatywny:

1. Użytkownik przechodzi z aktywności głównego menu do aktywności statystyk.
2. Aktywność pobiera informacje z bazy danych, niestety gracz nie rozegrał jeszcze żadnej gry więc zamiast statystyk wyświetlony zostaje komunikat o braku historii rozgrywek.

Identyfikator i nazwa: UC.1.2 – Bitwa rankingowa

Opis: Rozpoczęcie przygotowań do bitwy rankingowej

Scenariusz:

1. Gracz przechodzi z aktywności głównego menu do aktywności bitwy rankingowej która odpowiednio przygotowuje układ i aktywność przygotowania statków. Szczegółowy opis

przygotowania statków znajduje się w UC.1.6 Przygotowanie statków.

Scenariusz alternatywny: Brak.

Identyfikator i nazwa: **UC.1.3 – Bitwa przeciwko Androidowi**

Opis: Rozpoczęcie przygotowań do bitwy przeciwko Androidowi

Scenariusz: 1. Gracz przechodzi z aktywności głównego menu do aktywności bitwy przeciwko Androidowi która odpowiednio przygotowuje układ i aktywność przygotowania statków.
Szczegółowy opis przygotowania statków znajduje się w UC.1.6 Przygotowanie statków.

Scenariusz alternatywny: Brak.

Identyfikator i nazwa: **UC.1.4 – Bitwa lokalna (2 graczy)**

Opis: Rozpoczęcie przygotowań do bitwy lokalnej dla 2 graczy na tym samym urządzeniu

Scenariusz: 1. Gracz przechodzi z aktywności głównego menu do aktywności bitwy lokalnej która odpowiednio przygotowuje układ i aktywność przygotowania statków. Szczegółowy opis przygotowania statków znajduje się w UC.1.6 Przygotowanie statków.

Scenariusz alternatywny: Brak.

Identyfikator i nazwa: **UC.1.5 – Bitwa online – nie rankingowa**

Opis: Rozpoczęcie przygotowań do bitwy nie rankingowej

Scenariusz: 1. Gracz przechodzi z aktywności głównego menu do aktywności bitwy online - nie rankingowej która odpowiednio przygotowuje układ i aktywność przygotowania statków.
Szczegółowy opis przygotowania statków znajduje się w UC.1.6 Przygotowanie statków.

Scenariusz alternatywny: Brak.

Identyfikator i nazwa:	UC.1.6 – Przygotowanie Statków
Opis:	Przygotowanie statków do wybranego trybu gry
Scenariusz:	<p>1. Gracz po wybraniu trybu rozgrywki widzi układ ze swoimi statkami i planszą na którą może te statki umieszczać</p> <p>2. Gracz dotyka palcem wybrany statek i nie odrywając palca od ekranu przeciąga statek na wybrane przez siebie miejsce. Miejsca zostają podświetlone na żółto jeśli można w tym miejscu postawić statek.</p> <p>3. Gracz uznaje, że podświetlone zostało pole w które chciał umieścić statek. Podnosi tym samym palec aby upuścić statek na wskazane miejsce.</p> <p>4. Gracz przeciąga następny statek na obszar planszy i wybiera miejsce w którym chce upuścić statek.</p> <p>5. Ponownie, jeśli wszystkie pola dla danego statku są podświetlone na żółto oznacza to, że statek może być prawidłowo umieszczony w to miejsce.</p> <p>6. Gracz ponawia czynności w celu umieszczenia pozostałych statków na planszę.</p> <p>7. Gdy gracz umieścił już wszystkie statki, postanowił obrócić jeden ze statków. W tym celu dotyka go aby oznaczyć który statek chce obrócić (gdyż przycisk obraca ostatni dotknięty statek). Następnie wciska przycisk z napisem „Rotate” który odpowiada za obracanie statku.</p> <p>8. Jeśli obrócenie statku nie łamie zasad rozmieszczenia gry to statek zostanie obrócony.</p> <p>9. Gdy gracz jest zadowolony z rozstawienia swoich statków w zależności od wybranego trybu gry naciska przycisk Search! (Przy bitwie online oraz rankingowa) w celu wyszukania przeciwnika. Przycisk Next! (Bitwa lokalna) odpowiadający za przejście do układu rozstawiania statków dla drugiego gracza. Przycisk Fight! (Układ drugiego gracza w bitwie lokalnej oraz bitwa przeciwko Androidowi) odpowiadający za rozpoczęcie bitwy i przejście do aktywności jak i układu bitwy przenosząc informacje w postaci Intencji odnośnie rozmieszczenia statków.</p>

- Scenariusz alternatywny:**
- 1a. Podczas rozmieszczania statków gracz nie umieścił całego statku na planszy.
 - 2a. Statek nie zostaje umieszczony na planszy. Powraca on na swoje początkowe miejsce w puli statków.
 - 1b. Podczas rozmieszczania statków gracz chciał umieścić statek mimo tego, że jedno z podświetlanych pól było podświetlone na czerwono (znajdowało się w otoczce innego statku).
 - 2b. Statek nie zostaje umieszczony na planszy. Powraca on na swoje początkowe miejsce w pulu statków.
 - 1c. Podczas rozmieszczania statków gracz postanowił obrócić statek. Niestety naruszyło to poprawność rozmieszczenia statków gdyż po obróceniu statek znajdował się w otoczce innego statku.
 - 2c. Zostaje wyświetlony stosowny komunikat informujący gracza o tym, że nie może obrócić statku w tym miejscu.
 - 3c. Gracz ściąga statek z planszy, obraca go w puli statków i ponownie umieszcza go na planszy z już zmienioną orientacją.
 - 1d. Gracz próbuje rozpocząć rozgrywkę, bądź przejść do układu rozmieszczania statków przez drugiego gracza (w bitwie lokalnej).
 - 2d. Pojawia się stosowny komunikat informujący o tym, że gracz nie rozmieścił wszystkich swoich statków i musi to zrobić zanim będzie mógł przejść dalej.
 - 1e. Gracz przesunął statek ale nie umieścił go na obszarze planszy.
 - 2e. Statek powraca na swoją początkową pozycję w puli statków.

- Identyfikator i nazwa:** UC.1.7 – Rozpoczęcie bitwy
- Opis:** Rozpoczęcie bitwy
- Scenariusz:**
1. Gracz przechodzi z aktywności przygotowania statków do aktywności bitwy która wywołuje odpowiedni układ.
 2. Aktywność pobiera informacje z przesłanej wcześniej Intencji i obrazuje rozmieszczone przez gracza statki po lewej stronie. Po prawej stronie natomiast gracz widzi planszę poprzez którą będzie wybierał pola w które chce oddać strzał. Po załadowaniu się układu zostaje wyświetlony komunikat o rozpoczęciu bitwy jak i osobie rozpoczynającej rozgrywkę.
 3. Gracz wybiera pole poprzez dotyk ekranu w strefie planszy przeciwnika.
 4. Gracz może przesuwać palcem po ekranie bez odrywania go od ekranu w celu wybrania pożądanego pola. Odpowiednie pola zostają podświetlane na żółto w zależności od lokalizacji palca na ekranie. W ten sposób gracz dokładnie może wybrać pole w które chce oddać strzał.
 5. Gdy gracz wybrał już pole podnosi palec z ekranu i strzał zostaje oddany.
 6. System sprawdza co znajduje się we wskazanym polu. Jeśli nic tam nie było to strzał liczony jest jako „pudło” i pole zostaje podświetlone na niebiesko a gracz traci swoją kolejkę. Jeśli w tym polu znajdował się statek, pole zostaje podświetlone na czerwono a gracz może oddać następny strzał. Jeśli strzał doprowadził do zniszczenia statku „Trafiony zatopiony” następuje wybuch statku a system odsłania otoczkę statku. Gracz może kontynuować strzelanie.
 7. Gdy strzał gracza oznaczał „pudło” przychodzi kolej na strzał ze strony Androida. Android przeszukuje listę zawierającą informację na temat wcześniejszych rozgrywek z tym graczem, która została uzupełniona i przygotowana przed rozpoczęciem

bitwy. Dokładne działanie algorytmu Androida zostało opisane w rozdziale 7. Android wybiera z tej listy odpowiednie pole i oddaje strzał. Jeśli trafi to kontynuuje strzelanie. Natomiast jeśli spudłował, możliwość oddania strzału przechodzi do gracza i teraz to on wybiera miejsce w które chce oddać strzał. Gdy obaj gracze oddadzą strzał licznik tur zwiększa się.

8. Gra jest kontynuowana wg powyższych zasad do momentu w

którym jedna ze stron nie zniszczy wszystkich statków

przeciwnika.

9. Gdy to nastąpi zostanie wyświetlony odpowiedni komunikat

powiadamiający o tym, że bitwa została zakończona oraz w

komunikacie będzie zawarta informacja która ze stron odniosła

zwycięstwo.

10. Gracz teraz może dokonać wyboru, czy chce powrócić do

głównego menu, czy rozegrać bitwę ponownie, powracając do

aktywności związanej z przygotowaniem statków do gry.

11. Niezależnie od wyboru gracza baza danych zostaje

zaktualizowana o informacje o przeprowadzonej rozgrywce.

Scenariusz alternatywny: 1. Gracz próbuje nacisnąć obszar nienależący do planszy przeciwnika.

2. Akcja onTouchEvent ignoruje taki dotyk. Nic się nie dzieje.

3.4. Wymagania niefunkcjonalne

Podczas analizy wymagań pojawiły się również wymagania niefunkcjonalne.

Wyniknęły one wprost ze scenariuszy możliwego użytkowania aplikacji jak również z doświadczenia, wiedząc, że w większości przypadków każda gra będzie miała dodatki lub usprawnienia w działaniu systemu. W przypadku gry w statki można wprowadzić rozbudowanie polegające na modyfikacji zasad i dać graczowi możliwość np. posiadania pięcio masztowca zamiast czterech jednomasztowców, wprowadzić możliwość umieszczania

statków o nieco innym kształcie bądź nawet pozwolić na prowadzenie bitwy na planszy z większą ilością pól jak również i statków.

- **WNF1** – Skalowalność systemu – Projektowana aplikacja zapewnia możliwość prowadzenia rozgrywki w trybie klasycznym z Androidem. Dlatego budowa aplikacji musi umożliwiać dalszy rozwój przy jednoczesnym nieingerowaniu w istniejące już funkcjonalności oraz uniknięcia niepotrzebnej powtarzalności kodu.

- **WNF2** – Szybkość obsługi – wszelkie operacje związane z podejmowaniem strzału przez Androida, jak i rozstawianiem przez niego statków oraz odczyt i zapis danych z bazy mają zostać wykonane w czasie nie większym niż 1s. W pojedynczych przypadkach akceptowalny jest czas nie dłuższy niż 5 sekund. Czas może różnić się w zależności od urządzenia na który aplikacja jest uruchamiana.

3.5. Wymagania systemowe

Ze względu na to, że aplikacja jest przeznaczona na urządzenia mobilne systemem uruchomieniowym musi być jeden z dostępnych systemów na urządzenia przenośne. Z związku z tym, że chciałbym aby aplikacja była dostępna na jak największej ilości urządzeń i trafiła do jak największego grona odbiorców na podstawie analizy popularności systemów mobilnych aplikacja powstała dla urządzeń z systemem Android.

4. Analiza popularności systemów mobilnych

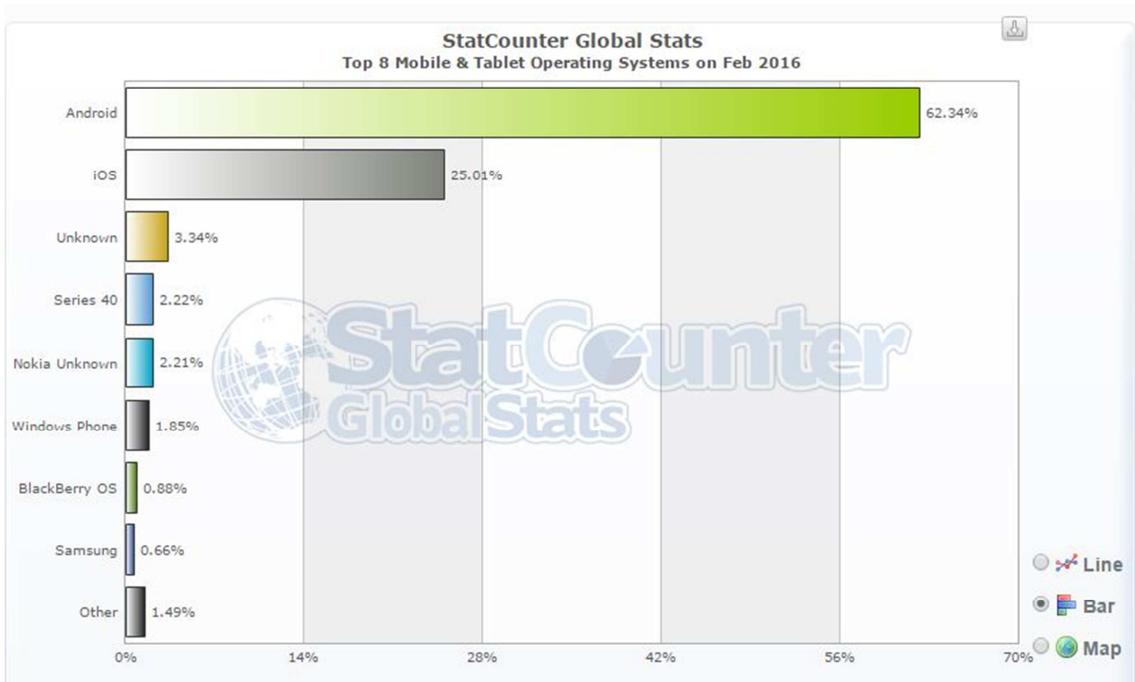
W tym rozdziale przedstawię najpopularniejsze systemy operacyjne jakie są używane obecnie na urządzeniach mobilnych. Oprócz wad i zalet pod uwagę wezmę również cenę, popularność i wsparcie techniczne.

Podobną analizę porównawczą zrobiłem podczas pisania swojej pracy Inżynierskiej. Dlatego też dzięki temu będę mógł odnieść się do szerszej przestrzeni lat oraz sprawdzić czy przez ostatnie 2 lata pozycja popularnych systemów mobilnych uległa zmianie.

Do przeprowadzenia analizy posłużę się stroną Stat Counter [3], która zajmuje się zbieraniem informacji na temat używanych systemów mobilnych. Strona Stat Counter zbiera dane na podstawie ruchu na ponad 3 milionach stron które posiadają zaimplementowany mechanizm Stat Counter. Miesięcznie ten mechanizm dostarcza ponad 15 bilionów wyświetleń.



Rysunek 6: Popularność systemów mobilnych w Polsce [3]



Rysunek 7: Popularność systemów mobilnych na świecie [3]

Przez znikomą pozycję systemu Windows Phone, zrezygnowałem z opisywania go i poddawania analizie.

4.1. Android

Jak możemy zaobserwować z powyższych wykresów system Android nadal jest dominującym systemem mobilnym na świecie. System Android jest wykorzystywany nie tylko do telefonów czy smartfonów ale również na innych urządzeniach multimedialnych. Nazwa Android wzięła się od małej firmy która produkuje oprogramowanie dla telefonów komórkowych. Firma Google Inc. znalazła potencjał w tym rozwiązaniu i postanowiła wykorzystać go do własnych celów wykupując niewielką firmę w Kalifornii w lipcu 2015 roku. [4].

Obecnie najnowszą wersją systemu jest 6.0.1 „Marshmallow” wydany 7 grudnia 2015 roku.

Wady:

- Systemu nie projektowano pod konkretne podzespoły. Dlatego też Android musi się zmagać z różnym sprzętem, różnymi rozdzielczościami i różnym układzie klawiszy, co sprawia, że nowe aktualizacje mogą czasem powodować pogorszenie działania niektórych funkcji i aplikacji.
- Starsze telefony często są wycofywane z dostępu do najnowszych aktualizacji przez problem opisany w pierwszym punkcie.
- Android jest dla wszystkich – każdy użytkownik może umieszczać w Google Play dowolną aplikację. Co powoduje, że oprócz setek profesjonalnych aplikacji możemy znaleźć tam ogólną liczbę bezwartościowych i bezużytecznych aplikacji. Dodatkowo nadzór nad tym procesem jest bardzo słaby.
- Otwartość Androida powoduje również, że jest on łatwo podatny na wirusy. Więc ryzyko zainfekowania swojego zestawu szkodliwym oprogramowaniem jest duże.
- Niespójność poszczególnych wersji systemu sprawia, że jego poziom fragmentacji na rynku jest duży. W praktyce oznacza to, że potencjalni nabywcy mogą kupić zestaw pracujący na trzech różnych wersjach systemu, dla których aktualizacje nie są wystawiane zbyt często.
- Tańsze modele są zazwyczaj wolniejsze i działają mniej stabilnie. Dzieje się tak w przypadkach tańszych urządzeń, gdzie mniej znani producenci dodają własne nakładki na system, oraz nowe funkcje. Chociaż i telefony z firmy Samsung również mają problem z lagami i spadkami płynności systemu.

Zalety:

- Wielozadaniowość, czyli możliwość uruchomienia w tym samym czasie kilku aplikacji oraz możliwość przełączania się między nimi.
- Od użytkownika zależy, jakie dodatkowe możliwości będzie posiadać jego zestaw. Może dowolnie dostosowywać system do własnych potrzeb.

- Stabilność i prędkość działania – Urządzenie, na którym Android został poprawnie zainstalowany i skonfigurowany działa praktycznie bez spowolnień. Nie pojawia się również problem z zawieszaniem się systemu.
- Gry i programy są dostępne za darmo.
- Interfejs jest intuicyjny, prosty i łatwy w obsłudze.
- Użytkownik ma dostęp do pulpitów, na których może umieszczać widżety, foldery i skróty.
- Posiada technologię Multitouch.
- Zaletą Androida jest bardzo dobra współpraca z usługami oferowanymi przez firmę Google. System zapewnia łatwy i przejrzysty dostęp do map, rozkładów lotów, Google Earth, Google Drive, Gmail, Youtube, Google+.
- Android jest dostępny dla urządzeń różnych producentów i marek. Nie istnieją tutaj ograniczenia jak w przypadku iOS który działa tylko na urządzeniach firmy Apple.
- Od wersji 2.2 możliwość obsługi routera WiFi
- Obsługa kart pamięci, obsługa Adobe Flash Player, HTML5, CSS. Dodatkowo w zależności od modelu obsługa HDMI, MHL, DLNA, NFC.

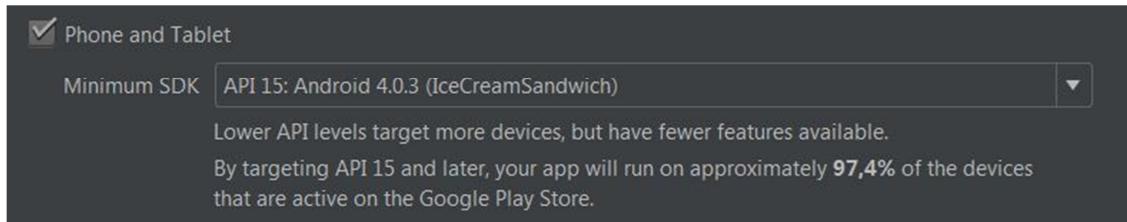
Podsumowanie:

Jak widzimy, Android jak każdy system ma swoje wady i zalety. Jest jednak nadal najbardziej popularnym systemem na świecie. Jest tańszy niż iOS i jest otwarty dla użytkownika pozwalając mu na modyfikowanie systemu tak aby jak najbardziej odpowiadał potrzebom użytkownika. Działając na różnych urządzeniach wielu różnych producentów daje nam możliwość wyboru takiego urządzenia jakie nam najbardziej odpowiada. Nie musimy się dzięki temu ograniczać do konkretnych modeli.

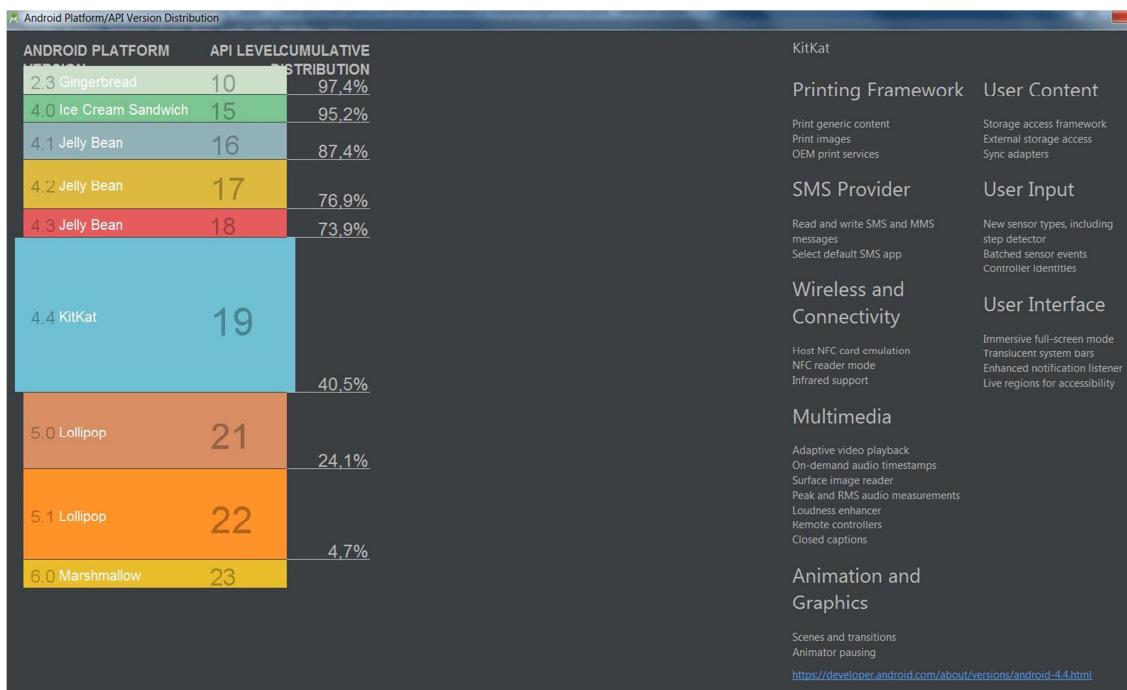
Jak już wspomniałem, wadą Androida jest niespójność poszczególnych wersji. Przez to, że jest duży rozrzut w użytkowaniu różnych wersji programista musi podjąć decyzję pod jaką wersję będzie tworzył oprogramowanie. Oznacza to, że musi wybrać tzw. Poziom

Aplikacji co określa jaka minimalna wersja Androida musi być zainstalowana dla poprawnego działania aplikacji. Wiąże się to z tym, że jeśli chcemy używać jakiś ciekawych widżetów które są dostępne od konkretnej wersji Androida to musimy zwiększyć Poziom Aplikacji. A to wiąże się z tym, że aplikacja będzie dostępna dla mniejszego grona odbiorców.

Poniższe rysunki pochodzą z środowiska programistycznego Android Studio:



Rysunek 8: Informacja jaką wersję najlepiej wybrać aby aplikacja była dostępna dla jak największej ilości urządzeń z systemem Android



Rysunek 9: Procentowa fragmentacja różnych wersji systemu Android które są obecnie używane

4.2. iOS

iOS jest mobilną wersją systemu operacyjnego wydaną przez firmę Apple Inc. dla urządzeń mobilnych iPhone, iPod touch, iPad. Najnowsza wersja: 9.3.2 wydana 16 maja 2016 [5].

Wady:

- System zamknięty i nie daje możliwości modyfikacji.
- Ze względu na politykę producenta do App Store nie trafia wiele usług sieciowych, platform muzycznych itp.
- Brak oficjalnego Adobe Flash Playera.
- Brak obsługi kart pamięci.
- Mała liczba dostępnych w standardzie programów
- Brak obsługi radia FM.
- Synchronizacja plików tylko poprzez program iTunes
- Cena urządzeń z iOS
- Brak komunikacji z innymi urządzeniami nie będącymi z rodziną Apple.

Zalety:

- Duża prostota i intuicyjność
- Foldery na aplikacje
- Szybkość i stabilność działania, dobra optymalizacja systemu
- Multitasking
- Świetna obsługa gestów i multitouch
- Spora liczba dostępnych programów w sklepie App Store

- Dostępność aplikacji, które nie są dostępne na innych platformach (głównie gry)
- Możliwość pobierania aplikacji przez komputerowy program iTunes.
- Klient poczty z obsługą Exchange ActiveSync.
- Uniwersalna wyszukiwarka w telefonie
- Przeglądarka Safari z obsługą HTML5
- Rozbudowane funkcje multimedialne i połączenie ze sklepem iTunes

Podsumowanie:

iOS jest zamkniętym, dobrze zabezpieczonym systemem. Przez politykę firmy nie mamy aż tyle dostępnych aplikacji w iTunes i App Store przez co czasem możemy doświadczyć brak czegoś aczkolwiek rekompensuje nam to, że mamy filtry i nie musimy przeglądać setek bezużytecznych aplikacji. Dużą wadą telefonów z systemem iOS jest to, że w Polsce nie wykorzystujemy w pełni jego potencjału gdyż wiele funkcji jest zablokowana. Również wysoka cena za urządzenie może być powodem dla którego większość osób mieszkających na terenie Polski decyduje się na urządzenie z systemem Android.

Dodatkowo przez to, że urządzenia z systemem iOS mają zablokowaną komunikację z telefonami i urządzeniami nie będącymi z rodziną Apple nie jesteśmy w stanie przesyłać zdjęć, filmów i innych danych z naszego urządzenia do urządzenia posiadający system Android.

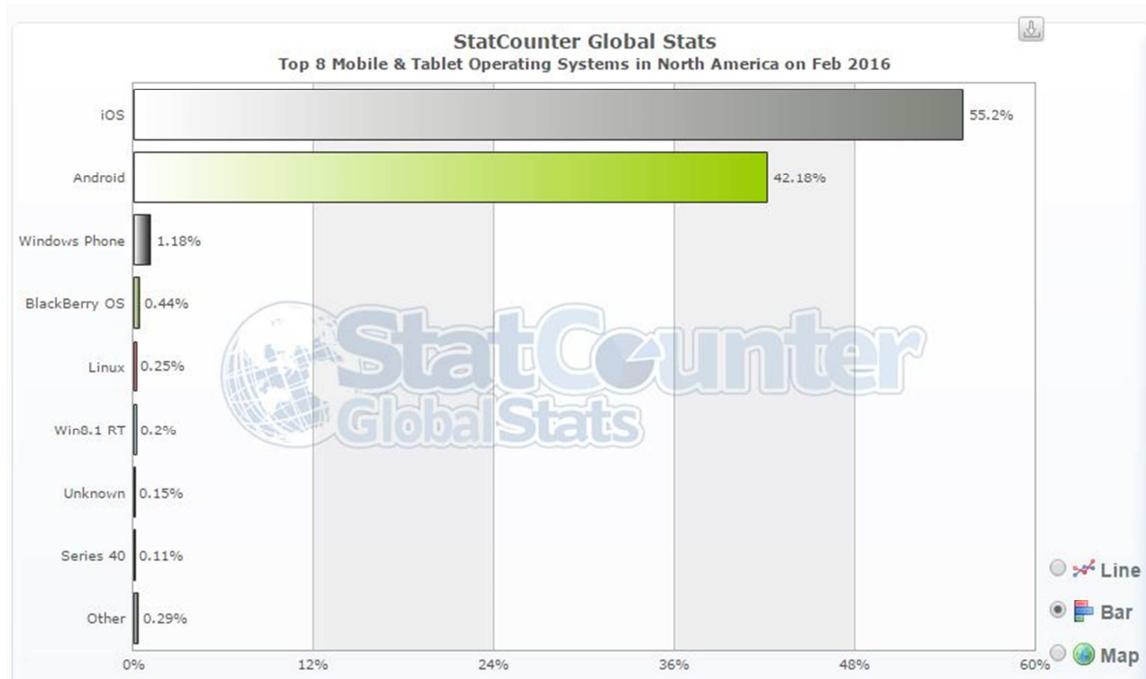
System jest zamknięty więc mamy ograniczone możliwości jego modyfikacji i dostosowywania go do własnych potrzeb. Jak również przez to, że ten system działa tylko na dedykowanych urządzeniach firmy Apple Inc. mamy węższy wybór urządzeń. Chociaż sprawność, szybkość i stabilność może w jakiś sposób być rekompensatą za tą niedogodność.

4.3. Podsumowanie

Android przez swoją otwartość i możliwość istnienia na wielu urządzeniach stał się najbardziej popularnym i najczęściej wybieranym systemem operacyjnym na urządzenia

mobilne. Jest to system który daje nam największą kontrolę nad telefonem. Nie ma problemu z dostępnością do różnych typów aplikacji. Jak również przez otwartość Google Play użytkownik ma do ogromny wybór różnych aplikacji. Jednocześnie to co jest największą zaletą Androida, jest jego największym problemem. Otwartość systemu sprawia, że mniej doświadczony użytkownik może zainstalować niepowołane oprogramowanie które doprowadzi do spowolnienia działania systemu bądź nawet jego uszkodzenia. Z kolei producenci sprzętu, którzy sami ustalają politykę wypuszczania najnowszych wersji Androida, mogą uznać, że dany model nie dostanie nowego oprogramowania, czy też przedłużają pracę nad dopasowaniem najnowszej wersji systemu do swoich urządzeń. Dzieje się tak najczęściej przy tańszych urządzeniach, lub urządzeniach wydawanych przez mniej znane firmy.

iOs jest również dobrze znany systemem. Niemniej jednak przez cenę urządzeń od firmy Apple jak i polityki zabezpieczeń jest rzadziej wybieranym urządzeniem. Również umieszczanie aplikacji na App Store jest trudniejsze niż w przypadku Google Play. iPhone uznawany jest za to jako produkt wyższej klasy który jest lepiej zabezpieczonym i bardziej odpornym na brak doświadczenia użytkownika. Jest za to bardziej stabilny i szybszy. Zwłaszcza w Polsce nie posiada on wielu zwolenników gdyż część funkcjonalności i aplikacji działa tylko na terenie Stanów Zjednoczonych. Jak można to zaobserwować na poniższym rysunku, gdzie iOS dominuje nad Androidem.



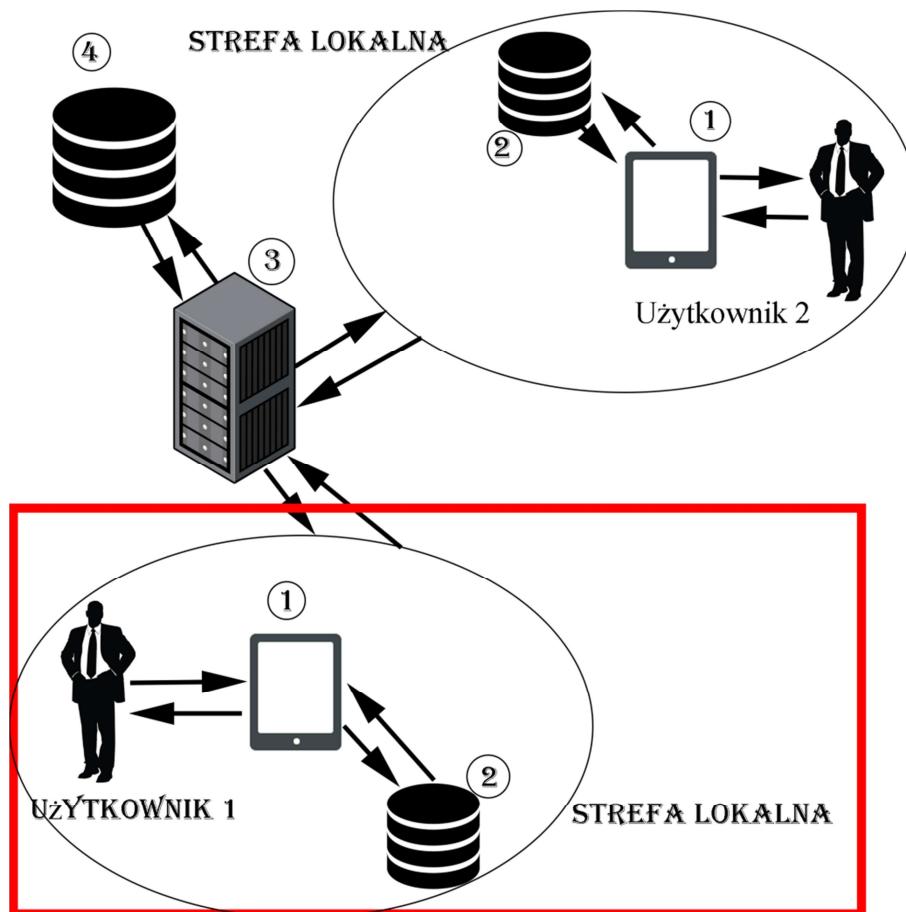
Rysunek 10: Popularność systemów mobilnych w Północnej Ameryce[3]

5. Projekt systemu

Po przeprowadzeniu dokładnej analizy wymagań i popularności systemów mobilnych następnym etapem będzie opis projektu aplikacji. Jest to istotny proces budowy aplikacji, gdyż decyzje podjęte w tej fazie będą miały znaczący wpływ na wybór technologii i cały etap implementacji.

5.1. Architektura systemu

Po przeprowadzeniu analizy strukturalnej została zbudowana aplikacja która wykorzystuje dwie bazy danych (nr 2 i 4) oraz dodatkowy serwer (3) do obsługi rozgrywki sieciowej. Na potrzeby przeprowadzenia badań ograniczymy się jednak do schematu oznaczonego w czerwonej ramce. Schemat ten pozwala na użytkownie aplikacji bez stałego połączenia z internetem. Gdzie (1) jest to urządzenie z którym użytkownik wchodzi w interakcję, natomiast (2) jest to lokalna baza danych.



Rysunek 11: Schemat architektury systemu

5.2. Wybór technologii implementacji

Często ignorowaną lub nie braną na poważnie kwestią jest dobór odpowiednich narzędzi i technologii. W praktyce, jest to kluczowy element mający wpływ na końcowy sukces projektu. Cel projektu narzuca nam stworzenie aplikacji na urządzenia mobilne. Dlatego po przeprowadzonej analizie popularności w celu wykonania podzielonego celu jakim jest dotarcie do jak największej liczby osób, wybrałem platformę Android jako system mobilny na który będę tworzyć moją aplikację. W związku z czym, środowiskiem w którym będę pisał aplikację i ją rozwijać będzie Android Studio.



Android Studio jest oficjalnym zintegrowanym środowiskiem programistycznym (IDE) dla tworzenia aplikacji w systemie Android. Bazuje on na środowisku IntelliJ IDEA.[2] Najważniejszymi rzeczami wziętymi z IntelliJ które Android Studio będzie oferowało są to:

- Technologia Gradle – narzędzie stworzone przez Google do automatyzacji projektów.
- Budowanie wariantów (variants) oraz generowanie wielu plików apk.
- Szablony pozwalające budować wspólne funkcje dla aplikacji.
- Bogaty edytor układów (layout editor) z wsparciem dla edycji przeciągnij i upuść.
- Narzędzie Lint pozwalające sprawdzać wydajność, użyteczność, kompatybilność wersji oraz inne problemy.
- Wbudowane wsparcie dla platformy Google Cloud, pozwalające na łatwiejszą integrację z Google Cloud Messaging oraz App Engine.
- Zawiera wbudowany emulator pozwalający na testowanie aplikacji na różnych wirtualnych urządzeniach (pozwala sprawdzić jak aplikacja wygląda i działa na różnych urządzeniach z systemem Android bez potrzeby posiadania fizycznych urządzeń)

Wybrane środowisko pracy narzuca nam również język programowania jaki należy użyć. Do napisania tej aplikacji użyję języka Java.



Java jest to obiektowy język programowania. Jest to język służący do tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną. Język cechuje się silnym typowaniem. Jego podstawowe koncepcje zostały przejęte z języka Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz z języka C++ (duża część składni i słów kluczowych). Javę najbardziej charakteryzuje kilka kluczowych koncepcji:

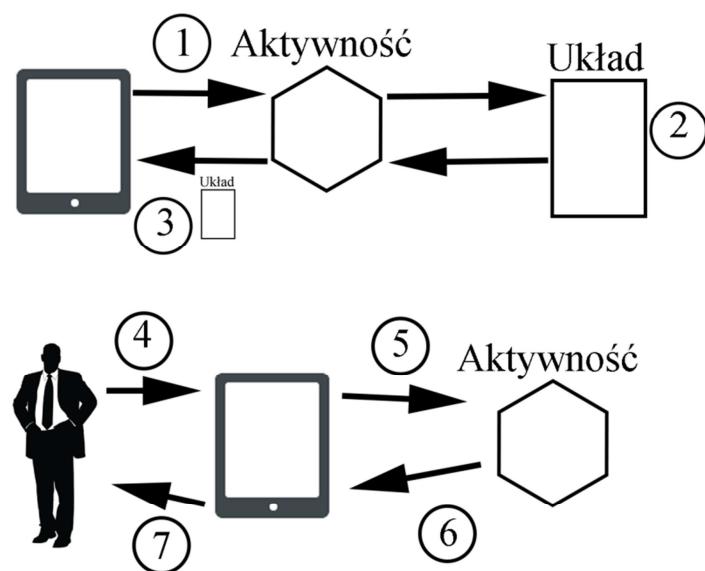
- Obiektowość – Wszelkie dane i akcje na nich podejmowane są pogrupowane w klasy obiektów. O obiekcie można myśleć jako o samoistnej części programu, która może przyjmować określone stany i ma określone zachowania, które mogą zmieniać te stany bądź przesyłać dane do innych obiektów.
- Dziedziczenie – W Javie wszystkie obiekty są pochodną obiektu nadzawanego (jego klasa to Object), z którego dziedziczą podstawowe zachowania i właściwości.
- Niezależność od architektury – Tę właściwość Java posiada dzięki temu, że kod źródłowy programów pisanych w Javie kompiluje się do kodu pośredniego. Powstały w ten sposób kod jest niezależny od systemu operacyjnego i procesora. Zostaje wykonany przez tzw. wirtualną maszynę Javy, która tłumaczy kod uniwersalny na kod dostosowany do specyfikacji konkretnego systemu i procesora.
- Sieciowość i obsługa programowania rozproszonego – Dzięki wykorzystaniu reguł obiektowości, Java nie widzi różnic między danymi płynącymi z pliku lokalnego a danymi z pliku dostępnego przez http czy FTP.

- Niezawodność i bezpieczeństwo – W zamierzeniu Java miała zastąpić C++ - obiektowego następcę języka C. Twórcy Javy rozpoczęli pracę od rozpoznania cech języka C++, które są przyczyną największej liczby błędów programistycznych, by stworzyć język prosty w użyciu , bezpieczny i niezawodny.



Tworzenie programów na Androida różni się nieznacznie od samego pisania programów z użyciem samej Javy. Część opisów dotyczących sposobu działania i procesów kompilacji plików pochodzi z książki [6] z której czerpałem swoją wiedzę podczas tworzenia aplikacji.

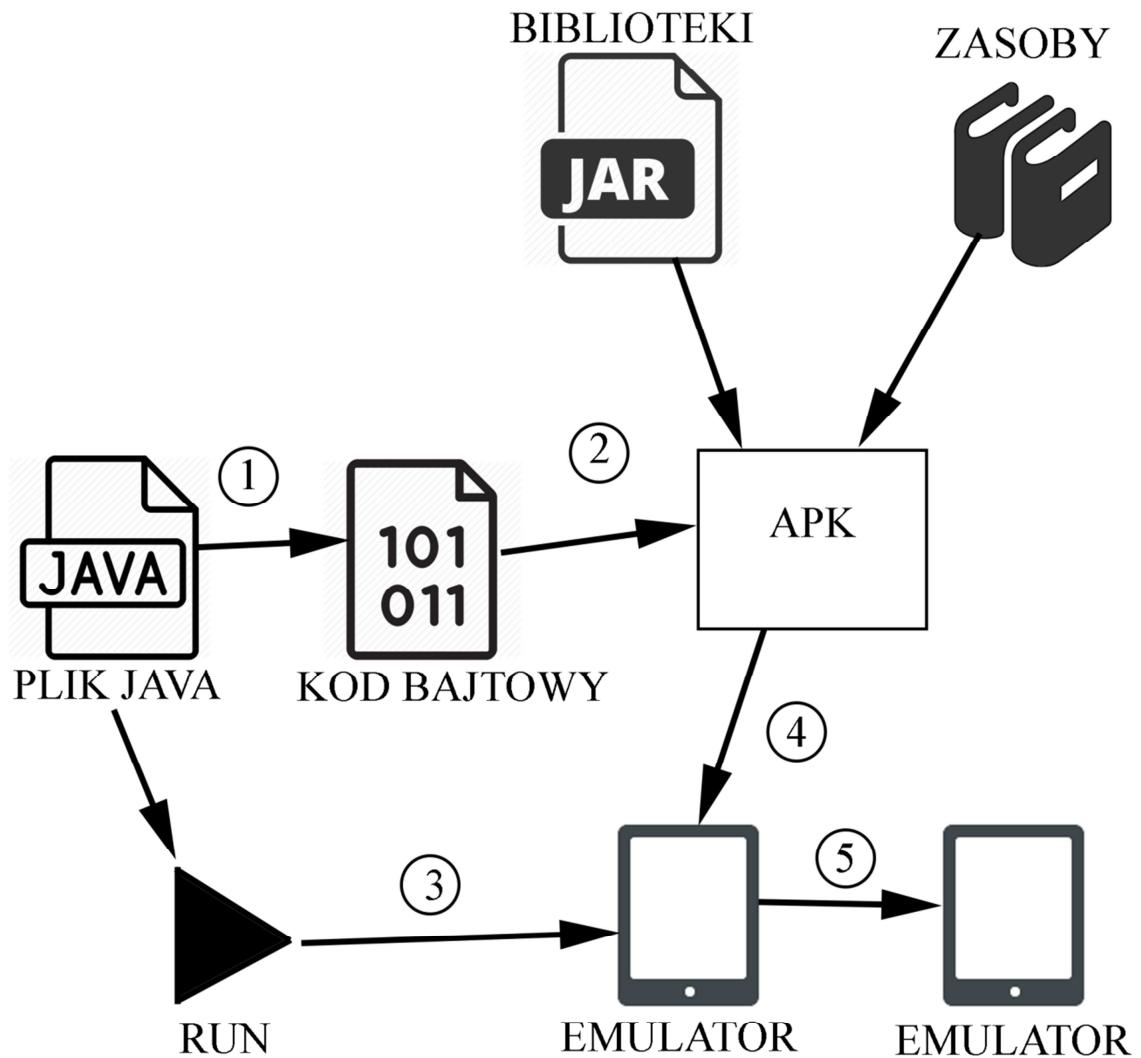
Android wykorzystuje pliki .xml w których tworzy się interfejsy graficzne aplikacji. Każdy obiekt w interfejsie dziedziczy po klasie View. Aplikacja na Androida to zbiór układów i aktywności które wspólnie wykonują określone zadanie.



Rysunek 12: Schemat przedstawiający jak działają aplikacje na Androidzie[6]

1. Urządzenie uruchamia aplikację i tworzy obiekt aktywności
2. Obiekt aktywności określa układ
3. Aktywność nakazuje urządzeniu wyświetlenie układu na ekranie
4. Użytkownik prowadzi interakcję z układem wyświetlonym na ekranie
5. Aktywność odpowiada na te interakcje, wykonując odpowiedni kod aplikacji
6. Aktywność aktualizuje treści w układzie...
7. Które użytkownik widzi na ekranie

Zasadniczą różnicą między zwykłymi programami pisanyymi za pomocą języka Java a programami pisanyymi jako aplikacje dla systemu Android jest to, że kod przeznaczony do uruchomienia na Androidzie zostaje skompilowany, spakowany, wdrożony i wykonany na urządzeniu bez użycia wirtualnej maszyny Javy – Java VM. Zamiast tego działa on w środowisku uruchomieniowym Android (ART), a na starszych urządzeniach – w poprzedniku ART., środowisku uruchomieniowym noszącym nazwę Dalvik. Oznacza to, że pisany kod w Javie zostaje skompilowany do postaci plików .class używając kompilatora Javy. Później natomiast pliki klasowe zostają zapisanie w postaci pliku DEX. Jest to plik który zajmuje mniej miejsca i jest bardziej wydajniejszy od zwyczajnych kodów bajtowych. Proces jest stworzony po to aby oszczędzać baterię i zasoby na urządzeniu mobilnym, gdyż wiadomy jest, że takie urządzenie nie dysponuje stałym zasilaniem i mocą aby można było uruchamiać wiele programów tak samo jak na np. komputerach stacjonarnych.



Rysunek 13: Proces komplikacji pliku APK i uruchamianie emulatora[6]

1. Pliki źródłowe Javy zostają skompilowane do postaci kodów bajtowych
2. Zostaje utworzony pakiet aplikacji, czyli plik APK. Plik APK zawiera skompilowane pliki Javy oraz wszystkie biblioteki i zasoby niezbędne do uruchomienia aplikacji.
3. Jeśli emulator jeszcze nie działa, to zostanie uruchomiony, a na nim zostanie uruchomione wybrane urządzenie wirtualne.
4. Po uruchomieniu emulatora i wybranego urządzenia wirtualnego zostanie na nie wgrany plik APK aplikacji, po czym aplikacja zostanie zainstalowana w systemie.

5. AVD uruchomi główną aktywność aplikacji. W tym momencie aplikacja zostanie wyświetlona na ekranie AVD i będzie gotowa do testowania

Proces 4 i 5 można pominąć gdy testujemy program na fizycznym urządzeniu. Wtedy aplikacja zostaje zainstalowana i uruchomiona na naszym podłączonym urządzeniu.

Ważną wiedzą na temat tworzenia aplikacji na urządzenia z systemem Android jest to, że musimy przestrzegać struktury plików. Android Studio zrobi całą strukturę za nas w momencie gdy utworzymy nowy projekt. Niemniej jednak warto wiedzieć o istnieniu kilku ważnych plików w tej strukturze gdyż czasem musimy ręcznie je zmieniać, w celu osiągnięcia jakiegoś zamierzonego celu np. Aby każdy widok był wyświetlany poziomo i nie zmieniał się gdy użytkownik obróci urządzenie.

- R.java – Każdy projekt aplikacji na Androida musi zawierać plik o nazwie R.java, który jest tworzony i umieszczany w automatycznie wygenerowanych katalogach. Służy on Androidowi za pomoc przy zarządzaniu zasobami aplikacji.
- MainActivity.java – (plik może nazywać się inaczej, zależy jak go nazwiemy samodzielnie). Definiuje on aktywność. Aktywność odpowiada za zarządzanie poczynaniami użytkownika i reagowania na jego interakcje z aplikacją.
- Activity_main.xml – plik ten definiuje układ. Jak wspomniałem wyżej Aktywność i układ stanowią jedność. Układ odpowiada jak ma wyglądać dana aplikacja.
- String.xml – jest to plik który przechowuje wszystkie łańcuchy znakowe w formacie <identyfikator, wartość>. Za jego pomocą można w jednym miejscu umieszczać wszystkie teksty (zdania, wyrazy) które pojawiają się w naszej aplikacji. Pozwala to potem na łatwiejszą zmianę informacji bądź też, gdy przyjdzie taka potrzeba, przetłumaczeniu tekstów na inny język. Dzięki temu wszystko znajduje się w jednym miejscu i nie trzeba przeszukiwać całego kodu programu.
- AndroidManifest.xml – Każda aplikacja na Androida musi zawierać w swoim katalogu głównym ten plik. Jest to plik manifestu który zawiera kluczowe informacje dotyczące aplikacji, takie jak komponenty z których się składa, wymagane biblioteki i deklaracje.



Jak wiadomo, każda aplikacja potrzebuje przechowywać swoje dane. W Androidzie robi się to za pomocą bazy danych SQLite. Jest to specjalny rodzaj bazy danych która nie wymaga uruchomienia procesu RDBMS³. Dlaczego akurat SQLite:

- Jest nieskomplikowana – Większość baz danych wymaga do prawidłowego działania specjalnego serwera baz danych. SQLite jest po prostu zwyczajnym plikiem.
- Jest zoptymalizowana do wykorzystywania przez jednego użytkownika – Moja aplikacja jest jedynym programem, który będzie komunikował się z bazą danych. Nie będzie potrzeby identyfikowania się przy użyciu nazwy użytkownika i hasła.
- Jest stabilna i szybka – Bazy danych SQLite są zadziwiająco stabilne. Są zdolne do obsługi transakcji. Co oznacza, że jeśli coś pójdzie nie tak w momencie gdy aktualizujemy kilka różnych danych to SQLite wycofa wszystkie wprowadzone zmiany. Dodatkowo kod odczytu i zapisu został napisany w języku C. Co sprawia, że działa to jeszcze szybciej a dodatkowo ogranicza ilość wymaganej mocy procesora do prawidłowego działania. Co powoduje mniejsze zużycie baterii.

Gdy stworzymy pomocnika SQLite i po raz pierwszy nasza aplikacja będzie się chciała połączyć z bazą danych Android samodzielnie utworzy odrębny katalog dla naszej aplikacji i rozpocznie przechowywanie w tym katalogu naszą bazę danych. Plik znajduje się w katalogu: `/data/data/com.robertmatejczuk.battleship/databases.[6]`

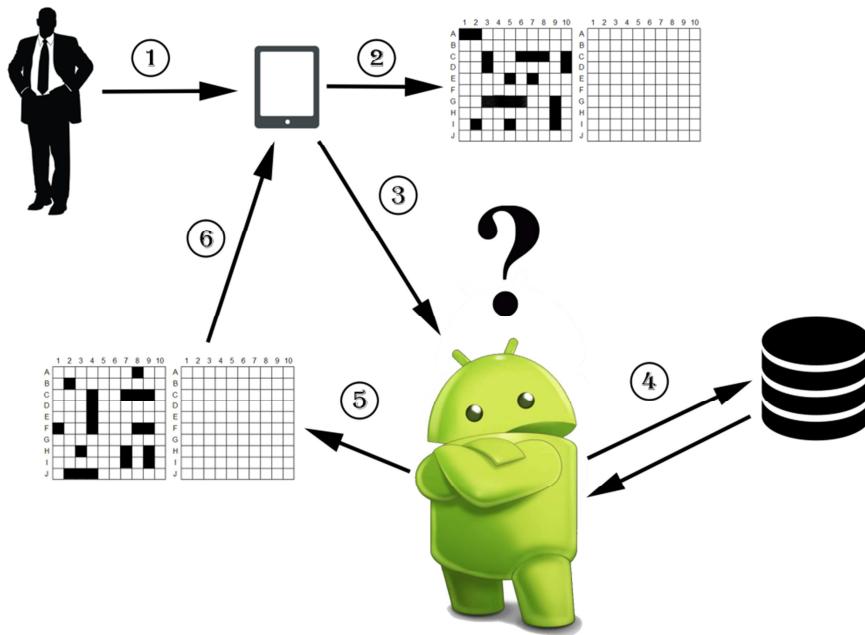
Znajdują się tam dwa pliki. Jeden z nich jest to plik bazy danych a drugi z dopiskiem –journal w nazwie oznacza plik magazynu.

Dokładny proces opisany jest w rozdziale 7.

³ Ang. Relational Database Management System – System zarządzania relacyjną bazą danych

6. Implementacja systemu

Implementację systemu zacznę od przedstawienia grafu użycia aplikacji:

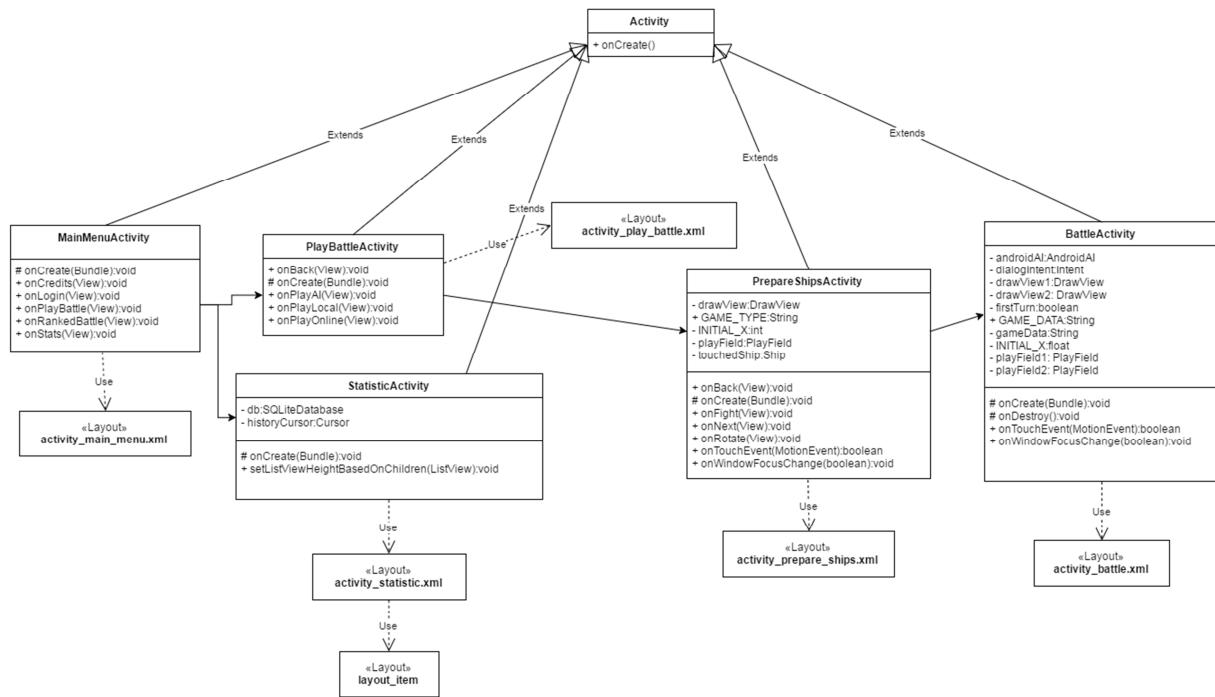


Rysunek 14: Graf użycia aplikacji

1. Gracz wybiera z głównego menu rozegranie bitwy przeciwko Androidowi
2. Pokazuje mu się układ poprzez który rozmieszcza swoje statki. Gdy będzie gotowy zatwierdza swoje ustawienie i rozpoczyna bitwę.
3. W tym momencie Android dostaje powiadomienie o gotowości gracza. Zastanawia się jak ustawić swoje statki.
4. Łączy się w tym celu z bazą danych SQLite znajdującą się na urządzeniu i pobiera z niej informacje na temat wcześniejszych rozgrywek z tym graczem.
5. Następnie na podstawie zebranych danych rozmieszcza swoje statki.
6. Gdy jest gotowy przekazuje informacje o tym, że rozstawił swoje statki i bitwa może się rozpocząć.

6.1. Diagram klas i opis działania algorytmu

Moja aplikacja posiada 11 klas i 6 układów. Klasy odpowiedzialne za wygląd aplikacji i funkcjonalność odpowiadającą za reagowanie na poczynania użytkownika nazywamy w Androidzie aktywnościami. Poniższy graf przedstawia jak aktywności współpracują ze sobą:

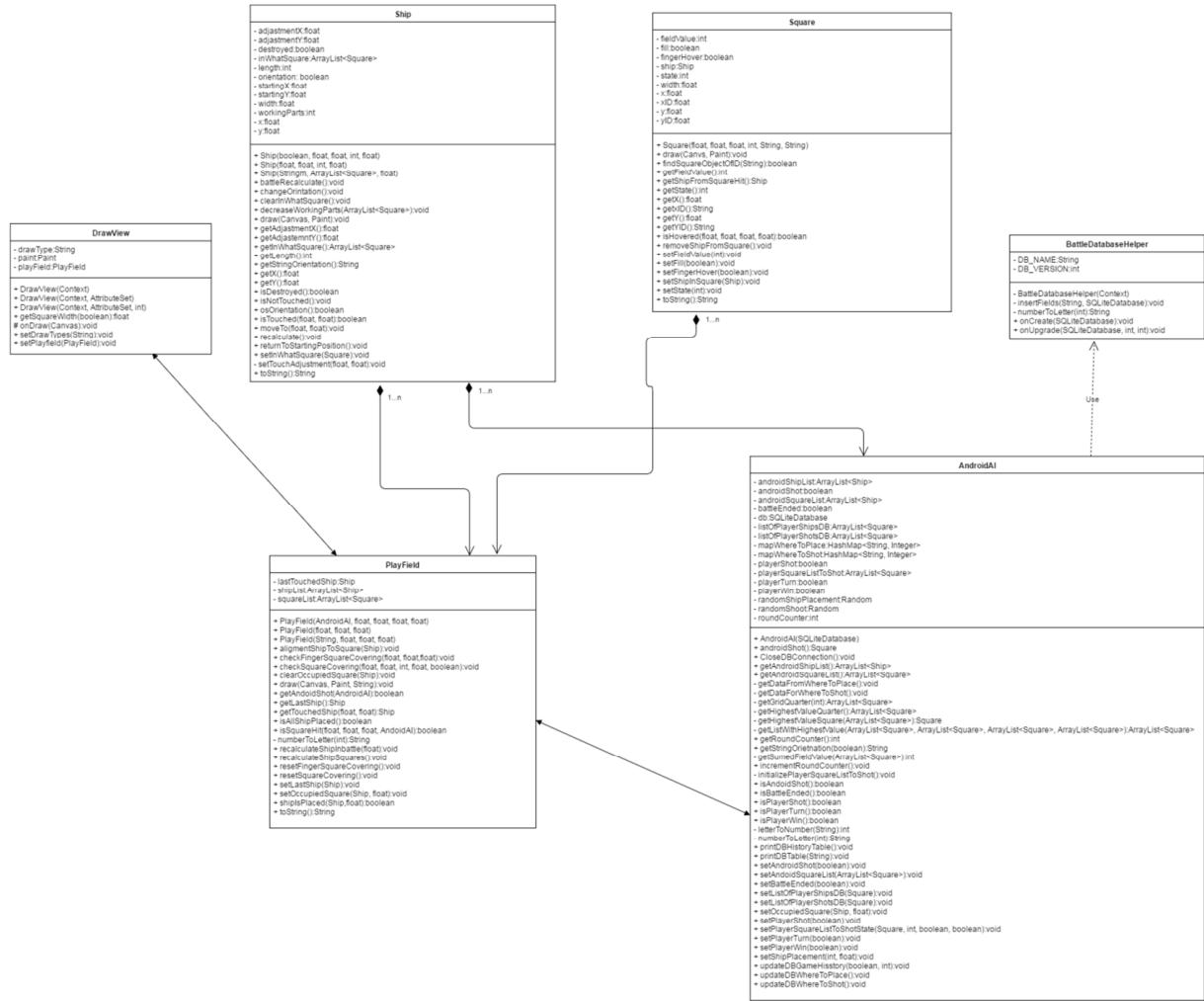


Rysunek 15: Diagram klas Activity wraz z wykorzystywanyimi przez nie układami

Aktywnością startową jest `MainMenuActivity`. Uruchamiana jest przez Andoida w momencie uruchomienia aplikacji. Użytkownik z tego miejsca może przejść do drugiej aktywności aby wybrać tryb gry nie rankingowej albo sprawdzić swoje statystyki.

Przechodząc do drugiej aktywności `PlayBattleActivity` użytkownik może rozpocząć bitwę z Andoidem. Po wybraniu tego trybu gry, gracz zostaje przeniesiony do aktywności `PrepearShipActivity`, gdzie rozmieszcza swoje statki. Aktywność ta rozpoczyna korzystanie już z klas odpowiadających za logikę aplikacji. Wykorzystuje tutaj klasę `DrawView` oraz `PlayField`, która natomiast wykorzystuje klasy `Ship` i `Square` które odpowiednio odpowiadają za logikę tworzenia statków i pól planszy. Klasa `Playfield` jest tak naprawdę klasą która łączy pozostałe klasy i zarządza mechanizmami gry. Klasa `PrepearShipActivity` jest klasą odpowiedzialną za reagowanie poczynań gracza i wywoływanie odpowiednich rzeczy z

pozostałych klas, jak również odpowiada za wywołanie układu i zachowanie prawidłowego interfejsu.

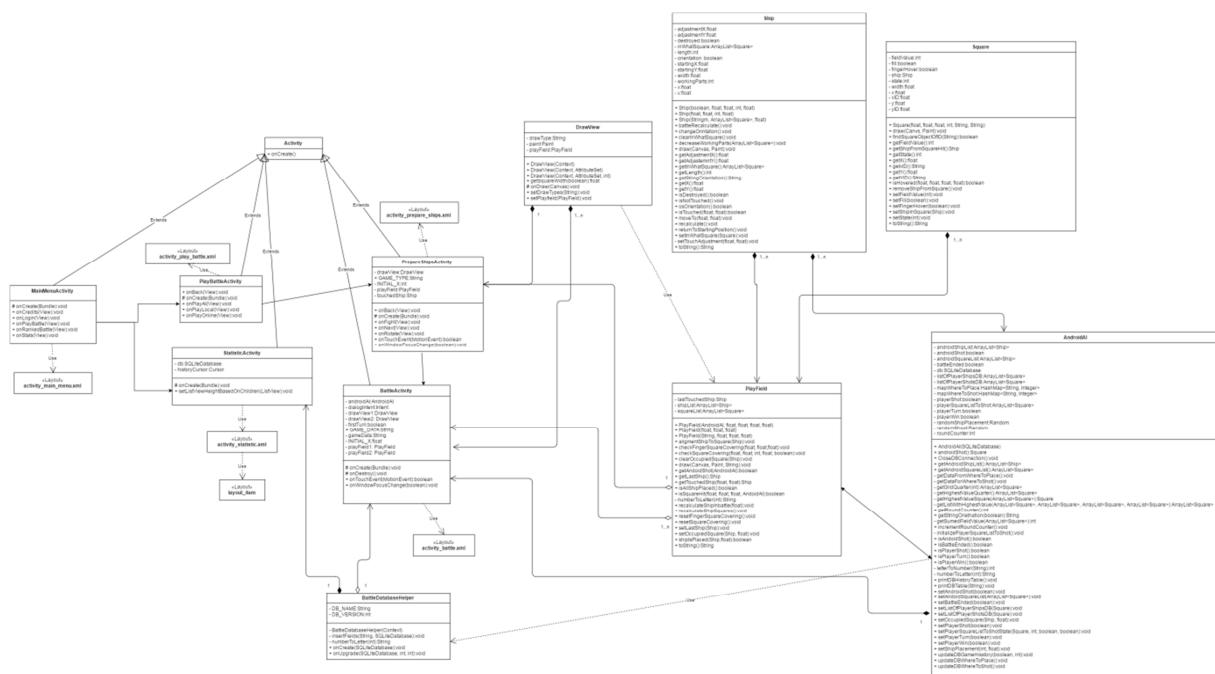


Rysunek 16: Diagram klas odpowiedzialnych za logikę programu

Gdy gracz zakończy rozmieszczenie swoich statków, zgłasza gotowość do bitwy. W tym czasie BattleActivity wywołuje nowy układ bitwy zapisany w activity_battle.xml i wywołuje klasy DrawView i PlayField w celu narysowania planszy, pobiera też intencję ze wcześniejszej aktywności w celu odpowiedniego odtworzenia obiektów statków które gracz rozmieścił. Ponowne przydzielenie miejsc statkom gracza na nowo stworzonej planszy zajmuje się klasa PlayField, która w jednym ze swoich konstruktorów ma zaimplementowany algorytm do przetwarzania łańcucha znaków przesłanego w intencji. Odpowiednio go dzieli i przesyła do klasy Ship wraz z listą obiektów typu Square odpowiadającej planszy gracza.

Następnie statki gracza zostaną już ustawione zostaje stworzony obiekt klasy AndoidAI który zostaje przekazany również do nowego obiektu PlayFielda który odpowiednio powiadamia Androida o tym aby przygotował swoje statki. Gdy Android przygotuje swoje statki cały układ ukazuje się graczowi i może on rozpocząć bitwę.

Gdy dotknie któregoś z pól planszy Androida zostanie podświetlone pole które aktualnie gracz dotyka, może one dowolnie przesuwać palec po ekranie aby zmienić wybrane pole. W momencie gdy podniesie palec strzał zostanie oddany. Klasa BattleActivity przekaże stosowny komunikat do odpowiedniej klasy PlayField (dwa obiekty, jeden dla gracza i zarządzanie jego statkami, drugi dla Androida i zarządzanie jego statkami. Tak aby nie miały dostępu do siebie i nie było możliwości, że któraś ze stron dowie się, gdzie przeciwnik umieścił statki). Gdyby gracz dotknął ekranu i nie trafił w żadne pole planszy Androida, mechanizm zabezpiecza się i w ten sposób taki ruch nie będzie liczony jako strzał. Więc gracz nie straci swojej kolejki.



Rysunek 17: Diagram klas w całości

Gdy gracz podniósł palec klasa PlayField sprawdza informacje czy w tym miejscu znajdował się statek (Klasa PlayField odpowiedzialna za statki Andoida, posiada jego listę pól typu `ArrayList<Square>` i sprawdza w ten sposób czy na tym polu jest statek czy go nie ma. Jeśli jest to zostaje wykonany odpowiedni algorytm który odpowiada za zmianę malowania

pola i ustawiona zostaje dla tego pola wartość pola fill = true, która powoduje, że przy kolejnym rysowaniu interfejsu, pole zostanie zamalowane na odpowiedni kolor (czerwone w przypadki trafienia, niebieskie w przypadku „pudła”).

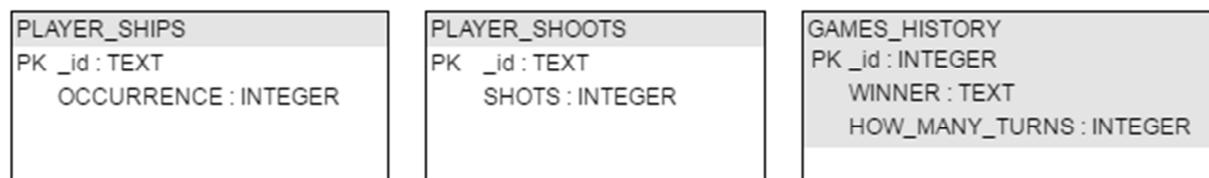
Następnie jeśli gracz nie trafił w statek Android wykonuje swój strzał. Kontynuuje ostrzeliwanie pól gracza do momentu aż nie spadnie. Przy każdym trafieniu statku, pobierana jest referencja z trafionego pola która zawiera referencję do statku który leży na tym polu i wywoływana jest metoda obniżająca wytrzymałość statku. Kiedy wytrzymałość spadnie do 0 (w zależności ile masztowy jest statek, 2 masztowy oznacza, że jego wytrzymałość jest równa 2) status statku określony w polu destroy zostaje zmieniony na wartość TRUE oraz algorytm przechodzi przez listę inWhatSquare którą ten statek zawiera i zamienia stan wszystkich pól które mają swój stan na -1 (oznaczają otoczkę statku) na stan równy 3 (co oznacza pudło) jak również wartość pola fill zostaje ustawiona na TRUE. W ten sposób zostaje obsłużony wybuch statku a wszystkie pola wokół statku zostaną odsłonięte i podświetlone na niebiesko. Następnie jeśli statek został zniszczony przez androida następuje zaktualizowanie listy potencjalnych pól do strzelania. Dzieje się to tak, gdyż nie można strzelić dwa razy w to samo pole a Android nie otrzymał informacji jakie pola zawierał statek, bo nie ma dostępu do listy statków gracza (było by to po prostu nieuczciwe gdyby Android znał położenie statków gracza). Dlatego musi mieć zaktualizowaną listę aby w ten sposób „zobaczyć” zmiany jakie zaszły na polu gracza i nie strzelać w te pola które już zostały odsłonięte przez wybuch.

Gra się toczy do momentu gdy ktoś ze stron nie zniszczy wszystkich statków przeciwnika. Gdy statek zostaje zniszczony zostaje on usunięty z listy statków jego posiadacza. Dlatego warunkiem kończącym grę jest moment kiedy lista statków któregoś z uczestników bitwy będzie równa 0. W tym momencie zostaje ustawione pole battleEnded na TRUE i w zależności kto oddał ostatni strzał zostaje przekazana informacja zmieniająca pole playerWin na TRUE jeśli gracz wygrał albo na FALSE jeśli zwyciężył Android. Następnie w BattleActivity przez zakończeniem metody odpowiadającej za wyłupywanie dotyku ekranu przez gracza zostają sprawdzone powyższe pola. Jeśli mają wartości TRUE, to zostaje stworzone i wywołane okno powiadające o zakończeniu gry. W zależności od wartości pola playerWin odpowiedni tekst zostaje na tym powiadomieniu wyświetlony. Użytkownik może teraz wybrać czy chce wrócić do głównego menu (wywołanie aktywności MainActivity z flagą aby zakończyć wszystkie procesy na stosie) albo wrócić do ustawień statków i rozpoczęcie bitwy na nowo (wywołanie metody Finish która zakańcza aktualną aktywność).

6.2. Baza danych SQLite

Jak już wcześniej wspomniałem, baza danych SQLite jest lokalną bazą i jednym ze sposobów przechowywania trale informacji na urządzeniu z systemem Android.

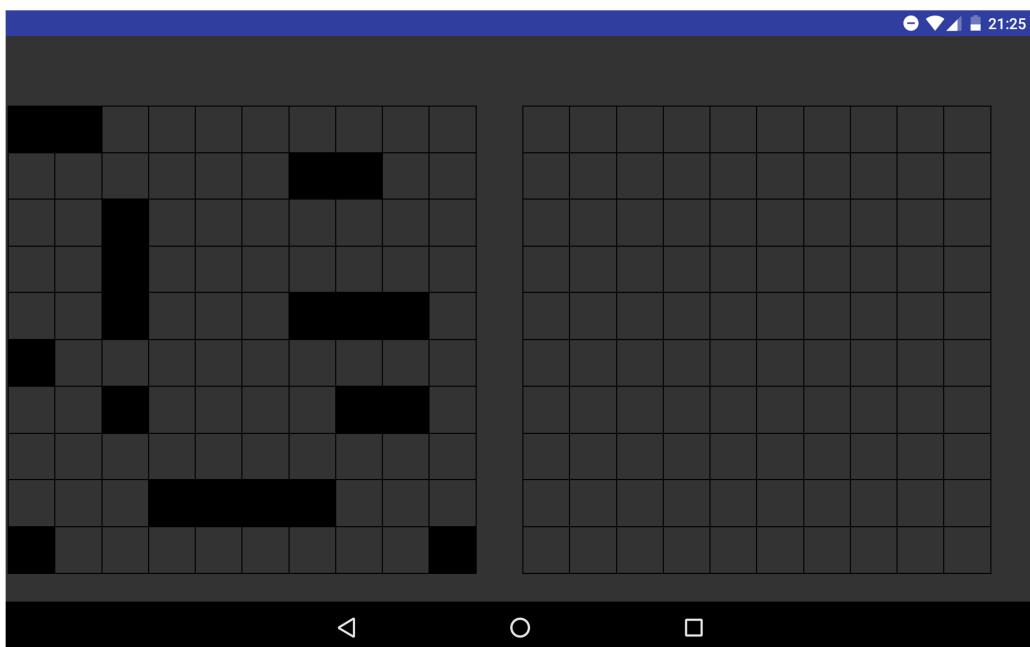
Wspomniałem też o wadach i zaletach tej bazy. Teraz natomiast przedstawię jak ta baza danych wygląda oraz jak się tworzy taką bazę danych.



Moja baza danych zawiera trzy tabele. Tabela `PLAYER_SHIPS` jest tabelą odpowiedzialną za przechowywanie informacji na temat położenia statków gracza z wcześniejszych gier. Należy tutaj zwrócić uwagę, że do bazy danych trafiają tylko informacje o statkach które Android samodzielnie odkrył. Jeśli gracz szybko pokona Androida i nie da mu odkryć wszystkich statków to nie dowie się on w żaden sposób o położeniu reszty statków. Tak samo jak i gracz jeśli zostanie wcześniej pokonany to nie będzie wiedział gdzie Android umieścił swoje statki. Tabela ta przechowuje kolumnę `_id` która odpowiada ID pola na planszy np. G5. Natomiast kolumna `OCCURRENCE` odpowiada za zapisanie ile razy w tym polu pojawił się statek. Jak łatwo się domyśleć. Aktualizacja bazy danych polega na przekazaniu listy (w naszym przypadku jest to lista o nazwie `listOfPlayerShipsDB`), która zapamiętywała podczas gry gdzie Android odkrył statki gracza. Tabela `PLAYER_SHOTS` działa na bardzo podobnej zasadzie, z tym, że jej wartości dotyczą pól w które gracz oddał strzał. Również aktualizacja odbywa się za pomocą listy (`listOfPlayerShotsDB`) która podczas gry zapisuje sobie w które pola gracz oddał strzał. Pola które zostały odsłonięte przez wybuch statku nie zostają zaliczone do strzałów gracza. Tabela `GAMES_HISTORY` zapisuje informacje statystyczne o przeprowadzonych bitwach. Zawiera trzy kolumny. Kolumna `_id` wskazuje nr bitwy jak i służy też do sprawdzenia ile bitw gracz rozegrał z Androidem. Kolumna `WINNER` przechowuje tekstową informację o zwycięzcy. Jeśli zwycięży gracz w kolumnie pojawi się zapis `PLAYER` natomiast jeśli zwycięży Android to w kolumnie będzie widniał tekst `ANDROID`. Kolumna `HOW_MANY_TURNS` służy do sprawdzenia czy Android nie tylko nauczył się wygrywać z graczem ale czy robi to znacznie szybciej niż wcześniej. Kolumna ta zawiera informacje ile tur trwała bitwa. Licznik tur zwiększa się za każdym razem gdy ruch wykona gracz i Android. Nie ma znaczenia kto zacznie grę.

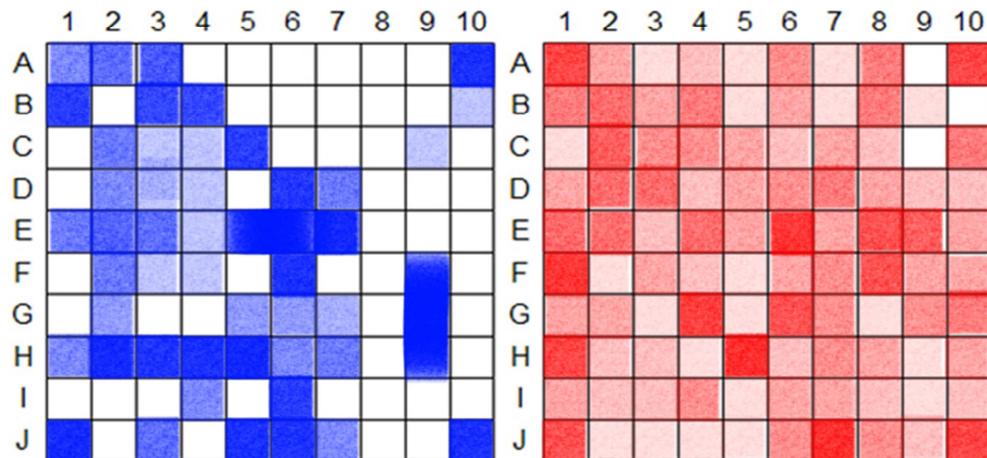
7. Adaptacyjny system strategii Androida

W tym rozdziale opiszę dokładnie jak działa algorytm adaptacyjny dla Androida który został zaimplementowany do wyżej opisanej aplikacji. Zaczniemy od przedstawienia jak wygląda oczami gracza plansza do bitwy gdy gra się rozpocznie:



Rysunek 18: Widok bitwy oczami gracza

Natomiast Android widzi planszę w taki sposób:



Rysunek 19: Widok bitwy oczami Androida

Po prawej stronie Android widzi gdzie gracz najczęściej strzelał. Bazował na tych informacjach podczas ustawienia swoich statków na planszy. Im kolor intensywniejszy tym częściej gracz oddawał w to miejsce strzał.

Pole po lewej stronie natomiast są to informacje jak gracz w przeciągu pewnej liczby bitew rozkładał swoje statki. Im intensywniejszy kolor tym gracz częściej w to miejsce umieszczał statki.

W momencie gdy gracz zgłasza gotowość i chce rozpocząć zostaje stworzony obiekt Androida który otrzymuje referencję do bazy danych z którą następnie się łączy w celu pobrania danych. Pobiera wszystkie informacje z tablicy PLAYER_SHIPS oraz PLAYER_SHOTS do dwóch różnych HashMap. Mapa mapWhereToPlace służy do przechowywania informacji z tabeli PLAYER_SHOTS (czyli tam gdzie gracz strzelał). W ten sposób wie gdzie gracz przeważnie nie celuje i tam umieszcza statki. Za umieszczanie statków odpowiada metoda setShipPlacement(int length, float width). Metoda ta przyjmuje dwa parametry. Jeden parametr to długość statku, drugi natomiast szerokość jednego pola. Gdy ta metoda zostanie wywołana Android ma już listę obiektów typu Square (androidSquareList) która zawiera referencje do obiektów pól odpowiadające planszy na której Android może umieszczać swoje statki. Ale zanim umieści statek musi wykonać trochę kalkulacji i obliczeń na temat pól w które opłaca się umieścić. W tym celu przeszukuje schodkowo całą planszę wzduż i w szerz w poszukiwaniu kombinacji o najmniejszej wartości. Schodkowo oznacza to, że pętla for ma dodatkowy licznik który zwiększa się o jeden za każdym razem gdy zostanie zsumowana odpowiednia liczba pól odpowiadająca długości statku przekazana jako pierwszy parametr. Poszukiwanie kombinacji pól zaczyna zawsze od umieszczenia 4-masztowca.

Przykład:

Wywołana została metoda dla 3-masztowca. Android przeszukuje swoją listę posegregowaną w sposób poziomy (A1, A2, A3). W taki sposób sprawdza wartości tych trzech pól pobierając ich wartości z mapy mapWhereToPlace przekazując jej klucz będący ID pola np. A1.

W taki sposób zsumowane zostają 3 pola. Są to pierwsze sprawdzone trzy pola dlatego ich wartość jest mniejsza niż wartość pola minimum, które zawiera maksymalną wartość jaką typ Integer może przyjąć. Uznamy też, że te pola nie mają wartości 1 (zajęte przez statek),

bądź wartości (-1) otoczka statku. Gdyż musimy pamiętać, że 4 masztowiec został już gdzieś umieszczony na planszy. Tak więc te 3 pola okazują się posiadać jak na razie najmniejszą sumę pól. Gdy pole counter1 osiągnie długość statku (w naszym przypadku 3) to zostaje wyzerowana, counter2 (który odpowiada za wspomniany dodatkowy licznik w celu przeszukiwania schodkowego). Następnie wartość zmiennej j w pętli wewnętrznej for zostaje ustawiona na wartość counter2 oraz następuje sprawdzenie czy suma tych pól jest większa niż aktualne minimum. Jak wspomniałem, są to pierwsze 3 pola które znaleźliśmy, dlatego z dużym prawdopodobieństwem są to pola o najmniejszej wartości. Dlatego lista przechowująca potencjalne zestawy pól zostaje wyczyszczona, lista zapamiętująca orientację statku jaka została znaleziona (na początku przeszukujemy planszę dla orientacji poziomej). Następnie pola zostają dodane do pomocniczej listy o nazwie potentialSquares. Do drugiej listy orientationList zostaje dodana orientacja statku. Suma pól zostaje wyszerowana a helperList (lista przechowująca kombinacje pól zanim zostaną pola zsumowane) zostaje wyczyszczona i przygotowana na pobieranie kolejnych kombinacji pól.

Tak więc dotarliśmy do końca pętli for i teraz wcześniej ustawiona wartość J zostaje zwiększona o 1. Więc zaczniemy sprawdzanie od pola A2 i pobierzemy trzy kolejne pola włącznie z A2. Czyli teraz sprawdzamy sumę pól dla A2, A3 oraz dla A4. Następnie A3, A4, A5. Gdy dojdziemy do końca, czyli zaistnieje sytuacja, że skończy nam się plansza do przeszukiwania poziomo w tym rzędzie A9, A10, koniec planszy. To wartości zostaną zresetowane (helperList, suma pól, counter1 oraz counter2) i zaczniemy przeszukiwanie pól w rzędzie B. I tak przeszukujemy całą planszę od lewej do prawej a później od góry do dołu.

W momencie gdy trafimy na kombinację pól które mają mniejszą wartość to czyścimy listy (potentialSquares oraz orientationList) i na miejsce usuniętych elementów dodajemy nowo znalezione o mniejszej sumie pól. Gdy natomiast trafimy na kombinację pól o takiej samej wartości jaka już aktualnie została znaleziona, to dodajemy je również do listy wraz z orientacją.

Gdy już przeszukamy całą planszę wzduż i wszerz sprawdzamy rozmiar listy orientacji. Przez to, że dodawaliśmy do niej po jednym elemencie na każdą kombinację wiemy dokładnie ile kombinacji pól udało nam się znaleźć. Następnie losujemy pseudolosową liczbę z klasy Random aby wybrać którąś ze znalezionych kombinacji pól. Gdybyśmy tego nie zrobili to statki były by umieszczanie przeważnie w górnej części bądź z lewej strony planszy. A tak to jest zawsze jeszcze dodatkowy element losowości który

zmienia nam ustawienie za każdym razem gdy jest to możliwe. Teraz możemy dodać nasz statek do listy statków Androida i ustawić odpowiednim zajętym polom stan 1 w liście pól. Aby oznaczyć w ten sposób, że są zajęte przez statek. Następnie wywołujemy jeszcze metodę setOccupiedSquare(Ship ship, float width). Gdzie przekazujemy referencję do przed chwilą dodanego statku do listy jak i szerokość kratki którą otrzymaliśmy przy wywołaniu metody umieszczającej statek na polu. Metodę setOccupiedSquare wywołujemy w celu aktualizacji listy o otoczkę nowo umieszczonego statku. Jest to bardzo ważny moment, gdyż zapomnienie o aktualizowaniu tej listy spowodowało by, że następne statki mogły by być umieszczone w niedozwolony sposób na planszy przez Androida (nie zachowywały by przynajmniej jednego pola odstępu od siebie).

W ten sposób Android podejmuje decyzję o umieszczaniu statków na planszy. Ma to szczególnie znacznie, gdyż dobre rozstawienie statków może wydłużyć grę i sprawić, że Android otrzyma wystarczającą ilość tur i czasu aby zlokalizować wszystkie statki przeciwnika i je zniszczyć. Kluczem tutaj do sukcesu jest umieszczenie najmniejszych statków 1 oraz 2-masztowych w taki sposób aby gracz miał problem z ich znalezieniem. Pola w które gracz najmniej bądź wcale jeszcze nie strzelał idealnie się do tego nadają. Gracz przeważnie nie zastanawia się nad polami w które strzela oraz czasami może mu się wydawać, że w to pole już kiedyś strzelał i nic tam nie było. A może jednak tym razem coś będzie? Czy zaryzykuje aby oddać strzał i stracić kolejkę jeśli jego strzał spudluje? To już zależy od samego gracza.

Przejdzmy teraz do algorytmu który pomaga Androidowi podjąć decyzję w jakie pole oddać strzał. Gdyż jak wspomniałem wyżej, sam musi się spieszyć aby znaleźć i zniszczyć wszystkie statki gracza zanim gracz zniszczy wszystkie jego statki. Gdyż czasem nawet i z dobrym rozstawieniem, jeśli gracz ma odrobinę szczęścia to może zabraknąć czasu Androidowi na zniszczenie wszystkich statków przeciwnika.

W podejmowaniu decyzji strzału pomaga Androidowi mapa mapWhereToShot, która zawiera informacje pobrane z bazy danych. Natomiast metoda która odpowiada za oddawanie strzału przez Androida to androidShot(). Metoda ta nie przyjmuje żadnych parametrów i jest typu Square, gdyż zwraca obiekt pola w które Android zdecydował się oddać strzał.

Algorytm zaczyna się od tego, że Android sprawdza czy w swojej liście pól w której już strzelał nie został mu jakiś statek który nie jest jeszcze zniszczony. Jest to bardzo ważny ruch który sprowadza się do skupienia się na odkrytym statku (tak też przeważnie gracze

myślą) gdyż zniszczenie statku powoduje wybuch. Co powoduje odsłonięcie dodatkowych pól planszy tak naprawdę za darmo. Jeśli popatrzymy np. na taki statek 4-masztowy to zniszczenie takiego statku nagradza nas odsłonięciem dodatkowych 14 pól planszy. Plansza ma 100 pól więc to prawie 1/5 planszy zostaje odsłonięta (doliczmy do tego jeszcze trafione 4 pola statku). Przeliczając to na procentowy wzrost szansy na trafienie to ze 100 pól 20 pój zajmują statki. Więc na początku szansa na trafienie statku to 20%. W momencie gdy zniszczymy w pierwszych paru strzałach 4 masztowiec będziemy mieć od razu odsłonięte około 20 pól planszy. Zostaje nam 80 pól do odkrycia z 16 polami zawierającymi statek. Niemniej jednak to nie poprawia naszej procentowej szansy na trafienie, gdyż nadal mamy 20% szansy, że trafimy w statek. Więc kolejność niszczenia statków nie ma tutaj znaczenia. Ale zawsze już odsłoniliśmy jakąś część planszy i mamy pewność, że na tych 20 polach nie występuje już statek. Dlatego algorytm skupia się na strzelaniu w pola na których potencjalnie najczęściej występują statki oraz dobijanie już znalezionych statków. Tak więc Android przeszukuje swoją listę pól w które może oddawać strzały sprawdzając czy są jakieś trafienia statku, jeśli tak to sprawdza czy statek jest zniszczony, jeśli nie ma takich statków to przechodzi do algorytmu poszukiwania statku. Natomiast jeśli jakiś statek nie jest zniszczony to odpowiednio przeszukuje obszar wokół tego statku. Zaczynając od lewej strony sprawdza czy pole na lewo od trafionego miejsca statku istnieje, jeśli tak to sprawdza jego stan. Jeśli stan oznacza również trafienie to zna już orientację statku i idzie dalej w lewo. Jeśli kolejne pole jest puste to strzela (gdyż zna już orientację statku). Jeśli pole jest trafione to idzie dalej w lewo. I tak do osiągnięcia maksymalnie 3 pól odległości od znalezionej na początku miejsca trafienia. Gdyż maksymalna długość statków to 4. Gdyby jednak podczas tego przeszukiwania trafił na nieistniejące pole bądź na pole z pudłem to porzuca dalsze poszukiwanie i przechodzi do przeszukania statku w górę. Powtarza czynności opisane wyżej. Sprawdza czy pierwsze pole w górę od miejsca trafienia jest puste, bądź trafione. Jeśli jest puste to dodaje je do listy potencjalnych pól do strzału, gdyż nie zna orientacji statku. Jeśli trafione to idzie w dalej w górę i jeśli trafi na puste pole to oddaje strzał, gdyż zna już orientację statku. Natomiast jeśli znów trafi na pole trafione to idzie o kolejny krok dalej i tak do 3 pól od miejsca trafienia od którego zaczął poszukiwania. Później jeśli nie oddał strzału idąc w górę to sprawdza prawą stronę a na samym końcu idzie w dół. Jeśli również nic nie znalazł (nie zna orientacji, gdyż ma tylko jedno pole trafione a wokół są pola nieodkryte bądź nie istnieją) to losuje pseudolosową liczbę w zakresie listy potencjalnych pól do strzału i oddaje strzał. Tak też zachował by się gracz który ma tylko jedno pole z trafionym statkiem i podejmuje decyzję w którą stronę oddać strzał.

Jeśli jednak po sprawdzeniu listy nie znajdzie się żaden niezniszczony statek, Android przystępuje do poszukiwania najbardziej korzystnego pola do strzału. Na początku wywołuje metodę getHighestValuedQuarter która dzieli całą listę której używa Android do podejmowania decyzji (lista ta zawiera informacje o wartościach pól pobranych z mapy mapWhereToShot) na 4 ćwiartki. Następnie oblicza w której ćwiartce gracz najczęściej umieszcza statki. Tak która będzie miała największą sumę wartości wszystkich pól zostaje wybrana. Następnie sprawdzane jest czy suma pól jest większa niż 0. Jeśli jest to oddaje strzał w najbardziej prawdopodobne pole. Natomiast jeśli suma pól jest równa 0 to sprawdza wszystkie pola które są dostępne do strzelania i oddaje losowy strzał. Gdyż nie ma informacji na temat potencjalnych miejsc występowania statku. Ważną rzeczą jest tutaj to, że Android nie bierze pod uwagę pól które zostały już odkryte (nie dodaje wartości pól już odkrytych do sumy szansy na trafienie w danej ćwiartce).

Pod koniec gdy, w momencie gdy zostało mniej niż 6 pól i nie ma informacji na temat pól w których mogą wystąpić statki strzela po prostu po kolej w każde pole. Na zakończenie w momencie gdy aktywność zostaje zniszczona (metoda onDestroy) zostaje wywołana po tym jak gracz wybrał czy chce wrócić do głównego menu czy rozpocząć grę od nowa i przygotować nowe ustawienia statków, baza danych zostaje zaktualizowana o przebieg rozgrywki. Pola w które gracz oddał strzał zwiększą swoją wartość o 1 w bazie danych tak samo wartość pól na których Android odkrył statki podczas gry zostaje zwiększona o 1 w bazie danych. W przyszłej grze prawdopodobnie Android już bardziej skupi się na tych polach więc gracz musi zmienić położenie statków. Chyba, że nie obawia się, że jego statki zostaną szybko wykryte i zestrzelone przez Androida.

8. Wyniki przeprowadzonych badań

Badania zostały przeprowadzone w głównej mierze przeze mnie ale również sprawdzałem aby potwierdzić wiarygodność tych badań na 8 innych osobach. W sumie około 200 partii zostało rozegranych. W tym też był jeden reset bazy danych gdyż podczas badań na innych osobach wykryłem, że jest źle odmierzana liczba rund.

Wyniki kształtują się w głównej mierze w następujący sposób:

ID:	WINNER:	ROUNDS:
1	PLAYER	54
2	PLAYER	59
3	PLAYER	42
4	PLAYER	66
5	PLAYER	58
6	PLAYER	53
7	ANDROID	61
8	ANDROID	62
9	ANDROID	60
10	ANDROID	60
11	ANDROID	58
12	ANDROID	61
13	ANDROID	60

Rysunek 20: Moje kilkanaście pierwszych gier (liczba rund jest nieprawidłowa)

Jak można zaobserwować, Android nauczył się mojej taktyki nie dał mi przez dłuższy czas możliwości wygranej. Gdy zacząłem stosować różne taktyki i co bitwę zmieniać rozstawienie statków wyniki kształtowały się już nieco inaczej:

ID: 15	WINNER: PLAYER	ROUNDS: 42
ID: 16	WINNER: PLAYER	ROUNDS: 35
ID: 17	WINNER: ANDROID	ROUNDS: 35
ID: 18	WINNER: PLAYER	ROUNDS: 28
ID: 19	WINNER: ANDROID	ROUNDS: 46
ID: 20	WINNER: PLAYER	ROUNDS: 43
ID: 21	WINNER: PLAYER	ROUNDS: 20
ID: 22	WINNER: PLAYER	ROUNDS: 23
ID: 23	WINNER: PLAYER	ROUNDS: 40
ID: 24	WINNER: ANDROID	ROUNDS: 34
ID: 25	WINNER: ANDROID	ROUNDS: 25
ID: 26	WINNER: ANDROID	ROUNDS: 25
ID: 27	WINNER: ANDROID	ROUNDS: 40

Rysunek 21: Wyniki po zresetowaniu bazy danych i naprawieniu licznika rund

ID: 23	WINNER: PLAYER	ROUNDS: 40
ID: 24	WINNER: ANDROID	ROUNDS: 34
ID: 25	WINNER: ANDROID	ROUNDS: 25
ID: 26	WINNER: ANDROID	ROUNDS: 25
ID: 27	WINNER: ANDROID	ROUNDS: 40
ID: 28	WINNER: ANDROID	ROUNDS: 42
ID: 29	WINNER: ANDROID	ROUNDS: 33
ID: 30	WINNER: ANDROID	ROUNDS: 21
ID: 31	WINNER: PLAYER	ROUNDS: 34
ID: 32	WINNER: ANDROID	ROUNDS: 37
ID: 33	WINNER: ANDROID	ROUNDS: 23
ID: 34	WINNER: PLAYER	ROUNDS: 42
ID: 35	WINNER: PLAYER	ROUNDS: 47

Rysunek 22: Dalsza część wyników z prawidłowym licznikiem

Statistics		
Historia Gier:		
ID: 40	WINNER: ANDROID	ROUNDS: 49
ID: 41	WINNER: ANDROID	ROUNDS: 43
ID: 42	WINNER: ANDROID	ROUNDS: 42
ID: 43	WINNER: ANDROID	ROUNDS: 35
ID: 44	WINNER: PLAYER	ROUNDS: 30
ID: 45	WINNER: PLAYER	ROUNDS: 39
ID: 46	WINNER: PLAYER	ROUNDS: 43
ID: 47	WINNER: PLAYER	ROUNDS: 35
ID: 48	WINNER: ANDROID	ROUNDS: 33
ID: 49	WINNER: ANDROID	ROUNDS: 35
ID: 50	WINNER: ANDROID	ROUNDS: 40
ID: 51	WINNER: ANDROID	ROUNDS: 38
ID: 52	WINNER: ANDROID	ROUNDS: 25

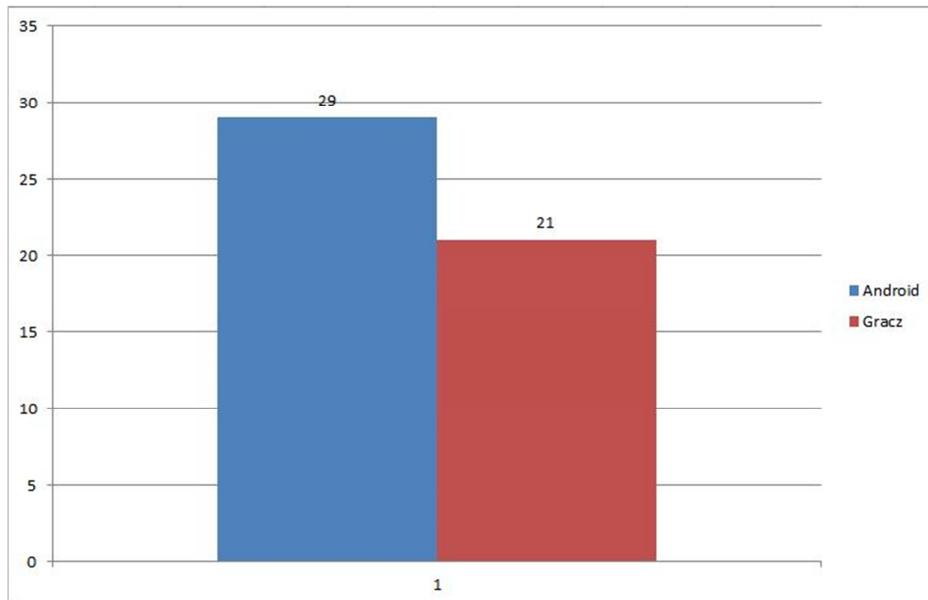
Rysunek 23: Dalsza część zwycięstw z prawidłowym licznikiem rund

Jak można zaobserwować, istnieje możliwość wygrywania z Androidem nawet i kilka razy pod rząd jeśli wie się jak algorytm działa. Niemniej jednak i tak sporo zależy od szczęścia. Gdyż jak widać, Androidowi udało się nas pokonać nawet i w 21 rund (ID: 30) oraz 25 rund (ID: 52). Nam co prawda przy odrobinie szczęścia udało się i w 20 rund pokonać Androida (ID: 21). Patrząc statystycznie:

Średnia ilość rund potrzebna na wygraną: 35,96

Android: 29 wygranych (58%)

Gracz: 21 Wygranych (42%)



Rysunek 24: Wykres wygrane w przeciągu 50 bitów

Statystyka miała się nieco inaczej gdy do gry zasiadły osoby które nie brały udziału w tworzeniu aplikacji i nie miały pojęcia jak wygląda algorytm. Każda z osób miała rozegrać przynajmniej 5 gier z Androidem. Można było zaobserwować, że gdy nowa osoba rozpoczynała grę, to przeważnie pierwszą grę wygrywała. Poczynając przeważnie już od kolejnej bitwy odnosiły porażki. Tak więc statystycznie wychodziło 1-2 (20-40%) wygrane 3-4 (80%-60%) przegrane. Co można powiedzieć, że częściowo się pokrywa z wynikami które sam osiągnąłem.

Z obserwacji również wynika, że im więcej gier się rozegra tym algorytm działa lepiej i gracz musi wymyślać nowe ustawienia statków, dostosowywać się do miejsc gdzie Android zaczyna przeważnie strzelać i zapamiętywać gdzie Android umieszcza swoje statki. W przeciwnym razie może w bardzo łatwy sposób przegrać. Przegrana jest niemal gwarantowana jeśli nie zmieni ustawienia w przeciągu 2-3 bitew lub zmiana będzie znikoma.

Z logicznego punktu widzenia można się spodziewać, że w sporadycznych sytuacjach gracz będzie prowadził rozgrywkę w taki sposób, że za każdym razem będzie stosował inne ustawienie jak również będzie w zupełnie inny sposób oddawał strzały. W takim przypadku może to doprowadzić do rozkładu normalnego informacji w bazie danych i tak naprawdę informacje zawarte w niej będą mało pomocne dla Androida. Ale algorytm przewiduje taką sytuację, dlatego też jest dodany mechanizm losowania liczb pseudolosowych po to aby Android w tak zaistniałej sytuacji nie rozmieszczał statków zawsze w ten sam sposób. Co tak naprawdę prowadzi do tego, że i gracz nie wie jak będą rozłożone statki Androida oraz Android będzie miał problem ze wskazaniem odpowiednich pól i wszystko będzie się opierać kto z prowadzących bitwę będzie miał większe szczęście. Można by było się tutaj ewentualnie doszukiwać taktyk związanych z umieszczaniem statków na krawędzi planszy aby w momencie wybuchu odkryć mniejszą część planszy i tym samym obniżyć szansę na trafienie innych statków. Ale to też może się wiązać z tym, że gracz zacznie akurat strzelanie od zrobienia otoczki wokół planszy strzelając akurat po samych krawędziach.

9. Fragmenty kodu i przedstawienie rozwiązań niektórych funkcjonalności

Utworzenie bazy danych za pomocą pomocnika do baz danych:

```
public class BattleDatabaseHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "AndroidMemory"; // nazwa naszej
    bazy danych
    private static final int DB_VERSION = 1; // Nr wersji bazy danych

    BattleDatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE PLAYER_SHIPS (" +
            "_id TEXT PRIMARY KEY, " +
            "OCCURRENCE INTEGER);");

        db.execSQL("CREATE TABLE PLAYER_SHOTS (" +
            "_id TEXT PRIMARY KEY, " +
            "SHOTS INTEGER);");

        db.execSQL("CREATE TABLE GAMES_HISTORY (" +
            "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "WINNER TEXT, " +
            "HOW_MANY_TURNS INTEGER);");

        insertFields("PLAYER_SHIPS", db);
        insertFields("PLAYER_SHOTS", db);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion) {
    }

    private static void insertFields(String tableName, SQLiteDatabase db)
    {
        ContentValues fieldsValues = new ContentValues();
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                fieldsValues.put("_id", numberToLetter(i) +
String.valueOf(j + 1) + "");

                if (tableName.equals("PLAYER_SHOTS")) {
                    fieldsValues.put("SHOTS", 0);
                } else if (tableName.equals("PLAYER_SHIPS")) {
                    fieldsValues.put("OCCURRENCE", 0);
                }

                db.insert(tableName, null, fieldsValues);
            }
        }
    }
}
```

Przekazanie intencji z PrepareShipsActivity.java do BattleActivity.java

```
Intent intent = new Intent(this, BattleActivity.class);
intent.putExtra(BattleActivity.GAME_DATA, playField.toString());
startActivity(intent);
```

PlayField->toString:

```
@Override
public String toString() {
    String returnValue = "";
    for (Ship sp : shipList) {
        returnValue += sp.toString() + ";";
    }
    return returnValue;
}
```

Ship->toString:

```
public String toString() {
//    System.out.println("Lista: " + inWhatSquare);
    String returnValue = "";
    for (Square sq : inWhatSquare) {
        if (sq.getState() == 1) {
            returnValue += sq.toString();
        }
    }
    returnValue += getStringOrientation();
//return "X: " + x + " Y: " + y + " Length: " + length + "
Orientation :" + orientation;" ;
    return returnValue;
}
```

Parser do przekazanego Stringa:

PlayField.java

```
public PlayField(String playFieldString, float width, float initialX,
float initialY) {
    // Tworzenie siatki do gry i zapamietanie kazdego pola w liscie
    squareList
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            float x = initialX + i * width; // poczatkowa wartosc + nr
kratki * szerokosc
            float y = initialY + j * width;
            Square square = new Square(x, y, width, 0, String.valueOf(i +
1), numberToLetter(j));
            squareList.add(square);
        }
    }

    String[] ships = playFieldString.split(";");
    for (int i = 0; i < ships.length; i++) {
        shipList.add(new Ship(ships[i], squareList, width));
    }
}
```

Ship.java

```
public Ship(String shipString, ArrayList<Square> squareList, float width)
{
    startingX = 0;
    startingY = 0;
    this.width = width;
    inWhatSquare = new ArrayList<>();

    String[] squares = shipString.split("!");
    // bo dlugosc tablicy zawiera informacje o orientacji co powoduje, ze
    statki mialy
    // by o jeden maszt za duzo
    this.length = squares.length-1;

    // zmienna ktora bedzie sprawdzana czy statek ma jakies funkcjonujace
    czesci podczas bitwy
    this.workingParts = this.length;

    // ustawiamy orientacje statku
    if (squares[length].equals("true")) {
        orientation = true;
    } else if (squares[length].equals("false")) {
        orientation = false;
    }

    //length--; // usuwamy informacje o orientacji, gdyz juz ja
    ustawilismy

    // dodanie obiektow pol do listy zajmowanych pol przez statek na
    podstawie ID pol
    for (int i = 0; i < this.length; i++) {
        for (Square s : squareList) {
            // Wypisanie wszystkich stworzonych pol w squareList
            // System.out.println(s);

            // sprawdzenie czy dane pole jest zajete przez statek, jest
            tak to dodajemy je do listy
            // pol zajmowanych przez dany statek
            if ((s.getyID() + s.getxID()).equals(squares[i])) {
                inWhatSquare.add(s);
            }
        }
    }
    // Sprzedzenie przekazanych wartosci do konstruktora Ship podczas
    bitwy
    //System.out.println("Ship String: " + shipString + "\n" +
    "inWhatSquare: " + inWhatSquare);

    // ustawienie wspolrzednych statkow na podstawie wspolrzednych pol
    ktore zajmuja
    x = Float.MAX_VALUE;
    y = Float.MAX_VALUE;
    for (Square s : inWhatSquare) {
        if (s.getX() < x && s.getY() < y) {
            x = s.getX();
            y = s.getY();
        }
    }
}
```

10.Podsumowanie

Podświadomie gracz zawsze będzie ustawał w podobny sposób swoje statki. Nawet jeśli czasem się zbuntuje i stwierdzi, że zagra inaczej i rozstawi statki na samych krawędziach planszy to Android i tak szybko się nauczy zmiany taktyki gracza i dostosuje się do tego aby precyzyjnie eliminować statki gracza. Co zmusza tym samym gracza do ciągłych przemyśleń i szukania różnych rozwiązań w sposobach ustawienia statków. Co może również i powodować ciekawość i podtrzymywać zapał do gry. Nieraz też miewałem sytuację, gdzie brakowało zarówno jak i Androidowi zestrzelić jeden jednomasztowiec aby odnieść zwycięstwo. W takich sytuacjach robiło się znacznie ciekawie. Mógłbym nawet śmiało stwierdzić, że w takich sytuacjach emocje wzrastały, gdyż do wygranej brakuje tak niewiele ale niestety Android znów wygrał. Działalność algorytmu oceniam na dobry. Ciężko było by osiągnąć algorytm który nie znając rozmieszczenia statków gracza wygrywał by za każdym razem. Do mojego algorytmu można by było jeszcze wprowadzić kilka bardziej zaawansowanych usprawnień jak np. zapamiętywanie nie tylko po polach występowanie statków ale również i rodzaje statków jakie w tych polach występowały. Dodatkowo aplikację można rozbudować wprowadzając do niej reklamy które pozwoliłyby na zarobek w formie Free2Play. Aplikacja jest darmowa ale twórca dostaje wynagrodzenie za oglądane reklamy przez użytkowników.

11.Bibliografia

- [1] Wikipedia: <https://pl.wikipedia.org/wiki/Okr%C4%99ty> - Z dnia 2016.06.06
- [2] Analiza popularności systemów mobilnych z pracy Inżynierskiej Robert Matejczuk
- [3] <http://gs.statcounter.com/#mobile+tablet-os-PL-monthly-201602-201602-bar> – Z dnia 2016.06.06
- [4] [https://pl.wikipedia.org/wiki/Android_\(system_operacyjny\)](https://pl.wikipedia.org/wiki/Android_(system_operacyjny)) – Z dnia 2016.06.06
- [5] <https://pl.wikipedia.org/wiki/IOS> - Z dnia 2016.06.06
- [6] Russz głową! Android Programowanie aplikacji - Dawn Griffiths & David Griffiths - ISBN: 978-83-283-2063-5, Wydawnictwo: 2016 Helion SA.
- [7] <https://developer.android.com/training/gestures/scale.html> - Z dnia 2016.06.06
- [8] <https://developer.android.com/guide/topics/ui/drag-drop.html#AboutDragging> – Z dnia 2016.06.06
- [9] <http://beginnersbook.com/2013/12/how-to-sort-hashmap-in-java-by-keys-and-values/> - Z dnia 2016.06.06
- [10] <http://stackoverflow.com/questions/3422388/android-get-value-from-hashmap> - Z dnia 2016.06.06
- [11] <http://beginnersbook.com/2014/07/java-finding-minimum-and-maximum-values-in-an-array/> - Z dnia 2016.06.06
- [12] Thinking in Java – Bruce Eckel – Wydanie IV Edycja Polska, ISBN: 978-83-246-3176-6, Wydawnictwo: Helion
- [13] Algorytmy, struktury danych i techniki programowania – Piotr Wróblewski – Wydanie IV, ISBN: 978-83-246-2306-8, Wydawnictwo: Helion
- [14] Java Efektywne programowanie – Joshua Bloch – Wydanie II, ISBN: 978-83-246-2084-5, Wydawnictwo: Helion
- [15] <https://developer.android.com/preview/index.html> - Z dnia 2016.06.06
- [16] <http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-klas,1,11.html> – Z dnia 2016.06.06