Міністерство освіти і науки України

Департамент освіти і науки Дніпропетровської облдержадміністрації

Дніпропетровське територіальне відділення МАН України

Відділення: комп’ютерні науки

Секція: комп’ютерні системи та мережі

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВЗАЄМОДІЇ У РЕАЛЬНОМУ ЧАСІ НА ПРИКЛАДІ МОБІЛЬНОГО ДОДАТКУ

Роботу виконав:   
Фоменко Олег Володимирович,

учень 11 класу КЗО «Дніпропетровський

обласний ліцей-інтернат фізико-математичного профілю»

Наукові керівники:

Безрукавий Юрій Васильович,

вчитель інформатики КЗО *«*Дніпропетровський обласний ліцей-інтернат фізико-математичного профілю»

Ільченко Ірина Григорівна,

вчитель інформатики КЗО *«*Дніпропетровський обласний ліцей-інтернат фізико-математичного профілю»

Дніпро-2019

ЗМІСТ

[ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 3](#_Toc3152516)

[ВСТУП 4](#_Toc3152517)

[РОЗДІЛ 1](#_Toc3152518) [АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ 6](#_Toc3152519)

[1.1. Дослідження існуючих архітектур 6](#_Toc3152520)

[1.2. Постановка задачі 7](#_Toc3152521)

[РОЗДІЛ 2](#_Toc3152522) [РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 9](#_Toc3152523)

[2.1. Обґрунтування вибору мови програмування та засобів розробки 9](#_Toc3152524)

[2.2. Обґрунтування вибору протоколу передачі даних 10](#_Toc3152525)

[2.3. Опис та термінологія гри 10](#_Toc3152526)

[2.4. Опис та розробка архітектури серверної та клієнтської частини 12](#_Toc3152527)

[2.4.1. Графічна частина 12](#_Toc3152528)

[2.4.2. Ігрові об’єкти 12](#_Toc3152529)

[2.4.3. Клієнтська частина 13](#_Toc3152530)

[2.4.5. Типи та структура JSON запитів 15](#_Toc3152531)

[2.4.6. Проблема черговості запитів 16](#_Toc3152532)

[2.4.7. Проблема загублення пакетів або втрати зв’язку 16](#_Toc3152533)

[2.4.8. Проблема затримки пакетів 17](#_Toc3152534)

[РОЗДІЛ 3](#_Toc3152535) [АПРОБАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 18](#_Toc3152536)

[3.1. Результат апробації роботи і перспективи розвитку 18](#_Toc3152537)

[ВИСНОВКИ 20](#_Toc3152538)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 22](#_Toc3152539)

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Real-time система – система, яка в реальному часі обмінюється інформацією з усіма її вузлами.

IDE – інтегроване середовище розробки, комплексне рішення для написання програмного забезпечення.

Socket – програмний інтерфейс для обміну даними між процесами. Процеси можуть знаходитися як на одному пристрої, так і на двох різних, пов’язаних мережею.

Мережевий протокол або протокол передачі даних – набір правил, що визначає взаємодію в мережі різноманітних програм, мережевих вузлів чи систем і створює таким чином єдиний простір передачі даних.

Android – операційна система і платформа для мобільних телефонів, створена корпорацією Google на базі ядра Linux.

JSON – JavaScript Object Notation, текстовий формат обміну даними між комп’ютерами. Дозволяє описувати об’єкти та інші структури даних. Головним чином використовується для передачі структурованої інформації через мережу Інтернет.

API – прикладний програмний інтерфейс, бібліотека або набір засобів для створення програмного забезпечення. Спрощено - це набір методів для розробки програмних компонентів.

Java – об'єктно-орієнтована мова програмування, випущена як основний компонент платформи Java. Синтаксис мови запозичений від C та C++. Java-програми запускаються у середовищі віртуальної машини Java.

# ВСТУП

Поява Інтернету в світі почала нову еру інформаційних технологій в житті людини. Сучасні програми неможливо уявити без обміну інформацією в цій мережі. Віддалене зберігання даних є невід’ємною частиною будь-якого додатку, тому для їх зручного, швидкого та надійного використання було розроблено велику кількість різних систем та інформаційних архітектур [1]. Але збої в роботі Інтернету виникають досить часто, так як ця мережа постійно відключає та підключає все нові вузли. Особливо критично це впливає на системи, що повинні в реальному часі опрацьовувати запити та забезпечувати своїх клієнтів інформацією. Тому дуже важливо надсилати необхідну кількість даних користувачам своєчасно та незалежно від їх кількості, швидкості підключення до мережі, тощо.

Однією з найбільш практичних архітектур для вирішення таких проблем є клієнт-серверна архітектура [2]. В ній один або декілька фіксованих вузлів виступають у ролі посередника в обміні інформацією між іншими клієнтами. У такій системі сервер повинен забезпечити надійну передачу даних, їх коректність та своєчасне отримання для всіх користувачів.

Серед відкритих літературних та інтернет-джерел можна знайти лише деякі рекомендації щодо написання таких систем, або дуже стисле пояснення їх принципів роботи. Саме відсутність повного опису основних аспектів проектування, розробки та тестування клієнт-серверної мережі для роботи в реальному часі й зумовлює актуальність цього дослідження.

Мета дослідження - на прикладі створення мобільної онлайн гри описати і розробити клієнтське та серверне програмне забезпечення для обміну інформацією, яке було б стійким до нестабільної роботи мережі Інтернет та своєчасно б надавало гравцям необхідні для роботи дані.

Для досягнення мети слід розв’язати такі задачі:

* проаналізувати особливості клієнт-серверної архітектури та мережевого програмування;
* спираючись на аналіз, сформулювати вимоги до власного програмного забезпечення;
* обрати інструменти розробки;
* спроектувати алгоритми та протоколи обміну даними;
* створити програмну реалізацію;
* здійснити тестування та відлагодження.

Об’єктом дослідження є система з клієнт-серверною архітектурою для обміну інформацією в реальному часі, а предметом – онлайн гра (її клієнтська та серверна частини), яка повинна коректно працювати незалежно від якості роботи мережі Інтернет.

Наукова новизна роботи полягає в тому, що було запропоноване комплексне рішення (архітектура та програмна реалізація), яке в умовах нестабільної роботи мережі Інтернет дозволяє забезпечувати надійний зв’язок мобільного додатку із сервером.

Результати роботи можуть бути використані для створення систем реального часу у сферах, де необхідно забезпечення безперервного обміну даними між сервером та клієнтом в умовах наявності збоїв в мережі.

# РОЗДІЛ 1

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

## Дослідження існуючих архітектур

Для взаємодії між собою, сучасні комп’ютери використовують комп’ютерні (обчислювальні) мережі. Основним призначенням таких мереж є:

1. Надання доступу до інформації.
2. Сумісне використання технічних ресурсів.
3. Віддалене керування.

Основними можливостями є:

1. Можливість швидкої передачі інформації на великі відстані.
2. Оперативний пошук інформації.
3. Обмін текстовою, звуковою та відеоінформацією у реальному часі.
4. Зберігання інформації.

Мережі класифікують за типом використаних протоколів, топологією, областю дії, швидкістю передачі даних, призначенням, тощо. За архітектурою вони бувають однорангові та клієнт-серверні.

Однорангова система – це інформаційна архітектура, в якій клієнти обмінюються даними напряму один з одним, на відміну від клієнт-серверної архітектури, де сервер є посередником.

Якщо проаналізувати різні системи обміну даними, можна виділити наступні, важливі з погляду збалансованої роботи мережі, характеристики:

1. Надійність – система повинна вчасно та без затримки надавати всім клієнтам актуальну інформацію в необхідній кількості.
2. Оптимізованість – система повинна бути оптимізованою з погляду споживання обчислювальних ресурсів.

Однорангова система є складнішою в реалізації та більш вимогливою до апаратних характеристик користувачів, оскільки зберігання й опрацювання всіх даних повинно відбуватися на їх пристроях. Також, швидкість та коректність її роботи напряму залежить від кількості під’єднаних до неї клієнтів. Тому для розробки системи з зазначеними вище характеристиками обрано клієнт-серверну архітектуру. Вона передбачає наступні компоненти:

1. Набір серверів і їх програмного забезпечення, які оброблюють та надають інформацію.
2. Набір клієнтів та їх програмного забезпечення, які використовують надану сервером інформацію.
3. Мережа в якій передається інформація.

В більшості сучасних програм дуже важливим є обмін інформацією в реальному часі – так звані «real-time» системи. Як приклад можна розглядати онлайн гру у танки. Нажаль, у відкритих джерелах не існує більш-менш повної інформації про проектування архітектури таких систем. Є багато статей на окремі теми[6-10], але дуже важко знайти хоча б одну, що їх об’єднує, тому кожному розробнику програмного забезпечення потрібно частинками збирати цю інформацію та проектувати всю систему самостійно.

## Постановка задачі

Було поставлено наступні задачі:

1. На прикладі онлайн гри розробити та описати архітектуру програми для взаємодії клієнта з сервером в реальному часі, яка б відповідала таким вимогам:

* бути стійкою до нестабільного підключення, втрати частини переданих даних та інших збоїв мережі, не пов’язаних з нашим програмним забезпеченням;
* бути стійкою до підключення одночасно великої кількості користувачів;
* оптимізовано працювати, споживаючи якнайменше обчислювальних ресурсів.

1. Створити опис архітектури клієнтської та серверної частин:

* описати їх взаємодію;
* описати принципи їх роботи;
* розробити методи уникнення збоїв в роботі через нестабільність мережі.

1. Протестувати систему з реальними гравцями та ботами. Для тестування ботами написати бот-програму, яка імітує задану кількість гравців.

# РОЗДІЛ 2

# РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Обґрунтування вибору мови програмування та засобів розробки

В якості мови програмування при розробці в наш час найчастіше використовують Java [3]. Це обумовлено наступними факторами:

1. Java – кросплатформенна мова програмування. Додатки, написані цією мовою, легко переносяться на пристрої, які керуються іншими операційними системами.
2. Java – сучасна та швидка мова програмування.
3. У стандартній бібліотеці Java є багато вбудованих алгоритмів які полегшують написання коду та спрощують його.
4. Java має автоматичне керування споживанням пам’яті, яке полегшує написання коду та економить оперативну пам’ять, що робить програму менш вимогливою до ресурсів пристрою.

В наш час велику популярність здобули мобільні пристрої під керуванням операційної системи Android. Програми для них дуже зручно писати за допомогою середовища Android Studio [4], використовуючи фреймворк LibGDX [5] для полегшення роботи.

Android Studio– це інтегроване середовище (IDE) для роботи з платформою Android, анонсоване 16 травня 2013 року на конференції Google I/O менеджером з продукції корпорації Google - Еллі Паверс (Ellie Powers).

8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0. Android Studio, заснована на програмному забезпеченні IntelliJ IDEA від компанії JetBrains, є офіційним засобом розробки Android додатків. Дане середовище доступне для Windows, MacOS і Linux. Мова програмування, яка використовується у цьому середовищі – Java.

LibGDX– це Java-фреймворк, який надає кросплатформний API для розробки ігор та додатків, що працюють в режимі реального часу. Він дозволяє використовувати один і той самий програмний код як для настільних комп’ютерів, так і для мобільних систем. Бібліотека є кросплатформенною - підтримує Windows, Linux, Mac OS X, Android та браузери з підтримкою WebGL.

На основі вищезазначеного, було вирішено в якості мови програмування для клієнтської та серверної частини обрати Java, в якості середовища розробки для клієнту – Android Studio.

## Обґрунтування вибору протоколу передачі даних

Перед розробкою програмного забезпечення постало ключове питання: яким протоколом передачі даних користуватися? Оскільки Java стандартно надає розробникам можливість використання socket, то для передачі інформації було розглянуто два варіанти: TCP та UDP [6].

Основною особливістю TCP-socket є автоматичне розбиття інформації на пакети, контроль потоку даних, та, що найголовніше, гарантування отримання та черговості інформації. Але ціною всіх цих плюсів є нестійка робота протоколу при нестабільному підключенні, що у підсумку дає величезну затримку у відправці та отриманні інформації.

UDP-socket не гарантує нам отримання інформації, не контролює її черговість та потрібно самостійно формувати пакети з даними, але всі ці проблеми можуть бути вирішені програмістом, тому не є критичними для real-time програмного забезпечення [7].

З огляду на все це, було вибрано та використано UDP-socket, який в мові програмування Java реалізовано класом *DatagramSocket*, а його пакети – класом *DatagramPacket*.

## Опис та термінологія гри

В грі бере участь певна кількість гравців, рухаючи та винищуючи танки розміром 64\*64 пікселі на полі 1000\*1000 пікселів. Танк рухається з постійною швидкістю, ніколи не зупиняючись. Поле обмежено стіною, від якої відбиваються снаряди й танки (рис. 2.1, рис. 2.2). Кожні 5 секунд танк випускає снаряд у тому напрямі, в якому він рухається. Оскільки снаряди й танки відбиваються від стін, всі елементи гри завжди знаходяться у русі. Танк поточного гравця знаходиться в центрі екрану і, при пересуванні, екран переміщується разом з ним. Зміна напрямку руху танку відбувається «свайпами» по екрану у сторону бажаного напрямку. Коли снаряд потрапляє в танк, він його знищує, але не знищується сам.

Термінологія гри:

* Танк – ігровий об’єкт користувача. Його гравець пересуває та саме з нього вилітають снаряди.
* Снаряд – ігровий об’єкт, що генерується кожні 5 секунд та випускається танком в напрямі свого руху, має швидкість більшу, ніж у танка, при зіткненні з танком знищує його, але не знищується сам.
* Стіна– ігровий об’єкт, що відбиває від себе танки і снаряди.

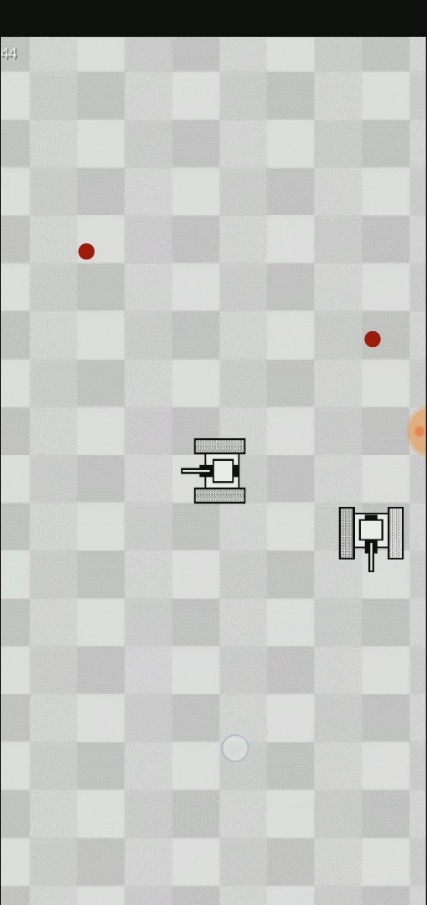
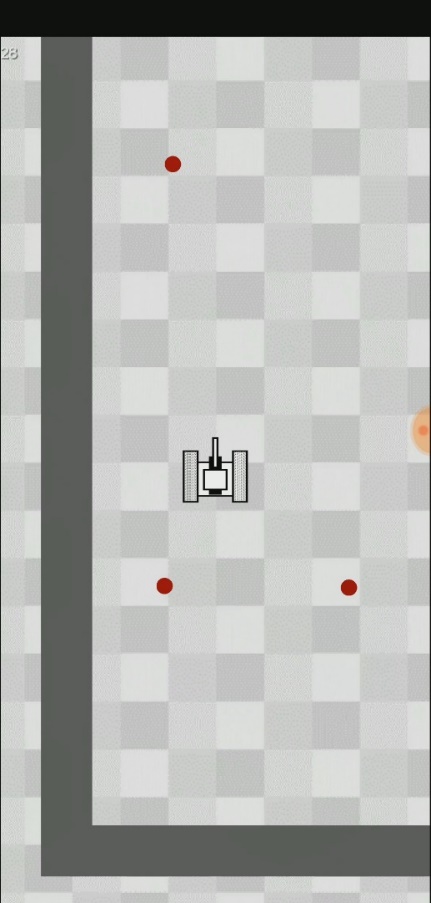
 

Рис 2.2. Скриншорт гри (2)

Рис 2.1. Скриншорт гри (1)

## Опис та розробка архітектури серверної та клієнтської частини

## 2.4.1. Графічна частина

Основним класом клієнтської частини є *Main.java* з методом *update()*. Система викликає цей метод кожен такт роботи програми і він повинен намалювати картинку (кожен виклик цього методу відповідає за наступний кадр).

Кожен екран гри (стартовий, основний (власне гра), фінальний) розширює клас *State.java* з методами *update()* та *render().* Перший відповідає за оновлення параметрів об’єктів (їх координат), що будуть намальовані методом *render().* За роботу різних екранів гри відповідає клас *GameStateManager*.java. В ньому зберігається структура даних стек з поточними екранами. Об’єкт класу *State.java,* який знаходиться зверху, є активним у цей момент. *GameStateManager.java* реалізує методи *update()* та *render(),* які викликають відповідні методи в об’єкті класу *State.java,* що знаходиться зверху стека.

Метод *update()* головного класу *Main.java* викликає спочатку *update(),* а потім *render()* в об’єкті класу *GameStateManager.java* (рис. 2.3).

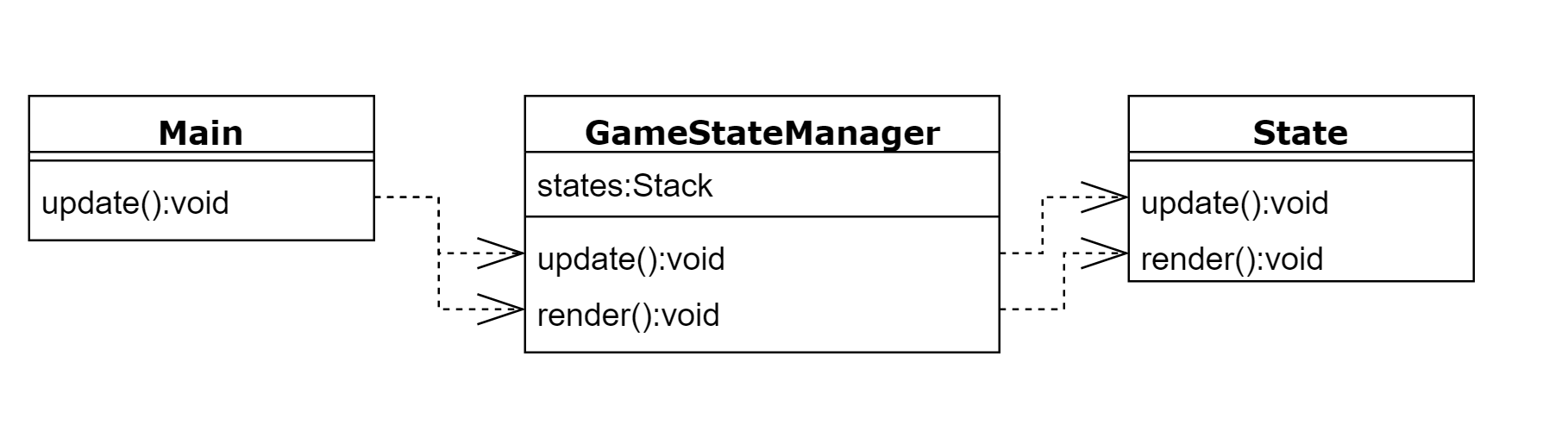


Рис. 2.3. Архітектура графічної підсистеми

## 2.4.2. Ігрові об’єкти

Ігровими об’єктами в нашій грі є танки, снаряди та стіни, що оточують поле. Кожен гравець, а точніше танк, яким він керує, ідентифікується за унікальним номером, який йому видає сервер. Для того, щоб розрізняти снаряди, кожен з них теж має свій унікальний порядковий номер. Всі ігрові об’єкти характеризуються їх розмірами, швидкістю, напрямком та поточними координатами.

За ігрові об’єкти відповідає клас *GameObject.java*, який є батьківським класом для *Tank.java* та *Ball.java* (власне танк та снаряди). Він містить інформацію про об’єкти та метод *update(),* який оновлює координати в залежності від часу, знаючи інформацію про швидкість та напрямок. За інформацію про характеристики стіни відповідає клас *Wall.java*. Всі ці класи з ігровими об’єктами реалізовані одночасно як на серверній частині, так і на клієнтській [8].

## 2.4.3. Клієнтська частина

За відправку та отримання інформації відповідає клас *Handler.java*. Він містить два внутрішніх класи *Get.java* та *Send.java,* які імплементують інтерфейс *Runnable.java*, а також метод *send().*

При створенні об’єкту класу *Handler.java* створюється окремий потік на основі класу *Get.java,* який неперервно приймає інформацію, що надсилається клієнту. Приймаючи запити, він одразу оновлює інформацію на ігровому об’єкті якому вона належить. Для більш зручного використання, всі ігрові об’єкти зберігаються в структурі даних HashMap (окремо танки, окремо снаряди), де ключем є ідентифікатор ігрового об’єкту, а значенням – власне сам об’єкт.

Для відправлення інформації викликається метод *send(*), який створює потік з *Send.java*. Він, в свою чергу, відправляє інформацію та «вмирає».

Створення окремих потоків для взаємодії з Інтернетом необхідно через те, що відправка та отримання інформації є дуже повільною та ресурсоємною операцією, тому її виконання в головному потоці програми може затримати всю роботу, зокрема відображення графічної частини додатку.

Головним класом саме гри є *Game.java,* який розширює клас *State.java*. Полями в ньому є масив з інформацією про стіни і структури даних *HashMap* для танків та снарядів (окремо для кожного), в яких ключем є їх ідентифікаційний код, а значенням відповідний об’єкт. Одним з найважливіших в цьому класі є метод *update()*. В ньому ми перевіряємо, чи змінює користувач напрямок руху (якщо так, то відправляємо нову інформацію на сервер), викликаємо методи *update()* в об’єктах *Tank.java* та *Ball.java*, відправляємо серверу запити на отримання нових даних про стани ігрових об’єктів. (Рис 2.4.)

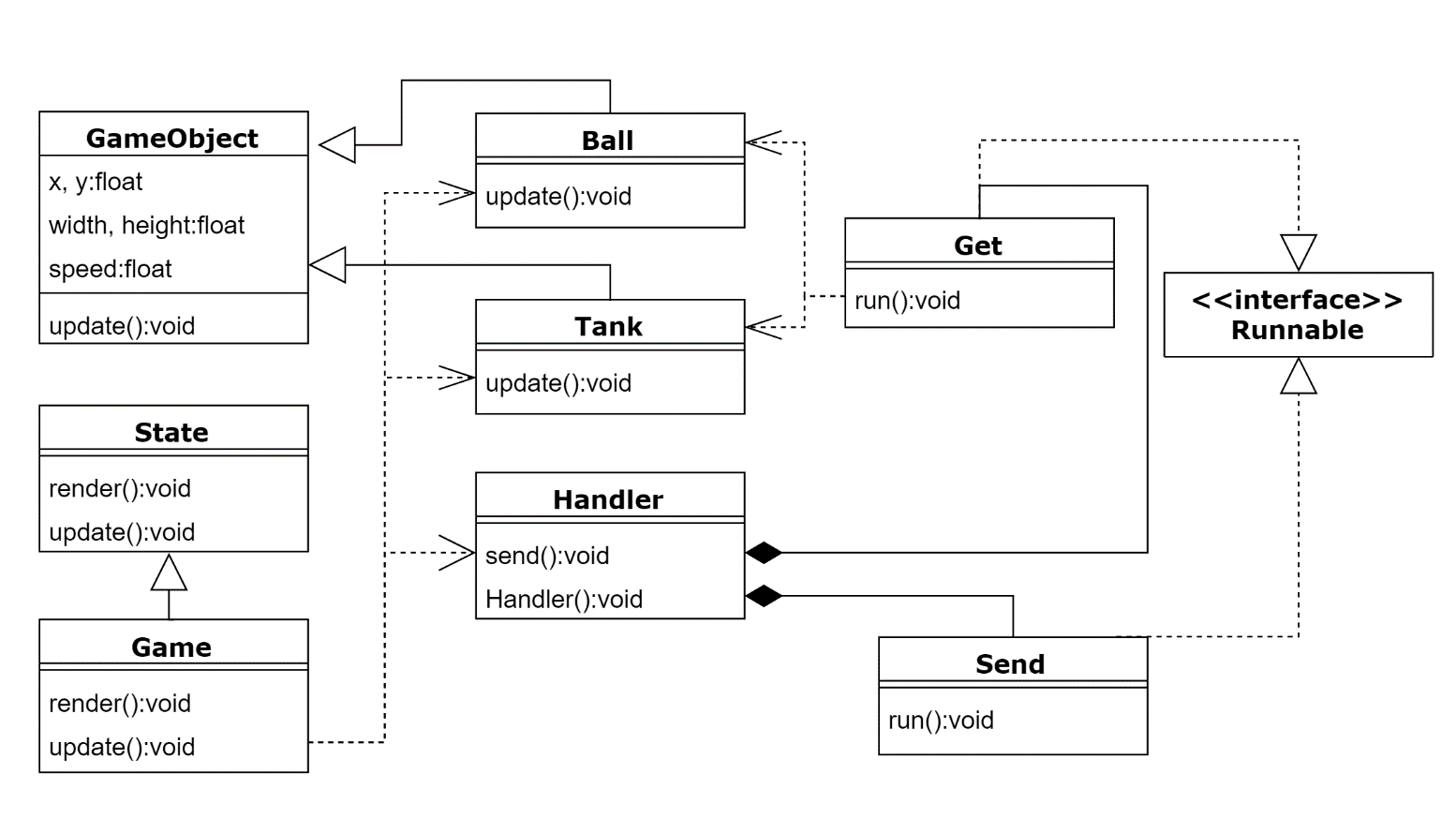


Рис. 2.4. Клієнтська частина

2.4.4. Серверна частина

Клас *Updater.java* імплементує інтерфейс *Runnable* та в своєму методі *run()* неперервно оновлює інформацію про всі ігрові об’єкти (викликає їх update() методи).

Головний клас сервера *Main.java* містить в собі структуру даних *HashMap,* в якій ключем є ідентифікаційний код гравця, а значенням – об’єкт відповідного танку. Також тут містяться поля з масивом об’єктів снарядів (клас *Ball.java*) та масивом об’єктів стін (*Wall.java*). Основним методом є *main()* в якому створюються об’єкти всіх полів, об’єкт *DatagramSocket* та паралельний потік класу *Update.java*. Потім в неперервному циклі *while(true)* приймаються запити від користувачів. Отримавши запит, створюється потік на основі внутрішнього класу *Handler.java* в який передається одержана інформація. В ньому запит оброблюється, після чого потік «вмирає». Це необхідно для того, щоб обробка запитів не затримувала отримання нових пакетів від інших клієнтів [9] (рис. 2.5).

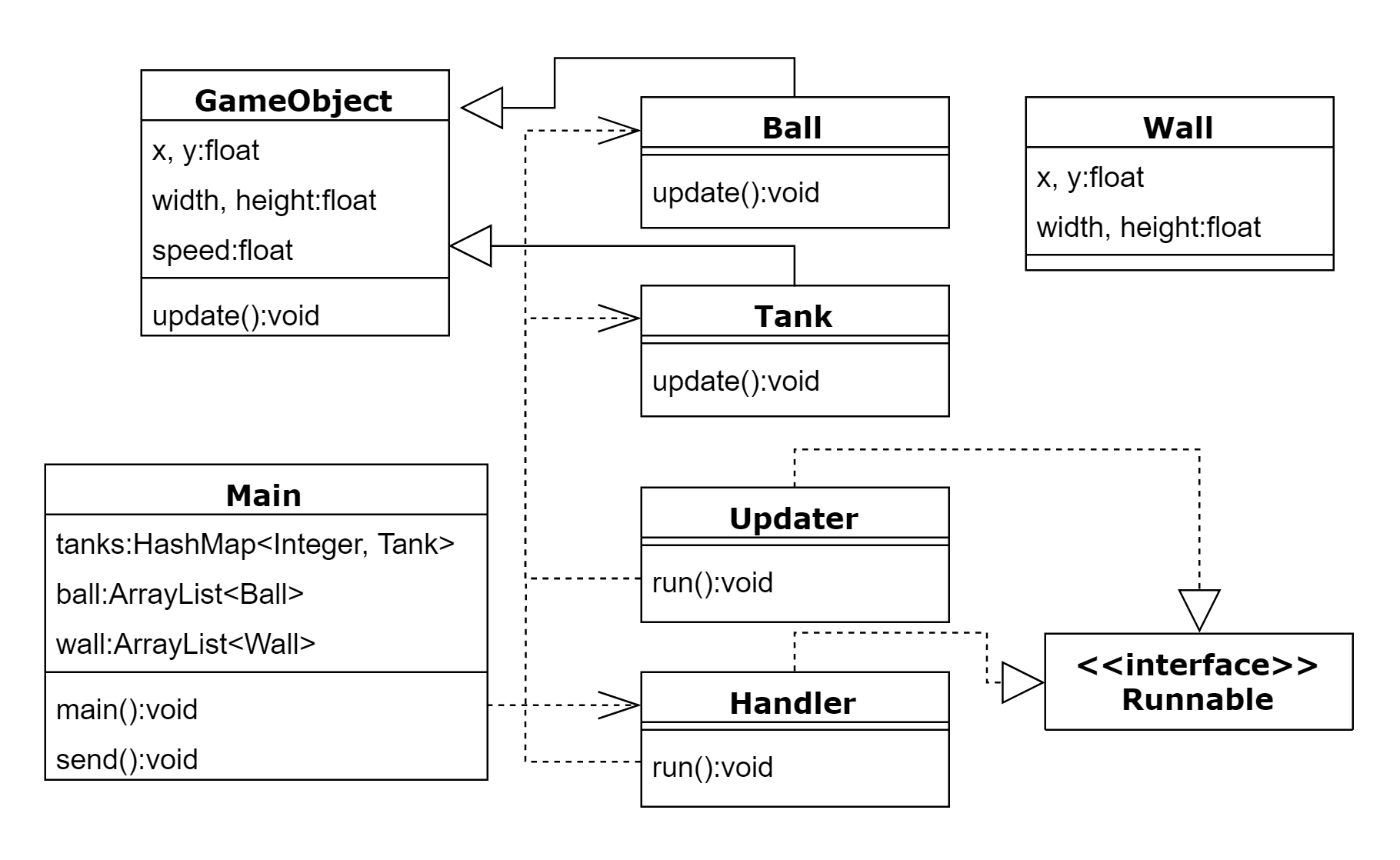


Рис. 2.5. Серверна частина

## 2.4.5. Типи та структура JSON запитів

При підключенні до гри, клієнт відправляє CREATE-запит для інформування сервера про під’єднання нового гравця. У відповідь сервер повертає CREARED-запит в якому містяться id гравця, інформація про всіх гравців, снаряди, стіни (JSONArray).

Якщо гравець змінює напрямок руху танка, клієнт відправляє UPDATE-запит з id гравця та інформацією про новий напрямок.

Після опрацювання програмного коду метода *update()* класу *Game.java* клієнтської частини, відбувається відправка на сервер GET-запиту з id гравця. У відповідь сервер надсилає інформацію з JSONArray танків та снарядів (UPDATE – запити) (рис. 2.6).

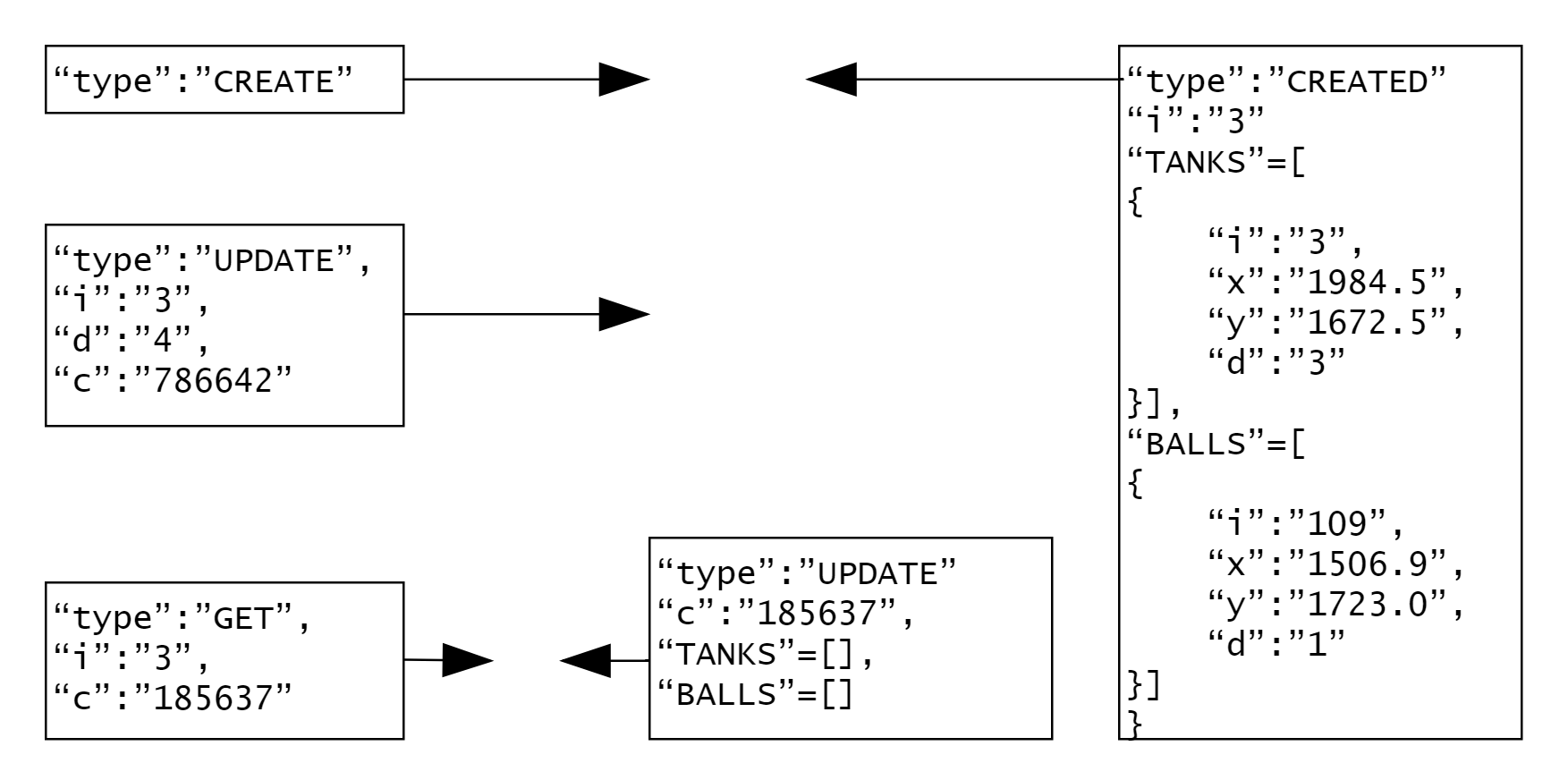


Рис. 2.6. Типи запитів

## 2.4.6. Проблема черговості запитів

Для гарантування черговості запитів на стороні клієнта, з кожним із них надсилається порядковий номер пакету. Коли сервер опрацьовує інформацію він зберігає цей номер та відправляє відповідь разом з ним. Клієнт в свою чергу зберігає останній оброблений (отриманий від сервера та опрацьований) номер пакету. Якщо сервер вже оброблював пакет з номером N, а зараз він отримав пакет з номером N–1, то цей пакет ігнорується. А якщо ми отримаємо пакет з номером N+1, то ми його обробимо та запам’ятаємо, що це був останній оброблений пакет. Також програма зберігає номер останнього відправленого UPDATE-запиту від клієнта. Клієнт буде ігнорувати всі пакети, які прийдуть з номером, меншим за номер останнього отриманого UPDATE-запиту. Це потрібно для того, щоб ми не обробили інформацію, яка була актуальною до заміни користувачем даних про напрямок танку.

## 2.4.7. Проблема загублення пакетів або втрати зв’язку

Для того, щоб бути хоч частково незалежним від якості зв’язку із сервером, на стороні клієнта виконується те ж саме оновлення координат об’єктів гри як і на сервері, а отримані дані з сервера лише корегують ці. Таким чином, все, що обчислюється на сервері, паралельно обчислюється і на клієнті, але дані на сервері є пріоритетними, і якщо виникає різниця між даними клієнта та сервера, пріоритет віддається даним з сервера.

Також, для гарантування надходження UPDATE-пакету до сервера, він дублюється. Оскільки такі пакети не відправляються дуже часто, вони, не перевантажуючи мережу, збільшують ймовірність того, що хоча б один з пакетів гарантовано буде отримано.

## 2.4.8. Проблема затримки пакетів

Оскільки заздалегідь неможливо знати точно, скільки часу буде йти інформація від сервера до клієнта, це може викликати непередбачувану затримку в отриманні даних, що зробить їх неактуальними. Якщо цю проблему не вирішувати, при кожній затримці пакету від сервера, буде виникати тремтіння зображення [10]. Щоб цього уникнути, введемо константу EPS=3.0 пікселі. Ця константа є дуже маленькою порівняно з розмірами поля (1000\*1000 пікселів) та розмірами танку (64\*64 пікселі). Якщо отриманні дані з сервера відрізняються від тих, що є у клієнта менше ніж на EPS, вони будуть ігноруватися.

# РОЗДІЛ 3

# АПРОБАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Результат апробації роботи і перспективи розвитку

Результатом роботи є пакет серверного програмного забезпечення та клієнтська програма для виконання під OS Android (починаючи від версії 6.0).

Для тестування роботи програмного забезпечення було створено бот-програму, яка імітує задану кількість «гравців» та їх взаємодію з сервером (запити на отримання даних, запити про зміну напрямку руху). Сервер було розгорнуто на ПК з наступними характеристиками: ОЗУ 8 ГБ DDR3, CPU AMD-A8 3.2 ГГц 4 ядра, OS Windows x64. Оскільки робота клієнтської частини у певній мірі залежить від параметрів обладнання гравця, тестування проводилось на пристроях з наступними характеристиками:

1. Xiaomi A2 Lite (Snapdragon 625 2.0 ГГц 8 ядер, ОЗУ 3 ГБ, Android 9.0)
2. Meizu M5 (MT6750 1.0 + 1.5 ГГц 8 ядер, ОЗУ 3 ГБ, Android 6.0)

Було проведено заміри часу отримання відповіді клієнтом від сервера (що включає час доставки пакету в обидві сторони та його обробку), залежно від кількості ботів, та отримано наступні результати:

Рис 3.1. Середній час відгуку сервера на запит (Mi A2 Lite)

Рис 3.2. Середній час відгуку сервера на запит (Meizu M5)

При тестуванні клієнт був підключений до 3G мережі зі швидкістю 2-5 Мбіт/с. Швидкість з’єднання сервера становила 40-60 Мбіт/с.

З діаграм видно, що середній час відгуку сервера на запит становить 20-30 мс, що є дуже добрим показником для мережі Інтернет, тому тестування можна вважати успішним. Надалі слід сконцентрувати зусилля на поліпшенні цього показника або ж його збереженні на цьому рівні при більшому навантаженні.

Також, даний продукт успішно пройшов тестування ліцеїстами та вчителями КЗО «Дніпропетровський обласний ліцей-інтернат фізико-математичного профілю». За результатами тестування він був покращений, були враховані всі знайдені недоліки у роботі архітектури та системи в цілому.

Ідеї щодо подальшого розвитку додатка:

1. Додати можливість проводити на одному сервері декілька ігрових сесій одночасно.
2. Додати можливість синхронізації окремо працюючих серверів для збільшення кількості одночасно граючих учасників.
3. Продовжити оптимізацію потоків на сервері та клієнті для гарантування більш стабільної роботи при більших навантаженнях.
4. Розробити систему для захисту від DoS-атак направлених на «забруднення» каналу, через що можливе погіршення зв’язку між клієнтом та сервером.

# ВИСНОВКИ

1. Проведено аналіз наявних схем та принципів проектування клієнт-серверних архітектур, їх особливостей, призначення, тощо. На його основі було виділено наступні проблеми:

- втрата пакетів;

- втрата з’єднання;

- черговість та синхронізація пакетів;

- затримка пакетів.

1. Проблема втрати переданої інформації вирішена дублюванням особливо важливих пакетів, що збільшило ймовірність їх отримання адресатом.
2. Показана можливість за допомогою організації паралельних обчислень на клієнті та сервері вирішити питання короткочасної втрати з’єднання.
3. Проблема черговості отримання пакетів вирішення введенням їх нумерації, що дало можливість відсікати застарілі дані.
4. Проблема затримки пакетів вирішена введенням константи, за допомогою якої приймається рішення про необхідність використання отриманих від сервера даних.
5. Описано підхід для проектування клієнт-серверної системи та його практичну реалізацію на прикладі онлайн гри в танки.
6. Спираючись на аналіз наявних мов програмування, для реалізації проекту обрано Java, середовище розробки Android Studio, та фреймворк libGDX.
7. Проведено тестування розробленої системи з реальними гравцями та ботами, і обчислено час відгуку сервера на запит, в залежності від кількості гравців. Середній час відповіді становить 20-30 мс, що є дуже добрим показником для мережі Інтернет.
8. Тестування системи підтвердило - система життєздатна та надійна, що в свою чергу доводить правильність обраного підходу до реалізації та вирішення описаних проблем. Розроблена система, за своїми показниками, відповідає виділеним на основі проведеного аналізу характеристикам систем реального часу.
9. Результати роботи можуть бути використані в різних сферах діяльності суспільства, де необхідний обмін даними в реальному часі, а саме: у військовій сфері (керування роботою безпілотних літальних апаратів), для реалізації соціальних суспільних проектів, які потребують гарантованого одночасного підключення великої кількості користувачів (організація онлайн опитувань, виборів), тощо.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вікіпедія. [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Комп%27ютерна\_мережа
2. Вікіпедія. [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Клієнт-серверна\_архітектура
3. Рейтинг мов програмування [Електронний ресурс]. – Режим доступу: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018
4. Офіційна документація Android. [Електронний ресурс]. – Режим доступу: <https://developer.android.com>
5. Офіційна документація libGDX [Електронний ресурс]. – Режим доступу: <http://www.libgdx.ru/p/guide.html>
6. Стаття на задану тему. [Електронний ресурс]. – Режим доступу: <https://habr.com/post/209144/>
7. Стаття на задану тему. [Електронний ресурс]. – Режим доступу: <https://habr.com/post/213559/>
8. Стаття на задану тему. [Електронний ресурс]. – Режим доступу: <https://habr.com/post/255561/>
9. Стаття на задану тему. [Електронний ресурс]. – Режим доступу: <https://habr.com/company/mailru/blog/220359/>
10. Стаття на задану тему. [Електронний ресурс]. – Режим доступу: <https://0fps.net/2014/02/17/replication-in-networked-games-latency-part-2/>