УДК 514.18

СТВОРЕННЯ ОНЛАЙН ГРИ МУЛЬТИПЛЕЄРА НА БАЗІ ДОМЕНУ

Федченко Г.В., к.т.н.,
anna-fedchenko@ukr.net, ORCID: 0000-0003-0690-6017
Матюшенко М.В., к.т.н.,
matushenkonikolay@ukr.net, ORCID: 0000-0003-4727-8993
Голотенко К.С.
Національний технічний університет «Харківський політехнічний інститут» (м. Харків, Україна)

В роботі досліджуються актуальна тема, що пов'язана з багатокористувацькими іграми, які завжди об'єднували велику кількість людей за проведенням дозвілля. Коли гра стане доступною публічно, вона якісно урізноманітнить жанр браузерних багатокористувацьких ігор. Об'єктом роботи є галузь інтернет технологій, яка займається вивченням та розробкою багатокористувацьких он-лайн ігор, орієнтованих на запуск та використання в інтернет браузерах.

Предметом роботи є двомірна багатокористувацька он-лайн гра. Робота присвячена виявленню найкращого способу обміну даним між клієнтською та серверною стороною, та правильною відмальовкою програмно анімованих персонажей. Що є можливим завдяки використанню сучасного стеку технологій та простий, але вичерпний архітектурі.

Налагоджений життєвий цикл забезпечить надійну продуктивність та відлагоджену роботу гри, навіть при нестабільному інтернет зв'язку, або техніці з низькою обчислювальною потужністю. В той час як нові ігрові цілі та спосіб їх досягнення, зроблять проект більш конкурентним. Головною вимогою до реалізації життєвого циклу є неперервний обмін даними між клієнтом і сервером. Технологія Socket.IO забезпечує лише орієнтоване на події безперервне з'єднання. Механізм обміну був реалізований власноруч. Клієнтська частина використовує canvas, як простір для створення ігрового майданчика і інтерфейс взаємодії персонажа і користувача.

Для реалізації ігрового процесу було використано низку математичних розрахунків та алгоритмів, що описують дії гри. Весь дизайн був розроблений в графічному редакторі в Figma та Inkscape. У розробці задіяний обширний стек сучасних технологій, до яких входять: TypeScript мова програмування, NodeJS – JavaScript незалежно від оточення, React для побудови користувацького інтерфейсу, styled-components для стилізації компонент, Express фреймоврк для сервера, Socket.IO для двонаправленного зв'язку клієнта і сервера. Тестування проводилось у Chrome, Safari і Firefox на одному з найменших мобільних

пристроїв iPhone SE. Результатом виконаної роботи є створена багатокористувацька онлайн гра.

Ключові слова: гра мультиплеер, socket.io, react, typescript,nodejs, клієнт-серверна комунікація, життєвий цикл даних,багатокористувацька онлайн гра.

Постановка проблеми. Вже існує велика кількість конкурентів у галузі браузерних ігор. Оскільки процес створення гри на базі браузера передбачає велику кількість обхідних шляхів, розробка стає цікавою, а досвід глибшим. Необхідно шукати нестандартні підходи, щоб запобігти можливим втручанням у ігровий процес з боку користувача: ненавмисні дії, що можуть спричинити некоректну роботу користувальницького інтерфейсу; навмисні дії, що можуть негативно вплинути на логіку роботи всієї гри; дії, націлені на отримку переваги над супротивниками.

Передбачення всіх явних і неявних аспектів розробки та використання передових технологій у цій галузі мотивувало до створення особистої версії багатокористувацької двомірної гри.

Аналіз останніх досліджень і публікацій. Найбільш вживана мова програмування у мережі інтернет є JavaScript (далі JS). Вона проста у освоєнні, мультиплатформенна та має величезну підтримку спільноти у сфері розробці. Вибір JS також обумовлений уніфікацією мови для обох сторін розробки клієнту та серверу. За допомогою технології NodeJS [1] мову JS тепер можна використовувати за межами браузерного оточення. Хоча JS безумовно популярна, та має цілу низку готових рішень, вона має і недоліки. Один із головних — відсутність типізації даних. Цю проблему вирішує TypeScript [2]. TypeScript (далі TS) – додає статичну типізацію даних, таким чином значно підвищує контроль над усим додатком і дозволяє знайтипомилки ще на етапі написання коду. Окрім цього, TS додає ще не затверджений стандартом JS функціональні рішення, тим самим розширює можливості розробки. У роботі використовується TS як провідна мова програмування на обох сторонах. Для відтворення та конфігурації користувальницького інтерфейсу використовується бібліотека React [3].

Для стилізації компонент використовується бібліотека styled-components [4]. В порівнянні зі звичним CSS файлом, з набором селекторів та властивостей, styled-components надає більш декомпозований та описовий підхід до стилізації React компонент. Більше того, бібліотека створена спеціально для React, що дає стовідсоткову узгодженість технологій. На відміну від звичайного CSS, стилі присвоюються одразу на HTML елемент, якому користувач дає самоописуючу назву.

Можливість передавати властивості у стилізовані компоненти через props, робить стилізацію більш гнучкою та чуйною до змін. Оскільки проект передбачає обмін даними у реальному часі, необхідно дотримуватись використання веб-сокетів. Використання сокетів дає змогу

запобігти спілкування клієнту та серверу через обмін запитів та відповідей. Напроти, виконується лише один запит і після успішного з'єднання з сервером, зв'язок не втрачається. У роботі не використовується нативний об'єкт веб-сокету, який пропонує браузерний API. Замість нього використовується бібліотека Socket.IO [5], що є обгорткою над початковою реалізацією веб-сокету та значно розширює функціонал останньої.

На стороні сервера використовується найпопулярніший NodeJS фреймворк Express. Щоб конфігурувати гравців, їх необхідно якось ідентифікувати. Для того, щоб згенерувати неповторний ідентифікатор, використовується uuid (universally unique identifier). Встановлюється як звичайний npm пакет і надає доступ до чотирьох версій генерування ідентифікатора. Хорошою практикою є використання helmet [6]. Цей пакет дозволяє убезпечити обмін даними між клієнтом і сервером, додаючи різноманітні HTTP заголовки. Helmet, по суті, являє собою набір з дев'яти дрібніших функцій проміжної обробки, що забезпечують налаштування заголовків HTTP, пов'язану із захистом.

Для того щоб мати змогу конфігурувати наведені вище пакети, на проекті використовується npm [7]. Менеджер пакетів дозволяє встановлювати, видаляти та оновлювати бібліотеки та інструменти, які задіяні у розробці. Всі необхідні дані стосовно назв та версій проектів зберігаються у файлі-маніфесті package.json та файлі для збереження залежностей package-lock.json. Це дає змогу зберігати репозиторій проекту на віддалених платформах для хостингу таких як GitHub або Bitbucket, не зберігаючи там самих бібліотек та інструментів, бо їх можна відновити локально, використовуючи файл маніфест.

Щоб мати змогу у майбутньому залучити у розробку більше учасників та мати доступ до репозиторію з будь-якого пристрою, проект хоститься на GitHub платформі.

Формулювання цілей статті. Мета роботи полягає у визначенні підходу до створення оптимізованого життєвого циклу обміну даними між сервером та клієнтом. У головну мету також закладається оновлений ігровий процес, що має якісно виділитися на фоні існуючих альтернатив та урізноманітнити жанр.

Основна частина. Для реалізації ігрового процесу, необхідно було використати низку математичних розрахунків, щоб зробити можливим обробку наступних подій:

1 Переміщення гравця по ігровій арені, використовуючи подію переміщення миші.

2 Атака гравця за рахунок пострілу. Ця функція можлива завдяки обробці події кліку миші у певній точці ігрової арени.

3 Зіткнення будь-яких ігрових одиниць на полі, обробляється дуже схожими алгоритмами. Один і той самий алгоритм використовують наступні зіткнення: а) гравець з гравцем; б) гравець та зброя іншого гравця; в) гравець та частки, що пришвидшують оберти гравця.

Налаштування комунікації між клієнтом та сервером треба почати з опису клієнт-серверного спілкування, з використанням подій Socket.IO бібліотеки. Ця комунікація лежить в основі ігрового процесу та робить можливим безперервний обмін даними обох сторін. Нижче наведена UML діаграма послідовності (рис.1), яка описує порядок відтворення основних подій, їх термін життя та того, хто ініціював виконання події. Розглянемо діаграму більш детально:

– є три виконавці: користувач, клієнт та сервер;

– користувач ініціює відмальовку арени методом init. Цей же метод запускає подію ініціації на сервері;

– подія ініціації на сервері відповідає подією initReturn, разом з якою йдуть дані їжі для гравця;

– клієнт в свою чергу запускає подію tick, яка безперервно відправляю на обробку сервером координати вектора гравця. Там вираховується нові координати гравця;

– нові координати гравця відправляються з сервера подією tickTock;

– сервер також відправляє усім клієнтам масив усіх гравців;

– користувач постійно ініціює подію переміщення, яка обробляється клієнтом і сервером послідовно;

– користувач також може ініціювати атаку кліком миші у довільній точці ігрового поля;

– клієнт обробить клік і подією oneTimeAttack відправить на сервер дані для обробки атаки на відстані;

– якщо гравець програв, сервер повідомляє клієнта використовуючи подію playerDeath.

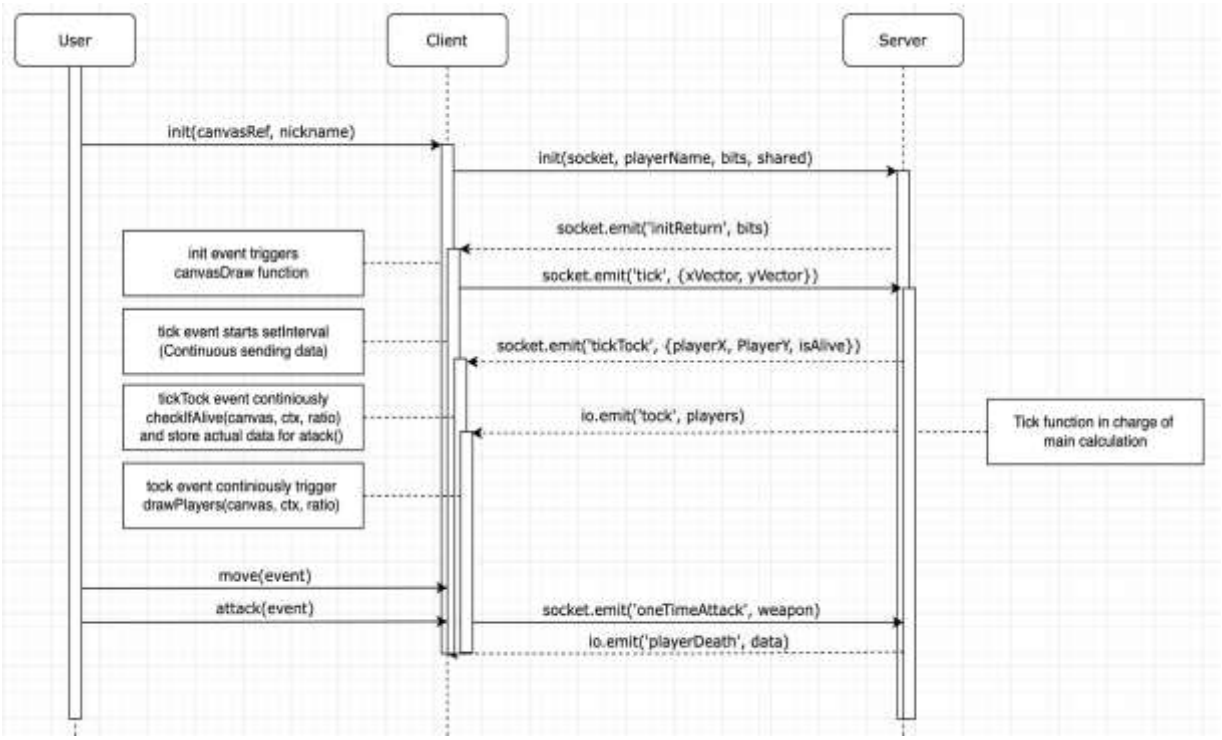

Рис. 1. Діаграма послідовностей

Були розроблені алгоритми, що описують вище переличені події. Приведемо один з них. Алгоритм “зіткнення” описує перетин двох будь-яких сутностей на просторі ігрової арени (рис. 2):

– отримуємо поточного гравця, всіх гравців та ідентифікатор;

– проходимо циклом по масиву поточного гравця;

– перевіряємо, щоб гравець не відкрив вогонь по собі;

– запускаємо цикл по масиву зброї;

– розраховуємо відстань від зброї до гравця;

– якщо відстань менша суми радіусі зброї та гравця: а) виключаємо зброю з масиву зброї; б) оновлюємо дошку лідерів.

– якщо більша, переходимо до наступного елемента циклу.

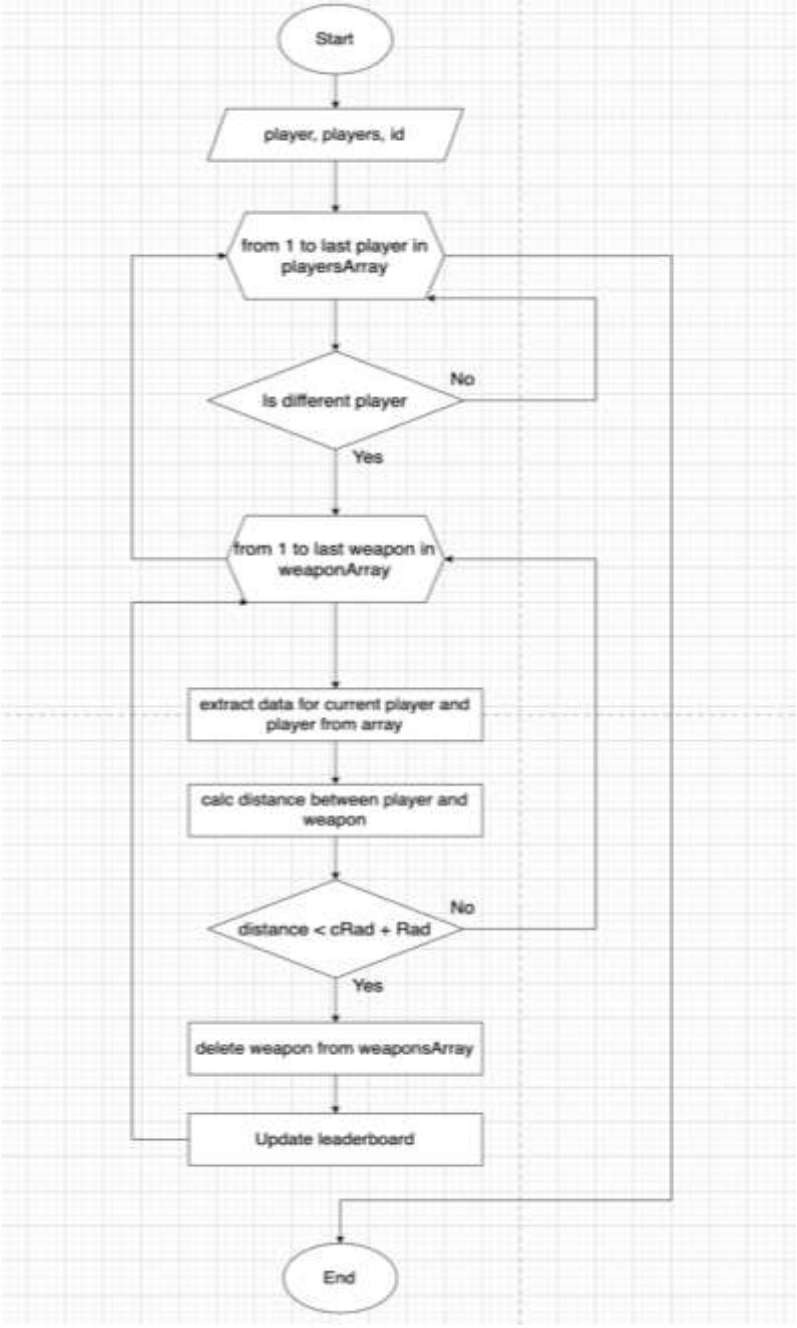

Рис. 2. Блок-схема алгоритму зіткнення

За допомогою бібліотеки React та styled-components було створено три стани користувацького інтерфейсу, котрі змінюються у процесі життя персонажа:

– стартова сторінка;

– ігрова арена;

– модальне вікно перезапуску.

Оскільки React це лише View у відомому патерні розробки MVC [10], вона не має інструментів настройки роутинга і це є плюсом. Таким чином ми використовуємо лише те, що необхідно проекту. Вищезгадані вигляди, контролюються змінами CSS властивостей, таких як display, opacity та visibility.

Цей підхід був запозичений у конкурентів Slither.io та Agar.io і виправдовується тим, що з сервером встановлений постійний зв’язок, який весь час використовується для відмальовки гри, навіть після того як персонаж програв. Він все ще може бачити ігрову арени під напівпрозорою модалкою. Тож переключення між роутерами не має сенсу, при такій кількості виглядів.

Головна сторінка відображає мінімальний інтерфейс характерний іграм даного жанру (рис.3).

Елементи користувальницького інтерфейсу:

– логотип Saw.io;

– напис у підтримку країни. Гра буде доступна публічно;

– поле для вводу нікнейму. Після завантаження сторінки фокус на полі;

– кнопка для початку гри;

– pop-up для пожертвувань.

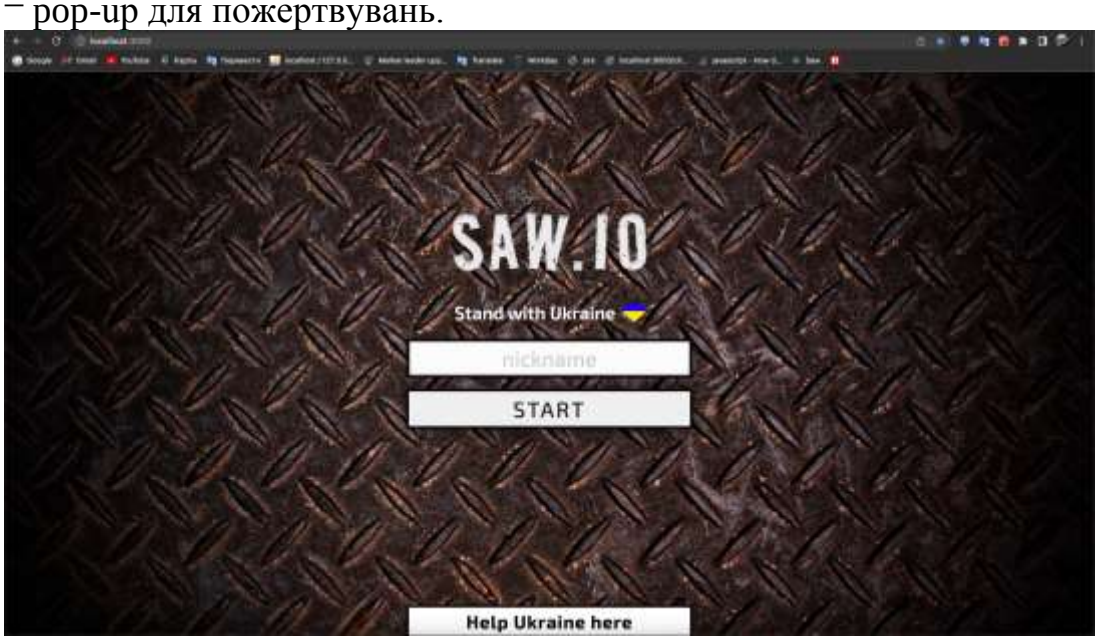

Рис. 3. Стартова сторінка

Ігрова арена займає всю сторінку та відображає наступні елементи інтерфейсу (рис.4):

1 Зліва вгорі таблиця лідерів. Першим відображається той гравець, швидкість обертів якого найбільша. У демонстраційних цілях видно лише одного.

2 По центру сторінки знаходиться гравець. Він має вигляд циркулярної пили. У центрі гравця його поточна кількість очок. Під ним його нікнейм. Гравець може не вводити ім’я, це його вибір.

3 Ігрова арена прямокутної форми окреслена червоною лінією; Торкнувшись її гравець одразу програє.

4 Червона заливка за межами арени це є не ігрова зона, там не може бути жодного ігрового елемента.

5 Різнокольорові точки розкидані по карті це їжа, що пришвидшує оберти гравця.

6 Подібна на гравця маленька циркулярна пила, це зброя дальнього ураження, як спрацьовує при натисканні лівої клавіші миші у будь-якій точці арени.

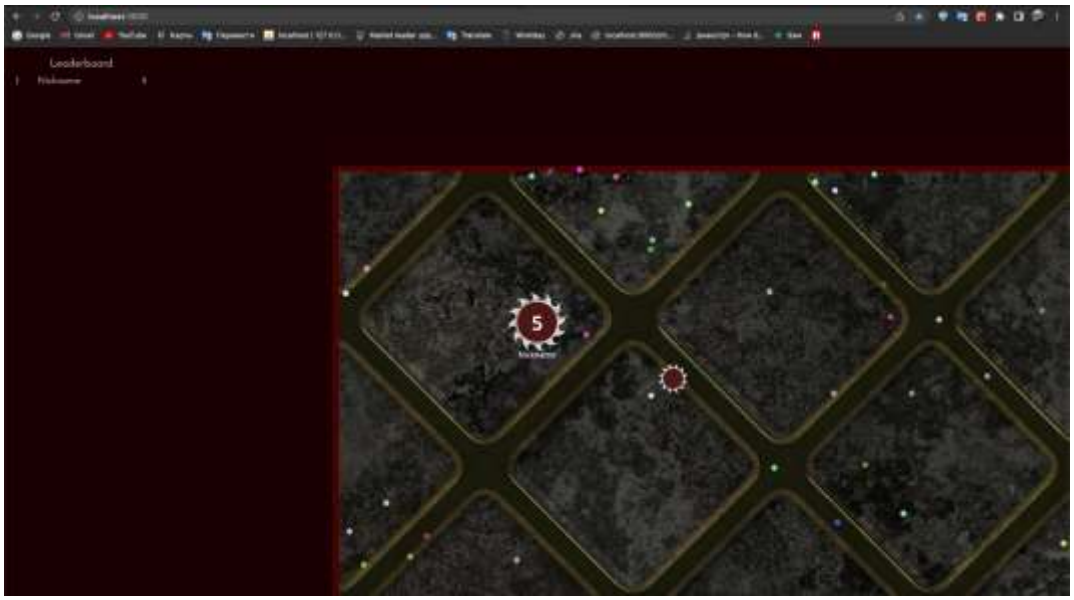

Рис. 4. Ігрова арена

Висновки. У підсумку роботи варто зазначити підхід, який дозволив встановити постійний обмін даними між клієнтом і сервером. Це стало можливим за рахунок використання вбудованих асинхронних методів setInterval на боці клієнта та сервера з нульовою затримкою. Під час реалізації виникали труднощі з продуктивністю, гра працювала з затримками і ривками. Причиною було те, що з кожним новим гравцем для нього заводився таймер, з використанням setInterval, але після поразки гравця, таймер не зупинявся, що спричинило фонове накопичення гравців у системі, навіть якщо їх не було на ігровому полі. Після ліквідації цієї проблеми, вдалось налагодити продуктивність. Було проведено ряд тестувань гри, а саме тестування у різних браузерах.

Література

1. Офіційна документація по NodeJS з використанням TypeScript. Режим доступу: https://nodejs.dev/learn/nodejs-with-typescript
2. Офіційна документація TypeScript. Режим доступу: https://www.typescriptlang.org/
3. Офіційна документація React. Режим доступу: https://uk.reactjs.org/
4. Офіційна документація по styled-components. Режим доступу: https://styled-components.com/docs
5. Офіційна документація Socket.IO. Режим доступу: https://socket.io/
6. Офіційна документація Helmet: Режим доступу: https://helmetjs.github.io/
7. Офіційна документація npm. Режим доступу: https://docs.npmjs.com/
8. Офіційна документація Git. Режим доступу: https://git-scm.com/doc
9. Керівництво по розрахункам зіткнення двох об'єктів у просторі. Електроний ресурс: https://www.youtube.com/watch?v=XYzA_kPWyJ8
10. Peter Späth (2020) Beginning Java MVC 1.0. Електроний ресурс: https://www.amazon.com/Beginning-Java-MVC-1-0-Microservices/dp/1484262794?asin=1484262794&revisionId=&format=4&depth=1

CREATION OF A DESIGN LAYOUT OF PAGES AND IDENTICS OF AN INTERNET COSMETICS STORE

Hanna Fedchenko, Mykola Matushenko, Kyrylo Holotenko

The work explores the actual topic associated with multiplayer games, which have always united a large number of people for leisure activities. When the game becomes available publicly, it will qualitatively diversify the genre of browser multiplayer games. The object of the work is the branch of Internet technologies, which is engaged in the study and development of multiplayer online games, focused on launch and use in Internet browsers.

The subject of the work is a two-dimensional online multiplayer game. The work is devoted to identifying the best way to exchange data between the client and server side, and the correct rendering of programmatically animated characters. Which is possible through the use of a modern technology stack and a simple but comprehensive architecture.

A well-established life cycle will ensure reliable performance and debugged operation of the game, even with unstable Internet connection, or equipment with low computing power. While new game goals and the way to achieve them, will make the project more competitive. The main requirement for the implementation of the life cycle is the continuous exchange of data between the client and the server. The Socket.IO technology provides only an event-

oriented connection. The exchange mechanism was implemented personally. The client part uses canvas as a space for creating a playground and an interface for character-user interaction.

To implement the gameplay, a number of mathematical calculations and algorithms describing the actions of the game were used. The entire design was developed in a graphic editor at Figma and Inkscape. The development involves an extensive stack of modern technologies, which include: TypeScript programming language, NodeJS - JavaScript regardless of the environment, React for building a user interface, styled-components for styling components, Express frame for server, Socket.IO for bidirectional communication between client and server. Testing was done in Chrome, Safari and Firefox on one of the smallest mobile devices, the iPhone SE.

Keywords: game multiplayer, socket.io, react, typescript, nodejs, client-server communication, data lifecycle, online multiplayer.

References

1. Official documentation on NodeJS using TypeScript. Retrieved from: https://nodejs.dev/learn/nodejs-with-typescript
2. Official TypeScript documentation. Retrieved from: https://www.typescriptlang.org/
3. Official Documentation of React. Retrieved from: https://uk.reactjs.org/
4. Official documentation on styled-components. Retrieved from: https://styled-components.com/docs
5. Official Documentation Socket.IO. Retrieved from: https://socket.io/
6. Official Documentation Helmet. Retrieved from: https://helmetjs.github.io/
7. Official Documentation npm. Retrieved from: https://docs.npmjs.com/
8. Official Documentation Git. Retrieved from: https://git-scm.com/doc
9. Guide to calculating the collision of two objects in space. Retrieved from: https://www.youtube.com/watch?v=XYzA_kPWyJ8
10. Peter Späth (2020) Beginning Java MVC 1.0. Retrieved from: https://www.amazon.com/Beginning-Java-MVC-1-0-Microservices/dp/1484262794?asin=1484262794&revisionId=&format=4&depth=1