Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Розробка браузерної гри «Over world» на базі Java Script із використанням Node is та Phaser

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 - інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	(прізвище та ініціали)
Керівник		Пастух О. А.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Бойко І. В.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		Тиш Є. В.
	(підпис)	(прізвище та ініціали)

Тернопіль
2020

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 54 с., 10 рис., 18 джер.

JAVASCRIPT, ІГРИ, ОНЛАЙН ІГРИ, ВЕБ, СИСТЕМИ РОЗПРИДІЛОНОЇ ОБРОБКИ ДАНИХ, NODE.JS, СИНХРОНІЗАЦІЯ, УЗГОДЖЕННЯ, ПІНГ.

Метою роботи є вирішення проблеми із надто довгою реакцією клієнта на дії гравця, у процесі розробки браузерної багатокористувацької онлайн гри «Over world».

Методи розробки базуються на технологіях синхронізації та узгодження, із застосуванням технологій ігрового движка із фіксованим часовим кроком, синхронізації часу за допомогою NTP, а також лінійної інтерполяції ігрових станів.

В результаті роботи сформовано алгоритм синхронізації та узгодження ігрових станів на клієнтах та сервері браузерної онлайн гри. Розроблений робочий зразок даного алгоритму для гри «Over world».

JAVASCRIPT, GAMES, ONLINE GAMES, WEB, DISTRIBUTED DATA PROCESSING SYSTEMS, NODE.JS, SYNCHRONIZATION, COORDINATION, PING.

The aim of the work is to solve the problem of too long reaction of the client to the player's actions, in the process of developing a browser multiplayer online game "Over world".

Development methods are based on synchronization and coordination technologies, with the use of game engine technologies with a fixed time step, time synchronization using NTP, as well as linear interpolation of game states.

As a result of work the algorithm of synchronization and coordination of game states on clients and the server of browser online game is formed. A working sample of this algorithm for the game "Over world" has been developed.

ЗМІСТ

ВСТУП

Комп’ютерні ігри в сучасному світі займають важливу роль у житті багатьох людей. Для когось це хобі, для інших – середовище спілкування, спосіб розслабитись та відволіктись від буденності, а хтось використовує ігри для творчої самореалізації. Та не залежно від методу досягнення, основна ціль комп’ютерних ігор - отримати задоволення від процесу чи результату. Цей ефект досягається шляхом винагородження гравця за певну дію, чи комплекс дій. При цьому в залежності від жанру, складності та реалізації гри, дією може бути як простий клік по ігровому полю, так і певна послідовність натискання клавіш та своєчасне прийняття вірних рішень - кожному своє. Так і з нагородою – це може бути як перемога над ворогом, чи розширення та доповнення ігрових механік, так і візуально естетична краса створеного в процесі, чи банальне збільшення цифри в лічильнику. Концепція загалом залишається тією ж - кожній дії гравця слідує наслідок, і якщо його дії правильні - гравець отримує бажаний результат.

В такій концепції швидка реакція ігрового стану на дії користувача є дуже важливою. Деякі жанри ігор буквально зав’язані на своєчасному та правильному натисканні клавіш. До таких можна віднести практично усі реал-тайм ігри, в особливості – шутери і стратегії. Яскравим прикладом можна вважати такі популярні ігри, як «dota 2», серія ігор «counter strike», чи «quake». В таких іграх запізніле натискання клавіші всього в одну десяту секунди може повністю змінити кінцевий результат. А якщо це запізнення зумовлене не реакцією гравця, а погано оптимізованим ігровим кодом, чи високим пінгом - задоволення від процесу доволі швидко перетворюється в цілий спектр негативних емоцій. Адже це протирічить самій основі - гравець зробив все вірно та своєчасно, але по вині сторонніх чинників, на які він не міг повипливати, не отримав нагороди. З іншої сторони, навіть в простих, покрокових чи повільних іграх надто помітна затримка реакції на натискання завжди викликає лише негативні емоції, хоч і не так гостро.

Із потужністю сучасних систем та ПК, зазвичай не складно звести затримку реакції програмного коду на дію гравця до рівня, практично не помітного людським сприйняттям. Чого не можна сказати про затримку, спричинену пінгом в онлайн іграх. Адже час надходження пакету даних від клієнта до сервера не залежить від розробника. І в момент, коли сервер отримає пакет із даними про дію гравця – вона може бути уже неактуальною. Особливо, коли йдеться про мобільний інтернет, чи невеликі проекти із одним-двома серверами, та клієнтами із всього освіту.

Вирішенням цієї проблеми є прогнозування та узгодження. Ігрова логіка дублюється на клієнтській і серверній частині. Логіка на клієнтській частині миттєво реагує на дії гравця, та відправляє повідомлення по них на сервер. Сервер в свою чергу прогнозує найбільш очікуваний розвиток ігрових подій, зазвичай такий, у якому жоден із гравців не виконує ніяких дій. Отримавши повідомлення від гравця про здійснення дії, сервер корегує свій прогноз, враховуючи нові дані, провіряючи їх, та повідомляє про це усіх інших клієнтів. З іншої сторони кожен клієнт так само прогнозує найбільш очікуваний розвиток подій, та отримавши дані від сервера – підлаштовує ігровий стан під них.

Метою даної магістерської роботи є формування і реалізація алгоритму прогнозування та узгодження для браузерної гри «Overworld».

Об’єктом дослідження є система розподіленої обробки даних в обличчі гри «Overworld», та аспекти синхронізації і узгодження даних у ній.

Предметом дослідження є затримка надходження пакетів – пінг, та його вплив на узгодження даних на різних пристроях.

В ході дослідження були використані як емпіричні, так і теоретичні методи дослідження. Проаналізувавши наявні рішення, було узагальнено алгоритм вирішення проблеми. Враховуючи особливості середовища реалізації – браузерної гри, підібрано найбільш відповідні технології та розробки, що реалізують ту чи іншу частину алгоритму. Для заповнення відсутніх елементів алгоритму було сформовано гіпотези, та провірено їх експериментальним шляхом. Синтезом усіх підібраних та сформованих елементів в одну цілісну систему – створено початкове рішення

проблеми. Визначивши, та проаналізувавши недоліки – сформовано пакет виправлень та налаштувань для повноцінно робочого рішення.

У процесі дослідження було уперше сформовано та реалізовано повноцінну систему синхронізації та узгодження даних з урахуванням особливостей та потреб середовища браузерної гри.

Отриману систему одразу буде використано у браузерній інді-грі «Overworld». Вона легко адаптується до практично будь якої іншої браузерної гри. Також, комплекс рішень, та загальний алгоритм реалізації можна використати у інших типах ігор, чи подібних по принципу дії системах розподіленої обробки даних.

Результати роботи були опубліковані у матеріалах і тезах VIII науково-технічної конференції «Інформаційні моделі, системи та технології», під заголовком «Реалізація синхронізації та узгодження даних у браузерній грі», у секції комп'ютерних систем та мереж, УДК 004.75 Розподілені системи оброблення даних (див. додаток Б).

1 АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ЛІТЕРАТУРИ

1.1 Аналіз вихідних умов

Розробка системи ведеться для браузерної онлайн реал тайм гри.

- Суть проблеми полягає у надто довгій затримці реакції ігрового стану на дії гравця при стандартній реалізації взаємодії.

- Ціль дослідження – скоротити цей час.

- Метод досягнення цілі – розробка системи синхронізації та узгодження, та його реалізація середовищі браузерної гри.

Браузерна комп’ютерна гра має ряд особливостей, що накладає певні обмеження на технології, які застосовуються при розробці звичайних комп’ютерних та мобільних ігор. Одним із них є обмеження на ресурси продуктивності – веб сторінка в браузері має вельми обмежений доступ до оперативної пам’яті та процесорного часу. Іншим – необхідність скачувати усі, або майже усі файли гри при кожному її запуску. Також, різні браузери можуть використовувати своєрідну оптимізацію, яка важким грузом лягає на ігровий движок. До прикладу, зупинка виконання коду у неактивних вкладках, чи умисне заниження кількості кадрів в секунду з цілю енергозбереження. Код та матеріали гри із браузера зазвичай можна спокійно скачати, та змінити. Через ряд цих недоліків, відомі розробники не особо бажають з ними зв’язуватись, а самі браузерні ігри користуються не найбільшою популярністю. Однак, у браузерних ігор є і ряд переваг, як крос платформна підтримка, та доступність.

Оскільки гра браузерна, мова розробки – JavaScript [1]. Це нетипізована скриптова мова програмування, реалізація стандарту ECMAScript [2]. Вона може виконуватись на будь яких пристроях із підтримкою браузерів, що полегшує рух в сторону крос платформної розробки. JavaScript є об’єктно-орієнтовною мовою, із використанням прототипів.

Серверна частина розробляється на Node.js [4] – програмній платформі, яка для розробки використовує мову JavaScript. Особливістю даної платформи є

движок V8, який транслює JavaScript напряму в машинний код. Також перевагою цієї платформи є велика кількість доступних, та можливість підключення зовнішніх бібліотек. Платформа орієнтована на асинхронну та подіє-орієнтовну розробку.

Оскільки запит до сервера, та очікування відповіді від нього відбувається кожен раз, як будь який із гравців натисне клавішу – для зв’язку використовується протокол вебсокет [5]. Його особливістю є можливість надсилання пакетів даних із будь якого місця коду, та їх асинхронна обробка одразу по отриманню. Для передачі, дані конвертуються у формат JSON [6] – текстовий формат представлення даних. За рахунок цього дані легко та зручно читати, та класифікувати. Також це надає можливість легко передавати цілі JS об’єкти між сервером та клієнтом.

Візуальна частина гри реалізована на ігровому движку Phaser [7]. Даний ігровий движок розроблений компанією «Photon Storm», та базується на візуальній бібліотеці Pixi.js. Pixi.js надає зручні можливості роботи із відображенням зображення за допомогою Canvas та WebGL, в залежності від можливостей браузера. Це робить даний движок гнучким та продуктивним в плані відображення ігрової графіки.

1.2 Огляд концепцій та технологій

1.2.1 Системи розподіленої обробки даних

Розподілена обробка даних [8] - обробка даних, виконувана на незалежних, але пов'язаних між собою комп'ютерах. Сучасні мережеві технології розподіленої обробки даних засновані на моделях архітектури «клієнт-сервер».

Клієнт – програмний комплекс, що знаходиться на робочій станції, обробляє запити користувача, та приймає відповіді від сервера.

Робоча станція – пристрій обчислення даних (ПК, смартфон, чи інший), підключений до мережу інтернет, та володіючий інтерфейсом взаємодії із користувачем. мережі функціонує як в мережевому, так і в локальному режимі. Він

оснащений власною операційною системою, та забезпечує користувача всіма необхідними інструментами для вирішення прикладних завдань.

Сервер – пристрій обчислення даних, що підключений до мережі і забезпечує її користувачів певними послугами. Сервери можуть здійснювати віддалену обробку завдань, зберігати дані, керувати базами даних, і ряд інших функцій, потреба в яких може виникнути у користувачів мережі. Сервер - джерело ресурсів мережі. Робоча станція-персональний комп'ютер, підключений до мережі, через який користувач отримує доступ до її ресурсів.

В процесі обробки даних клієнт за потреби формує запити на сервер. До прикладу, для обробки складних даних, отримання інформації із сервера, чи тестування отриманих клієнтом результатів. Сервер, згідно із записаною в нього програмою, виконує запит. Результати виконання запиту передаються клієнту. Клієнт обробляє отримані дані і надає результати обробки у зрозумілому для користувача вигляді.

1.2.2 Взаємодія сервера із клієнтом

Найпростішою реалізацією взаємодії є варіант, у якому сервер працює як ретранслятор. Уся ігрова логіка працює на стороні клієнта. По натисканню клавіші – клієнт миттєво реагує на неї, та відправляє дані про натискання. Сервер зберігає лише список підключених клієнтів. По отриманні даних – він просто пересилає ці дані іншим клієнтам. По отриманні даних із сервера – клієнт підлаштовує свій ігровий стан відповідно до них.

Плюси:

- Миттєва реакція на дії користувача. Як тільки користувач натиснув клавішу – його клієнт одразу відображає результат.
- Низьке навантаження на сервер. Сервер лише пересилає запити, не беручи участі у інших обчислюваннях.

Мінуси:

- Висока затримка дій інших гравців. За час подорожі пакету даних стан на інших клієнтах зміниться, та буде відрізнятись від того, у якому користувач натиснув клавішу. Це не лише спричинить затримку, але й може привести до критичної розсинхронізації.
- Вразливість перед втручанням в ігровий код для отримання переваг. Код у браузері легко змінити, та підмінивши дані – отримати перевагу над іншими гравцями.

Висновки: такий варіант розподілення обчислень у багатокористувацькій грі годиться лише для локальних мереж, або мікро-проектів, коли у гравців не достатньо мотивації змінювати ігровий код, а також не високі вимоги до якості синхронізації.

Наступним варіантом реалізації є перенос ігрової логіки на сервер. В даній реалізації клієнт володіє лише візуальним движком. Після натискання клавіші, клієнт відправляє дані про натискання на сервер. Сервер в цій реалізації тримає в пам’яті ігровий стан, та в замкнутому циклі постійно оновлює його згідно із ігровою логікою. Коли сервер отримує дані про натискання кнопки гравцем – він вносить відповідні корективи у ігровий стан, згідно із ігровою логікою. На кожному ігровому циклі сервер відправляє усім гравцям дані про актуальний ігровий стан. Клієнт відтворює кожен ігровий стан згідно із візуальним движком одразу по отриманні.

Плюси:

- Ігрова логіка захищена від втручання. Вся ігрова логіка обробляється сервером, і звичайні користувачі не мають можливості на неї повипливати іншим способом, окрім заданого розробником.
- Плавність зображення. Уся логіка обробляється в одному місці, немає різких змін ігрового стану, непередбачених ігровою логікою.

Мінуси:

- Високе навантаження на з’єднання. Кожен ігровий цикл сервер повинен відправляти кожному гравцю дані про стан ігрового поля.

- Сильна залежність від пінгу. Оскільки пінг – явище не стабільне, ігрові стани можуть надходити з різною затримкою, що може навіть порушити їх порядок.

Висновки: такий варіант розподілення обчислень у багатокористувацькій грі підходить для комп’ютерних ігор із низькою частотою ігрового циклу, не великою кількістю користувачів, та невеликим обсягом даних ігрового стану.

Іще одним варіантом попередньої реалізації є встановлення на клієнтську частину основних рушіїв ігрової логіки, в доповнення до візуального движка. В такому випадку після натискання клавіші клієнт відправляє данні про це на сервер. Сервер тримає у пам’яті дані усього ігрового стану. На ньому також реалізований власний движок із ігровою логікою. По отриманні повідомлення від клієнта – сервер провіряє його, обробляє відповідно до ігрової логіки, та вносить відповідні корективи в ігровий стан. Результат відправляється усім клієнтам. На відміну від попереднього варіанту, сервер надсилає дані не кожен ігровий цикл, а лише коли отримує пакет даних про дію гравця. Клієнти ж обновляють ігровий стан відповідно до основних рушіїв ігрової логіки (на приклад, якщо персонаж іде – він продовжить іти в ту ж сторону). Отримавши дані від сервера, клієнт підлаштовує свій ігровий стан під актуальний серверний.

Плюси:

- Як і в попередньому варіанті, ігрова логіка захищена від втручання. Основна ігрова логіка обробляється лише сервером, а зміни, що може вести гравець в доступну клієнту логіку – повипливають лише на візуальне відображення у цього ж клієнта.
- Зменшення кількості проблем з розсинхронізацією. За рахунок того, що уся логіка обробляється однаково та одночасно для всіх – проблеми із розсинхронізацією ігрової логіки не виникають.

Мінуси:

- Затримка у всіх. Оскільки навіть гравцю, який натиснув клавішу, потрібно очікувати відповіді сервера, щоб його клієнт виконав дію – проблема із затримкою лише загострюється.

- Візуальна розсинхронізація. Оскільки клієнт продовжує розраховувати ігровий стан на основі доступних йому даних, при надходженні нових даних із станом, що надто відрізняється від поточного, можуть виникати помітні візуальні баги.

Висновки: даний варіант реалізації підходить для більшості стандартних онлайн ігор, за умови що вони не потребують часу відгуку нижче, ніж середній очікуваний пінг.

Останній – варіант реалізації із повним дублюванням логіки. У цьому варіанті кожен із клієнтів володіє повною копією ігрової логіки. Клієнт може миттєво відреагувати на дію гравця, разом із цим відправивши повідомлення із дією та своєю реакцією на сервер. Сервер, згідно із своєю версією ігрового стану, провіряє реальність дії, та правильність реакції на неї. При необхідності, сервер коректує їх, та повідомляє клієнта про його помилку. По опрацюванні сервер відправляє актуальний ігровий стан усім іншим клієнтам. По отриманні ігрового стану від сервера, клієнт підлаштовує свій стан під нього.

Плюси:

- Можливість миттєво реагувати на дії гравця. Володіючи всією логікою гри, клієнт може одразу міняти ігровий стан відповідно до дій гравця.
- Захищеність від втручання в ігрову логіку. Гравець може втрутитись в ігрову логіку на своєму клієнті, але тоді повідомлення не пройде повірку сервером, і буде виправлено.

Мінуси:

- Неодночасність ігрових станів через пінг. Коли серверу приходить повідомлення із дією гравця – на клієнті ця дія уже обробляється певний час (час проходження шляху пакетом). Так само, доки повідомлення пройде шлях від сервера до іншого клієнта – воно може бути уже не актуальним.
- Як і в попередньому варіанті - легка візуальна розсинхронізація, у випадку якщо стан, наданий сервером, надто відрізняється від активного стану клієнту.

Висновки: цей варіант важчий, ніж попередні, але задовольняє потреби ігор, у яких необхідна швидка реакція ігрового стану на дії гравця. Однак він потребує доопрацювання, а саме реалізації узгодження та синхронізації.

1.2.3 Ігровий цикл

Для коректної роботи взаємодії сервера з клієнтами у варіанті із повним дублюванням логіки необхідно щоб ігровий цикл сервера та ігровий цикл кожного із клієнтів давали однакові результати за однаковий проміжок часу при однакових вхідних даних.

Розглянем поняття ігрового циклу – це цикл, в якому з певною періодичністю проходить обробка ігрової логіки, зчитування введених даних, та у випадку із клієнтською частиною – обновлення візуального стану гри на моніторі. Загальний алгоритм ігрового циклу зображено на рис. 1.1. Результат роботи ігрового циклу залежить від коду ігрової логіки, яку він виконує, та частоти її виконання. Для однакових результатів необхідно щоб ці дві складові співпадали, або компенсували одна одну, як на клієнті, так і на сервері.

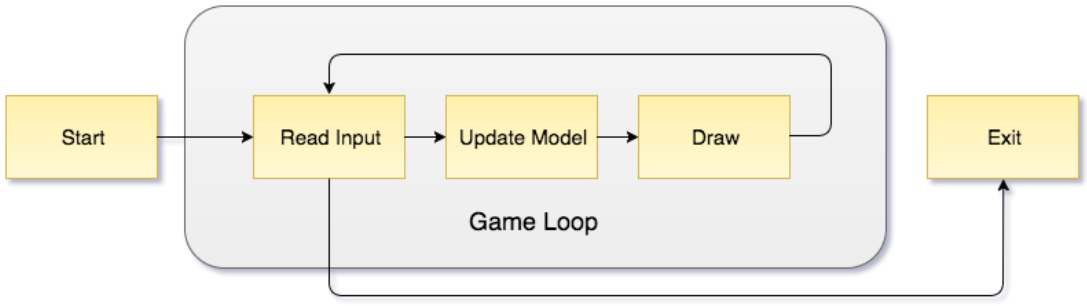

Рисунок 1.1 – Загальний вигляд алгоритму ігрового циклу

Однією із найбільш стандартних реалізацій ігрового циклу у браузері є його реалізація за допомогою функції setInterval(). Дана функція приймає два параметра, перший із яких повинен бути функцією, а другий – числом. Після запуску функції

setInterval(func, delay) із заданими параметрами, функція func буде періодично виконуватись через проміжок часу delay. Запрограмувавши функції func виконання одного циклу ігрової логіки, зчитування введених даних, та при потребі – обновлення візуального стану гри на моніторі, отримаєм готову реалізацію ігрового циклу. Недоліком такої реалізації є некоректна робота із неактивними вкладками. Так як на неактивній вкладні в цілях оптимізації виконання коду призупиняється, то після повернення на таку вкладку цикл викликається багаторазово, що приводить до некоректної поведінки гри. Також дана функція не надає високої точності частоти виклику.

Інший спосіб реалізації – функція requestAnimationFrame(). Дана функція приймає лише один параметр – функцію, яка буде виконуватись на наступний раз, коли браузер буде готовий обновити зображення на моніторі. Задавши як параметр requestAnimationFrame() ігрову логіку, зчитування введених даних та обновлення зображення ігрової сцени, та замкнувши цю функцію в циклі, отримаєм робочий ігровий цикл (рис. 1.2).

```
function update (){
  requestAnimationFrame(() => {
    logic();
    render();
    update();
  });
}
update();
```

Рисунок 1.2 – робочий варіант ігрового циклу на основі функції requestAnimationFrame

Недоліком цієї реалізації є сильна залежність швидкості від доволі нестабільної частоти обновлення екрану. Також даний ігровий цикл не працює на сервері, так як сервер не займається обновленням зображення на моніторі.

У статті «Разработка игр на JavaScript» [9] описані різні варіанти реалізації ігрового циклу для клієнтської гри на JavaScript, включаючи вище описані.

Кульмінацією цієї статті є розробка ігрового циклу із фіксованим часовим кроком, базова реалізація якого наведена на рисунку 1.3:

```
let last = performance.now(),
    step = 1 / 60, dt = 0, now;
let frame = () => {
  now = performance.now();
  dt = dt + Math.min(1, (now - last) / 1000);
  while(dt > step) {
    dt = dt - step;
    update(step);
  }
  last = now;
  render(dt);
  requestAnimationFrame(frame);
}
requestAnimationFrame(frame);
```

Рисунок 1.3 – базова реалізація ігрового циклу із фіксованим часовим кроком

Суть роботи даного ігрового циклу полягає у прив’язці частоти спрацювання до часу. Основний код цього циклу повинен виконуватись за першої можливості. Автор статті пропонує виконувати його як функцію, яку викликає requestAnimationFrame(). Перед запуском циклу виставляється параметр часу на 1 ігровий цикл, та фіксується початкова часова точка. Під час запуску циклу, з допомогою функції performance.now() фіксується активна часова мітка. Далі – визначається число, кількість часу, який пройшов з моменту останнього виконання коду ігрової логіки (або ж з початкової мітки запуску) в мілісекундах. Оскільки ігрова логіка в даній реалізації може виконуватись не кожен цикл – це число визначається як сума усіх попередніх різниць між циклами, до якої додається остання різниця між активною і останньою часовими мітками. Якщо це число більше, ніж заданий параметр часу на цикл – виконується код ігрової логіки. При цьому число зменшується на заданий параметр часу на цикл. По закінченню обробки ігрової логіки, число знову порівнюється із параметром, і якщо воно все ще більше – обробка ігрової логіки проводиться іще раз. Так відбувається до тих пір, поки число не стане меншим, ніж параметр часу на один цикл. По закінченні цього циклу, остання часова мітка виконання коду обновляється, згідно із

створеною в процесі активною часовою міткою. Проводиться обновлення зображення ігрового поля, після чого запускається новий requestAnimationFrame() для зациклення процесу.

Перевагами такої реалізації ігрового циклу є:

- Чітка і відносно стабільна частота спрацювання коду ігрової логіки. Частота прив'язана до часу, та при потребі само коректується: якщо частота requestAnimationFrame() менша, ніж задана, то ігрова логіка буде прораховуватись кілька разів за цикл. Якщо ж більша – обробка ігрової логіки буде пропускатись.
- Ігрова логіка відділена від функції оновлення зображення. Це дозволяє знизити частоту прорахунки ігрової логіки, не зменшивши кількість FPS.
- Можливість вирішити проблему неактивних вкладок, обмеживши кількість циклів ігрової логіки, які можуть бути запущені за один ігровий цикл.

Також, у статті коротко описується реалізація простої лінійної інтерполяції, яка вписується в ігровий цикл із розділеним виконанням ігрової логіки та оновлення зображення.

1.2.4 Узгодження

Узгодження в даному контексті – звіряння ігрових станів клієнта та сервера, та зведення їх відмінностей.

В статті «Синхронизация состояний в многопользовательских играх» [10] описана загальна концепція синхронізації та узгодження для варіанту із повним дублюванням ігрової логіки. Основна концепція, яку висвітлює стаття – зберігання в пам'яті сервера і клієнта певної кількості останніх ігрових станів. Суть концепції полягає в тому, що як клієнт, так і сервер роблять прогнози – розраховують актуальний ігровий стан на основі ігрової логіки відповідно до відомих їм даних.

Клієнт, зафіксувавши натискання клавіші гравцем, не миттєво реагує на неї, а лише формує реакцію та відкладає її на певний час. Паралельно клієнт відправляє дані про натискання, із прикріпленою часовою міткою, відповідно до запланованого часу. Під час надходження нових даних на сервер, вони добавляються в чергу на обробку. Коли приходить черга обробки отриманих даних, із пам’яті дістається відповідний часовій мітці ігровий стан, в який і вносяться зміни. Якщо часова мітка надто стара, або реакція не є коректною – сервер відмовляє клієнту, та змінює час та реакцію відповідно до ігрової логіки. Якщо виправлення не проводилось – сервер надсилає часову мітку дії, та реакцію усім, окрім клієнта, від якого прийшла дія. У іншому випадку, виправлена реакція і мітка відправляються усім клієнтам. Прогноз севрера перераховується згідно із новими змінами. Клієнт ж по отриманні даних, ідентично серверу, дістає із пам’яті відповідний часовій мітці ігровий стан, та симулювавши в ньому реакцію на дію відповідного гравця, виправляє свій прогноз. Перерахування активного ігрового стану здійснюється шляхом почергового перерахунку кожного ігрового стану від зміненого до активного.

Недоліками такого підходу є високе навантаження на ігровий клієнт, що особливо критично для браузерних ігор. Також автор згадує про проблему із різним системним часом на різних пристроях, та не вказує варіанту її вирішення. Однак корисною є пропозиція використання інтерполяції для згладжування візуальної розсинхронізації.

1.2.5 Синхронізація

Для узгодження даних сервера та клієнта необхідно щоб вони могли обмінюватись даними із часовими мітками. Та оскільки в умовах браузерної розробки наявний швидкий доступ лише до локального часу клієнта, який може відрізнятись від локального часу інших клієнтів та сервера – часові мітки втрачають сенс та інформативність. Із таким підходом, може виникнути проблема,

коли по часовій мітці клієнта, гравець зробив дію уже після того, як сервер про неї дізнався.

У статті «Протоколи синхронізації часу» [11] описані протоколи синхронізації часу. Особливо висвітлений протокол NTP - Network Time Protocol. Даний протокол використовує обмін спеціальними пакетами даних між сервером та клієнтом для синхронізації їх годинників. NTP пакет складається із таких полів:

Leap indicator (LI), 2 бітне число, що визначає секунду координації. Значення цього числа може коливатись від 0 до 3, де 0 – секунда координації відсутня, 1 – в останній хвилині дня на одну секунду більше, 2 – в останній хвилині дня на одну секунду менше, і 3 – сервер володіє неточним часом.

Version number (VN), 2 бітне число — номер версії NTP. Усього є 4 версії NTP, відповідно дане значення може приймати цифру від 1 до 4.

Mode — число, що відповідає за режим роботи. Якщо в даному полі значення 3 — значить відправник пакету - клієнт, якщо 4 — сервер.

Stratum — кількість посередників між клієнтом, та точним (атомним) годинником, включаючи сам NTP сервер.

Poll — число, що відповідає за частоту відправки послідовних запитів. Якщо відправник пакету – клієнт, він вказує через які проміжки часу в секундах буде надісланий наступний пакет. Якщо ж відправник – сервер, він вказує через які проміжки часу він приймає запити.

Precision — двійковий логарифм секунд, що характеризує точність локального годинника.

Root delay — час затримки пакетів еталонного часу від точного годинника до сервера.

Root dispersion — величина діапазону часових показань сервера.

RefID — ідентифікатор еталонного годинника. Якщо сервер налаштовувався відповідно до часу точного атомного годинника, то в цьому полі буде ідентифікатор цього атомного годинника (чотири ASCII символи). Якщо ж сервер налаштовувався відповідно до часу іншого NTP сервера – в цьому полі буде IP адреса цього сервера.

Reference — часова мітка, що відповідає останнім показанням серверного часу.

Originate — час відправки пакету згідно годинника сервера.

Receive — час отримання пакету сервером.

Transmit — час, який сервер витратив на опрацювання запиту.

Суть роботи NTP протоколу доволі проста – клієнт відправляє відповідний пакет NTP серверу, фіксуючи при цьому час відправки пакету відповідно до свого локального годинника. Сервер, отримавши запит, фіксує час його отримання. Далі пакет обробляється сервером. В нього вносяться дані про час прийому пакета сервером (по локальній версій серверного часу), кількість часу, витраченого на обробку пакету, та орієнтовний час відправки пакету назад клієнту, після чого пакет відправляється. Клієнт, отримавши оброблений пакет, він фіксує час отримання. Клієнт визначає скільки часу пройшло з його відправки сервером, шляхом знаходження половини різниці часу від відправи до отримання пакету, з вирахуванням часу на обробку пакету сервером. Знаючи цей час, клієнт може звірити свій час із серверним, та відкоригувати його.

Оскільки пакети можуть приходити із різною затримкою від клієнта до сервера, і назад – одноразове звіряння часу таким методом не є точним. Тому дана операція обміну пакетами, і корегування часу проводиться багаторазово, із різними серверами паралельно.

Оскільки в процесі гри до сервера може бути підключений лише один клієнт, а для вирішення задачі достатньо визначення лише приблизної різниці у часі – даний алгоритм доречно спростити.

1.2.6 Інтерполяція

Інтерполяція [12] – знаходження значення певної величини (точне, або приблизне) певним методом, виходячи із інших відомих значень даної величини. В

даному контексті - знаходження невідомих проміжних станів між двома відомими фактичними. В тому числі лінійна інтерполяція – знаходження певної проміжної позиції між двома точками – початковою та кінцевою, відповідно до коефіцієнту завершення. Ця точка вираховується як сума координат стартової позиції із різницею між кінцевою і стартовою позицією, помножена на коефіцієнт завершення. Далі наведені формули (1.1) для лінійної інтерполяції у декартовій системі координат, та її візуалізація (рис. 1.4):

$$x_t = x_0 + (x_1 - x_0) * t, \qquad y_t = y_0 + (y_1 - y_0) * t \tag{1.1}$$

Де: x_t та y_t – шукані проміжні координати по осям x та y відповідно, x_0 та y_0 – координати початкових точок по осям x та y відповідно, x_1 та y_1 – координати кінцевих точок по осям x та y відповідно, t – коефіцієнт завершення.

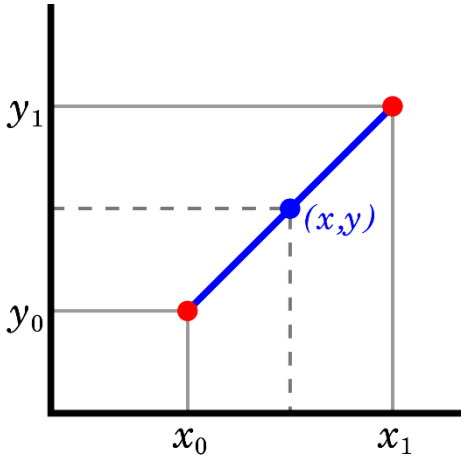

Рисунок 1.4 – вигляд лінійної інтерполяції у декартовій системі координат

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Вимоги до системи

Для вирішення проблеми із надто довгою реакцією ігрового стану на дії гравця було вирішено реалізувати систему синхронізації та узгодження в середовищі браузерної гри.

Для роботи даної системи необхідно виконання наступних умов:

1) Сервер та клієнт повинні мати можливість обмінюватись даними про дії гравця, реакцію ігрової логіки, та ігрові стани.
2) Як клієнт, так і сервер повинні повноцінно обробляти ігрову логіку. Це зумовлено специфікою прогнозування, і дасть можливість клієнту миттєво реагувати на дію гравця, не очікуючи відповіді сервера.
3) Результат обробки однакових вхідних даних за однаковий час на будь якому клієнті, чи сервері, повинен бути ідентичним. Якщо результати клієнта і сервера відрізняються – необхідна функція згладжування, яка буде нівелювати цю різницю. У іншому випадку накопичена різниця приведе до розсинхронізації, візуальних багів, та некоректної поведінки гри.
4) Сервер повинен мати можливість визначати в який момент часу клієнт прийняв та почав обробляти дію гравця. Відповідно до цього - часова шкала кожного клієнта повинна співпадати із часовою шкалою сервера. У випадку різниці між показниками часових шкал – ця різниця повинна бути визначена та врахована. Допускається неточність у розмірі, менше одного ігрового циклу. У іншому випадку узгодження буде працювати некоректно, а також можливе виникнення некоректних ситуацій.
5) Сервер повинен мати можливість прийняти та обробити дію гравця. Результат дії гравця повинен обробитись в контексті того ігрового стану, в який він був здійснений.
6) Сервер повинен мати можливість оновити активний ігровий стан, враховуючи усі зміни, що внесли дії гравців у попередніх станах.

7) Клієнт повинен мати можливість відкоригувати свій ігровий стан відповідно до серверного. Це коригування повинно проходити достатньо плавно, щоб не викликати візуальних дефектів.

2.2 Розробка концепції системи

У варіанті реалізації розподілення обчислень між клієнтом та сервером із повним дублюванням ігрової логіки, як клієнт так і сервер можуть опрацьовувати дії гравця однаковим чином. Для цього уся ігрова логіка дублюється. При цьому ігрова логіка на сервері спрощується, методом відкидання логіки візуалізації. На клієнті ж відкидається частина ігрової логіки, яка не підлягає прогнозуванню. До такої логіки відносяться односторонні дії, коригування яких недопустиме (на приклад смерть персонажа).

Виходячи із ознак ігрового циклу, окрім однакового методу обробки даних необхідна однакова частота їх оновлення. Для цього доречно використати ігровий цикл із заданим часовим кроком. requestAnimationFrame() доступний лише на клієнтській частині. Альтернативою для нього на серверній частині є setImmediate(). setImmediate() запускає код одразу, як тільки у процесора з’явиться можливість його обробити. В процесі обробки ігрового циклу, функція само викликається одразу після закінчення роботи. setImmediate() не рекомендовано використовувати в таких ситуаціях через створення високого навантаження на процесор. Альтернативою є повернення до старого setInterval(), із коротким часом виклику. При такій реалізації, цикл із заданим часовим кроком відрегулює частоту виклику, а setInterval() створить паузи між викликами для обробки інших асинхронних задач.

Також, за допомогою ігрового циклу із заданим часовим кроком, можна встановити необхідну частоту обробки ігрової логіки, не залежну від частоти оновлення зображення, що зменшить навантаження на сервер та клієнти. Для цього

згідно із ігровою логікою прогнозується один наступний ігровий стан, а проміжні стани та позиції ігрових об’єктів знаходяться шляхом інтерполяції між цими двома станами, відповідно до пройденого часу.

При здійсненні дії гравцем, браузер реагує на неї, а також відправляє дані про дію, та свою реакцію серверу, разом із часовою міткою дії. Для валідності часових міток необхідна синхронізація годинників сервера та клієнта. Для цього розроблена функція, яка працює по схожому до NTP принципу. Після під’єднання клієнта до сервера, йому відправляється черга пінг-понг запитів, із заданою періодичністю. При цьому кожному запиту призначається відповідний його черзі id, а час відправки фіксується в масиві. Клієнт, по отриманні запиту, вказує свій час, та відправляє його назад на сервер. При отриманні поверненого запиту, сервер фіксує час отримання, визначає та зберігає пінг, як різницю між цим часом, та часом відправки запиту із відповідним id. Також, відповідно до пінгу, зберігається проміжна різниця у часі, отримана шляхом вирахування із отриманої різниці половини величини пінгу. В загальному визначення проміжної різниці в часі наведено у вигляді формули 2.1:

$$Рч = (Чс - Чк) - \frac{1}{2}(Чо - Чв) \quad (2.1)$$

Де: Р – проміжна різниця в часі, Чо – час отримання запиту, Чв – час відправки запиту, Чс – значення часової мітки сервера, Чк – значення часової мітки клієнта

Після цього обновляється значення орієнтовної різниці в часі між клієнтом та сервером. Вона визначається шляхом знаходження середнього арифметичного між всіма уже отриманими проміжними різницями в часі.

$$Ро = \sum_0^n Рч \quad (2.2)$$

Де: Ро – орієнтовна різниця в часі, Рч – проміжна різниця в часі, n – кількість уже відомих проміжних різниць

Це значення також відправляється клієнтові, із зміненим знаком, для зворотної синхронізації. За рахунок великої вибірки, різниця між затратами часу на проходження пакетом шляху від сервера до клієнта і від клієнта до сервера, нівелює одна одну. Такий алгоритм дозволяє достатньо точно, в рамках поставленої задачі, визначити різницю часу між сервером та клієнтом, без необхідності змінювати системний час, чи використовувати сторонні сервери. Також перевагою такого алгоритму є компактність, простота, та можливість отримати наближене значення різниці в часі іще до закінчення роботи алгоритму.

Для узгодження, сервер зберігає в пам’яті масив останніх оброблених ігрових станів, та масив із часовими мітками їх початків. По закінченню кожного ігрового циклу, обидва ці масиви зсуваються на позицію в сторону збільшення. Позицію першого елементу першого масиву займає новостворений ігровий стан, а другого – відповідна йому часова мітка. Таким чином між цими двома масивами зберігається відповідність. Масив часових міток потрібен для визначення, до якого саме із минулих ігрових станів потрібно застосувати дію. Масив станів – для отримання цього стану. Також зберігається масив останніх дій, які надходили від гравців. Цей масив необхідний для повторення змін в потоці ігрового циклу, спричинених діями гравців, під час перерахунку активного ігрового стану.

При отриманні запиту, сервер додає значення орієнтовної різниці у часі до часової мітки клієнта, щоб вона відповідала серверному часу. Якщо мітка надто стара – вибирається найстаріша із доступних, а варіант дії гравця відмічається як неправильний. Знайшовши відповідний часовій мітці стан, сервер симулює дію гравця через ігрову логіку, та порівнює результат із отриманим від клієнта. При різниці в результатах, запит відмічається як неправильний. Далі сервер перераховує стільки ігрових станів, скільки було обраховано з часу зміненого стану до часу активного стану, враховуючи дії інших гравців. Після закінчення обрахунків сервер відправляє дані про дію гравця, та активний ігровий стан із його часовою міткою усім клієнтам окрім того, що здійснив дію. Якщо ж запит помічений як неправильний – то і йому теж.

По отриманні активного ігрового стану, згідно із часовою міткою та даними про різницю у часі, клієнт визначає скільки ігрових циклів пройшло з моменту відправки повідомлення сервером. До цього числа додається коефіцієнт згладжування, для створення запасу часу на роботу інтерполяції. Отримане число – кількість раз із мінімальною затримкою проходить через цикл ігрової логіки надісланий сервером стан.

В результаті клієнт володіє двома ігровими станами – активним неузгодженим, та майбутнім отриманим від сервера. Шляхом інтерполяції за кількість циклів, що рівні коефіцієнту згладжування, активний стан переводиться у отриманий від сервера.

Таким чином по закінченню процесу згладжування ігровий стан на сервері буде ідентичним ігровому стані на клієнті у той же момент часу.

На рисунку 2.1 зображено діаграму варіантів використання системи:

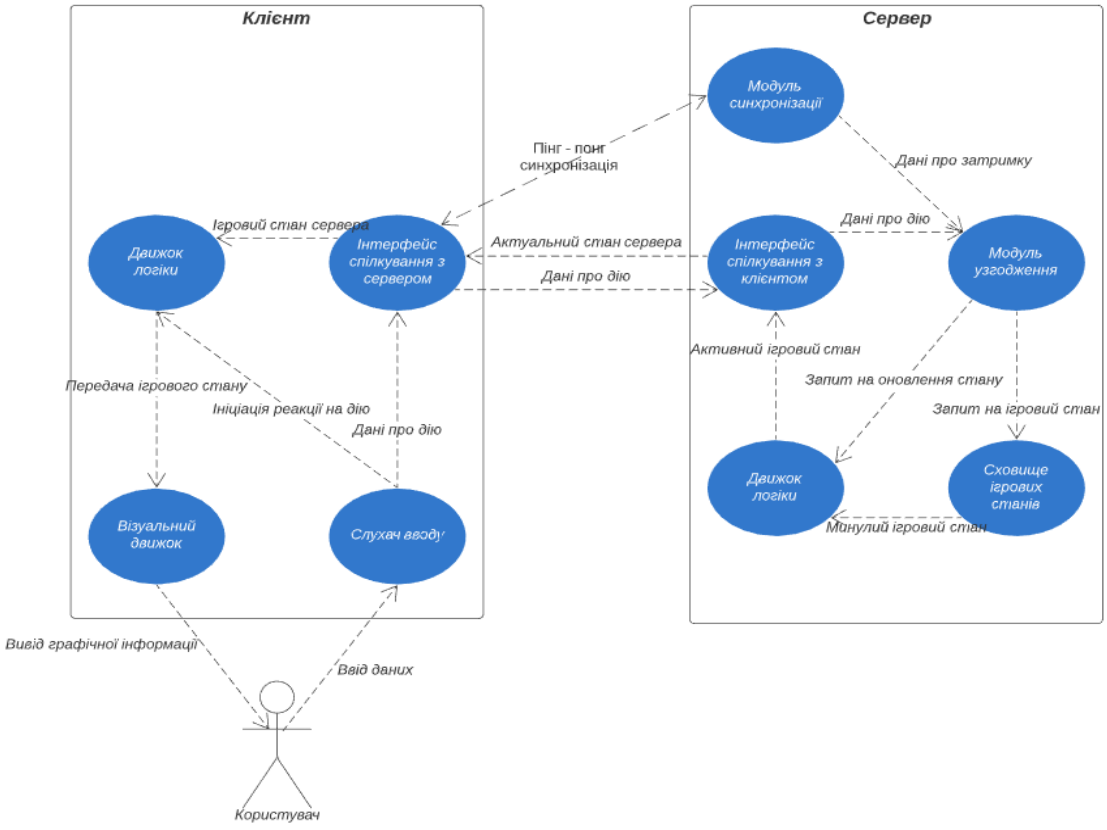

Рисунок 2.1 – діаграма варіантів використання системи

3 РОЗРОБКА РОБОЧОГО ЗРАЗКА

3.1 Реалізація сервера

Сервер являє собою файл server.js, та ряд бібліотек, що він підключає, включаючи самописні. Файл server.js запускається одноразово, із командної строки сервера, за допомогою команди node server.js. Загрузивши та ініціалізувавши необхідні бібліотеки, та функцію обміну файлами, за допомогою бібліотеки express [13] створюється об’єкт серверу. Запускається прослуховування 3000-го порту. У відповідь на запит, сервер надсилає файл index.html , та набір файлів і зображень, що в сукупності складають клієнтську частину гри.

Ініціалізується ігровий движок. За це відповідає окрема бібліотека із ігровою логікою. Вона створює ігрове середовище, та циклічно обробляє його з допомогою циклу із фіксованим часовим кроком. Ігрове середовище складається із незмінних об’єктів, як карта рівня, чи карта шляхів, і змінних, як позиції і стани гравців, та інтерактивних об’єктів, що формують ігровий стан. Карта рівня, та карта шляхів формується із JSON файлів за допомогою самописної бібліотеки mapEngine.js. Також ця бібліотека відповідає за функціонал кожного інтерактивного тайла.

Ігровий стан, а саме – стан усіх гравців та інтерактивних тайлів, після кожної обробки його ігровою логікою, зберігається у масив попередніх ігрових станів. Цей масив працює по принципу черги – зберігається 50 останніх циклів. Після обробки кожного нового циклу, він добавляється в масив з допомогою методу unshift, який зсуває усі елементи на одну позицію в сторону збільшення, а на місце першої позиції ставить переданий йому елемент. Паралельно таким ж методом формується масив із часовими мітками запису кожного стану. Якщо розмір масивів більше ніж 49, перед записом нового, останній елемент видаляється з допомогою методу pop. Дана функція є методом об’єкта timeRep, що відповідає за роботу із попередніми ігровими станами. Її код зображено на рисунку 3.1:

```
tick : function (state){
    if(this.stateRep.length >= 50){
        this.stateRep.pop();
        this.timeBasa.pop();
    }
    this.stateRep.unshift(state);
    this.timeBasa.unshift(Date.now());
},
```

Рисунок 3.1 – Код функції роботи із масивами попередніх ігрових станів

Ігровий світ – поле із тайлів розміром 64*64 пікселя. Файл світу створюється окремо, в програмній утиліті Tiled Map Editor. Карта шляхів генерується відповідно до тайлів світу. Вона необхідна для роботи алгоритму пошуку шляхів, який підключається та ініціалізується як окрема бібліотека.

Клас гравця прописаний у самописній бібліотеці playersEngine.js). Цей клас описує усі характеристики гравця, а також методи що описують його ігрову логіку. Одним із таких є метод проходження шляху goPath, що запускається кожен ігровий цикл для кожного гравця. Якщо у властивостях гравця відсутній шлях (параметр path рівний пустому масиву), то обробка проходження шліху активного гравця завершується. У іншому випадку, з допомогою лічильника кроків, визначаються координати наступної точки шляху, та відстань до неї. Якщо вона далі, ніж параметр швидкості гравця (фактично – відстань, яку гравець проходить за ігровий цикл) то координати гравця змінюються у сторону точки на відстань параметра швидкості. Якщо ж точка ближче – провіряється, чи ця точка не є кінцевою. Якщо це так – то координати гравця прирівнюються до координат точки, а лічильник кроків та шлях гравця обтуляться. Шлях рахується закінченим. Якщо ж ця точка являється проміжною, то відстань до неї вираховується із швидкості гравця. Координати гравця просуваються від цієї точки шляху до наступної на отриману величину. Для зміни координат гравця використовується функція moveTo.

Далі ініціалізується веб-сокет, для обміну даними. З’єднання сервера із клієнтом через веб-сокет налагоджується одразу після підключення, та зберігається у масив активних з’єднань. Обмін даними проходить наступним чином: по

отриманні повідомлення, воно розшифровується, та перетворюється у JS об’єкт. Об’єкт складається із обов’язкового текстового поля «type», яке вказує на вміситиме, та спосіб його обробки. Можливі типи об’єктів:

- "Login" – цей тип об’єкта надсилає ново підключений клієнт при спробі зайти у свій ігровий аккаунт. Він містить введені користувачем логін та пароль. По отриманні такого пакета, у базі проводиться пошук відповідного логіна, після чого звіряється пароль. Якщо паролі співпадають – клієнту надсилається дані про його персонажа, та дані ігрового середовища, включаючи дані про всіх інших персонажів. Також, персонаж ініціалізується, та добавляється в ігровий стан, що обробляється на сервері. Паралельно запускається функція знаходження різниці у часі findTimeDifference(). Параметром, який вона приймає, є саме з’єднання. Дана функція прив’язує до з’єднання об’єкт із даними про часову різницю, та запускає ряд пінг-понг запитів для її визначення. Код цієї функції наведено на рисунку 3.2:

```
let findTimeDifference = function (connection){
        connection.timeDif = {
                timeMas : [],
                pingMas : [],
                i : 0,
                difMas : [],
                dif : 0
        }

        let timerId = setInterval(function (){
                let time = Date.now();
                connection.timeDif.timeMas.push(time);
                connection.send(JSON.stringify({type : "ping", time : time, i : connection.timeDif.
                connection.timeDif.i++;
                if (connection.timeDif.i > 20) clearInterval(timerId);
        }, 1000);
}
```

Рисунок 3.2 – Код функції findTimeDifference

- "Pong" – цей тип об’єкта надсилається клієнтом у відповідь на повідомлення із типом "ping". Він містить ідентифікатор пінг-запиту, та матку локального часу сервера. По отриманні такого повідомлення, сервер визначає за формулою 2.1 проміжну різницю в часі, та оновлює орієнтовну

часову різню згідно із формулою 2.2. Дані значення записуються у відповідні поля, прикріплені до об’єкту з’єднання. Уривок коду із даною обробкою такого об’єкта наведений на рисунку 3.3:

```
let time = Date.now();
let ping =  time - connection.timeDif.timeMas[msg.i];
connection.timeDif.pingMas[msg.i] = ping;
console.log(ping + " - ping");
connection.timeDif.difMas[msg.i] = time - (msg.time + ping/2);for(let j=0;
j <  connection.timeDif.difMas.length; j++){
        let count = 0, k = 0;
        if(connection.timeDif.difMas[j]){
                count += connection.timeDif.difMas[j];
                k++;
}
connection.timeDif.dif = count/k;
connection.send(JSON.stringify({type : "timeDif", timeDif : connection.timeDif.dif));
}
```

Рисунок 3.3 – Код функції обробки об’єкту типу "Pong"

- "patsReqest" – тип об’єкта, сформованого клієнтом з цілю проінформувати сервер про дію гравця. Об’єкт містить часову мітку із часом клієнта, дані про дію гравця (координати кліку), та результат реакції на цю дію (шлях, згенерований алгоритмом пошуку шляху). Сервер конвертує мітку часу клієнта в таку, що відповідає серверному часу, та відправляє запит на обробку в ігровий движок.
- "tileActivate" – тип об’єкта, що вказує на взаємодію із інтерактивним тайлом.
- "voice" – тип об’єкта, що містить уривок голосового потоку, надісланий клієнтом. Відправляється усім підключеним клієнтам, окрім відправника.

По отриманні даних про дію, відповідно до часової мітки ігровий движок визначає до якого із попередніх ігрових станів її потрібно застосувати. З допомогою ігрової логіки симулює реакцію на дію гравця (з допомогою алгоритму пошуку шляху, відповідно до отриманих координат кліку, відомих координат кравця та карти шляхів, генерує шлях). Отриманий шлях порівнюється з шляхом із повідомлення для визначення коректності, та необхідності відправляти правлений

ігровий стан ініціатору дії. Дані про зміну шляху записуються у відповідний елемент масиву дій, та записуються як поточний шлях гравця у отриманому із пам’яті ігровому стані. До цього ігрового стану застосовується цикл ігрової логіки стільки раз, яка позиція була у цього стану у масиві, із врахуванням дій інших гравців. По закінченні, отриманий ігровий стан заміняє активний у потоці ігрового циклу, а данні про нього надсилаються усім гравцям у вигляді об’єктів із позиціями всіх гравців, їх активними шляхами, часовою міткою та типом "correct", що вказує на необхідність коректувати ігровий стан.

За відправку повідомлень усім підключеним клієнтам відповідає функція sendEveryone(), яка приймає два параметри. Перший – об’єкт, який потрібно відправити, а другий – винятки, тобто клієнти, яким це повідомлення відправляти не потрібно. Функція перетворює формат отриманого об’єкта у JSON. Проходячи циклом масив усіх активних клієнтів, вони порівнюються із винятком, та якщо не співпадають з ним – повідомлення надсилається.

При відключенні, об’єкт гравця зберігається у базу, та видаляється із активного стану гри. З’єднання також видаляється із масиву активних з’єднань, а усім іншим підключеним клієнтам відправляється повідомлення типу «disconectUser», із ідентифікатором відповідного користувача.

3.2 Реалізація клієнта

Клієнтська частина являє собою файл index.html, та набір файлів і зображень, які сервер надсилає клієнту після підключення. Ігрова сцена, та всі візуальні об’єкти формуються з допомогою phaser.js. Візуальні складові гри підвантажуються з допомогою JSON паків.

Після запуску клієнт запускає сцену стартового ігрового меню, та очікує введення даних. Ініціалізується та запускається ігровий движок. Сцена стартового ігрового меню складається із поля вводу логіну, поля вводу паролю, та кнопки

«увійти». По натисканню кнопки, на сервер відправляється значення полів, та очікується відповідь. При успішній авторизації сцена стартового ігрового меню завершує свою роботу, та запускається ігрова сцена. Як ігрова карта, так і стани всіх об’єктів та гравців формуються згідно із даними, отриманими із сервера.

Перехід вклади в неактивний режим запускає слухача onblur, що зупиняє ігровий цикл. При переході вкладки в активний режим, відправляється запит на сервер із типом "blur". По отриманні відповіді, клієнт визнає на скільки циклів отриманий стан застарів. Часова мітка сервера приводиться у відповідність клієнтському часу шляхом додавання отриманої орієнтовної різниці у часі. Фактична кількість циклів, на які відстає отриманий стан від активного визначається шляхом розділення часу, що пройшов з моменту відправки повідомлення сервером, на протяжність одного циклу.

Персонажі створюються з допомогою конструктору, подібного до такого ж на сервері, та з певними додатковими параметрами для обробки відображення. Також об’єкти персонажів володіють тією ж ігровою логікою – методом goPath, та його доповненням moveTo. На відміну від серверного варіанту, дані методи не надсилають повідомлень, але фіксують напрям руху гравця для використання відповідно повернутого спрайту.

Натискання клавіш фіксує функція-слухач. По натисканні клавіші мишки, визначаються координати мишки. Альтернативно функція адаптована для роботи із тач-скріном, у такому випадку фіксуються координати дотику. Визначається тип тайлу під мишкою в момент натискання. Якщо цей тайл інтерактивний – визначається чи може гравець змінити його стан (чи знаходиться він достатньо близько). Якщо цей тайл не інтерактивний – з допомогою карти та алгоритму пошуку шляху визначається чи може гравець дійти до нього. Якщо тайл не інтерактивний, та гравець не може дійти до нього – нічого не відбувається. Така перевірка зменшить кількість непотрібних запитів на сервер. Якщо ж дія гравця певним чином впливає на ігровий стан – клієнт гри реагує на цю дію, та відправляє повідомлення відповідного типу на сервер.

По отриманні повідомлення типу "Ping", клієнт добавляє до нього дані про свій час, отримані з допомогою функції Date.now(), міняє тип повідомлення на "Pong", та відправляє назад на сервер.

По отриманні повідомлення типу "correct" із ігровим станом, а саме – координатами та активними шляхами всіх гравців, клієнт прогнозує положення усіх гравців згідно ново отриманої інформації через секунду. Для цього визначається затримка повідомлення, згідно із часовою міткою сервера, та орієнтовної різниці часів. До цього числа додається 1000, а отримана сума ділиться на кількість мілісекунд, виділених на один ігровий цикл. Результат – число циклів, які асинхронно проходить отриманий із сервера стан, для створення прогнозу. Визначення цього числа також наведене у вигляді формули 3.1:

$$Ц = \frac{(Чк-(Чс+Рч))+1000}{t} \quad (3.1)$$

Де: Ц – необхідна кількість циклів, t – протяжність одного циклу, Чк – активний час клієнта, Чс – час сервера, отриманий в повідомленні, Рч – орієнтовна різниця часу.

3.3 Тестування

Для тестування коректності роботи розробленої функції визначення орієнтовної часової різниці, було добавлено виведення її результатів в консоль сервера. На рисунку 3.4 зображено кінцеві результати виводу функції при повторному запуску. Для тестування було взято по 7 запусків функції для двох різних пристроїв із різними пінгами. Останній пінг вказаний над кожним із результатів. Виходячи із отриманих результатів, а саме із різниці між найвищим та найнижчим значеннями, можна зробити висновок, що функція дає результат із точністю до 4-х мілісекунд, чого цілком достатньо для поставленої задачі.

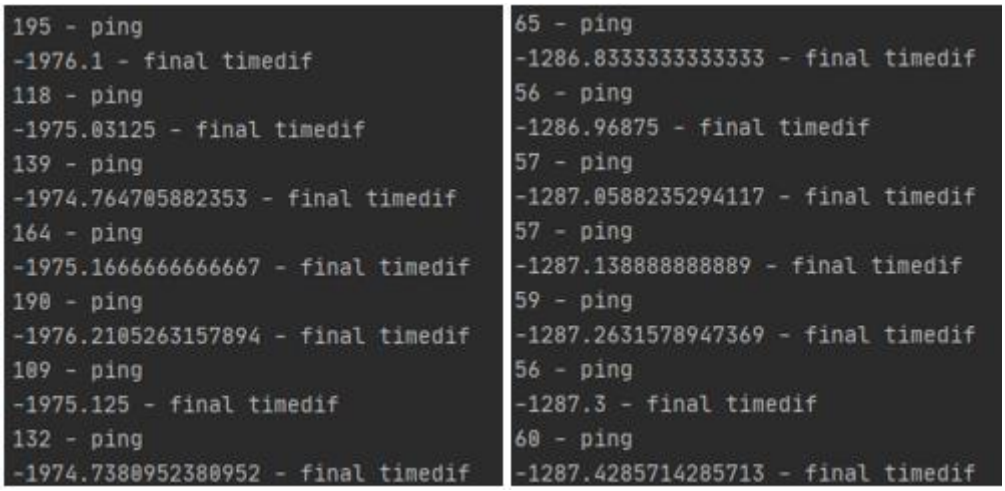

Рисунок 3.4 – Результати виводу функції знаходження орієнтовної різниці в часі між пристроями

Для тестування ефективності та дієздатності алгоритму було реалізовано вивід часу початку шляху та його закінчення на сервері та клієнті. Початок шляху на клієнті фіксується в момент початку обробки натискання клавіші гравцем, та позначається у консолі поміткою start. Початок руху на сервері фіксується з моменту отримання повідомлення від клієнта, та помічається відповідною поміткою start у консолі сервера. Кінець руху як на клієнті, так і на сервері, фіксується в момент останнього кроку для досягнення фінальної точки шляху, та помічається поміткою end. Для валідності результатів, до часових міток сервера було додано орієнтовну різницю в часі. Результати відображені на рисунку 3.5:

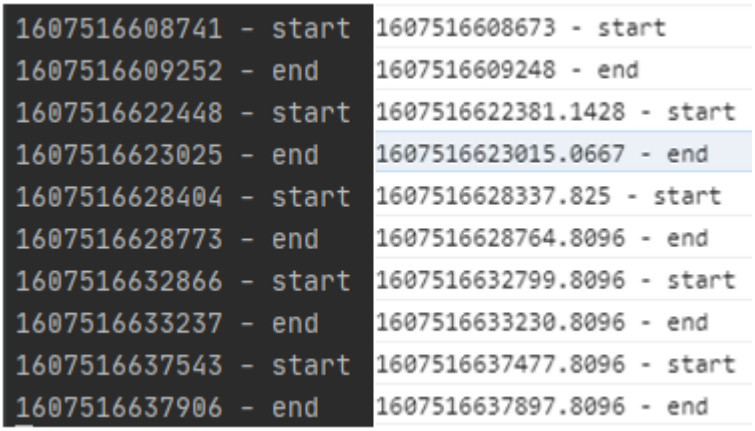

Рисунок 3.5 – Часові мітки сервера (справа) та клієнта (зліва), фіксуючі початок та кінець шляху

По отриманих в результаті тестування данних наглядно прослідковується ефект роботи алгоритму: початок руху гравця на сервері відбувається із певною затримкою відносно початку руху гравця на стороні клієнта. Ця затримка зумовлена затримкою відправки даних, пінгом. Однак момент досягнення кінцевої точки шляху відбувається із набагато меншою різницею у часі. Ця різниця не перевищує протяжність одного ігрового циклу, що задовольняє першопочаткові вимоги, та не повинно приводити до суттєвих розсинхронізацій.

Виходячи із результатів тестування, можна зробити висновок, що розроблена реалізація алгоритму вірно та коректно працює в умовах реальної онлайн гри. Отже, розробку можна вважати завершеною.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Безпека та захист здоров'я під час роботи з екранними пристроями

При розробці ПЗ використовуються екранні пристрої, а саме — монітори комп'ютерів та ноутбуків. Тому розробка проводилась згідно із вимогами, вказаними в наказі № 207 від 14.02.2018 про затвердження «Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» Верховної Ради України [15].

Згідно з розділом третім наказу:

– Робоче місце програміста має достатні розміри для зміни робочого положення та рухів;

– Усе випромінювання від екранних пристроїв зведено до рівня, який не спричиняє соматичних чи психічних розладів;

– Усі елементи робочого місця розташовані відповідно та задовільнять ергономічні, антропологічні, психофізіологічні вимоги, а також відповідають характеру виконуваних робіт;

– Освітлення робочого місця відповідає усім вимогам ДСанПІН 3.3.2.007-98 [16], та створює достатній контраст між навколишнім середовищем та екраном;

– Мікроклімат приміщення підтримується на постійному рівні, що відповідає вимогам санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [17];

– Робочий стіл достатніх розмірів для зручного розміщення екрана, клавіатури та комп'ютерної миші. Поверхня стола має низьку відбивну здатність.

– Робоче крісло є стійким та зручним, із можливістю регулювання по висоті, та нахилу спинки, що дозволяє легко рухатися, та займати зручне положення.

Також, згідно із розділом IV, на початку кожного робочого дня проводилась очистка екранів від пилу та інших забруднень, а по його завершенню – монітори відключались від електромережі.

Усі монітори повністю відповідали вимогам, вказаним у розділі V вищевказаного наказу, а саме:

- Не являлись та не створювали ризик для програміста;
- Кількість стороннього випромінювання не перевищувала допустиму норму;
- Не миготіли та забезпечували стабільне зображення;
- Не відблискували, та мінімально відбивали світло;
- Не виділяли надлишкового тепла під час роботи.

Згідно із пунктом 3 розділу V - шрифт, відстань, стиль, колір символів, як і колір фону були обрані такими, щоб текст був чітким та легким для сприйняття, і мінімально напружував очі.

Згідно із пунктом 12 розділу V - програмне забезпечення було підібране відповідно до поставлених завдань, та є простим у використанні та зручним в індивідуальному налаштуванні.

Згідно із карантинними заходами, уся розробка відбувалась дистанційно. Усі необхідні консультації та конференції проводились у онлайн режимі.

Розроблена система не несе загрози чи шкоди здоров’ю користувача, його психологічному чи психічному стану.

4.2 Охорона праці в умовах пандемії

У ряді положень Конвенції № 155 [17] та відповідної Рекомендації передбачені запобіжні та захисні заходи, спрямовані на пом’якшення негативних наслідків пандемії, такої як пандемія COVID-19, для безпеки та здоров’я у світі праці. Нижче наведено деякі з цих положень.

Роботодавці повинні забезпечувати, наскільки це обґрунтовано й практично здійсненно, щоб робочі місця, механізми, обладнання та процеси, котрі перебувають під їхнім контролем, були безпечними і не загрожували здоров'ю, щоб хімічні, біологічні й фізичні речовини та агенти, котрі перебувають під їхнім контролем, були безпечними для здоров'я, коли вживано відповідних захисних заходів. Роботодавці повинні, у разі потреби, надавати відповідні захисні одяг і засоби, щоб запобігти, наскільки це обґрунтовано й практично здійсненно, виникненню нещасних випадків чи шкідливих наслідків для здоров'я (Конвенція № 155, ст. 16).

Захисний одяг і захисні засоби слід надавати безплатно (Рекомендація № 164, п. 10(e)).

Роботодавці повинні, коли це потрібно, вживати заходів у разі виникнення аварійних ситуацій та нещасних випадків, зокрема належних заходів для надання першої допомоги (Конвенція № 155, ст. 18).

Роботодавці також повинні забезпечити консультування, інформування і підготовку працівників та їхніх представників із питань БЗР, пов'язаних з їхньою роботою (Конвенція № 155, ст. 19).

Працівники та їхні представники мають право отримувати належну інформацію та проходити належну підготовку з питань БЗР. Їм також слід створити можливості для розгляду всіх аспектів БЗР, пов'язаних з їхньою роботою, а роботодавці повинні консультувати їх щодо цих аспектів. Працівники також мають право залишити робоче місце, якщо вони мають достатні підстави вважати, що їхня робота становить безпосередню і серйозну небезпеку для їхнього життя чи здоров'я, причому без будь-яких необґрунтованих утисків (Конвенція № 155, ст. 13). У таких випадках працівники повинні повідомляти свого безпосереднього керівника про таку ситуацію; доти, доки роботодавець, у разі потреби, не вжив заходів для її усунення, він не може вимагати, щоб працівники повернулися на робоче місце, де зберігається безпосередня і серйозна небезпека для життя чи здоров'я (Конвенція № 155, ст. 19(f)). Працівники та їхні представники зобов'язані співпрацювати з роботодавцем у галузі БЗР (Конвенція № 155, ст. 19). Зокрема,

вони повинні: виявляти розумне піклування про свою власну безпеку та безпеку інших осіб, які можуть бути наражені на загрозу внаслідок їхньої дії або бездіяльності під час роботи;

дотримуватися інструкцій, передбачених для забезпечення їхньої власної безпеки та здоров'я, а також безпеки та здоров'я інших осіб; правильно користуватися запобіжними пристроями та захисними засобами і не виводити їх із ладу; негайно повідомляти свого безпосереднього керівника про будь-яку ситуацію, яка, на їхню думку, може становити небезпеку і яку вони не можуть самі усунути; повідомляти про будь-який нещасний випадок або випадок ушкодження здоров'я, що виник у ході роботи чи у зв'язку з нею (Рекомендація № 164, п. 16).

В останні десятиріччя як у промислово розвинених країнах, так і у країнах, що розвиваються, запроваджено методику на основі системи управління БЗР (СУ БЗР). Застосування цієї методики здійснюється на різних засадах: в одних країнах діють юридичні вимоги, згідно з якими вона повинна впроваджуватися на рівні підприємства, тоді як в інших її прийняття має добровільний характер. Досвід показує, що СУ БЗР є логічним і корисним інструментом для безперервного підвищення ефективності БЗР на організаційному рівні (ILO, 2011).

Як визначено у виданому МОП «Пораднику з систем управління безпекою і здоров'ям на роботі» (МОП-СУБЗР-2001), необхідно вжити відповідних заходів для створення СУ БЗР, яка має включати такі ключові елементи: політику, організацію планування і здійснення, оцінку та дії з удосконалення (ILO, 2001). Методика СУ БЗР забезпечує:

- реалізацію запобіжних і захисних заходів у ефективний та узгоджений спосіб;

- прийняття належної політики;

- взяття відповідних зобов'язань;

- урахування всіх елементів підприємства в оцінюванні небезпек і ризиків;

- залучення керівництва і працівників до вищезгаданого процесу на їхньому рівні відповідальності (ILO, 2011).

ВИСНОВКИ

У результаті дослідження було знайдене рішення проблеми довгої затримки відгуку ігрового стану на дії користувача у браузерній онлайн грі. Дана ціль досягнена шляхом поєднання та модифікації технологій, серед яких:

- Взаємодія ігрового клієнта із сервером із дублюванням ігрової логіки, з допомогою якої клієнт може миттєво реагувати на дії користувача, а сервер – провіряти та синхронізувати їх;
- Ігровий цикл із заданим часовим кроком, завдяки якому частота оновлення ігрового стану на сервері зрівняна із частотою його оновлення на кожному із підключених клієнтів. Також з його допомогою вирішена проблема неактивних вкладок;
- Технологія узгодження даних заднім ігровим циклом шляхом зберігання останніх ігрових циклів та дій гравців у відповідні масиви. Ново отримані дії гравців застосовуються безпосередньо до того ж ігрового стану, до якого вони були застосовані на клієнті, а активний ігровий цикл перераховується у відповідності до змін;
- Розроблена в процесі технологія знаходження орієнтовної різниці між показниками локальних годинників сервера та клієнта, заснована на принципі роботи NTP;
- Технологія інтерполяції для обробки та формування зображення між ігровими циклами, та формування проміжних ігрових станів в процесі згладжування при переході від прогнозу до виправленого ігрового стану.

У процесі роботи було розроблено робочу реалізацію алгоритму для браузерної онлайн гри «Overworld». На практиці алгоритм показав ефективність за рахунок надання можливості відображення результатів обробки ігрової логіки клієнтом без очікування результатів сервера. При цьому розсинхронізація завершення дій на усіх клієнтах та сервері не перевищує допустиму, а процес синхронізації непомітний гравцю.

ПЕРЕЛІК ПОСИЛАНЬ

1. Дуглас Крокфорд, Как устроен JavaScript – Питер, 2019 – 304 с.
2. Стандарт ECMA-262, ECMAScript® 2020 Language Specification, 11-е видання, червень 2020.
3. Современный учебник JavaScript, електронний посібник, URL: https://learn.javascript.ru/ (дата звернення: 11.09.2020).
4. Node.js v14.15.1 Documentation, програмний інтерфейс додатку, URL: https://nodejs.org/dist/latest-v14.x/docs/api/ (дата звернення: 17.09.2020)
5. WebSocket // «Ресурси для розробників, від розробників», MDN web docs, програмний інтерфейс додатку, URL: https://developer.mozilla.org/uk/docs/Web/API/WebSocket (дата звернення: 22.09.2020).
6. Introducing JSON, онлайн документація, URL: https://www.json.org/json-en.html (дата звернення: 25.09.2020).
7. PHASER, Desktop and Mobile HTML5 game framework, веб ресурс, URL: https://phaser.io/ (дата звернення: 27.09.2020).
8. Трофімов В.В., Інформаційні технології в економіці і управлінні // Системи колективного використання інформації, 2014, онлайн версія, URL: https://stud.com.ua/50110/informatika/informatsiyni_tehnologiyi_v_ekonomitsi_i_upravlinni (дата звернення: 29.09.2020).
9. Разработка игр на JavaScript, DOU: Сообщество программистов, веб стаття, URL: https://dou.ua/lenta/articles/javascript-gamedev/ (дата звернення : 02.10.2020).
10. Синхронизация состояний в многопользовательских играх, Хабр, веб стаття, URL: https://habr.com/ru/post/328702/ (дата звернення: 05.09.2020).
11. Протоколи синхронізації часу, Сервис точного времени, веб стаття, URL: http://time.in.ua/ua/ntp.html (дата звернення: 05.10.2020).

12. Интерполяция, экстраполяция и сглаживание, Кафедра Технологии воды и топлива НИУ МЭИ, веб стаття, URL: http://twt.mpei.ac.ru/ochkov/stat.html#_ftn1 (дата звернення: 07.10.2020).
13. Express – фреймворк веб приложений Node.js, веб ресурс, URL: https://expressjs.com/ru/ (дата звернення: 10.10.2020).
14. Міністерство соціальної політики україни, наказ № 207 від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», Законодавство україни, веб версія, URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text (дата звернення: 15.10.2020).
15. Міністерство охорони здоров'я україни, Головне санітарно-епідеміологічне управління, документ N 7 від 10.12.1998 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», веб версія, URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 15.10.2020).
16. Міністерство охорони здоров'я україни, постанова N 42 від 01.12.1999 «Санітарні норми мікроклімату виробничих приміщень», веб версія, URL: https://zakon.rada.gov.ua/rada/show/va042282-99#Text (дата звернення: 15.10.2020).
17. Конвенція 1981 року про безпеку й гігієну праці та виробниче середовище N 155, Документ 993_050, веб версія, URL: https://zakon.rada.gov.ua/laws/show/993_050#Text (дата звернення: 20.10.2020).
18. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – інженерія програмного забезпечення, міністерство освіти і науки україни, тернопільський національний технічний університет імені Івана Пулюя, від 26.08.2020 р.

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку проекту

«Розробка браузерної онлайн гри "Over world" на базі Java Script із використанням Node is та Phaser»

Розробники:
виконавець ст. гр. СПм-61
Верницький І. Р.

_____
(підпис)

керівник проекту
Пастух О. А.

_____
(підпис)

Тернопіль 2020

ЗМІСТ

–

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2020 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка браузерної онлайн гри "Over world" на базі Java Script із використанням Node is та Phaser».

2 ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Мета дипломної роботи - дослідження та вирішення проблем в процесі розробки браузерної багатокористувацької онлайн гри.

Предмет дослідження: затримка реакції ігрового клієнта на дії гравця. Пінг.

Мета роботи: розробка алгоритму синхронізації та узгодження, та його реалізація для браузерної гри.

Результатом роботи має бути робоча реалізація алгоритму синхронізації та узгодження, яка дозволятиме клієнту реагувати на дії гравця не очікуючи відповіді від сервера.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система повинна передбачати одну роль:

- Користувач

Програмне забезпечення має виконувати наступні дії:

– Реалізувати синхронізацію клієнтів та сервера;
– організовувати обмін повідомленнями між клієнтом та сервером;
– реалізувати ігрову логіку та її обробку;
– коректно відображати ігровий стан на клієнті;
– узгоджувати ігрові стані клієнтів та сервера;

3.2 Технічні вимоги

Данна система повинна працювати в середовищі будь-якого сучасного браузера. Необхідна стабільна робота, не залежно від величини затримки повідомлень між клієнтом та сервером. Система повинна забезпечувати захист ігрової логіки від несанкціонованого втручання.

3.3 Програмні вимоги

Програмний продукт повинен коректно функціонувати на мобільних пристроях та у браузерах. Розроблювана програмна система повинна бути пристосована для будь якої браузерної гри. Розробку виконувати з використанням мови Java Script, бібліотеки Phaser, та платформи Node.js.

4. ЕТАПИ РОЗРОБКИ

В ходів реалізації роботи проект повинен пройти крізь наступні стадії розробки:

- аналіз предметної області;
- вибір та модифікація технологій;
- проектування алгоритму;
- реалізація системи;
- тестування результатів розробки;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Пояснювальна записка;
- Технічне завдання;
- Презентаційний матеріал;
- Додатки.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	
Модифікація технологій	
Проектування алгоритму	
Використання системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9–10 грудня 2020 року

ТЕРНОПІЛЬ 2020

УДК 001 М34

ПРОГРАМНИЙ КОМІТЕТ

Голова: Лупенко Сергій Анатолійович – докт. техн. наук, професор.

Співголови: Марущак Павло Орестовищ – проректор з наукової роботи, докт. техн. наук, професор. Баран Ігор Олегович – канд. техн. наук, доцент, декан факультету ФІС.

Науковий секретар: Семенишин Галина Мирославівна – старший викладач.

Члени: докт. фіз.-мат. наук, професор В. Кривень; докт. техн. наук, професор М. Приймак; канд. техн. наук, доцент, Г. Осухівська; докт. техн. наук, професор М. Карпінський; канд. пед. наук, доцент Ж. Баб'як; докт. фіз.-мат. наук, професор М. Петрик; канд. техн. наук, доцент Н. Загородна.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Голова: Скоренький Юрій Любомирович – канд. техн. наук, доцент.

Члени: канд. екон. наук, доцент І. Струтинська; канд. техн. наук, доцент Я. Кінах; асистент М. Стадник; асистент Н. Шаблій; ст. викладач Л. Джиджора.

Матеріали VIII науково-технічної конфції «Інформаційні моделі, системи та
М34 технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 9 – 19грудня 2020р.). – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2020. – 196 с.

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001, тел. (0352) 52-41-33, факс (0352) 254983. E-mail: conffis2020@gmail.com Редагування, оформлення, верстка: Семенишин Г.М.

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання;
- Інформаційні системи та технології;
- Комп'ютерні системи та мережі;
- Програмна інженерія та моделювання складних розподілених систем;
- Новітні фізико-технічні та освітні технології.

В збірнику надруковано тези доповідей VIII науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 9–10 грудня 2020 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

© Тернопільський національний технічий університет імені Івана Пулюя, …… 2020

УДК 004.75

Верницький І. Р. студент
(Тернопільський національний технічний університет ім. І. Пулюя)

РЕАЛІЗАЦІЯ СИНХРОНІЗАЦІЇ ТА УЗГОДЖЕННЯ ДАННИХ У БРАУЗЕРНІЙ ГРІ

UDC 004.75

Vernytskyi I. R. Student

IMPLEMENTING DATA SYNCHRONIZATION AND RECONCILIATION IN A BROWSER GAME

В сучасному світі комп’ютерні ігри займають важливу роль у житті багатьох людей. Суть комп’ютерних ігор – отримання задоволення від процесу. Такий ефект досягається завдяки простій схемі – за певну дію, чи послідовність дій гравцю видається винагорода. В більшості ігор цей алгоритм складніший – дія повинна бути правильною, вчасною, комбінуватись з рядом інших, або навіть усе разом. В цій концепції швидкий відгук ігрового стану на дії гравця грає дуже велику роль. Деякі із жанрів ігор буквально зав’язані на правильному, а головне – вчасному натисканні клавіш. Будь то від погано оптимізованого ігрового процесу, чи великого пінгу до сервера, затримка може стати критичним чинником в таких іграх, та перетворити задоволення в цілий ряд негативних емоцій. Адже це руйнує основну концепцію – гравець зробив все правильно та вчасно, але через затримку не отримав нагороди. І якщо оптимізувавши ігровий код доволі просто досягнути непомітної для людського ока затримки – в випадку з пінгом в онлайн іграх ця задача стає в рази складнішою. Адже час проходження пакету від клієнта до сервера зазвичай не залежить від розробника. І поки сервер отримає дані про дію гравця – ця дія може бути уже не актуальною. Рішенням цієї проблеми є прогнозування та узгодження. Після дії гравця клієнт гри одразу відображає її, прогнозуючи поведінку ігрового стану. Паралельно клієнт відправляє дані про дію, свою реакцію, та часову мітку на сервер. Сервер тримає в пам’яті декілька десятків останніх ігрових станів. За допомогою часової мітки він визначає під час якого із станів дія була здійснена, повертається до нього, та провіряє правильність реакції клієнта. Проводиться перерахунок усіх ігрових циклів від моменту натискання гравцем клавіші до актуального на даний момент стану. Після цього сервер відправляє часову мітку, та актуальні дані усім підключеним клієнтам. Отримавши дані, опираючись на часову мітку, та ігрову логіку, клієнти розраховують актуальний ігровий стан, та з використанням інтерполяції переходять в нього.

Таким чином клієнт може майже миттєво реагувати на дії гравця, сервер не втрачає можливості провіряти інформацію, а на інші клієнти відображають найбільш актуальний ігровий стан. Особливо ефективно такий алгоритм показує себе в іграх, де дії гравця породжують інерційні зміни в ігрових станах, на приклад – клік по ігровому полю генерує шлях, по якому рухається персонаж до точки кліку. В такому випадку, хоч початок руху персонажа на різних клієнтах все ж відбудеться із затримкою – основна частина шляху, та прибуття в кінцеву точку будуть практично синхронними. Для браузерних ігор однією із проблем такого рішення є відсутність однієї часової шкали для всіх пристроїв. На кожному пристрої, включаючи сервер, час зазвичай відрізняється. Ця різниця може коливатись від декількох мілісекунд до декількох секунд. Таким чином часова мітка, створена клієнтом, буде не актуальною. Для вирішення цієї проблеми на початку ігрового процесу потрібно синхронізувати годинники. Сервер відправляє

низку пінг-понг запитів, а клієнт до кожного з них добавляє часову мітку отримання. Різниця часу визначається як усереднене значення усіх різниць часу між клієнтом і сервером з вирахуванням половини величини пінгу кожного запиту. **Література.**

1. Стаття What Every Programmer Needs To Know About Game Networking. URL: https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/.
2. Стаття On-line игры: взаимодействие с сервером https://gamedev.ru/code/articles/?id=4255
3. Стаття NTP. URL: https://ru.wikipedia.org/wiki/NTP.

ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ

Матеріали

тез доповідей

VIII науково-технічної конференції

9 – 10 грудня 2020 року

Комп’ютерне макетування та верстка *Г. М. Семенишин*

Формат 60х90/16. Обл. вид. арк. 13,35. Тираж 300 прим. Зам. № 7310.

Тернопільський національний технічний університет імені Івана Пулюя.
46001, м. Тернопіль, вул. Руська, 56.

Свідоцтво суб’єкта видавничої справи ДК № 4226 від 08.12.11.

ДОДАТОК В

Диск з программною систем