Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)

(рівень вищої освіти)

на тему: **Розроблення гри «Bubble Shooter» засобами Javascript**

Виконала: студентка 5 курсу, групи ІСТз-51
спеціальності
126 – "Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)

Маєр О.В.
(прізвище та ініціали)

Керівник Прусак Ю.В.
(прізвище та ініціали)

Рецензент Денюк М. В.
(прізвище та ініціали)

Львів – 2025 р.

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра інформаційних систем та комп'ютерного моделювання

Рівень вищої освіти перший (бакалаврський)

Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСКМ

_____ Сторожук О.Л.
"15" 11 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Маєр Оксані Володимирівній
(прізвище, ім'я, по батькові)

1. Тема роботи **Розроблення гри «Bubble Shooter» засобами Javascript**

керівник роботи Прусак Ю.В, канд. техн. наук, доцент.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 15.11. 2024 р. № С-886

2. Термін подання студентом роботи 05.04. 2025 р.

3. Вихідні дані до роботи:

- вивчити предметну область, проаналізувати існуючі логічні ігри;
- розглянути і використати алгоритми, які лежать в основі математичної моделі ігрового поля та ігрової динаміки;
- спроєктувати логічну гру з допомогою мови Javascript;
- представити результати роботи ігрового додатку.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Розділ 1. Стан проблемної області

Розділ 2. Інформаційне та математичне забезпечення

Розділ 3. Програмне та технічне забезпечення

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

додаток А, додаток Б

6. Дата видачі завдання 15 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних даних	15.11-30.12.2024	виконано
2	Розділ 1. Стан проблемної області	01.01-30.01.2025	виконано
3	Розділ 2. Інформаційне та математичне забезпечення	01.02-28.02.2025	виконано
4	Розділ 3. Програмне та технічне забезпечення	01.03-30.03.2025	виконано
5	Оформлення дипломної роботи	01.04-10.04.2025	виконано
6	Підготовка до захисту дипломної роботи, оформлення презентації	11.04-15.04. 2025	виконано

Студентка _____ Маєр О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Прусак Ю.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 50 сторінок пояснювальної записки, 4 рисунки, 2 таблиці, 15 джерел, 2 додатки.

У дипломній роботі розроблено браузерну гру Bubble Shooters з використанням мови програмування JavaScript. Основною метою було створення інтерактивного ігрового додатку, що включає механіку стрільби кульками та їх збирання в кластери. В ході роботи реалізовано графічний інтерфейс, який включає анімації руху кульок, управління гравцем за допомогою миші та відображення результатів гри. Застосовано алгоритми для визначення кластерів кульок, перевірки їх з'єднання та знищення. Робота дозволила дослідити основи розробки ігор на JavaScript та отримати навички створення ігор з використанням HTML5 Canvas та обробки подій користувача.

Ключові слова: *розроблення логічної гри, алгоритми зіткнень, графічний інтерфейс, ігрова логіка*, *Javascript*.

ABSTRACT

Diploma paper contains 50 pages of explanatory note, 4 figures, 2 tables, 15 sources, 2 appendix.

In my thesis, I developed a browser game Bubble Shooters using the JavaScript programming language. The main goal was to create an interactive gaming application that includes the mechanics of shooting balls and collecting them in clusters. In the course of the work, a graphical interface was implemented that includes animations of ball movement, player control with the mouse, and display of game results. Algorithms were used to determine the clusters of balls, check their connection and destruction. The work allowed us to explore the basics of JavaScript game development and gain skills in creating games using HTML5 Canvas and handling user events.

Keywords: *logic game development, collision algorithms, graphical interface, game logic, Javascript*.

ТЕХНІЧНЕ ЗАВДАННЯ

В дипломній роботі потрібно розробити логічну аркадну гру Booble Shooters, для цього потрібно вирішити такі завдання.

1. Розробити структуру ігрового поля, забезпечити можливість налаштування кількості рядів і колонок.

2. Розробити функціонал управління кулькою гравця, який дозволяє задавати напрямок пострілу та запускати її по визначеній траєкторії.

3. Реалізувати алгоритми визначення кластерів однакових за кольором кульок, що стикаються між собою та їх видалення з ігрового поля.

4. Розробити умови закінчення гри, коли кульки заповнюють ігрове поле або гравець втрачає можливість виконувати ходи.

5. Реалізувати логіку підрахунку очок за знищені кластери кульок, відображення поточного результату на екрані та збереження найкращого результату.

6. Розробити графічний інтерфейс користувача з використанням HTML5 Canvas, включаючи анімацію руху кульок, індикатора кута пострілу та відображення поточної і наступної кульки.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Adobe Flash – програмне забезпечення для створення та відтворення інтерактивного контенту, анімацію, відео та веб-ігри;

Babylon.js – ігровий 3D-рушій для розробки інтерактивної графіки та ігор у веб-броузері;

Canvas API – набір інтерфейсів в JavaScript, який дозволяє створювати та маніпулювати графікою;

CSS3 – мова каскадних таблиць стилів, яка використовується для опису зовнішнього вигляду та форматування вебсторінок;

CodePen – онлайн-платформа для створення, тестування та демонстрації фрагментів HTML, CSS та JavaScript-коду;

DOM (Document Object Model) – програмна модель для представлення HTML-документів у вигляді дерева об'єктів;

JavaScript – мова програмування, що використовується для створення інтерактивних елементів на веб-сторінках;

JSFiddle – онлайн-редактор коду для веб-розробки;

JSON (JavaScript Object Notation) – текстовий формат обміну даними, що використовується для зберігання та передачі структурованої інформації між сервером і клієнтом;

GUI (Graphical User Interface) – графічний інтерфейс користувача, який дозволяє взаємодіяти з комп’ютерними програмами через візуальні елементи;

HTML5 – п’ята версія мови розмітки гіпертексту, яка використовується для створення та структурування веб-сторінок, що підтримує мультимедійні елементи, інтерактивні функції;

Phaser.js – фреймворк для розробки 2D ігор на JavaScript;

Three.js – бібліотека JavaScript для створення 3D графіки в браузері;

XML (eXtensible Markup Language) – мова розмітки для зберігання та транспортування даних у структурованому вигляді;

WebGL API (Web Graphics Library) – JavaScript API, який дозволяє рендерити 3D графіку в броузері.

ВСТУП

Актуальність дипломної роботи

Розроблення ігрового додатку є актуальним через зростання попиту на веб-ігри, які не потребують встановлення додаткового програмного забезпечення. Ігри, такі як Bubble Shooters, мають популярність серед широкої аудиторії завдяки простоті механіки та цікавому ігровому процесу. Використання сучасних технологій, таких як JavaScript і HTML5 Canvas, дозволяє створювати ефективні, адаптивні та платформонезалежні ігрові рішення. Розробка подібних ігор сприяє вдосконаленню навичок програмування, алгоритмізації та роботи з графічними елементами.

В умовах постійного розвитку ринку мобільних та веб-ігор створення таких проектів може стати базою для комерційних рішень або подальшого вдосконалення. Ігровий додаток забезпечує інтерактивний досвід користувачів, дозволяючи їм відволіктися, відпочити та отримати задоволення від гри. Розробка ігор є хорошим прикладом інтеграції теоретичних знань з програмування в реальний проект. Подібні проекти сприяють популяризації веб-розробки як сучасної і перспективної галузі. Актуальність роботи також полягає у створенні навчального матеріалу, який може бути використаний іншими студентами чи викладачами для освоєння принципів створення веб-ігор.

Предмет дослідження – методи розробки інтерактивних броузерних ігор на основі JavaScript з використанням HTML5 Canvas для реалізації графічного інтерфейсу.

Об’єкт дослідження – процес створення гри Bubble Shooter як прикладу інтерактивної веб-ігри з використанням сучасних веб-технологій.

Мета роботи – розроблення функціонального ігрового додатку Bubble Shooter, який демонструє можливості сучасних технологій веб-розробки для створення ігор.

Завдання:

1. Розробити структуру ігрового поля, забезпечити можливість налаштування кількості рядів і колонок.

2. Розробити функціонал управління кулькою гравця, який дозволяє задавати напрямок пострілу та запускати її по визначеній траєкторії.

3. Реалізувати алгоритми визначення кластерів однакових за кольором кульок, що стикаються між собою, та їх видалення з ігрового поля.

4. Розробити умови закінчення гри, коли кульки заповнюють ігрове поле або гравець втрачає можливість виконувати ходи.

5. Реалізувати логіку підрахунку очок за знищені кластери кульок, відображення поточного результату на екрані та збереження найкращого результату.

6. Розробити графічний інтерфейс користувача з використанням HTML5 Canvas, включаючи анімацію руху кульок, індикатора кута пострілу та відображення поточної і наступної кульки.

Практичне значення одержаних результатів

Розроблений ігровий додаток Bubble Shooters може бути використаний як базовий приклад для навчання програмуванню з використанням JavaScript і HTML5 Canvas. Створений код гри демонструє інтеграцію графічних і алгоритмічних рішень, що може бути корисним для розробників-початківців. Додаток може бути адаптований для комерційного використання, як частина веб-розваг або мобільних платформ. Реалізовані механіки гри, такі як обробка колізій, виявлення кластерів і відстеження стану гри, можуть бути використані у схожих ігрових проектах. Результати роботи можуть бути основою для розробки більш складних ігор або інтерактивних додатків, що дозволяє масштабувати ідеї на нові рівні.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Розроблення логічних ігор засобами Javascript

Розроблення ігор є однією з найбільш творчих і технічно цікавих галузей програмування. JavaScript став популярною мовою для створення ігор завдяки своїй універсальності, широкій підтримці веб-броузерами та багатому інструментарію для розробників. Розглянемо особливості створення ігор засобами JavaScript, інструментів і підходів, які допомагають реалізувати ігровий процес.

JavaScript має низку переваг, які роблять його ідеальним вибором для створення ігор. Це міжплатформність: JavaScript працює у веб-броузерах на всіх сучасних платформах (Windows, Linux, Android). Це дозволяє створювати ігри, які запускаються без необхідності встановлення додаткового програмного забезпечення.

JavaScript має простий синтаксис і велику кількість навчальних матеріалів, що дозволяє швидко розпочати розроблення ігор. Завдяки бібліотекам і фреймворкам, таким як Phaser, Three.js або Babylon.js, можна швидко створювати прототипи ігор. JavaScript інтегрується з технологіями HTML5, включаючи Canvas API для 2D-графіки та WebGL для 3D-графіки [1].

Розглянемо, які є інструменти для створення ігор засобами JavaScript. Canvas API це вбудована технологія HTML5, яка дозволяє будувати графіку безпосередньо у веб-броузері. Підходить для створення 2D-ігор різного рівня складності.

Приклад використання Canvas API для створення ігрового об’єкта:

```
const canvas = document.getElementById('gameCanvas');
const context = canvas.getContext('2d');
context.fillStyle = 'red';
context.fillRect(50, 50, 100, 100);
```

WebGL API для відображення інтерактивної 3D-графіки працює на основі графічного процесора, що забезпечує високу продуктивність.

Бібліотеки та фреймворки. Phaser - популярний фреймворк для 2D-ігор, який пропонує зручні методи для роботи з анімаціями, фізикою та керуванням ігровими об’єктами. Three.js - бібліотека для створення 3D-графіки. Використовується для

складних візуальних ефектів. PixiJS - потужний інструмент для рендерингу 2D-графіки з високою продуктивністю [2].

Розробницькі середовища. Visual Studio Code - текстовий редактор із підтримкою розширень для JavaScript. CodePen та JSFiddle - онлайн-редактори для тестування JavaScript-коду. Розроблення гри зазвичай складається з кількох етапів. Постановка задачі та планування: визначення жанру гри, механіки, дизайну та аудиторії. Створення ігрового поля та об’єктів: відображення ігрового поля реалізується за допомогою Canvas API. Ігрові об’єкти створюються як JavaScript-класи:

```
class Bubble {
    constructor(x, y, color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }

    draw(context) {
        context.beginPath();
        context.arc(this.x, this.y, 10, 0, Math.PI * 2);
        context.fillStyle = this.color;
        context.fill();
    }
}
```

Логіка гри включає правила перемоги, взаємодії об’єктів, підрахунок очок. Для плавного руху та взаємодії об’єктів використовується запит анімації:

```
function animate() {
    context.clearRect(0, 0, canvas.width, canvas.height);
    // Логіка анімації
    requestAnimationFrame(animate);
}
animate();
```

Розроблення ігор засобами JavaScript є потужним підходом до створення інтерактивного контенту, який охоплює широке коло користувачів. Використання сучасних API та фреймворків спрощує реалізацію складних проектів, дозволяючи зосередитися на творчих аспектах гри. JavaScript є незамінним інструментом для розробників у середовищі веб-броузера [3].

Booble Shooters - це типова аркадна гра, де гравець керує гарматою, яка стріляє різнокольоровими кульками (бульбашками) по групі куль, розташованих у верхній частині екрану. Основною метою гри є влучати кульками в інші кульки одного кольору, аби зібрати групу з трьох або більше кульок однакового кольору, що призводить до їх зникнення.

Основні особливості гри. Графіка: кулі зазвичай мають різні кольори. Поле для гри - це сітка або контейнери, що складаються з кульок. Кулі падають, гравець повинен їх по черзі очищати. Механіка гри: гравець може стріляти кульками в напрямку групи куль, що рухаються або є нерухомими. Головна мета - очищати поле від кульок, збиваючи їх групами однакового кольору. Після того як кульки видаляються з екрана, нові кульки з’являються знову.

Гравець керує гарматою, яка переміщується по горизонталі і вибирає напрямок для пострілу. Щоб видалити кульки, необхідно влучити в інші кульки того ж кольору. Три або більше кульок одного кольору повинні бути разом. Існують рівні складності, коли кульки рухаються швидше або з’являються нові типи кульок. Рівні та прогрес: гра може мати різні рівні складності, на яких швидкість кульок та інші умови можуть змінюватися. Гравець може отримувати бонуси або бонусні рівні за певні досягнення.

Для гри використовується мишка для прицілювання та стрільби або клавіші для руху гармати. Для створення гри можна використовувати HTML5 canvas для побудови кульок, гармати та анімацій. Керування гарматою може бути за допомогою миші або клавіатури (клавіші для руху вліво/ вправо і для стрільби). Логіка зіткнень: перевірка на зіткнення між кульками (щоб визначити, чи утворилася група одного кольору). Анімація: створення плавної анімації кульок, коли вони рухаються по екрану або вибухають. Збереження прогресу: можна додати можливість зберігати і відновлювати прогрес на різних рівнях.

Термін HTML5 часто використовується для позначення збору різних технік, ці техніки зазвичай оцінюються за їх кінцевим ефектом, а не за технологією, яка їх створила. Шлях для HTML5 був прокладений Всесвітньою мережею консорціуму (W3C) за підтримки основних постачальників броузерів [4].

Ігри є скрізь, в них все частіше грають на підключених веб-пристроях та в середовищах настільних і мобільних броузерів. Оскільки броузерні ігри стають дедалі популярнішими, гравці звертаються до таких сайтів, як Facebook, щоб знайти прості, казуальні ігри, для яких не потрібно диска чи багато налаштувань для початку гри. Гра - це просто ще одне посилання, на яке можна натискати. Протягом останнього десятиліття покращення плагіна Adobe Flash сприяли зростанню веб-броузера як платформи для ігор. Більшість броузерів підтримували Flash, надаючи розробникам ігор доступ до потужної платформи. Однак до нещодавнього часу використання HTML та JavaScript як платформи для ігор відставало від Flash через обмеження в графіці, звуці та швидкості. Однак броузери та мобільні платформи для ігор значно покращились, і ситуація змінюється [5]. Мобільні операційні системи відмовились від Flash як підтримуваного плагіна, в результаті розробники ігор потребують інструментів, які забезпечують подібну продуктивність та гнучкість, зберігаючи при цьому універсальність, якою володів Flash.

Броузери зазнали швидких покращень у графічних та звукових можливостях за останні кілька років. Зростання можливостей HTML відображає зростаючий попит на платформу, яка забезпечує багаті ігрові враження та має підтримку кількох постачальників платформ. Добре підтримувана, відкрита платформа вважається менш схильною до комерційних обмежень та менталітету закритих систем, HTML5 є такою платформою.

Багато розробників ігор звертаються до HTML5, намагаючись створити ті самі типи ігор, які вони б створювали у Flash. HTML5 є одним із найкращих варіантів: він має велику аудиторію користувачів, і HTML5 та Flash мають багато подібних можливостей та обмежень. Крім того, HTML5 все ще знаходиться на відносно ранній стадії розвитку. Платформа розвивається швидко, іноді важко встигати за тим, які нові можливості підтримуються з місяця в місяць.

Так само, як і при створенні веб-застосунку, ключ до створення успішної гри - це розуміння можливостей і обмежень платформи. Потрібно проектувати та розробляти ігри, які максимально використовують потенціал платформи, одночасно уникаючи або мінімізуючи її обмеження.

HTML5 ігри можуть працювати в будь-якому сучасному броузері, незалежно від операційної системи чи пристрою. Це означає, що гра буде доступна на десктопах, планшетах і мобільних телефонах без необхідності створювати окремі версії для кожної платформи. HTML5 не потребує додаткових плагінів, таких як Flash або Java, що знижує бар'єри для користувачів і розробників. Це робить гру більш доступною і зручною для гравців, оскільки немає необхідності завантажувати додаткове програмне забезпечення [6].

HTML5 надає потужні можливості для роботи з графікою, відео та аудіо. Завдяки таким технологіям, як canvas, CSS3 і WebGL можна створювати складні графічні елементи, анімації та ефекти без необхідності в сторонніх плагінах. HTML5 дозволяє інтегрувати JavaScript для програмної логіки, CSS для стилізації і анімацій, а також інші сучасні веб-технології. Це забезпечує високу гнучкість і можливість використовувати широкий спектр інструментів для розробки. Ігри на HTML5 можуть працювати на будь-яких пристроях, де є броузер, без необхідності в окремих версіях для мобільних і настільних платформ. Це дозволяє охопити широку аудиторію гравців. Оскільки HTML5 ігри не потребують специфічних інструментів або ресурсів, розробка стає дешевшою і швидшою, ніж створення мобільних ігор для кожної операційної системи окремо. HTML5 має велику спільноту розробників і безліч ресурсів для навчання, що дозволяє швидко освоїти нові інструменти і технології. Це також означає більше готових рішень і бібліотек, які можна використати для розробки [7]. Ігри на HTML5 можна легко монетизувати через рекламу, мікроплатежі або за допомогою платних версій гри, оскільки веб-технології дозволяють інтегрувати ці стратегії без зайвих ускладнень.

HTML5 - це потужна, універсальна платформа для створення ігор, що надає багато переваг для розробників, дозволяючи створювати ігри, які доступні на багатьох пристроях і операційних системах. Веб-розробники, які мають навички роботи з JavaScript та CSS, будуть почуватися більш впевнено, вступаючи в розробку ігор на платформі HTML5. Розгортання файлів HTML і JavaScript також є звичним процесом, можна створювати онлайн-компоненти, використовуючи серверні мови програмування, які перетинаються з веб-розробкою.

Тим не менш, ігри, написані на HTML5 та JavaScript, мають дуже хороші шанси працювати з мінімальними змінами на різних операційних системах. Це дозволяє здійснювати одночасні релізи та мати єдину команду розробників, а не окрему команду для кожної системи. Можна писати код і тестувати його в настільному броузері, навіть якщо фінальне середовище буде іншим [8].

HTML5 постійно і швидко покращується. Швидкість обробки JavaScript також зростає, складні інтерпретатори наближаються до рідної швидкості виконання деяких операцій. З огляду на збільшення швидкості процесорів за останні 10 років, ігри, написані на JavaScript, можуть показувати кращу продуктивність, ніж багато з тих, що були написані на рідному коді лише кілька років тому. Завдяки зусиллям постачальників броузерів та виробників апаратного забезпечення ця траєкторія покращення продовжуватиметься, HTML5 зростає як життєздатна ігрова платформа.

Ці ігри зазвичай двовимірні, для одного гравця та з відносно короткими ігровими циклами. Досягнення в галузі 3D можливостей, таких як WebGL, означають, що великі, складні, захоплюючі багатокористувацькі ігри можуть бути реалізовані вже зараз або найближчим часом, але проєкт казуальної гри - це більш природне місце для старту розробника ігор. Простота проєктів також полегшує ілюстрацію основних принципів, що беруть участь у створенні гри.

1.2. Створення ігрового додатка на JavaScript

Ігровий цикл має справу з динамічними аспектами гри. Багато чого відбувається під час гри. Є також спеціальні ефекти, такі як вибухи, звуки і багато іншого. Усі ці різні завдання, які повинні вирішити ігровий цикл, можна розділити на дві категорії:

- задачі, які пов’язані з оновленням і підтриманням ігрового світу;
- задачі, які пов’язані з відображенням ігрового світу гравцеві.

Ігровий цикл постійно виконує ці завдання одну за іншою (рис. 1.1). У якості прикладу можна подивитися, як можна обробити навігацію користувача в простій грі, такій як Pac-Man. Ігровий світ в основному складається з лабіринту, по якому переміщується кілька привидів. Пакман знаходиться в цьому лабіринті і рухається у

визначеному напрямку. У першій задачі (оновлення та обслуговування ігрового світу) перевіряють, чи натиснута клавішу гравця зі стрілкою. Якщо це так, тоді потрібно оновити положення у відповідності з напрямком, у якому гравець хоче, щоб він рухався [9]. Потрібно перевірити, чи була ця остання точка на рівні, тому що це означало, щоб гравець закінчив рівень. Потім потрібно оновити інший ігровий світ.

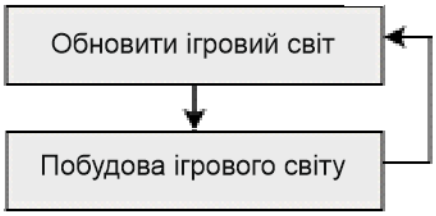

Рисунок 1.1 – Ігровий цикл постійно оновлюється, потім будує ігровий світ.

Другий набір завдань пов’язаний з відображенням ігрового світу гравця. У разі гри Pac-Man це означає малювання лабіринту, інформацію про гру, яку важливо знати гравцеві. Ця інформація може відображатися на різних ігрових екранах угорі чи внизу. Ця частина дисплея також називається проекційним дисплеєм (HUD). Сучасні 3D-ігри мають набагато більш складний набір завдань малювання. Ці ігри потрібні для роботи з освітленням і тінями.

В міру того, як ігри стали все більш складними, а операційні системи стали все більш незалежними від апаратного забезпечення, для ігрових компаній мав сенс почати повторне використання коду з більш ранніх ігор. Для чого написати абсолютно нову програму рендерингу або програму перевірки колізій кожної гри, якщо можна просто використовувати програму з раніше випущеної гри. Термін ігровий рух був придуманий в 1990-х роках, коли шутери, такі як Doom і Quake стали дуже популярним жанром. Ці ігри були настільки популярними, що їх виробник вирішив надати ліцензію на частину ігрового коду іншим ігровим компаніям у якості окремої програми. Перепродаж основного коду гри в якості ігрового рушія був прибутковим ділом, оскільки інші компанії були готові платити більші гроші за ліцензію на використання рушія у своїх іграх. Цим компаніям більше не потрібно було писати власний код гри з нуля - вони могли повторно використовувати програми, що містяться в ігровому рушії, більше зосередитися на

графічних моделях, персонажах, рівнях. Сьогодні доступно багато різноманітних ігрових рушіїв. Їх можна використовувати на різних платформах без зміни програм, які використовують код ігрового рушія [10]. Це особливо корисно для ігрових компаній, які хочуть опублікувати свої ігри на різних платформах.

Сучасні ігрові рушії надають розробникам ігор багато функціональних можливостей, таких як рушій 2D- і 3D-рендерингу, спеціальні ефекти, звук, анімація, штучний інтелект, сценарії та багато іншого. Ігрові рушії використовуються часто, тому що розробка всіх цих різних інструментів вимагає багато роботи.

Із-за такого розподілу основних ігрових функцій і самих ігор багато ігрових компаній наймають більше художників, ніж програмістів. Тим не менш програмісти все ще необхідні для покращення коду ігрового рушія, а також для опису програм, які мають справу з речами, які не включені в ігровий рушій або специфічні для ігор. Крім того, ігрові компанії часто розробляють програмне забезпечення для підтримки розробки. ігор, таких як програми для редагування рівнів, розширення програмного забезпечення для 3D-моделювання для експорту моделей і анімацій у потрібному форматі, інструменти для прототипування.

Для JavaScript поки немає рушія, які всі використовують. Більшість людей програмують відносно прості ігри на JavaScript, щоб переконатися, що ігри працюють на різних пристроях, особливо на пристроях з обмеженими можливостями. Тому замість використання рушія програмісти пишуть гру напряму, використовуючи такі елементи HTML5, як канва.

У розробці ігор зазвичай використовуються два підходи [11]. Рисунок 1.1 ілюструє ці підходи: зовнішній охоплює внутрішній. Коли люди вперше починають програмувати, вони зазвичай відразу ж починають писати код, що призводить до циклу написання, тестування та внесення змін. Професійні програмісти наіпаки тратять значну кількість часу на проектування ще до того, як напишуть код.

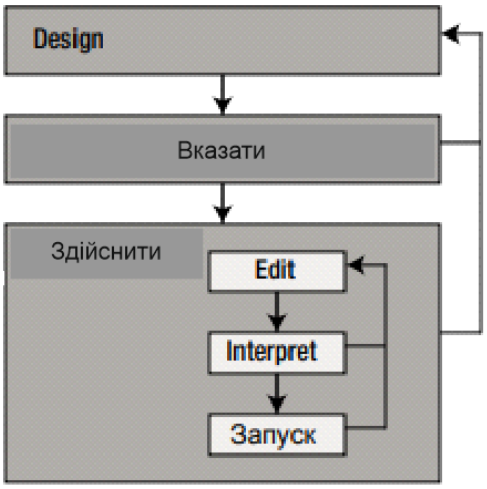

Рисунок 1.2 – Програмування в малому масштабі і в більшому масштабі

Якщо потрібно створити гру на JavaScript, треба написати програму, що містить багато рядків інструкцій. За допомогою текстового редактора треба відредагувати сценарії. Після того, як створять ці інструкції, запускають броузер запускають програму. Коли все добре, броузер інтерпретує скрипт і виконує його.

Консольна програма - це тільки один приклад типу додатків. Іншим дуже поширеним типом є програма Windows. Така програма показує екран, що містить вікно, кнопки та інші частини графічного інтерфейсу користувача (GUI). Додатки цього типу часто керуються подіями: вони реагують на такі події, як натиснути кнопки або вибрати пункт меню.

Іншим типом додатків є додатки, що працюють на мобільному телефоні або планшеті. Просторність на екрані в таких додатках зазвичай обмежена, але доступні нові можливості взаємодії, таких як датчиків, що визначають орієнтацію пристроїв, сенсорний екран.

При розробці додатків досить складно написати програму, яка працює на всіх платформах. Створення додатків Windows сильно відрізняється від створення додатків. Повторне використання коду між різними типами додатків ускладнено. З цієї причини веб-додатки стають все більш популярними. У цьому випадку програма зберігається на сервері, користувач запускає програму у веб-броузері.

Оскільки JavaScript є процедурною мовою, інструкції можуть бути згруповані у функції. Функції передбачають дублювання коду, інструкції містяться тільки в

одному місці, програміст дозволяє виконувати ці інструкції, викликаючи одне ім’я. Групування інструкцій у функціях виконується за допомогою фігурних дужок{...}. Такий зібраний блок інструкцій називається body - тіло функції. Над тілом пишуть заголовок функції.

Заголовок міститьім'я функції. Ігровий цикл складається з двох частин: оновлення та відрисовка. Потім в цих функціях поміщають інструкції, які потрібно виконати, щоб оновити або створити ігровий світ.

Синтаксис мови програмування відноситься до формальних правил, які визначають, що є допустимою програмою. Семантика програми відноситься до її фактичного значення.

Синтаксичні діаграми допомагають візуалізувати правила мови програмування, такого як JavaScript. На рисунку 1.3 представлена синтаксична діаграма, яка показує, як визначити функцію в JavaScript.

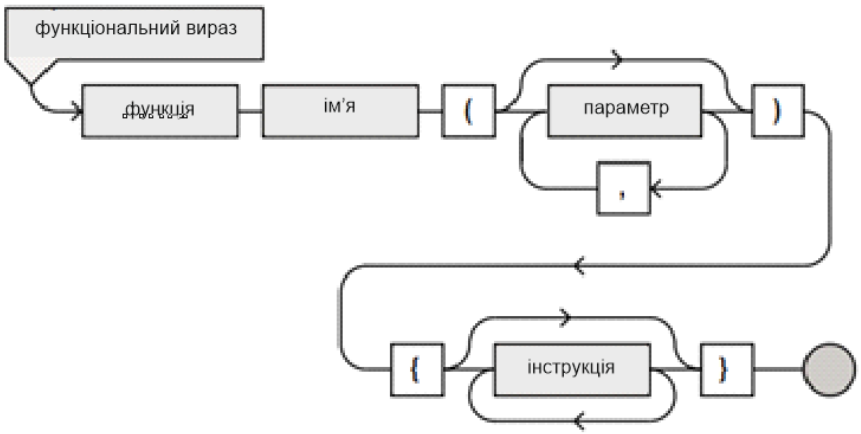

Рисунок 1.3 – Синтаксична діаграма функціонального виразу

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Розроблення ігор засобами Javascript

Інформаційне забезпечення є важливим аспектом розробки будь-якої гри, оскільки воно включає в себе структуру і організацію всіх даних, необхідних для правильного функціонування гри, таких як зображення, аудіофайли, налаштування гри, а також дані, що використовуються для взаємодії з користувачем. Розглянемо, які саме елементи інформаційного забезпечення є необхідними для реалізації гри Booble Shooters.

Гра Booble Shooters вимагає обробки та зберігання певних даних, таких як дані про стан гри. Це поточний рівень, кількість очок гравця, кількість бульбашок на полі, стан гри (активна чи завершена), час, що минув від початку гри.

Графічні ресурси включають в себе зображення для бульбашок (різних кольорів та типів), фонові зображення для ігрового поля, графічні елементи інтерфейсу (кнопки, панелі, елементи управління), анімації для руху бульбашок. Аудіо ресурси вкючають в себе звукові ефекти для пострілів, вибухів бульбашок, досягнення нових рівнів, музика на фоні гри. До конфігураційних налаштувань входять рівень складності, налаштування звуку та графіки, вибір мови інтерфейсу.

Для організації та зберігання даних гри можна використовувати різні методи, залежно від складності гри та вимог до зберігання великої кількості даних. Для Booble Shooters найбільш кращий підхід - це використання JSON або XML файлів для зберігання даних гри (статистики та налаштувань).

JSON файли для налаштувань гри є легким та зручним форматом для зберігання та передачі даних. Для зберігання налаштувань гри, таких як рівень, кількість очок, налаштування користувача, можна використовувати файл settings.json, який буде зчитуватися під час завантаження гри.

Приклад структури файлу settings.json:

```
{
  "level": 1,
  "score": 0,
  "sound": true,
```

```
  "language": "en"
}
```

Графічні зображення бульбашок та інших елементів гри зберігаються в окремій директорії в папці assets/images/. Вони включають різні варіації зображень для бульбашок (різних кольорів), а також фонові зображення та інші елементи інтерфейсу.

```
/assets
  /images
    bubble_red.png
    bubble_blue.png
    bubble_green.png
    background.jpg
  /audio
    shoot.mp3
    pop.mp3
    background_music.mp3
```

Звукові ефекти та музика зберігаються у папці assets/audio/. Вони використовуються для реалізації звукових ефектів (для пострілів, вибухів бульбашок) і фонової музики.

```
/assets
  /audio
    shoot.mp3
    pop.mp3
    background_music.mp3
```

Для ефективної взаємодії з користувачем необхідно організувати зручний інтерфейс гри, в якому будуть відображатися важливі дані, такі як поточний рівень, кількість очок, статистика. Це потребує регулярного оновлення елементів, таких як панель інтерфейсу, повідомлення про завершення гри та інші елементи, що повідомляють гравцеві про результати. Інтерфейс гри повинен показувати поточний рівень гравця та його кількість очок. Ці дані можуть оновлюватися в реальному часі за допомогою JavaScript, використовуючи DOM (Document Object Model) для маніпуляції елементами сторінки [7].

Приклад HTML елементів для відображення результатів:

```
<div id="score">Score: 0</div>
<div id="level">Level: 1</div>
```

Для підвищення інтерактивності гри можна реалізувати повідомлення про завершення рівня або гри, а також можливість відображати повідомлення про досягнення певних результатів (очок). Це можна зробити через модальні вікна або спливаючі елементи інтерфейсу.

Приклад HTML для повідомлення:

```
<div id="game-over-message" class="hidden">Game Over! Try again!</div>
```

У грі Booble Shooters важливо зберігати поточний стан гри, наприклад, кількість бульбашок на полі, поточний рівень, очки, а також виконувати обробку цих даних для визначення результату гри. Результати гри, такі як кількість набраних очок, рівень та інші статистичні дані зберігаються в об'єктах гри та можуть бути записані у файл або передані на сервер для збереження в базі даних. Для збереження статистики можна використати JavaScript API для роботи з локальним сховищем (localStorage) або за допомогою серверного коду з базою даних. За допомогою localStorage можна зберігати дані на стороні користувача для подальшого доступу до них. Наприклад, зберігання поточного рахунку гри в локальному сховищі:

```
localStorage.setItem('score', currentScore);
```

Інформаційне забезпечення гри Booble Shooters охоплює широкий спектр даних - від налаштувань гри та графічних ресурсів до аудіофайлів і статистики. Врахування таких аспектів, як оптимізація та безпека допомагає зробити гру зручною та безпечною для користувачів.

2.2. Математична модель гри Booble Shooters

Гра Booble Shooters є класичним прикладом аркадної гри, де гравець повинен запускати кольорові бульбашки на поле з іншими бульбашками, щоб сформувати групи з трьох або більше однакових за кольором бульбашок. Коли така група утворюється, бульбашки знищуються, гравець отримує бали.

Математичне забезпечення гри включає в себе кілька основних компонентів [8]:

- визначення траєкторії польоту бульбашки;
- колізії між бульбашками;

- створення і перевірка утворення груп бульбашок однакового кольору;
- обчислення балів і прогресу гри;
- пошук сиріт - бульбашок, які залишилися без підтримки після видалення груп.

Математичні моделі гри

Траєкторія польоту бульбашки. Коли гравець запускає бульбашку, необхідно розрахувати її траєкторію, яка залежить від кута запуску і початкової швидкості. Для спрощення можна використовувати класичні рівняння руху, враховуючи гравітацію та інші фактори, як-то сила удару.

Нехай x_0, y_0 - початкові координати бульбашки, v_0 - початкова швидкість, θ - кут запуску бульбашки, g - прискорення вільного падіння (якщо в грі використовується гравітація). Тоді координати бульбашки в будь-який момент часу t будуть обчислюватися за такими рівняннями:

$$\begin{aligned} x(t) &= x_0 + v_0 \cdot \cos(\theta) \cdot t \\ y(t) &= y_0 + v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} g t^2 \end{aligned} \tag{2.1}$$

Ці рівняння дозволяють обчислити траєкторію руху бульбашки до моменту, поки вона не досягне іншої бульбашки або стіни. Основною математичною задачею є визначення, чи відбулася колізія між бульбашкою, що летить, і бульбашками на полі. Для цього використовують геометричну модель, де кожна бульбашка є колом. Колізія між двома колами відбудеться, якщо відстань між їх центрами менша або рівна сумі їх радіусів [9]. Нехай r_1та r_2 - радіуси двох бульбашок, (x_1, y_1) та (x_2, y_2) - координати їх центрів. Тоді умова для колізії має вигляд:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq r_1 + r_2 \tag{2.2}$$

Якщо ця умова виконується, то бульбашка, що летить, потрапляє в контакт з іншою бульбашкою, тоді необхідно обчислити її нове місце розташування або визначити, чи утворюється група з трьох або більше однакових бульбашок.

Колізії в комп'ютерних іграх означають ситуацію, коли два або більше об'єктів (бульбашки в грі) взаємодіють одне з одним, через те, що їхні простори (області або координати на екрані) перетинаються або стикаються. Геометричні колізії - це

ситуації, коли об'єкти, що рухаються в грі, стикаються або перетинаються. У грі Booble Shooters, коли бульбашка, якою стріляє гравець, стикається з іншою бульбашкою на полі, це є колізією. Є такі типи колізій [10]:

- колізія з об'єктами: наприклад, якщо бульбашка летить і стикається з іншими бульбашками чи стінами;
- колізія між об'єктами на полі: кілька бульбашок можуть утворювати групу, коли вони потрапляють одна до одної після того, як гравець їх запустив;
- колізії з межами екрану: об'єкти можуть зіткнутися з межами ігрового поля або екрану, наприклад, коли бульбашка може досягти верхньої або нижньої частини екрану.
- алгоритм колізій: для перевірки колізій у багатьох іграх використовуються геометричні моделі, наприклад:
- перевірка колізії між колами: якщо бульбашки мають форму кола, то колізія між ними визначається як момент, коли відстань між їх центрами менша за суму їхніх радіусів;
- перевірка прямокутників або полігонів: для складніших об’єктів коли вони мають форму прямокутників або полігонів, можна використовувати алгоритми для перевірки, чи перекриваються їхні області;
- розв'язання колізій: коли колізія визначена, потрібно вирішити, що з нею робити - чи об’єкти повинні змінити своє положення, чи один із них повинен зникнути (бульбашки, що утворюють групи, зникають), чи має відбутися інша взаємодія (зміна шляху руху).

Колізії є важливою частиною механіки гри, оскільки вони визначають, як об'єкти взаємодіють між собою, і дозволяють створити реалістичні або цікаві ситуації в грі.

Перевірка групи бульбашок. Після кожного запуску бульбашки необхідно перевірити, чи утворюється група з трьох або більше однакових за кольором бульбашок. Для цього використовується алгоритм пошуку в ширину або глибину для пошуку всіх сусідніх бульбашок одного кольору.

Пошук сиріт**.** Після видалення бульбашок, що утворили групу, може з’явитися проблема сиріт - бульбашок, які залишилися висіти без опори [11]. Для виявлення таких сиріт використовують алгоритм, який перевіряє, чи можна з’єднати кожну бульбашку з верхньою частиною екрана. Якщо бульбашка не має з’єднання з верхом, її потрібно видалити.

Алгоритм гри можна описати наступними кроками:

- визначення траєкторії запуску бульбашки за допомогою обчислення кута та швидкості;
- перевірка на колізію з іншими бульбашками на полі;
- якщо колізія відбулася, перевірка на утворення групи з трьох або більше бульбашок;
- видалення утвореної групи та перевірка на сиріт.
- оновлення балів та перевірка умови закінчення гри (чи залишилися бульбашки);
- якщо рівень завершено, переходити до наступного рівня або завершення гри.

2.3. Математична модель оцінки результату гри

Результат гри оцінюється на основі кількості знищених бульбашок, швидкості завершення рівня та залишкових бульбашок на полі. Для кожного рівня можна використовувати наступну формулу для обчислення балів:

$$S = (N_{\text{popped}} \cdot 10) - (N_{\text{remaining}} \cdot 2) \tag{2.3}$$

де N_{popped} - кількість знищених бульбашок, $N_{\text{remaining}}$ - кількість бульбашок, що залишилися на полі.

Оцінка результату в грі Booble Shooters є важливим аспектом ігрової механіки, оскільки визначає ефективність гравця та впливає на подальший прогрес у грі. Основною метою є створення математичної моделі для підрахунку очок гравця, яка враховуватиме різні аспекти гри: поповнення бульбашок, формування комбінацій та знищення бульбашок.

Розглянемо основні фактори, які впливають на оцінку результату. Оцінка результату в грі Booble Shooters базується на кількох основних факторах. Попадання в ціль: кількість влучень гравця в інші бульбашки, що призводить до утворення групи з трьох або більше бульбашок одного кольору. Розмір утворених груп: якщо гравець збиває більше трьох бульбашок за один постріл, кількість очок має бути пропорційно більшою, ніж за одну стандартну групу. Видалення бульбашок з поля: за кожну бульбашку, що була видалена з поля (після утворення групи), гравець отримує певну кількість очок. Час виконання дій: час, який потрібен гравцеві для досягнення певного результату, також може впливати на оцінку. Швидкість, з якою гравець збиває бульбашки, дозволяє підвищити множник очок.

Розглянемо формули, на основі яких проводиться підрахунок очок в грі. Математична модель оцінки результату в Booble Shooters включає кілька важливих аспектів:

Оцінка за кожен постріл: за кожен постріл гравець отримує певну кількість очок залежно від того, скільки бульбашок він зіб'є в групу. Формула для підрахунку очок за один постріл має вигляд:

$$S_{\text{shot}} = C_{\text{match}} \times P_{\text{pop}} \tag{2.4}$$

де S_{shot} - очки за один постріл, C_{match} - кількість бульбашок, які утворюють групу одного кольору, P_{pop} - коефіцієнт, що залежить від величини утвореної групи. Наприклад, якщо група складається з трьох бульбашок, коефіцієнт може дорівнювати 1, якщо з чотирьох - 1.5, а з п'яти - 2. Із збільшенням кількості бульбашок в групі коефіцієнт множиться, що дозволяє гравцеві заробляти більше очок.

Оцінка за знищені бульбашки. Оскільки в грі Booble Shooters бульбашки зникають з поля після утворення групи, необхідно підрахувати очки за знищення кожної з них. Формула для цього етапу виглядає так:

$$S_{\text{destroy}} = N_{\text{destroyed}} \times P_{\text{destroy}} \tag{2.5}$$

де $S_{destroy}$ - очки за знищені бульбашки, $N_{destroyed}$ - кількість бульбашок, що були знищені, $P_{destroy}$ - коефіцієнт, що залежить від кольору або рівня бульбашок. Наприклад, бульбашки на вищих рівнях можуть мати більший коефіцієнт.

Час, необхідний для завершення рівня, може додавати додаткові очки. Бонус залежить від часу, який гравець витратив на виконання певних дій. Чим швидше завершено рівень, тим вищий бонус. Формула виглядає так:

$$S_{\text{bonus}} = \frac{T_{\text{max}} - T_{\text{spent}}}{T_{\text{max}}} \times P_{\text{bonus}} \tag{2.6}$$

де S_{bonus} - бонус за час, T_{max} - максимальний час для рівня, T_{spent} - час, витрачений гравцем для завершення рівня, P_{bonus} - множник бонусу, що визначається складністю рівня. Загальна оцінка за рівень в грі Booble Shooters розраховується як сума всіх етапів:

$$S_{\text{total}} = S_{\text{shot}} + S_{\text{destroy}} + S_{\text{bonus}} \tag{2.7}$$

Цей результат може бути збережений як оцінка рівня, а також використовуватися для порівняння з іншими гравцями чи для створення таблиці лідерів.

У грі передбачено кілька рівнів складності, які впливають на підрахунок очок. Чим складніший рівень, тим більше очок можна отримати за знищення бульбашок і за швидкість виконання. З кожним новим рівнем також змінюються кольори бульбашок, швидкість їх руху та складність влучання, що також збільшує потенціал для заробітку очок. Загальний бал, отриманий гравцем після завершення гри, може бути підсумковим результатом за всі рівні. Зокрема, підсумкова оцінка може враховувати не тільки поточний результат, але й бонуси за особливі досягнення, такі як набір найбільшої кількості балів на кожному рівні або швидкість проходження рівнів.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розроблення логічної аркади Booble Shooters

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <i class="./images/1.svg"></i>
    <title> Гра Bubble Shooter </title>
    <script type="text/javascript" src="1.js"> </script>
</head>
<body>
<canvas id="viewport" width="628" height="628"> </canvas>
</body>
</html>
```

Цей HTML-код є основою для гри Bubble Shooter. HTML забезпечує структуру сторінки. Тег <canvas> слугує основою для гри. Файл game.js містить логіку для анімацій, фізики, ігрових механік.

Оголошення структури HTML-документа:

```
<!DOCTYPE html>
<html lang="ua">
```

Це стандартний формат, що визначає документ як HTML5. Атрибут lang="ua" вказує мову контенту сторінки (українська). Метатеги встановлюють кодування символів на UTF-8, що дозволяє відображати різні мови, дозволяють сторінці коректно масштабуватися на різних пристроях (мобільних чи десктопних).

Відображають піктограму гармати через тег <img>:

```
<img src="./images/1.svg" alt="піктограма гармати">
```

Назва, яка відображається на вкладці броузера:

```
<title> Гра Bubble Shooter </title>
```

Підключення JavaScript-файлу:

```
<script type="text/javascript" src="game.js"></script>
```

Цей тег підключає зовнішній скрипт game.js, де знаходиться логіка гри. Файл має бути розміщений у тій самій директорії, що й HTML-документ. Полотно для гри:

```
<canvas id="viewport" width="628" height="628"> </canvas>
```

Тег <canvas> створює графічне полотно для візуалізації гри. id="viewport" - унікальний ідентифікатор для доступу до полотна через JavaScript. width=628 і height=628 задають розмір полотна в пікселях. У JavaScript цей <canvas> використовується для побудови кульок, гармати.

У додадку Б у файлі game.js представлено код гри мовою Javascript.

```
window.onload = function() {
    var canvas = document.getElementById("viewport");
    var context = canvas.getContext("2d");
    var lastframe = 0;
    var fpstime = 0;
    var framecount = 0;
    var fps = 0;
    var initialized = false;
```

window.onload = function() - вказують, що цей код виконається після того, як сторінка повністю завантажиться, включаючи HTML-структуру та пов’язані ресурси (зображення, стилі). Це забезпечує доступ до елементів DOM після їх ініціалізації.

var canvas = document.getElementById("viewport") - знаходить HTML-елемент <canvas> і зберігає посилання на нього в змінній canvas. Canvas є графічним полотном, де будуть будуватися елементи гри (кульки, фон).

var context = canvas.getContext("2d") - отримують контекст для малювання на canvas. Context надає методи для будування форм, ліній, зображень. Це основний інструмент для графічного відтворення гри.

var lastframe = 0 - відстежують час останнього оновлення кадру гри. Використовують для розрахунку інтервалу часу між кадрами, що потрібно для плавної анімації.

var fpstime = 0 - зберігають час, протягом якого рахувалися кадри, для обчислення FPS (кадрів на секунду).

var framecount = 0 - лічильник кадрів, які буде відтворено за певний проміжок часу. Разом із fpstime використовується для підрахунку FPS.

var fps = 0 - зберігають значення кадрів на секунду (Frames Per Second).

var initialized = false - флаг, який позначає, чи була гра повністю ініціалізована.

Цей блок коду виконує початкову ініціалізацію. Він завантажує полотно (canvas) для малювання, створює змінні для контролю часу, кадрів і продуктивності, налаштовує базові параметри для гри, щоб почати основну логіку

```
var level = {
    x: 4,
    y: 83,
    width: 0,
    height: 0,
    columns: 15,
    rows: 14,
    tilewidth: 40,
    tileheight: 40,
    rowheight: 34,
    radius: 20,
    tiles: []
};
```

Цей блок визначає об’єкт level, який містить налаштування та властивості рівня гри. Він визначає структуру ігрового рівня, включаючи його розташування, розміри, кількість кульок, їхні властивості, а також масив для зберігання ігрового поля. Ці дані є фундаментом для правильного малювання та роботи логіки гри.

```
var Tile = function(x, y, type, shift) {
    this.x = x;
    this.y = y;
    this.type = type;
    this.removed = false;
    this.shift = shift;
    this.velocity = 0;
    this.alpha = 1;
    this.processed = false;
};
```

Створюють клас Tile, який використовується для опису властивостей ігрових кульок у грі. Кожна кулька на ігровому полі є об'єктом цього класу.

```
var player = {
    x: 0,
    y: 0,
    angle: 0,
    tiletype: 0,
    bubble: {
                x: 0,
```

```
                y: 0,
                angle: 0,
                speed: 1000,
                dropspeed: 900,
                tiletype: 0,
                visible: false
            },
    nextbubble: {
                    x: 0,
                    y: 0,
                    tiletype: 0
                }
};
```

Цей блок коду визначає об’єкт player, який відповідає за управління діями гравця у грі. Гравець управляє запуском кульок у рівень, цей об’єкт містить усі необхідні властивості для контролю поточної та наступної кульки. Вказують позицію гармати (розташованої внизу екрана). x - горизонтальна координата гравця на ігровому полі, y - вертикальна координата гравця на ігровому полі. angle - кут, під яким буде запущена кулька, визначає напрямок руху кульки, коли гравець стріляє. Кут обчислюється залежно від позиції миші. tiletype - колір кульки, яку гравець запускає. bubble - описує поточну кульку, яку гравець запускає. x та y - координати поточної кульки, angle - кут, під яким кулька рухається.

speed - швидкість кульки під час запуску (пікселі за секунду). dropspeed - швидкість падіння кульки (якщо вона не досягла стелі). tiletype – колір поточної кульки. nextbubble - описує наступну кульку, яка буде в гарматі. x та y - координати, де відображається наступна кулька (біля гармати). tiletype - колір наступної кульки. Це попомагає гравцеві знати, який колір кульки буде доступний для наступного пострілу. Цей об'єкт управляє основною взаємодією гравця з грою - стрільбою кульками, їхньою фізикою та анімацією.

```
var bubblecolors = 7;
```

Змінна bubblecolors визначає кількість різних кольорів, які використовуються для кульок у грі. У цьому випадку їх кількість дорівнює **7**. Ця змінна задає механіку збігів. Гравцеві потрібно збирати кульки одного кольору разом, щоб вони зникли. Кількість кольорів безпосередньо впливає на складність гри: більше кольорів -

складніше формувати збіги, менше кольорів - гра стає простішою. При додаванні нових кульок на поле їхній колір обирається випадково з доступних кольорів. Змінюючи значення bubblecolors, можна регулювати складність гри. Якщо встановити bubblecolors = 3, то гра стане простішою, адже буде менше кольорів для збігів. Якщо збільшити до bubblecolors = 10, гра стане складнішою, оскільки важче знайти групи кульок одного кольору.

```
    var gamestates = { init: 0, ready: 1, shootbubble: 2, removecluster: 3, gameover:
4 };
    var gamestate = gamestates.init;
```

gamestates - об’єкт, який містить перелік можливих станів гри. Кожен стан гри позначений певним числовим значенням. init: 0 - початковий стан гри, коли гра ще не розпочалася або ініціалізується. ready: 1 - стан готовності, коли гравець може почати гру. shootbubble: 2 - стан, коли гравець може вистрілити кулькою. removecluster: 3 - стан, коли зібрана група кульок одного кольору видаляється. gameover: 4 - стан кінця гри, коли гра закінчена.

```
    var score = 0;
    var turncounter = 0;
    var rowoffset = 0;
```

score - змінна, яка зберігає поточний рахунок гри. Її значення збільшується в залежності від дій гравця, за успішні постріли кульками або за видалення груп кульок одного кольору. turncounter - змінна для підрахунку кількості ходів, які зробив гравець. Кожен постріл кульки або певна дія може збільшувати цей лічильник. rowoffset — змінна, яка відповідає за зміщення рядів у грі.

```
    var animationstate = 0;
    var animationtime = 0;
```

animationstate - змінна, яка зберігає поточний стан анімації. Вона використовується для відстеження різних етапів анімацій, що виконуються в грі, наприклад, анімації пострілу кульки, падіння кульок, вибухи або переходи між рівнями.

```
    var images = [];
    var bubbleimage;
```

images - масив, який містить усі необхідні зображення, що використовуються в грі. Це може включати зображення кульок, фону, інтерфейсні елементи, анімації.

Масив дозволяє зберігати всі графічні ресурси гри для подальшого їх використання. bubbleimage - змінна для зберігання зображення для кульки.

```
function loadImages(imagefiles) {
    loadcount = 0;
    loadtotal = imagefiles.length;
    preloaded = false;
```

Ця функція призначена для завантаження зображень, використовуючи масив файлів зображень.

```
function init() {
    images = loadImages(["./images/1.png"]);
    bubbleimage = images[0];
```

Функція init() виконується при ініціалізації гри. Вона відповідає за завантаження необхідних ресурсів (зображень) та налаштування деяких початкових значень.

```
canvas.addEventListener("mousemove", onMouseMove);
canvas.addEventListener("mousedown", onMouseDown);
```

Цей фрагмент коду додає слухачі подій для елементу canvas, що дозволяє взаємодіяти з грою через мишу.

```
player.x = level.x + level.width/2 - level.tilewidth/2;
player.y = level.y + level.height;
player.angle = 90;
player.tiletype = 0;
player.nextbubble.x = player.x - 2 * level.tilewidth;
player.nextbubble.y = player.y;
```

Визначають початкове розташування гравця та його властивості в грі, зокрема позицію на екрані, кут стрільби та наступну бульбашку. Встановлюється початковий тип бульбашки (якою гравець буде стріляти), а також її початковий кут. Для гравця задається позиція наступної бульбашки, яку він може запустити, це забезпечує постійне оновлення і підготовку нових бульбашок для стрільби.

```
    newGame();
    main(0);
}
```

Цей фрагмент коду виконується після ініціалізації гри, запуску всіх необхідних налаштувань, завантаження зображень і встановлення початкових параметрів. Тут починається фактична гра.

Функція main(tframe) є основною функцією циклу гри. Вона відповідає за оновлення стану гри, рендеринг кадрів і відображення на екрані. Ця функція постійно викликає саму себе, що дозволяє реалізувати безперервний ігровий процес з плавними анімаціями.

```
function update(tframe) {
    var dt = (tframe - lastframe) / 1000;
    lastframe = tframe;
    updateFps(dt);
    if (gamestate == gamestates.ready) {
    } else if (gamestate == gamestates.shootbubble) {
        stateShootBubble(dt);
    } else if (gamestate == gamestates.removecluster) {
        stateRemoveCluster(dt);
    }
}
```

Функція update(tframe) відповідає за оновлення стану гри. Вона отримує параметр tframe, який є часом поточного кадру, та використовує його для обчислення різниці в часі між поточним і попереднім кадром, що дозволяє розраховувати темп оновлення ігрових елементів. Функція update(tframe) є центральною для оновлення ігрового процесу. Вона обробляє час, оновлює FPS і в залежності від поточного ігрового стану виконує потрібні функції для просування гри вперед.

```
function setGameState(newgamestate) {
    gamestate = newgamestate;
    animationstate = 0;
    animationtime = 0;
}
```

Функція setGameState(newgamestate) призначена для зміни поточного ігрового стану гри. Вона приймає один параметр: newgamestate - новий ігровий стан, який необхідно встановити.

```
function stateShootBubble(dt) {
    player.bubble.x += dt * player.bubble.speed *
Math.cos(degToRad(player.bubble.angle));
    player.bubble.y += dt * player.bubble.speed * -
1*Math.sin(degToRad(player.bubble.angle));
    if (player.bubble.x <= level.x) {
        player.bubble.angle = 180 - player.bubble.angle;
```

```
        player.bubble.x = level.x;
    } else if (player.bubble.x + level.tilewidth >= level.x + level.width) {
        player.bubble.angle = 180 - player.bubble.angle;
        player.bubble.x = level.x + level.width - level.tilewidth;
    }
```

Функція stateShootBubble(dt) контролює рух бульбашки в грі, враховуючи швидкість і кут її руху. Вона також обробляє випадки, коли бульбашка стикається з межами екрану, відбиваючи її і коригуючи її координати для забезпечення безперервного руху. Ця функція відповідає за рух бульбашки після того, як вона була випущена гравцем. Вона виконується кожного кадру, коли бульбашка рухається в межах ігрового поля. Параметр dt представляє час, що пройшов між кадрами, і використовується для коректної анімації руху незалежно від частоти кадрів.

Якщо бульбашка виходить за ліву межу екрану, її x-координата коригується, щоб не перевищувати межі. Бульбашка також відбивається від стіни, змінюючи кут руху на протилежний. Функція виконує оновлення координат бульбашки в залежності від її швидкості і кута. Це дозволяє гравцю бачити, як бульбашка рухається по ігровому полю після того, як вона була випущена. Коли бульбашка досягає лівої або правої межі екрану, вона відбивається, що робить гру більш динамічною і додає реалізму руху бульбашки. Це забезпечує інтерактивний ігровий процес, де бульбашка постійно змінює своє положення і реагує на межі екрану, таким чином, додаючи більше варіантів для розв'язку гри (коректне відображення стін, які можуть відбивати кулю).

```
    if (player.bubble.y <= level.y) {
        player.bubble.y = level.y;
        snapBubble();
        return;
    }
```

Цей блок коду ефективно обробляє колізію бульбашки з верхньою межею рівня. Коли бульбашка досягає цієї межі, її координати коригуються, і вона фіксується на місці, готуючись до подальших взаємодій з іншими об'єктами або плитками. Колізія з верхньою межею гарантує, що бульбашка не може вийти за межі екрану в напрямку вгору. У грі це важливо для коректної обробки руху бульбашки,

оскільки вона повинна залишатися на рівні ігрового поля. Функція snapBubble()відповідає за закріплення бульбашки в певній точці, що може бути важливо для наступного етапу гри, наприклад, для того, щоб бульбашка потрапила до певного місця на рівні і взаємодіяла з іншими бульбашками.

Далі проводять перевірку колізій бульбашки з іншими бульбашками на рівні гри. Перевірка виконується за допомогою функції circleIntersection(), яка визначає, чи перетинаються два кола: одне - це бульбашка, а інше - це перешкода на рівні.

Функція stateRemoveCluster(dt) відповідає за обробку процесу видалення бульбашок після того, як бульбашка потрапила в рівень та з'єдналася з іншими бульбашками того ж кольору. Вона включає в себе анімацію видалення кластеру бульбашок, нарахування очок і перевірку на наявність плаваючих бульбашок, що згодом падають вниз.

Функція snapBubble() відповідає за фіксацію бульбашки на решітці гри після того, як вона рухається. Обчислюється центр бульбашки, з урахуванням її ширини та висоти. Викликається функція getGridPosition(), яка визначає, до яких координатів на решітці (ряд і стовпець) потрібно прив'язати бульбашку. Функція snapBubble() управляє логікою фіксації бульбашки на решітці, перевіркою на можливість додавання плитки, а також перевіркою на утворення кластерів, додавання нового ряду плиток після певної кількості ходів та перевіркою на завершення гри.

Функція addBubbles() відповідає за додавання нового ряду бульбашок до рівня після того, як гравець зробив кілька ходів. Функція addBubbles() переміщає бульбашки вниз на одну позицію. Вона додає новий ряд бульбашок на верхівку рівня, заповнений випадковими кольорами, вибраними з уже існуючих. Це забезпечує додавання нових бульбашок після певної кількості ходів або по мірі розвитку гри.

Функція findColors() визначає, які кольори плиток ще присутні на полі гри. Вона перевіряє всі бульбашки на рівні гри і знаходить унікальні кольори, що присутні на полі. Вона використовує масив colortable для того, щоб відслідковувати

вже знайдені кольори і додавати їх до списку foundcolors. Повертає масив з усіма кольорами, які є на полі в поточний момент гри.

Функція updateFps(dt) використовується для обчислення кадрів на секунду (FPS) в ігровому циклі. Вона працює шляхом підрахунку кількості кадрів, які були оброблені за певний проміжок часу, а потім оновлює значення FPS. Якщо час, протягом якого проводиться підрахунок кадрів, перевищує 0.25 секунди, це вказує на те, що настав час для обчислення FPS. FPS обчислюється як кількість кадрів, що були оброблені, поділена на кількість секунд, які пройшли.

Функція render() відповідає за відображення графіки на канвасі, зокрема малювання рівня, бульбашок, а також виведення балів. Вона є основною функцією для малювання і відображення графіки на канвасі. Вона відображає рамку рівня, бульбашки рівня, фон для інтерфейсу, поточний рахунок

```
function drawFrame() {
    context.fillStyle = "#e8eaec";
    context.fillRect(0, 0, canvas.width, canvas.height);
    context.fillStyle = "#303030";
    context.fillRect(0, 0, canvas.width, 79);
    context.fillStyle = "#ffffff";
    context.font = "24px Verdana";
    context.fillText("Bubble Shooter", 10, 37);
    context.fillStyle = "#ffffff";
    context.font = "12px Verdana";
    context.fillText("Fps: " + fps, 13, 57);
}
```

Функція drawFrame() малює фрейм або рамку для гри, а також відображає основні елементи інтерфейсу, такі як назва гри та інформація про кількість кадрів на секунду (FPS). Встановлюється колір фону світло-сірий, після чого весь екран заповнюється цим кольором. Встановлюється білий колір для тексту, виводиться назва гри Bubble Shooter. Текст розташовується в лівому верхньому куті. Встановлюється білий колір для тексту, виводиться поточний показник FPS (кількість кадрів на секунду). Текст розташовується трохи нижче заголовка гри.

```
function renderPlayer() {
    var centerx = player.x + level.tilewidth/2;
    var centery = player.y + level.tileheight/2;
    context.fillStyle = "#7a7a7a";
```

```
        context.beginPath();
        context.arc(centerx, centery, level.radius+12, 0, 2*Math.PI, false);
        context.fill();
        context.lineWidth = 2;
        context.strokeStyle = "#8c8c8c";
        context.stroke();
        context.lineWidth = 2;
        context.strokeStyle = "#0000ff";
        context.beginPath();
        context.moveTo(centerx, centery);
        context.lineTo(centerx + 1.5*level.tilewidth *
Math.cos(degToRad(player.angle)), centery - 1.5*level.tileheight *
Math.sin(degToRad(player.angle)));
        context.stroke();
        drawBubble(player.nextbubble.x, player.nextbubble.y,
player.nextbubble.tiletype);
        if (player.bubble.visible) {
            drawBubble(player.bubble.x, player.bubble.y, player.bubble.tiletype);
        }
    }
```

Функція renderPlayer() відповідає за побудову гравця та його елементів (бульбашки, яку він вистрілює) на екрані. Вона також малює приціл гравця та майбутню бульбашку, яку він планує відстріляти. Функція renderPlayer() малює графічні елементи, що відображають гравця та його поточний стан:

- приціл, що відображає напрямок пострілу;
- лінія прицілу, яка вказує на напрямок, в якому буде зроблений постріл;
- бульбашку, яку гравець збирається відстріляти, та поточну бульбашку, якщо вона видима.

```
    function getTileCoordinate(column, row) {
        var tilex = level.x + column * level.tilewidth;
        if ((row + rowoffset) % 2) {
            tilex += level.tilewidth/2;
        }
        var tiley = level.y + row * level.rowheight;
        return { tilex: tilex, tiley: tiley };
    }
```

Функція getTileCoordinate(column, row) обчислює координати верхнього лівого кута бульбашки в грі на основі її стовпця та рядка. Вона повертає об’єкт з

двома властивостями tilex та tiley, які є координатами верхнього лівого кута бульбашки на екрані.

```
function getGridPosition(x, y) {
    var gridy = Math.floor((y - level.y) / level.rowheight);
    var xoffset = 0;
    if ((gridy + rowoffset) % 2) {
        xoffset = level.tilewidth / 2;
    }
    var gridx = Math.floor(((x - xoffset) - level.x) / level.tilewidth);
    return { x: gridx, y: gridy };
}
```

Функція getGridPosition(x, y) визначає позицію сітки для заданих координат на екрані. Вона переводить екранні координати у пікселях в координати сітки, які використовуються для розташування бульбашок у грі. Це дозволяє точно визначити, в якому рядку та стовпці сітки знаходиться точка на екрані.

```
function drawBubble(x, y, index) {
    if (index < 0 || index >= bubblecolors)
        return;
    context.drawImage(bubbleimage, index * 40, 0, 40, 40, x, y, level.tilewidth,
level.tileheight);
}
```

Функція drawBubble(x, y, index) використовується для побудови бульбашки на екрані. Вона відображає зображення бульбашки на зазначеній позиції з заданим індексом кольору.

```
function newGame() {
    score = 0;
    turncounter = 0;
    rowoffset = 0;
    setGameState(gamestates.ready);
    createLevel();
    nextBubble();
    nextBubble();
}
```

Функція newGame() ініціалізує нову гру, скидаючи всі необхідні параметри та налаштовуючи початковий стан гри. Рахунок гри скидається до нуля (score = 0). Лічильник ходів turncounter скидається до нуля. Змінна rowoffset також скидається до нуля, щоб почати гру з початкового стану відображення рівня.

```
function createLevel() {
    for (var j=0; j<level.rows; j++) {
        var randomtile = randRange(0, bubblecolors-1);
        var count = 0;
        for (var i=0; i<level.columns; i++) {
            if (count >= 2) {
                var newtile = randRange(0, bubblecolors-1);
                if (newtile == randomtile) {
                    newtile = (newtile + 1) % bubblecolors;
                }
                randomtile = newtile;
                count = 0;
            }
            count++;
            if (j < level.rows/2) {
                level.tiles[i][j].type = randomtile;
            } else {
                level.tiles[i][j].type = -1;
            }
        }
    }
}
```

Функція createLevel() відповідає за створення нового рівня гри, заповнюючи плитки на полі відповідними типами бульбашок. Вона заповнює ігрове поле випадковими кольорами бульбашок у верхній половині рівня. Бульбашки на верхній частині рівня генеруються з обмеженням, щоб не було більше двох однакових бульбашок поспіль у стовпці.

```
function nextBubble() {
    player.tiletype = player.nextbubble.tiletype;
    player.bubble.tiletype = player.nextbubble.tiletype;
    player.bubble.x = player.x;
    player.bubble.y = player.y;
    player.bubble.visible = true;
    var nextcolor = getExistingColor();
    player.nextbubble.tiletype = nextcolor;
}
```

Функція nextBubble() відповідає за підготовку нової бульбашки для гравця, замінюючи поточну бульбашку на нову. Позиція нової бульбашки встановлюється в координати гравця (player.x, player.y), і бульбашка стає видимою на екрані.

Генерується колір для наступної бульбашки, викликаючи функцію getExistingColor(). Це дає новий випадковий колір для бульбашки.

```
function getExistingColor() {
    existingcolors = findColors();
    var bubbletype = 0;
    if (existingcolors.length > 0) {
        bubbletype = existingcolors[randRange(0, existingcolors.length-1)];
    }
    return bubbletype;
}
```

Функція getExistingColor() генерує випадковий колір для нової бульбашки, вибираючи один із вже існуючих кольорів, що використовуються в грі. Ця функція вибирає випадковий колір з тих, що вже є на полі гри і використовує його для нової бульбашки. Це дозволяє створювати нові бульбашки того ж типу, що вже є в грі.

```
function shootBubble() {
    player.bubble.x = player.x;
    player.bubble.y = player.y;
    player.bubble.angle = player.angle;
    player.bubble.tiletype = player.tiletype;
    setGameState(gamestates.shootbubble);
}
```

Функція shootBubble() відповідає за запуск кулі в грі. Вона виконує кілька кроків, щоб передати поточну інформацію про гравця та його кулю до стану стрільби. player.x і player.y визначають поточне положення гравця на екрані. Куля, що запускається, розташовується на цих координатах. player.angle зберігає кут, під яким гравець направляє свою кулю. Цей кут використовується для визначення напрямку руху кулі після її запуску. Функція shootBubble() запускає кулю, надаючи їй позицію, кут і тип, який був у гравця. Потім вона змінює стан гри, щоб почати процес руху кулі по екрану.

```
function circleIntersection(x1, y1, r1, x2, y2, r2) {
    var dx = x1 - x2;
    var dy = y1 - y2;
    var len = Math.sqrt(dx * dx + dy * dy);
    if (len < r1 + r2) {
        return true;
    }
    return false;
```

```
    }
```

Функція circleIntersection() перевіряє, чи перетинаються два кола в двовимірному просторі. Вона приймає координати центрів та радіуси двох кіл і повертає true, якщо кола перетинаються та false в іншому випадку.

Якщо відстань між центрами менша за суму радіусів двох кіл r1 + r2, то кола перетинаються (true). Якщо відстань більша або рівна цій сумі, кола не перетинаються, функція повертає false. Функція базується на геометрії кіл: два кола перетинаються, якщо відстань між їхніми центрами менша за суму їхніх радіусів.

```
    function onMouseMove(e) {
        var pos = getMousePos(canvas, e);
        var mouseangle = radToDeg(Math.atan2((player.y+level.tileheight/2) - pos.y,
pos.x - (player.x+level.tilewidth/2)));
        if (mouseangle < 0) {
            mouseangle = 180 + (180 + mouseangle);
        }
        var lbound = 8;
        var ubound = 172;
        if (mouseangle > 90 && mouseangle < 270) {
            if (mouseangle > ubound) {
                mouseangle = ubound;
            }
        } else {
            if (mouseangle < lbound || mouseangle >= 270) {
                mouseangle = lbound;
            }
        }
```

Функція onMouseMove() обробляє рух миші на канвасі. Вона визначає кут між мишкою та центром гравця, обмежує цей кут в певних межах і коригує його залежно від положення миші. getMousePos() отримує координати миші відносно канваса. Використано функцію Math.atan2() для знаходження кута між точкою на канвасі (положенням миші) та центром гравця. Функція atan2(y, x) повертає кут в радіанах між вектором від (0, 0) до (x, y) і віссю X. Після цього перетворюють отриманий кут в градуси за допомогою функції radToDeg(). Якщо кут менший за 0, коригують його, щоб він був в діапазоні від 0 до 360 градусів. Задають ліву і праву межі кута, в межах яких можна рухати мишу. Це необхідно для обмеження руху пускової гармати в певному діапазоні (щоб стріляти в певному секторі).

```
    function onMouseDown(e) {
        var pos = getMousePos(canvas, e);
        if (gamestate == gamestates.ready) {
            shootBubble();
        } else if (gamestate == gamestates.gameover) {
            newGame();
        }
    }
```

Функція onMouseDown() обробляє подію натискання кнопки миші. В залежності від поточного стану гри, вона виконує різні дії. Якщо гра знаходиться в стані готовий (гравець може почати гру), викликається функція shootBubble(), яка відповідає за постріл кулі. Якщо гра в стані гра закінчена, викликається функція newGame(), яка ініціює нову гру.

```
    function getMousePos(canvas, e) {
        var rect = canvas.getBoundingClientRect();
        return {
            x: Math.round((e.clientX - rect.left)/(rect.right -
rect.left)*canvas.width),
            y: Math.round((e.clientY - rect.top)/(rect.bottom -
rect.top)*canvas.height)
        };
    }
```

Функція getMousePos(canvas, e) використовується для визначення позиції миші на канвасі під час події миші (натискання або рух). e.clientX та e.clientY - координати миші відносно вікна браузера, передані в події миші. Для того, щоб отримати позицію миші на канвасі, потрібно перевести ці координати з системи координат вікна в систему координат канваса. Для цього враховуються розміри канваса та його позиція на екрані. Розраховуються відносні координати миші на канвасі у пікселях.

3.2. Результати роботи гри Bubble Shooters

Після запуску гра виглядає таким чином:

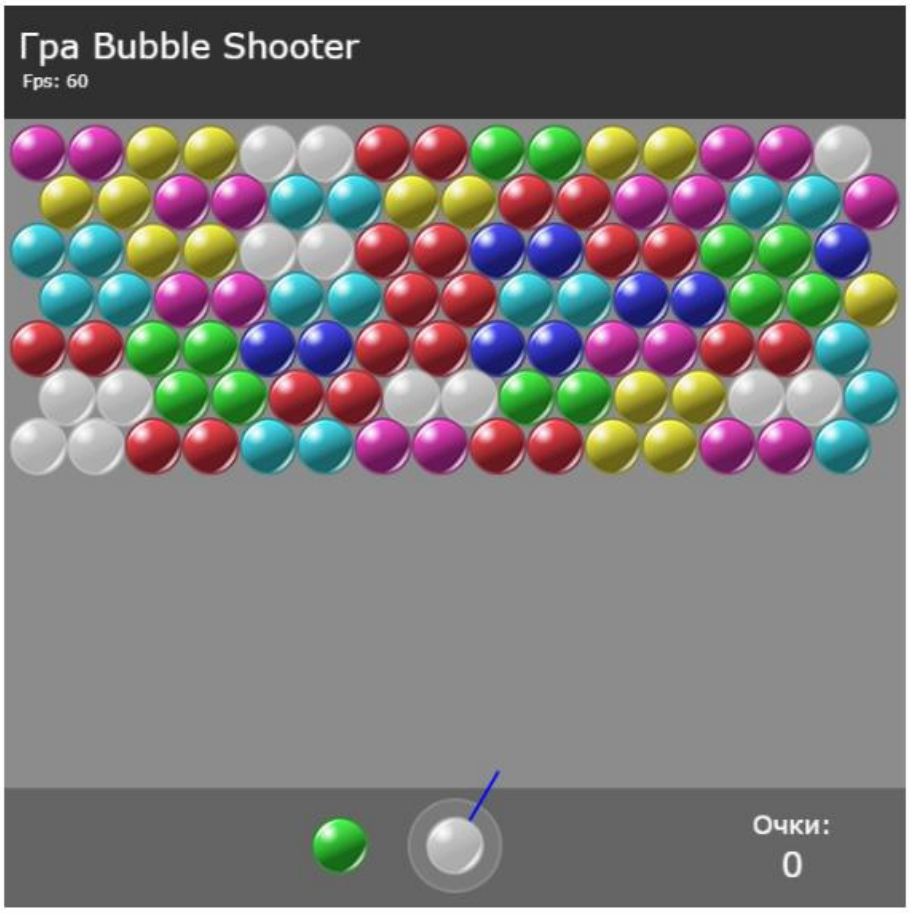

Рисунок 3.1 – Гра Bubble Shooters після запуску в броузері

Розглянемо основні елементи коду програми. Глобальні змінні:

- gamePlay – визначає, чи триває гра;
- dots – масив кульок на ігровому полі;
- booms – масив вибухів після знищення кульок;
- player – об’єкт, який представляє поточну кульку, якою грає гравець;
- nbrPlayer – лічильник кульок, запущених за рівень;
- countPoints – кількість очок, які набрав гравець.

Класи:

- DotS – базовий клас для кульок, зберігає координати (x, y), радіус (rad) і колірний індекс;

- Dot – розширює DotS для кульок на полі, містить методи для падіння (fall) і візуалізації (draw, update);
- Player – кулька, яку контролює гравець, містить логіку запуску кульки в напрямку миші або дотику;
- Boom – відповідає за ефекти вибуху кульок.

Функції:

- робота з Canvas – drawLineShoot відображає лінію прицілу для запуску кульки, drawLineEnd відображає лінію, яка позначає кінець гри.
- обробка подій – eventsListener обробляє події миші для управління прицілом і запуском кульок.
- механіка гри – checkColor перевіряє, які кульки одного кольору з’єднані і додає їх до масиву same; checkPlayer відповідає за логіку обробки запуску кульки, перевірки на зіткнення з іншими кульками та лінією кінця гри; checkDotsIsolate визначає, які кульки залишилися ізольованими.
- графічні ефекти – createEffet створює ефекти вибуху;
- аудіо – loadAudio завантажує звукові ефекти для кульок, вибухів і кінця гри;
- ініціалізація та анімація init налаштовує гру, створює стартові кульки; animate: основний цикл гри, який виконується через requestAnimationFrame.

Механіка кульок:

- кульки розміщуються в рядках із невеликим зсувом (newRow);
- коли гравець запускає кульку, вона рухається в напрямку, яка вказана гарматою.
- якщо кулька стикається з іншою, перевіряється збіг кольорів. Якщо знайдено три і більше кульки одного кольору, вони видаляються, а рахунок гравця збільшується.

Гра завершується, якщо кульки досягають нижньої межі екрану. Відображається повідомлення "Гра закінчена", після чого гра перезапускається. Встановлюється розмір Canvas, генеруються стартові кульки, реєструються обробники подій. Гравець керує прицілом, наводячи мишу. При натисканні миші кулька запускається у вибраному напрямку. Кульки взаємодіють, видаляючи збіги

та генеруючи вибухи. Після закінчення гри всі змінні скидаються, починається новий раунд.

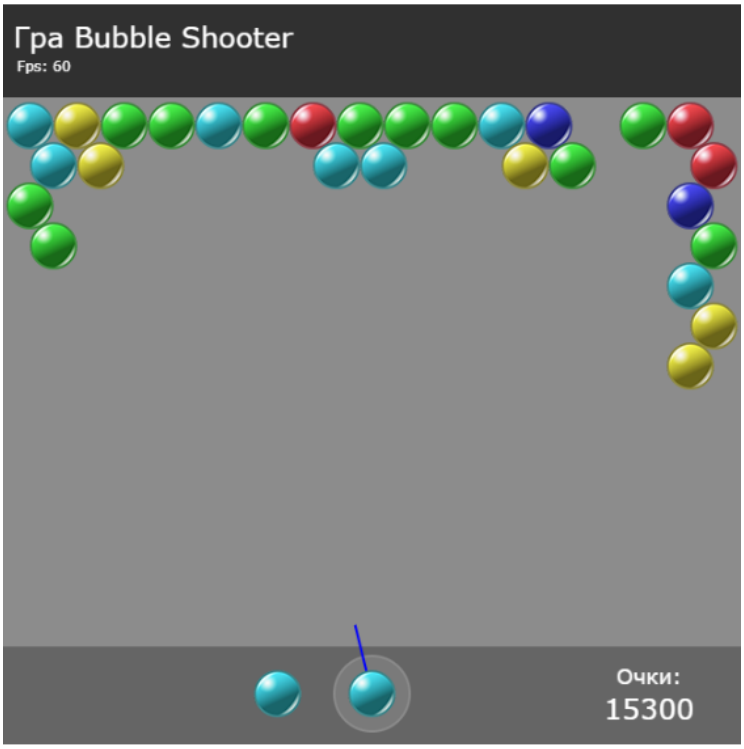

Рисунок. Стан гри на кінцевому етапі

Bubble Shooters - популярна логічна аркадна гра, основною метою якої є знищення кольорових бульбашок шляхом запуску кульок у групи однакових кольорів. Вона має просту механіку, проте захоплює гравців завдяки поєднанню стратегії, швидкості реакції та елементів головоломки.

Ігрове поле складається з сітки кольорових бульбашок, що заповнюють верхню частину екрана. Кольори бульбашок можуть бути різними, зазвичай 4-6 кольорів залежно від складності гри. Гравець керує прицілом, який розташований в нижній частині екрана, який стріляє кольоровими кульками. Напрямок пострілу задається за допомогою миші.

Бульбашки знищуються, коли утворюється група з 3 і більше бульбашок одного кольору. Якщо внаслідок знищення бульбашок інші кульки залишаються підвішеними (не торкаються інших), вони також падають і зникають з поля. Головна мета - повністю очистити ігрове поле від бульбашок. Також потрібно набрати максимальну кількість балів за певний час. У грі важливо враховувати баланс кольорів, щоб забезпечити можливість знищення груп бульбашок.

На початкових рівнях бульбашок мало, кольори прості для комбінування. Зі зростанням складності додаються нові кольори, бульбашки розташовуються складніше. Гравець може мати обмежену кількість кульок для запуску. Кульки відбиваються від стінок, що дозволяє виконувати складні траєкторії. Важливо враховувати кут пострілу та відбивання для досягнення важкодоступних місць.

За успішне знищення кульок гравець отримує бали. Час або кількість кульок для завершення рівня можуть бути обмежені. Гравець задає напрямок пострілу за допомогою миші або дотику. Напрямок розраховується на основі кута між прицілом і позицією курсора. При запуску кулька рухається в напрямку, доки не зіткнеться з іншою бульбашкою чи з межами ігрового поля. Після зіткнення перевіряється, чи є поруч група бульбашок одного кольору. Якщо така група існує, вона зникає. Якщо бульбашки залишаються без опори (не зв'язані з іншими), вони падають вниз і зникають.

В цій грі гравець керує пострілами куль, використовуючи мишу для вибору кута та напрямку стрільби. Після натискання на кнопку миші кулька відправляється у вибраному напрямку та зупиняється при зіткненні з іншими кульками. Якщо кулька потрапляє в групу куль однакового кольору, вони зникають, збільшуючи рахунок гравця.

Ігровий процес включає в себе кілька рівнів, кожен з яких має різну складність завдяки різним конфігураціям куль на ігровому полі. На кожному рівні кулі розташовуються в певному порядку, гравець повинен знищувати їх, перш ніж вони досягнуть нижньої частини екрану. За допомогою математичних алгоритмів у грі реалізовано пошук та видалення кластерів куль однакового кольору. Встановлено обмеження на кут, під яким можна стріляти кульки - гравець не може вільно вибирати будь-який кут, а лише в межах певного діапазону.

Ігрове поле складається з рівних розмірів клітин, кульки спочатку випадковим чином розподіляються по полю. Кожен рівень гри вимагає від гравця стратегії, оскільки для очищення поля необхідно об’єднувати кульки у великі групи. Якщо кульки не знищуються після кількох пострілів, рівень стає складнішим, кульки починають опускатися донизу. Гравець має обмежену кількість куль для стрільби,

тому необхідно ретельно планувати кожен постріл. Для кожного пострілу кулька вибирається випадковим чином з кольорів, що залишилися.

Гра має функцію підрахунку очок, що збільшуються кожного разу при знищенні куль. Для кожного рівня є певна кількість куль, які необхідно знищити для проходу до наступного рівня. Кожен наступний рівень має більше куль, що ускладнює гру. Гравець також може бачити на екрані кулю, яку він може стріляти наступною, що дозволяє йому планувати свої дії.

Якщо кулька не потрапляє в правильну позицію, вона залишається на місці, гравець може зробити ще один постріл. Гра відображає підсумки кожної гри, показуючи набрані очки гравця. В разі програшу, відображається екран Гра закінчена з можливістю почати нову гру.

Всі елементи гри, такі як кульки, рівні та підсумки, обробляються в реальному часі без затримок. Звукові ефекти, які супроводжують кожен постріл та знищення куль, значно покращують атмосферу гри. Графіка гри оптимізована, що дозволяє запускати її на більшості сучасних пристроїв. Всі кульки та ігрові елементи були розроблені з урахуванням розмірів екранів різних пристроїв, забезпечуючи зручний перегляд на десктопних пристроях.

Система зіткнень куль була реалізована таким чином, щоб кульки не проскакували через інші об'єкти, що робить гру більш реалістичною. Пристрій відображення куль і їх анімації створює ефект реалістичності, коли кульки рухаються та взаємодіють з іншими кульками. Гравець має можливість зберігати прогрес гри, що дозволяє йому продовжувати гру після перерви. Система рівнів дозволяє гравцю поступово збільшувати складність і отримувати нові виклики, тим самим підтримуючи інтерес до гри. Гравець може покращувати свої результати, прагнучи пройти більше рівнів та набрати більше очок у кожній новій грі.

3.3. Характеристики апаратного та програмного забезпечення

Для проектування логічної аркади Booble Shooters використовувався комп’ютер з такими характеристиками.

Таблиця 3.1 – Характеристики персонального комп’ютера

Процесор	Intel Pentium	Core i5
Материнська плата	Asus	ZY60-24-Premium
ОП	DDR3	4 GB
Відеокарта	Nvidia GeForce	1024 МГц
Жорсткий диск	Kingstone	300 Гб
Монітор	22″	Samsung

В дипломній роботі використано такі програмні продукти.

Таблиця 3.2 – Програмне забезпечення

Операційна система	Windows 11
Середовище розробки	HTML5, Javascript
Графічні редактори	CorelDraw X6, Adobe Photoshop CC
Текстовий редактор	Microsoft Word 2019

ВИСНОВКИ

У дипломній роботі було створено інтерактивну броузерну гру Bubble Shooter із використанням мови програмування JavaScript та технологій HTML5 і Canvas. Реалізовано механіку гри, яка включає стрільбу бульбашками, перевірку їх зіткнень та видалення кластерів однакового кольору. Забезпечено динамічне створення ігрового поля з налаштуванням рівнів складності, що підвищує інтерес до гри. Реалізовано функцію розпізнавання плаваючих кластерів, які видаляються після відокремлення від основного поля.

Гра адаптована до різних розмірів екранів завдяки використанню відносних координат та масштабування. Створено механізм підрахунку балів, який мотивує гравців покращувати свої результати. Запроваджено систему керування мишею для вибору напрямку стрільби, що забезпечує інтуїтивно зрозумілий ігровий процес. Додано елементи візуалізації, такі як анімації руху бульбашок і відображення наступної бульбашки, що підвищує візуальну привабливість гри.

Проведено тестування гри, яке підтвердило її стабільну роботу та відсутність критичних помилок. Розробка гри стала демонстрацією практичного використання алгоритмів таких як пошук кластерів. Впроваджено алгоритм визначення наявних кольорів на ігровому полі для забезпечення різноманітності під час генерації нових бульбашок. Гра може використовуватися як основа для подальшого вдосконалення або розширення функціоналу, наприклад, додавання нових рівнів чи спеціальних бонусів. Проект має освітню цінність, адже демонструє використання базових структур даних, алгоритмів та принципів побудови ігор. Отримані результати можуть бути використані для навчання студентів веб-розробці та створенню інтерактивних застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ishak S., Hasran U., Din R. Media education through digital games: A review on design and factors influencing learning performance. Active teaching and learning: educational trends and practices. 2023. V.2, № 13. P.102.

2. Surer E., Erkayaoglu M., Ozturk Z. Developing a scenario-based video game generation framework for computer and virtual reality environments: a comparative usability study. Multimodal User Interfaces. 2021. № 15. P. 393-411.

3. Berg B., Engström H., Hellkvist M. What empirical research tells us about game development. Computer game. 2019, №8. P.179-198.

4. Heeter C. Interactivity in educational games: A qualitative analysis of how students respond to game features. Educational Technology Research and Development. 2003. V.2, № 51. P.81-114.

5. Squire K., Jenkins H. Harnessing the power of games in education. Insight: scholarly teaching. 2008. V.1, № 3. P. 7-33.

6. Vlachopoulos D., Makri A. The effect of games and simulations on higher education: a systematic literature review. Educational technology. 2017. V.14, № 22.

7. Khalfallah H. Crafting clean code with JavaScript and React. Apress. 2024. 443 p.

8. Freeman E., Robson E. Head first JavaScript programming. O'Reilly. 2024. 662 p.

9. Simon M. JavaScript for web developers. Apress. 2023. 406 p.

10. Arjan E. Building JavaScript Games: for Phones, Tablets, and Desktop. Apress. 2014. 422 p.

11. Stuart G. Introducing Javascript game development. Apress. 2017. 211 p.

12. Brown E. Game Development with HTML5 Canvas. Packt Publishing. 2021. 75 p.

13. Gregory J. Game Engine Architecture. CRC Press. 2023. 150 p.

14. McShaffry M., Graham D. Game Coding Complete. CRC Press. 2022. 250 p.

15. Wexler J. Learning HTML5 Game Programming. Packt Publishing. 2022. 67 p.

ДОДАТКИ

ДОДАТОК А

game.html

```
<!DOCTYPE html>
<html lang="ua">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <i class="./images/1.svg"></i>
    <title> Гра Bubble Shooter </title>
    <script type="text/javascript" src="1.js"> </script>
</head>
<body>
<canvas id="viewport" width="628" height="628"> </canvas>
</body>
</html>
```

ДОДАТОК Б

game.js

```
window.onload = function() {
    var canvas = document.getElementById("viewport");
    var context = canvas.getContext("2d");

    var lastframe = 0;
    var fpstime = 0;
    var framecount = 0;
    var fps = 0;

    var initialized = false;

    var level = {
        x: 4,
        y: 83,
        width: 0,
        height: 0,
        columns: 15,
        rows: 14,
        tilewidth: 40,
        tileheight: 40,
        rowheight: 34,
        radius: 20,
        tiles: []
    };

    var Tile = function(x, y, type, shift) {
        this.x = x;
        this.y = y;
        this.type = type;
        this.removed = false;
        this.shift = shift;
        this.velocity = 0;
        this.alpha = 1;
        this.processed = false;
    };

    var player = {
        x: 0,
        y: 0,
```

```
        angle: 0,
        tiletype: 0,
        bubble: {
                    x: 0,
                    y: 0,
                    angle: 0,
                    speed: 1000,
                    dropspeed: 900,
                    tiletype: 0,
                    visible: false
                },
        nextbubble: {
                        x: 0,
                        y: 0,
                        tiletype: 0
                    }
    };

    var neighborsoffsets = [[[1, 0], [0, 1], [-1, 1], [-1, 0], [-1, -1], [0, -1]]
                            [[1, 0], [1, 1], [0, 1], [-1, 0], [0, -1], [1, -1]]]
    var bubblecolors = 7;

    var gamestates = { init: 0, ready: 1, shootbubble: 2, removecluster: 3, gameover:
4 };
    var gamestate = gamestates.init;

    var score = 0;

    var turncounter = 0;
    var rowoffset = 0;

    var animationstate = 0;
    var animationtime = 0;

    var showcluster = false;
    var cluster = [];
    var floatingclusters = [];

    var images = [];
    var bubbleimage;

    var loadcount = 0;
    var loadtotal = 0;
```

```
var preloaded = false;

function loadImages(imagefiles) {
    loadcount = 0;
    loadtotal = imagefiles.length;
    preloaded = false;

    var loadedimages = [];
    for (var i=0; i<imagefiles.length; i++) {
        var image = new Image();

        image.onload = function () {
            loadcount++;
            if (loadcount == loadtotal) {
                preloaded = true;
            }
        };

        image.src = imagefiles[i];
        loadedimages[i] = image;
    }
    return loadedimages;
}

function init() {
    images = loadImages(["./images/2.png"]);
    bubbleimage = images[0];

    canvas.addEventListener("mousemove", onMouseMove);
    canvas.addEventListener("mousedown", onMouseDown);

    for (var i=0; i<level.columns; i++) {
        level.tiles[i] = [];
        for (var j=0; j<level.rows; j++) {
            level.tiles[i][j] = new Tile(i, j, 0, 0);
        }
    }

    level.width = level.columns * level.tilewidth + level.tilewidth/2;
    level.height = (level.rows-1) * level.rowheight + level.tileheight;

    player.x = level.x + level.width/2 - level.tilewidth/2;
    player.y = level.y + level.height;
```

```
        player.angle = 90;
        player.tiletype = 0;

        player.nextbubble.x = player.x - 2 * level.tilewidth;
        player.nextbubble.y = player.y;
        newGame();
                main(0);
    }

    function main(tframe) {
        window.requestAnimationFrame(main);
        if (!initialized) {
            context.clearRect(0, 0, canvas.width, canvas.height);
            drawFrame();
            var loadpercentage = loadcount/loadtotal;
            context.strokeStyle = "#ff8080";
            context.lineWidth=3;
            context.strokeRect(18.5, 0.5 + canvas.height - 51, canvas.width-37, 32);
            context.fillStyle = "#ff8080";
            context.fillRect(18.5, 0.5 + canvas.height - 51,
loadpercentage*(canvas.width-37), 32);
            var loadtext = "Loaded " + loadcount + "/" + loadtotal + " images";
            context.fillStyle = "#000000";
            context.font = "16px Verdana";
            context.fillText(loadtext, 18, 0.5 + canvas.height - 63);
            if (preloaded) {
                setTimeout(function(){initialized = true;}, 1000);
            }
        } else {
            update(tframe);
            render();
        }
    }

    function update(tframe) {
        var dt = (tframe - lastframe) / 1000;
        lastframe = tframe;
        updateFps(dt);
        if (gamestate == gamestates.ready) {
        } else if (gamestate == gamestates.shootbubble) {
            stateShootBubble(dt);
        } else if (gamestate == gamestates.removecluster) {
            stateRemoveCluster(dt);
```

```
        }
    }

    function setGameState(newgamestate) {
        gamestate = newgamestate;
        animationstate = 0;
        animationtime = 0;
    }

    function stateShootBubble(dt) {
        player.bubble.x += dt * player.bubble.speed *
Math.cos(degToRad(player.bubble.angle));
        player.bubble.y += dt * player.bubble.speed * -
1*Math.sin(degToRad(player.bubble.angle));
        if (player.bubble.x <= level.x) {
            player.bubble.angle = 180 - player.bubble.angle;
            player.bubble.x = level.x;
        } else if (player.bubble.x + level.tilewidth >= level.x + level.width) {
            // Right edge
            player.bubble.angle = 180 - player.bubble.angle;
            player.bubble.x = level.x + level.width - level.tilewidth;
        }
        if (player.bubble.y <= level.y) {
            // Top collision
            player.bubble.y = level.y;
            snapBubble();
            return;
        }
        for (var i=0; i<level.columns; i++) {
            for (var j=0; j<level.rows; j++) {
                var tile = level.tiles[i][j];
                if (tile.type < 0) {
                    continue;
                }
                var coord = getTileCoordinate(i, j);
                if (circleIntersection(player.bubble.x + level.tilewidth/2,
                                       player.bubble.y + level.tileheight/2,
                                       level.radius,
                                       coord.tilex + level.tilewidth/2,
                                       coord.tiley + level.tileheight/2,
                                       level.radius)) {
                    snapBubble();
                    return;
```

```
            }
        }
    }
}

function stateRemoveCluster(dt) {
    if (animationstate == 0) {
        resetRemoved();
        for (var i=0; i<cluster.length; i++) {
            // Set the removed flag
            cluster[i].removed = true;
        }
        score += cluster.length * 100;
        floatingclusters = findFloatingClusters();
        if (floatingclusters.length > 0) {
            for (var i=0; i<floatingclusters.length; i++) {
                for (var j=0; j<floatingclusters[i].length; j++) {
                    var tile = floatingclusters[i][j];
                    tile.shift = 0;
                    tile.shift = 1;
                    tile.velocity = player.bubble.dropspeed;
                    score += 100;
                }
            }
        }

        animationstate = 1;
    }

    if (animationstate == 1) {
        var tilesleft = false;
        for (var i=0; i<cluster.length; i++) {
            var tile = cluster[i];
            if (tile.type >= 0) {
                tilesleft = true;
                tile.alpha -= dt * 15;
                if (tile.alpha < 0) {
                    tile.alpha = 0;
                }
                if (tile.alpha == 0) {
                    tile.type = -1;
                    tile.alpha = 1;
                }
```

```
                }
            }

            for (var i=0; i<floatingclusters.length; i++) {
                for (var j=0; j<floatingclusters[i].length; j++) {
                    var tile = floatingclusters[i][j];
                    if (tile.type >= 0) {
                        tilesleft = true;
                        tile.velocity += dt * 700;
                        tile.shift += dt * tile.velocity;
                        tile.alpha -= dt * 8;
                        if (tile.alpha < 0) {
                            tile.alpha = 0;
                        }
                        if (tile.alpha == 0 || (tile.y * level.rowheight + tile.shift
> (level.rows - 1) * level.rowheight + level.tileheight)) {
                            tile.type = -1;
                            tile.shift = 0;
                            tile.alpha = 1;
                        }
                    }

                }
            }
            if (!tilesleft) {
                nextBubble();
                var tilefound = false
                for (var i=0; i<level.columns; i++) {
                    for (var j=0; j<level.rows; j++) {
                        if (level.tiles[i][j].type != -1) {
                            tilefound = true;
                            break;
                        }
                    }
                }
                if (tilefound) {
                    setGameState(gamestates.ready);
                } else {
                    setGameState(gamestates.gameover);
                }
            }
        }
    }
```

```
function snapBubble() {
    var centerx = player.bubble.x + level.tilewidth/2;
    var centery = player.bubble.y + level.tileheight/2;
    var gridpos = getGridPosition(centerx, centery);
    if (gridpos.x < 0) {
        gridpos.x = 0;
    }
    if (gridpos.x >= level.columns) {
        gridpos.x = level.columns - 1;
    }
    if (gridpos.y < 0) {
        gridpos.y = 0;
    }
    if (gridpos.y >= level.rows) {
        gridpos.y = level.rows - 1;
    }
    var addtile = false;
    if (level.tiles[gridpos.x][gridpos.y].type != -1) {
        for (var newrow=gridpos.y+1; newrow<level.rows; newrow++) {
            if (level.tiles[gridpos.x][newrow].type == -1) {
                gridpos.y = newrow;
                addtile = true;
                break;
            }
        }
    } else {
        addtile = true;
    }
    if (addtile) {
        player.bubble.visible = false;
        level.tiles[gridpos.x][gridpos.y].type = player.bubble.tiletype;
        if (checkGameOver()) {
            return;
        }
        cluster = findCluster(gridpos.x, gridpos.y, true, true, false);
        if (cluster.length >= 3) {
            // Remove the cluster
            setGameState(gamestates.removecluster);
            return;
        }
    }
    turncounter++;
    if (turncounter >= 5) {
```

```
            addBubbles();
            turncounter = 0;
            rowoffset = (rowoffset + 1) % 2;
            if (checkGameOver()) {
                return;
            }
        }
        nextBubble();
        setGameState(gamestates.ready);
    }

    function checkGameOver() {
        for (var i=0; i<level.columns; i++) {
            // Check if there are bubbles in the bottom row
            if (level.tiles[i][level.rows-1].type != -1) {
                nextBubble();
                setGameState(gamestates.gameover);
                return true;
            }
        }
        return false;
    }

    function addBubbles() {
        for (var i=0; i<level.columns; i++) {
            for (var j=0; j<level.rows-1; j++) {
                level.tiles[i][level.rows-1-j].type = level.tiles[i][level.rows-1-j-
1].type;
            }
        }
        for (var i=0; i<level.columns; i++) {
            // Add random, existing, colors
            level.tiles[i][0].type = getExistingColor();
        }
    }
    function findColors() {
        var foundcolors = [];
        var colortable = [];
        for (var i=0; i<bubblecolors; i++) {
            colortable.push(false);
        }

        for (var i=0; i<level.columns; i++) {
```

```
        for (var j=0; j<level.rows; j++) {
            var tile = level.tiles[i][j];
            if (tile.type >= 0) {
                if (!colortable[tile.type]) {
                    colortable[tile.type] = true;
                    foundcolors.push(tile.type);
                }
            }
        }
    }

    return foundcolors;
}

function findCluster(tx, ty, matchtype, reset, skipremoved) {
    if (reset) {
        resetProcessed();
    }
    var targettile = level.tiles[tx][ty];
    var toprocess = [targettile];
    targettile.processed = true;
    var foundcluster = [];
    while (toprocess.length > 0) {
        var currenttile = toprocess.pop();
        if (currenttile.type == -1) {
            continue;
        }
        if (skipremoved && currenttile.removed) {
            continue;
        }
        if (!matchtype || (currenttile.type == targettile.type)) {
            foundcluster.push(currenttile);
            var neighbors = getNeighbors(currenttile);
            for (var i=0; i<neighbors.length; i++) {
                if (!neighbors[i].processed) {
                    toprocess.push(neighbors[i]);
                    neighbors[i].processed = true;
                }
            }
        }
    }
```

```
return foundcluster;
    }

    function findFloatingClusters() {
        resetProcessed();
        var foundclusters = [];
        for (var i=0; i<level.columns; i++) {
            for (var j=0; j<level.rows; j++) {
                var tile = level.tiles[i][j];
                if (!tile.processed) {
                    var foundcluster = findCluster(i, j, false, false, true);
                    if (foundcluster.length <= 0) {
                        continue;
                    }
                    var floating = true;
                    for (var k=0; k<foundcluster.length; k++) {
                        if (foundcluster[k].y == 0) {
                            // Tile is attached to the roof
                            floating = false;
                            break;
                        }
                    }
                    if (floating) {
                        foundclusters.push(foundcluster);
                    }
                }
            }
        }
        return foundclusters;
    }

    function resetProcessed() {
        for (var i=0; i<level.columns; i++) {
            for (var j=0; j<level.rows; j++) {
                level.tiles[i][j].processed = false;
            }
        }
    }

    function resetRemoved() {
        for (var i=0; i<level.columns; i++) {
            for (var j=0; j<level.rows; j++) {
                level.tiles[i][j].removed = false;
```

```
            }
        }
    }

    function getNeighbors(tile) {
        var tilerow = (tile.y + rowoffset) % 2;
        var neighbors = [];
        var n = neighborsoffsets[tilerow];
        for (var i=0; i<n.length; i++) {
            var nx = tile.x + n[i][0];
            var ny = tile.y + n[i][1];
            if (nx >= 0 && nx < level.columns && ny >= 0 && ny < level.rows) {
                neighbors.push(level.tiles[nx][ny]);
            }
        }
        return neighbors;
    }

    function updateFps(dt) {
        if (fpstime > 0.25) {
            fps = Math.round(framecount / fpstime);
            fpstime = 0;
            framecount = 0;
        }
        fpstime += dt;
        framecount++;
    }

    function drawCenterText(text, x, y, width) {
        var textdim = context.measureText(text);
        context.fillText(text, x + (width-textdim.width)/2, y);
    }
    function render() {
        drawFrame();
        var yoffset =  level.tileheight/2;
        context.fillStyle = "#8c8c8c";
        context.fillRect(level.x - 4, level.y - 4, level.width + 8, level.height + 4
- yoffset);
        renderTiles();
        context.fillStyle = "#656565";
        context.fillRect(level.x - 4, level.y - 4 + level.height + 4 - yoffset,
level.width + 8, 2*level.tileheight + 3);
        context.fillStyle = "#ffffff";
```

```
        context.font = "18px Verdana";
        var scorex = level.x + level.width - 150;
        var scorey = level.y+level.height + level.tileheight - yoffset - 8;
        drawCenterText("Очки:", scorex, scorey, 150);
        context.font = "24px Verdana";
        drawCenterText(score, scorex, scorey+30, 150);
        if (showcluster) {
            renderCluster(cluster, 255, 128, 128);
            for (var i=0; i<floatingclusters.length; i++) {
                var col = Math.floor(100 + 100 * i / floatingclusters.length);
                renderCluster(floatingclusters[i], col, col, col);
            }
        }
        renderPlayer();
        if (gamestate == gamestates.gameover) {
            context.fillStyle = "rgba(0, 0, 0, 0.8)";
            context.fillRect(level.x - 4, level.y - 4, level.width + 8, level.height
+ 2 * level.tileheight + 8 - yoffset);
            context.fillStyle = "#ffffff";
            context.font = "24px Verdana";
            drawCenterText("Гра закінчена!", level.x, level.y + level.height / 2 +
10, level.width);
            drawCenterText("Старт", level.x, level.y + level.height / 2 + 40,
level.width);
        }
    }

    function drawFrame() {
        context.fillStyle = "#e8eaec";
        context.fillRect(0, 0, canvas.width, canvas.height);
        context.fillStyle = "#303030";
        context.fillRect(0, 0, canvas.width, 79);
        context.fillStyle = "#ffffff";
        context.font = "24px Verdana";
        context.fillText("Гра Bubble Shooter", 10, 37);
        context.fillStyle = "#ffffff";
        context.font = "12px Verdana";
        context.fillText("Fps: " + fps, 13, 57);
    }

    function renderTiles() {
        for (var j=0; j<level.rows; j++) {
            for (var i=0; i<level.columns; i++) {
```

```
                var tile = level.tiles[i][j];
                var shift = tile.shift;
                var coord = getTileCoordinate(i, j);
                if (tile.type >= 0) {
                    context.save();
                    context.globalAlpha = tile.alpha;
                    drawBubble(coord.tilex, coord.tiley + shift, tile.type);
                    context.restore();
                }
            }
        }
    }

    function renderCluster(cluster, r, g, b) {
        for (var i=0; i<cluster.length; i++) {
            var coord = getTileCoordinate(cluster[i].x, cluster[i].y);
            context.fillStyle = "rgb(" + r + "," + g + "," + b + ")";
            context.fillRect(coord.tilex+level.tilewidth/4,
coord.tiley+level.tileheight/4, level.tilewidth/2, level.tileheight/2);
        }
    }

    function renderPlayer() {
        var centerx = player.x + level.tilewidth/2;
        var centery = player.y + level.tileheight/2;
        context.fillStyle = "#7a7a7a";
        context.beginPath();
        context.arc(centerx, centery, level.radius+12, 0, 2*Math.PI, false);
        context.fill();
        context.lineWidth = 2;
        context.strokeStyle = "#8c8c8c";
        context.stroke();
        context.lineWidth = 2;
        context.strokeStyle = "#0000ff";
        context.beginPath();
        context.moveTo(centerx, centery);
        context.lineTo(centerx + 1.5*level.tilewidth *
Math.cos(degToRad(player.angle)), centery - 1.5*level.tileheight *
Math.sin(degToRad(player.angle)));
        context.stroke();

        drawBubble(player.nextbubble.x, player.nextbubble.y,
player.nextbubble.tiletype);
```

```
        if (player.bubble.visible) {
            drawBubble(player.bubble.x, player.bubble.y, player.bubble.tiletype);
        }

    }

    function getTileCoordinate(column, row) {
        var tilex = level.x + column * level.tilewidth;
        if ((row + rowoffset) % 2) {
            tilex += level.tilewidth/2;
        }
        var tiley = level.y + row * level.rowheight;
        return { tilex: tilex, tiley: tiley };
    }

    function getGridPosition(x, y) {
        var gridy = Math.floor((y - level.y) / level.rowheight);
        var xoffset = 0;
        if ((gridy + rowoffset) % 2) {
            xoffset = level.tilewidth / 2;
        }
        var gridx = Math.floor(((x - xoffset) - level.x) / level.tilewidth);
        return { x: gridx, y: gridy };
    }

    function drawBubble(x, y, index) {
        if (index < 0 || index >= bubblecolors)
            return;
        context.drawImage(bubbleimage, index * 40, 0, 40, 40, x, y, level.tilewidth,
level.tileheight);
    }

    function newGame() {
        score = 0;
        turncounter = 0;
        rowoffset = 0;
        setGameState(gamestates.ready);
        createLevel();
        nextBubble();
        nextBubble();
    }
    function createLevel() {
        for (var j=0; j<level.rows; j++) {
```

```
            var randomtile = randRange(0, bubblecolors-1);
            var count = 0;
            for (var i=0; i<level.columns; i++) {
                if (count >= 2) {
                    var newtile = randRange(0, bubblecolors-1);
                    if (newtile == randomtile) {
                        newtile = (newtile + 1) % bubblecolors;
                    }
                    randomtile = newtile;
                    count = 0;
                }
                count++;
                if (j < level.rows/2) {
                    level.tiles[i][j].type = randomtile;
                } else {
                    level.tiles[i][j].type = -1;
                }
            }
        }
    }

    function nextBubble() {
        player.tiletype = player.nextbubble.tiletype;
        player.bubble.tiletype = player.nextbubble.tiletype;
        player.bubble.x = player.x;
        player.bubble.y = player.y;
        player.bubble.visible = true;
        var nextcolor = getExistingColor();
        player.nextbubble.tiletype = nextcolor;
    }

    function getExistingColor() {
        existingcolors = findColors();
        var bubbletype = 0;
        if (existingcolors.length > 0) {
            bubbletype = existingcolors[randRange(0, existingcolors.length-1)];
        }
        return bubbletype;
    }


    function randRange(low, high) {
        return Math.floor(low + Math.random()*(high-low+1));
```

```
    }

    function shootBubble() {
        player.bubble.x = player.x;
        player.bubble.y = player.y;
        player.bubble.angle = player.angle;
        player.bubble.tiletype = player.tiletype;
        setGameState(gamestates.shootbubble);
    }

    function circleIntersection(x1, y1, r1, x2, y2, r2) {
        var dx = x1 - x2;
        var dy = y1 - y2;
        var len = Math.sqrt(dx * dx + dy * dy);
        if (len < r1 + r2) {
            return true;
        }
        return false;
    }

    function radToDeg(angle) {
        return angle * (180 / Math.PI);
    }

    function degToRad(angle) {
        return angle * (Math.PI / 180);
    }

    function onMouseMove(e) {
        var pos = getMousePos(canvas, e);
        var mouseangle = radToDeg(Math.atan2((player.y+level.tileheight/2) - pos.y,
pos.x - (player.x+level.tilewidth/2)));
        if (mouseangle < 0) {
            mouseangle = 180 + (180 + mouseangle);
        }
        var lbound = 8;
        var ubound = 172;
        if (mouseangle > 90 && mouseangle < 270) {
            if (mouseangle > ubound) {
                mouseangle = ubound;
            }
        } else {
            // Right
```

```
            if (mouseangle < lbound || mouseangle >= 270) {
                mouseangle = lbound;
            }
        }
        player.angle = mouseangle;
    }

    function onMouseDown(e) {
        var pos = getMousePos(canvas, e);
        if (gamestate == gamestates.ready) {
            shootBubble();
        } else if (gamestate == gamestates.gameover) {
            newGame();
        }
    }

    function getMousePos(canvas, e) {
        var rect = canvas.getBoundingClientRect();
        return {
            x: Math.round((e.clientX - rect.left)/(rect.right -
rect.left)*canvas.width),
            y: Math.round((e.clientY - rect.top)/(rect.bottom -
rect.top)*canvas.height)
        };
    }

    init();
};
```