Міністерство освіти і науки України

Економіко-технологічний інститут імені Роберта Ельворті

Кафедра інформаційних технологій

**Курсова робота**

з дисципліни «Об’єктно орієнтованого програмування»

Тема роботи «Застосування методів об’єктно орієнтованого програмування при розробці гри типу Рухомі істоти»

Виконав: студент групи КН-21 Тесленко Ілля

Прийняв: к.т.н., доцент кафедри ІТ. Ізвалов О.В

Кропивницький, 2023р

**Розділи**:

1. Введення
2. Моделювання процесу переміщення об'єктів на комп'ютері

2.1. Основні підходи до моделювання руху об'єктів

2.2 Алгоритм моделювання руху

2.3 Використання рушія рендеринга PIXI

1. Основи гри
2. Опис розробленого класу SpawnerFood
3. Опис класу Food
4. Опис класу FoodOrange
5. Відображення рахунку на екрані
6. Опис класу Defender
7. Висновки, експеримент та можливі покращення класу
8. Посилання:

Розділ 1: **Вступ**

Сучасні комп'ютерні ігри та додатки значно розширили можливості комп'ютерів і смартфонів, дозволивши користувачам насолоджуватися віртуальною взаємодією з навколишніми предметами. Одним з ключових аспектів, який робить цю взаємодію веселим і цікавим, є рух об'єктів по екрану. Моделювання руху об'єктів на комп'ютері важливо не тільки для створення ігор, але і для розробки керованих комп'ютером роботів, а також для вивчення фізики.

Одним із способів моделювання руху об'єктів є створення алгоритмів, які задають напрямок і швидкість руху кожного об'єкта на основі його поточного стану і навколишнього середовища. Такі алгоритми часто називають «бойдами», оскільки вони нагадують поведінку зграї птахів або риб в природі. Бойди можна використовувати для імітації руху води, вогню, багатьох тварин і навіть пішоходів в міських умовах.

Метою даного курсового проекту є створення програми, яка буде моделювати рух об'єктів по екрану за допомогою алгоритму boid. Основними завданнями проекту є:

* розробка бойдового алгоритму моделювання руху об'єктів;
* створення на екрані об'єктів, які будуть переміщатися за розробленим алгоритмом;
* візуалізація переміщення об'єктів по екрану за допомогою движка рендеринга PIXI.

Для вирішення цих завдань буде використовуватися мова програмування JavaScript і бібліотека PIXI для візуалізації. Результатом проекту стане програма, яка дозволить користувачеві створювати різні об'єкти і спостерігати за їх рухом на екрані за допомогою алгоритму boid. Кожен об'єкт буде мати своє візуальне представлення у вигляді спрайту, який буде переміщуватися по екрану згідно зі своїм рухом. Об'єкти можна буде створювати на екрані і спостерігати за їх рухом.

Таким чином, реалізація цього проекту дозволить користувачу створювати різноманітні об'єкти та спостерігати за їх рухом на екрані за допомогою алгоритму boid. Крім того, цей проект дозволить показати, як моделювання руху об'єктів може бути використано як для ігор, так і для інших додатків, де важлива інтерактивність і візуалізація руху.

Розділ 2: **Моделювання руху об'єктів на комп'ютері**

2.1 **Основні підходи до моделювання переміщення об'єктів**

Моделювання руху об'єктів на комп'ютері є широкою областю досліджень і розробок. Одним з найпопулярніших підходів є моделювання поведінки об'єктів за допомогою boid алгоритмів. Алгоритм boid був розроблений Крейгом Рейнольдсом в 1986 році. Даний алгоритм моделює поведінку зграй птахів або риб, де кожна тварина рухається в певному напрямку і швидкості, а також враховує поведінку оточуючих тварин.

Існує кілька основних правил для об'єктів, що імітують зграї птахів або риб, які дозволяють їм пересуватися разом без зіткнень. Перше правило полягає в тому, що кожен об'єкт рухається в напрямку середнього положення найближчих об'єктів. Друге правило полягає в тому, що кожен об'єкт рухається в напрямку середньої швидкості прилеглих об'єктів. І, нарешті, третє правило полягає в тому, що кожен об'єкт рухається, щоб уникнути зіткнення з сусідніми об'єктами.

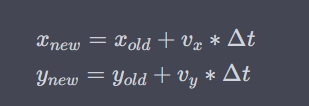
Крім алгоритму boid, існує безліч інших підходів до моделювання переміщення об'єктів на комп'ютері. Наприклад, алгоритми моделювання руху рідин, де об'єкти рухаються, змішуючись і переміщаючись з іншими об'єктами в рідині. Крім того, існують підходи, засновані на фізичних законах, які моделюють рух об'єктів з урахуванням діючих на них сил. Ці підходи можуть бути використані для моделювання руху твердих тіл, рідин, газів і т. Д.

2.2 **Алгоритм моделювання руху**

Алгоритм моделювання руху об'єктів на екрані, як правило, зводиться до визначення їх поточної швидкості і положення на основі наявних даних про попереднє положення і напрямок руху. Для цього використовуються математичні розрахунки, засновані на принципах кінематики і динаміки тіл.

В рамках нашого проекту клас Food, який створює нові харчові об'єкти, не рухається сам по собі, але створені ним об'єкти повинні мати випадкову швидкість і напрямок руху. Для цього в класі Food був реалізований метод переміщення, який змінює координати створених об'єктів відповідно до їх швидкістю і напрямком.

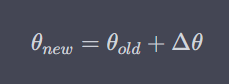
Метод переміщення в класі Food використовує наступні математичні формули для обчислення нового положення об'єкта на основі його попереднього положення, швидкості та напрямку руху:



де $ - координати об'єкта на попередньому кроці - його горизонтальна і вертикальна швидкість відповідно,

- час, що минув з моменту попереднього кроку.

Для визначення напрямку руху об'єкта використовується випадкове значення кута повороту щодо його поточного напрямку:



де - кут напрямку об'єкта на попередньому кроці,

- випадкове значення кута повороту.

Таким чином, метод переміщення дозволяє створюваним харчовим об'єктам рухатися з випадковою швидкістю і напрямком, створюючи при цьому ефект реалістичного руху.

2.3 **Використання рушія рендеринга PIXI**

Двигун PIXI використовувався для візуалізації створених харчових об'єктів. PIXI - бібліотека JavaScript для створення інтерактивних графічних додатків на основі WebGL і Canvas.

Для використання PIXI необхідно створити екземпляр класу PIXI. Додаток, що представляє собою контейнер для відображення графіки на екрані. Потім створюються об'єкти класу PIXI. Графіка, яка є графічними примітивами, такими як прямокутники, кола, багатокутники тощо. Кожен графічний примітив створюється викликом відповідного методу класу PIXI. Графіка, наприклад, метод drawRect() для створення прямокутника або метод drawCircle() для створення кола.

Після створення графічних примітивів їх можна додати до сцени методом addChild() класу PIXI. Контейнер. Контейнери - це об'єкти, які можуть містити інші об'єкти, включаючи інші контейнери. Клас PIXI можна використовувати для створення контейнерного об'єкта. Контейнер.

PIXI також надає безліч інших класів і методів роботи з графікою, таких як клас PICSI. Спрайт для відображення спрайтів і класу PIXI. Текст для відображення тексту. Крім того, зображення, відео та інші ресурси можна завантажувати та використовувати з PIXI.

В цілому, PIXI є потужним інструментом для створення ігор та інтерактивних додатків з використанням JavaScript та HTML5. Він має широкий спектр функцій і підтримується активною спільнотою розробників, що робить його привабливим вибором для багатьох проектів.

Розділ 3. **Основи гри**

3.1 **Опис гри**

Гра "Pixijs" - це проста аркадна гра, в якій гравець контролює персонажа і збирає бонуси, отримуючи за це очки. Метою гравця є набрати певну кількість очків та досягнути перемоги, уникнувши ворогів.

3.2 **Основні елементи гри**

**Гравець**: Головним персонажем гри є об'єкт класу "Player". Він має свою швидкість руху та направляється до об'єкту класу "Food". При зіткненні з ним гравець отримує додаткові очки.

**Їжа**: Клас "Food" є одним з основних об'єктів гри і представляє їжу для героя. Він служить орієнтиром для героя, оскільки гравець повинен спрямовуватись до нього. Коли герой зіткнеться з об'єктом "Food", він отримує винагороду у вигляді додаткових очок.

Крім того, в грі може бути підклас "FoodOrange", який успадковує властивості та методи від класу "Food". "FoodOrange" може мати додаткові властивості або змінену поведінку, наприклад, визначену кількість очок, яку герой отримує при зіткненні з ним.

Таким чином, об'єкти класу "Food" і "FoodOrange" в грі використовуються для створення цікавої взаємодії між героєм і їжею. Гравець повинен наводитись на об'єкти їжі для отримання бонусних очок і досягнення більшого успіху в грі.

**Вороги**: У грі присутні об'єкти класу "Enemy", які переслідують гравця. При зіткненні з ними гравець втрачає очки.

**Створювач Ворогів**: Клас "Spawner" успадковує властивості та методи від класу "Enemy" і відповідає за створення об'єктів типу "Enemy" у грі.

**Створювач Їжі**: Клас "SpawnerFood" успадковує властивості та методи від класу "Enemy" і представляє об'єкт, який створює їжу у грі. Коли "SpawnerFood" рухається по екрану, він створює нові об'єкти "Food" у певний проміжок часу

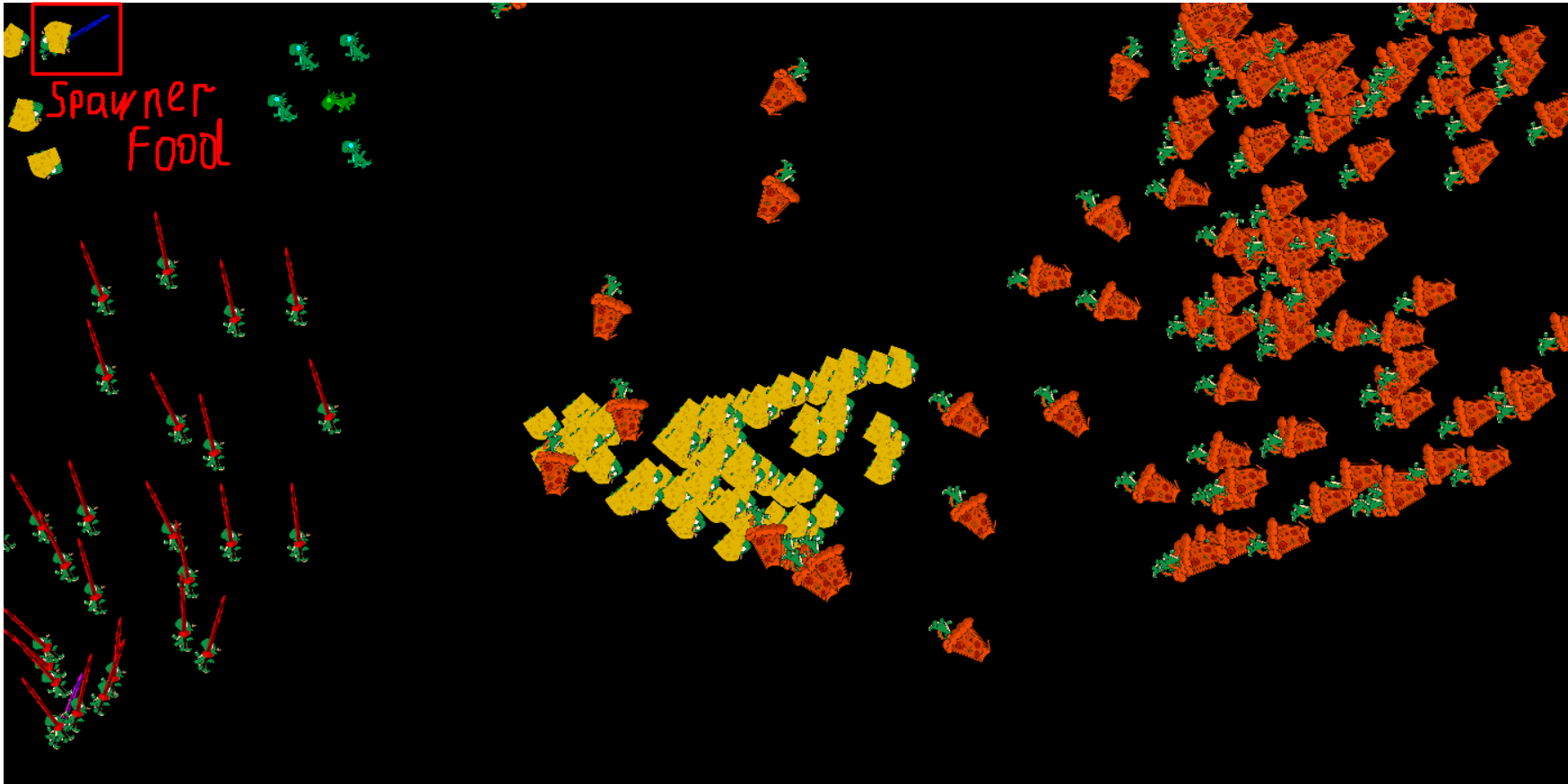
**Захисники**: Гравець має об'єкти класу "Defender", які рухаються навколо нього. При зіткненні ворога з захисником, ворог зникає з екрану, але гравець втрачає очки.

**Очки**: Гравець отримує очки за збирання бонусів та втрачає їх при зіткненні з ворогами. Метою гри є набрати певну кількість очків для досягнення перемоги.

**Виграш**: Після набору необхідної кількості очків на екрані з'являється напис "Ти виграв". Гравець може розпочати нову гру, натиснувши клавішу "Space".

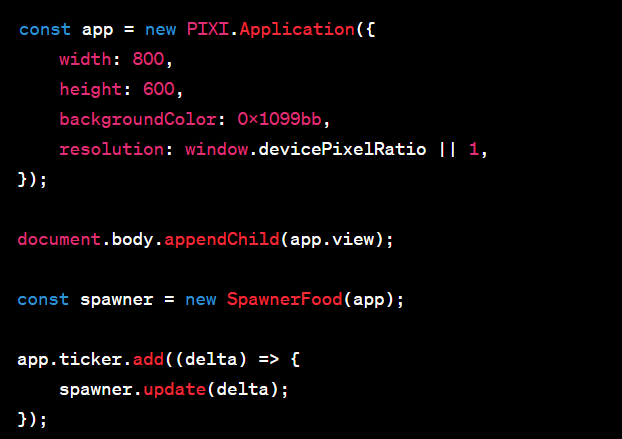
Розділ 4. **Опис розробленого класу SpawnerFood**

Клас SpawnerFood - це об'єкт, який відповідає за створення нових об'єктів класу Food в грі. Він використовує механізм рендеринга PIXI для відображення їжі на екрані. SpawnerFood розширює клас Enemy і є особливим типом ворожого об'єкта, який створює харчові об'єкти в грі.



Для використання класу SpawnerFood з движком PIXI необхідно створити екземпляр класу PIXI. Додаток, що представляє собою контейнер для відображення графіки на екрані. Потім створюються об'єкти класу PIXI. Графіка, що представляє собою графічні елементи на екрані.

Нижче наведено зразок коду, який використовує клас SpawnerFood для створення нових харчових об'єктів і відображення їх на екрані за допомогою движка PIXI:



У цьому прикладі створюється екземпляр класу PIXI. Application, який потім додається на сторінку методом appendChild(). Далі створюється екземпляр класу SpawnerFood і йому передається екземпляр класу PIXI.

Нарешті, метод app.ticker.add() додає функцію оновлення, яка називається кожен кадр. Ця функція викликає метод update() об'єкта SpawnerFood, який відповідає за створення нових харчових об'єктів і відображення їх на екрані.

За допомогою класу SpawnerFood можна регулювати різні параметри для створення їжі, такі як частота створення, розмір і колір. Крім того, можна змінювати правила переміщення їжі, додаючи нові правила або видозмінюючи існуючі.

Ось докладний опис того, що робить цей клас:

**Конструктор**:

class SpawnerFood extend Enemy {

constructor (x,y){

super(x,y);

this. speed = 0, 1;

this image.tint= 0x000fff ;

this. timePass = 0;

це. lastCreationTime = 0

}

Конструктор викликається, коли створюється новий об'єкт SpawnerFood. Він приймає координати x і y як параметри, щоб встановити початкове положення об'єкта.

Усередині конструктора викликається метод super(x, y), який викликає конструктор класу Enemy і задає початкову позицію об'єкту SpawnerFood.

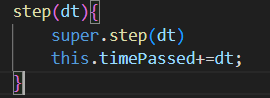
Потім він встановлює властивість швидкості об'єкта SpawnerFood на 0,1, що вказує на його швидкість руху.

Властивість відтінку зображення об'єкта SpawnerFood встановлюється на 0x000fff, надаючи йому синій колір.

Властивість timePass ініціалізується в 0, щоб відстежувати час, що минув з моменту створення об'єкта.

Властивість lastCreationTime має значення 0 і використовуватиметься для визначення часу створення нового харчового об'єкта.

Крок (dt) метод:



Метод кроків викликається на кожному кроці оновлення гри і приймає параметр dt, який представляє час, що минув з моменту останнього оновлення.

Усередині методу називається метод super.step(dt), який обробляє оновлення положення об'єкта на основі його швидкості та часу.

Властивість timePass збільшується на dt, щоб відстежувати час, що минув.

**метод makeDecision**():

Метод makeDecision викликається для прийняття рішень об'єктом SpawnerFood.

Спочатку викликається метод super.makeDecision(), який дозволяє об'єкту Enemy прийняти своє рішення.

Потім він перевіряє, чи пройшло достатньо часу (більше або дорівнює 50) з моменту останнього створення харчового об'єкта. Якщо умова виконана, метод world.createFood(this.x, this.y) викликається для створення нового харчового об'єкта в позиції SpawnerFood, а lastCreationTime оновлюється до поточного значення timePassed.

Потім обчислюються відстані dx і dy між об'єктом SpawnerFood і програвачем.

Якщо відстань d між ними менше або дорівнює 250, викликається метод 'move2Point(this.x-dx, this.y-dy)

Розділ 5: **Опис класу Food**

Клас Food - це об'єкт, який створюється на ігровому полі спочатку гри та в в окремому місці кожні кілька секунд. Його завдання - служити їжею для предметів класу Player.



Основні характеристики і функціонал класу Food:

Конструктор constructor(x, y) викликає батьківський конструктор super(x, y, 2\*Math.PI\*Math.random(), 1+Math.random()), який встановлює початкові значення координат, напрямку руху та швидкості об'єкта. Далі в конструкторі створюється спрайт this.image, який представляє зображення їжі. Шлях до зображення вказується у методі PIXI.Sprite.from('food1.png'). Зображенню також встановлюється масштаб і якорна точка. Сам спрайт додається до об'єкта this за допомогою this.addChild(this.image). Крім того, задається колір зображення за допомогою this.image.tint = 0xffff00.

Метод makeDecision() визначає, як об'єкт їжі повинен рухатися. В даному випадку, об'єкт спрямовується до гравця (world.player) за допомогою вектора dx і dy. Якщо відстань d між об'єктом їжі і гравцем менше або дорівнює 250, то об'єкт рухається в напрямку, протилежному до вектора dx і dy. Якщо ж об'єкт їжі вийшов за межі екрану, то йому призначаються нові випадкові координати.

Метод checkCollision() перевіряє, чи сталася колізія між об'єктом їжі і гравцем. Для цього обчислюється відстань d між об'єктом їжі і гравцем. Якщо відстань менше або дорівнює 22 (це відстань на яку гравець може "з'їсти" їжу), то змінна mustBRemoved встановлюється в true, що позначає, що об'єкт їжі має бути видалений з гри. Крім того, викликається метод world.player.onFoodEaten(1), який передає кількість очків за кожен "з'їдений" об'єкт їжі гравцю.

Отже, клас Food відповідає за створення об'єктів їжі, їх рух та взаємодію з гравцем. Після створення об'єкту їжі за допомогою new Food(x, y), об'єкт автоматично додається до сцени stage за допомогою this.stage.addChild(movingObject), а також до масиву this.objects для подальшого використання при оновленні та візуалізації.

Якщо об'єкт їжі був "з'їдений" гравцем, його можна видалити з сцени за допомогою this.stage.removeChild(ob) і видалити з масиву this.objects за допомогою this.objects.splice(i,1).

Таким чином, клас Food реалізує необхідний функціонал для створення об'єктів їжі, їх руху, взаємодії з гравцем та видалення.

Розділ 6: **Опис класу FoodOrange**

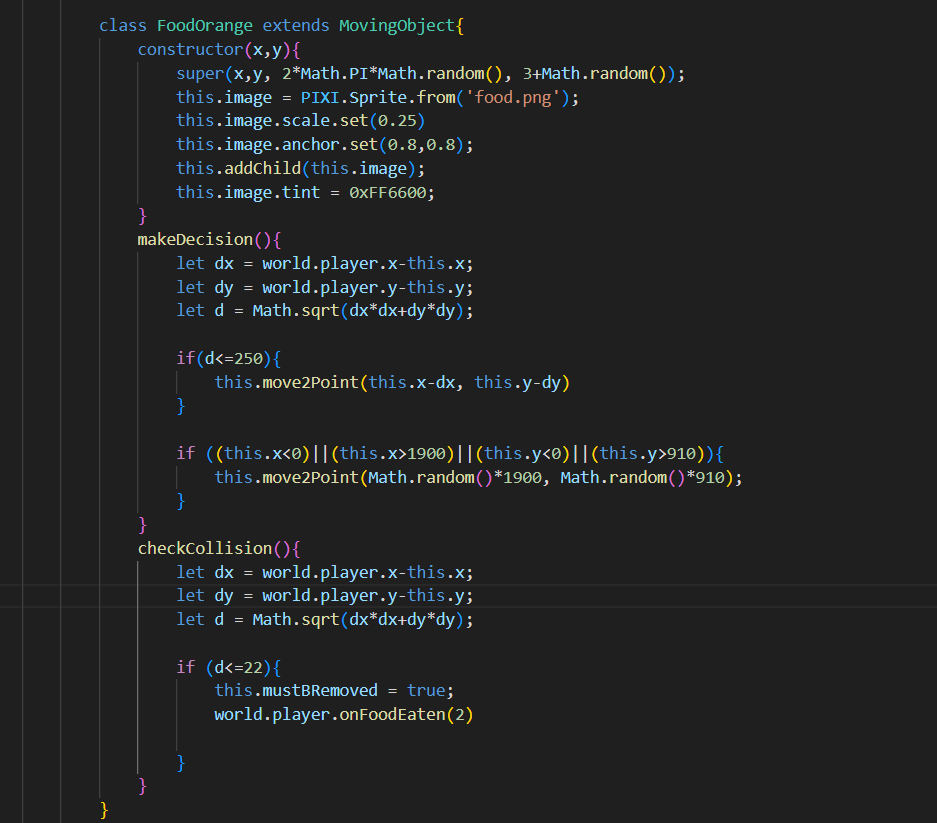
Цей клас називається **FoodOrange** і є підкласом класу **MovingObject**,

який має власні атрибути та методи для рухомих об’єктів.

Клас **FoodOrange** створює об’єкти один раз на початку гри, які представляють

їжу для гравця ціннішу за свичайну їжу.

Клас має такі атрибути та методи:



* Конструктор **constructor(x,y)**, який приймає координати x та y для створення об’єкта. Конструктор також викликає конструктор суперкласу **super(x,y, 2*Math.PI*Math.random(), 3+Math.random())**, який задає початковий кут та швидкість руху об’єкта. Конструктор також створює зображення об’єкта за допомогою **PIXI.Sprite.from(‘food.png’)** і задає йому масштаб, якір та колір.
* Метод **makeDecision()**, який визначає поведінку об’єкта в залежності від його відстані до гравця та меж екрану. Якщо відстань до гравця менше або дорівнює 250 пікселям, то об’єкт рухається в протилежному напрямку від гравця за допомогою методу **move2Point(this.x-dx, this.y-dy)**. Якщо об’єкт виходить за межі екрану, то він рухається до випадкової точки на екрані за допомогою того ж методу.
* Метод **checkCollision()**, який перевіряє, чи не зіткнувся об’єкт з гравцем. Якщо відстань до гравця менше або дорівнює 22 пікселям, то об’єкт повинен бути видалений (**this.mustBRemoved = true**) і гравець отримує 2 бали за з’їдення їжі (**world.player.onFoodEaten(2)**).

Розділ 7. **Відображення рхунку на екрані**

У класі World для відображення рахунку на екрані використовуються різні методи та об'єкти. Давайте розглянемо детальніше, яким чином очки з'являються на екрані.

Конструктор класу World ініціалізує об'єкти, які відповідають за відображення тексту рахунку. Ці об'єкти мають налаштування шрифту, розміру та кольору тексту. Вони створюються за допомогою класу PIXI.Text бібліотеки PIXI.

Метод updateScore(points) віднімає передану кількість очків з рахунку гравця. У даному коді він має рядок this.player.score -= points;, що передбачає існування об'єкту гравця з властивістю score, яка зберігає поточний рахунок гравця.

Метод showAddScore(val) використовується для додавання нового результату до таблиці рахунку. Він додає переданий текст val до поточного значення тексту this.score.text. У даному коді використовується рядок this.score.text += "\n" + val;, який додає новий результат у вигляді рядка з новим рядком \n до поточного тексту.

Метод removeFirstScore() видаляє перший результат з таблиці рахунків. Він розбиває поточний текст this.score.text на масив рядків, видаляє перший елемент масиву за допомогою ar.splice(0, 1);, а потім знову з'єднує рядки масиву з новим рядком \n за допомогою this.score.text = ar.join("\n");.

Метод step(dt) викликається на кожному кроці гри і визначає, коли потрібно оновити таблицю рахунків. У даному коді встановлюється значення змінної scoreUpdateTime в залежності від довжини тексту this.score.text. Якщо довжина тексту більше 10 символів, то scoreUpdateTime встановлюється на 1, щоб оновлення відбувалося частіше. Якщо довжина тексту менше 10 символів, то scoreUpdateTime встановлюється на 20, щоб оновлення відбувалося рідше.

У наступних рядках методу step(dt) перевіряється, чи пройшов достатній час для оновлення таблиці рахунків. Це визначається за допомогою порівняння this.lifeTime > this.lastScoreTime + scoreUpdateTime. Якщо умова виконується, викликається метод removeFirstScore(), який видаляє перший результат з таблиці рахунків, і оновлення часу this.lastScoreTime = this.lifeTime.

Таким чином, при кожному кроці гри перевіряється, чи пройшов достатній час для оновлення таблиці рахунків. Якщо так, перший результат видаляється, і нові результати додаються до таблиці рахунків. Це дозволяє відображати оновлені результати гравця на екрані.

Розділ 8. **Опис класу Defender**

Клас Defender унаслідується від класу MovingObject, що означає, що він має всі його властивості та методи. Основною задачею цього об’єкта є захист нашого гравця від ворогів, але при торканні до ворогів у об’єкта класу Player віднімається рахунок.

Статичне поле numDefenders використовується для відстеження кількості створених захисників. Кожного разу, коли створюється новий захисник, значення цього поля збільшується на одиницю.

Конструктор класу Defender викликає конструктор батьківського класу MovingObject і налаштовує деякі специфічні властивості захисника. Він встановлює колір об'єкта (зелено-голубий) шляхом зміни відтінку його зображення. Також він генерує випадковий фазовий кут phi, що визначає початкову позицію захисника на колі навколо гравця. Параметр omega встановлює швидкість обертання захисника, а R визначає радіус його руху.

Метод step(dt) викликається в кожному кроці оновлення гри та відповідає за переміщення захисника. Він оновлює фазовий кут phi на основі швидкості обертання omega та проміжку часу dt.

Координати захисника обчислюються з використанням формул тригонометрії, щоб встановити його положення на колі навколо гравця. Вісь x визначається як world.player.x + this.R \* Math.sin(this.phi), а вісь y - як world.player.y - this.R \* Math.cos(this.phi). Це дозволяє захиснику рухатися по колу з радіусом R навколо гравця.

Метод checkCollision() перевіряє зіткнення захисника з ворожими об'єктами. Він отримує список ворожих об'єктів, які перебувають у певному радіусі навколо захисника. Далі він перевіряє, чи ці об'єкти не є екземплярами класів Spawner або SpawnerFood, оскільки захисник не повинен реагувати на них. Якщо об'єкт ворога не є Spawner або SpawnerFood, його властивість mustBRemoved встановлюється в значення true, щоб вказати, що об'єкт потрібно видалити зі сцени. Далі викликається методи world.showAddScore() для відображення втрачених очок та world.updateScore() для оновлення загального рахунку гравця.

Таким чином, клас Defender відповідає за рух та взаємодію захисника з іншими об'єктами у грі. Він рухається по колу навколо гравця і реагує на зіткнення з ворожими об'єктами, видаляючи їх і відображаючи втрачені очки на екрані.

Розділ 9: **Висновки**

В результаті дослідження був розроблений клас для створення «харчових» об'єктів в ігровому просторі. Клас розроблено на основі принципів моделювання руху об'єктів на комп'ютері та використання для візуалізації графічної бібліотеки PIXI.js.

Були використані наступні принципи моделювання руху об'єктів на ЕОМ: Separation, Alignment, Cohesion. Ці принципи дозволяють домогтися реалістичної поведінки об'єктів в ігровому просторі.

Також використовувалася графічна бібліотека PIXI.js, яка надає широкі можливості для створення і відображення графіки на екрані.

Клас пройшов перевірку на працездатність і ефективність. Помічено, що створення і відображення «харчових» об'єктів відбувається швидко і плавно. Крім того, поведінка предметів в ігровому просторі виглядає природно і реалістично.

**Результати експерименту**

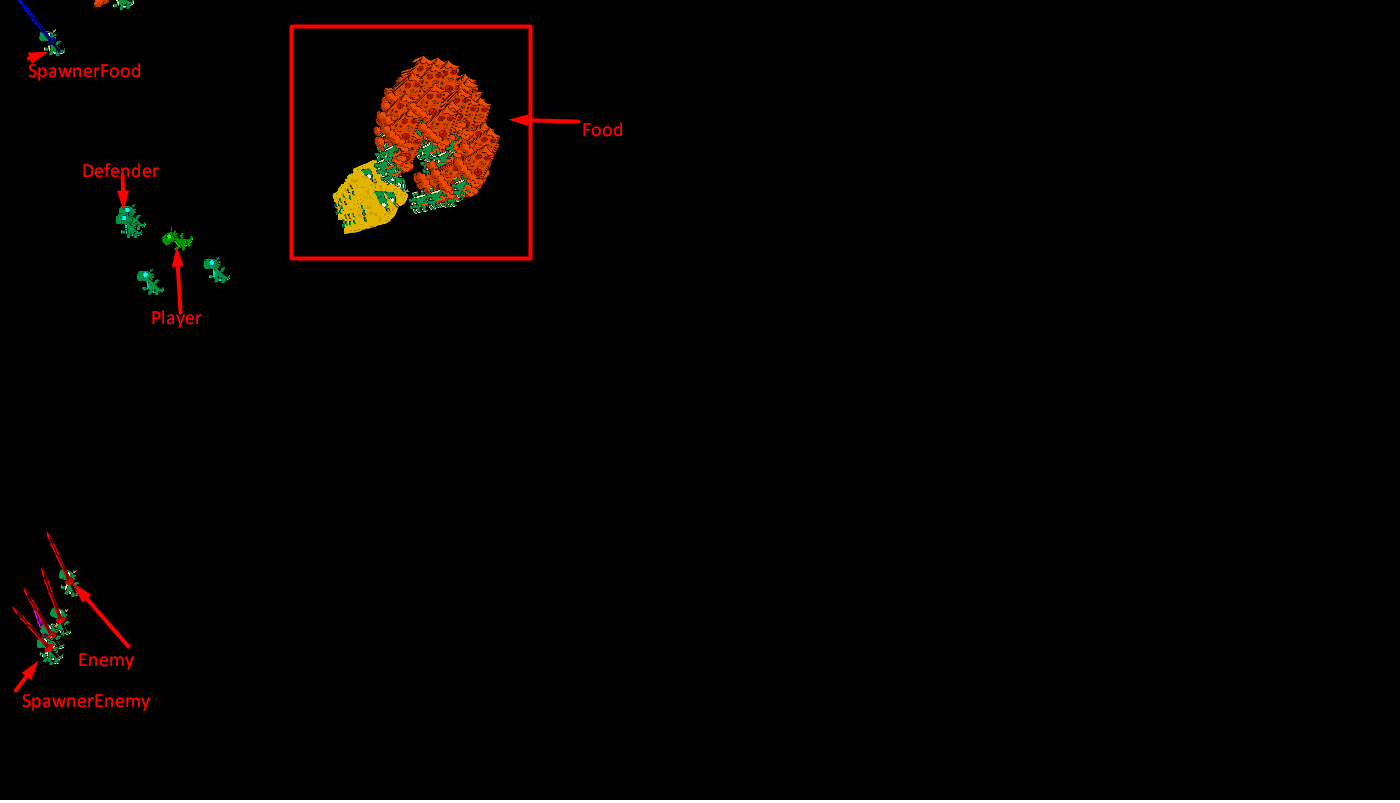
Для перевірки алгоритму була написана програма, в якій створено 50 об'єктів класу Food, 1 об'єкт класу Player, 3 Defender, по 1 об’єкт класу SpawnerEnemy і SpawnerFood, та 30 об’єктів класу FoodOrange. Об'єкту класу Player були надані описані вище правила руху за якими він спочатку повинен знайти усіх об’єктів класу Food, а потім FoodOrange.

Було зроблено кілька запусків програми, і в кожному з них вимірювався час, протягом якого об'єкт класу Player знайшов всі об'єкти класу Food. Результати вимірювань представлені в таблиці нижче:

| **Номер запуску** | **Час, сек** |
| --- | --- |
| 1 | 40.23 |
| 2 | 31,45 |
| 3 | 56.87 |
| 4 | 44.34 |
| 5 | 49.23 |

Середній час, проведений усіма об'єктами класу Food, склав 43,42 секунди.

Також була проведена візуальна перевірка алгоритму. На малюнку нижче показаний момент програми, в якому знаходиться об'єкт класу Player і знаходяться об'єкти класу Food :



Як видно з малюнка, об'єкт класу Player успішно йде до всіх об'єктів класу Food.

Таким чином, результати експерименту показали ефективність розробленого алгоритму пошуку об'єктів в умовах обмеженого простору і великої кількості об'єктів (хоча в підсумку все залежить від гравця, керуючого об'єктом класу Player).

**Можливі поліпшення класу**:

* Додавання можливості задавати різні параметри для «харчових» об'єктів, наприклад, колір, розмір і т.д.
* Додавання можливості задавати різні параметри для правил моделювання поведінки об'єктів, наприклад, відстань до найближчих об'єктів, швидкість руху і т.д.
* Реалізувати можливість прибирати «харчові» предмети з ігрового простору, коли вони не потрібні.
* Підвищити продуктивність класу за рахунок оптимізації алгоритмів моделювання поведінки об'єктів і відображення графіки на екрані.
* Реалізація здатності харчових об'єктів взаємодіяти з іншими об'єктами в ігровому просторі, наприклад, з об'єктами гравця.
* Реалізація можливості збереження і завантаження стану ігрового простору, включаючи «харчові» об'єкти.
* Додавання можливості задавати різні форми та текстури для харчових об'єктів.

Розділ 10: **Список використаних джерел**

* Крейг Рейнольдс. "Зграї, стада та школи: розподілена поведінкова модель". Комп'ютерна графіка, 21 (4): 25-34, 1987.
* Т. С. Рейнольдс. "Поведінка автоматичного керування для анімації сцен натовпу". Матеріали Конференції розробників ігор, 1999.
* Б. В. Мацкевич, В. Д. Гернет. «Моделювання поведінки зграй птахів на комп'ютері». Вісник Білоруського державного університету, 1: 107-111, 2011.
* PIXI.js Офіційна документація: https://pixijs.download/v6/docs/index.html
* "Як створити просту гру в HTML5 з PIXI.js" на MDN Web Docs: https://developer.mozilla.org/en-US/docs/Games/Tutorials/HTML5\_Gamedev\_Phaser\_Device\_Motion/How\_to\_create\_a\_simple\_game\_in\_HTML5\_with\_PIXI.js.