**Лабораторна робота №4**

**Тема: Створення простої гри**

***Мета роботи:*** навчитися працювати з HTML API

**Основні теоретичні відомості**

**Основи використання HTML5 Canvas**

Полотно HTML5 – прозорий елемент, який використовується для малювання та анімації.

Типовий HTML-код для додавання полотна на веб-сторінку:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Draw a monster in a canvas</title>

</head>

<body>

<canvas id="myCanvas" width="200" height="200"></canvas>

</body>

</html>

Можна використати атрибути, щоб надати полотну ширину та висоту, але якщо не задати деякі властивості CSS, його не буде видно на екрані, оскільки полотно прозоре.

Скористаємося CSS, щоб додати рамку до полотна, наприклад,

canvas {

border: 1px solid black;

}

Декілька порад для використання полотна:

1) Після завантаження сторінки повністю (DOM готовий) викликаємо функцію, яка дозволить вибрати полотно в DOM.

2) Потім отримуємо 2D графічний контекст для цього полотна (контекст - це об'єкт, який будемо використовувати для малювання на полотні, для встановлення глобальних властивостей, таких як колір, градієнти, візерунки та ширина лінії).

3) Можна скористатися глобальним змінними для полотна та контекстних об’єктів. Також можна зберігати в ідентифікатори ширину та висоту полотна. Вони можуть бути корисні пізніше.

4) Кожну функцію, яка змінює контекст (колір, ширини лінії, систему координат тощо), розпочинаємо із збереження контексту, а закінчуємо його відновленням.

Наприклад,

// useful to have them as global variables

let canvas, ctx;

window.onload = function init() {

// called AFTER the page has been loaded

canvas = document.querySelector("#myCanvas");

// important, we will draw with this object

ctx = canvas.getContext('2d');

// ready to go! We can use the context for drawing

// or changing colors, line widths, etc.

// filled rectangle

ctx.fillStyle = 'red';

ctx.fillRect(10, 10, 30, 30);

// wireframe rectangle

ctx.strokeStyle = 'green';

ctx.lineWidth = 4;

ctx.strokeRect(100, 40, 40, 40);

// fill circle, will use current ctx.fillStyle

ctx.beginPath();

ctx.arc(60, 60, 10, 0, 2\*Math.PI);

ctx.fill(); // or ctx.stroke() for a wireframe circle

// some text

ctx.fillStyle = "purple";

ctx.font = "20px Arial";

ctx.fillText("Hello!", 60, 20); // or ctx.strokeText for wireframe

}

**Анімація**

Типовий цикл анімації буде виконувати через рівні проміжки часу наступні дії:

1) Очистити полотно

2) Намалювати графічні об’єкти / фігури

3) Здійснити переміщення графічних фігур / об’єктів

4) Перейти до першого кроку

Необов’язковими кроками можуть бути:

Звернення до клавіатури / мишки / геймпаду, якщо потрібно зробити щось відповідно до їх статусу (тобто, якщо натиснута стрілка ліворуч: перемістити гравця вліво)

Ігрові зіткнення: гравець зіткнувся з ворогом, видаляємо одне життя

Можливі стани гри: якщо життя більше немає, переходимо до стану «гра завершена» та показуємо меню «гра завершена».

Для створення об’єктів гри будемо використовувати кола та прямокутники, які назвемо «гравець» та «кулька»:

const ball = {

x: 100,

y: 100,

radius: 15,

color:'green',

speedX: 2,

speedY: 1,

}

const player = {

x: 10,

y: 10,

width: 20,

height: 20,

color: 'red',

}

За допомогою цього синтаксису простіше маніпулювати координатами кульки – достатньо використовувати ball.x чи ball.y. Також встановлено дві властивості для кульки: speedX та speedY. Їх значення - кількість пікселів, які будуть додані до поточної позиції ball.x та ball.y на кожному кадрі анімації.

Розглянемо цикл анімації:

function mainLoop() {

  // 1 - clear the canvas

  ctx.clearRect(0, 0, w, h);

**// draw the ball and the player**

  drawFilledRectangle(player);

  drawFilledCircle(ball);

  // animate the ball that is bouncing all over the walls

  moveBall(ball);

  // ask for a new animation frame

  requestAnimationFrame(mainLoop);

}

На кожному кадрі анімації будемо очищати полотно, малювати гравця у вигляді прямокутника, малювати кульку у вигляді кола і рухати кулькою.

function moveBall(b) {

b.x += b.speedX;

b.y += b.speedY;

testCollisionBallWithWalls(b);

}

Ця функція викликається 60 разів на секунду.

Після зміщення позиції кульки, викликається функція testCollisionBallWithWalls(b), який перевірятиме, чи куля b потрапляє у вертикальну або горизонтальну стіну.

function testCollisionBallWithWalls(b) {

// COLLISION WITH VERTICAL WALLS?

if((b.x + b.radius) > w) {

// the ball hit the right wall

// change horizontal direction

b.speedX = -b.speedX;

// put the ball at the collision point

b.x = w - b.radius;

} ...

...

}

Умова (b.x + b.radius) > w, дозволяє перевірити чи частина кульки виходить за межі правої стінки (де w – ширина полотна)

Кожну 1/60-ту секунду м'яч зміщується певну кількість пікселів праворуч (точне значення – b.speedX). Припустимо, що м’яч рухається на 10 пікселів праворуч на кожному кадрі анімації. У якийсь момент він «перетне праву стінку». Якщо просто змінити знак b.speedX, щоб перейти до руху в протилежну сторону, то він може залишитися притиснутим до сторони однією половиною кулі з обох боків стіни.

Якщо зараз видалити значення b.speedX, то повернемо м'яч у положення, в якому він був до того, як він вдарився про стіну (він не доторкнеться до стіни), але для отримання якісного візуального ефекту удару м’яча об стінку , слід перевести м'яч "у контактне положення", а не в його попереднє положення, перш ніж інвертувати значення швидкості.

**Події мишки на полотні**

Для відслідковування подій мишки на полотні слід додати на полотно слухача подій, а браузер викликає цього слухача, коли подія відбувається.

canvas.addEventListener('mousedown', function (evt) {

// do something with the mousedown event

});

canvas.addEventListener('mousedup', function (evt) {

// do something with the mouseup event

});

**Завдання на лабораторну роботу**

**Основні завдання**

1) Додайте поля вводу для параметризації гри

– кількість кульок, які слід «з’їсти», що виграти гру

– максимальна швидкість куль

– розмір куль (мінімальне та максимальне значення)

– розмір гравця

– імя гравця

2) Забезпечте запуск та зупинку гри за допомогою кнопок.

3) Додайте в гру «рівні»: коли всі хороші кульки будуть з'їдені, перезапускаємо гру автоматично, але на цей раз з ще одним м'ячем у початковому наборі.

Додайте відображення поточного номера рівня

4) Стилізуйте гру за допомогою CSS

5) Збереження гри в LocalStorage (рейтинг гравців)

6) Збереження даних про гру на локальному сервері (json-server) та відповідно завантаження даних при старті гри

<https://www.npmjs.com/package/json-server>

Встановлення сервера:

npm install -g json-server

Створюємо каталог для гри. Створюємо в каталозі папку Public та файл db.json

Запуск сервера (в папці з грою):

json-server --watch db.json

**Додаткові завдання**

1) Змініть алгоритм гри, так щоб гравець, для того щоб виграти мав би «їсти» кульки лише заданого кольору, а якщо помилився, то це б зменшувало його рівень життєвої сили.

Додайте інші поля для введення для подальшого налаштування гри.

2 Спробуйте зробити рухи м’яча різними залежно від їх кольору (випадковий напрямок, який змінюється по ходу м'яча, руху зигзагом тощо)

3) Спробуйте змусити кулі змінювати свій розмір під час анімації.

4) Додайте гравцю можливість пострілами знищувати ворожі кулі.

Створіть репозиторій з грою <https://github.com/>. Закомітьте у нього виконані завдання та надайте доступ користувачу [serhii.zikratyi@gmail.com](mailto:serhii.zikratyi@gmail.com)