ЗБЕРІГАННЯ ІНФОРМАЦІЇ ПРО ДИНАМІЧНІ ОБ'ЄКТИ У ДВОВИМІРНІЙ КЛІТИНКОВІЙ ГРІ

Київський О.Л.
Науковий керівник – ст. викл. каф. ПІ Новіков Ю. С.
Харківський національний університет радіоелектроніки
(61166, Харків, просп. Науки,14, каф. ПІ, тел. (066) 028-60-41)
e-mail: oleksandr.kyivksyi@nure.ua

When making a grid-based game, you need the way to store all of your enemies or items information and coordinates, so they know about each other and can interract. Storing game objects on a grid is a tricky task, because it requires fast and convenient information access, as well as ability to extend. Performance is also an issue, and our data storage should work the same speed, when the grid is scaled to a larger degree. Associative arrays can solve all of this problems, and allow quick and easy way to get and manipulate positions of objects.

При створенні клітинкової двовимірної гри, на єтапі проектування постає питання про те, як саме зберігати інформацію про об'єкти, що повинні мати можливість рухатися та взаємодіяти один з одним. Деякі спеціалізовані ігрові редактори мають вбудовані засоби для роботи з клітинками, але що робити, якщо обраний редактор не має таких засобів за замовчуванням, або ми хочемо створити власну реалізацію?

Перш за все необхідно розібратися, як саме буде представлена інформація, що повинна зберігатися. Беручи до уваги різноманітність ворогів і предметів, та їх властивості, зберігати інформацію про них у статичних об'єктах(як, наприклад, текстури оточення) погана ідея. Тобто кожен елемент клітинки повинен представлятися об'єктом, що має свої властивості.

Також, необхідно пам'ятати, що об'єкти, які пересуваються по цій сітці повинні мати можливість знаходити шлях до цілі, тобто доступ до зберігаємої інформації не повинен витрачати багато обчислювальних ресурсів.

Один із способів вирішення цієї проблеми, який вже вбудований в більшість ігрових редакторів – детектор коллізії. І справді, кожного разу коли ми пересуваємо об'єкт, або робимо перевірку, можна перевіряти клітинку на фізичну колізію, та отримати об'єкт, з яким можна взаємодіяти, або зрозуміти, що клітинка порожня. Але такий підхід має багато недоліків, такі як високе використання ресурсів, труднощі зі знаходженням об'єктів, проблеми взаємодії об'єктів різних розмірів.

Стандартною імплементацією можна вважати двовимірний масив. Такий спосіб зберігання перевірений часом і, хоч, є надійним, але не дуже зручним. Потрібно багато налаштувань, щоб зробити такий метод працювати на полі будь-якої величини, з від'ємними координатами. Також,

при кожному виході об’єкту за границі цього масиву, необхідно його розширювати – створювати новий, та переносити всю інформацію до нього.

Спосіб, що можна вважати одночасно ефективним, та зручим є зберігання ігрових об’єктів у асоціативному масиві.

Асоціативний масив, це спосіб зберігання інформації у вигляді пар ключ/значення, де ключ представляє собою просте значення, за допомогою котрого можна знайти необхідну нам інформацію.

У нашому випадку, ключем слугує координата об’єкту, а точніше двувимірний, цілочисленний вектор, що відображає позицію об’єкта на сітці, а значення, що буде зберігатися – сам об’єкт, який на цій позиції знаходиться.

Такий спосіб зберігання дозволяє швидко отримати об’єкт, що знаходиться(або відсутній) за вказаними координатами. При цьому, зміна позиції об’єкта досить проста задача, а розмір ігрового поля не суттєвий, бо ми можемо зберігати буд-яку кількість ключів без обмеження величини координат.

Також, у більшості мов програмування, асоціативні масиви(або словники) оптимізовані і позволяють ефективно знаходити значення по ключу(але не навпаки). У деяких випадках, часова складність знаходження елементів словника наближується до O(n), і швидкість доступу до інформації не залежить від кількості елементів масиву.

Асоціативні масиви можуть зберігати однакові значення, але ключі мають бути унікальні. Це дозволяє уникнути ситуації, коли на одній координаті знаходиться декілька об’єктів.

Отже, існує багато способів зберігання інформації про об’єкти на двовимірній сітці, що мають свої особливості і підходять для різних випадків. У випадку розробки гри, важливими якостями слугують ефективність, зручність редагування, розширюваність, та можливість зберігання об’єктів, що будуть створюватися у майбутньому. Асоціативні масиви впевнено справляються з цими задачами.

Список використаних джерел

1. SDL Tutorials [Електронний ресурс] – Режим доступу до ресурсу: http://www.sdltutorials.com/sdl-collision
2. Brilliant – Associative arrays [Електронний ресурс] – Режим доступу до ресурсу: https://brilliant.org/wiki/associative-arrays/
3. Metanit – Коллекція Dictionary [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/tutorial/4.9.php