УДК 004.021:004.75

ОБРАБОТКА И РАСПОЗНАВАНИЕ ГЕОМЕТРИЧЕСКИХ ПАТТЕРНОВ В ИГРОВЫХ ДАННЫХ

С. А. Берговин
Учащийся учреждения образования «Национальный детский технопарк», учащийся ГУО «Гимназия имени Я. Купалы» г. Мозыря

А. Ю. Саливон
Учащаяся учреждения образования «Национальный детский технопарк», учащаяся ГУО «Высоковская средняя школа» Пинского района

Ф.В. Усенко
ассистент кафедры инженерной психологии и эргономики БГУИР, магистр
f.usenko@bsuir.by

А.М. Прудник
Доцент кафедры инженерной психологии и эргономики БГУИР, кандидат технических наук, доцент
aleksander.prudnik@bsuir.by

С. А. Берговин
Учащийся гимназии имени Я. Купалы г. Мозыря, область интересов связана с информатикой и изучением точных наук.

А. Ю. Саливон
Учащаяся Высоковской средней школы Пинского района, область интересов связана с информатикой и изучением точных наук.

Ф. В. Усенко
Окончил Белорусский государственный университет информатики и радиоэлектроники. Областью научных интересов является моделирование пользователей информационных систем и разработка интерфейсов.

А. М. Прудник
Окончил Белорусский государственный университет информатики и радиоэлектроники. Область научных интересов связана с взаимодействием человека с компьютером, интерфейсами информационных систем, пользовательскими интерфейсами, front-end web development.

Аннотация. В работе рассматривается применение методов аналитической геометрии для реализации механики рисования заклинаний в игре Cave Enchant. Описан алгоритм распознавания геометрических фигур,

учитывающий координатные преобразования и выпуклость многоугольников. Применённый метод минимизирует влияние дрожания руки игрока и обладает высокой вычислительной эффективностью, а также предоставляет возможности его использования в других аспектах разработки игр.

Ключевые слова: аналитическая геометрия, декартова система координат, алгоритм распознавания фигур, игровая механика, Cave Enchant, Unity, C#, Visual Studio, виртуальная реальность, интерактивное взаимодействие.

Введение. Разработка игр — сложный мультидисциплинарный процесс, состоящий из множества этапов, на каждом из которых математические методы играют важную роль. Для создания простых игр могут понадобиться базовые навыки программирования и проектирования, но углубленные знания в области математики позволяют значительно повысить качество и сложность реализуемых игровых элементов.

Аналитическая геометрия – это раздел геометрии, в котором геометрические фигуры и их свойства исследуются средствами алгебры [1]. В основе метода аналитической геометрии лежит метод координат, который каждому геометрическому соотношению ставит в соответствие некое уравнение, связывающее координаты фигуры или тела.

В игре «Cave Enchant», пользователь при помощи контроллеров виртуальной реальности может рисовать заклинания для сражения с противниками (рисунок 1). Для этой механики и используются методы аналитической геометрии. Несмотря на сложность данного математического раздела, в проекте были использованы его упрощенные методы, обеспечивающие требуемую функциональность.

Рисунок 1. Демонстрация механики

Игра «Cave Enchant» была создана на игровом движке Unity, программный код был написан на языке C# в среде разработки Visual Studio, 3D-модели были созданы в редакторе Blender.

Описание алгоритма. При нажатии игроком на триггер контроллера, перед ним появляется рабочая область, предназначенная для рисования простых геометрических фигур. В механике игры предусмотрено шесть заклинаний, каждому из которых соответствует определенная геометрическая фигура: треугольник, выпуклый и невыпуклый четырёхугольники, выпуклый и невыпуклый пятиугольники, а также пятиконечная звезда. Если нарисованная пользователем фигура совпадает с одной из предопределённых форм, то после отпускания триггера контроллера создается заклинание; в противном случае оно не активируется.

Реализация алгоритма. Когда игрок нажимает на курок контроллера, перед ним создаётся объект SpellInputSpace, представляющий собой плоскость. Локальная система координат этого объекта установлена так, что точки с локальной координатой z, равной 0, лежат в этой плоскости. Также из контроллера игрока создаётся луч, который сталкивается

с этой плоскостью и создаёт в месте столкновения точку HitObject. Так как эта точка лежит на данной плоскости, то её координата *z* в системе координат этой плоскости равна нулю. Таким образом, любое положение этой точки можно представить всего в двух координатах – *x* и *y* (рисунок 2).

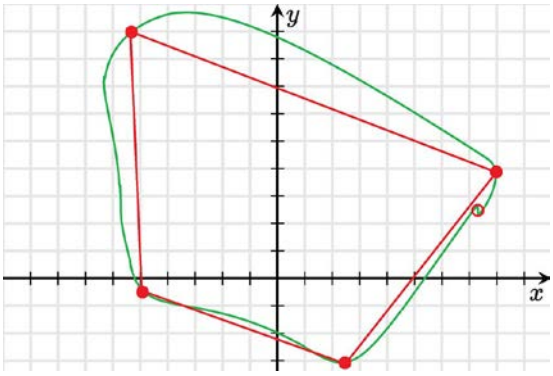

Рисунок 2. Визуализация алгоритма

Далее скрипт (рисунок 3) фиксирует величину угла, нарисованную игроком, и если она превышает заданное пороговое значение, то создаётся точка-угол.

```
private void CalculatingDelta()
{
    float hitDiff = hitDiffDelta;
    Vector2 hitDelta = hitObject.transform.position;
    if ((hitDelta - hitPos).sqrMagnitude > cornerSensitivity * cornerSensitivity)
    {
        hitDiffDelta = Mathf.Atan2(hitDelta.y - hitPos.y, hitDelta.x - hitPos.x);
        hitDiffSecond = hitDiffDelta - hitDiff;
    }
    float pointDistance = points.Count > 0 ? (points[points.Count - 1].transform.position - hit.point).sqrMagnitude : 999;
    if (Mathf.Abs(hitDiffSecond * Mathf.Rad2Deg) > 20 && pointDistance > cornerSensitivity*5)
    {
        hitDiffSecond = 0;
        points.Add(new SpellObject(Instantiate(pointPrefab)));
        points[points.Count - 1].transform.parent = spellSpace.transform;
        points[points.Count - 1].transform.position = hit.point;
        points[points.Count - 1].transform.parent = spellSpace.transform;
        if (points.Count > 2) Points();
    }
    hitPos = hitDelta;
}
```

Рисунок 3. Метод расчёта изменения угла.

Пусть ось *X* – множество всех точек, имеющих координату y, равную нулю в системе координат SpellInputSpace, а ось *Y* — множество всех точек, имеющих координату *x*, равную нулю в той же системе координат. Сначала определяется угол α между осью *X* и прямой, проходящей через точки с координатами HitObject в текущем и предыдущем кадре. Для этого используется метод Mathf.Atan2, который в отличие от стандартного метода Atan позволяет избежать ошибки, заключающейся в том, что период функции тангенса равен 180º [4]. Затем определяется, насколько быстро изменяется угол в текущем кадре, по формуле:

$$\Delta\alpha = \frac{\alpha_{\text{предыдущий кадр}} - \alpha_{\text{текущий кадр}}}{\Delta t},$$

где Δt – время, прошедшее между предыдущим и текущим кадром, взятое при помощи метода Time.DeltaTime, а Δα – скорость изменения угла.

Если значение Δα достаточно большое, то на месте HitObject ставится точка, обозначающая угол фигуры.

Поскольку движения руки игрока не могут быть абсолютно ровными и плавными, метод фиксирует изменение угла только в том случае, если HitObject прошёл достаточное расстояние. Также, когда игрок проводит прямую линию, удаляются лишние точки, созданные случайным дрожанием руки. Для этого рассматривается треугольник из трёх точек-углов. Если разность суммы длин двух его меньших сторон и длины наибольшей

стороны достаточно мала, или же если этот треугольник имеет угол близкий к 180º, то удаляется лишняя точка, и остаётся прямая линия. Данный скрипт представлен на рисунке 4.

```
private void Points()
{
    int i = points.Count - 2;
    int j=-1;
    float a = (points[i - 1].transform.position - points[i].transform.position).magnitude;
    float b = (points[i].transform.position - points[i+1].transform.position).magnitude;
    float c = (points[i - 1].transform.position - points[i+1].transform.position).magnitude;
    if (a == Mathf.Max(a, b, c) && b + c - a < a * filterSensitivity) j = i + 1;
    if (b == Mathf.Max(a, b, c) && a + c - b < b * filterSensitivity) j = i - 1;
    if (c == Mathf.Max(a, b, c) && b + a - c < c * filterSensitivity) j = i;
    if (j > -1)
    {
        Destroy(points[j].gameObject);
        points.RemoveAt(j);
        foreach (SpellObject line in lines)
        {
            Destroy(line.gameObject);
        }
        lines.Clear();
        DrawLines();
    }
}
```

Рисунок 4. Метод удаления лишних точек.

Далее, между точками рисуются линии. Линии и точки перекрашиваются в зависимости от того, какая фигура нарисована в данный момент.

Определение типа фигуры осуществляется с помощью метода координат. Для распознавания треугольника достаточно наличие трёх точек. Для определения остальных фигур необходимо проанализировать выпуклости многоугольника.

Многоугольник считается выпуклым, если для каждой прямой, содержащей одну из его сторон, все его вершины, не лежащие на этой прямой, расположены по одну сторону от него. В противном случае он является невыпуклым [2].

Определение пятиконечной звезды основано на следующем критерии: для каждой прямой, содержащей одну из её сторон, две вершины фигуры находятся по одну сторону от прямой, а одна вершина — по другую.

Как мы можем видеть на рисунке 5, эти утверждения верны. Например, на рисунке 5 (б) для невыпуклого четырёхугольника условие расположения вершин по одну сторону от прямой выполняется только для двух его сторон, тогда как для остальных двух — нет.

В случае пятиконечной звезды (рисунок 5 (в)) её характеристическое условие выполняется для каждой стороны: для каждой прямой, содержащей одну из сторон звезды, две вершины расположены по одну сторону, а одна вершина — по другую.

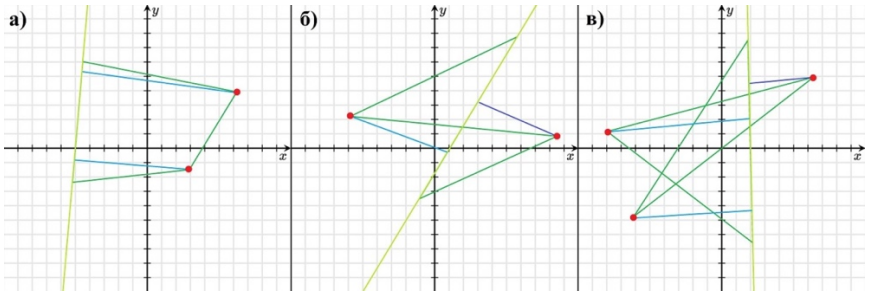

Рисунок 5. Положение вершин относительно прямой для а) выпуклого четырёхугольника, б) невыпуклого четырёхугольника, в) пятиконечной звезды.

Для того чтобы определить положение вершин относительно прямой, скрипт (рисунок 6) представляет каждую сторону многоугольника в виде уравнения прямой:

$$y = kx + b,$$

где k – тангенс угла между этой прямой и осью X, а b – смещение этой прямой вдоль оси Y.

```
case 5:
    float[] currLines5k = new float[5];
    float[] currLines5b = new float[5];
    bool pentagon = true;
    bool star = true;
    for (int i = 0; i < 5; i++)
    {
        currLines5k[i] = (points[i == 4 ? 0 : i + 1].transform.localPosition.y - points[i].transform.localPosition.y) /
            (points[i == 4 ? 0 : i + 1].transform.localPosition.x - points[i].transform.localPosition.x);
        currLines5b[i] = points[i].transform.localPosition.y - points[i].transform.localPosition.x * currLines5k[i];
        int p2 = 0;
        int[] sign = new int[3];
        for(int j =0;j<5;j++)
        {
            if (j == i || j == (i == 4 ? 0 : i + 1)) continue;
            sign[p2] = (int)Mathf.Sign(currLines5k[i] * points[j].transform.localPosition.x + currLines5b[i] - points[j].transform.localPosition.y);
            p2++;
        }
        Array.Sort(sign);
        if (sign[0] != sign[2]) pentagon = false;
        if (!((sign[0] == sign[1] && sign[1] != sign[2]) || (sign[0] != sign[1] && sign[1] == sign[2]))) star = false;
        if (!star && !pentagon) i = 5;
    }
    if (pentagon) spell = 2;
    else if (star) spell = 5;
    else spell = 4;
    break;
```

Рисунок 6. Алгоритм определения типа пятиугольника.

Коэффициент k можно найти как:

$$\mathrm{k} = \frac{y_2 - y_1}{x_2 - x_1},$$

где x_1, y_1, x_2, y_2 – координаты двух вершин, лежащих на данной прямой.

Коэффициент b можно найти, решив уравнение:

$$y_1 = \mathrm{k}x_1 + b, \qquad b = y_1 - kx_1.$$

Для определения положения точки относительно прямой подставляется координата x точки в полученное уравнение, после чего из координаты y этой точки вычитается вычисленное значения y. Знак полученной разности Δy определяет, по какую сторону от прямой находится данная точка (рисунок 7).

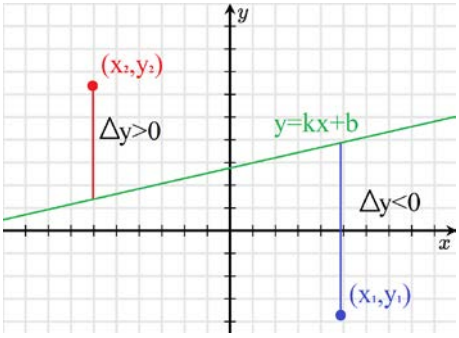

Рисунок 7. Положение точки относительно прямой

Существуют и другие способы определения выпуклости многоугольника [3]. Однако данный алгоритм был выбран, так как он позволяет не только определить выпуклость, но ещё и распознает пятиконечную звезду. Поскольку в рассматриваемом случае число сторон ограничено пятью, алгоритм остаётся малозатратным с точки зрения вычислительных ресурсов, так как при увеличении числа сторон он перестанет быть выполнимым.

Заключение. Результатом использования методов аналитической геометрии стало реализация механики рисования заклинаний в игре Cave Enchant. Данный подход обеспечил точное распознавание геометрических фигур, используемых для активации заклинаний, и эффективную обработку пользовательского ввода. Следует отметить, что эти методы находят широкое применение в других сферах разработки игр, включая

рендеринг и симуляцию физических процессов. Кроме того, они могут быть использованы для создания различных интерактивных механик, расширяя возможности игровой индустрии.

Список литературы

[1] Ласуков Владимир Васильевич, Томский политех., Математика 1, лекция 7, «Аналитическая геометрия», https://portal.tpu.ru/SHARED/l/LASUKOV/umkd/math/math_1_6_Lecture-07.pdf.

[2] Джон М. Ли, «Axiomatic Geometry», https://books.google.ca/books?id=9Z0xAAAAQBAJ&pg=PA155&source=gbs_toc_r&cad=4#v=onepage&q&f=false

[3] Хекберт Пауль (ред.), «Graphic Gems IV», 1994 г, «I.2 Testing the convexity of the polygon» (стр. 7-15), https://archive.org/details/isbn_9780123361554/page/7.

[4] Unity Scripting API, Документация скриптинга в юнити, https://docs.unity3d.com/6000.0/Documentation/ScriptReference/index.html.

PROCESSING AND RECOGNIZING GEOMETRIC PATTERNS IN GAME DATA

S. A. Bergovin
Student of the educational Institution "National Children's Technopark", student GUO «Yanka Kupala Gymnasium», Mosyr

A. Y. Salivon
Student of the educational Institution "National Children's Technopark", student GUO «Vysokoe school», Pinsk district

Ph.V. Usenko
Assistant of the Department of Engineering Psychology and Ergonomics of BSUIR, M.Sc

A. M. Prudnik
Associate Professor of Engineering Psychology and Ergonomics Department of BSUIR, Candidate of Technical Sciences, Associate Professor

Abstract. This paper discusses the application of analytic geometry methods to the implementation of spell drawing mechanics in the game Cave Enchant. An algorithm for recognizing geometric shapes that takes into account coordinate transformations and convexity of polygons is described. The method minimizes the influence of player's hand shaking and has high computational efficiency, as well as provides opportunities for its use in other aspects of game development.

Keywords: analytical geometry, Cartesian coordinate system, shape recognition algorithm, game mechanics, Cave Enchant, Unity, C#, Visual Studio, virtual reality.