ЛАБОРАТОРНАЯ РАБОТА №4
Коллизия по AABB (Axis-aligned minimum bounding box). Создание примитивного уровня.

Цель работы: добавить возможность взаимодействия и перемещения. Вывод уровня основанного на тайловой карте.

Задание: написать код для коллизии персонажа/объекта с границами экрана и последующей картой, вывода карты.

Среда работы: IDE Code::Blocks.

Axis-aligned minimum bounding box

Минимальный ограничивающий прямоугольник, выровненный по оси (или AABB) для данного набора точек, является его минимальным ограничивающим прямоугольником с учетом ограничения, заключающегося в том, что края прямоугольника параллельны (декартовым) координатным осям. Это декартово произведение N интервалов, каждый из которых определяется минимальным и максимальным значением соответствующей координаты для точек в S.

Минимальные ограничивающие рамки, выровненные по оси, используются для приблизительного определения местоположения рассматриваемого объекта и в качестве очень простого описания его формы. Например, в вычислительной геометрии и ее приложениях, когда требуется найти пересечения в наборе объектов, первоначальной проверкой являются пересечения между их минимально ограничивающими прямоугольниками (MBBs). Поскольку это обычно гораздо менее затратная операция, чем проверка фактического пересечения (поскольку для этого требуется только сравнение координат), она позволяет быстро исключить проверки пар, которые находятся далеко друг от друга или с ограничениями реализовывать коллизию в 2d и 3d пространстве.

Пример 1: Обычное столкновение AABB. Синяя рамка находится в начале движения. Зеленое поле - это место, где коробка должна быть в конце движения. Бирюзовая рамка показывает, где AABB разместит коробку после столкновения. Серый прямоугольник представляет собой статический (неподвижный) блок, который проверяется на столкновение. Это показывает нормальное столкновение. Рамка перемещается в ближайшую точку, в котором не работает столкновение. Это нормально и является ожидаемым результатом AABB.

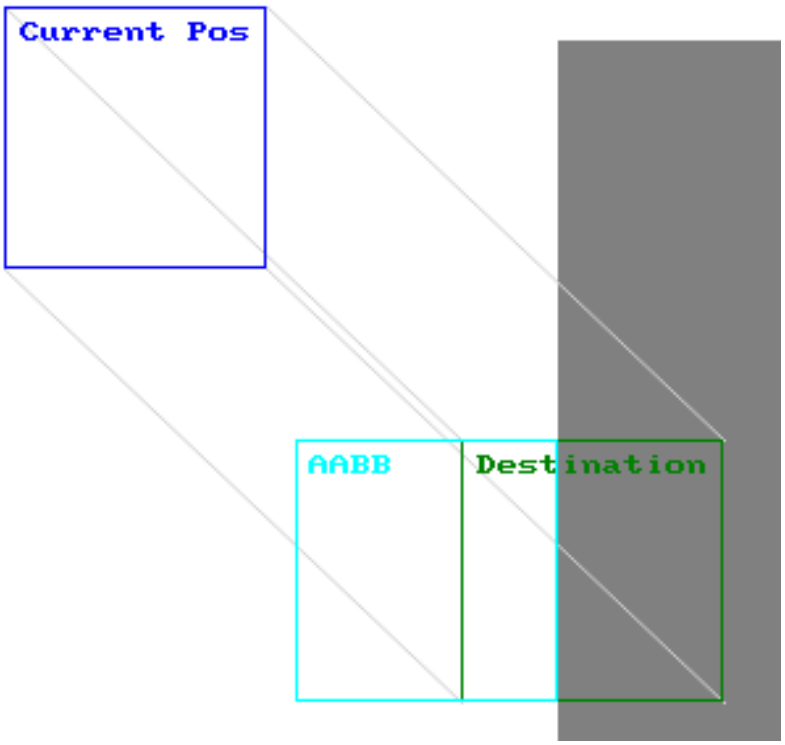

Рисунок 1 Первый пример работы AABB

Пример 2: Показывает аналогичное столкновение, когда пункт назначения находится дальше на противоположной стороне. Как видите, AABB разместила поле ответа на противоположной стороне блока. С логической точки зрения это не имеет смысла, так как оно волшебным образом прошло через объект.

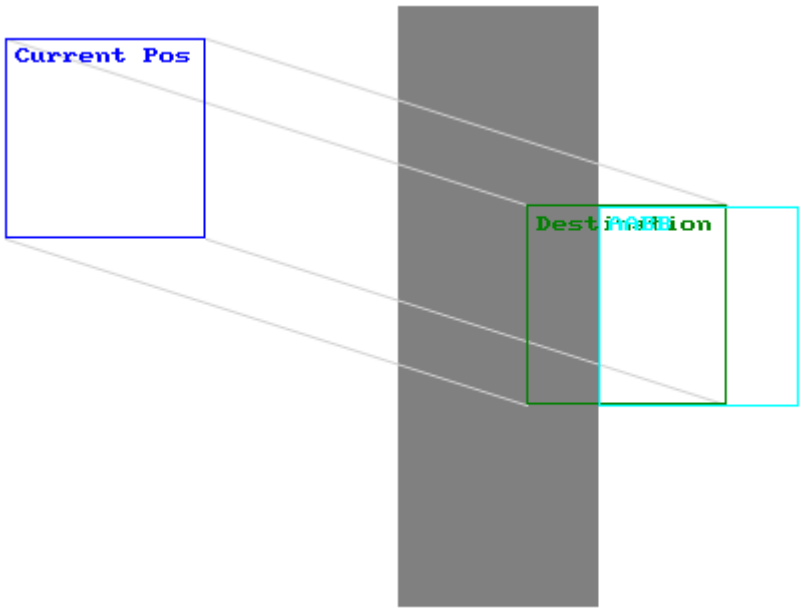

Рисунок 2 Второй пример работы AABB

Пример 3: Есть место назначения, которое находится за объектом. AABB будет предполагать, что столкновения не было, и движущаяся коробка будет двигаться через нее, как будто столкновения вообще не было. Так когда же возникают эти проблемы? Эти проблемы обычно возникают, когда объекты движутся быстро и/или программа работает с низкой частотой кадров. Чтобы этого избежать, нам нужно каким-то образом предсказать, где коробка окажется в следующем кадре.

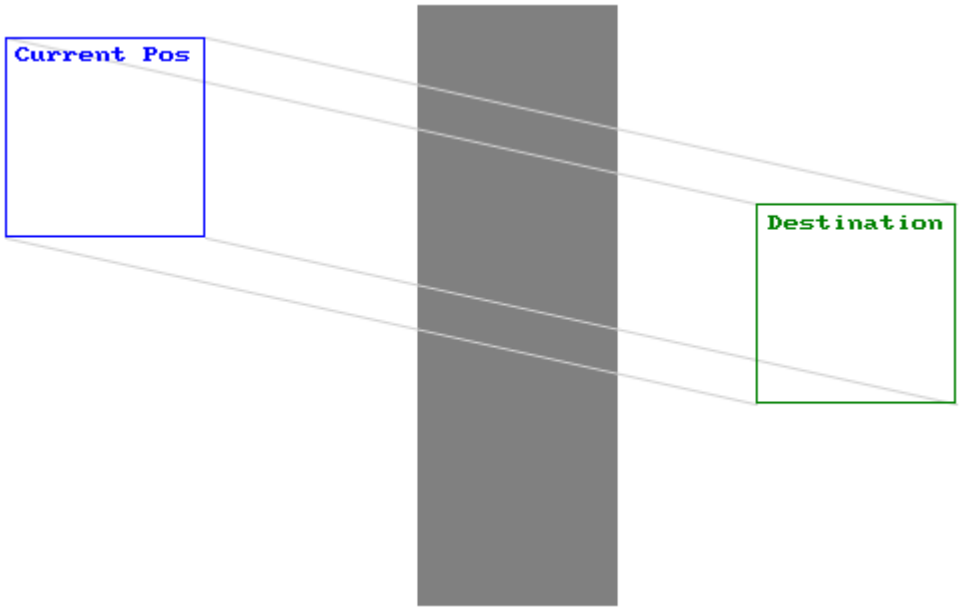

Рисунок 3 Третий пример работы AABB

В нашей реализации перемещения есть необходимые переменные:

```
typedef struct
{
   float x, y, //координаты нахождения в пространстве
   dx, dy; // первая производная по координате (скорость)
} Hero;
```

При определении структуры не хватает размеров примитива, их можно получать из функции вывода спрайт листа или добавить в структуру.

Для начала сделаем функцию ограничивающую передвижение в рамках экрана. Она будет выглядеть следующим образом:

```
void Cheek_ Collision(Hero *obj, bool directional) // первое – наш
//подвижный персонаж, переменная directional – для разделения проверки по
                                                                    //X и У
```

Далее необходимо проверить не выходит ли в новом кадре подвижный элемент за границы экрана:

```
if (obj->y<0) //если координата y в результате смещения меньше 0
{
obj->y=0; //поменять координату y на 0
}
```

И подобной конструкцией еще 3 раза пройтись по оставшимся краям экрана.

Ограничения AABB

Описанная реализация имеет некоторые ограничения, которые вы, возможно, уже поняли. К ним относятся:

- Не учитывает изменение размера (т.е. если размер коробки изменяется между кадров, после проверки на столкновение).
- Допускается только линейное движение. Если ваш движущийся объект движется по кругу, он не будет проверять, как он был повернут на кривой в момент столкновения.
- Позволяет двигаться только одной объекту (т.е. если два объекта движутся навстречу друг другу и сталкиваются). Это то, что здесь намеренно не приведенно, поскольку это начинает включать в себя многие физические концепции, такие как масса и сила.
- Код сделан изначально для 2D. К счастью, довольно легко преобразовать этот код для 3D. Пока вы хорошо понимаете концепцию, у вас не должно быть особых проблем с ним.
- Это пока только квадратные фигуры! Вы даже не можете повернуть их! Это очевидно, потому что название алгоритма как бы уже говорит об этом. Но если вы победили AABB, то вы можете быть готовы перейти на следующий уровень (например, SAT или GJK колизии).

Создание уровня

Создание уровня можно разделить на две части – дизайн уровня и его организация с технической точки зрения. Мы рассматриваем только с технической точки зрения.

Для создания уровня из тайлов создают карту уровня, где на каждый отдельно взятый тайл при выводе уровня на экран, по соответствию наносится свое изображение, а в последствии задаются и разные характеристики.

Представление карты также может быть разным: текстовый файл, файл XML разметки, определенный массив данных (один элемент отвечает за расположение, а несколько последующих за его параметры).

Перед загрузкой уровня файл/массив считывается и в соответствии с написанным циклом выводится его изображение.

Рисунок 4 Пример фрагмента уровня с границами тайлов

Для упрощения задания уровня пока не будем выводить его разметку в отдельный файл.

Как пример задания внутри программы через массив строк:

```
string TileMap[] = {
/12345678901234567890123456789012/
"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB", //1
"BBBBBB                    B       B", //2
"B                         B       B", //3
"B                         B       B", //4
"B                         B       B", //5
"B                         B       B", //6
"BB                        B       B", //7
"B                         B       B", //8
"B                         B       B", //9
"B                                 B", //10
"B                                 B", //11
"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB" //12
};
```

Рисунок 5 Фрагмент кода разметки уровня через массив строк (элементы массива дополнительно пронумерованы)

Дальше необходимо его вывести:

…

```
const float tsize = 40; //в этом варианте размер тайла задан жестко
for (int i=0; i<H; i++) //чтение массива по вертикали
{
   for (int j=0; j<W; j++) // чтение массива по горизонтали
```

```
{
   if (TileMap[i][j]=='B') //проверка содержащегося элемента
                                //разметки в массиве
      *здесь определяем массив с вершинами тайлов*
      glVertexPointer(2,GL_FLOAT, 0, *имя массива с вершинами*);
      glEnableClientState(GL_VERTEX_ARRAY);

      glColor3f(r,g,b); //цвет тайла (сделайте зависимым от его
                                    положения в массиве разметки)
      glDrawArrays(GL_QUADS,0,4);

      glDisableClientState(GL_VERTEX_ARRAY);
   }
   if (TileMap[i][j]==' ') continue; //пустые элементы разметки
                                                просто игнорируем
…
```

Соединение коллизии и уровня

Последние что осталось соединить проверку коллизии относительно тайлов уровня. Для этого необходимо проверять соседние тайлы, и в случае обнаружения определять необходимость коллизии.

```
for (int i=obj->y/tsize; i<(obj->y+80)/tsize; i++)  //проверка по вертикале
   {    //относительно текущей координаты (80 – размер персонажа по
                                                        //вертикале)
      for (int j=obj->x/tsize; j<(obj->x+80)/tsize; j++) //проверка по
//горизонтали (80 – размер персонажа по горизонтали (квадратный))
      {
```

И на каждое направление необходимо по своей проверке и определение итоговой позиции, и параметров. Как пример – проверка по вертикале:

```
if (obj->dy<0 && directional==0) //условие для проверки
            {
               obj->y=i*tsize+tsize; //итоговая координата персонажа
               if (obj->y==i*tsize+tsize) obj->OnGround=1;
                                                //дополнительные
   }                        //параметры (переменная условия для прыжка)
```

В процессе может возникнуть ошибка – при одновременной проверке по вертикале и горизонтали, из-за этого персонажа будет выбрасывать за

границы. Для этого необходимо разграничить проверку по вертикале и горизонтали.

Дальше остается только добавлять вариации от пересечения с различными тайлами карты, как исчезающие платформы или отдельные пространства с другим набором свойств, влияющих на передвижение персонажа. Как пример тайлы воды в которых передвижение замедляется, или платформы дающие увеличенный коэффициент для прыжка.