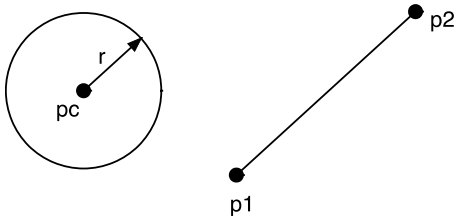
「　ベクトルの利用２　」

東京情報クリエイター

ベクトルの利用２

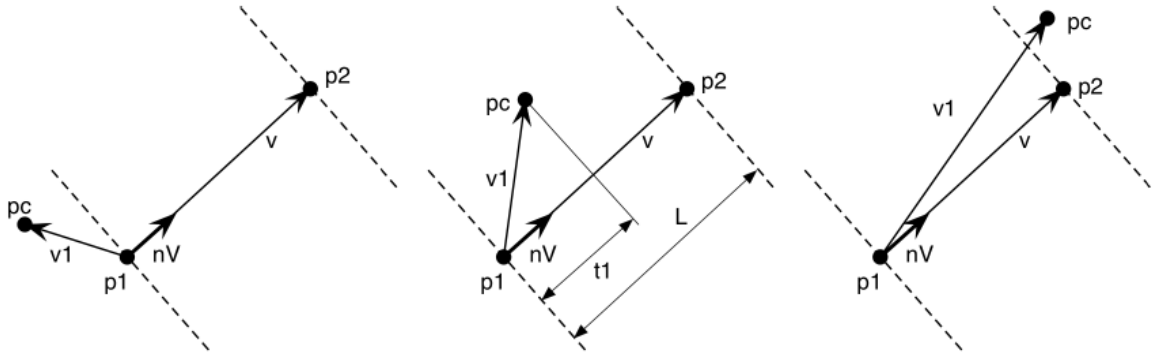
◆円と線分の衝突判定

円の中心を $\boldsymbol{P_c}$、円の半径を $\boldsymbol{r}$、線分の端点を $\boldsymbol{P_1}\,\boldsymbol{P_2}$ と定義します。



$\boldsymbol{L}$ と $\boldsymbol{t_1}$ の比を $\boldsymbol{t}$ とすると次の３つの場合に分けられます。

$\boldsymbol{t<0}$　　$\boldsymbol{0<t<1}$　　$\boldsymbol{t>1}$



$\boldsymbol{t=1}$ のとき、$\boldsymbol{t_1=L}$ 、比をとれば、$\boldsymbol{t=t_1/L}$ となります。

(1)円の中心位置　$\boldsymbol{P_c}$ が $\boldsymbol{P_1}$ から $\boldsymbol{P_2}$ の間にある場合

$\boldsymbol{t}$ は、次のように求めることができます。

$\vec{\boldsymbol{v}}=$　　$\overrightarrow{\boldsymbol{v_1}}=$　　$\overrightarrow{\boldsymbol{nv}}=$

$\boldsymbol{L}=$　　$\boldsymbol{t_1}=$

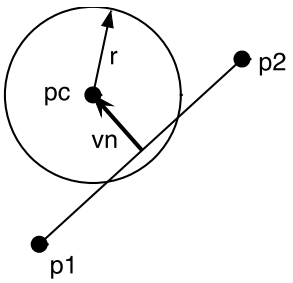
$\boldsymbol{t}=$

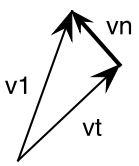
衝突の条件は、

① $\boldsymbol{t}$ の値が、$\boldsymbol{0}$ から $\boldsymbol{1}$ の間にある。

②円の中心位置と線分の最短距離が、円の半径以下にある。

最短距離は、次のように考えると簡単です。





$\overrightarrow{\boldsymbol{v_1}}=$

$\overrightarrow{\boldsymbol{v_t}}=$

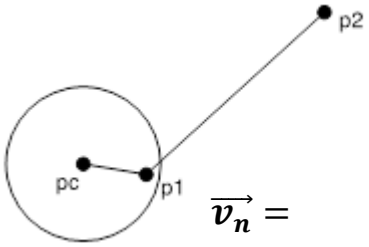
$\overrightarrow{\boldsymbol{v_n}}=$

「　ベクトルの利用２　」

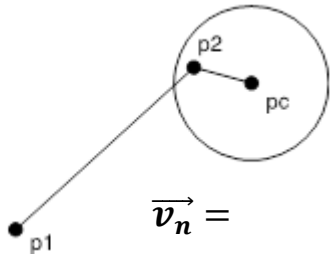
東京情報クリエイター

(2)円の中心位置　P_c が P_1 から P_2 の間にない場合

P_1 または P_2 と円の中心 P_c の距離が最短距離となるので、それぞれの距離を半径と比較すれば、衝突しているかどうかがわかります。



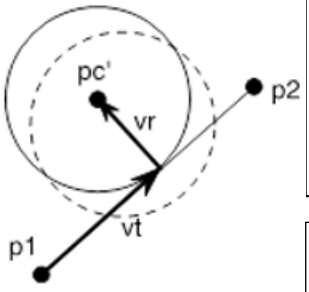
$\overrightarrow{v_n} =$



$\overrightarrow{v_n} =$

衝突判定は、これで終了ですが、ついでに、衝突していた場合に、円が線分と接する位置まで戻すには、どうしたらよいか考えてみます。

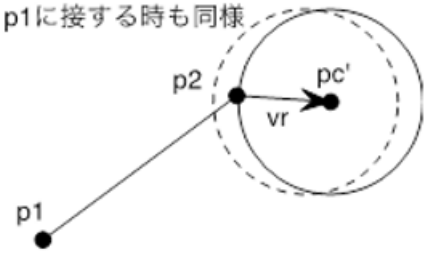
先ほど求めた、$\overrightarrow{v_t}$ $\overrightarrow{v_n}$ を使用して、衝突後の中心点 P'_c を求めることができます。



$\overrightarrow{v_r} =$

$P'_c =$

$=$



$\overrightarrow{v_r} =$

P_2の外

P_1の外

◆演習問題1

以上のことを利用し、円と線分の衝突判定を行うプログラムを作成しましょう。

MyLibrary の中に Collision2D クラスを作成し「Circle_Segment」メソッドを作成しなさい。戻り値は bool とする。

・ライブラリ名[MyLibrary]－クラス名[Collision2D.cs]：**衝突判定用**

・「Circle_Segment」メソッドを作成　戻り値は bool とする。

円の中心を戻す場合と戻さず判定のみの場合の 2 つを作成。

・このクラスに随時判定のメソッドを追加していく。

・プロジェクト名[Math03Ex04]をテンプレートで作成する。

「　ベクトルの利用２　」

東京情報クリエイター

[Collision2D.cs]円と線の衝突(円の中心を戻す)

```
namespace MyLibrary

{
    public class Collision2D
    {
        // <summary>
        // 円と線の衝突判定（境界円の戻りあり）
        // </summary>
        // <param name="p1">線分始点</param>
        // <param name="p2">線分終点</param>
        // <param name="center">円の中心</param>
        // <param name="radius">円の半径</param>
        public static bool Circle_Segment
                (ref Vector2 pc, float r, Vector2 p1, Vector2 p2)
        {
            Vector2 v = p2 - p1;
            float r2 = r * r;
            //p1 の外側にあって境界円より中心までの距離が半径以下か？
            Vector2 v1 = pc - p1; //中心から p1 までのベクトル

            //中心から線分に下した点と p1 までの距離の比
            float t = Vector2.Dot(v, v1) / Vector2.Dot(v, v);
            if ((t < 0) && (v1.LengthSquared() <= r2))
            {
                pc = p1 + r * Vector2.Normalize(v1);//中心を移動
                return true;
            }

            //p2 の外側にあって境界円より中心までの距離が半径以下か？
            Vector2 v2 = pc - p2;   //中心から p2 までのベクトル
            if ((1 < t) && (v2.LengthSquared() <= r2))
            {
                pc = p2 + r * Vector2.Normalize(v2);//中心を移動
                return true;
            }

            //境界円の中心が p1p2 の間にあって線分に接触している場合
            Vector2 vn = v1 - v * t; //法線方向のベクトル
            //p1p2 線分の間
            if ((0 <= t) && (t <= 1) && (vn.LengthSquared() <= r2))
            {
                pc = p1 + v * t + r * Vector2.Normalize(vn); //中心点を移動
                return true;
            }
            return false;
        }
    }
}
```

[Collision2D.cs]円と線の衝突(判定のみ)

```
// <summary>
// 円と線の衝突（判定のみで　境界円の戻りなし）
// </summary>
// <param name="p1">線分始点</param>
// <param name="p2">線分終点</param>
// <param name="center">境界円中心</param>
// <param name="radius">境界円半径</param>
// <returns></returns>
public static bool Circle_Segment(Vector2 pc, float r, Vector2 p1, Vector2 p2)
{
    float r2 = r * r;
    Vector2 v = p2 - p1;
    Vector2 v1 = pc - p1;
    float t = Vector2.Dot(v, v1) / Vector2.Dot(v, v);

    //p1の外側で円の半径内
    if (t < 0 && v1.LengthSquared() <= r2) return true;

    //p2の外側で円の半径内
    Vector2 v2 = pc - p2;
    if (1 < t && v2.LengthSquared() <= r2) return true;
    //p1p2間で円の半径内
    Vector2 vn = v1 - t * v; //線分の法線ベクトル
    if (0 < t && t < 1 && vn.LengthSquared() <= r2) return true;
    return false;
}
```

[MyDrawClass.cs]

```
//円と線分の交差判定
namespace Math03Ex04
{
    public class MyDrawClass:Draws,IDraws
    {
        //フィールドは必要に応じて記述
        InputState input;
        Vector2 p0, p1, p2, p3;   //★修正
        float r;    //★追加

        //コンストラクタは初期設定などを記述
        public MyDrawClass()
        {   ☆変更なし   }

        //図形データの入力処理
        public void InputData()
        {
                :  //略
            List<PointF> p = new List<PointF>();
            p = input.GetPoint(2, "線分の端点入力");  //★修正　線分の端点：2つ入力

            //入力キャンセル処理（入力点数が0のときで判断する）されたら以下を実行しない
            if (p.Count == 0) {   ☆変更なし   }

            p0 = new Vector2(p[0].X, p[0].Y);  //線分の端点1
            //★以降追加
            p1 = new Vector2(p[1].X, p[1].Y);  //線分の端点2
            Line(p0, p1);
            Render(); //線分1の描画；
```

「　ベクトルの利用２　」

東京情報クリエイター

```
            p = input.GetPoint(2, "円の中心と円周上の点を入力");
            if (p.Count == 0) { Clear(); return; }//入力キャンセル処理

            p2 = new Vector2(p[0].X, p[0].Y);   //円の中心
            p3 = new Vector2(p[1].X, p[1].Y);   //円周上の点
            r = (p3 - p2).Length();   //円の半径
            Circle(p2, r);  //中心 p2 半径 r の円

            //交差判定
            if (Collision2D.Circle_Segment(ref p2, r, p0, p1))
            {
                Text(new Vector2(-200, 100), "交差したので中心を移動します。");
                Dot(p2);
                SetColor(Color.Red); //色を変えると以降その色のまま
                Circle(p2, r);
            }
            else
            {
                Text(new Vector2(-200, 100), "交差してません。");
            }
            Render();
        }

        //☆以降変更なし
    }
}
```

「　ベクトルの利用２　」

東京情報クリエイター

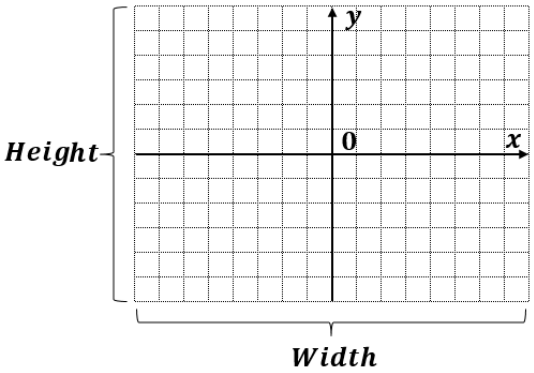
◆演習問題2

円を動かして、アニメーションを付けてみましょう。
・プロジェクト名[Math03Ex04]に処理を追加します。

円を動かすには、「円の中心を速度分だけ動かす→動かした先で円を描く」これを繰り返すことできますね。

$Vector2\ mov;$　**//初速度**
$mov = new\ Vector2(1,2);$　**//初速度の値**
$P_2 += mov;$　　**//P_2 が円の中心**

ここで、円を動かしていくと、表示ウィンドウから矩形がはずれます。そこで、表示ウィンドウに衝突したら、速度の方向を逆にします。
表示ウィンドウと座標の関係は左図のようになっています。ここに、円を配置して、どのような状態になったときに、速度の方向を逆にするか考えてみましょう。

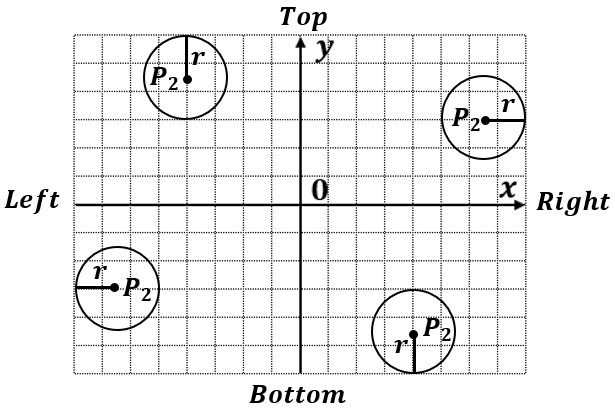


$Left =$

$Right =$

$Top =$

$Bottom =$



速度の x 方向を逆にするのは、

速度の y 方向を逆にするのは、

次に、線分を衝突したときに、円が線分に反射するようにします。反発の処理は、最初にVector2.csにベクトルの反射を求める「Reflct」を追加します。

```
//Vector2.cs に追加
// <summary>
// ベクトルの反射を求める。
// 入射ベクトルと法線ベクトルから反射ベクトルを求めます。
// </summary>
// <param name="vector">入射ベクトル</param>
// <param name="normal">法線ベクトル</param>
// <returns>反射ベクトル</returns>
public static Vector2 Reflect(Vector2 vector, Vector2 normal)
{
        float L = Vector2.Dot(-normal, vector);
        vector += 2 * L * normal;
        return vector;
}
```

「　ベクトルの利用２　」

東京情報クリエイター

MyDrawClass.cs には、以下のように記述します。ここで、IntersectionFlag は、衝突判定用のフラグです。

```
//反射
if (IntersectionFlag)
{
    Vector2 LineVec = p1 - p0;
    Vector2 normal = Vector2.Normalize(new Vector2(LineVec.Y, -LineVec.X));
    mov = Vector2.Reflect(mov, normal);
}
```

◆2つの線分の交点を求める。

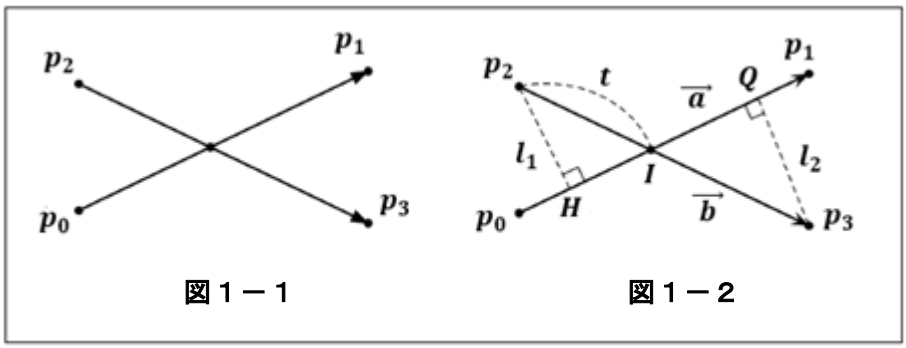
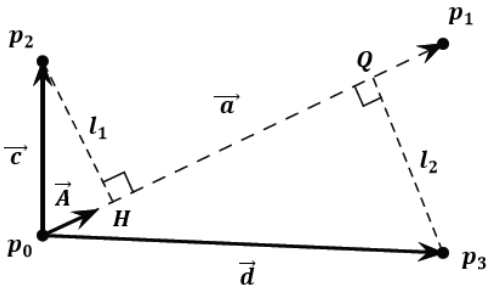


図１－１　　図１－２

図１－１のような平行でない 2 つの線分において交点を求めるには、図１－２のように $\overrightarrow{p_0p_1}$ を $\vec{a}$、$\overrightarrow{p_2p_3}$ を $\vec{b}$、交点を I、p_2 から $\vec{a}$ に下ろした垂線との交点を H とその長さ l_1、同様に p_3 から下ろした点 Q とその長さ l_2、p_2p_3 と p_2I の比（$p_2p_3 : p_2I$）を t とします。

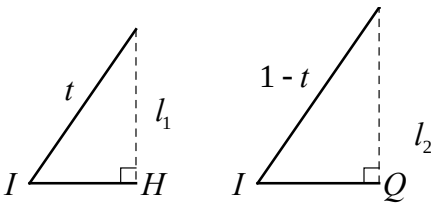
l_1 は$\overrightarrow{p_0p_2}$ を $\vec{c}$ とし、$\vec{a}$ を正規化したベクトルを $\vec{A}$ とすれば、外積によって求めることができます。同様に l_2 も求めることができます。（ここで l_1, l_2 は長さなので絶対値を取る。）



$l_1 =$　　$l_2 =$

$l_1 =$　　$l_2 =$

次に、t を求めます。$l_1 : l_2 = t : (1 - t)$ より



整理すると、

$t =$

t が求まれば、p_2 から I へのベクトルを求めることができるから、$\overrightarrow{p_2I} =$

したがって、交点 I は次のように求められます。　　$I =$

「　ベクトルの利用２　」

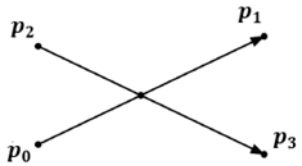
東京情報クリエイター

【練習問題】

次の点を通る2つの線分の交点をベクトルを使用して求めなさい。

線分１：$\boldsymbol{p_0(5,\ 5)\ p_1(15,11)}$

線分２：$\boldsymbol{p_2(5,\ 10)\ p_3(15,6)}$



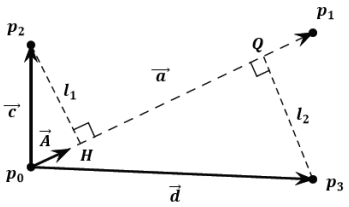
$\boldsymbol{p_0p_1}$, $\boldsymbol{p_0p_2}$, $\boldsymbol{p_2p_3}$, $\boldsymbol{p_0p_3}$ のベクトルを求める。

$\overrightarrow{\boldsymbol{p_0p_1}} = \vec{\boldsymbol{a}} = \boldsymbol{p_1 - p_0} =$

$\overrightarrow{\boldsymbol{p_0p_2}} = \vec{\boldsymbol{c}} = \boldsymbol{p_2 - p_0} =$

$\overrightarrow{\boldsymbol{p_2p_3}} = \vec{\boldsymbol{b}} = \boldsymbol{p_3 - p_2} =$

$\overrightarrow{\boldsymbol{p_0p_3}} = \vec{\boldsymbol{d}} = \boldsymbol{p_3 - p_0} =$



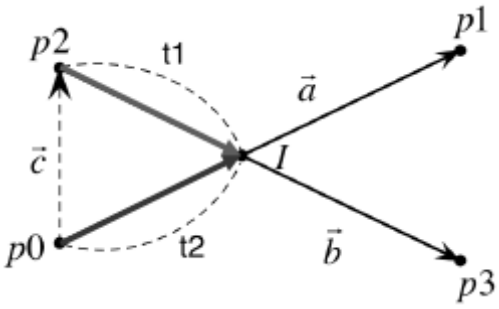
$\overrightarrow{\boldsymbol{p_2p_3}}$ の交点までの比 $\boldsymbol{t}$ を求める。

$$\boldsymbol{t} = \frac{|\vec{a}\times\vec{c}|}{|\vec{a}\times\vec{c}|+|\vec{a}\times\vec{d}|} =$$

したがって、交点 $\boldsymbol{I}$ の座標値は

$\boldsymbol{I = p_2 + t\overrightarrow{b}} =$

【線分同士の衝突判定】



交点までの比を $\boldsymbol{t}$ として、それぞれの線分の始点からの比を $\boldsymbol{t_1}$ $\boldsymbol{t_2}$ します。

「**2 つの線分の交点を求める**」を利用すると、それぞれの端点がそれぞれの線分の間に来る時が交差している時です。

よって、**交差する条件は**

「　　」

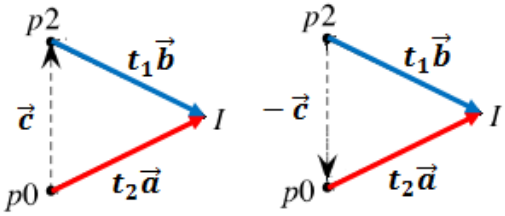
「　　」

ここで、$\boldsymbol{t_1}$ $\boldsymbol{t_2}$ は、以下のように求められます。

$\overrightarrow{\boldsymbol{c}} + \boldsymbol{t_1}\overrightarrow{\boldsymbol{b}} = \boldsymbol{t_2}\overrightarrow{\boldsymbol{a}}$　　$\boldsymbol{t_2}\overrightarrow{\boldsymbol{a}} \times \overrightarrow{\boldsymbol{a}} = \boldsymbol{0}$

$-\overrightarrow{\boldsymbol{c}} + \boldsymbol{t_2}\overrightarrow{\boldsymbol{a}} = \boldsymbol{t_1}\overrightarrow{\boldsymbol{b}}$　　$\boldsymbol{t_1}\overrightarrow{\boldsymbol{b}} \times \overrightarrow{\boldsymbol{b}} = \boldsymbol{0}$　より、

$\boldsymbol{t_2}\overrightarrow{\boldsymbol{a}} \times \overrightarrow{\boldsymbol{a}} = \boldsymbol{t_1}\overrightarrow{\boldsymbol{b}} \times \overrightarrow{\boldsymbol{b}}$　①



①式に、$\overrightarrow{\boldsymbol{c}} + \boldsymbol{t_1}\overrightarrow{\boldsymbol{b}} = \boldsymbol{t_2}\overrightarrow{\boldsymbol{a}}$ を代入すると、

よって、$\boldsymbol{t_1} =$　　$\boldsymbol{I} =$

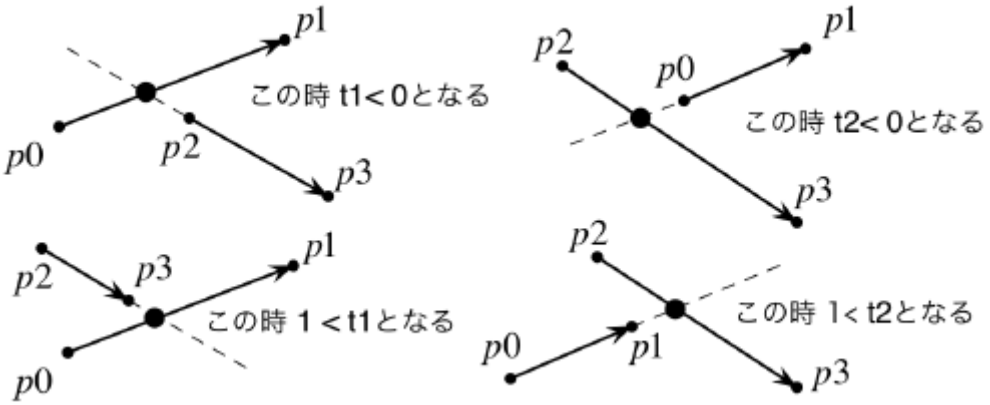
「　ベクトルの利用２　」

東京情報クリエイター

次に①式に、$-\overrightarrow{c}+t_2\overrightarrow{a}=t_1\overrightarrow{b}$　を代入すると、

よって、$t_2=$　　$I=$

また、次のようなときは、交差していないときです。



◆演習問題3

上記のことを利用し、Collistion2D クラスに「LineIntersection」メソッドを作成しなさい。

- **メソッドは交点を求めるものと交点を求めないものの２つ作成する。**
- **戻り値は bool とする。戻り値については、ref（参照）を使用する。**
- **「円と線分の衝突判定」を参考にしながら作成すること。**

プロジェクト名[Math03Ex05]をテンプレートで新規に作成する。
描画が完成したら、アニメーションもつけてみましょう。

Math03Ex04 アニメーション：[Math03Ex04]を修正・追加

【MyDrawClass.cs】

```
namespace Math03Ex04
{
    public class MyDrawClass:Draws,IDraws
    {
        InputState input;
        Vector2 p0, p1, p2, p3; //★修正
        float r;  //★追加

        //★★追加
        Vector2 mov;
        bool IntersectionFlag;

        //コンストラクタは初期設定などを記述
        public MyDrawClass()
        {
            input = new InputState();    //入力操作の定義
            input.MouseOn();             //マウス入力を有効にする

            //★★追加
            IntersectionFlag = false;    //交差フラグ
            mov = new Vector2(1, 2);     //初速度
        }

        //図形データの入力処理
        public void InputData()
        {
              //変更なし
        }

        //円を動かします
        public void Update()
        {
            if ((p2.X - r < -Width / 2) || (Width / 2 < p2.X + r)) mov.X *= -1;
            if ((p2.Y - r < -Height / 2) || (Height / 2 < p2.Y + r)) mov.Y *= -1;
            p2 += mov;

            //交差判定
            IntersectionFlag = Collision2D.Circle_Segment(ref p2, r, p0, p1);

            //反射
            if (IntersectionFlag)
            {
                Vector2 LineVec = p1 - p0;
                Vector2 normal = Vector2.Normalize(new Vector2(LineVec.Y, -LineVec.X));
                mov = Vector2.Reflect(mov, normal);
            }
        }

        //アニメーション時に更新する描画処理
        public void Draw()
        {
            SetColor(Color.Blue);
            Line(p0, p1);
            SetColor(Color.Red);
            Circle(p2, r);
        }
    }
}
```

「　ベクトルの利用２　」

東京情報クリエイター

[Collision2D.cs に追加]　線分と線分(交点は求めない)

```
// <summary>
// 線分同士の交差（交点は求めない）
// </summary>
// <param name="p0">線分１の始点</param>
// <param name="p1">線分１の終点</param>
// <param name="p2">線分２の始点</param>
// <param name="p3">線分２の終点</param>
// <returns></returns>
public static bool LineIntersection
    (Vector2 p0,Vector2 p1,Vector2 p2,Vector2 p3)
{
    Vector2 a = p1 - p0;
    Vector2 b = p3 - p2;
    Vector2 c = p2 - p0;  //線分の始点同士を結ぶベクトル

    float ab = Vector2.Cross(a, b);
    float t1 = Vector2.Cross(c, a) / ab ;   //p2→t  のｂベクトルに対する比
    float t2 = Vector2.Cross(c, b) / ab ;   //p0→t  のａベクトルに対する比

    if ((0 < t1 && t1 < 1) && (0 < t2 && t2 < 1))
    {
        return true;
    }
    return false;
}
```

[Collision2D.cs に追加]　線分と線分(交点も求める)

```
// <summary>
// 線分同士の交差（交点も求める）
// </summary>
// <param name="p0">線分１の始点</param>
// <param name="p1">線分１の終点</param>
// <param name="p2">線分２の始点</param>
// <param name="p3">線分２の終点</param>
// <param name="Intersection">交点</param>
// <returns></returns>
public static bool LineIntersection
    (Vector2 p0, Vector2 p1, Vector2 p2, Vector2 p3, ref Vector2 Intersection)
{
    Vector2 a = p1 - p0;
    Vector2 b = p3 - p2;
    Vector2 c = p2 - p0;  //線分の始点同士を結ぶベクトル

    float ab = Vector2.Cross(a, b);
    float t1 = Vector2.Cross(c, a) / ab;      //p2→t  のｂベクトルに対する比
    float t2 = Vector2.Cross(c, b) / ab;        //p0→t  のａベクトルに対する比

    Intersection = p0 + t2 * a;  //交差しない場合も交点を求める。

    if ((0 < t1 && t1 < 1) && (0 < t2 && t2 < 1))
    {
        return true;
    }
    return false;
}
```

[MyDrawClass.cs]　描画

```
//線分同士の交差
namespace Math03Ex05
{
    public class MyDrawClass:Draws,IDraws
    {
        InputState input;
        Vector2 p0, p1, p2, p3, Intersection;  //★修正

        public MyDrawClass()
        {
            input = new InputState();    //入力操作の定義
            input.MouseOn();             //マウス入力を有効にする
            Intersection = new Vector2();//★追加　交点
        }

        public void InputData()
        {
            Init(); //属性の初期化（色・太さ・文字高さを既定値にする）
            Clear();//画面をクリアします。(座標軸はクリアしない)
            //マウス入力
            List<PointF> p = new List<PointF>();
            p = input.GetPoint(2, "線分１の端点を入力");  //★修正
            if (p.Count == 0)
            {
                Clear(); //画面をクリアします。（ここでは入力点をクリアしている）
                return;
            }

            p0 = new Vector2(p[0].X, p[0].Y);

            //★以降を追加
            p1 = new Vector2(p[1].X, p[1].Y);
            Line(p0, p1);
            Render();   //線分１の描画

            p = input.GetPoint(2, "線分２の２点を入力");
            if (p.Count == 0) return;  //キャンセルされたら以下を実行しない

            p2 = new Vector2(p[0].X, p[0].Y);
            p3 = new Vector2(p[1].X, p[1].Y);
            Line(p2, p3);
            Render();   //線分２の描画

            //交差判定
            if (Collision2D.LineIntersection(p0, p1, p2, p3, ref Intersection))
            {
                Text(new Vector2(-200, 100), "交差しました。");
                SetColor(Color.Red);
                Dot(Intersection);
            }
            else
            {
                Text(new Vector2(-200, 100), "交差してません。");
            }
            Render();
        }

        public void Update()  { }

        public void Draw()  { }
    }
}
```

「　ベクトルの利用２　」

東京情報クリエイター

[MyDrawClass.cs] 描画の[MyDrawClass.cs]に追加・修正（アニメーション）

```
    //★★フィールドに追加
    Vector2 mov0, mov1, mov2, mov3;
    bool IntersectionFlag; //交差フラグ

   //★★コンストラクタに追加
   IntersectionFlag = false;
   mov0 = new Vector2(1, 2);    //p0
   mov1 = new Vector2(1, 1);    //p1
   mov2 = new Vector2(-1, -1);  //p2
   mov3 = new Vector2(1, -1);   //p3

   public void InputData()
   {    //★★変更なし   }

    //★★座標点の移動処理を追加
    private void AreaMove(ref Vector2 position, ref Vector2 mov)
    {
        if ((position.X < -Width / 2) || (Width / 2 < position.X)) mov.X *= -1;
        if ((position.Y < -Height / 2) || (Height / 2 < position.Y)) mov.Y *= -1;

        position += mov;
    }

    public void Update()
    {
        AreaMove(ref p0, ref mov0);    //p0 の移動範囲
        AreaMove(ref p1, ref mov1);    //p1 の移動範囲
        AreaMove(ref p2, ref mov2);    //p2 の移動範囲
        AreaMove(ref p3, ref mov3);    //p3 の移動範囲

        //交差判定
        IntersectionFlag = Collision2D.LineIntersection(p0, p1, p2, p3, ref Intersection);
    }

    public void Draw()
    {
        Line(p0, p1);
        Line(p2, p3);

        if (IntersectionFlag)
        {
            SetColor(Color.Red);
            Text(new Vector2(-200, 100), "交差しました。");
            Dot(Intersection);
        }
        else
        {
            SetColor(Color.Black);
            Text(new Vector2(-200, 100), "交差してません。");
        }
    }
```