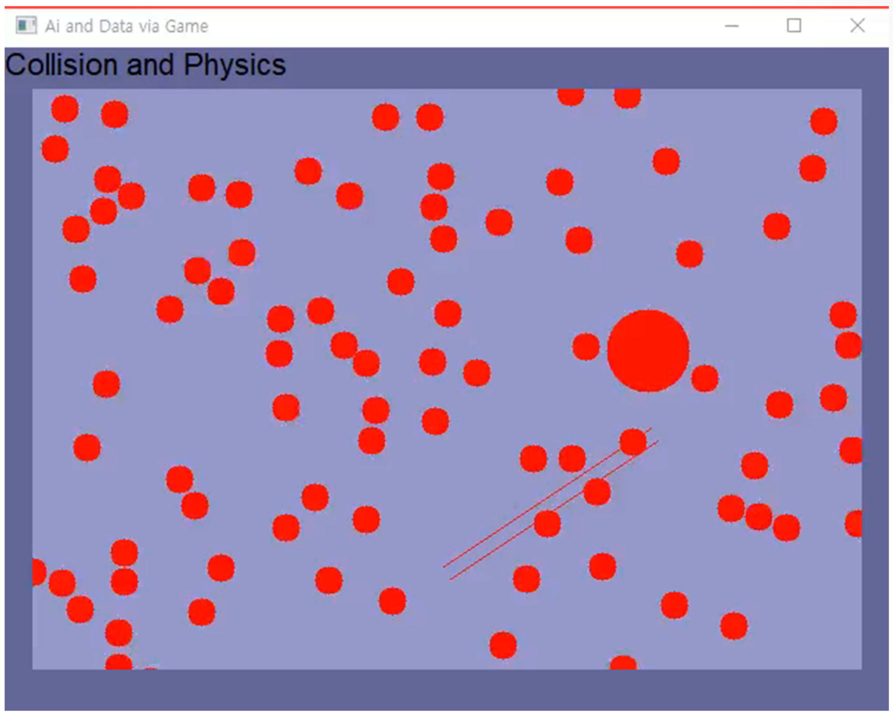
3. Collision & Physics



KhuGleSprite.h/ KhuGleSprite. cpp

```
class CKhuGleSprite : public CKhuGleComponent {
  …
  int m_bCollided;
};



void CKhuGleSprite::Render() {
  if(!m_Parent) return;

  CKhuGleLayer *Parent = (CKhuGleLayer *)m_Parent;
  KgColor24 SaveColor = m_fgColor;

  if(m_bCollided) m_fgColor = KG_COLOR_24_RGB(255, 255, 0);
  …
  m_fgColor = SaveColor;
}
```

Main.cpp (1)

```
#include "KhuGleWin.h"
#include <iostream>

class CCollision : public CKhuGleWin {
public:
  CKhuGleLayer *m_pGameLayer;

  CKhuGleSprite *m_pCircle1;
  CKhuGleSprite *m_pCircle2;
  CKhuGleSprite *m_pLine;
  CKhuGleSprite *m_pNewCircle[100];

  CCollision(int nW, int nH);
  void Update();

  CKgPoint m_LButtonStart, m_LButtonEnd;
  int m_nLButtonStatus;
};
```

Main.cpp (2)

```
CCollision::CCollision(int nW, int nH) : CKhuGleWin(nW, nH) {
  m_nLButtonStatus = 0;

  m_Gravity = CKgVector2D(0., 98.);
  m_AirResistance = CKgVector2D(0.1, 0.1);

  m_pScene = new CKhuGleScene(640, 480, KG_COLOR_24_RGB(100, 100, 150));
  m_pGameLayer = new CKhuGleLayer(600, 420, KG_COLOR_24_RGB(150, 150, 200),
    CKgPoint(20, 30));
  m_pScene->AddChild(m_pGameLayer);

  m_pCircle1 = new CKhuGleSprite(GP_STYPE_ELLIPSE, GP_CTYPE_DYNAMIC,
    CKgLine(CKgPoint(30, 30), CKgPoint(90, 90)),
    KG_COLOR_24_RGB(255, 0, 0), true, 100);
  m_pCircle2 = new CKhuGleSprite(GP_STYPE_ELLIPSE, GP_CTYPE_DYNAMIC,
    CKgLine(CKgPoint(70, 70), CKgPoint(130, 130)),
    KG_COLOR_24_RGB(255, 0, 0), false, 100);
  m_pLine = new CKhuGleSprite(GP_STYPE_LINE, GP_CTYPE_STATIC,
    CKgLine(CKgPoint(300, 350), CKgPoint(450, 250)),
    KG_COLOR_24_RGB(255, 0, 0), false, 10);
```

Main.cpp (3)

```
  m_pGameLayer->AddChild(m_pCircle1);
  m_pGameLayer->AddChild(m_pCircle2);
  m_pGameLayer->AddChild(m_pLine);

  m_pCircle1->m_Velocity = CKgVector2D(900, 50);
  m_pCircle2->m_Velocity = CKgVector2D(-100, -300);

  for(int i = 0 ; i < 100 ; i++) {
    m_pNewCircle[i] = new CKhuGleSprite(GP_STYPE_ELLIPSE,
      GP_CTYPE_DYNAMIC,
      CKgLine(CKgPoint(30, 30), CKgPoint(50, 50)),
      KG_COLOR_24_RGB(255, 0, 0), true, 100);

    m_pGameLayer->AddChild(m_pNewCircle[i]);
  }
}
```

Main.cpp (4)

```
void CCollision::Update() {
  if(m_bMousePressed[0]) {
    if(m_nLButtonStatus == 0){
      m_LButtonStart = CKgPoint(m_MousePosX, m_MousePosY);
    }
    m_LButtonEnd = CKgPoint(m_MousePosX, m_MousePosY);
    m_nLButtonStatus = 1;
  }
  else {
    if(m_nLButtonStatus == 1) {
      std::cout << m_LButtonStart.X << "," << m_LButtonStart.Y << std::endl;
      std::cout << m_LButtonEnd.X << "," << m_LButtonEnd.Y << std::endl;

      m_nLButtonStatus = 0;
    }
  }
```

Main.cpp (5)

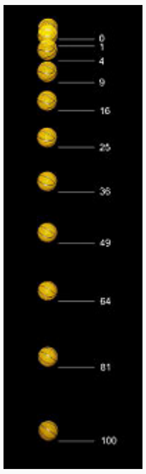
```
  if(m_bKeyPressed['S']) {
    m_pCircle1->m_Velocity = CKgVector2D(0, 0);
  }

  if(m_bKeyPressed[VK_LEFT]) m_pCircle1->m_Velocity = CKgVector2D(-500, 0);
  if(m_bKeyPressed[VK_UP]) m_pCircle1->m_Velocity = CKgVector2D(0, -500);
  if(m_bKeyPressed[VK_RIGHT]) m_pCircle1->m_Velocity = CKgVector2D(500, 0);
  if(m_bKeyPressed[VK_DOWN]) m_pCircle1->m_Velocity = CKgVector2D(0, 500);
```

Main.cpp (6)

```
  for(auto &Layer : m_pScene->m_Children) {
    for(auto &Sprite : Layer->m_Children) {
      CKhuGleSprite *Ball = (CKhuGleSprite *)Sprite;
      Ball->m_bCollided = false;
      if(Ball->m_nType == GP_STYPE_RECT) continue;
      if(Ball->m_nType != GP_STYPE_ELLIPSE) continue;
      if(Ball->m_nCollisionType != GP_CTYPE_DYNAMIC) continue;

      Ball->m_Acceleration.x
        = m_Gravity.x - Ball->m_Velocity.x * m_AirResistance.x;
      Ball->m_Acceleration.y
        = m_Gravity.y - Ball->m_Velocity.y * m_AirResistance.y;

      Ball->m_Velocity.x += Ball->m_Acceleration.x * m_ElapsedTime;
      Ball->m_Velocity.y += Ball->m_Acceleration.y * m_ElapsedTime;

      Ball->MoveBy(Ball->m_Velocity.x*m_ElapsedTime,
        Ball->m_Velocity.y*m_ElapsedTime);
```

$$\mathbf{v} = \frac{d\mathbf{x}}{dt},\ \mathbf{a} = \frac{d\mathbf{v}}{dt}$$

$$\mathbf{F} = m\mathbf{a} = m\mathbf{g} - k\mathbf{v}$$



```
Ball->m_Acceleration.x
= m_Gravity.x – (Ball->m_Velocity.x * m_AirResistance.x)/Ball->m_Mass;
```

https://upload.wikimedia.org/wikipedia/commons/thumb/0/02/Falling_ball.jpg/100px-Falling_ball.jpg

Main.cpp (7)

```
      if(Ball->m_Center.x < 0)
        Ball->MoveTo(m_nW+Ball->m_Center.x, Ball->m_Center.y);
      if(Ball->m_Center.x > m_nW)
         Ball->MoveTo(Ball->m_Center.x-m_nW, Ball->m_Center.y);
      if(Ball->m_Center.y < 0)
         Ball->MoveTo(Ball->m_Center.x, m_nH+Ball->m_Center.y);
      if(Ball->m_Center.y > m_nH)
         Ball->MoveTo(Ball->m_Center.x, Ball->m_Center.y-m_nH);

      if(CKgVector2D::abs(Ball->m_Velocity) < 0.01)
        Ball->m_Velocity = CKgVector2D(0, 0);
    }
```

CCollision:: Update

Main.cpp (8)

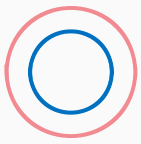
```
std::vector<std::pair<CKhuGleSprite*, CKhuGleSprite*>> CollisionPairs;

for(auto &SpriteA : Layer->m_Children) {
  CKhuGleSprite *Ball = (CKhuGleSprite *)SpriteA;
  if(Ball->m_nType != GP_STYPE_ELLIPSE) continue;

  for(auto &SpriteB : Layer->m_Children) {
    CKhuGleSprite *Target = (CKhuGleSprite *)SpriteB;
    if(Ball == Target) continue;

    if(((CKhuGleSprite *)Target)->m_nType == GP_STYPE_ELLIPSE)  {
      CKgVector2D PosVec = Ball->m_Center - Target->m_Center;
      double Overlapped
        = CKgVector2D::abs(PosVec)
            - Ball->m_Radius - Target->m_Radius;
```

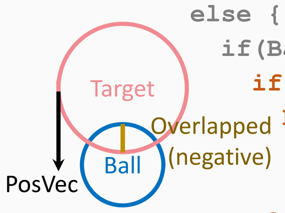
$$D-(R_1+R_2)$$

CCollision:: Update

Main.cpp (9)

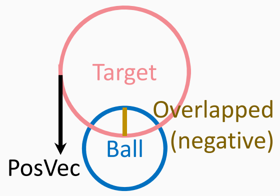
```
      if(Overlapped <= 0) { // collision detection
        CollisionPairs.push_back({Ball, Target});
        if(CKgVector2D::abs(PosVec) == 0) {
          if(Ball->m_nCollisionType != GP_CTYPE_STATIC)
            Ball->MoveBy(rand()%3-1, rand()%3-1);
          if(Target->m_nCollisionType != GP_CTYPE_STATIC)
            Target->MoveBy(rand()%3-1, rand()%3-1);
        }
        else {
          if(Ball->m_nCollisionType != GP_CTYPE_STATIC) {
            if(Target->m_nCollisionType == GP_CTYPE_STATIC)
              Ball->MoveBy(
                -PosVec.x*Overlapped/CKgVector2D::abs(PosVec),
                -PosVec.y*Overlapped/CKgVector2D::abs(PosVec));
            else
              Ball->MoveBy(
                -PosVec.x*Overlapped/CKgVector2D::abs(PosVec)*0.5,
                -PosVec.y*Overlapped/CKgVector2D::abs(PosVec)*0.5);
          }
```



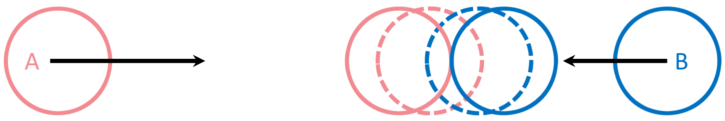


Main.cpp (10)

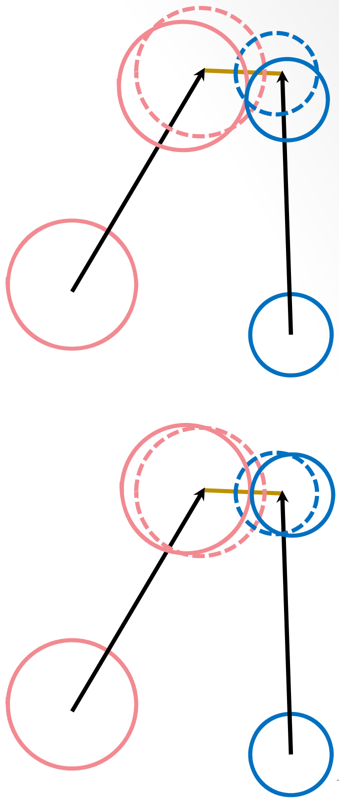
```
                if(Target->m_nCollisionType != GP_CTYPE_STATIC) {
                  if(Ball->m_nCollisionType == GP_CTYPE_STATIC)
                    Target->MoveBy(PosVec.x*Overlapped/CKgVector2D::abs(PosVec),
                              PosVec.y*Overlapped/CKgVector2D::abs(PosVec));
                  else
                    Target->MoveBy
                      (PosVec.x*Overlapped/CKgVector2D::abs(PosVec)*0.5,
                      PosVec.y*Overlapped/CKgVector2D::abs(PosVec)*0.5);
                }
              }
              Ball->m_bCollided = true;
              Target->m_bCollided = true;
            }
          }
        }
      }
```

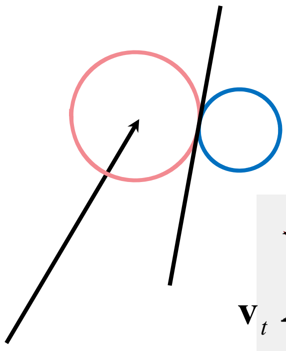


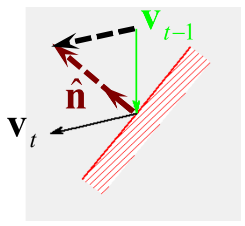
Main.cpp (11)



$$v_A(t+1) = \frac{m_A - m_B}{m_A + m_B} v_A + \frac{2m_B}{m_A + m_B} v_B$$







$$\hat{\mathbf{n}} = \overrightarrow{AB}$$

$$p = \frac{2\left(\mathbf{v}_A \cdot \hat{\mathbf{n}} - \mathbf{v}_B \cdot \hat{\mathbf{n}}\right)}{m_A + m_B}$$

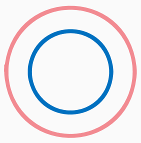
$$\mathbf{v}_A(t+1) = \mathbf{v}_A - p m_B \hat{\mathbf{n}}$$
$$\mathbf{v}_B(t+1) = \mathbf{v}_B + p m_A \hat{\mathbf{n}}$$

$$\mathbf{v}(t+1) = \mathbf{v} - 2(\mathbf{v} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}$$

CCollision:: Update

Main.cpp (12)

```
    for(auto &Pair : CollisionPairs) {
      CKhuGleSprite *BallA = Pair.first;
      CKhuGleSprite *BallB = Pair.second;

      CKgVector2D PosVec = BallB->m_Center - BallA->m_Center;
      double Distance = CKgVector2D::abs(PosVec);
      if(Distance == 0) Distance = 1E-6;
      CKgVector2D Normal = (1./Distance)*PosVec;

      double kx = (BallA->m_Velocity.x - BallB->m_Velocity.x);
      double ky = (BallA->m_Velocity.y - BallB->m_Velocity.y);
      double p = 2.0
         * (Normal.x * kx + Normal.y * ky) / (BallA->m_Mass + BallB->m_Mass);

      BallA->m_Velocity.x = BallA->m_Velocity.x - p * BallB->m_Mass * Normal.x;
      BallA->m_Velocity.y = BallA->m_Velocity.y - p * BallB->m_Mass * Normal.y;

      BallB->m_Velocity.x = BallB->m_Velocity.x + p * BallA->m_Mass * Normal.x;
      BallB->m_Velocity.y = BallB->m_Velocity.y + p * BallA->m_Mass * Normal.y;
    }
  }
```



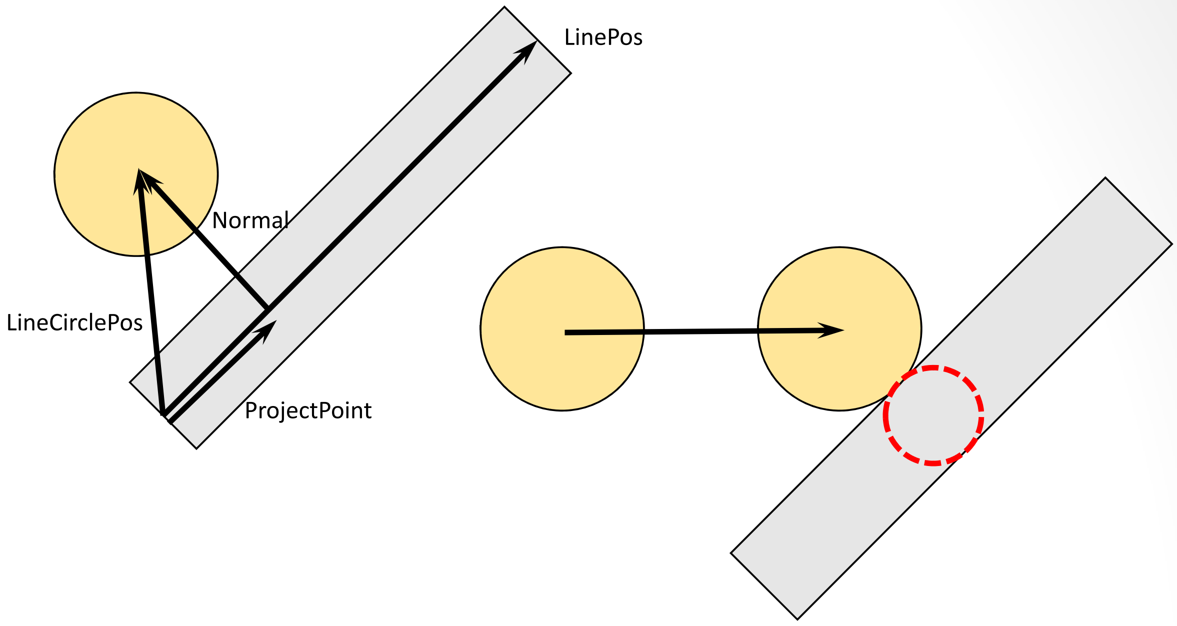
CCollision:: Update & main

Main.cpp (13)

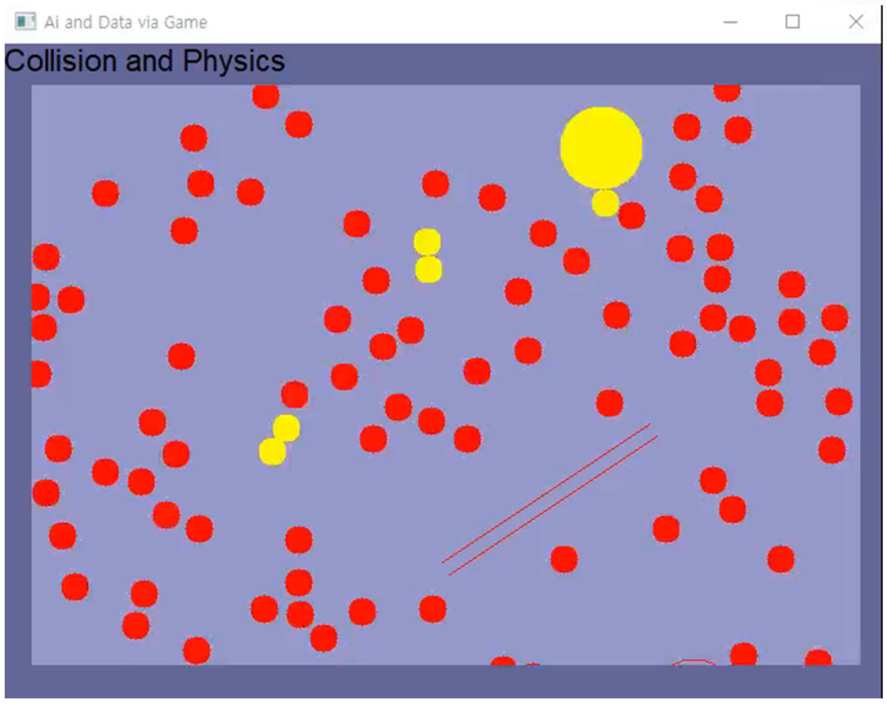
```
  m_pScene->Render();
  DrawSceneTextPos("Collision and Physics", CKgPoint(0, 0));
  CKhuGleWin::Update();
}

int main() {
  CCollision *pCollision = new CCollision(640, 480);

  KhuGleWinInit(pCollision);

  return 0;
}
```

Practice I (1)

- Circle-line collision



Practice I (2)



Advanced Courses

- Friction
- Elasticity

Project I

Game Design

Game Design

- Pong
- Simple platformer