```cpp
#include "stdafx.h"
#include "EnemyShooter.h"
#include "Transform.h"
#include "MovementInput.h"
#include "Global/Utils.h"
#include "System/AudioManager.h"

EnemyShooter::EnemyShooter(float offset) : m_timer(0.f), m_spawn_offset(offset), m_shoot_time(0.f), m_shooter_transform(nullptr), m_target_position(nullptr)
{
    bullet_pool.Init();
}

void EnemyShooter::Init()
{
    m_shooter_transform = &Component::object->GetComponent<Transform>();
    m_shoot_time = Utils::RandomFloat(20.f, 80.f);
}

void EnemyShooter::Update(float deltaTime)
{
    Shoot(deltaTime);
}

void EnemyShooter::Shoot(float deltaTime)
{
    if (m_timer >= m_shoot_time)
    {
        //spawn pos for bullet
        Vector2 spawn_pos = m_shooter_transform->position - m_shooter_transform->forward * m_spawn_offset;

        //if target is moving - predict it's next pos
        if (m_target_input->IsMoving())
        {
            //calculate future position
            float angle_to_shoot = PredictTargetNextAngle();

            //shoot towards the predicted pos
            Vector2 shoot_pos;
            shoot_pos.x = m_center->x + cos(angle_to_shoot) * m_radius;
            shoot_pos.y = m_center->y + sin(angle_to_shoot) * m_radius;

            Vector2 shoot_direction = (shoot_pos - m_shooter_transform->position).normalize();

            bullet_pool.Spawn(spawn_pos, shoot_direction, m_shooter_transform->rotation_angle);
```

```
45          }
46          //if not, shoot towards target
47          else
48          {
49              Vector2 shoot_direction = (*m_target_position -
                  m_shooter_transform->position).normalize();
50              bullet_pool.Spawn(spawn_pos, shoot_direction,
                  m_shooter_transform->rotation_angle);
51          }
52
53          m_timer = 0.f;
54
55          //sound
56          AudioManager::GetInstance().PlaySoundEffect(SoundID::ENEMY_SHOOT,
              false);
57      }
58
59      m_timer += deltaTime / 100.f;
60  }
61
62  float EnemyShooter::PredictTargetNextAngle()
63  {
64      //calculate the time for bullet to hit the target
65      float distance_to_target = Utils::Distance(*m_target_position,
          m_shooter_transform->position);
66      float delta_t = distance_to_target / m_bullet_speed;
67
68      //predicting the amount of distance player will move in delta_t seconds
69      float angular_displacement = m_target_input->GetAngularVelocity() *
          delta_t;
70
71      //setting direction for future position
72      if (m_target_input->IsClockWise()) angular_displacement = -
          angular_displacement;
73
74      //calculate future angle to shoot at based on the amount of distance
          predicted
75      return (m_target_input->GetAngle() + angular_displacement);
76  }
77
78  void EnemyShooter::SetBulletPool()
79  {
80      bullet_pool.SetUp();
81  }
82
83  void EnemyShooter::SetTarget(const Object::Ref target, const Vector2&
      planet_pos)
84  {
85      m_target_input = &target->GetComponent<MovementInput>();
```

```
86      m_radius = m_target_input->GetDistanceToCenter();
87      m_center = &planet_pos;
88      m_target_position = &target->GetComponent<Transform>().position;
89  }
90
```