🎓 LESSON 07 Player's collision

Start

Video Link

LESSON GOALS

🎯 Goal Summary:

- Detect proximity collisions between players
- Only the local player checks for collisions
- Trigger a forced backwards throw on tackle
- Ensure network consistency

Next

Learning Objectives + Deliverables

Learning Objectives

- Iterate through players to detect collisions
- Use VRCPlayerApi.GetPosition() for distance checks
- Ensure collision logic only runs locally
- Track opponents using team arrays
- Trigger forced ball drop on impact

Deliverables

- ✓ Collision detection system
- ✓ Forced backwards throw
- ✓ Local-only processing (no network conflicts)
- ✓ Smooth multiplayer behavior

GENERAL DESCRIPTION OF THE LESSON

High-Level Description

🎯 **Goal:** In order to quit the ball from a player, the other player only should touch him and the ball will be automatically thrown backwards.

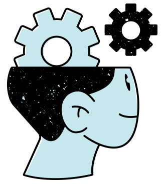
🔴 LEVEL 1: Advanced Challenge

In this lesson we will:

- Extend ThrowBall() to support backward throws
- Add collision logic inside GameController → CheckPlayersCollision()
- Run collision check only during GAME state
- Use team arrays to test only enemy players

🧠AI Lesson Prompt: Player Collision

Next



Exercise 1: We are going to detect in each individual client the distance with the other players. When the distance is close enough we will consider that there is a collision and the same local player with throw his ball backwards.

- Actions:

1) Before starting we are going to do this in (**GameController**) to verify that we got the right number of players in all the connected clients when we start the game:

2) We are going to extend the functionality of (**GameBall::ThrowBall(bool direction)**) so the the ball is throw backwards by the same player when there is collision with another player of the opposite team.

3) Still in the script (**GameController**) we are going to define these 2 member variables that will be initilitzed in when entering the GAME state that will keep references to the VRCPlayerApi to all the players in the game:

4) Initialize the previous arrays with the references of VRCPlayerApi of the players

5) Continuing in the script (**GameController**), we are going to implement the method (**CheckPlayersCollision()**) that will check the collision by distance of the players. When the collision that we are looking for happens for the local player, the same player will throw backwards the ball (**ThrowBall(false)**)

6) You won't be able to test this behavior in the Unity editor. Run multiple VRChat instances to test it.

```
public class GameController : UdonSharpBehaviour
{
   // OTHER CODE ...

   private void CheckPlayersCollision()
   {
      for (int i = 0; i < _playersForTeam; i++)
      {
         VRCPlayerApi A = _playersTeamBlueInGame[i];
         if (A == null) continue;
         for (int j = 0; j < _playersForTeam; j++)
         {
             VRCPlayerApi B = _playersTeamRedInGame[j];
             if (B == null) continue;
             if (Vector3.Distance(A.GetPosition(), B.GetPosition()) < CollideDistance)
             {
                 if ((A == Networking.LocalPlayer) || (B == Networking.LocalPlayer))
                 {
                     GameBall playerBall = ballController.GetBallOwnedByLocalPlayer();
                     if (playerBall != null)
                     {
                         playerBall.ThrowBall(false);
                     }
                 }
              }
          }
      }
   }
```

```
public class GameBall : UdonSharpBehaviour
{
   // OTHER CODE ...

   public void ThrowBall(bool direction)
   {
      if (gameController.IsVR)
      {
         VRCPlayerApi.TrackingData handData = _currentOwner.GetTrackingData(VRCPlayerApi.TrackingDataType.RightHand);
         impulseVector = (handData.rotation * Quaternion.Euler(new Vector3(0, 55.3f, 0))) * Vector3.forward;
      }
      else
      {
         VRCPlayerApi.TrackingData headData = _currentOwner.GetTrackingData(VRCPlayerApi.TrackingDataType.Head);
         impulseVector = headData.rotation * Vector3.forward;
      }
      impulseVector = (direction ? 1 : -1) * impulseVector;
      if (!direction) impulseVector.y = 0.5f;
      impulseVector.Normalize();
      ClearOwner();
      _ignoreTime = 1;
   }
```

```
public class GameController : UdonSharpBehaviour
{
   // OTHER CODE ...

   private void StateChanged()
   {
    // OTHER CODE ...

      case GameState.GAME:
         // OTHER CODE ...
         // GET ALL THE VRCPlayerApi OF THE PLAYERS IN THE GAME
         _playersTeamBlueInGame = new VRCPlayerApi[_playersForTeam];
         int indexPlayersInGame = 0;
         for (int i = 0; i < MaxPlayersPerTeam; i++)
         {
            if (_teamBluePlayers[i] > 0)
            {
               _playersTeamBlueInGame[indexPlayersInGame] = VRCPlayerApi.GetPlayerById(_teamBluePlayers[i]);
               indexPlayersInGame++;
            }
         }
         _playersTeamRedInGame = new VRCPlayerApi[_playersForTeam];
         indexPlayersInGame = 0;
         for (int i = 0; i < MaxPlayersPerTeam; i++)
         {
            if (_teamRedPlayers[i] > 0)
            {
               _playersTeamRedInGame[indexPlayersInGame] = VRCPlayerApi.GetPlayerById(_teamRedPlayers[i]);
               indexPlayersInGame++;
            }
         }
```

```
public void SetState(GameState newState)
{
   if (Networking.IsMaster)
   {
      syncedState = (int)newState;
      _currentState = (GameState)syncedState;
      RequestSerialization();
      StateChanged();
   }
}
```

```
public class GameController : UdonSharpBehaviour
{
  public int GetTotalNumberPlayers()
  {
     int total = 0;
     if ((_teamBluePlayers == null) || (_teamRedPlayers == null)) return 0;

     for (int i = 0; i < MaxPlayersPerTeam; i++)
     {
        if (_teamBluePlayers[i] > 0)  total++;
     }
     for (int i = 0; i < MaxPlayersPerTeam; i++)
     {
        if (_teamRedPlayers[i] > 0) total++;
     }
     return total;
   }

   private void StateChanged()
   {
      // OTHER CODE ...
      case GameState.GAME:
       // OTHER CODE ...
       ... _totalPlayersInGame = GetTotalNumberPlayers();
```

```
public class GameController : UdonSharpBehaviour
{
  // OTHER CODE ...

  private VRCPlayerApi[] _playersTeamBlueInGame;

  private VRCPlayerApi[] _playersTeamRedInGame;
```

LESSON 07 COMPLETED

You now have:

- A complete defensive tackle mechanic
- Local-only collision computation
- Smooth, network-safe physics results
- Backwards throw on contact

Code Checkpoint: Player Collision

Self-Evaluation

It's time to put what we've learned into practice! Here are 4 questions to check by yourself what you have learnt in this lesson.

Question 1

Question 2

Question 3

Question 4

Why is player-to-player collision detection performed only by the local player?

Because only the local player has a collider

To reduce the visual complexity of the game

To prevent multiple players from triggering the same collision logic simultaneously

What condition must be true for a tackle (forced ball drop) to occur?

Two players must be on the same team

The ball must be moving at high speed

The local player must be carrying the ball and collide with an opponent

Why does the tackle force the ball to be thrown backward instead of dropped straight down?

To simplify the physics calculations	To create clear gameplay feedback and a fair defensive outcome	To ensure the ball respawns faster

What is the purpose of adding a cooldown to the collision detection system?

To avoid repeated collision triggers in a very short time

To prevent players from moving too fast

To improve rendering performance

Help us to improve

Was the proximity collision logic intuitive?

Write your answer here.

Send

Did you understand how to trigger forced ball dropping?

Write your answer here.

Send

What part of the tackle system caused difficulty?

Write your answer here.

Send

How could collision detection be taught more clearly or interactively?

Write your answer here.

Send