

Section 3 - Crazy Karts

These are the slides that accompany the Unreal Multiplayer course.

Looking for something, try searching [our GitHub repo.](#)

Enjoy your stay!

*Sam & Ben
GameDev.tv*

Live Google Slides at bit.ly/multiplayerslides



Introduction to Crazy Karts

Creating A Go-Kart Pawn

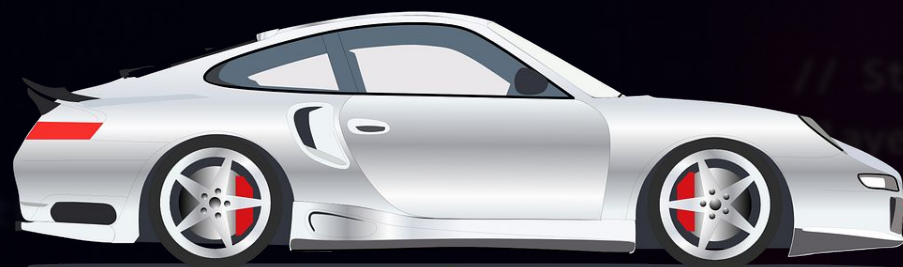
Make It Move!

- Add an appropriate camera
- Hook up the forward axis
- Use to set a velocity member variable
- Update position in tick.



Understanding Forces And Movement

How Things Move



$$\mathbf{F} = \mathbf{m} * \mathbf{a}$$

$$\mathbf{a} = \mathbf{F} / \mathbf{m}$$

Live Google Slides at bit.ly/multiplayerslides



Acceleration

$$dx / dt = v$$

$$dx = v * dt$$

$$dv / dt = a$$

$$dv = a * dt$$

Live Google Slides at bit.ly/multiplayerslides



Provide The Driving Force.

- Where should it come from?
- Make sure that it is configurable.
- Test!



Blocking Movement Without Physics

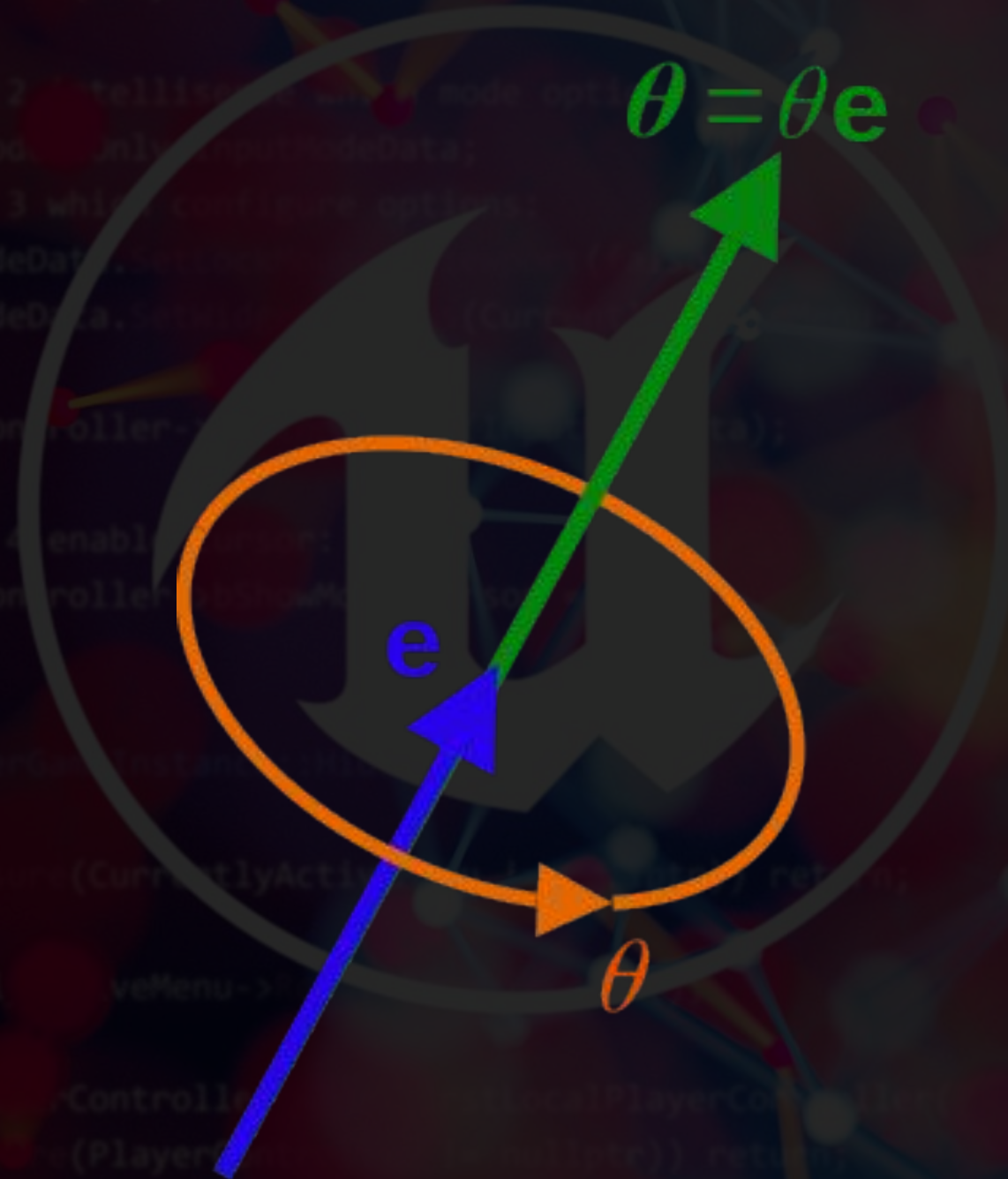
Reset The Velocity

- Read up on your **FHitResult**
- Check if there was a collision
- Reset velocity
- Refactor for clarity.



Rotations With Quaternions

Angle Axis Rotations



Live Google Slides at bit.ly/multiplayerslides



Simulating Air Resistance

Rotate Velocity

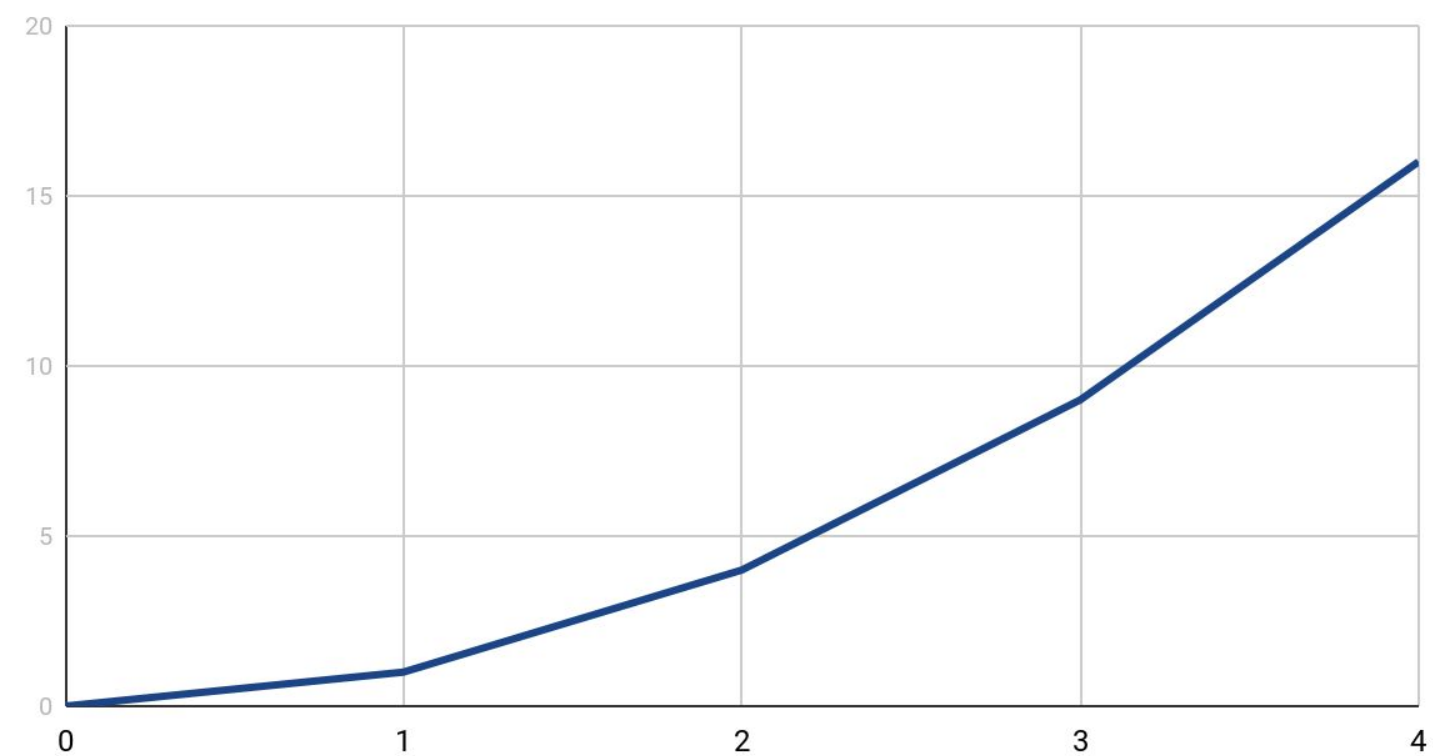
- Read the FQuat documentation
- Update the velocity
- Test
- Refactor.



Drag



Force vs Speed



Live Google Slides at bit.ly/multiplayerslides

Air Resistance Formula

$$\text{AirResistance} = - \text{Speed}^2 \times \text{DragCoefficient}$$

Live Google Slides at bit.ly/multiplayerslides



Add The Air Resistance

- Create a function to calculate resistance
- Use the formula from last slide
- Which direction should it be in?
- Sum the forces on the car.



Calculating Drag Coefficient

$$\text{AirResistance} = - \text{Speed}^2 \times \text{DragCoefficient}$$

$$\text{AirResistance} / \text{Speed}^2 = \text{DragCoefficient}$$

$$10,000 / 25^2 = \text{DragCoefficient}$$

Live Google Slides at bit.ly/multiplayerslides



Simulating Rolling Resistance

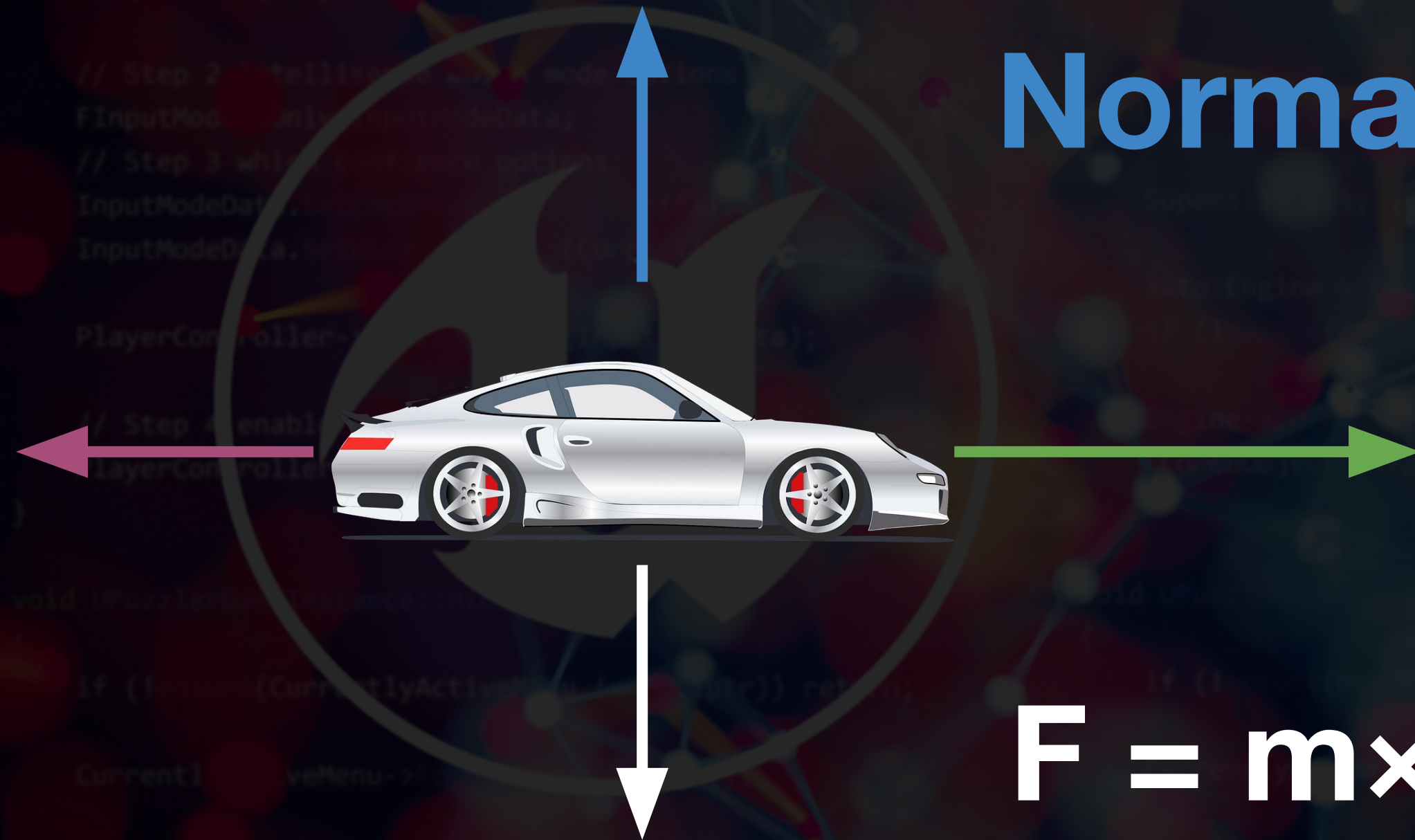
Rolling Resistance

$$\text{RollingResistance} = \text{RRCoefficient} \times \text{NormalForce}$$

Live Google Slides at bit.ly/multiplayerslides



Gravity



Normal Force

$$F = m \times g$$

Live Google Slides at bit.ly/multiplayerslides

Add Rolling Resistance

- Try different coefficients
- How long does it take to stop?
- Tweak!



Simulating Rolling Resistance

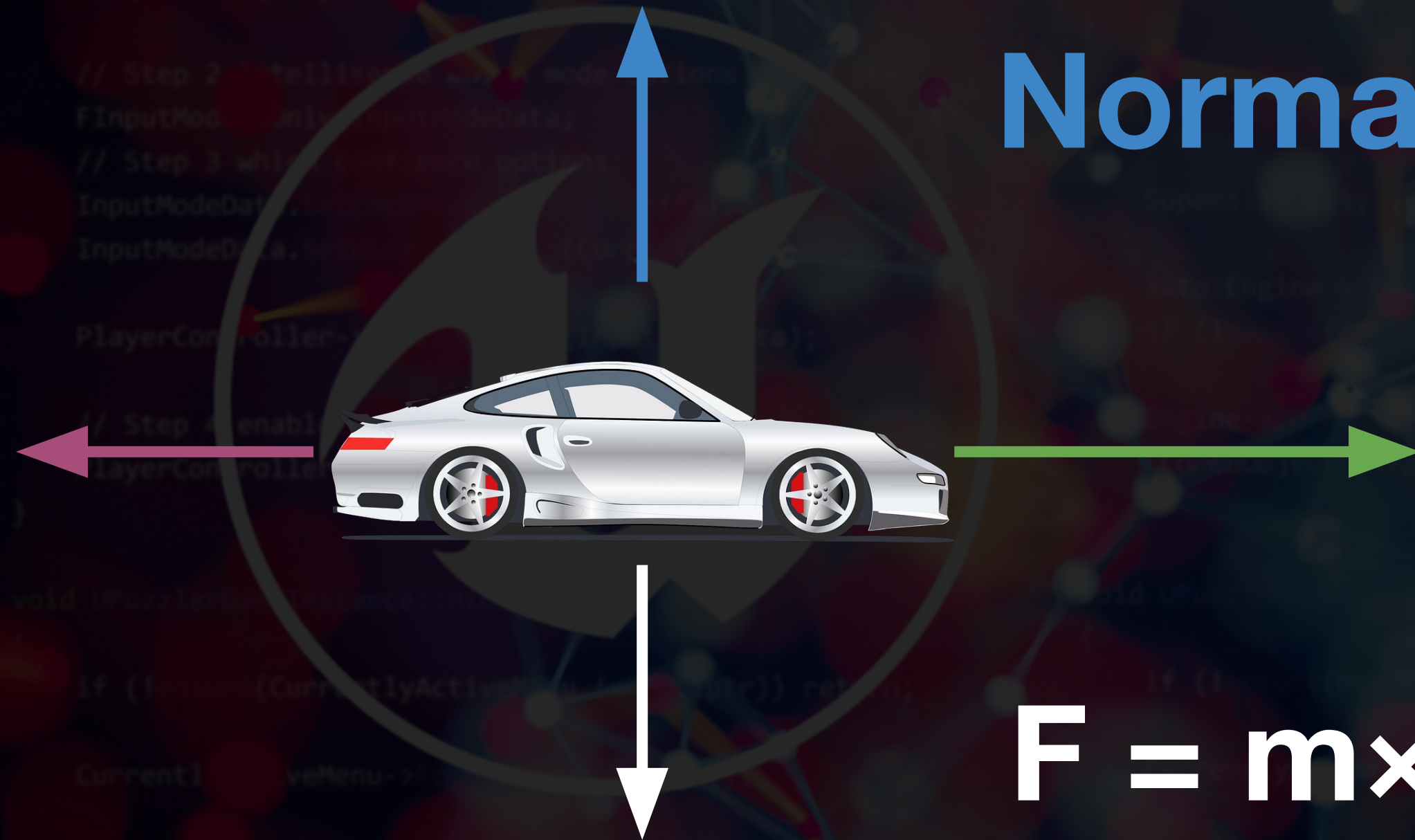
Rolling Resistance

$$\text{RollingResistance} = \text{RRCoefficient} \times \text{NormalForce}$$

Live Google Slides at bit.ly/multiplayerslides



Gravity



Normal Force

$$F = m \times g$$

Live Google Slides at bit.ly/multiplayerslides

Steering And Turning Circles

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// All players use all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()  
  
// Listen is important.  
// Server is the only one that can listen.
```

```
Instance::GetErrorMessage()  
  
// message; //To test use some other string
```

```
Instance::ShowMenu()  
  
if (CurrentlyActiveMenu != nullptr)) return;  
  
CurrentlyActiveMenu->AddToViewport();
```

```
// player controller.  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;
```

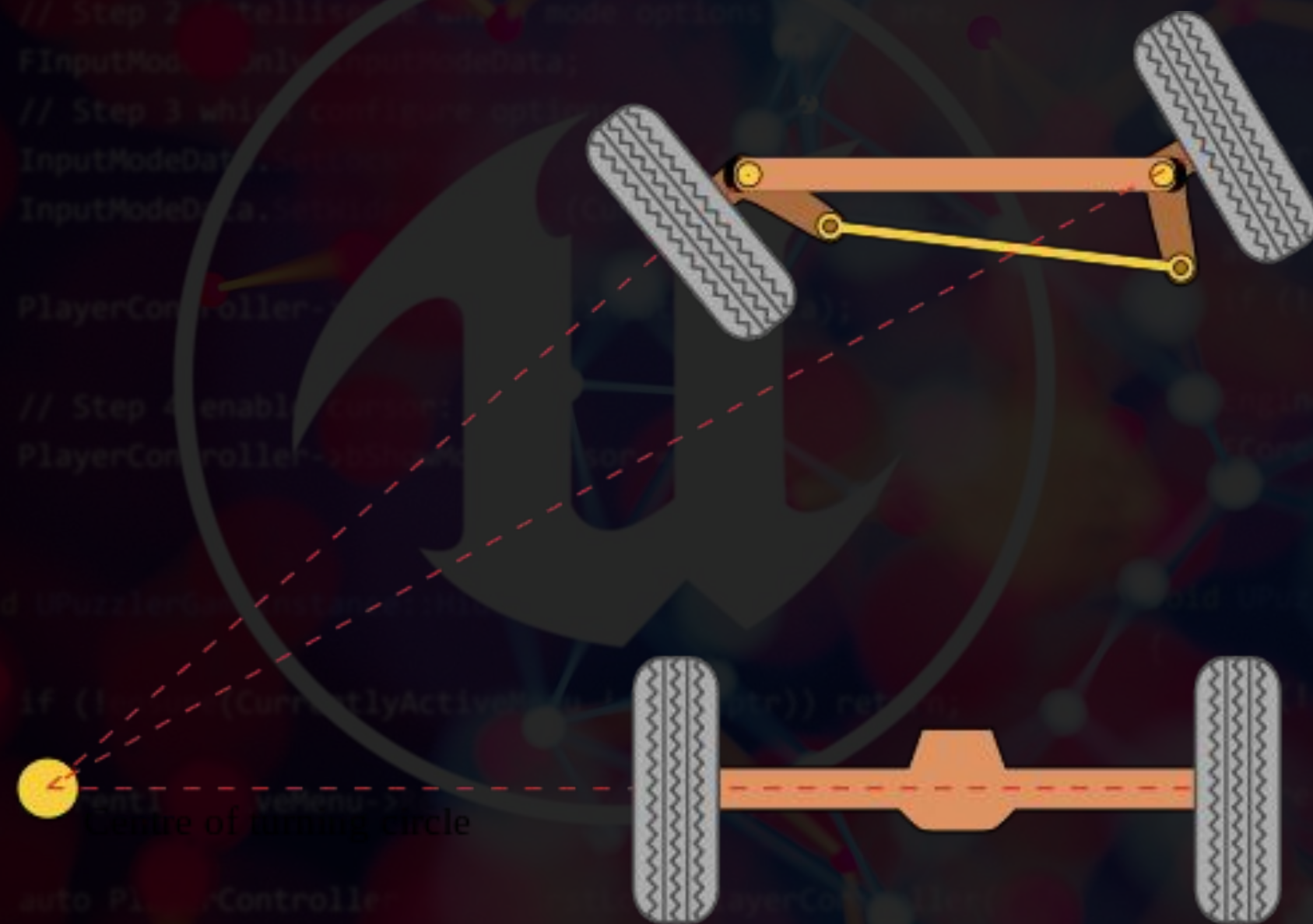
```
if (CurrentlyActiveMenu != nullptr) return;  
CurrentlyActiveMenu->AddToViewport();  
  
// Step 0 get player controller.  
auto PlayerController = GetFirstLocalPlayerController();  
if (PlayerController != nullptr) return;  
  
// Step 2 Intellisece which mode options are.  
FInputModeGameOnly InputModeData;  
// Step 3 which window options are.  
InputModeData.SetInputMode(FInputModeGameOnly);  
PlayerController->SetInputMode(InputModeData);  
  
// Step 4 enable cursor.  
PlayerController->bShowMouseCursor = true;
```

```
void UPuzzlerGameInstance::HitMenu()  
  
if (CurrentlyActiveMenu != nullptr) return;  
  
CurrentlyActiveMenu->AddToViewport();  
  
auto PlayerController = GetFirstLocalPlayerController();  
if (PlayerController != nullptr) return;  
  
FInputModeGameOnly InputModeData;  
PlayerController->SetInputMode(InputModeData);  
  
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()  
  
ConstructorHelpers::FClassFinder<UserWidget> MenuClass(  
    "MenuClass", MenuClassFinder.Class);  
  
ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass(  
    "JoinMenuClass", JoinMenuClassFinder.Class);  
  
void UPuzzlerGameInstance::LoadMainMenu()  
  
void UPuzzlerGameInstance::LoadJoinServerMenu()  
  
if (JoinMenuClass != nullptr) return;
```

```
CoreObjectDelegates::PreLoadMap.Add(this, &UPuzzlerGameInstance::PreLoadMap);  
  
void UPuzzlerGameInstance::LoadMainMenu()  
  
if (MenuClass != nullptr) return;  
  
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);  
CurrentlyActiveMenu->AddToViewport();  
  
void UPuzzlerGameInstance::LoadJoinServerMenu()  
  
if (JoinMenuClass != nullptr) return;
```


How To Steer



Live Google Slides at bit.ly/multiplayerslides

Turning Circle



$$dx = d\theta \times r$$

Live Google Slides at bit.ly/multiplayerslides

Add A Turning Circle

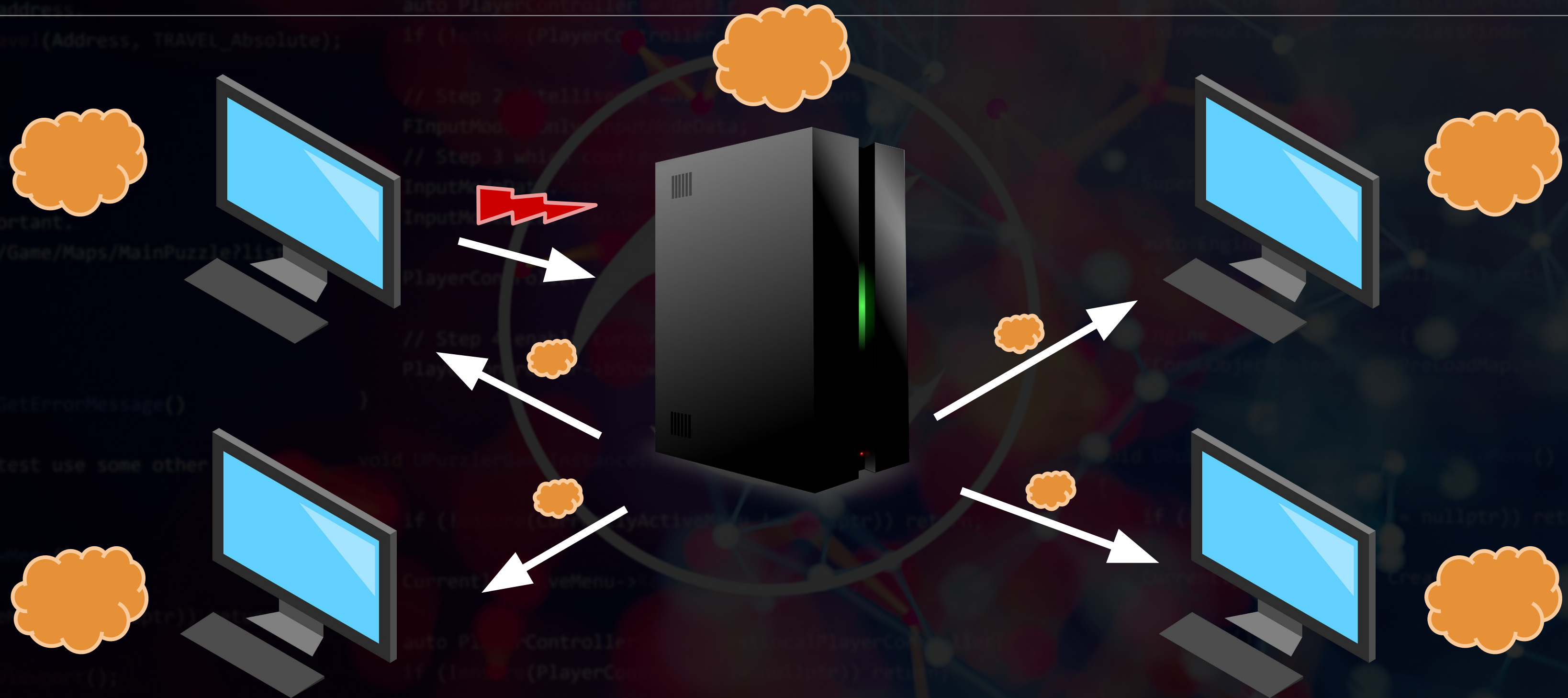
- Make the circle radius configurable
- Calculate the angle from the speed and radius
- What should happen when we aren't moving?
- What about reversing?



Sneaky Rafiki

Server Functions & Cheat Protection

Client-Server



Live Google Slides at bit.ly/multiplayerslides

Cheat Protection

✓
SetThrottle()
Jump()
Attack()

✗
SetLocation()
SetEnemyHealth()
KillEnemy()

Live Google Slides at bit.ly/multiplayerslides



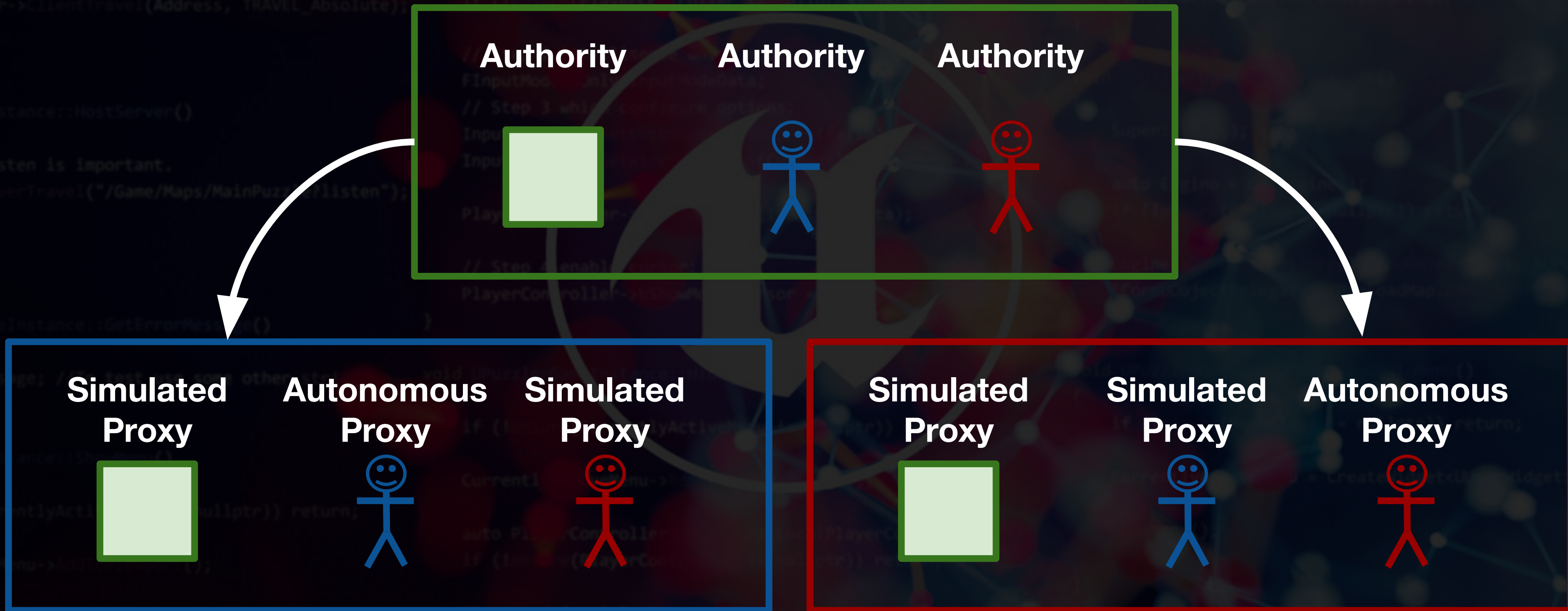
Cheat Protect The Game

- Implement steering
- Think of how the client might cheat
- Implement validation
- What happens if the validation fails?



AutonomousProxy vs SimulatedProxy

Actor Roles



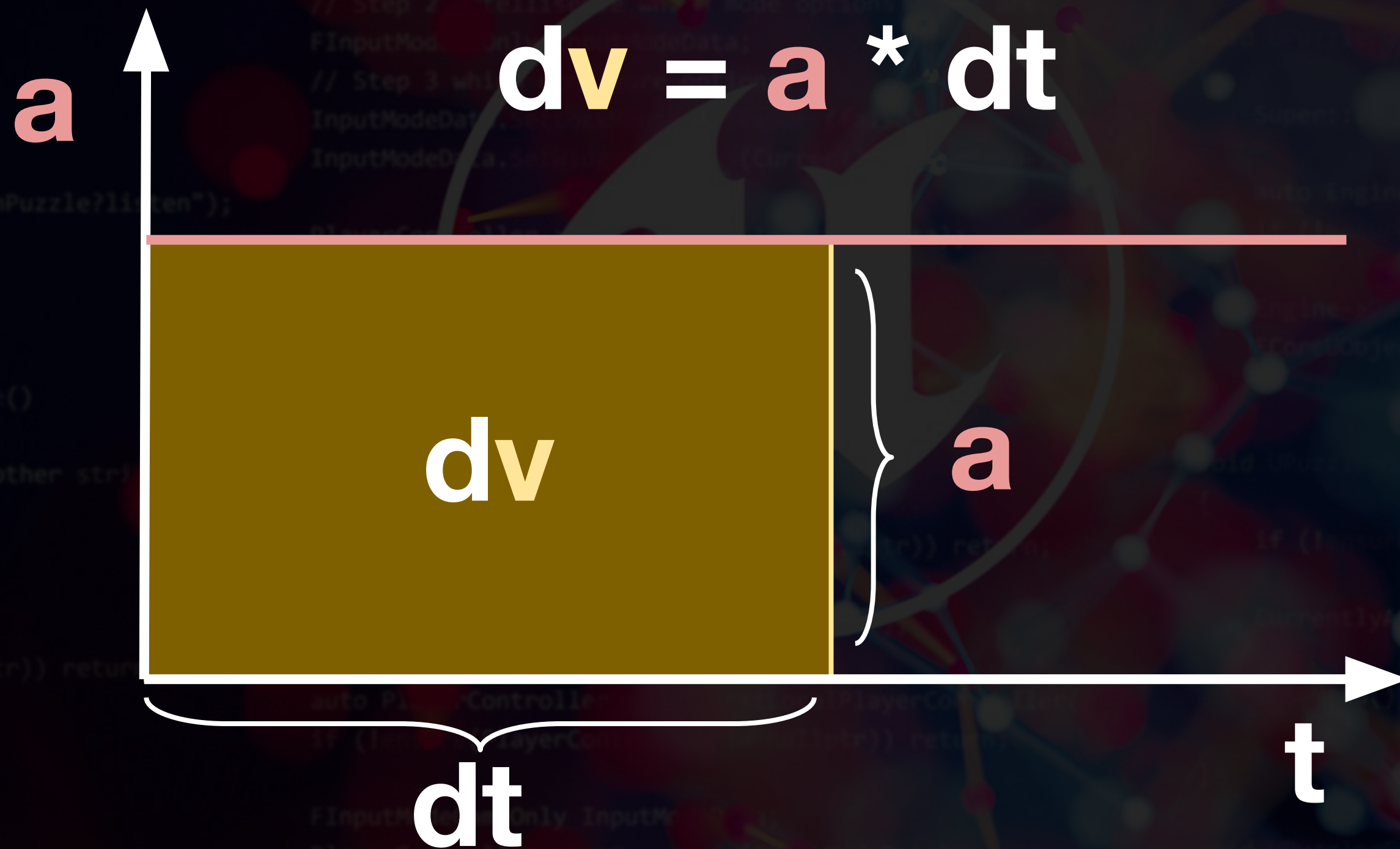
Update The **AutonomousProxy**

- Handle the bindings locally first
- Then pass them up to the server
- Test to check your positions coincide.



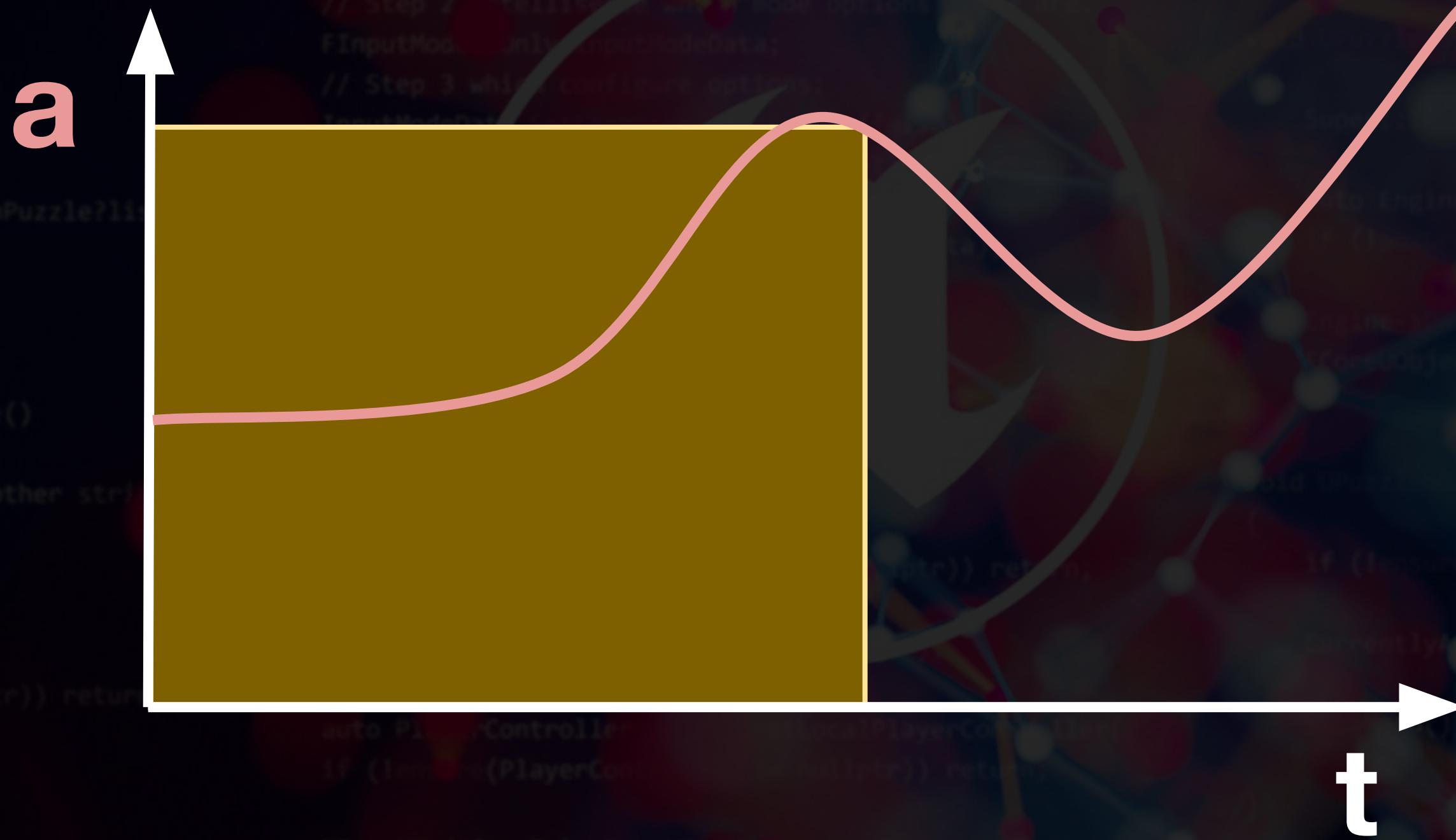
Sources Of Simulation Error

Numerical Integration



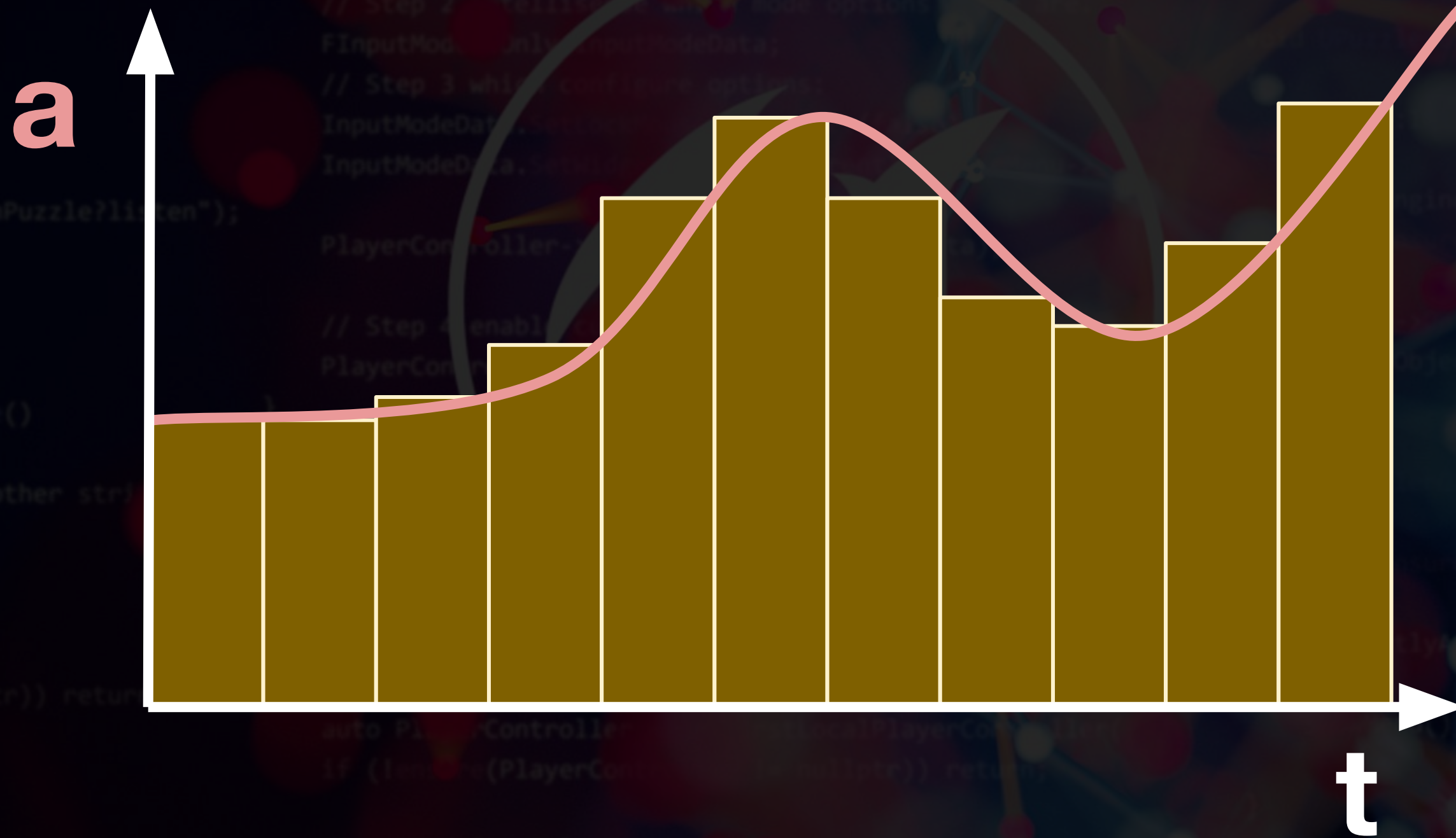
Live Google Slides at bit.ly/multiplayerslides

Numerical Integration: Errors



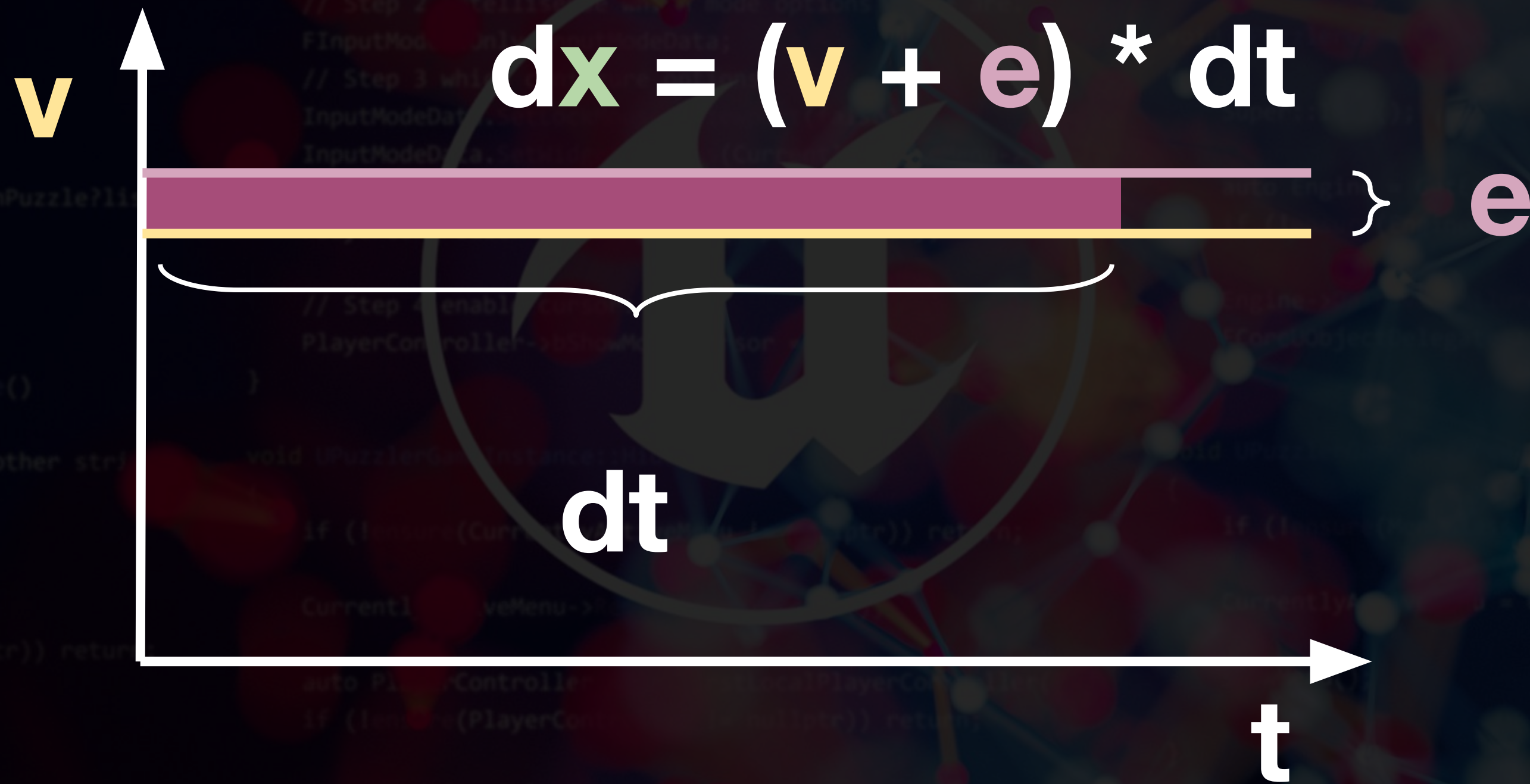
Live Google Slides at bit.ly/multiplayerslides

Numerical Integration: Errors



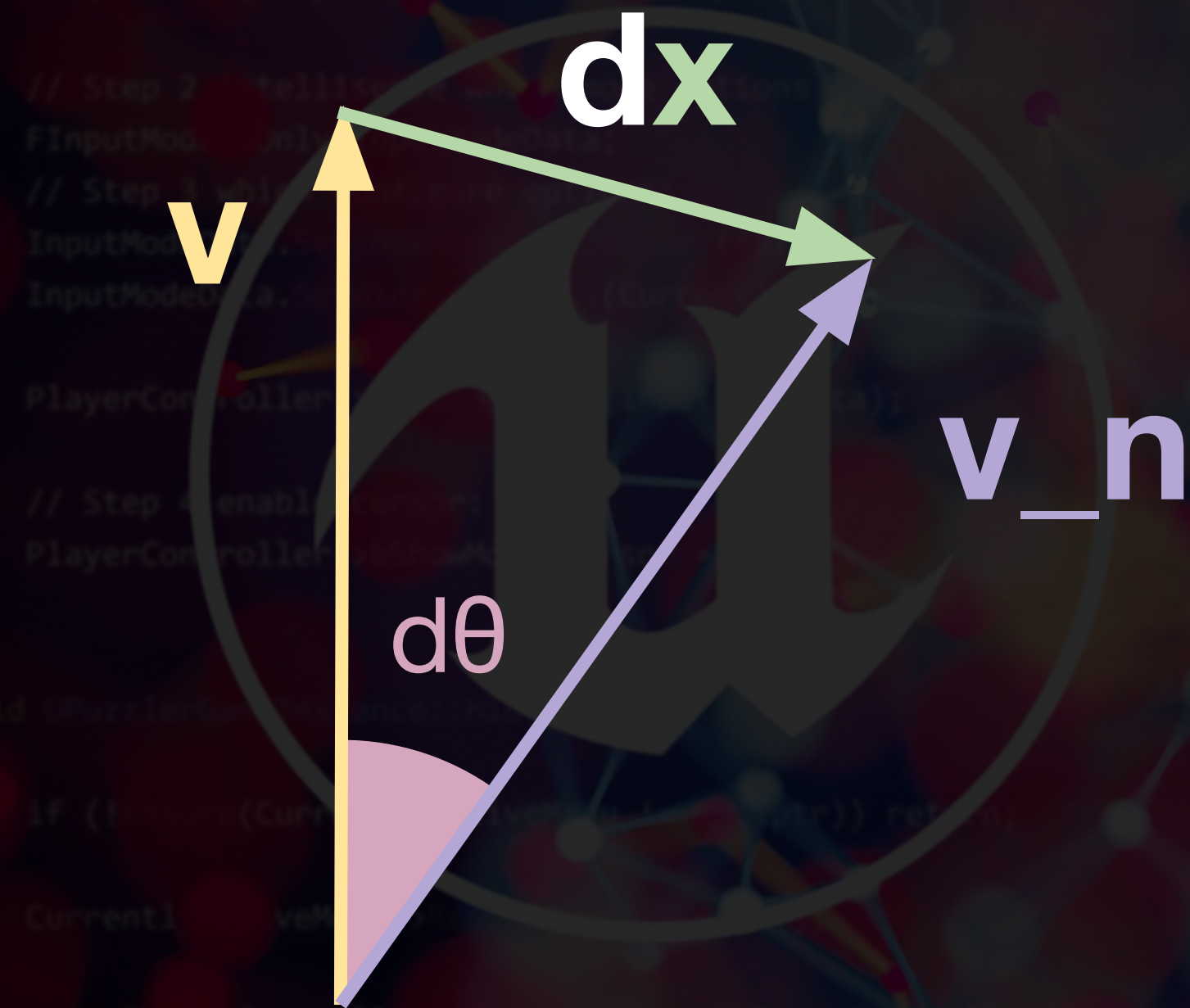
Live Google Slides at bit.ly/multiplayerslides

Integration: Error Magnification



Live Google Slides at bit.ly/multiplayerslides

Rotation: Error Magnification



Live Google Slides at bit.ly/multiplayerslides



How Could You Solve it?

- Review the sources of errors.
- Make a list of 3 potential solutions.
- Write the pros and cons of each.
- Share on the community site.
- Comment on somebody else's share.



Approaches

1. Synchronise velocity from the server
2. Overwrite location and rotation from the server
3. Simulate with a fixed time step.



Replicating Variables From The Server

Set And Get The Position

- Set on the server
- Get on all clients
- Test!
- Repeat for rotation.



Triggering Code On Replication

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// All players use all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
Listen is important.
```

Triggering Code On Replication

```
Instance::GetErrorMessage()  
  
//To test use some other string
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
Menu->AddToViewport();
```

```
player controller.
```

```
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisenet which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which configures options.
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputModeData(InputModeData);
```

```
// Step 4 enable cursor.
```

```
PlayerController->bShowMouseCursor = true;
```

```
}
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> MenuClass;
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
void UPuzzlerGameInstance::LoadMainMenu()
```

```
{  
    if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{  
    if (!ensure(JoinMenuClass != nullptr)) return;
```

```
CoreObjectDelegates::PreLoadMap.AddStatic(this, &UPuzzlerGameInstance::PreLoadMap);
```

```
void UPuzzlerGameInstance::PreLoadMap()
```

```
{  
    if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

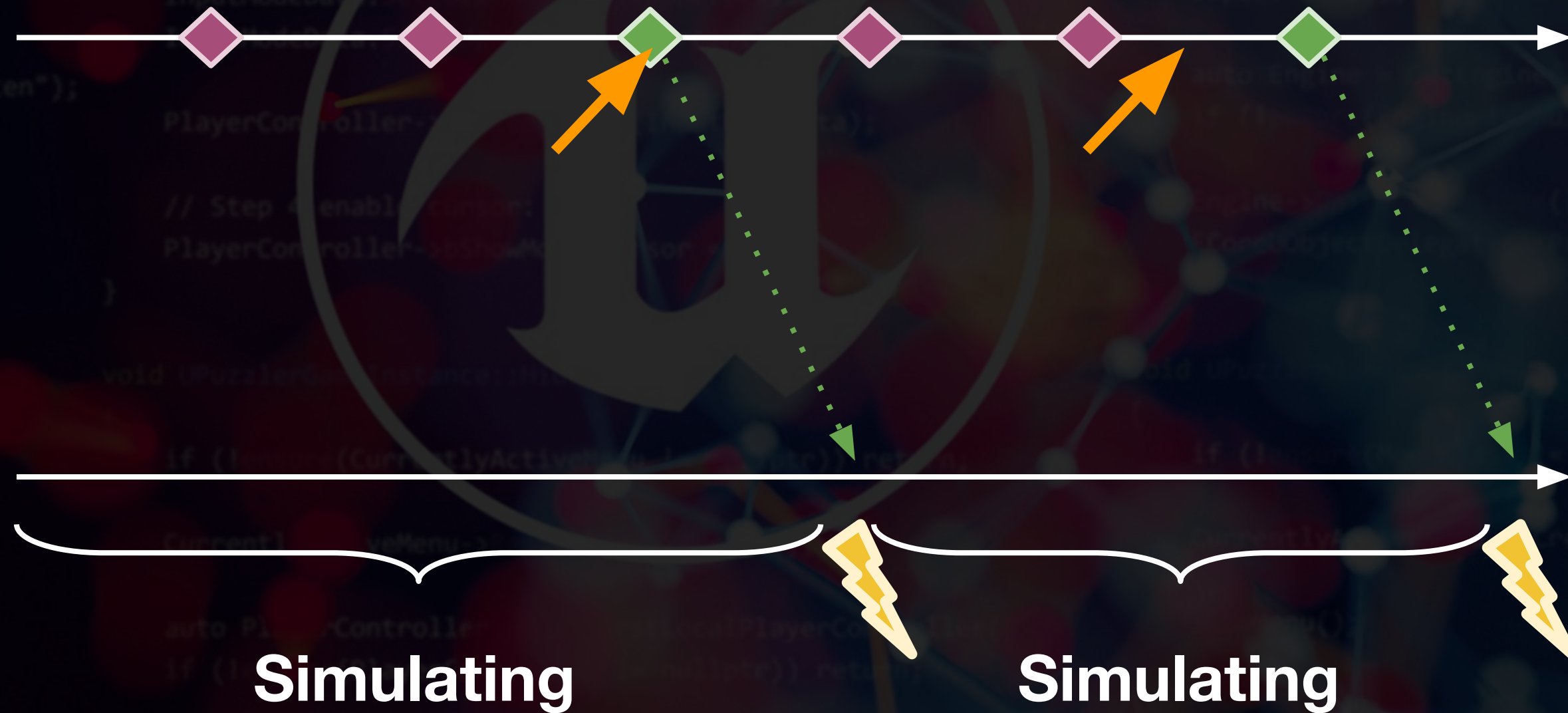
```
{  
    if (!ensure(JoinMenuClass != nullptr)) return;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```


How Replication Works

Server



Client

Simulating

Simulating

Live Google Slides at bit.ly/multiplayerslides



Simulate Between Updates

- Replicate the transform for simplicity
- Ensure we are simulating locally
- Only overwrite when there is an update.



Smooth Simulated Proxies

Make It Smooth

- Why is the motion still jerky?
- What else could you replicate?
- Can you do any better?



Simulating Lag And Packet Loss

Packets

Message

Address

```
10110111111001100001001010001110101011
01111110011000010010100011101010110111
1110011000010010100011101010110111110
0110000100101000111010101101111100110
0001001010001110101011011111001100001
0010100011101010110111110011000010010
1000111010101101111100110000100101000
1110101011011111001100001001010001110
1010110111110011000010010100011101010
1101111100110000100101000111010101101
1111100110000100101000111010101101111
1001100001001010001110101011011111001
10000100101000111010
```

192.168.1.93

Live Google Slides at bit.ly/multiplayerslides



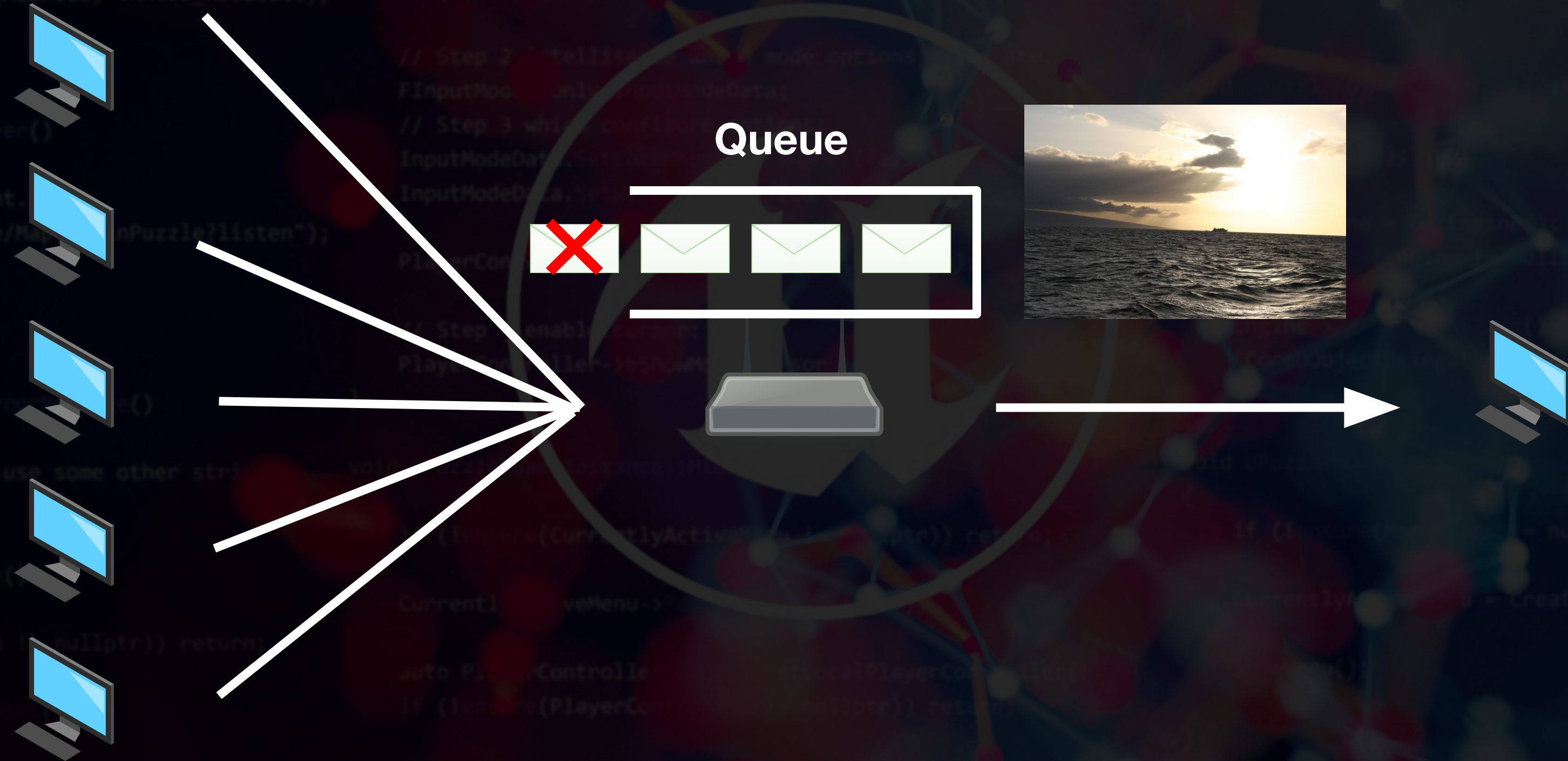
Sending Packets



Live Google Slides at bit.ly/multiplayerslides

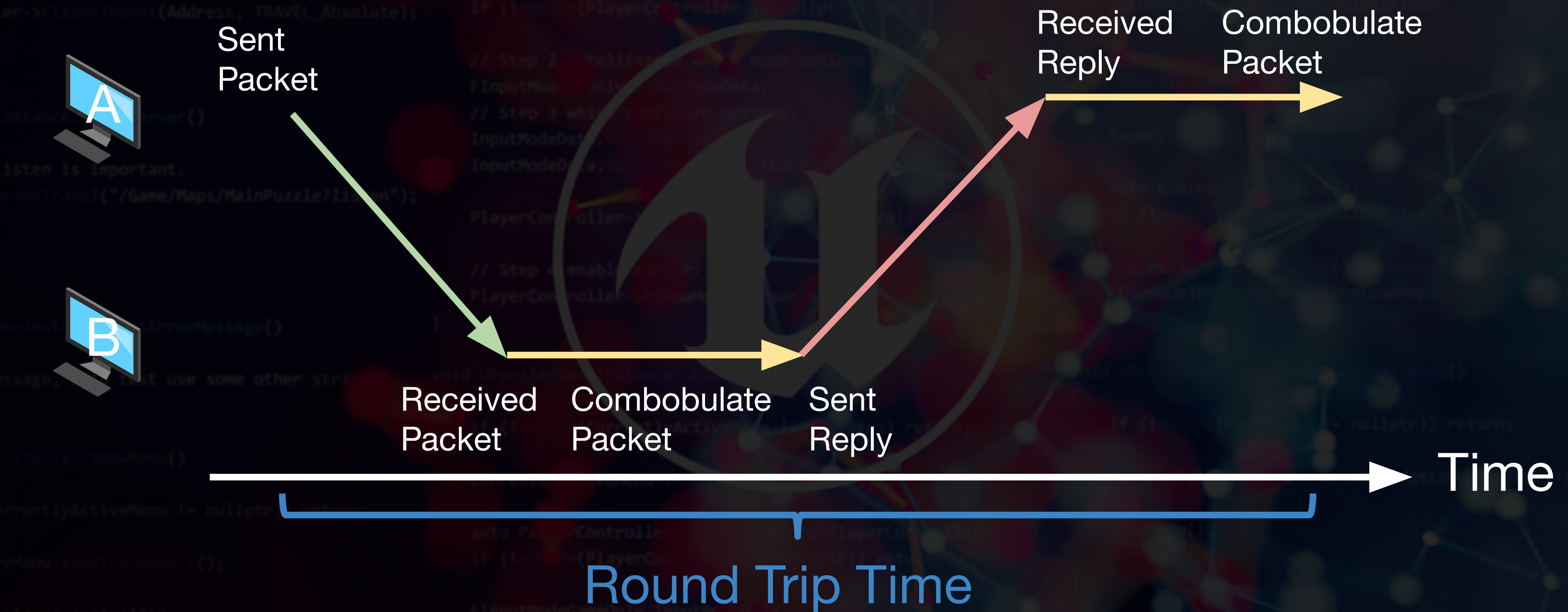


Best Effort Delivery



Live Google Slides at bit.ly/multiplayerslides

Lag



Live Google Slides at bit.ly/multiplayerslides



Why is the car glitching?

- Enable lag for yourself
- Play around with the game
- Take a look at what the server is seeing
- Can you explain why the game is glitching?
- Write up your explanation

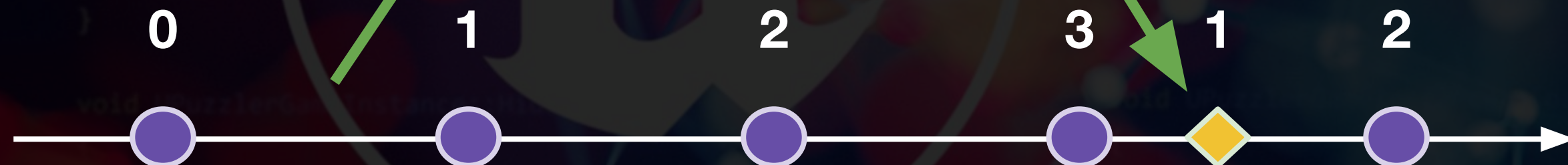


Lag Glitching

Server



Client



Live Google Slides at bit.ly/multiplayerslides

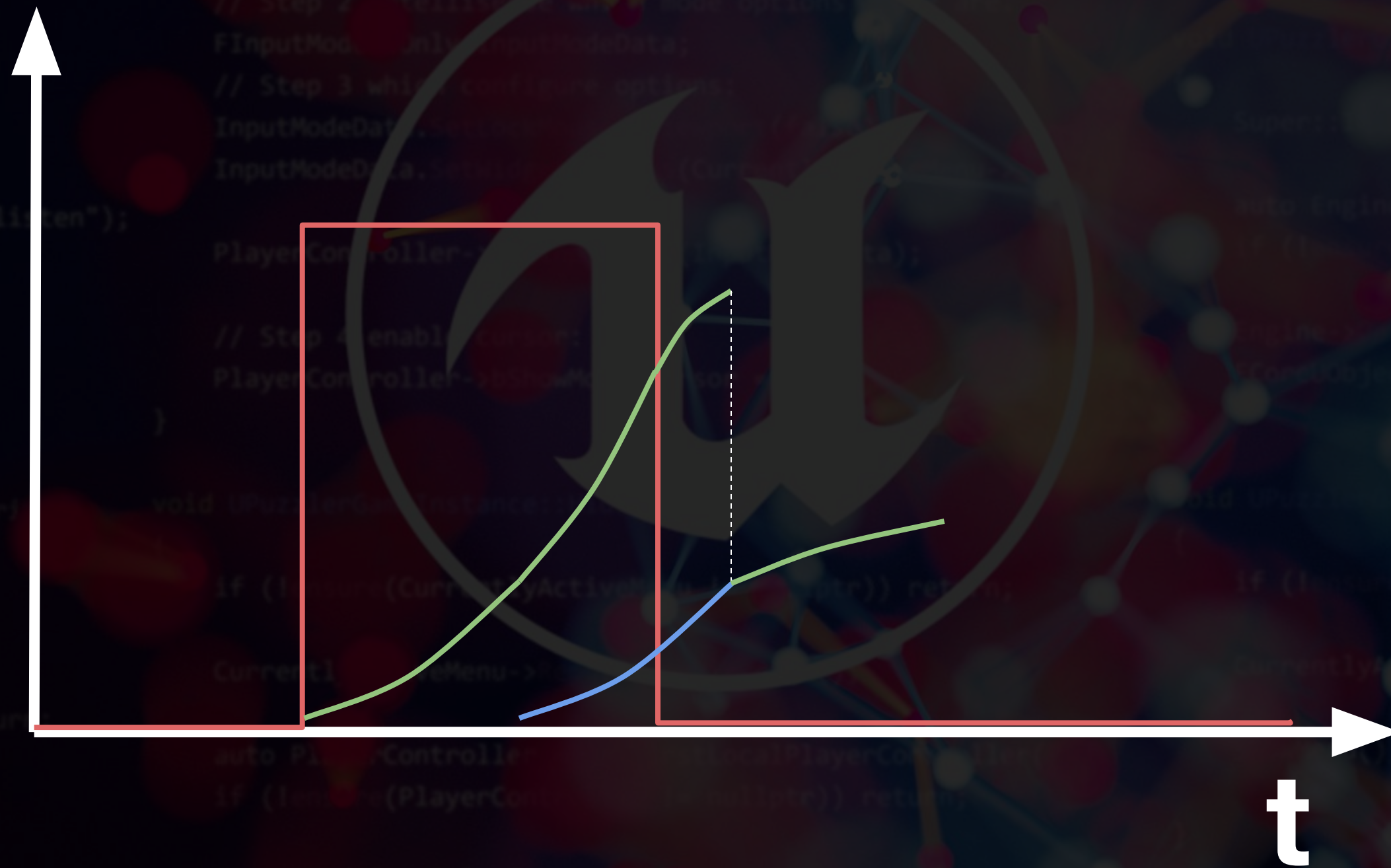


Replay Autonomous Moves



Acceleration With Lag

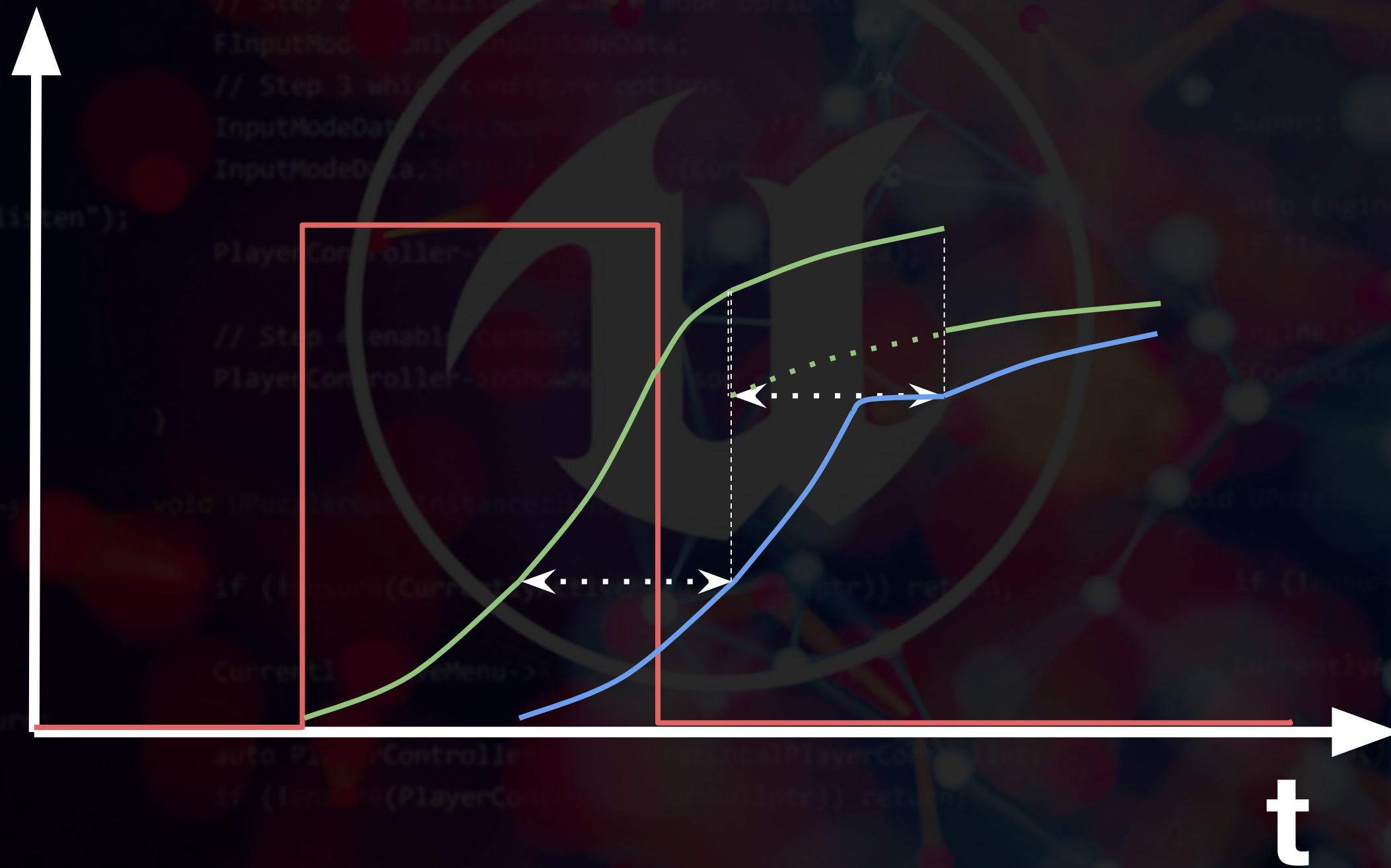
a
 x
 x_{rep}



Live Google Slides at bit.ly/multiplayerslides

Keeping Ahead Of The Server

a
x
x_{rep}



Live Google Slides at bit.ly/multiplayerslides

Comparison Of Methods

| | v1 |
|----------------------------|---------------------|
| Problem | Not smooth |
| Information sent to server | Throw |
| Between updates | Do nothing |
| Information received | Transform, Velocity |
| On Receipt | Overwrite local |

Live Google Slides at bit.ly/multiplayerslides



Comparison Of Methods

| | v1 | v2 |
|----------------------------|---------------------|----------|
| Problem | Not smooth | Lag |
| Information sent to server | Throw | |
| Between updates | Do nothing | Simulate |
| Information received | Transform, Velocity | |
| On Receipt | Overwrite local | |

Live Google Slides at bit.ly/multiplayerslides



Fill Out The Table

- What information needs to be sent?
- What do you do between updates?
- What information comes back?
- What do you do on server update?



Comparison Of Methods

| | v1 | v2 | v3 |
|----------------------------|---------------------|----------|---|
| Problem | Not smooth | Lag | - |
| Information sent to server | Throw | | |
| Between updates | Do nothing | Simulate | |
| Information received | Transform, Velocity | | Transform, Velocity, ServerTime |
| On Receipt | Overwrite local | | Replay controls since ServerTime |

Live Google Slides at bit.ly/multiplayerslides



Planning Client-Side Prediction

The High Level

- **OnTick:**

Create a move and send to the server.

- **OnReceiveMove:**

Simulate it on the server.

- **OnReceiveServerState:**

Replay local moves on top.



Pseudocode: OnTick

1. Create a new **Move**,
2. Save to a list of unacknowledged moves,
3. Send the move to the server,
4. Simulate the move locally.



Pseudocode: **OnReceiveMove**

1. Check that the move is valid, (No cheating!)
2. Simulate the move,
3. Send the canonical **State** to the clients.



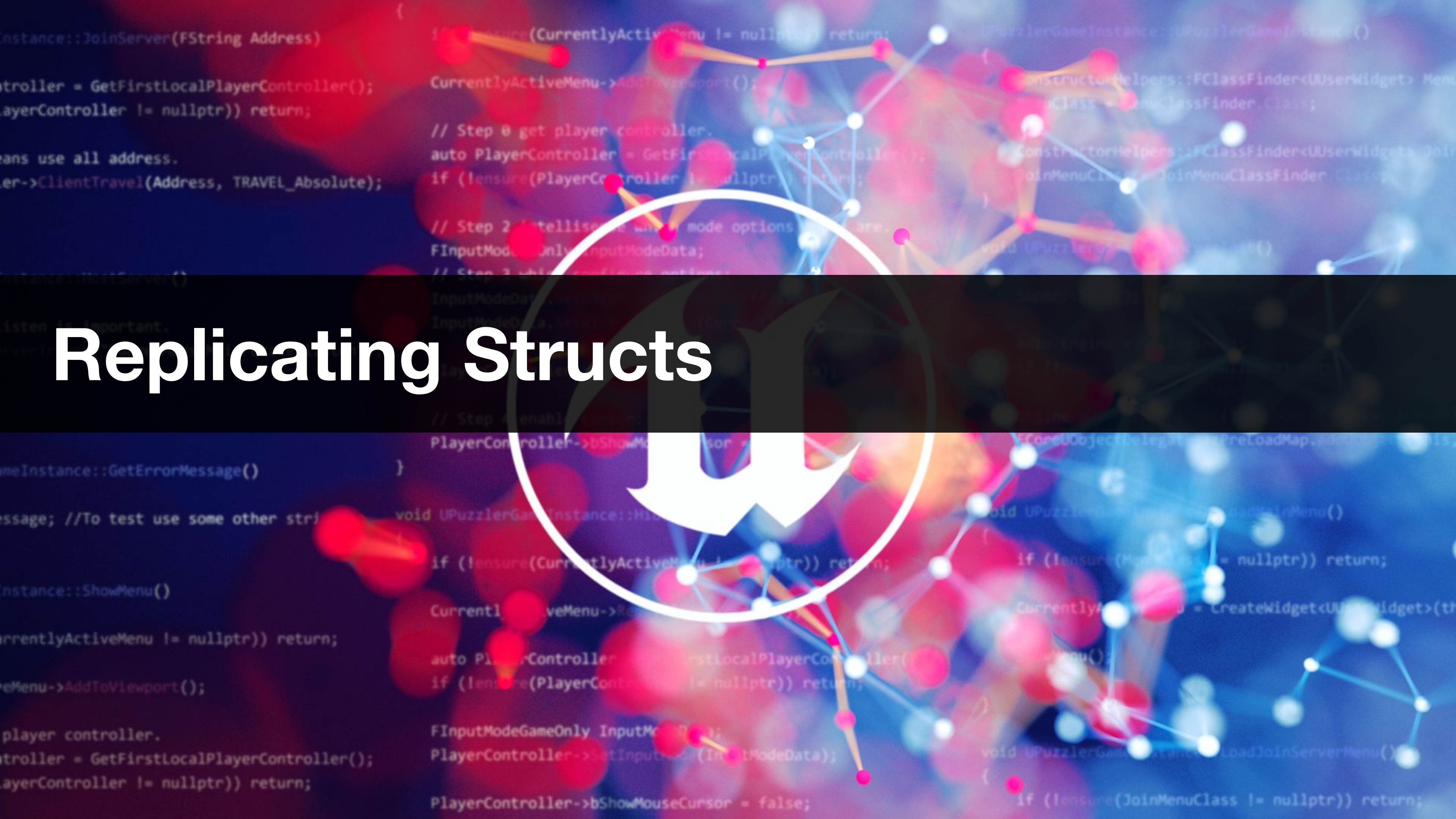
Pseudocode: **OnReceiveServerState**

1. Remove all moves included in state,
2. Reset to server state,
3. Replay/simulate unacknowledged moves.

Create The Structs

- Review the pseudocode
- What data does the **Move** and **State** include?
- Create structs for both
- No need to use them yet.





```
Instance::JoinServer(FString Address)  
  
controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
rans use all address.  
er->ClientTravel(Address, TRAVEL_Absolute);
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;  
CurrentlyActiveMenu->AddToViewport();  
  
// Step 0 get player controller.  
auto PlayerController = GetFirstLocalPlayerController();  
if (!ensure(PlayerController != nullptr)) return;  
  
// Step 2 Intellisece which mode options are.  
FInputModeGameOnly InputModeData;  
// Step 3 which confides options:  
InputModeData.SetInputMode(FInputModeGameOnly);  
InputModeData.SetCapture(true);  
PlayerController->SetInputMode(InputModeData);  
  
// Step 4 enable cursor:  
PlayerController->bShowMouseCursor = true;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()  
{  
    ConstructorHelpers::FClassFinder<UserWidget> MenuClass(TEXT("/Game/MenuClass"));  
    MenuClass = MenuClass.Class;  
  
    ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass(TEXT("/Game/JoinMenuClass"));  
    JoinMenuClass = JoinMenuClass.Class;  
}  
  
void UPuzzlerGameInstance::LoadMainMenu()  
{  
    Super::LoadMainMenu();  
  
    auto Engine = GEngine->GetWorld();  
    if (!ensure(MenuClass != nullptr)) return;  
    Engine->AddWidgetFromLibrary(MenuClass);  
}
```

```
Instance::HostServer()  
  
listen is important.  
ServerT
```

```
void UPuzzlerGameInstance::HideMainMenu()  
{  
    if (!ensure(CurrentlyActiveMenu != nullptr)) return;  
    CurrentlyActiveMenu->RemoveFromParent();  
  
    auto PlayerController = GetFirstLocalPlayerController();  
    if (!ensure(PlayerController != nullptr)) return;  
  
    FInputModeGameOnly InputModeData;  
    PlayerController->SetInputMode(InputModeData);  
  
    PlayerController->bShowMouseCursor = false;
```

```
CoreObjectDelegates::PreLoadMap.Add(this, &UPuzzlerGameInstance::PreLoadMap);  
  
void UPuzzlerGameInstance::LoadJoinServerMenu()  
{  
    if (!ensure(MenuClass != nullptr)) return;  
    CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);  
    CurrentlyActiveMenu->AddToViewport();  
  
    void UPuzzlerGameInstance::LoadJoinServerMenu()  
    {  
        if (!ensure(JoinMenuClass != nullptr)) return;
```

Replicating Structs

Pseudocode Overview

OnTick

1. Create a new **Move**,
2. Save to a list of unacknowledged moves,
3. Send the move to the server,
4. Simulate the move locally.

OnReceiveServerState

1. Remove all moves included in state,
2. Reset to server state,
3. Replay/simulate unacknowledged moves.

OnReceiveMove

1. Check that the move is valid, (No cheating!)
2. Simulate the move,
3. Send the canonical **State** to the clients.



What We Already Have (Sort Of)

OnTick

1. Create a new **Move**,
2. *Save to a list of unacknowledged moves,*
3. Send the move to the server,
4. Simulate the move locally.

OnReceiveServerState

1. *Remove all moves included in state,*
2. Reset to server state,
3. *Replay/simulate unacknowledged moves.*

OnReceiveMove

1. Check that the move is valid, (No cheating!)
2. Simulate the move,
3. Send the canonical **State** to the clients.



Consolidate The RPC

- Use only the **Move** struct
- Where should the struct be created?
- Which client should create it?
- What should the server do with the data?



Simulating A Move

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
rans use all address.  
er->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
listen is important.
```

```
Server
```

```
meInstance::GetErrorMessage()
```

```
essage; //To test use some other str
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
reMenu->AddToViewport();
```

```
player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisece which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which confides options;
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> Menu
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> Join
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```


Pseudocode

OnTick

1. Create a new **Move**,
2. *Save to a list of unacknowledged moves,*
3. Send the move to the server,
4. Simulate the move locally.

OnReceiveServerState

1. *Remove all moves included in state,*
2. Reset to server state,
3. *Replay/simulate unacknowledged moves.*

OnReceiveMove

1. Check that the move is valid, (No cheating!)
2. **Simulate the move,**
3. Send the canonical **State** to the clients.

Implement SimulateMove

- Pull out of **Tick**.
- Where will you get the input data?
- Where will you get **DeltaTime**?



Unacknowledged Move Queue

Pseudocode

OnTick

1. Create a new **Move**,
2. *Save to a list of unacknowledged moves,*
3. Send the move to the server,
4. Simulate the move locally.

OnReceiveServerState

1. *Remove all moves included in state,*
2. Reset to server state,
3. *Replay/simulate unacknowledged moves.*

OnReceiveMove

1. Check that the move is valid, (No cheating!)
2. Simulate the move,
3. Send the canonical **State** to the clients.



Prune The Queue

- Write **ClearAcknowledgedMoves**
- Call it from **OnRep_ServerState**
- Use the **TArray** documentation to help you
- What makes a move stale?



Simulating Unacknowledged Moves

Pseudocode

OnTick

1. Create a new **Move**,
2. *Save to a list of unacknowledged moves,*
3. Send the move to the server,
4. Simulate the move locally.

OnReceiveServerState

1. *Remove all moves included in state,*
2. Reset to server state,
3. *Replay/simulate unacknowledged moves.*

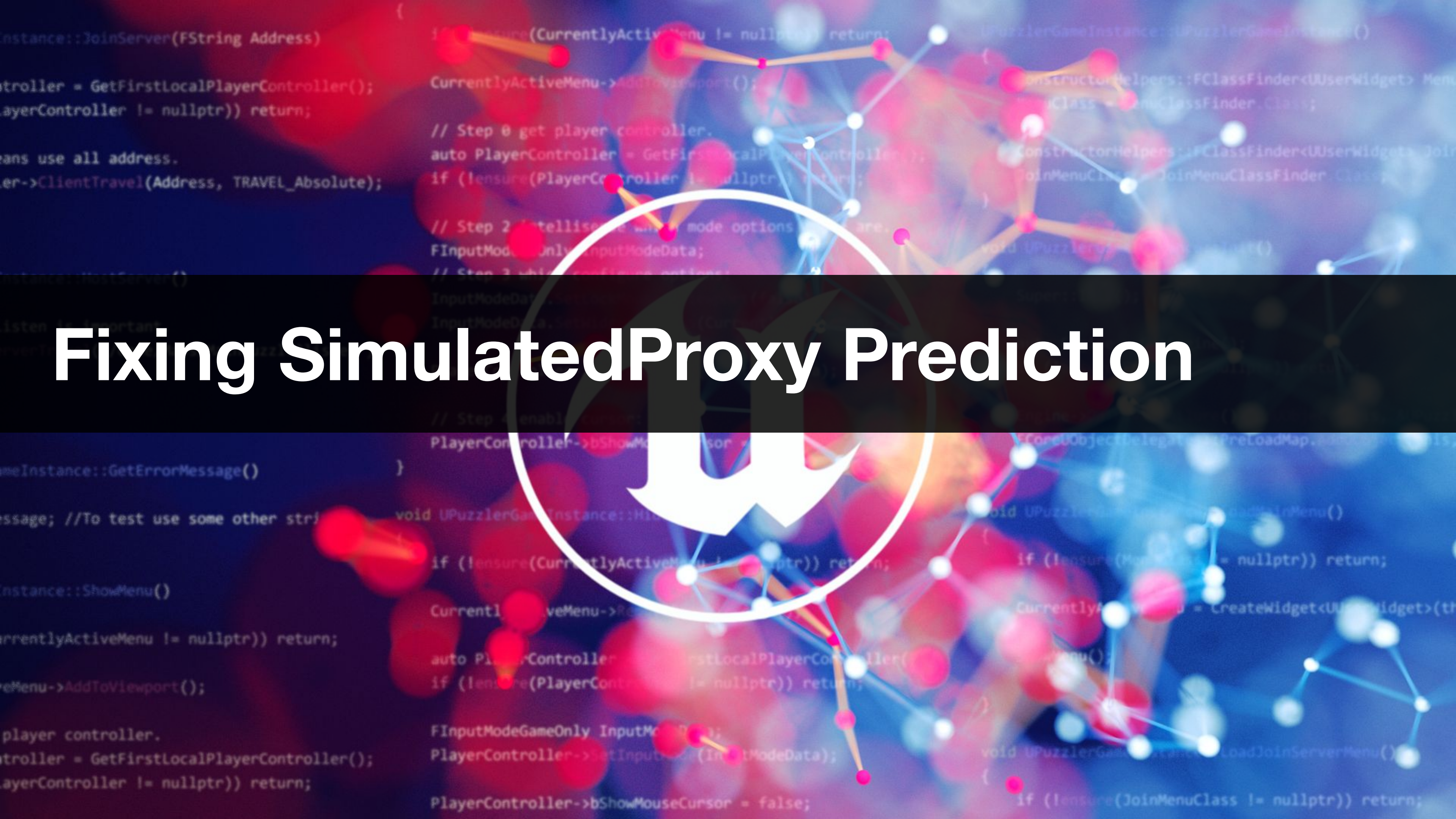
OnReceiveMove

1. Check that the move is valid, (No cheating!)
2. Simulate the move,
3. Send the canonical **State** to the clients.

Make It Glitch

- Glitching is still possible if we disagree
- Brainstorm why we might disagree
- Share on the forum
- Try to reproduce a glitch.





Fixing SimulatedProxy Prediction

Fix Client Prediction

- Why is the client jumping?
- What could we do between updates?
- Implement your solution.



Our Current Solution



Live Google Slides at bit.ly/multiplayerslides



Interpolating Client Position



Live Google Slides at bit.ly/multiplayerslides



Refactoring Into Components

Red-Green-Refactor Loop

Introduce a new
requirement

Red

Implement it
quick and dirty

Clean

Green

(and messy)

Refactor

Live Google Slides at bit.ly/multiplayerslides



Plan The Refactor

- Identify the “code smell”
- Choose a refactor
- Plan that refactor.



Refactor Plan

GoKart Actor

- Input binding

Movement Component

- Simulating physics

Replication Component

- Replicating movement

Live Google Slides at bit.ly/multiplayerslides





```
Instance::JoinServer(FString Address)  
  
controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
rans use all address.  
er->ClientTravel(Address, TRAVEL_Absolute);
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;  
CurrentlyActiveMenu->AddToViewport();  
  
// Step 0 get player controller.  
auto PlayerController = GetFirstLocalPlayerController();  
if (!ensure(PlayerController != nullptr)) return;  
  
// Step 2 Intellisece which mode options are.  
FInputModeGameOnly InputModeData;  
// Step 3 which confides options:  
InputModeData.SetInputMode(FInputModeGameOnly);  
InputModeData.SetCapture(true);  
PlayerController->SetInputMode(InputModeData);  
  
// Step 4 enable cursor:  
PlayerController->bShowMouseCursor = true;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()  
{  
    ConstructorHelpers::FClassFinder<UserWidget> MenuClass(TEXT("/Game/Menu/MenuClass"));  
    MenuClass = MenuClass.Class;  
    ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass(TEXT("/Game/Menu/JoinMenuClass"));  
    JoinMenuClass = JoinMenuClass.Class;  
}  
  
void UPuzzlerGameInstance::LoadMainMenu()  
{  
    Super::LoadMainMenu();  
}
```

```
Instance::HostServer()  
  
listen is important.  
Server->
```

```
void UPuzzlerGameInstance::HideMainMenu()  
{  
    if (!ensure(CurrentlyActiveMenu != nullptr)) return;  
    CurrentlyActiveMenu->RemoveFromParent();  
    auto PlayerController = GetFirstLocalPlayerController();  
    if (!ensure(PlayerController != nullptr)) return;  
  
    FInputModeGameOnly InputModeData;  
    PlayerController->SetInputMode(InputModeData);  
  
    PlayerController->bShowMouseCursor = false;
```

```
CoreObjectDelegates::PreLoadMap.AddLambda([this]  
{  
    void UPuzzlerGameInstance::LoadMainMenu()  
    {  
        if (!ensure(MenuClass != nullptr)) return;  
        CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);  
        AddWidget(CurrentlyActiveMenu);  
    }  
});  
  
void UPuzzlerGameInstance::LoadJoinServerMenu()  
{  
    if (!ensure(JoinMenuClass != nullptr)) return;
```

Extracting A Movement Component

Refactor Plan

GoKart Actor

- Input binding

Movement Component

- Simulating physics

Replication Component

- Replicating movement



Make It Build

- Pull across the method implementations
- Change the class namespace
- Prefix all calls to Actor functions with:
GetOwner()->
- Fix references in **GoKart.cpp**
- Create any accessor methods you need.
- Fix other build errors.



Extracting A Replication Component

Refactor Plan

GoKart Actor

- Input binding

Movement Component

- Simulating physics

Replication Component

- Replicating movement



Refactor!

- Pull across the method implementations
- Change the class namespace
- Prefix all calls to Actor functions with:
GetOwner()->
- Fix references in **GoKart.cpp**
- Create any accessor methods you need.
- Fix other build errors.





Decoupling Movement & Replication

Refactor Plan

GoKart Actor

- Input binding

Movement Component

- Simulating physics

Replication Component

- Replicating movement



Make Replication Work

- Expose the last move
- Replace the simulating code in Replicator
- How to change **Server_SendMove()**?
- Can we make some methods private now?
- When shouldn't the MC simulate?



Linear Interpolation For Position

Linear Interpolation (Lerp)



Live Google Slides at bit.ly/multiplayerslides



Client Interpolation

X



Live Google Slides at bit.ly/multiplayerslides

Early Updates

X



Live Google Slides at bit.ly/multiplayerslides

Late Updates

X



Live Google Slides at bit.ly/multiplayerslides

Write The Pseudocode

- Consider what you will do on tick
- How about **OnRep**?
- User the Lerp function:

Lerp(A, B, alpha)

- What data do you need to store?



Pseudo Code

OnTick:

TargetLocation = ServerState.Location

LerpRatio = TimeSinceUpdate / TimeBetweenLastUpdates

NextLocation = Lerp(StartLocation, TargetLocation, LerpRatio)

SetLocation(NextLocation)

OnRep:

StartLocation = GetLocation()

Live Google Slides at bit.ly/multiplayerslides





FMath::Lerp For Client Interpolation

Pseudo Code

OnTick:

TargetLocation = ServerState.Location

LerpRatio = TimeSinceUpdate / TimeBetweenLastUpdates

NextLocation = Lerp(StartLocation, TargetLocation, LerpRatio)

SetLocation(NextLocation)

OnRep:

StartLocation = GetLocation()

Live Google Slides at bit.ly/multiplayerslides



Implement The Pseudocode

- Which member variables are missing?
- Implement **Tick**.
- Implement **OnRep**.
- Keep it neat.



FQuat::Slerp For Rotation



Why Can't We Lerp?

A = -100

B = 90



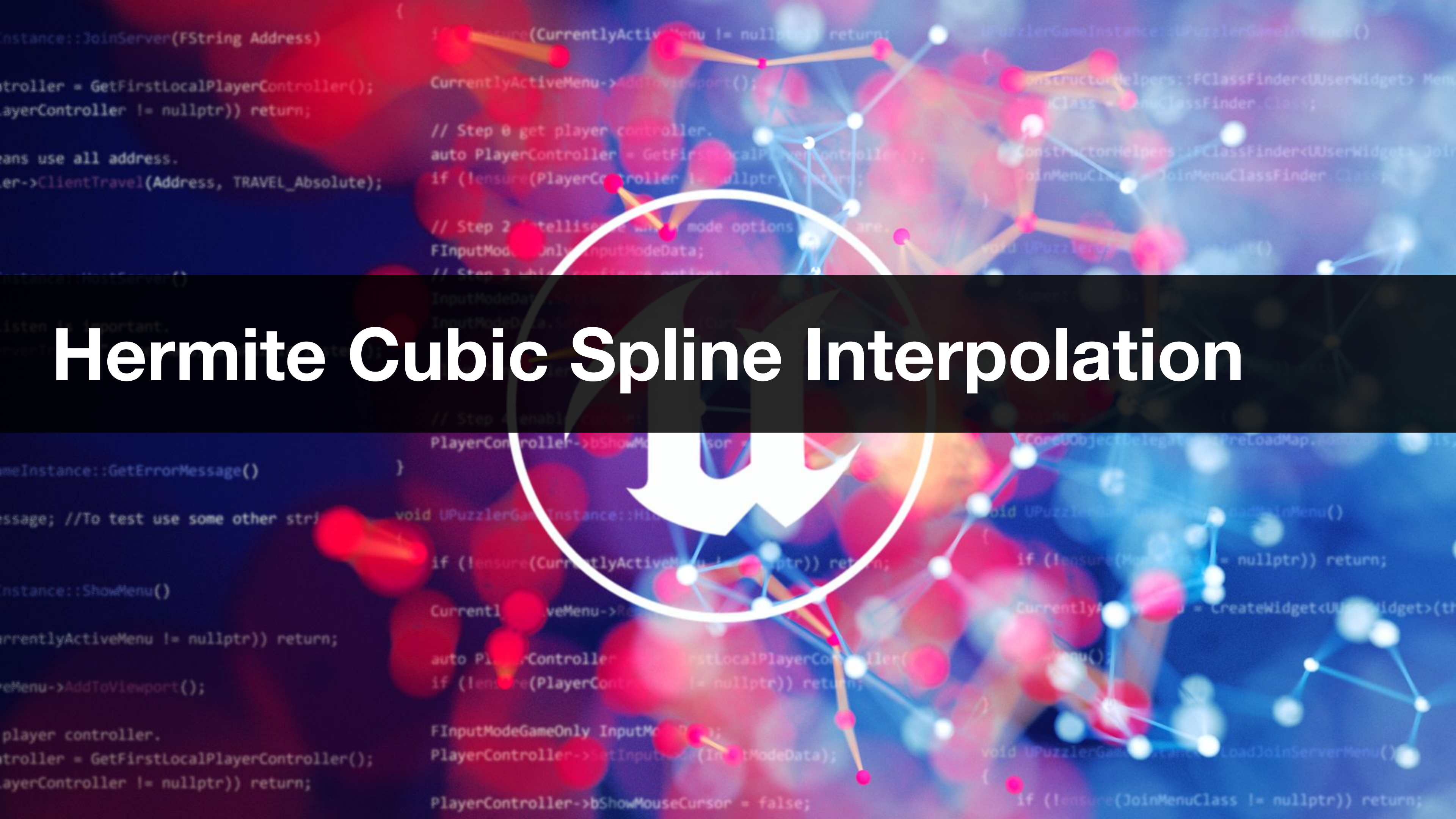
Live Google Slides at bit.ly/multiplayerslides



Slerp The Rotation

- Repeat the pattern
- Use **FQuat::Slerp**
- Test!





Hermite Cubic Spline Interpolation

Problem: Jarring Movement

y



x

Live Google Slides at bit.ly/multiplayerslides

Incorporate Velocity?

y



x

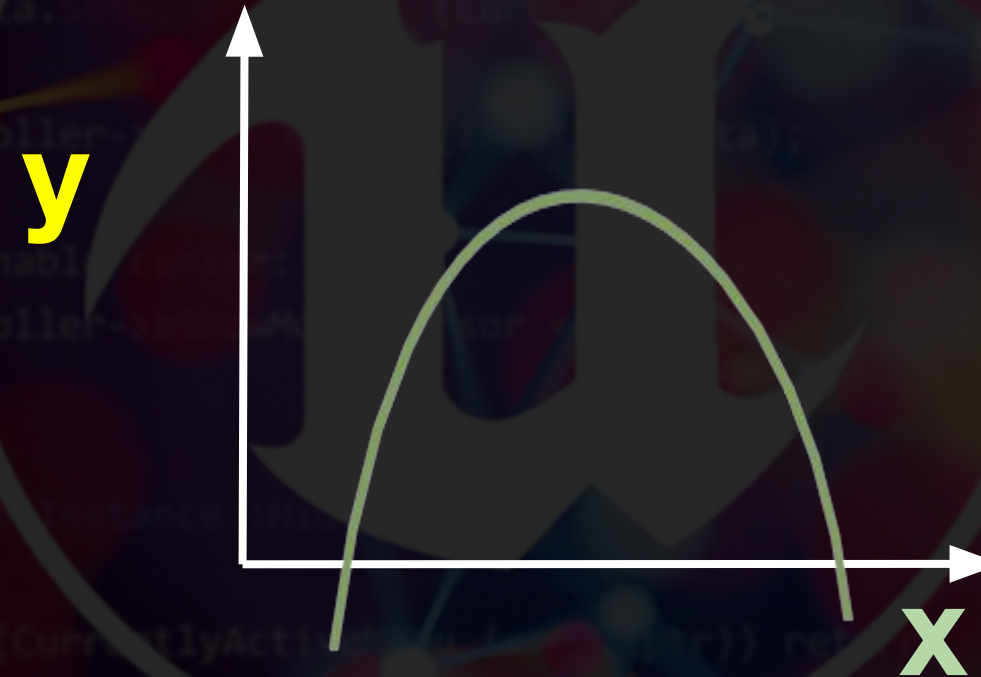
Live Google Slides at bit.ly/multiplayerslides

New Tool: Polynomials

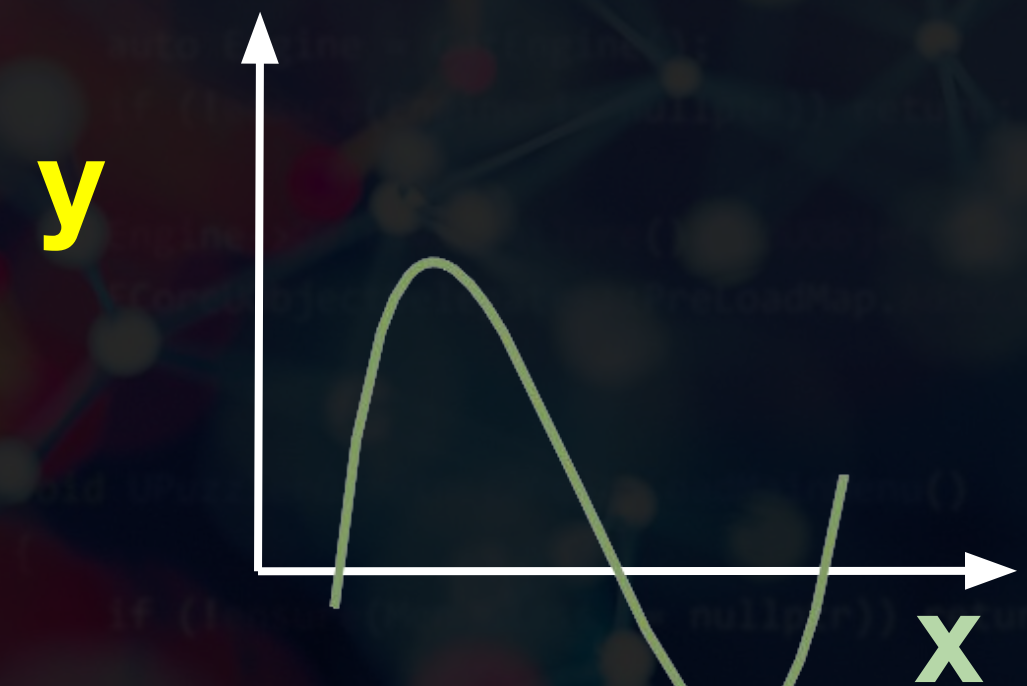
Linear



Quadratic



Cubic



More Complex

Live Google Slides at bit.ly/multiplayerslides

Which Is The Simplest Curve?

```
e all address.  
ientTravel(Address, TRAVEL_Absolute);
```

```
er::HostServer()
```

```
is important.
```

```
avel("/Game/Maps/MainPuzzle?listen");
```

```
ance::GetErrorMessage()
```

```
//To test use some other str
```

```
er::ShowMenu()
```

```
yActiveMenu != nullptr)) return;
```

```
>AddToViewport();
```

```
controller
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
If (IsValid(PlayerController) != nullptr)
```

```
// Step 2 - tell the game mode options
```

```
FInputMode_GameOnly InputModeData;
```

```
// Step 3 while configure options:
```

```
InputModeData.SetOverrideScalarProperty(FKey, 1.0f);
```

```
InputModeData.SetOverrideProperty(FKey, (CurrentGameMode));
```

```
PlayerController->SetInputMode(InputModeData);
```

```
// Step 4 - enable cursor:
```

```
PlayerController->ShowMouseCursor = EMouseCursor_Default;
```

```
)
```

```
void UPuzzleGame::Instance::HostServer()
```

```
If (IsValid(CurrentlyActiveMenu) != nullptr)) return;
```

```
CurrentlyActiveMenu->Play();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
If (IsValid(PlayerController) != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
Super::HostServer();
```

```
auto Engine = GEngine;
```

```
If (IsValid(Engine) != nullptr) return;
```

```
Engine->AddOnScreenDebugText(-1, 0, 0, "PuzzleGame::HostServer()");
```

```
CoreObjectDelegate::PreLoadMap;
```

```
void UPuzzleGame::Instance::HostServer()
```

```
{
```

```
If (IsValid(CurrentlyActiveMenu) != nullptr)) return;
```

```
CurrentlyActiveMenu->Play();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
If (IsValid(PlayerController) != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
InputModeData.SetOverrideScalarProperty(FKey, 1.0f);
```

```
InputModeData.SetOverrideProperty(FKey, (CurrentGameMode));
```

```
PlayerController->SetInputMode(InputModeData);
```

```
// Step 4 - enable cursor:
```

```
PlayerController->ShowMouseCursor = EMouseCursor_Default;
```



Which Is The Simplest Curve?

```
...e all address.  
...entTravel(Address, TRAVEL_Absolute);
```

```
...e::HostServer()
```

```
...is important.
```

```
...avel("/Game/Maps/MainPuzzle?listen");
```

```
...ance::GetErrorMessage()
```

```
...//To test use some other str
```

```
...e::ShowMenu()
```

```
...yActiveMenu != nullptr)) return;
```

```
...>AddToViewport();
```

```
...controller
```

```
...auto PlayerController = GetFirstLocalPlayerController();  
...if (!IsValid(PlayerController)) return;
```

```
...// Step 2 - tell the game mode options
```

```
...FInputMode_GameOnly InputModeData;
```

```
...// Step 3 while the game mode options
```

```
...InputModeData.SetProperty("bNoMouse", true);
```

```
...InputModeData.SetProperty("bNoKeyboard", true);
```

```
...PlayerController->SetInputMode(InputModeData);
```

```
...// Step 4 - enable cursor
```

```
...PlayerController->ShowMouseCursor();
```

```
...void UPuzzleGame::Instance::H
```

```
...if (!IsValid(CurrentlyActiveMenu)) return;
```

```
...CurrentlyActiveMenu->P
```

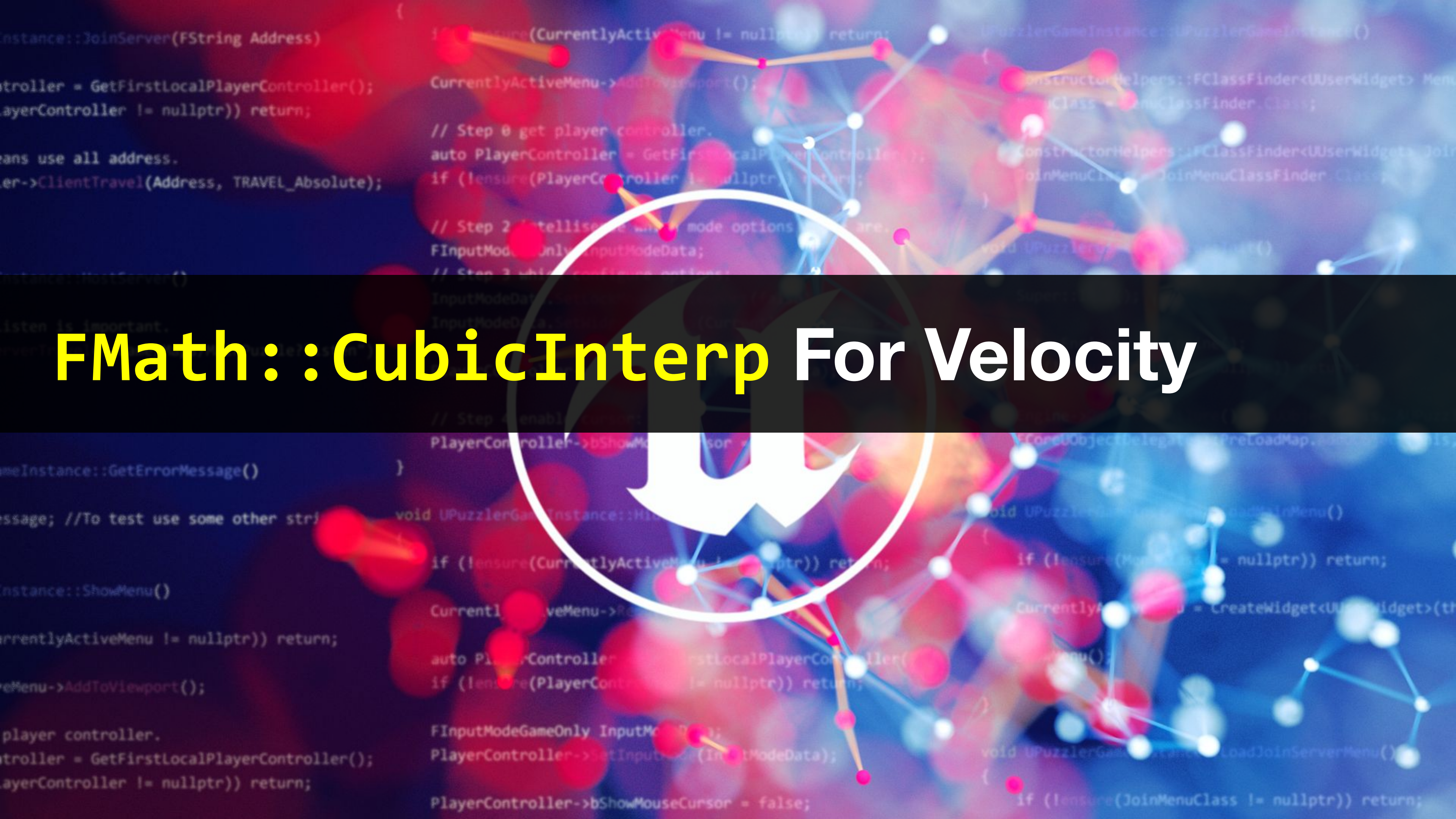
```
...auto PlayerController = GetFirstLocalPlayerController();
```

```
...if (!IsValid(PlayerController)) return;
```

```
...FInputMode_GameOnly InputModeData;
```

Cubic!





FMath::CubicInterp For Velocity

Cubic Interpolation And Velocity



Live Google Slides at bit.ly/multiplayerslides

Slope, Derivative And Velocity

$$\text{Slope} = \text{Derivative} \\ = \text{DeltaLocation} / \text{DeltaAlpha}$$

$$\text{Velocity} = \text{DeltaLocation} / \text{DeltaTime}$$

$$\text{DeltaAlpha} = \text{DeltaTime} / \text{TimeBetweenLastUpdates}$$

$$\text{Derivative} = \text{Velocity} * \text{TimeBetweenLastUpdates}$$

Live Google Slides at bit.ly/multiplayerslides



Interpolate Velocity

- Research **FMath::CubicInterpDerivative**
- Convert this to velocity
- Set the velocity on the movement component
- Test!



Refactoring With Structs

Pull Out Some Methods

- Pull out **CreateSpline**,
- **InterpolateLocation**,
- **InterpolateVelocity**,
- **InterpolateRotation**



Client Interpolation Mesh Offset

Manipulate The Offset

- Reset the actor location on rep
- Elsewhere, use the component location
- Do the same for rotation.



Advanced Cheat Protection

Prevent the DeltaTime Cheat

- Track the simulated time
- Ensure it's less than the server time
- Prevent multiplying moves
- Prevent long delta times.



End Of Course Wrap-up

That's all for now!

Slides after this point are being recorded as you read this. Comment on them to have your say!

Live Google Slides at bit.ly/multiplayerslides

