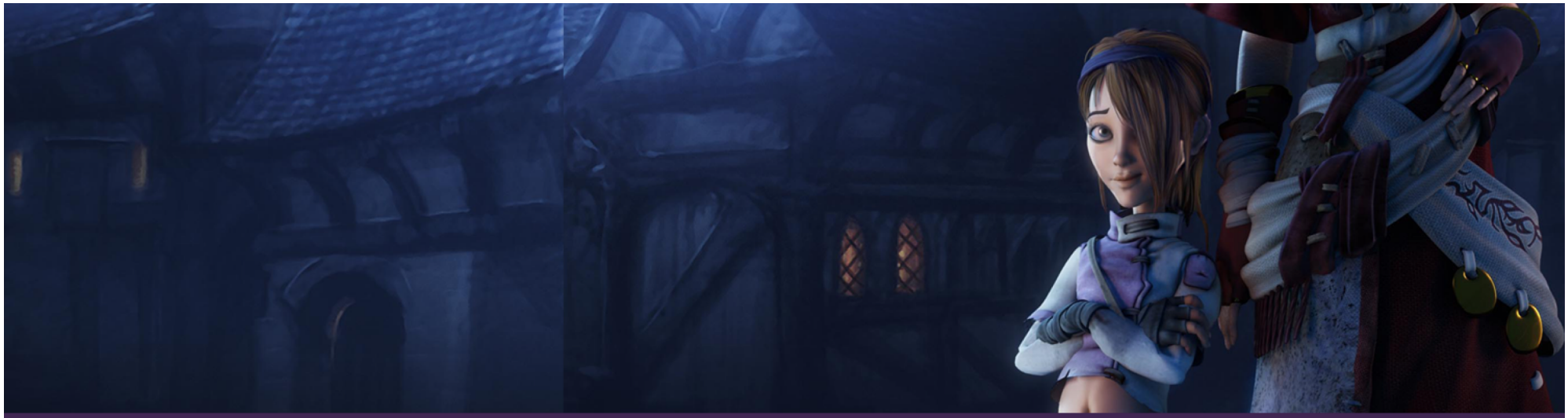


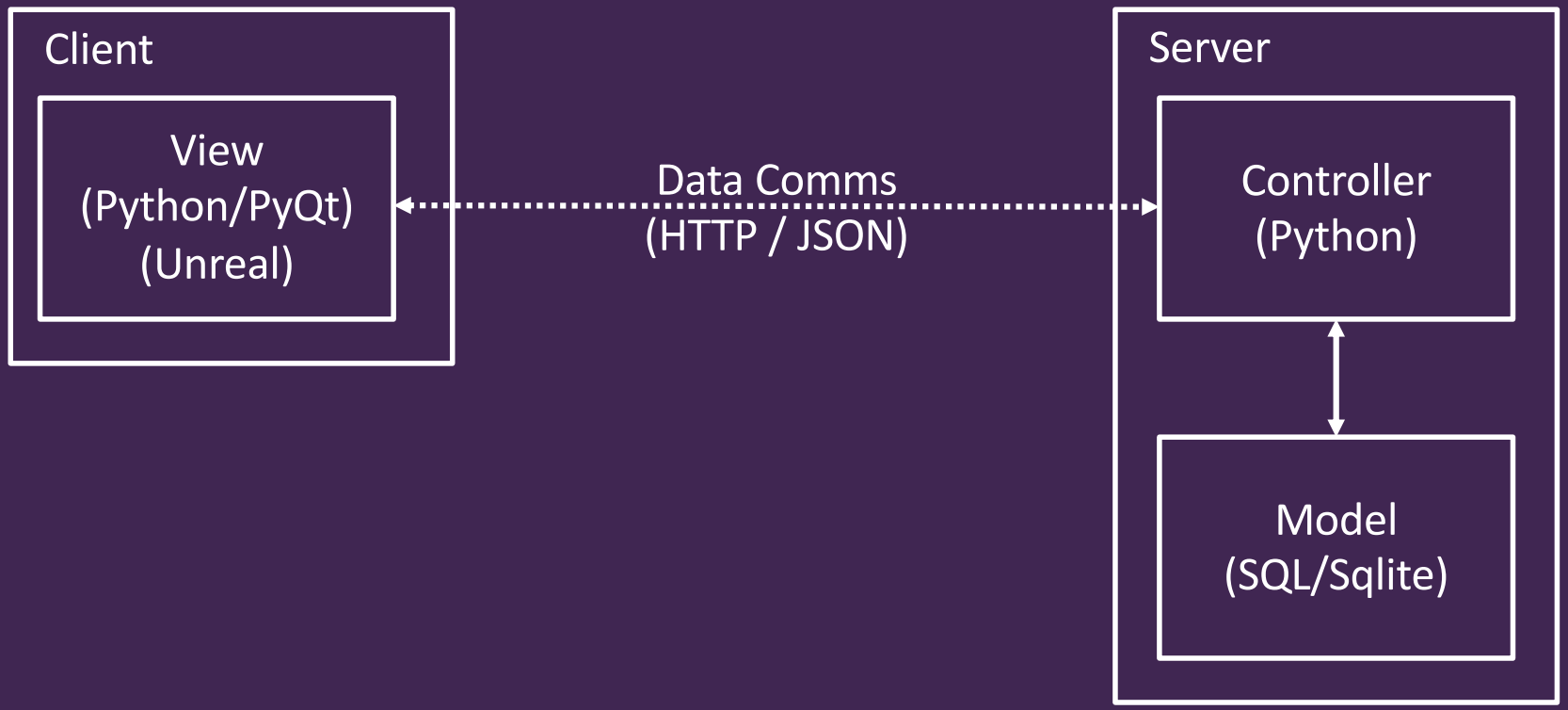
FALMOUTH
UNIVERSITY

Lecture 5: Networking with Game Engines

COMP280: Creative Computing
BSc(Hons) Computing for Games
BA(Hons) Game Development: Programming

- Learning outcomes
  - **Configure** Unreal Engine to support HTTP
  - **Integrate** JSON functionality into Unreal Engine
  - **Develop** a simple Unreal Engine application to show GET & POST functionality
  - **Extend** your Unreal Engine testbed for a real application

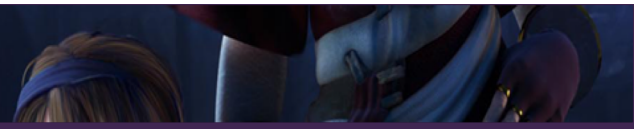
- Key Full stack Technologies



- Key Full stack Technologies
  - Last week:
    - Looked at broad Python-based client-server architecture
      - Addressed issues with JSON for better data transfer
      - SQL for persistent data management

    → Basis of a powerful backend framework

- This week
  - Game engine integration approaches
  - Game & Architectural design considerations



Unity Integration

- Unity Integration
  - Unity leverages C# & .NET as its run-time architecture
    - .NET is very good for client/server development
      - Often used as an alternative to Python
      - Choice is often:
        - » Python (bespoke / Django)
        - » C# (.Net / ASP.NET)
        - » JavaScript (bespoke / Node.js)

Unity Integration

– Generic C# HTTP Client functionality

```
using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("http://localhost/");
    ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };


    try
    {
        var response = client.GetAsync("player_data").Result;

        if (response.IsSuccessStatusCode)
        {
            var responseContent = response.Content;

            // by calling .Result you are synchronously reading the result
            string responseString = responseContent.ReadAsStringAsync().Result;

            var obj = JsonConvert.DeserializeObject<List<Entry>>(responseString);

            foreach( var entry in obj)
            {
                Console.WriteLine(entry.name +" " + entry.score);
            }
        }

    }
    catch (HttpRequestException e)
    {
        Console.WriteLine("\nException Caught!");
        Console.WriteLine("Message :{0} ", e.Message);
    }
```

- Functionally identical to Python
  - GetAsync() is ‘GET’
  - Slightly more code, Python is more geared towards rapid development
- Python uses 3rd party s/w for JSON support
  - Newtonsoft
  - Typed nature of C# combined with refection makes JSON relatively easy to work with

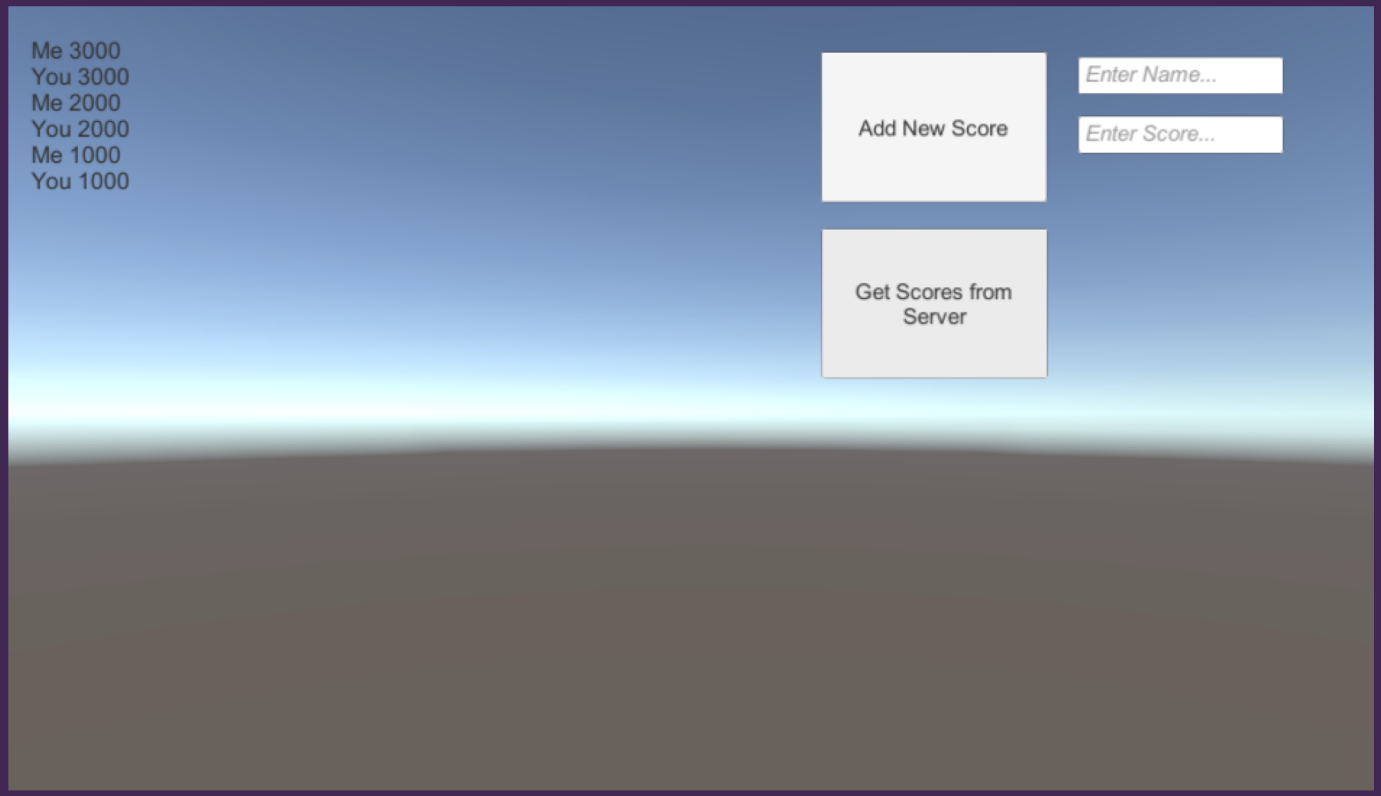
Unity Integration

– Generic C# HTTP Client functionality

```
try
{
    var new_score = new Entry("c# test", 999);

    var content = new StringContent(JsonConvert.SerializeObject(new_score));

    var response = client.PostAsync("add_score",content).Result;

    if (response.IsSuccessStatusCode)
    {
        var responseContent = response.Content;

        // by calling .Result you are synchronously reading the result
        string responseString = responseContent.ReadAsStringAsync().Result;

        var obj = JsonConvert.DeserializeObject<List<Entry>>(responseString);

        Console.WriteLine(responseString);
    }
}
catch (HttpRequestException e)
{
    Console.WriteLine(e.Message);
    throw;
}
```

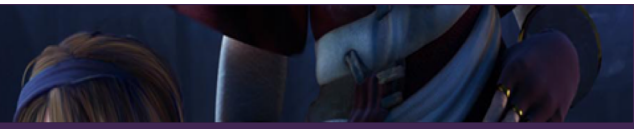
- Functionally identical to Python
  - PostAsync() is ‘POST’
  - Slightly more code, Python is more geared towards rapid development

- Unity Integration
  - Generic C# HTTP Client functionality
  - Can take this straight into Unity project



- Unity Integration
  - Create a UI with:
    - A multi-line textbox to display score info
    - A button widget to get scores from the server
    - A button widget to send score to the server
    - 2 text input widgets to enter name + score for new data

- Unity Integration
  - Create a UI with:
    - Add Score Button
      - Uses OnClick()
        - » Calls scorewidget.AddScore
    - Get Scores Button
      - Uses OnClick()
        - » Calls scorewidget.GetScores()
    - These functions call the standalone .NET code from before
      - Also copy data for AddScore
      - And paste data into textbox for GetScores



- Unreal Integration

- Unreal Integration
  - Works in much the same way
    - However, uses Unreal C++ wrapper code to deal with:
      - HTTPClient
      - JSON data
    - Unreal ‘documentation’
      - https://wiki.unrealengine.com/Http-requests
      - Create an actor derived class to hold requests and associated responses
      - ‘broadly’ similar to Unity approach

- Unreal Integration
  - HTTP data definitions

```
USTRUCT()
struct FRequest_ScoreDetails {
    GENERATED_BODY()
    UPROPERTY() FString name;
    UPROPERTY() int score;

    FRequest_ScoreDetails() {}
};

USTRUCT()
struct FResponse_ScoreDetails {
    GENERATED_BODY()
    UPROPERTY() TArray< FRequest_ScoreDetails> details;

    FResponse_ScoreDetails() {}
};
```

```
class HiScoreEntry
{
    public HiScoreEntry()
    {
    }

    public HiScoreEntry(String name, int score)
    {
        this.name = name;
        this.score = score;
    }

    public String name;
    public int score;
}

class HiScoreEntries
{
    public HiScoreEntries()
    {

    }

    public List<HiScoreEntry> details;
}
```

- Unreal Integration
  - HTTP data transfer (GET)

```
void AMyHTTPRequest::OnGetScoresButtonPress()
{
    GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::Green, TEXT("OnGetScoresButtonPress"));

    TSharedRef<IHttpRequest> Request = GetRequest("get_scores");
    Request->OnProcessRequestComplete().BindUObject(this, &AMyHTTPRequest::GetScores_Response);
    Send(Request);
}
```

  - GET (GetRequest)
    - Uses ‘subroute’ as a command
    - Passes response function to be called on completion

- Unreal Integration
  - HTTP data transfer (GET)

```
void AMyHTTPRequest::GetScores_Response(FHttpRequestPtr Request, FHttpResponsePtr Response, bool bWasSuccessful)
{
    if (!ResponseIsValid(Response, bWasSuccessful))
    {
        GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::Red, TEXT("GetScores_Response-FAIL"));

        return;
    }

    GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::White, TEXT("GetScores_Response-SUCCESS"));

    FString rawJson = Response->GetContentAsString();

    GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::White, rawJson);

    FResponse_ScoreDetails result;
    FJsonObjectConverter::JsonObjectStringToUStruct<FResponse_ScoreDetails >(Response->GetContentAsString(), &result, 0, 0);

    for (int32 Index = 0; Index != result.details.Num(); ++Index)
    {
        FString JoinedStr;
        JoinedStr += FString::FromInt(Index);
        JoinedStr += TEXT(" ");
        JoinedStr += result.details[Index].name;
        JoinedStr += TEXT(" ");
        JoinedStr += FString::FromInt(result.details[Index].score);

        GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::Yellow, JoinedStr);
    }

}
```

- Unreal Integration
  - HTTP data transfer (POST)

```
void AMyHTTPRequest::OnAddScoreButtonPress()
{
    GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::Blue, TEXT("OnAddScoreButtonPress"));

    FString ContentJsonString;
    FRequest_ScoreDetails score;
    score.name = "aaa";
    score.score = 100;

    GetJsonStringFromStruct<FRequest_ScoreDetails>(score, ContentJsonString);

    TSharedRef<IHttpRequest> Request = PostRequest("add_score", ContentJsonString);
    Request->OnProcessRequestComplete().BindUObject(this, &AMyHTTPRequest::PostScoreDetails_Response);
    Send(Request);
}
```

  - POST (PostRequest)
    - Uses ‘subroute’ as a command
    - Passes response function to be called on completion
    - Passes specific send data as json

- Unreal Integration
  - HTTP data transfer (POST)

```
void AMyHTTPRequest::PostScoreDetails_Response(FHttpRequestPtr Request, FHttpResponsePtr Response, bool bWasSuccessful)
{
    if (!ResponseIsValid(Response, bWasSuccessful))
    {
        GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::Red, TEXT("PostScoreDetails_Response-FAIL"));

        return;
    }

    GEngine->AddOnScreenDebugMessage(-1, 100.0f, FColor::White, TEXT("PostScoreDetails_Response-SUCCESS"));

    FResponse_Login LoginResponse;
    GetStructFromJsonString<FResponse_Login>(Response, LoginResponse);

    UE_LOG(LogTemp, Warning, TEXT("Id is: %d"), LoginResponse.id);
    UE_LOG(LogTemp, Warning, TEXT("Name is: %s"), *LoginResponse.name);
}
```

  - Response
    - Unpack JSON response (this is unrelated in this instance)

- Unreal Integration
  - Unreal's JSON engine doesn't support anonymous collections
    - Will silently fail with errors logged in output log
      - (window->developer tools->output log)
    - Add a label to contain data

```
result = json.dumps({"details": list})
self.wfile.write(result.encode())
```

- Unreal Integration
  - Unreal's HTTP engine will fail without trailing / in url

```
FHttpModule* Http;
FString ApiBaseUrl = "http://localhost:8000/";
```

    - Just reports that server rejected connection

- In general
  - Typed languages (C# & C++) lose the immediacy (and speed of development) of Python
    - As JSON data needs to be made explicit
      - Need to think about your data transfer formats
      - Cost in changing it between client and server
  - POST
    - Transmit & receive data
  - GET
    - Receive data only
      - Embed data into subroute / path and unpack

- In general
  - GET data
    - Embed using <subroute>?<label>=<value>&<label>=<value>

```
TSharedRef<IHttpRequest> Request = GetRequest("get_scores?name=dave%20smith&age=47.5");
```

      - No spaces, = or & in labels & values (use ascii %20 etc)

    - Unpack with urlib.parse

```
query_components = parse_qs(urlparse(self.path).query)
```

      - Becomes a dictionary of values

- Conclusions
  - HTTP /JSON is are fine transport & data layers
    - Works with Python, C# / C++ / Unity / Unreal
  - Moving from untyped (Python) to typed (C# / C++) creates complications
    - Particularly with JSON
    - These are soluble, but be aware of scope for failure / issues
  - Scope for issues (Unity) with time to complete HTTP requests
    - Threading / yield are solutions

- Questions