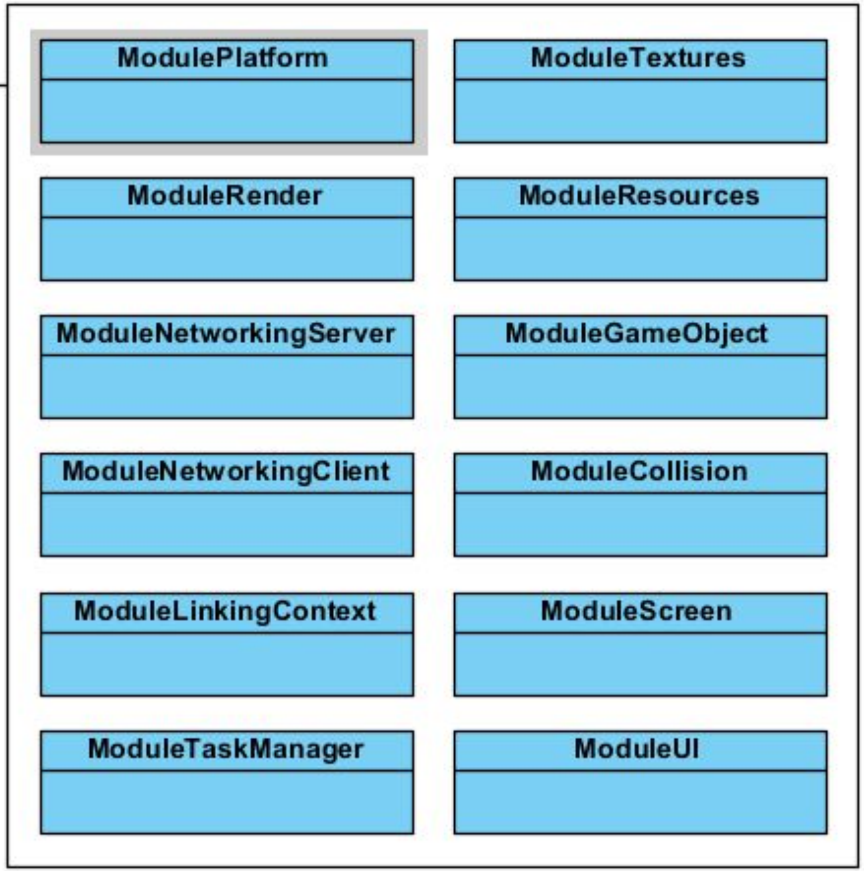
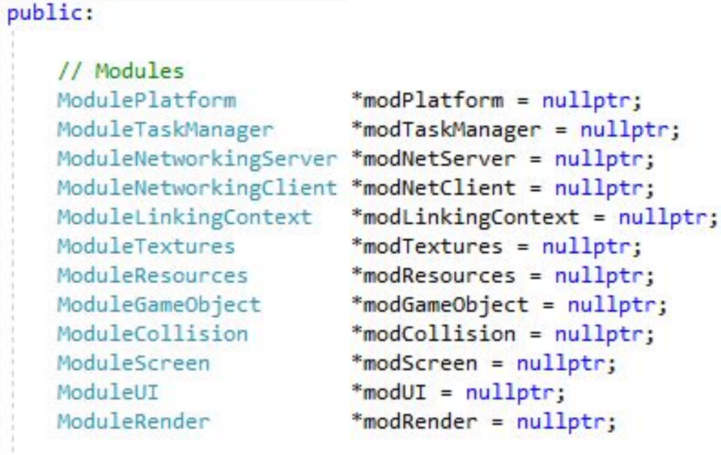
Multiplayer Game in C++

Engine basics

Networks and Online Games

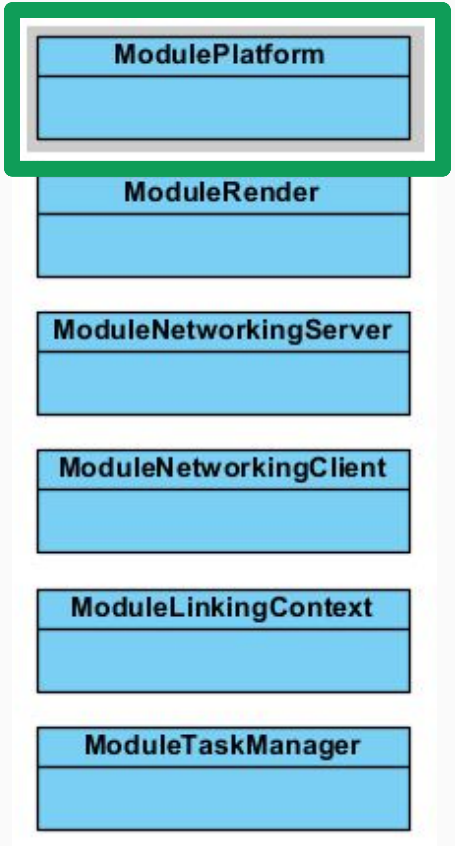
Modules



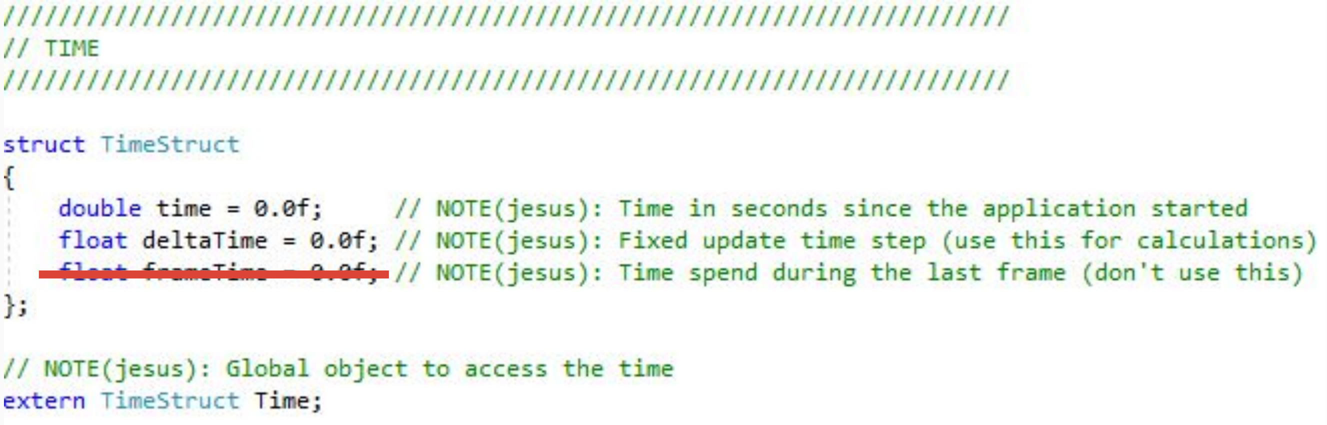


```
public:

    // Modules
    ModulePlatform         *modPlatform = nullptr;
    ModuleTaskManager      *modTaskManager = nullptr;
    ModuleNetworkingServer *modNetServer = nullptr;
    ModuleNetworkingClient *modNetClient = nullptr;
    ModuleLinkingContext   *modLinkingContext = nullptr;
    ModuleTextures         *modTextures = nullptr;
    ModuleResources        *modResources = nullptr;
    ModuleGameObject       *modGameObject = nullptr;
    ModuleCollision        *modCollision = nullptr;
    ModuleScreen           *modScreen = nullptr;
    ModuleUI               *modUI = nullptr;
    ModuleRender           *modRender = nullptr;
```

Platform

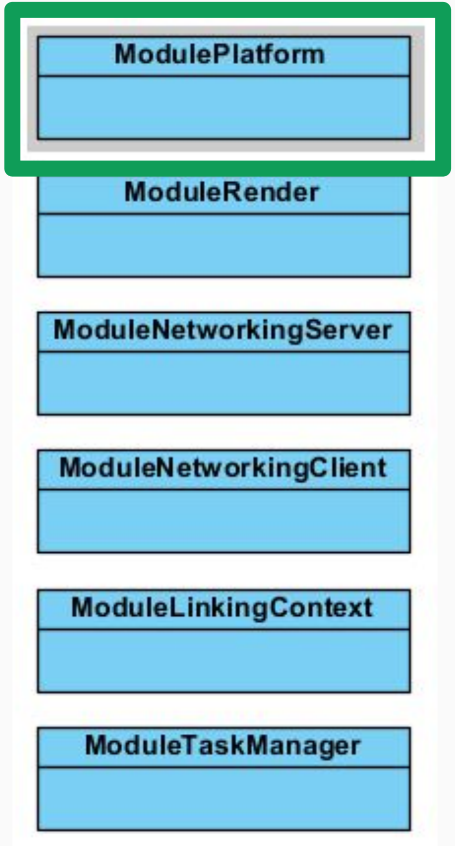


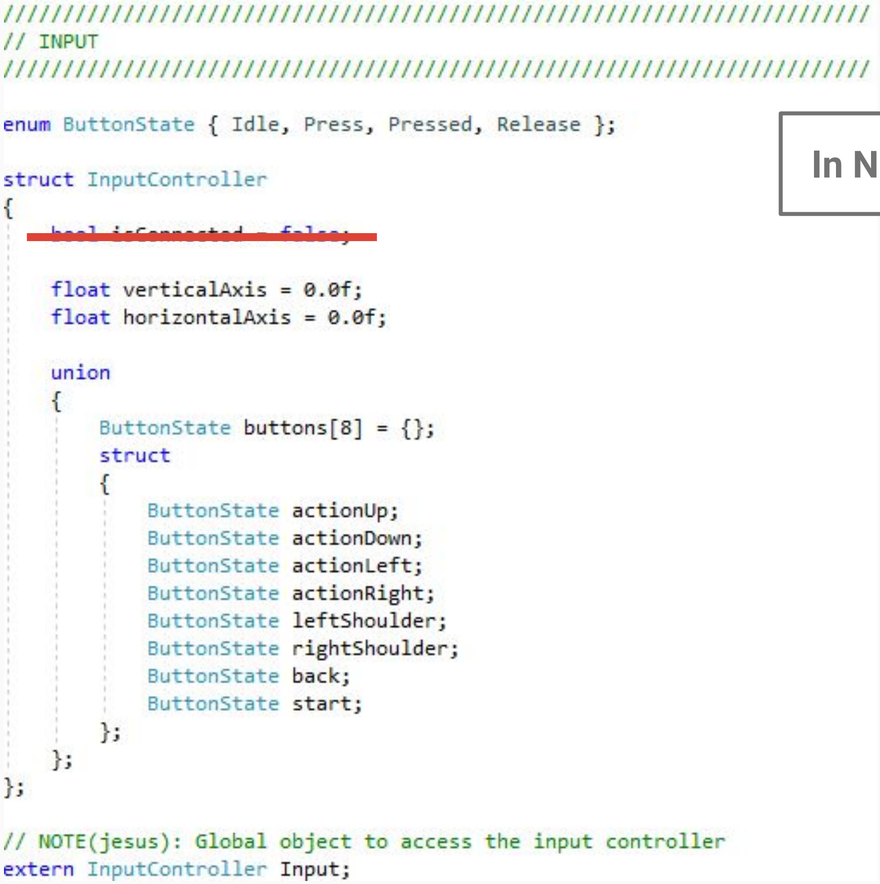
In Networks.h



```
//////////////////////////////////////////////////////////////////////
// TIME
//////////////////////////////////////////////////////////////////////

struct TimeStruct
{
    double time = 0.0f;      // NOTE(jesus): Time in seconds since the application started
    float deltaTime = 0.0f; // NOTE(jesus): Fixed update time step (use this for calculations)
    ~~float frameTime = 0.0f;~~ // NOTE(jesus): Time spend during the last frame (don't use this)
};

// NOTE(jesus): Global object to access the time
extern TimeStruct Time;
```

Platform

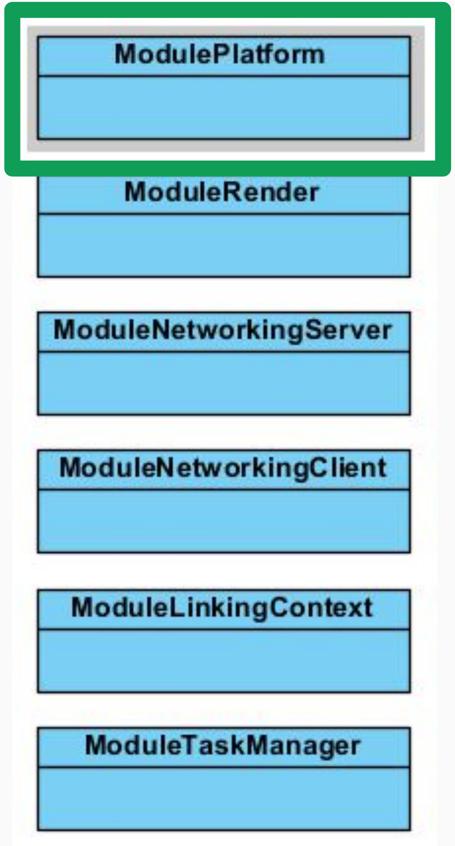




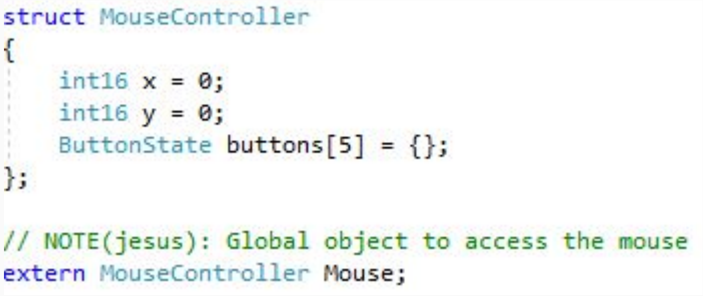
```
//////////////////////////////////////////////////////////////////////////
// INPUT
//////////////////////////////////////////////////////////////////////////

enum ButtonState { Idle, Press, Pressed, Release };

struct InputController
{
    ~~bool isConnected = false;~~

    float verticalAxis = 0.0f;
    float horizontalAxis = 0.0f;

    union
    {
        ButtonState buttons[8] = {};
        struct
        {
            ButtonState actionUp;
            ButtonState actionDown;
            ButtonState actionLeft;
            ButtonState actionRight;
            ButtonState leftShoulder;
            ButtonState rightShoulder;
            ButtonState back;
            ButtonState start;
        };
    };
};

// NOTE(jesus): Global object to access the input controller
extern InputController Input;
```

In Networks.h

Platform



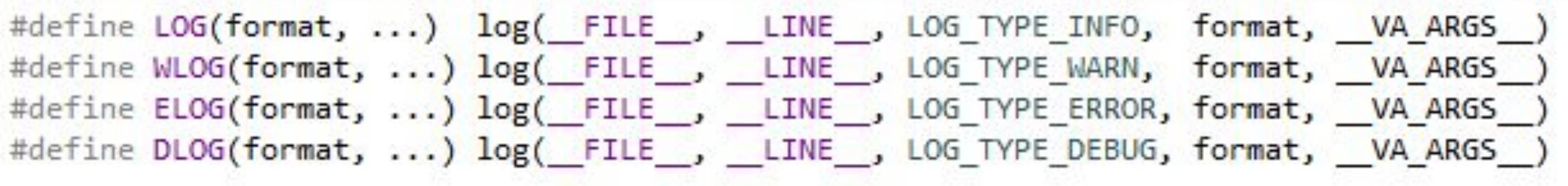
In Networks.h



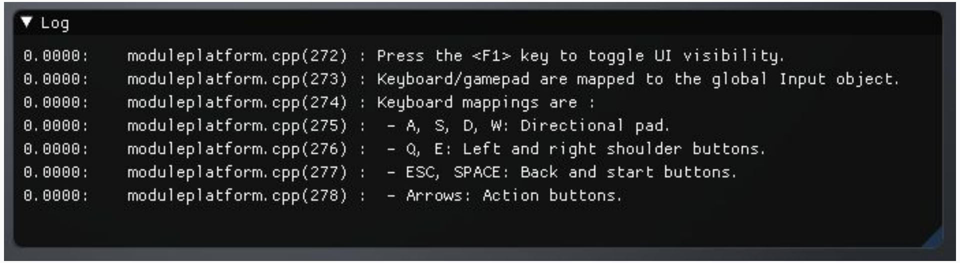
```
struct MouseController
{
	int16 x = 0;
	int16 y = 0;
	ButtonState buttons[5] = {};
};

// NOTE(jesus): Global object to access the mouse
extern MouseController Mouse;
```

Logging

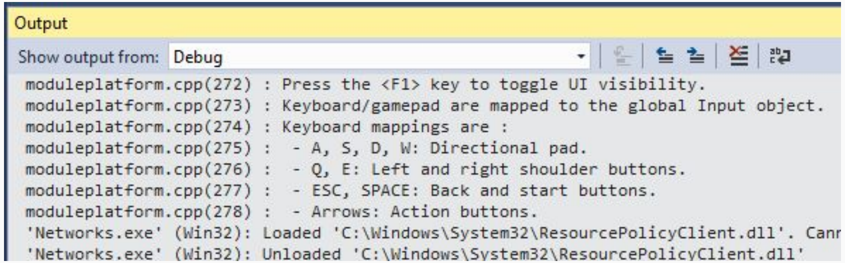
In Networks.h



```
#define LOG(format, ...)  log(__FILE__, __LINE__, LOG_TYPE_INFO,  format, __VA_ARGS__)
#define WLOG(format, ...) log(__FILE__, __LINE__, LOG_TYPE_WARN,  format, __VA_ARGS__)
#define ELOG(format, ...) log(__FILE__, __LINE__, LOG_TYPE_ERROR, format, __VA_ARGS__)
#define DLOG(format, ...) log(__FILE__, __LINE__, LOG_TYPE_DEBUG, format, __VA_ARGS__)
```



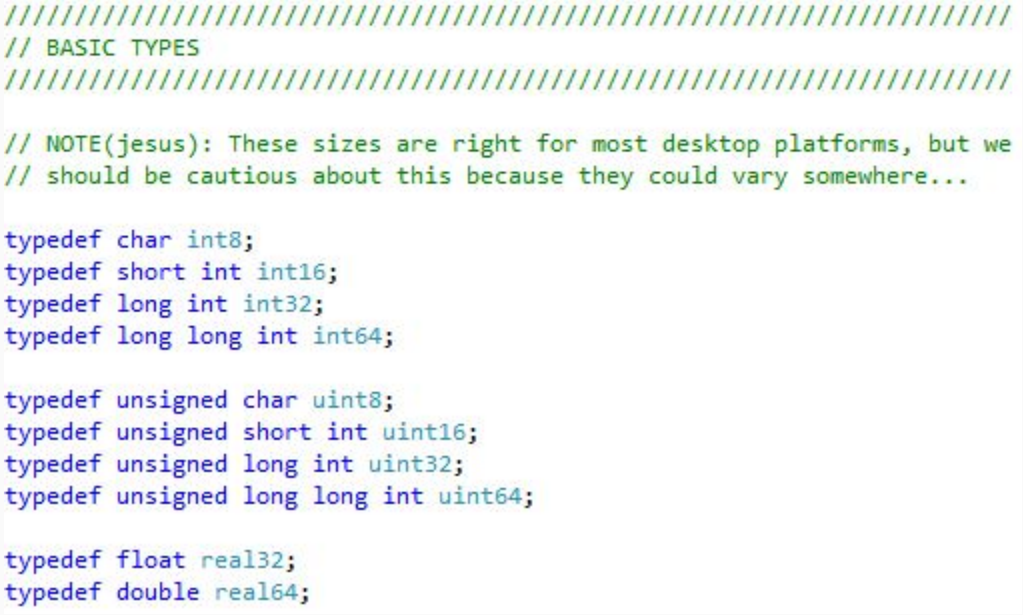
```
0.0000:    moduleplatform.cpp(272) : Press the <F1> key to toggle UI visibility.
0.0000:    moduleplatform.cpp(273) : Keyboard/gamepad are mapped to the global Input object.
0.0000:    moduleplatform.cpp(274) : Keyboard mappings are :
0.0000:    moduleplatform.cpp(275) :  - A, S, D, W: Directional pad.
0.0000:    moduleplatform.cpp(276) :  - Q, E: Left and right shoulder buttons.
0.0000:    moduleplatform.cpp(277) :  - ESC, SPACE: Back and start buttons.
0.0000:    moduleplatform.cpp(278) :  - Arrows: Action buttons.
```



Show output from: Debug

```
moduleplatform.cpp(272) : Press the <F1> key to toggle UI visibility.
moduleplatform.cpp(273) : Keyboard/gamepad are mapped to the global Input object.
moduleplatform.cpp(274) : Keyboard mappings are :
moduleplatform.cpp(275) :  - A, S, D, W: Directional pad.
moduleplatform.cpp(276) :  - Q, E: Left and right shoulder buttons.
moduleplatform.cpp(277) :  - ESC, SPACE: Back and start buttons.
moduleplatform.cpp(278) :  - Arrows: Action buttons.
'Networks.exe' (Win32): Loaded 'C:\Windows\System32\ResourcePolicyClient.dll'. Canr
'Networks.exe' (Win32): Unloaded 'C:\Windows\System32\ResourcePolicyClient.dll'
```

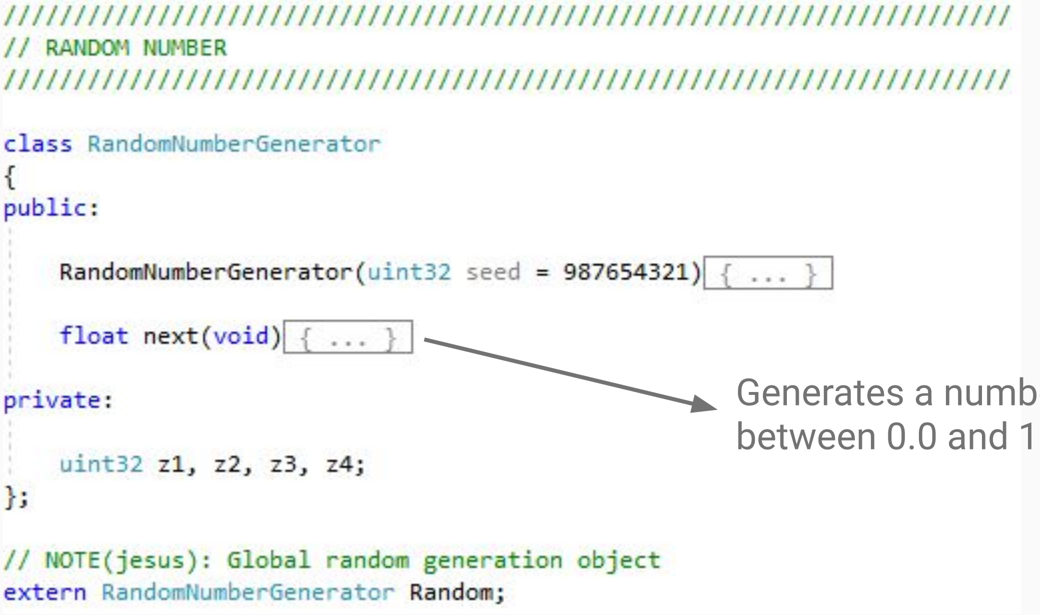
Basic data types



```
////////////////////////////////////////////////////////////////////////
// BASIC TYPES
////////////////////////////////////////////////////////////////////////

// NOTE(jesus): These sizes are right for most desktop platforms, but we
// should be cautious about this because they could vary somewhere...

typedef char int8;
typedef short int int16;
typedef long int int32;
typedef long long int int64;

typedef unsigned char uint8;
typedef unsigned short int uint16;
typedef unsigned long int uint32;
typedef unsigned long long int uint64;

typedef float real32;
typedef double real64;
```

In Networks.h

Random numbers

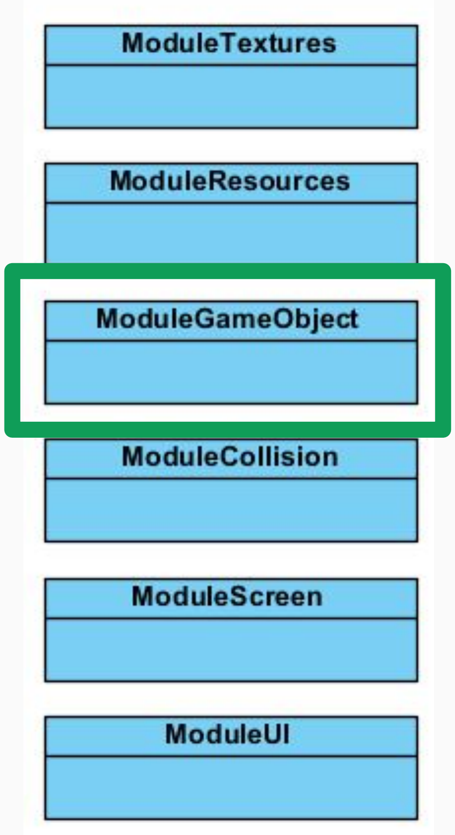


```
//////////////////////////////////////////////////////////////////////////////////
// RANDOM NUMBER
//////////////////////////////////////////////////////////////////////////////////

class RandomNumberGenerator
{
public:

    RandomNumberGenerator(uint32 seed = 987654321) { ... }

    float next(void) { ... }

private:

    uint32 z1, z2, z3, z4;
};

// NOTE(jesus): Global random generation object
extern RandomNumberGenerator Random;
```

In Networks.h

Generates a number between 0.0 and 1.0

ModuleGameObject



ModuleTextures

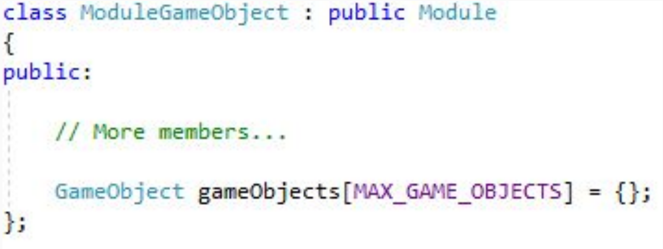
ModuleResources

ModuleGameObject

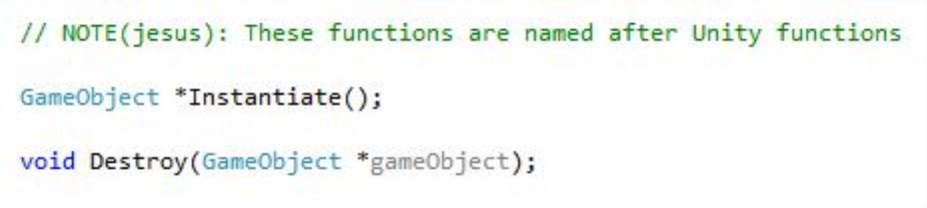
ModuleCollision

ModuleScreen

ModuleUI

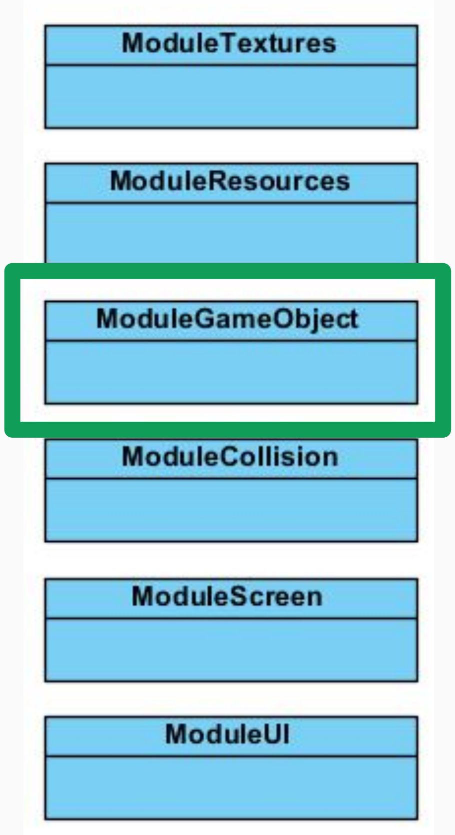


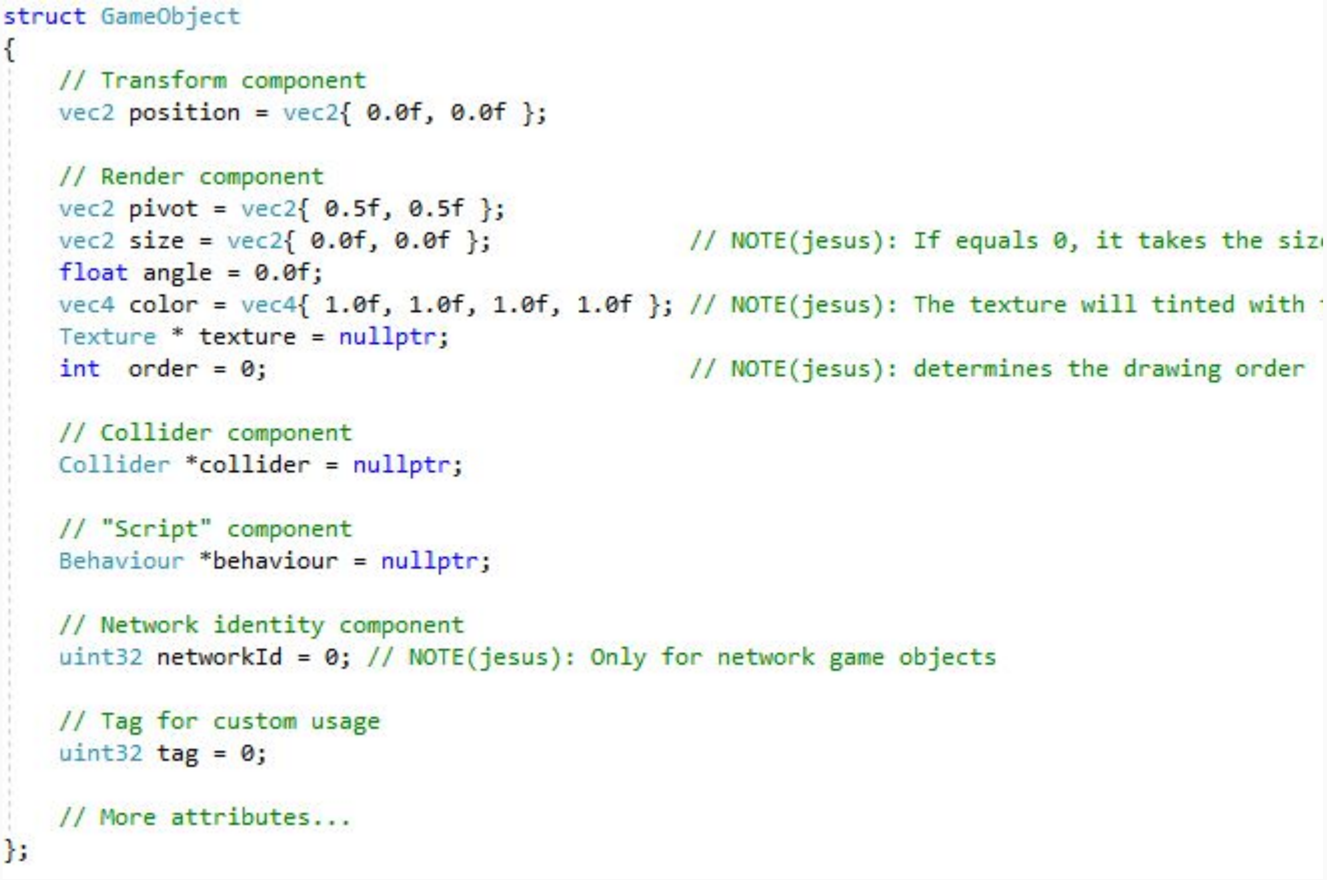
```
class ModuleGameObject : public Module
{
public:

    // More members...

    GameObject gameObjects[MAX_GAME_OBJECTS] = {};
};
```



```
// NOTE(jesus): These functions are named after Unity functions

GameObject *Instantiate();

void Destroy(GameObject *gameObject);
```

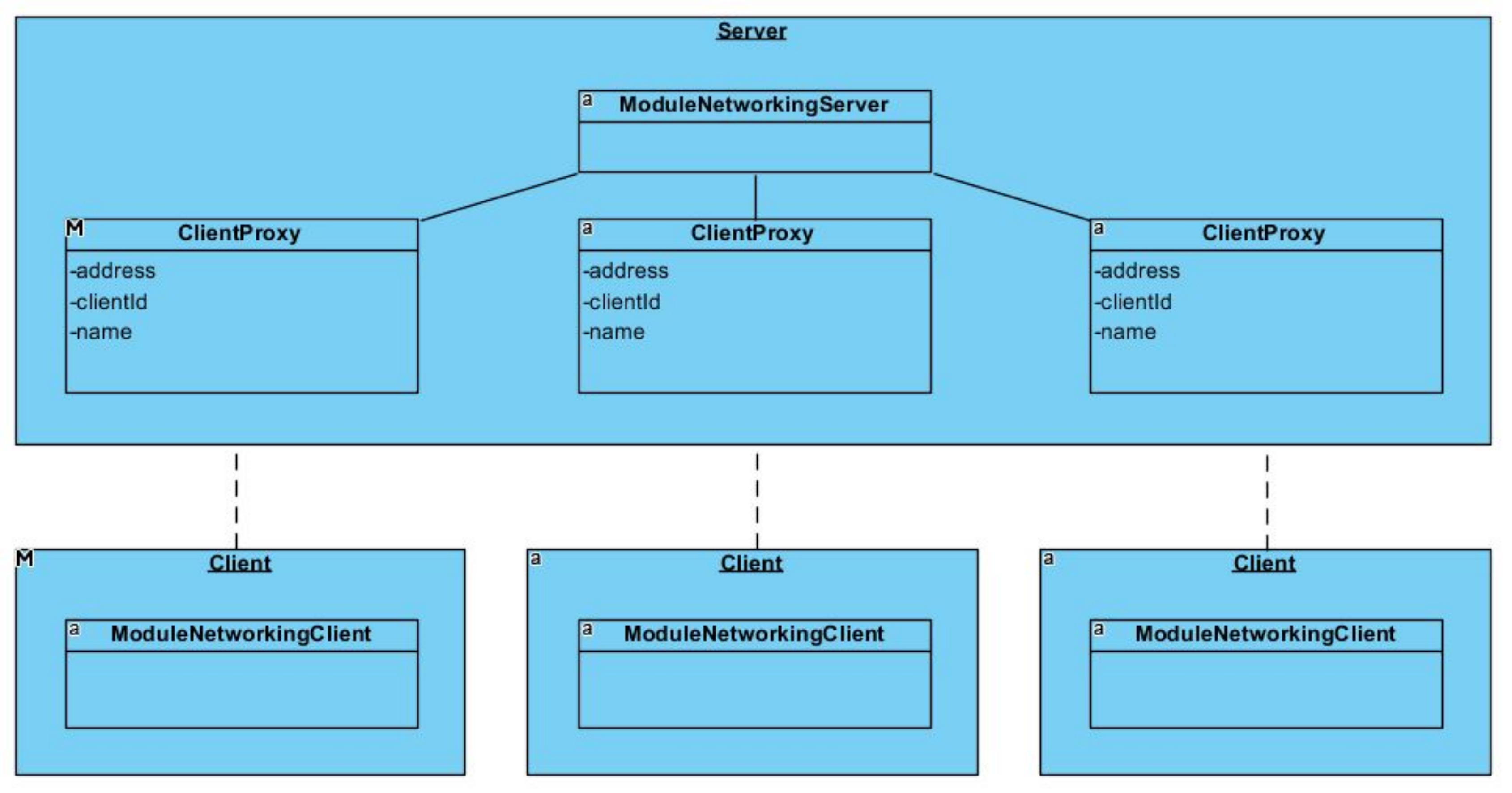
ModuleGameObject



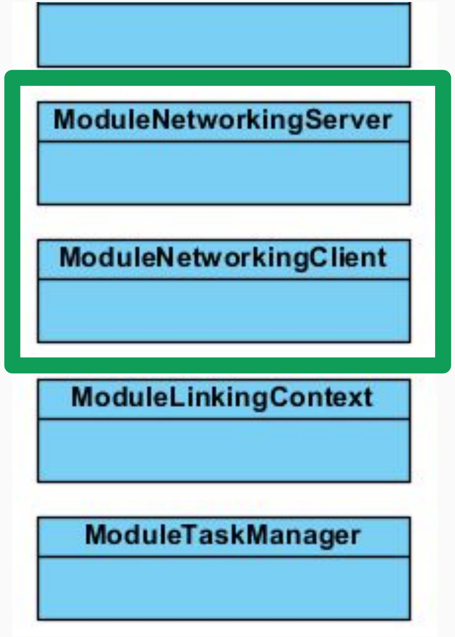


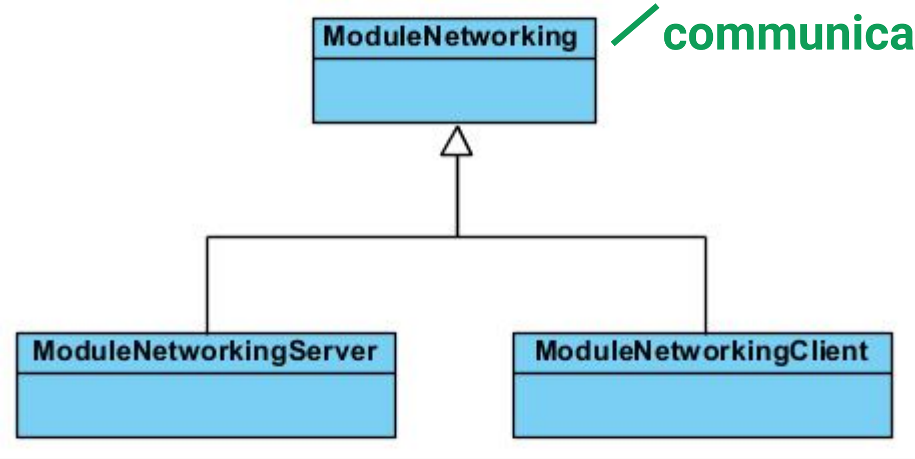
```
struct GameObject
{
    // Transform component
    vec2 position = vec2{ 0.0f, 0.0f };

    // Render component
    vec2 pivot = vec2{ 0.5f, 0.5f };
    vec2 size = vec2{ 0.0f, 0.0f };                 // NOTE(jesus): If equals 0, it takes the siz
    float angle = 0.0f;
    vec4 color = vec4{ 1.0f, 1.0f, 1.0f, 1.0f }; // NOTE(jesus): The texture will tinted with
    Texture * texture = nullptr;
    int  order = 0;                                 // NOTE(jesus): determines the drawing order

    // Collider component
    Collider *collider = nullptr;

    // "Script" component
    Behaviour *behaviour = nullptr;

    // Network identity component
    uint32 networkId = 0; // NOTE(jesus): Only for network game objects

    // Tag for custom usage
    uint32 tag = 0;

    // More attributes...
};
```

Networking modules



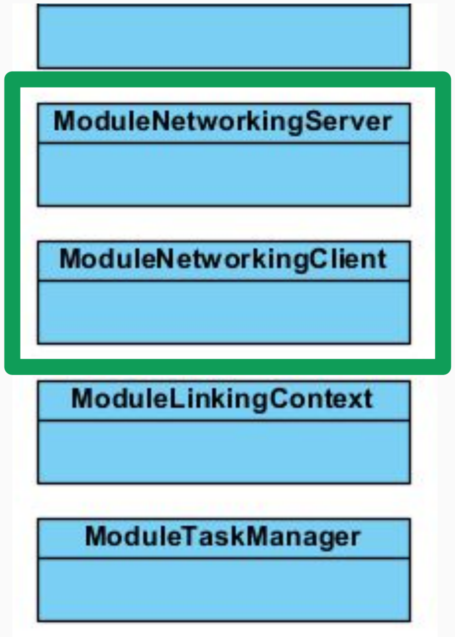
Networking modules

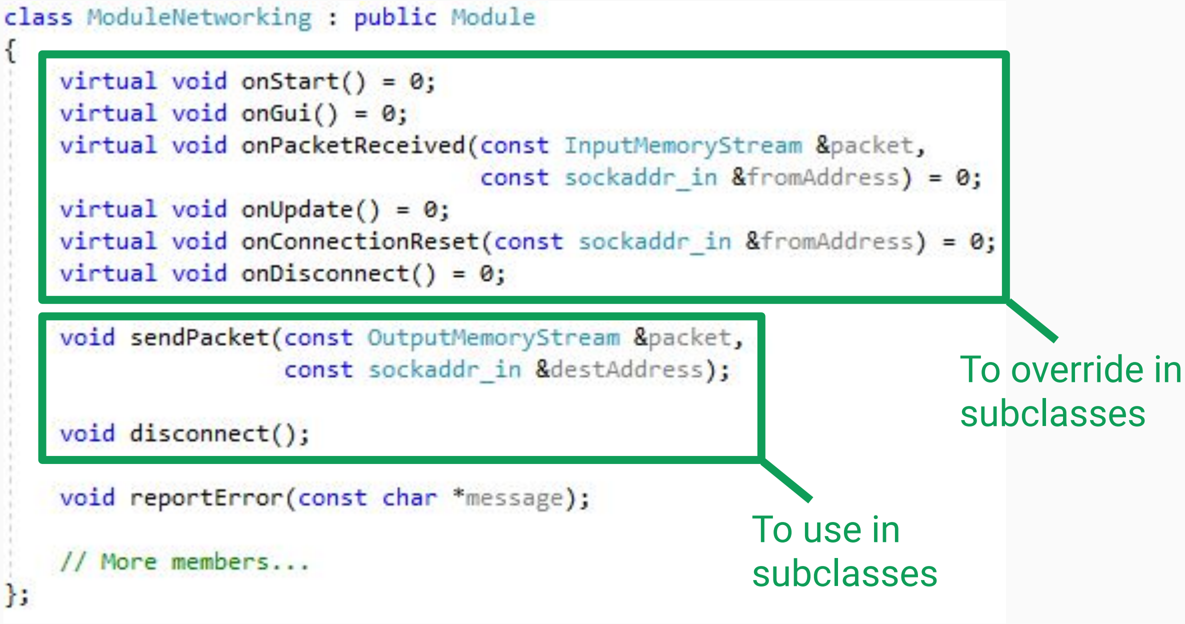




Basics for UDP socket communication

ModuleNetworking (base class)



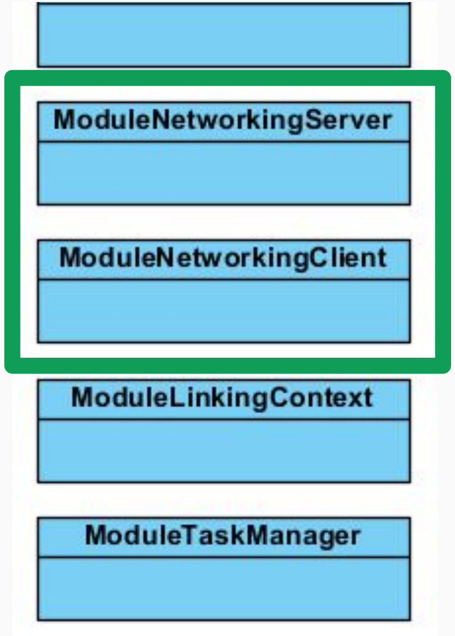


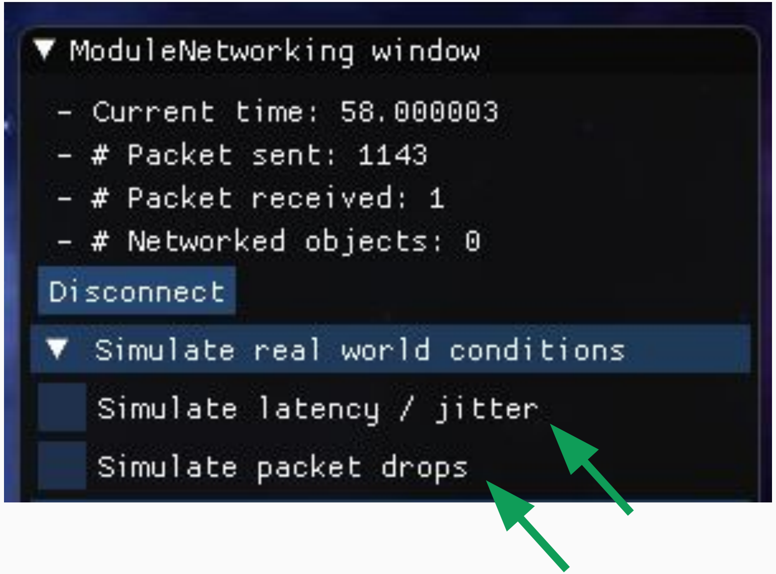
```cpp
class ModuleNetworking : public Module
{
    virtual void onStart() = 0;
    virtual void onGui() = 0;
    virtual void onPacketReceived(const InputMemoryStream &packet,
                                  const sockaddr_in &fromAddress) = 0;
    virtual void onUpdate() = 0;
    virtual void onConnectionReset(const sockaddr_in &fromAddress) = 0;
    virtual void onDisconnect() = 0;

    void sendPacket(const OutputMemoryStream &packet,
                    const sockaddr_in &destAddress);

    void disconnect();

    void reportError(const char *message);

    // More members...
};
```

To override in subclasses

To use in subclasses

ModuleNetworking (base class)



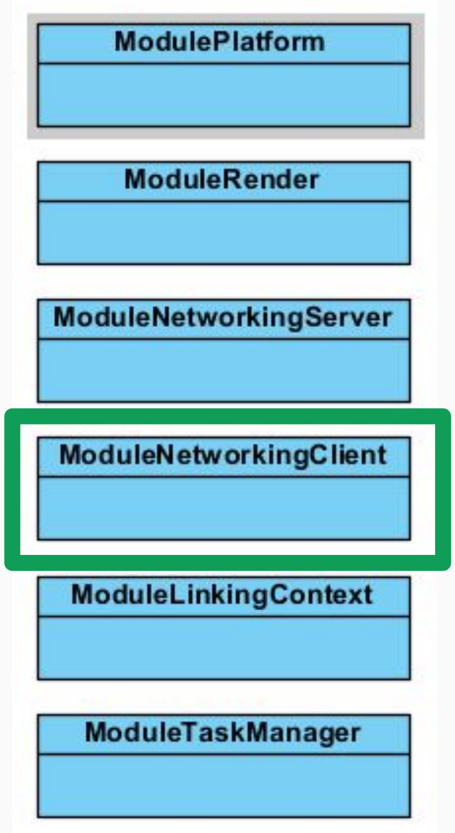


```
- Current time: 58.000003
- # Packet sent: 1143
- # Packet received: 1
- # Networked objects: 0
```

□ Simulate latency / jitter

□ Simulate packet drops

ModuleNetworkingClient

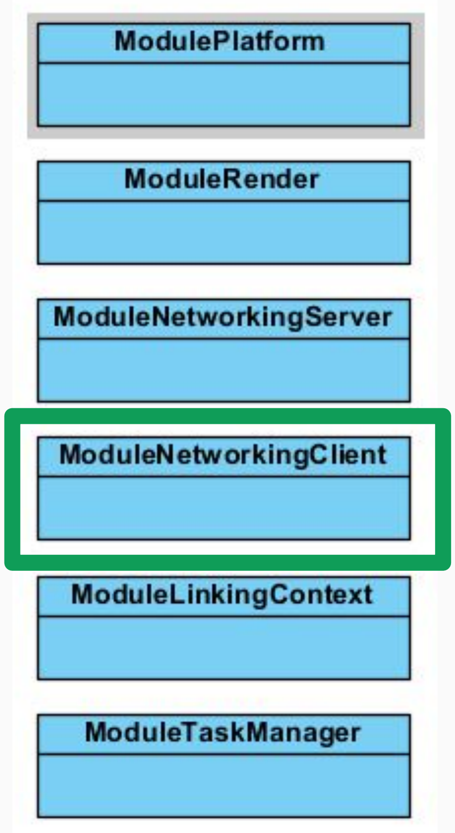


ModuleNetworkingClient

What it does right now:

- Connects to server
- Sends a “Hello” packet
- Waits receiving a “Welcome” packet
- Accumulates and periodically sends input data
  - Queue of gamepad input states

ModuleNetworkingClient



ModulePlatform

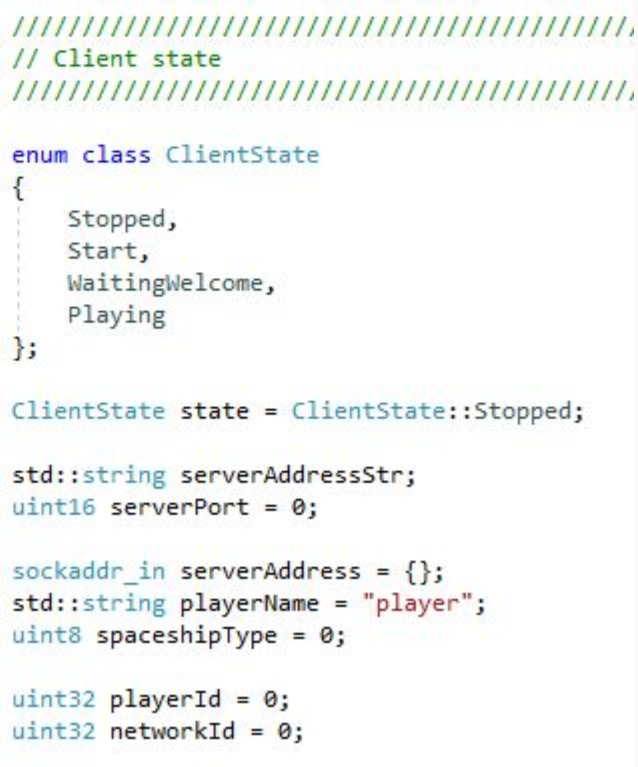
ModuleRender

ModuleNetworkingServer

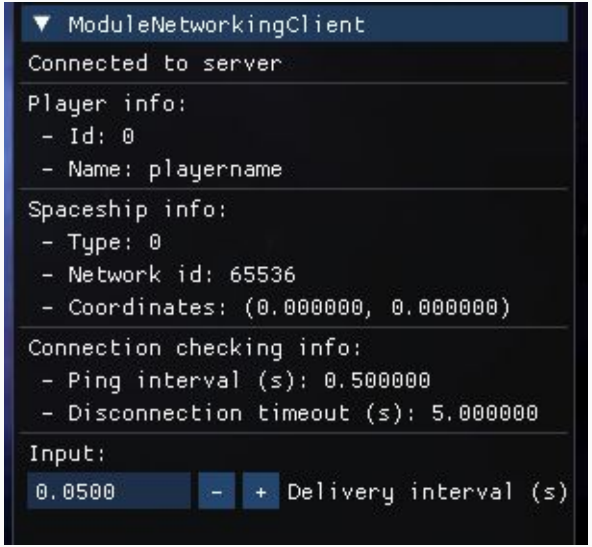
ModuleNetworkingClient

ModuleLinkingContext

ModuleTaskManager

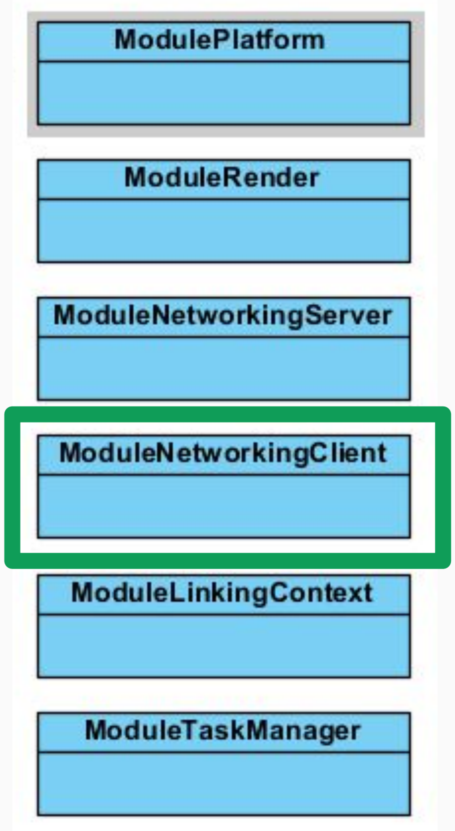


```
////////////////////////////////////////////
// Client state
////////////////////////////////////////////

enum class ClientState
{
    Stopped,
    Start,
    WaitingWelcome,
    Playing
};

ClientState state = ClientState::Stopped;

std::string serverAddressStr;
uint16 serverPort = 0;

sockaddr_in serverAddress = {};
std::string playerName = "player";
uint8 spaceshipType = 0;

uint32 playerId = 0;
uint32 networkId = 0;
```



```
▼ ModuleNetworkingClient
Connected to server
Player info:
 - Id: 0
 - Name: playername
Spaceship info:
 - Type: 0
 - Network id: 65536
 - Coordinates: (0.000000, 0.000000)
Connection checking info:
 - Ping interval (s): 0.500000
 - Disconnection timeout (s): 5.000000
Input:
0.0500  [ - ] [ + ] Delivery interval (s)
```

ModuleNetworkingClient



ModulePlatform

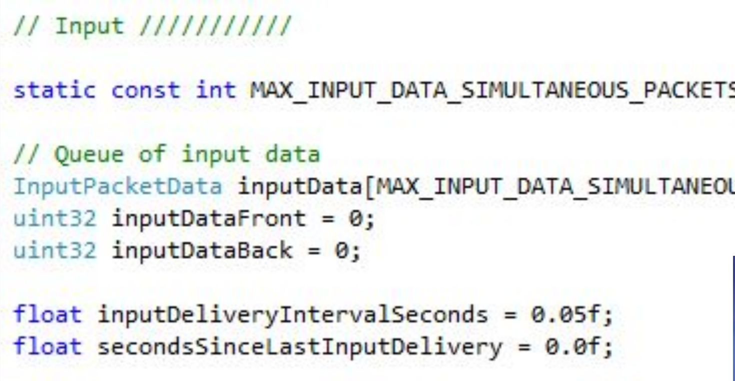
ModuleRender

ModuleNetworkingServer

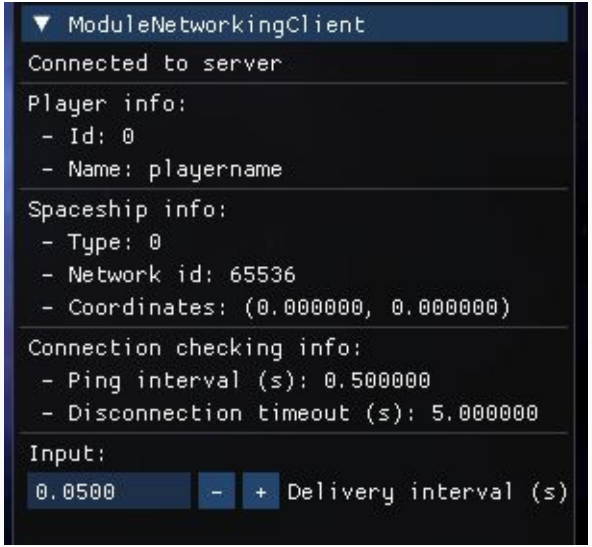
ModuleNetworkingClient

ModuleLinkingContext

ModuleTaskManager



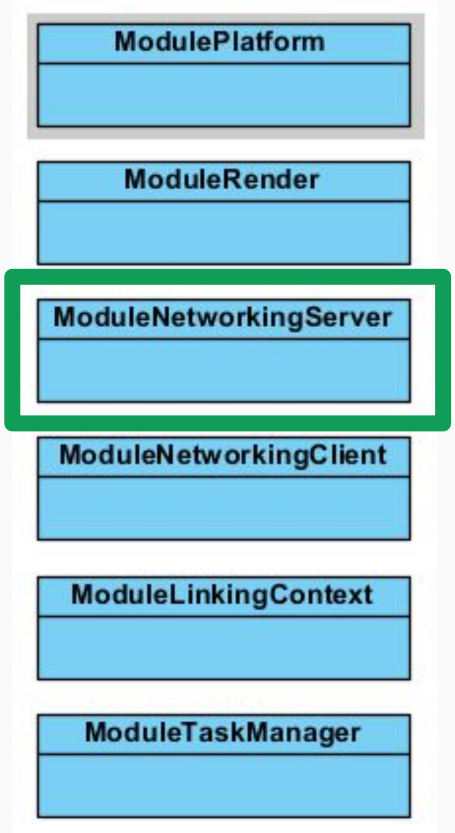
```
// Input ///////////

static const int MAX_INPUT_DATA_SIMULTANEOUS_PACKETS = 64;

// Queue of input data
InputPacketData inputData[MAX_INPUT_DATA_SIMULTANEOUS_PACKETS];
uint32 inputDataFront = 0;
uint32 inputDataBack = 0;

float inputDeliveryIntervalSeconds = 0.05f;
float secondsSinceLastInputDelivery = 0.0f;
```



▼ ModuleNetworkingClient

```
Connected to server
Player info:
 - Id: 0
 - Name: playername
Spaceship info:
 - Type: 0
 - Network id: 65536
 - Coordinates: (0.000000, 0.000000)
Connection checking info:
 - Ping interval (s): 0.500000
 - Disconnection timeout (s): 5.000000
Input:
0.0500   - + Delivery interval (s)
```

ModuleNetworkingServer

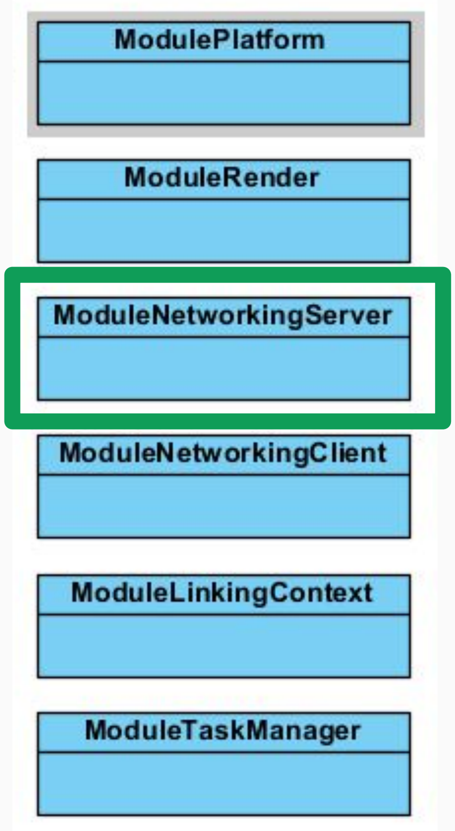


ModuleNetworkingServer

What it does right now:

- Creates a listen socket
- Accepts new client connections
  - Receives “Hello” packets
  - Creates player data
  - Sends “Welcome” packets
- Handles input packets from clients
  - Receives input packets
  - Updates players with received inputs

ModuleNetworkingServer



ModulePlatform

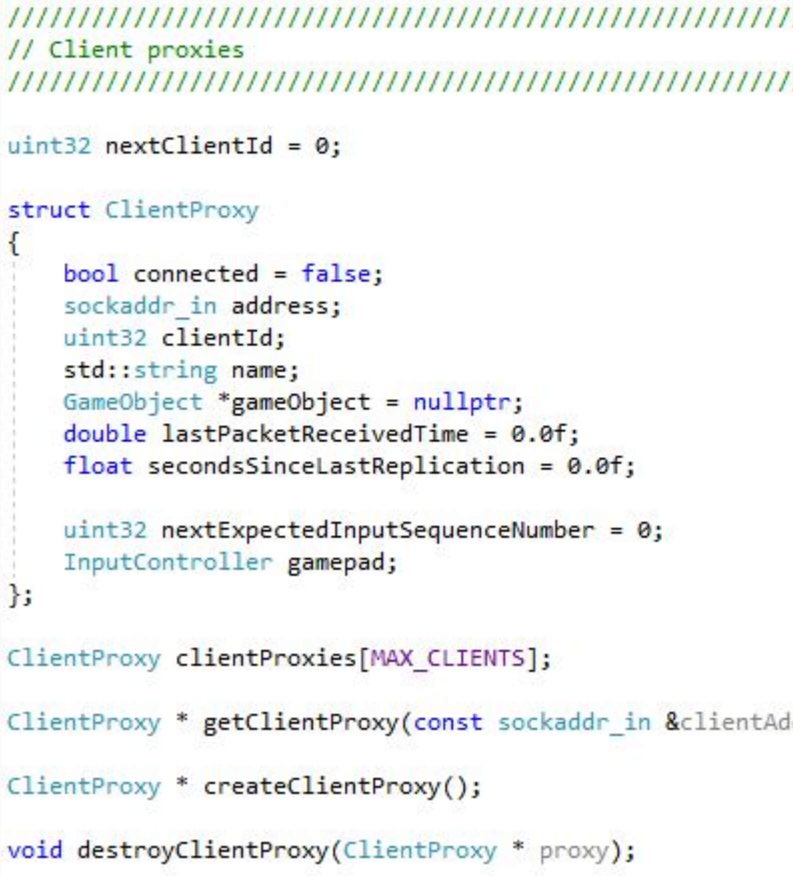
ModuleRender

ModuleNetworkingServer

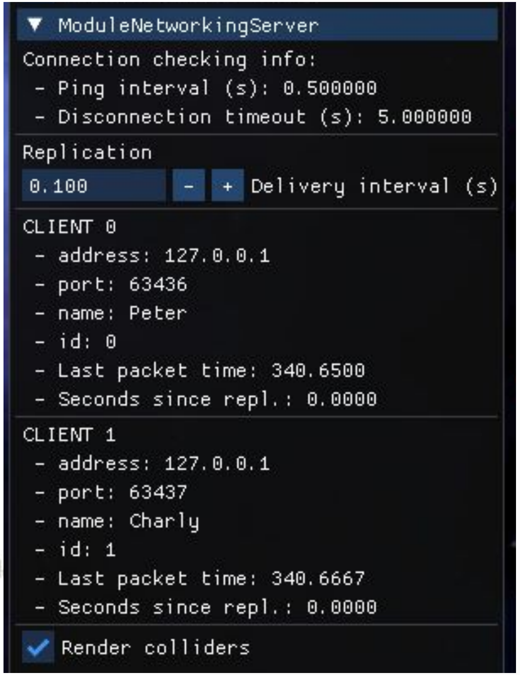
ModuleNetworkingClient

ModuleLinkingContext

ModuleTaskManager

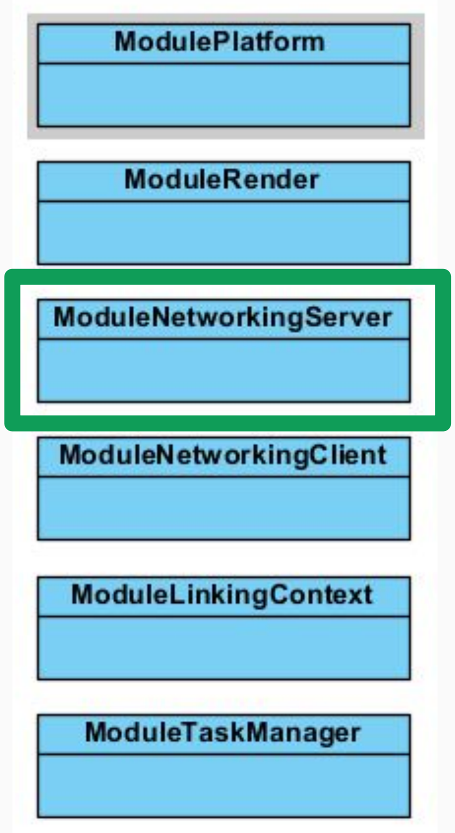


```
////////////////////////////////////////////////////////////////////////
// Client proxies
////////////////////////////////////////////////////////////////////////

uint32 nextClientId = 0;

struct ClientProxy
{
    bool connected = false;
    sockaddr_in address;
    uint32 clientId;
    std::string name;
    GameObject *gameObject = nullptr;
    double lastPacketReceivedTime = 0.0f;
    float secondsSinceLastReplication = 0.0f;

    uint32 nextExpectedInputSequenceNumber = 0;
    InputController gamepad;
};

ClientProxy clientProxies[MAX_CLIENTS];

ClientProxy * getClientProxy(const sockaddr_in &clientAd

ClientProxy * createClientProxy();

void destroyClientProxy(ClientProxy * proxy);
```



```
▼ ModuleNetworkingServer
Connection checking info:
 - Ping interval (s): 0.500000
 - Disconnection timeout (s): 5.000000
Replication
0.100  - + Delivery interval (s)
CLIENT 0
 - address: 127.0.0.1
 - port: 63436
 - name: Peter
 - id: 0
 - Last packet time: 340.6500
 - Seconds since repl.: 0.0000
CLIENT 1
 - address: 127.0.0.1
 - port: 63437
 - name: Charly
 - id: 1
 - Last packet time: 340.6667
 - Seconds since repl.: 0.0000
☑ Render colliders
```

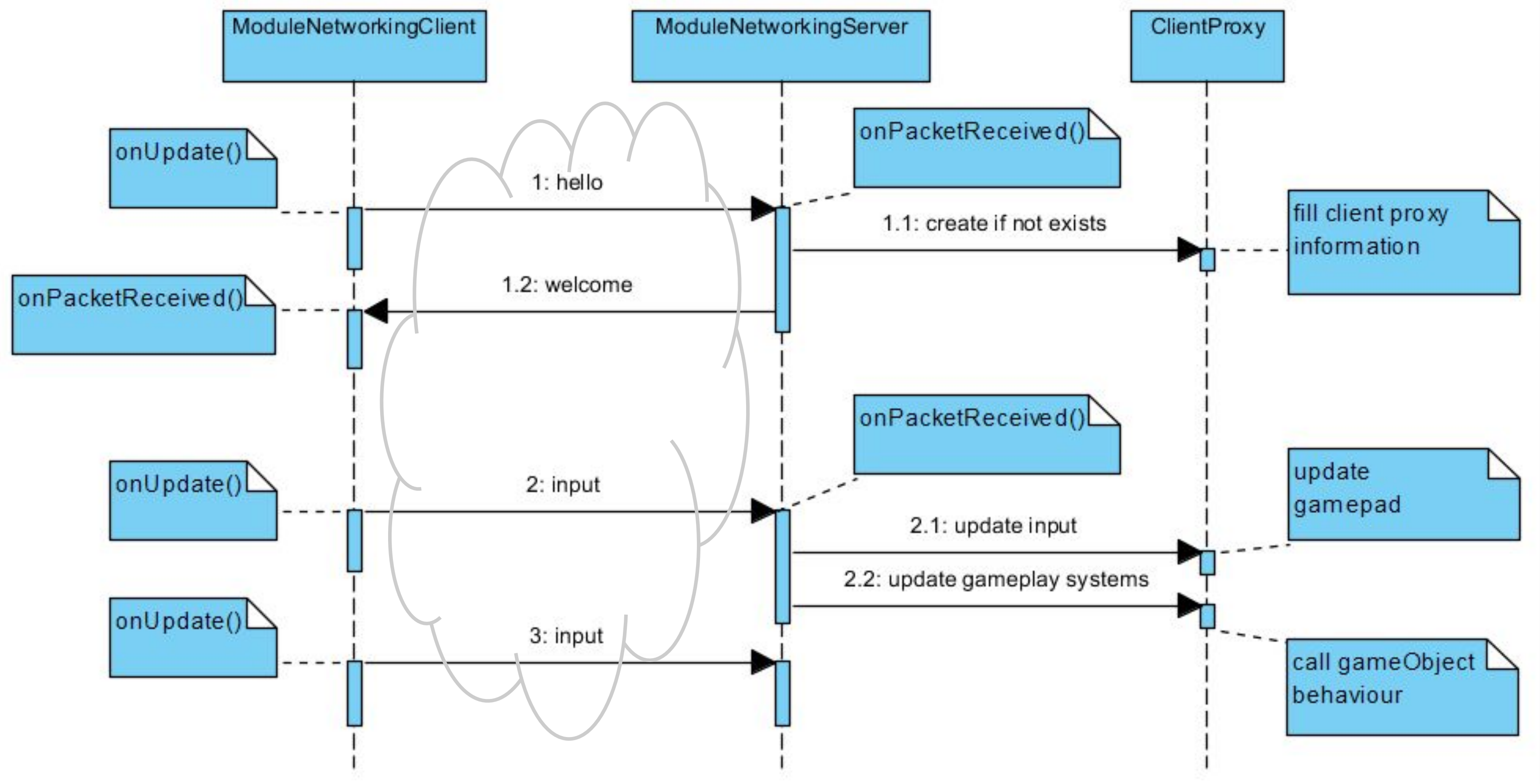
ModuleNetworkingServer



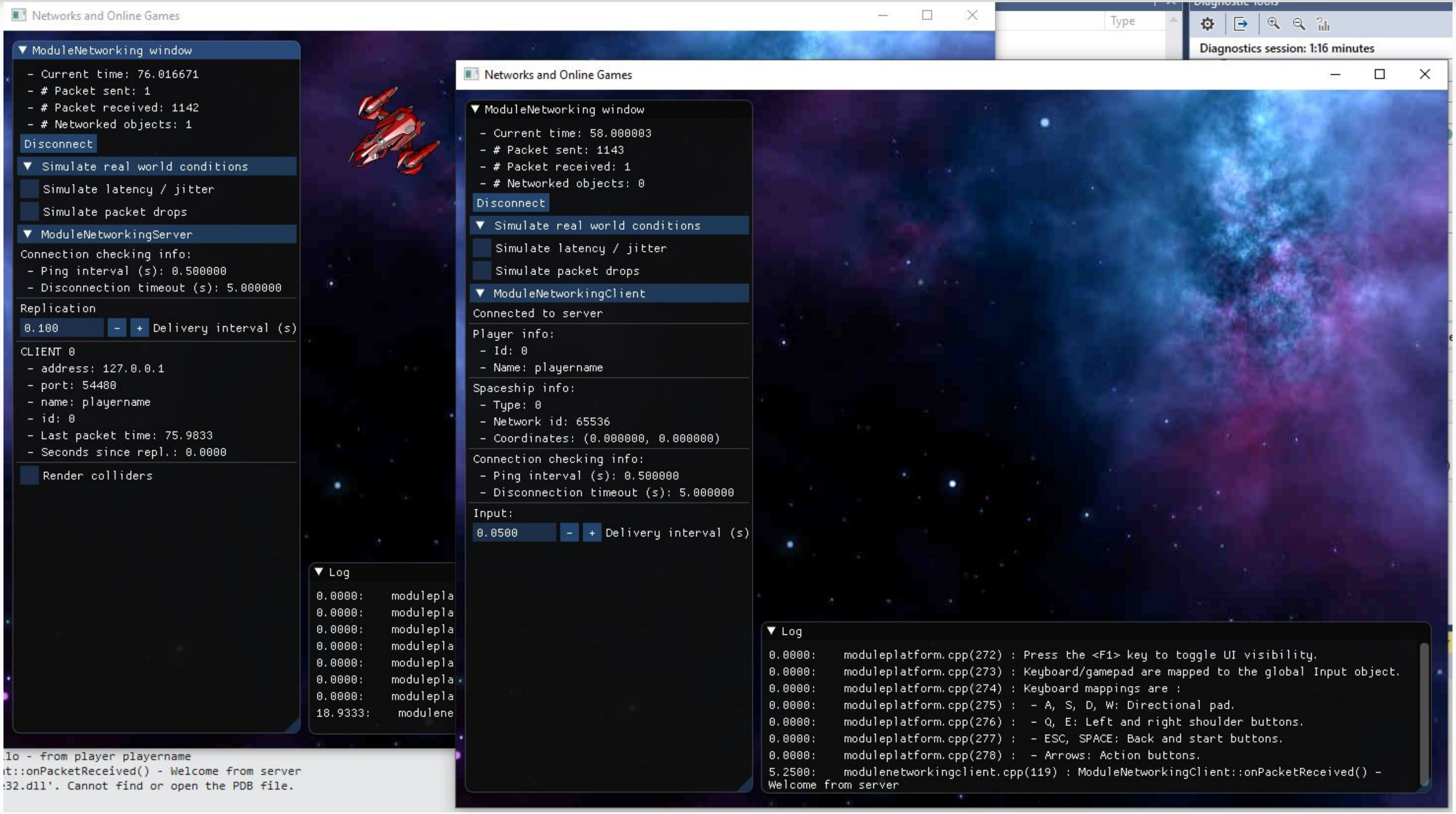
Responsible for spawning network game objects

- Specialized methods in ModuleNetworkingServer
- Have a look at
  - ModuleNetworkingServer::spawnPlayer()
  - ModuleNetworkingServer::spawnBullet()

Networking overview



Networking overview



Networks and Online Games
Type
Diagnostic Tools
Diagnostics session: 1:16 minutes
Networks and Online Games

```
▼ ModuleNetworking window
  - Current time: 76.016671
  - # Packet sent: 1
  - # Packet received: 1142
  - # Networked objects: 1
 Disconnect
 ▼ Simulate real world conditions
   Simulate latency / jitter
   Simulate packet drops
 ▼ ModuleNetworkingServer
 Connection checking info:
  - Ping interval (s): 0.500000
  - Disconnection timeout (s): 5.000000
 Replication
 0.100        - + Delivery interval (s)
 CLIENT 0
  - address: 127.0.0.1
  - port: 54480
  - name: playername
  - id: 0
  - Last packet time: 75.9833
  - Seconds since repl.: 0.0000
   Render colliders
```

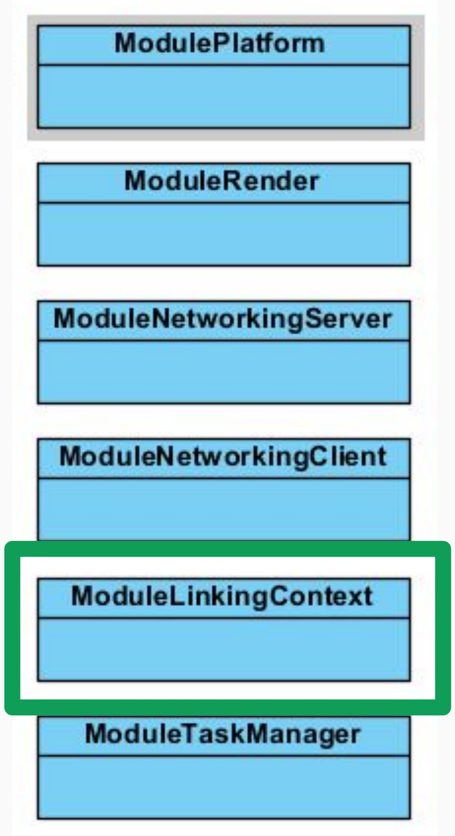
```
▼ Log
0.0000:    modulepla
0.0000:    modulepla
0.0000:    modulepla
0.0000:    modulepla
0.0000:    modulepla
0.0000:    modulepla
0.0000:    modulepla
18.9333:    modulene
```

```
lo - from player playername
t::onPacketReceived() - Welcome from server
32.dll'. Cannot find or open the PDB file.
```

```
▼ ModuleNetworking window
  - Current time: 58.000003
  - # Packet sent: 1143
  - # Packet received: 1
  - # Networked objects: 0
 Disconnect
 ▼ Simulate real world conditions
   Simulate latency / jitter
   Simulate packet drops
 ▼ ModuleNetworkingClient
 Connected to server
 Player info:
  - Id: 0
  - Name: playername
 Spaceship info:
  - Type: 0
  - Network id: 65536
  - Coordinates: (0.000000, 0.000000)
 Connection checking info:
  - Ping interval (s): 0.500000
  - Disconnection timeout (s): 5.000000
 Input:
 0.0500       - + Delivery interval (s)
```

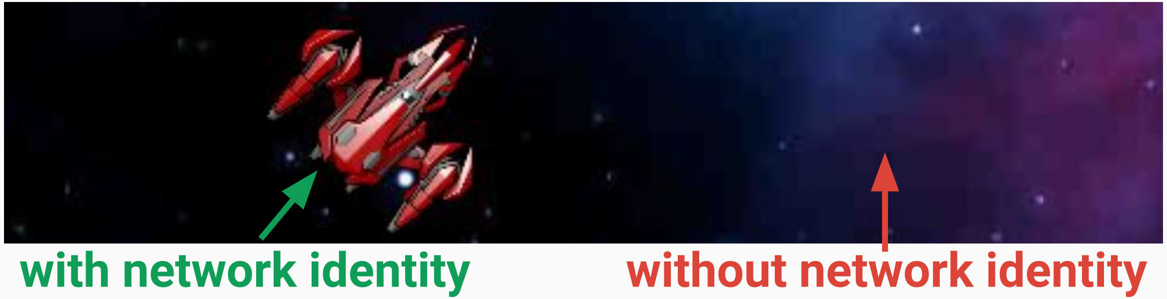
```
▼ Log
0.0000:    moduleplatform.cpp(272) : Press the <F1> key to toggle UI visibility.
0.0000:    moduleplatform.cpp(273) : Keyboard/gamepad are mapped to the global Input object.
0.0000:    moduleplatform.cpp(274) : Keyboard mappings are :
0.0000:    moduleplatform.cpp(275) :  - A, S, D, W: Directional pad.
0.0000:    moduleplatform.cpp(276) :  - Q, E: Left and right shoulder buttons.
0.0000:    moduleplatform.cpp(277) :  - ESC, SPACE: Back and start buttons.
0.0000:    moduleplatform.cpp(278) :  - Arrows: Action buttons.
5.2500:    modulenetworkingclient.cpp(119) : ModuleNetworkingClient::onPacketReceived() -
Welcome from server
```

ModuleLinkingContext

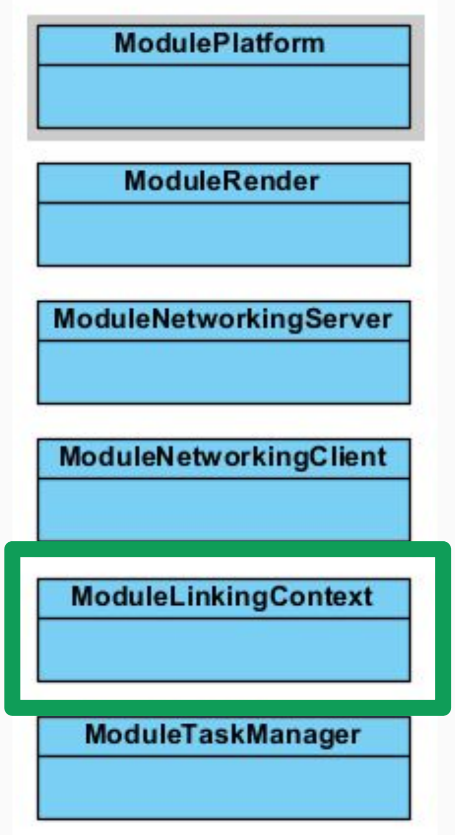


Network identity

- Identifies an object over the network
  - Object changes in server -> same object in clients must change
- Examples of game objects **with network identity**
  - Player character
  - Dynamic interactable objects
- Examples of game objects **without network identity**
  - Static environment objects
  - UI elements



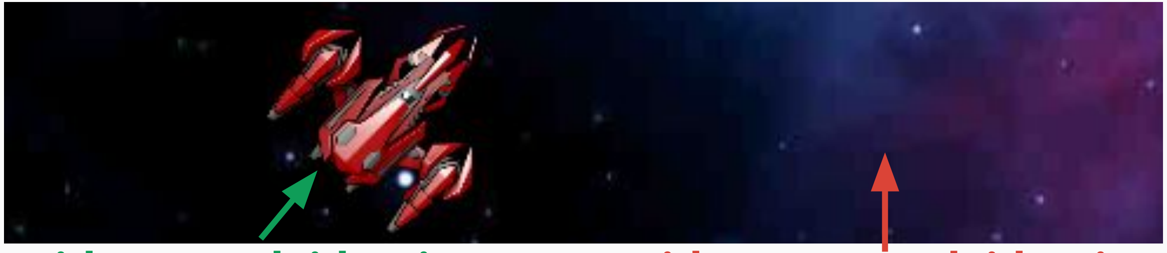
ModuleLinkingContext



The linking context:

Registry of game objects with a network identity

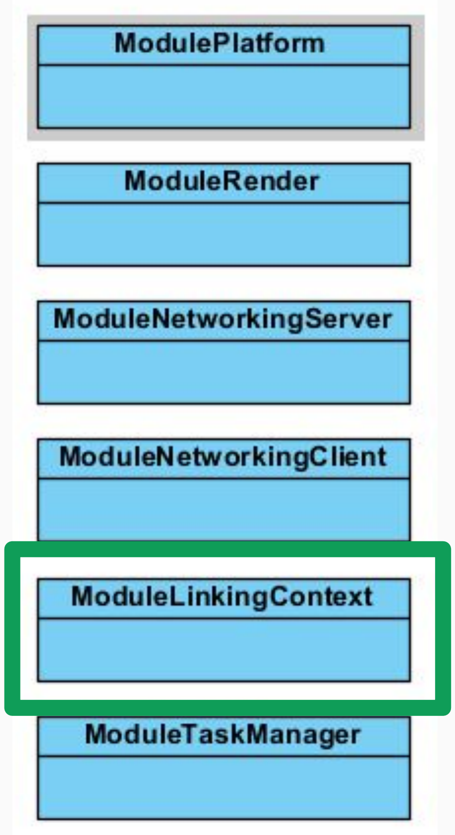
- Register **creating a new** network identity **(server)**
- Register **using a specific** network identity **(client)**
- Find an object by its network identity
- Find all network game objects
- Unregister



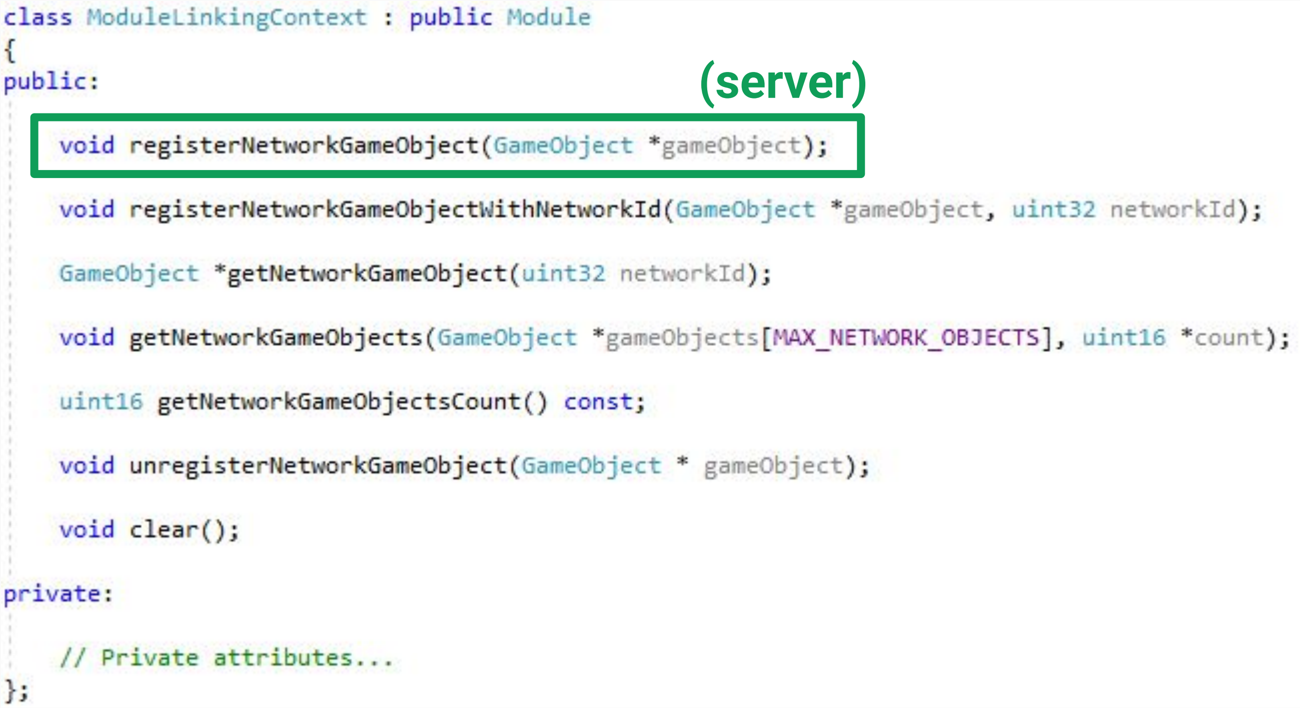
with network identity

without network identity

ModuleLinkingContext



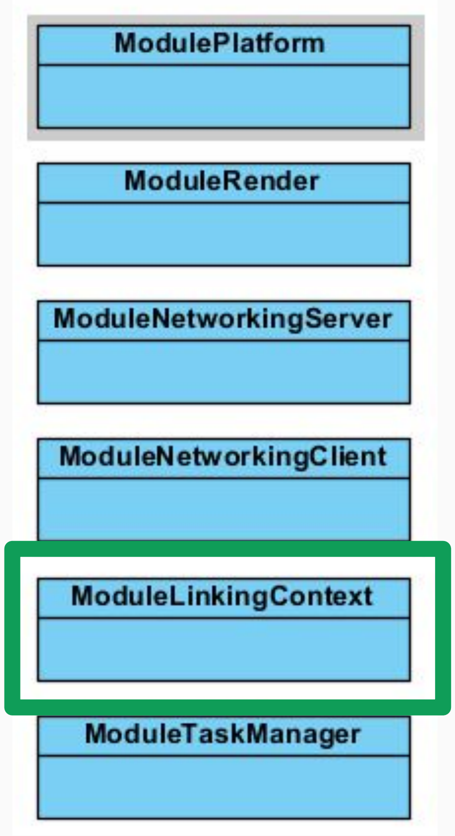
The server will register new network game objects



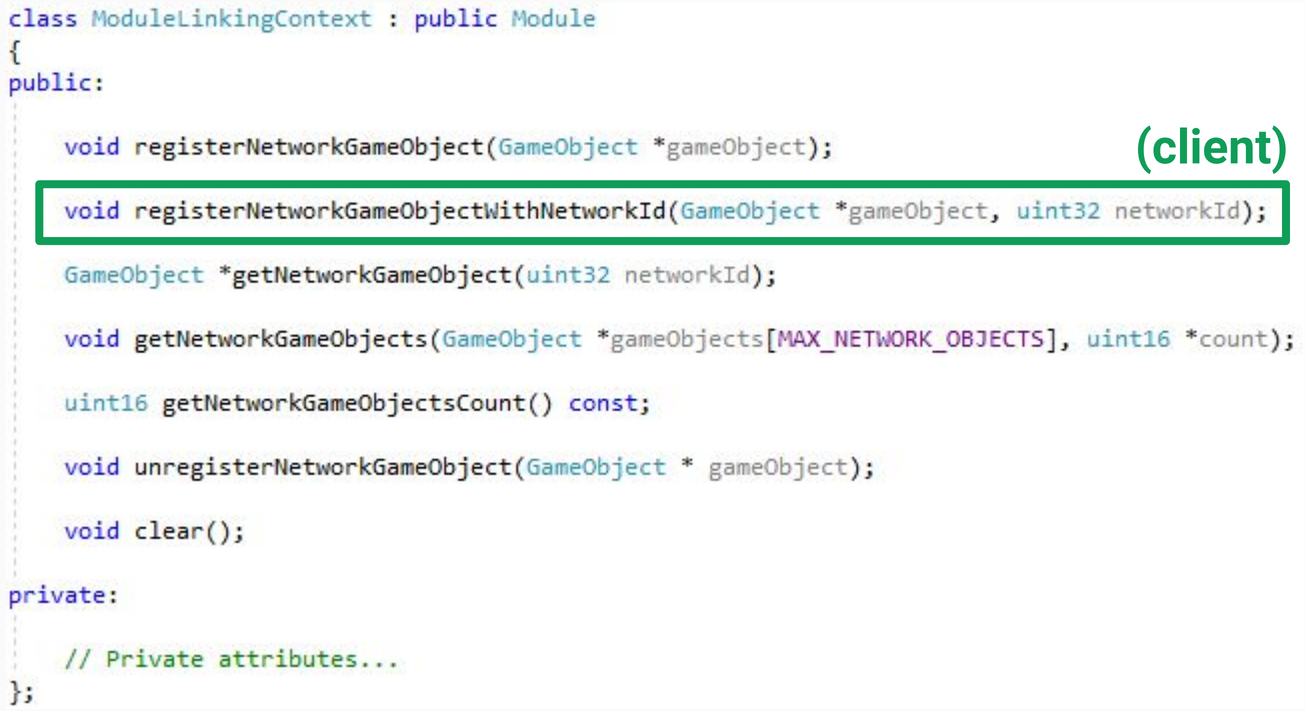
(server)

```
class ModuleLinkingContext : public Module
{
public:

    void registerNetworkGameObject(GameObject *gameObject);

    void registerNetworkGameObjectWithNetworkId(GameObject *gameObject, uint32 networkId);

    GameObject *getNetworkGameObject(uint32 networkId);

    void getNetworkGameObjects(GameObject *gameObjects[MAX_NETWORK_OBJECTS], uint16 *count);

    uint16 getNetworkGameObjectsCount() const;

    void unregisterNetworkGameObject(GameObject * gameObject);

    void clear();

private:

    // Private attributes...
};
```

ModuleLinkingContext



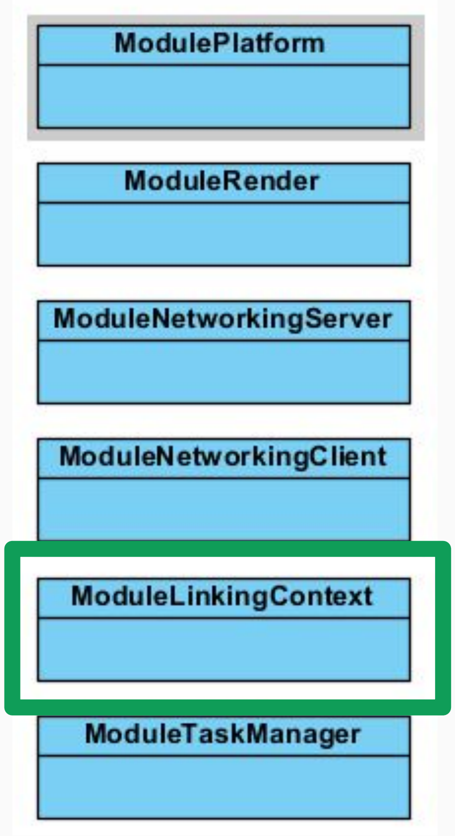
Client will register network game objects with an existing network identity (informed by the server)



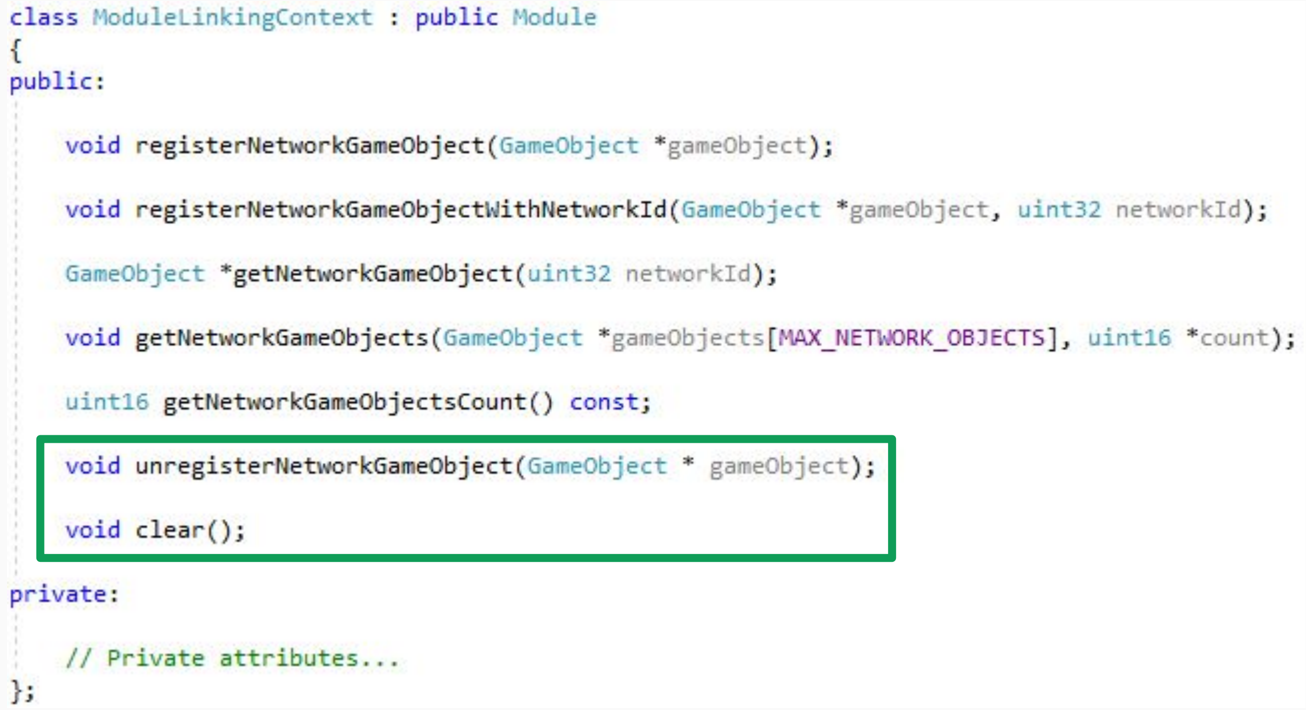
(client)

```
class ModuleLinkingContext : public Module
{
public:

    void registerNetworkGameObject(GameObject *gameObject);

    void registerNetworkGameObjectWithNetworkId(GameObject *gameObject, uint32 networkId);

    GameObject *getNetworkGameObject(uint32 networkId);

    void getNetworkGameObjects(GameObject *gameObjects[MAX_NETWORK_OBJECTS], uint16 *count);

    uint16 getNetworkGameObjectsCount() const;

    void unregisterNetworkGameObject(GameObject * gameObject);

    void clear();

private:

    // Private attributes...
};
```

ModuleLinkingContext



Both server and client will unregister game objects with the same functions

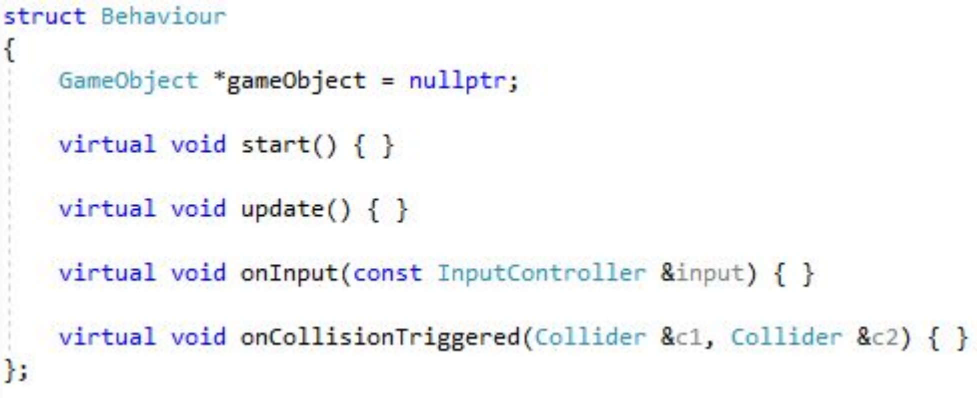


```
class ModuleLinkingContext : public Module
{
public:

    void registerNetworkGameObject(GameObject *gameObject);

    void registerNetworkGameObjectWithNetworkId(GameObject *gameObject, uint32 networkId);

    GameObject *getNetworkGameObject(uint32 networkId);

    void getNetworkGameObjects(GameObject *gameObjects[MAX_NETWORK_OBJECTS], uint16 *count);

    uint16 getNetworkGameObjectsCount() const;

    void unregisterNetworkGameObject(GameObject * gameObject);

    void clear();

private:

    // Private attributes...
};
```

Behaviours

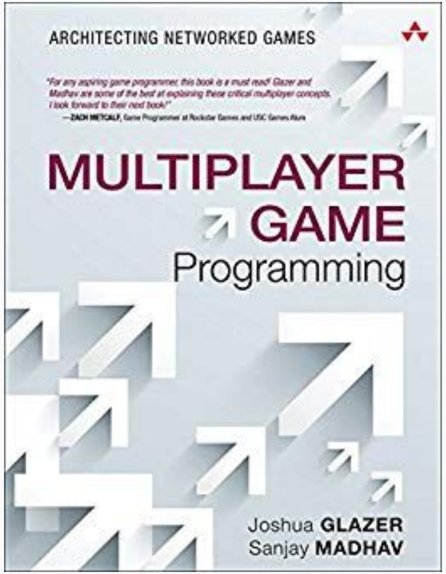
GameObjects can have a behaviour

- Like MonoBehaviour in Unity
- Inherit and override
- Gameplay systems
- Have a look at **Behaviours.h**



```
struct Behaviour
{
    GameObject *gameObject = nullptr;

    virtual void start() { }

    virtual void update() { }

    virtual void onInput(const InputController &input) { }

    virtual void onCollisionTriggered(Collider &c1, Collider &c2) { }
};
```

References


ARCHITECTING NETWORKED GAMES
"For any aspiring game programmer, this book is a must read! Glazer and Madhav are some of the best at explaining these critical multiplayer concepts. I look forward to their next book!"
—ZACH METCALF, Game Programmer at Rockstar Games and USC Games Alum
MULTIPLAYER
GAME
Programming
Joshua GLAZER
Sanjay MADHAV

Related material and help can be found in:

- Multiplayer Game Programming book (Joshua Glazer and Sanjay Madhav)
- Making a Multiplayer FPS in C++ blog (Joe Best-Rotheray)
- Fast paced multiplayer blog (Gabriel Gambetta)
- Gaffer on Games posts in Github (Glenn Fiedler)