**C++ Event & Delegate system**

1. **Function Pointers (from C)**

A function resides in memory and also have addresses. **Function name** can be used to find the address of the function (without parenthesis ())

* Declaring Function pointer

return\_type (\*FuncPtr) (parameter type, ....);

* Passing to another function (callbacks)

A screenshot of a computer code

Description automatically generated with low confidence

1. **Member function pointers**

If two classes Foo and Bar have a function named func() in their declarations, the pointers to these two functions are of two different types. That means, we need a templated class representing our event callbacks, so that we can use any member function of any class as the callback for our event.

We need IEventCallback because else, EventCallback<Character> and EventCallback<AnotherClass> are considered two different types

Each IEventCallback (EventCallback to be exact) is an **action** that corresponds to an “instance” of any class type (template T)

**class** **IEventCallback**

{

public:

**virtual** **void** **operator**() () **=** 0;

**virtual** **bool** **operator** **==** (IEventCallback**\*** other) **=** 0;

};

**template<typename** **T**>

**class** **EventCallback** **:** **public** IEventCallback

{

public:

EventCallback(T**\*** instance, **void** (T**::\***function)())

**:** instance(instance), function(function) {}

**virtual** **void** **operator**() () **override**

{

(instance**->\***function)();

}

**virtual** **bool** **operator** **==** (IEventCallback**\*** other) **override**

{

EventCallback**\*** otherEventCallback **=**

**dynamic\_cast<**EventCallback>(other);

**if** (otherEventCallback **==** nullptr) **return** false;

**return** (**this->**function **==** otherEventCallback**->**function) **&&**

(**this->**instance **==** otherEventCallback**->**instance);

}

private:

**void** (T**::\***function)();

T**\*** instance;

};

**Event class:**

*//Event.h*

**#include "EventCallback.h"**

**class** **Event**

{

public:

Event();

**~**Event();

**void** addListener(IEventCallback**\*** action);

**void** removeListener(IEventCallback**\*** action);

**void** fire();

private:

**typedef** std**::**vector**<**IEventCallback**\*>** CallbackArray;

CallbackArray actions;

};

The actions array contains the list of delegates that would be called when the event is triggered.

*// Event.cpp*

Event**::**Event() { }

Event**::~**Event() { }

**void** Event**::**addListener(IEventCallback**\*** action)

{

auto position **=** find(actions.begin(), actions.end(), action);

**if** (position **!=** actions.end()) **return**;

actions.push\_back(action);

}

**void** Event**::**removeListener(IEventCallback**\*** action)

{

auto position **=** find(actions.begin(), actions.end(), action);

**if** (position **==** actions.end()) **return**;

actions.erase(position);

}

*// Execute all actions (EventCallbacks) 🡪 run the overloaded operator “()”*

**void** Event**::**fire()

{

**for** (IEventCallback**\*** action **:** actions)

{

(**\***action)();

}

}

**Apply to Application**

**class** **InputHandler** **:** **public** Singleton *// Like static in C#*

{

public:  **void** registerAction(**const** std**::**string**&** actionName, IEventCallback**\*** action);

private: std**::**map**<**std**::**string, Event**\*>** inputEvents;

}

**void** InputHandler**::**registerAction(**const** string **&**actionName, IEventCallback**\*** action)

{

**if** (inputEvents.count(actionName) **<** 1)

{

Debug**::**LogError("There is no action named " **+** actionName);

**return**;

}

inputEvents[actionName]**->**addListener(action);

}

Input handler has a Map. Each entry in the map (“Jump”, “Attack”) matches with an Event. Each Event has multiple Actions (IEventCallbacks)

Say we have an Actor object that has “uselessFunction()” – an **action (EventCallback)**. We then register it to

Actor**\*** actor **=** **new** **Actor**();

EventCallback**\*** callback **=** **new** **EventCallback**(actor, **&**Actor**::**uselessFunction);

InputHandler**::**getInstance()**->**registerAction("Jump", callback);