```cpp
/* Generic event class to create an event in the game
* It uses function pointer for observers
*/
#pragma once
#include <map>

template<typename T, typename... TArgs>
class Event
{
private:
    T* m_instance; //instance pointer
    void(T::* m_function)(TArgs...); //function pointer with TArgs arguments

    //a container for all the observers
    std::vector<std::pair<T*, void(T::*)(TArgs...)>> m_observers;


public:
    Event() : m_instance(nullptr) {};
    ~Event() {};

    //notify all registered observers
    //by calling their member function pointer with the provided arguments
    inline void Notify(const TArgs&...args)
    {
        if (m_observers.size() == 0) return;

        for (const auto& observer : m_observers)
        {
            (observer.first->*observer.second)(args...);
        }
    }

    //registers an observer by storing its instance pointer and function pointer
    //in the m_observers vector
    inline void Register(T* instance, void (T::* function)(TArgs...))
    {
        m_instance = instance;
        m_function = function;

        auto function_pointer = std::make_pair(m_instance, m_function);

        m_observers.emplace_back(function_pointer);
    }

    //unregister an observer by removing its instance pointer and function pointer
```

```
47      //from the m_observers vector
48      inline void Unregister(T* instance, void (T::* function)(TArgs...))
49      {
50          auto iter = std::remove_if(m_observers.begin(), m_observers.end(),
51              [=](const auto& observer) {
52                  return observer.first == instance && observer.second ==
                      function;
53              });
54
55          m_observers.erase(iter, m_observers.end());
56      }
57  };
```