```cpp
#ifndef EVENTMANAGER_HPP
#define EVENTMANAGER_HPP

#include <vector>
#include <unordered_map>
#include <functional>
#include <SFML/Graphics.hpp>

/// Enumerations of the different type of events.
enum class EventType {
    KeyDown         = sf::Event::KeyPressed,
    Key             = sf::Event::KeyReleased,
    MButtonDown     = sf::Event::MouseButtonPressed,
    MButtonUp       = sf::Event::MouseButtonReleased,
    MouseWheel      = sf::Event::MouseWheelMoved,
    WindowResized   = sf::Event::Resized,
    GainedFocus     = sf::Event::GainedFocus,
    LostFocus       = sf::Event::LostFocus,
    MouseEntered    = sf::Event::MouseEntered,
    MouseLeft       = sf::Event::MouseLeft,
    Closed          = sf::Event::Closed,
    TextEntered     = sf::Event::TextEntered,
    Keyboard        = sf::Event::Count + 1, Mouse, Joystick
};

/// Structure to store the informations about the keys been pressed.
struct EventInfo {
    EventInfo() { m_code = 0; }
    EventInfo(int l_event) { m_code = l_event; }

    union {
        int m_code;
    };
};

/// Data type to hold the information on the event.
using Events = std::vector<std::pair<EventType, EventInfo>>;

/// Details about the event actually handling.
struct EventDetails {
    EventDetails(std::string& l_bindName)
        : m_name(l_bindName)
    {
        Clear();
    }

    std::string m_name;
    sf::Vector2i m_size;
    sf::Uint32 m_textEntered;
    sf::Vector2i m_mouse;
    int m_keyCode;      /// Single key code.

    void Clear()
    {
        m_size = sf::Vector2i(0, 0);
        m_textEntered = 0;
        m_mouse = sf::Vector2i(0, 0);
        m_keyCode = -1;
    }
};

/// Structure that is going  to hold all the event infos.
struct Binding {
    Binding(std::string l_name)
        : m_name(l_name), m_details(l_name), c(0) {}

```

```
 67      void BindEvent(EventType l_type, EventInfo l_info = EventInfo())
 68      {
 69          m_event.emplace_back(l_type, l_info);
 70      }
 71
 72      Events m_event;
 73      std::string m_name;
 74      int c;      /// Count of events that are "happening".
 75
 76      EventDetails m_details;
 77  } ;
 78
 79  /// Data structure to hold all the bindings.
 80  using Bindings = std::unordered_map<std::string, Binding*>;
 81
 82  /// Type of the callback container.
 83  using Callbacks = std::unordered_map<std::string, std::function<void(EventDetails
*)>>;
 84
 85
 86  /// The EventManager class.
 87  class EventManager {
 88  public:
 89      EventManager();
 90      ~EventManager();
 91
 92      bool AddBinding(Bindings* l_binding);
 93      bool RemoveBinding(std::string l_name);
 94      void SetFocus(const bool& l_focus);
 95
 96      /// The callback.
 97      template<class T>
 98      bool AddCallback(const std::string& l_name, void(T::*l_func) (EventDetails*), T*
l_instance)
 99      {
100          auto temp = std::bind(l_func, l_instance, std::placeholders::_1);
101          return m_callbacks.emplace(l_name, temp).second;
102      }
103
104      void RemoveCallback(const std::string& l_name)
105      {
106          m_callbacks.erase(l_name);
107      }
108
109      void HandleEvent(sf::Event& l_event);
110      void Update();
111
112      sf::Vector2i GetMousePos(sf::RenderWindow* l_wind = nullptr)
113      {
114          return (l_wind ? sf::Mouse::getPosition(*l_wind) : sf::Mouse::getPosition
());
115      }
116
117  private:
118      void LoadBindings();
119
120      Bindings m_bindings;
121      Callbacks m_callbacks;
122      bool m_hasFocus;
123  };
124
125  #endif // EVENTMANAGER_HPP
```