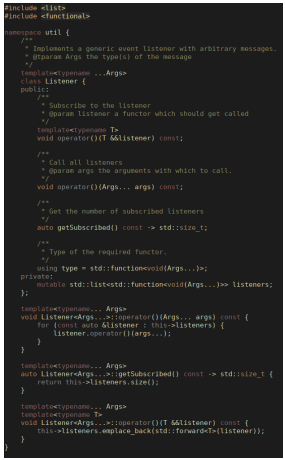
SoPra-Team-10

Server

May 5, 2019

Überblick



```
#include <list>
#include <functional>

namespace util {
    /**
     * Implements a generic event listener with arbitrary messages.
     * @tparam Args the type(s) of the message
     */
    template<typename ...Args>
    class Listener {
    public:
        /**
         * Subscribe to the listener
         * @param listener a functor which should get called
         */
        template<typename T>
        void operator()(T &&listener) const;

        /**
         * Call all listeners
         * @param args the arguments with which to call.
         */
        void operator()(Args... args) const;

        /**
         * Get the number of subscribed listeners
         */
        auto getSubscribed() const -> std::size_t;

        /**
         * Type of the required functor.
         */
        using type = std::function<void(Args...)>;
    private:
        mutable std::list<std::function<void(Args...)>> listeners;
    };

    template<typename... Args>
    void Listener<Args...>::operator()(Args... args) const {
        for (const auto &listener : this->listeners) {
            listener.operator()(args...);
        }
    }

    template<typename... Args>
    auto Listener<Args...>::getSubscribed() const -> std::size_t {
        return this->listeners.size();
    }

    template<typename... Args>
    template<typename T>
    void Listener<Args...>::operator()(T &&listener) const {
        this->listeners.emplace_back(std::forward<T>(listener));
    }
}
```

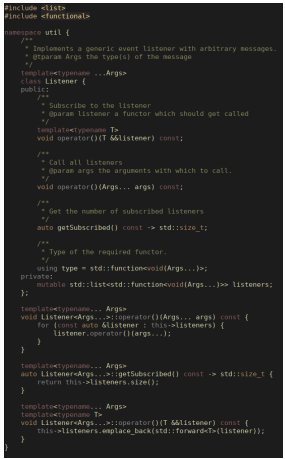
- Vollständig in C++17 geschrieben

Überblick

```cpp
#include <list>
#include <functional>

namespace util {
    /**
     * Implements a generic event listener with arbitrary messages.
     * @tparam Args the type(s) of the message
     */
    template<typename ...Args>
    class Listener {
    public:
        /**
         * Subscribe to the listener
         * @param listener a functor which should get called
         */
        template<typename T>
        void operator()(T &&listener) const;

        /**
         * Call all listeners
         * @param args the arguments with which to call.
         */
        void operator()(Args... args) const;

        /**
         * Get the number of subscribed listeners
         */
        auto getSubscribed() const -> std::size_t;

        /**
         * Type of the required functor.
         */
        using type = std::function<void(Args...)>;
    private:
        mutable std::list<std::function<void(Args...)>> listeners;
    };

    template<typename... Args>
    void Listener<Args...>::operator()(Args... args) const {
        for (const auto &listener : this->listeners) {
            listener.operator()(args...);
        }
    }

    template<typename... Args>
    auto Listener<Args...>::getSubscribed() const -> std::size_t {
        return this->listeners.size();
    }

    template<typename... Args>
    template<typename T>
    void Listener<Args...>::operator()(T &&listener) const {
        this->listeners.emplace_back(std::forward<T>(listener));
    }
}
```

- Vollständig in C++17 geschrieben
- Modularer Aufbau (Trennung in Network-, Messages-, GameLogic- und Serverkomponente)

Überblick



```
#include <list>
#include <functional>

namespace util {
    /**
     * Implements a generic event listener with arbitrary messages.
     * @tparam Args the type(s) of the message
     */
    template<typename ...Args>
    class Listener {
    public:
        /**
         * Subscribe to the listener
         * @param listener a functor which should get called
         */
        template<typename T>
        void operator()(T &&listener) const;

        /**
         * Call all listeners
         * @param args the arguments with which to call.
         */
        void operator()(Args... args) const;

        /**
         * Get the number of subscribed listeners
         */
        auto getSubscribed() const -> std::size_t;

        /**
         * Type of the required functor.
         */
        using type = std::function<void(Args...)>;
    private:
        mutable std::list<std::function<void(Args...)>> listeners;
    };

    template<typename... Args>
    void Listener<Args...>::operator()(Args... args) const {
        for (const auto &listener : this->listeners) {
            listener.operator()(args...);
        }
    }

    template<typename... Args>
    auto Listener<Args...>::getSubscribed() const -> std::size_t {
        return this->listeners.size();
    }

    template<typename... Args>
    template<typename T>
    void Listener<Args...>::operator()(T &&listener) const {
        this->listeners.emplace_back(std::forward<T>(listener));
    }
}
```

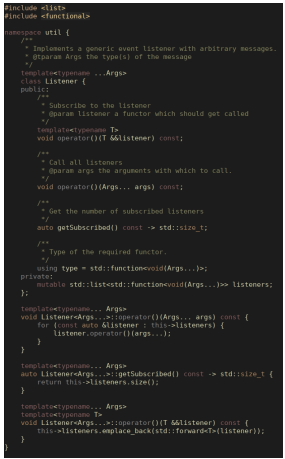
- Vollständig in C++17 geschrieben
- Modularer Aufbau (Trennung in Network-, Messages-, GameLogic- und Serverkomponente)
- MVC

Überblick

```cpp
#include <list>
#include <functional>

namespace util {
    /**
     * Implements a generic event listener with arbitrary messages.
     * @tparam Args the type(s) of the message
     */
    template<typename ...Args>
    class Listener {
    public:
        /**
         * Subscribe to the listener
         * @param listener a functor which should get called
         */
        template<typename T>
        void operator()(T &&listener) const;

        /**
         * Call all listeners
         * @param args the arguments with which to call.
         */
        void operator()(Args... args) const;

        /**
         * Get the number of subscribed listeners
         */
        auto getSubscribed() const -> std::size_t;

        /**
         * Type of the required functor.
         */
        using type = std::function<void(Args...)>;
    private:
        mutable std::list<std::function<void(Args...)>> listeners;
    };

    template<typename... Args>
    void Listener<Args...>::operator()(Args... args) const {
        for (const auto &listener : this->listeners) {
            listener.operator()(args...);
        }
    }

    template<typename... Args>
    auto Listener<Args...>::getSubscribed() const -> std::size_t {
        return this->listeners.size();
    }

    template<typename... Args>
    template<typename T>
    void Listener<Args...>::operator()(T &&listener) const {
        this->listeners.emplace_back(std::forward<T>(listener));
    }
}
```

- Vollständig in C++17 geschrieben
- Modularer Aufbau (Trennung in Network-, Messages-, GameLogic- und Serverkomponente)
- MVC
- Replay implementiert

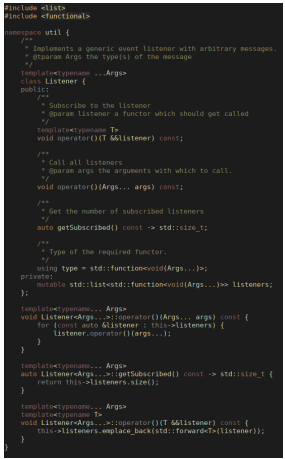
Überblick



```
#include <list>
#include <functional>

namespace util {
    /**
     * Implements a generic event listener with arbitrary messages.
     * @tparam Args the type(s) of the message
     */
    template<typename ...Args>
    class Listener {
    public:
        /**
         * Subscribe to the listener
         * @param listener a functor which should get called
         */
        template<typename T>
        void operator()(T &&listener) const;

        /**
         * Call all listeners
         * @param args the arguments with which to call.
         */
        void operator()(Args... args) const;

        /**
         * Get the number of subscribed listeners
         */
        auto getSubscribed() const -> std::size_t;

        /**
         * Type of the required functor.
         */
        using type = std::function<void(Args...)>;
    private:
        mutable std::list<std::function<void(Args...)>> listeners;
    };

    template<typename... Args>
    void Listener<Args...>::operator()(Args... args) const {
        for (const auto &listener : this->listeners) {
            listener.operator()(args...);
        }
    }

    template<typename... Args>
    auto Listener<Args...>::getSubscribed() const -> std::size_t {
        return this->listeners.size();
    }

    template<typename... Args>
    template<typename T>
    void Listener<Args...>::operator()(T &&listener) const {
        this->listeners.emplace_back(std::forward<T>(listener));
    }
}
```

- Vollständig in C++17 geschrieben
- Modularer Aufbau (Trennung in Network-, Messages-, GameLogic- und Serverkomponente)
- MVC
- Replay implementiert
- Alle Mods implementiert

Überblick



```
#include <list>
#include <functional>

namespace util {
    /**
     * Implements a generic event listener with arbitrary messages.
     * @tparam Args the type(s) of the message
     */
    template<typename ...Args>
    class Listener {
    public:
        /**
         * Subscribe to the listener
         * @param listener a functor which should get called
         */
        template<typename T>
        void operator()(T &&listener) const;

        /**
         * Call all listeners
         * @param args the arguments with which to call.
         */
        void operator()(Args... args) const;

        /**
         * Get the number of subscribed listeners
         */
        auto getSubscribed() const -> std::size_t;

        /**
         * Type of the required functor.
         */
        using type = std::function<void(Args...)>;
    private:
        mutable std::list<std::function<void(Args...)>> listeners;
    };

    template<typename... Args>
    void Listener<Args...>::operator()(Args... args) const {
        for (const auto &listener : this->listeners) {
            listener.operator()(args...);
        }
    }

    template<typename... Args>
    auto Listener<Args...>::getSubscribed() const -> std::size_t {
        return this->listeners.size();
    }

    template<typename... Args>
    template<typename T>
    void Listener<Args...>::operator()(T &&listener) const {
        this->listeners.emplace_back(std::forward<T>(listener));
    }
}
```

- Vollständig in C++17 geschrieben
- Modularer Aufbau (Trennung in Network-, Messages-, GameLogic- und Serverkomponente)
- MVC
- Replay implementiert
- Alle Mods implementiert
- Mehrere zeitgleiche Spiele

Krasse Features

- 167 Unit Tests

Krasse Features

- 167 Unit Tests
- Mit Doxygen dokumentiert

Krasse Features

- 167 Unit Tests
- Mit Doxygen dokumentiert
- Statische und Dynamische Analyse (Clang-Tidy und Address-Sanitizer)

Krasse Features

- 167 Unit Tests
- Mit Doxygen dokumentiert
- Statische und Dynamische Analyse (Clang-Tidy und Address-Sanitizer)
- Fertiges Docker-Image

Krasse Features

- 167 Unit Tests
- Mit Doxygen dokumentiert
- Statische und Dynamische Analyse (Clang-Tidy und Address-Sanitizer)
- Fertiges Docker-Image
- CI (Testen des Docker-Images, Unittests (mehrfache Wiederholung in unterschiedlicher Reihenfolge))