Arcade Documentation

Introduction

The are multiple Inheritance, one for the Graphical libraries, another one for the games. First you will see a short explanation of some of the functions of both inheritances, and later you will see both inheritances.

```
namespace arcade {
    enum Type {
        Graphic,
        Text
    };

    julesd7, 3 weeks ago | 1 author (julesd7)
    class IObject {
        public:
            virtual ~IObject() = default;
            virtual std::array<int, 3> const &getRGB(void) const = 0;
            virtual std::array<int, 2> const &getPos(void) const = 0;
            virtual std::array<int, 2> const &getSize(void) const = 0;
            virtual std::string const &getAsset(void) const = 0;
            virtual char getAscii(void) const = 0;
            virtual Type getType(void) const = 0;
    };

    julesd7, 3 weeks ago | 1 author (julesd7)
    class IEvent {
        public:
            virtual ~IEvent() = default;
            virtual const std::string &type() const = 0;
    };

    julesd7, 3 weeks ago | 1 author (julesd7)
    typedef struct data_s {
        std::vector<std::unique_ptr<IObject>> objects;
        std::pair<int, int> size;
    } data_t;
```

```
class IDisplayModule {
    public :
        virtual ~IDisplayModule() = default;
        virtual void display(const data_t& data) = 0;
        virtual void clear(void) = 0;
        virtual std::vector<std::unique_ptr<IEvent>> const &getEvent(void) = 0;
};

julesd7, 3 weeks ago | 1 author (julesd7)
class IGameModule {
    public:
        virtual ~IGameModule() = default;
        virtual void manageEvent(std::vector<std::unique_ptr<IEvent>> const &) = 0;
        virtual void iterate(void) = 0;
        virtual data_t const &getMap(void) const = 0;
};
```

Graphic:

This line is declaring a possible value for the Type enumeration. If a variable of type Type is set to Graphic, it represents some kind of graphical object.

Text:

This line is declaring another possible value for the Type enumeration. If a variable of type Type is set to Text, it represents some kind of text object.

virtual ~IObject() = default;

This is the destructor for the IObject class. It's declared as virtual to ensure that the destructors of derived classes are called correctly when an object is deleted through a pointer to the base class. The = default; means that the compiler will generate a default destructor.

virtual std::array<int, 3> const &getRGB(void) const = 0;

This is a pure virtual function that must be implemented by any class that inherits from IObject. It should return a constant reference to an std::array<int, 3>. This array represents the RGB color values of the object. The function is const, meaning it doesn't modify any member variables of the class.

virtual std::array<int, 2> const &getPos(void) const = 0;

This is a pure virtual function that must be implemented by any class that inherits from IObject. It should return a constant reference to an std::array<int, 2>. This array represents the position of the object in a 2D space. The function is const, meaning it doesn't modify any member variables of the class.

virtual std::array<int, 2> const &getSize(void) const = 0;

This is a pure virtual function that must be implemented by any class that inherits from IObject. It should return a constant reference to an std::array<int, 2>. This array represents the size of the object in a 2D space. The function is const, meaning it doesn't modify any member variables of the class.

virtual std::string const &getAsset(void) const = 0;

This is a pure virtual function that must be implemented by any class that inherits from IObject. It should return a constant reference to a std::string. This string represents the asset associated with the object (like a texture or sprite). The function is const, meaning it doesn't modify any member variables of the class.

virtual char getAscii(void) const = 0;

This is a pure virtual function that must be implemented by any class that inherits from IObject. It should return a char. This character represents the ASCII representation of the object. The function is const, meaning it doesn't modify any member variables of the class.

virtual Type getType(void) const = 0;

This is a pure virtual function that must be implemented by any class that inherits from IObject. It should return a Type. Type is likely an enumeration that represents the type of the object. The function is const, meaning it doesn't modify any member variables of the class.

virtual ~IEvent() = default;

This line is declaring the destructor for the IEvent class. It's declared as virtual to ensure that the destructors of derived classes are called correctly when an object is

deleted through a pointer to the base class. The = default; means that the compiler will generate a default destructor.

virtual const std::string &type() const = 0;

This line is declaring a pure virtual function named type. This function must be implemented by any class that inherits from IEvent. It should return a constant reference to a std::string, which likely represents the type of the event. The function is const, meaning it doesn't modify any member variables of the class.

std::vector<std::unique_ptr<IObject>> objects;

This line is declaring a member of the struct named objects. This member is a std::vector of std::unique_ptr<IObject>. std::unique_ptr is a smart pointer that retains sole ownership of an object through a pointer. IObject is presumably a class defined elsewhere in your code. This means objects is a vector of unique pointers to IObject instances.

std::pair<int, int> size;

This line is declaring another member of the struct named size. This member is a std::pair<int, int>, which can hold two int values. This is likely used to represent the size of something, possibly the size of a game board or a window.

virtual ~IDisplayModule() = default;

This line is declaring the destructor for the IDisplayModule class. It's declared as virtual to ensure that the destructors of derived classes are called correctly when an object is deleted through a pointer to the base class. The = default; means that the compiler will generate a default destructor.

virtual void display(const data_t& data) = 0;

This line is declaring a pure virtual function named display. This function must be implemented by any class that inherits from IDisplayModule. It takes a constant reference to a data_t as a parameter, which likely represents the data to be displayed.

virtual void clear(void) = 0;

This line is declaring another pure virtual function named clear. This function must be implemented by any class that inherits from IDisplayModule. It likely clears the display.

virtual std::vector<std::unique_ptr<IEvent>> const &getEvent(void) = 0;

This line is declaring another pure virtual function named getEvent. This function must be implemented by any class that inherits from IDisplayModule. It should return a constant reference to a std::vector of std::unique_ptr<IEvent>. This likely represents a list of events that have occurred.

virtual ~IGameModule() = default;

This line is declaring the destructor for the IGameModule class. It's declared as virtual to ensure that the destructors of derived classes are called correctly when an object is deleted through a pointer to the base class. The = default; means that the compiler will generate a default destructor.

virtual void manageEvent(std::vector<std::unique_ptr<IEvent>> const &) = 0;

This line is declaring a pure virtual function named manageEvent. This function must be implemented by any class that inherits from IGameModule. It takes a constant reference to a std::vector of std::unique_ptr<IEvent> as a parameter, which likely represents a list of events to be managed.

virtual void iterate(void) = 0;

This line is declaring another pure virtual function named iterate. This function must be implemented by any class that inherits from IGameModule. It likely performs an iteration of the game loop.

virtual data_t const &getMap(void) const = 0;

This line is declaring another pure virtual function named getMap. This function must be implemented by any class that inherits from IGameModule. It should return a constant reference to a data_t, which likely represents the game map.