Wiesn-Run

Erzeugt von Doxygen 1.8.6

Son Jul 12 2015 12:06:32

Inhaltsverzeichnis

Kapitel 1

Ausstehende Aufgaben

Element Enemy::Enemy (int posX, int posY, int speedX, objectType enemy)

Skalieren der Werte und fireCooldown erhöhen

Element Game::step ()

Erfolgreich Schriftzug einfügen

GameOver schriftzug einfügen

Element Player::decreaseAlcoholLevel (int decreaseLevel)

Überflüssig, da nie aufgerufen. Auch wenn der Name es nicht vermuten lässt: increaseAlcoholLevel kann den Level auch verringern und wird benutzt.

Klasse scoreStruct

Das Konzept der Alkohol-Punkte muss noch ausgearbeitet werden.

Autor

Simon

Klasse stateStruct

Diese Struktur ist vermutlich überflüssig.

Autor

Simon

Kapitel 2

Hierarchie-Verzeichnis

2.1 Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 5

Klassen-Dokumentation

5.1 Audio Klassenreferenz

Audio-Klasse Die Audio-Klasse erzeugt Audioobjekte. Mehrere Instanzen dieser Klasse werden in der Klasse Audio-Control erstellt. Jedes Audioobjekt liest die zum Ojekt gehörigen Audiosamples ein und übergibt diese an die Kontrollklasse Audiocontrol, welche im Anschluss die Samples aller Objekte mischt und abspielt. Die einzelnen Methoden werden in der audio.cpp erklärt.

`#include <audio.h>`

Öffentliche Methoden

- Audio (std::string state_name)

  Audio::Audio Konstruktor instanziert ein Objekt der Klasse Audio.
- ∼Audio ()

  Audio::∼Audio Destruktor löscht ein Objekt der Klasse Audio.
- std::string getSource ()

  Audio::getSource "getSource" gibt bei Aufruf den Namen des Objektes zurück welcher welcher dem Pfad in der Ressourcendatenbank entspricht.
- float getSample (int pos)

  Audio::getSample "getSample" gibt bei Aufruf das Sample an Position = pos der zu Audioobjekt gehörigen Wave Datei mit Bittiefe 16 bit zurück.
- int getSamplenumber ()

  Audio::getSamplenumber "getSamplenumber" gibt bei Aufruf die Anzahl an Samples der zu Audioobjekt gehörigen Wave Datei zurück.

5.1.1 Ausführliche Beschreibung

Audio-Klasse Die Audio-Klasse erzeugt Audioobjekte. Mehrere Instanzen dieser Klasse werden in der Klasse Audio-Control erstellt. Jedes Audioobjekt liest die zum Ojekt gehörigen Audiosamples ein und übergibt diese an die Kontrollklasse Audiocontrol, welche im Anschluss die Samples aller Objekte mischt und abspielt. Die einzelnen Methoden werden in der audio.cpp erklärt.

Autor

Felix Pfreundtner

5.1.2 Beschreibung der Konstruktoren und Destruktoren

5.1.2.1 Audio::Audio (std::string *state_name*)

Audio::Audio Konstruktor instanziert ein Objekt der Klasse Audio.

Autor

Felix Pfreundtner

5.1.2.2 Audio::~Audio ( )

Audio::~Audio Destruktor löscht ein Objekt der Klasse Audio.

Autor

Felix Pfreundtner

5.1.3 Dokumentation der Elementfunktionen

5.1.3.1 float Audio::getSample (int *pos*)

Audio::getSample "getSample" gibt bei Aufruf das Sample an Position = pos der zu Audioobjekt gehörigen Wave Datei mit Bittiefe 16 bit zurück.

Rückgabe

float sample

Autor

Felix Pfreundtner

gebe Sample des Audioobjekts an der Stelle pos zurück

5.1.3.2 int Audio::getSamplenumber ( )

Audio::getSamplenumber "getSamplenumber" gibt bei Aufruf die Anzahl an Samples der zu Audioobjekt gehörigen Wave Datei zurück.

Rückgabe

int samplenumber

Autor

Felix Pfreundtner

5.1.3.3 std::string Audio::getSource ( )

Audio::getSource "getSource" gibt bei Aufruf den Namen des Objektes zurück welcher welcher dem Pfad in der Ressourcendatenbank entspricht.

Rückgabe

std::string source

Autor

Felix Pfreundtner

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/audio.h
- Wiesn-Run/src/audio.cpp

5.2 AudioControl Klassenreferenz

AudioControl-Klasse Die AudioControl-Klasse synchronisiert alle Audioausgabeanweisungen und spielt passende Audioobjekte ab. Eine Instanz dieser Klasse wir innerhalb der game.h angelegt. Die einzelnen Methoden werden in der audiocontrol.cpp erklärt.

`#include <audiocontrol.h>`

Klassen

- struct paData
- struct playStruct

Öffentliche Typen

- enum **statusFilter** { **no**, **alcohol**, **life**, **lifecritical** }

Öffentliche Methoden

- AudioControl ()

  AudioControl::AudioControl Konstruktor instanziert ein Objekt der Klasse AudioControl.
- ∼AudioControl ()

  AudioControl::∼AudioControl Destruktor löscht ein Objekt der Klasse AudioControl.
- void playInitialize ()

  playInitialize playInitialize initialisiert die Abspielbibliothek Portaudio, öffenet den PortAudio Stream pastream und startet eine Callback Audiowiedergabe
- void playTerminate ()

  AudioControl::playTerminate Stoppt bei Aufruf PortAudio Audioausgabe, beendet im Anschluss Portaudio Stream und beendet zuletzt PortAudio.
- void updatePlayevents (std::list< struct audioStruct > ∗audioevents)

  updatePlayevents Nach Aufruf über Game::step aktualisiert updatePlayevents alle im Moment abgespielten, in der Liste "playevents" gespeicherten playStruct's mit aktuellen audioStruct's aus der übergebenen Liste audioevents.

5.2.1 Ausführliche Beschreibung

AudioControl-Klasse Die AudioControl-Klasse synchronisiert alle Audioausgabeanweisungen und spielt passende Audioobjekte ab. Eine Instanz dieser Klasse wir innerhalb der game.h angelegt. Die einzelnen Methoden werden in der audiocontrol.cpp erklärt.

Autor

Felix Pfreundtner

5.2.2 Beschreibung der Konstruktoren und Destruktoren

5.2.2.1 AudioControl::AudioControl ( )

AudioControl::AudioControl Konstruktor instanziert ein Objekt der Klasse AudioControl.

Autor

Felix Pfreundtner

erstelle für jede objektgruppe "type" ein audio Objekt welches unter anderem die Samples beinhaltet

setzte Status Filter auf kein Filter

setzte blockcounter auf 0 Blöcke

setzte Wartezeit von Portaudio auf 100 ms

setzte maximale Anzahl an Playevents auf 5 Wird der Wert höher gesetzt wird die Lautstärke geringer. Wird der Wert geringer gesetzt steigt die Gefahr des Clippings des Ausgabesignals Ein Normalisieren aller Ausgabeblöcke wäre möglich, würde jedoch 2D Audio nicht erlauben, da auch die Dynamik zwischen zwei Blöcken normalisiert wird -> Distanz- und somit Lautstärkeänderungen von Objekten werden mit normalisiert.

5.2.2.2 AudioControl::~AudioControl ( )

AudioControl::~AudioControl Destruktor löscht ein Objekt der Klasse AudioControl.

Autor

Felix Pfreundtner

5.2.3 Dokumentation der Elementfunktionen

5.2.3.1 void AudioControl::playInitialize ( )

playInitialize playInitialize initialisiert die Abspielbibliothek Portaudio, öffenet den PortAudio Stream pastream und startet eine Callback Audiowiedergabe

Parameter

Qlist	audioevents

Autor

Felix Pfreundtner

initialisiere Port Audio

Öffene Ausgabe Stream pastream

erstelle keine Eingangskänale

erstelle Mono Audio Ausgabe

setze Bittiefe der Audioausgabe 16 bit Integer

setze Samplerate der Audioausgabe zu 44100 Hz

setze Anzahl an Samples per Bufferblock auf 1024

verweise auf Static Callback Funktion

übergebe User-Data

Starte PortAudio Stream pastream

Pausiere Funktion wenn Audiostream gerade aktiv ist (Audiowiedergabe übernimmt Callback Funktion)

5.2.3.2 void AudioControl::playTerminate ( )

AudioControl::playTerminate Stoppt bei Aufruf PortAudio Audioausgabe, beendet im Anschluss Portaudio Stream und beendet zuletzt PortAudio.

Autor

Felix Pfreundtner

Stoppe den Portaudio Stream

Schließe den Portaudio Stream

Beende PortAudio

5.2.3.3 void AudioControl::updatePlayevents (std::list< struct audioStruct > * *audioevents*)

updatePlayevents Nach Aufruf über Game::step aktualisiert updatePlayevents alle im Moment abgespielten, in der Liste "playevents" gespeicherten playStruct's mit aktuellen audioStruct's aus der übergebenen Liste audioevents.

Parameter

Qlist	audioevents

Autor

Felix Pfreundtner

erstelle neues temporäres audioStruct, welches stets das aktuelle audioStruct Element der Liste audioevents beinhaltet.

erstelle neues temporäres audioStruct, welches stets das aktuelle playStruct Element der Liste playevents beinhaltet.

erstelle Variable welche true ist wenn die id von newplaystruct bereits in playevents vorhanden ist

erstelle einen Iterator für playevents Liste

intialsiere status_filter mit no (Annahme kein Audioevent mit type status_alcohol / status_life / status_lifecritical in audioevents Liste)

lock audioevents

initialisiere die Abspielinformation aller playstructs in playevents auf false (verhindere weiteres abspielen im nächsten Step)

aktualisiere die Abspielinformation und Distanzwerte aller playstructs aus playevents mit aktuellen GameLogik Werten iteriere über alle audioStructs der audioevents Liste

entnehme neues audioStruct aus audioevents Liste

setzte Variable nasidexistinpe auf false, da newaudiostruct bisher nicht in playevents gefunden wurde

iteriere über alle bereits bestehenden playStructs in Liste playevents

falls die id eines neuen audiostruct bereits in diesem playStruct von playevents vorhanden ist (also bereits abgespielt wird)

übernehmen die aktuellen Distanzwerte des neuen audiostructs und wandle sie in eine Volumen Information um (volume = 1 - distance).

setzte playnext auf true, da das playstruct auch im nächsten Step abgespielt werden soll

setzte Variable nasidexistinpe auf true, da newaudiostruct bisher in playevents gefunden wurde

wenn die id von newaudiostruct noch nicht in audioevents vorhanden ist (struct noch nicht abgespielt wird)

schreibe ID des neuen audioStruct in ein neues playStruct

schreibe Gruppen Type des neuen audioStruct in ein neues playStruct

übernehmen die aktuellen Distanzwerte des neuen audiostructs und wandle sie in eine Volumen Information um (volume = 1 - distance).

speichere einen Zeiger auf das (Audio-)Objekt in audioobjects in newplaystruct

setzte Abspielposition auf 0 Samples (Beginne Abspielen)

setzte playnext auf true, da das playstruct auch im nächsten Step abgespielt werden soll

füge das neue playstruct der Liste playevents hinzu

Lösche audioStruct aus audioevents Liste da alle Werte in playevents übernommen wurden

für alle Playevents in der playevents liste

falls Playevent im nächsten Step nicht mehr abgespielt werden sollen (playnext = false).

lösche playevent aus playevents List

falls playevent im nächsten Step abgespielt werden soll

falls Type des aktuellen playevents status_alcohol ist (niedrige Priorität)

setze Alkohol Status auf aktiv

falls Type des aktuellen playevents status_life ist (mittlere Priorität)

setze Alkohol Status auf aktiv

falls Type des aktuellen playevents status_lifecritical ist (höchste Priorität)

setze Alkohol Status auf aktiv

unlock audioevents

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/audiocontrol.h
- Wiesn-Run/src/audiocontrol.cpp

5.3 audioCooldownstruct Strukturreferenz

Struktur für audioevents mit ihrer abspielzeit als Cooldown.

`#include <definitions.h>`

Öffentliche Attribute

- struct audioStruct **audioEvent**
- std::chrono::duration< int, std::milli > **cooldown**

5.3.1 Ausführliche Beschreibung

Struktur für audioevents mit ihrer abspielzeit als Cooldown.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.4 audioCooldownStruct Strukturreferenz

Typdef Struct mit Konstanten für den Audiocooldown jedes Audiotypes In diesen Konstanten wird festgelegt wie viele millisekunden für ein Event (mit "id=...") eines audioTypes trotz verschwinden in der Grafik nachwievor audioStructs gesendet werden. Ein 0 bedeteutet, dass kein Cooldown erfolgt, die Audiostructs werden hier solange gesendet wie das Event sichtbar ist.

`#include <definitions.h>`

Öffentliche Attribute

- std::chrono::duration< int, std::milli > **scene_flyingbeer** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **scene_enemy_security** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **scene_enemy_tourist** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **scene_enemy_boss** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **scene_collision_obstacle** = std::chrono::milliseconds(500)
- std::chrono::duration< int, std::milli > **scene_collision_enemy** = std::chrono::milliseconds(1639)
- std::chrono::duration< int, std::milli > **scene_collision_player** = std::chrono::milliseconds(899)
- std::chrono::duration< int, std::milli > **scene_collision_flyingbeer** = std::chrono::milliseconds(1211)
- std::chrono::duration< int, std::milli > **powerup_beer** = std::chrono::milliseconds(2989)
- std::chrono::duration< int, std::milli > **powerup_food** = std::chrono::milliseconds(3989)
- std::chrono::duration< int, std::milli > **status_alcohol** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **status_life** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **status_lifecritical** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **status_dead** = std::chrono::milliseconds(4989)
- std::chrono::duration< int, std::milli > **player_walk** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **player_jump** = std::chrono::milliseconds(700)
- std::chrono::duration< int, std::milli > **background_menu** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **background_highscore** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **background_level1** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **background_level2** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **background_level3** = std::chrono::milliseconds(0)
- std::chrono::duration< int, std::milli > **background_startgame** = std::chrono::milliseconds(3000)
- std::chrono::duration< int, std::milli > **background_levelfinished** = std::chrono::milliseconds(5365)

5.4.1 Ausführliche Beschreibung

Typdef Struct mit Konstanten für den Audiocooldown jedes Audiotypes In diesen Konstanten wird festgelegt wie viele millisekunden für ein Event (mit "id=...") eines audioTypes trotz verschwinden in der Grafik nachwievor audioStructs gesendet werden. Ein 0 bedeteutet, dass kein Cooldown erfolgt, die Audiostructs werden hier solange gesendet wie das Event sichtbar ist.

Autor

Felix

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.5 audioDistanceStruct Strukturreferenz

Typdef Struct mit Konstanten für die Distance jedes Audiotypes In diesen Konstanten wird festgelegt wie weit entfernt ein Event (mit ïd=...) eines audioTypes vom Spieler standardmäßig auftritt [Werbereicht 0 (beim spieler) bis 1(maximale Distanz des Fensters). Ist die Konstante -1 ist die Distance eines Events vom Typ audioType variabel und muss von der Gamelogik bestimmt werden.

`#include <definitions.h>`

Öffentliche Attribute

- float **scene_flyingbeer** = -1
- float **scene_enemy_tourist** = -1
- float **scene_enemy_security** = -1
- float **scene_enemy_boss** = -1
- float **scene_collision_obstacle** = 0
- float **scene_collision_enemy** = 0
- float **scene_collision_player** = 0
- float **scene_collision_flyingbeer** = 0
- float **powerup_beer** = 0
- float **powerup_food** = 0
- float **status_alcohol** = 0
- float **status_life** = 0
- float **status_lifecritical** = 0
- float **status_dead** = 0
- float **player_walk** = 0
- float **player_jump** = 0
- float **background_menu** = 0.0
- float **background_highscore** = 0.1
- float **background_level1** = 0.3
- float **background_level2** = 0.3
- float **background_level3** = 0.3
- float **background_startgame** = 0.0
- float **background_levelfinished** = 0.0

5.5.1 Ausführliche Beschreibung

Typdef Struct mit Konstanten für die Distance jedes Audiotypes In diesen Konstanten wird festgelegt wie weit entfernt ein Event (mit ïd=...) eines audioTypes vom Spieler standardmäßig auftritt [Werbereicht 0 (beim spieler) bis 1(maximale Distanz des Fensters). Ist die Konstante -1 ist die Distance eines Events vom Typ audioType variabel und muss von der Gamelogik bestimmt werden.

Autor

Felix

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.6 audioStruct Strukturreferenz

Struktur für einzelne Audio Events AudioControl arbeitet Events von dieser Struktur ab. Jedes Audioevent hat eine eindeutige int "id", einen enum->integer Gruppen "type" und eine float "distance" und ordnet somit jedem Objekt einen Sound zu, wobei sich die Distanzinformation des Sounds ändern kann. Ein Distanzwert beträgt dabei minimal 0 und maximal 1 (größte Entfernung im Gamefenster). Die Standarddistanzwerte sind in "typedef struct audioDistance" für jeden AudioStruct "type" definiert.

`#include <definitions.h>`

Öffentliche Attribute

- int **id**
- audioType **type**
- float **distance**

5.6.1 Ausführliche Beschreibung

Struktur für einzelne Audio Events AudioControl arbeitet Events von dieser Struktur ab. Jedes Audioevent hat eine eindeutige int "id", einen enum->integer Gruppen "type" und eine float "distance" und ordnet somit jedem Objekt einen Sound zu, wobei sich die Distanzinformation des Sounds ändern kann. Ein Distanzwert beträgt dabei minimal 0 und maximal 1 (größte Entfernung im Gamefenster). Die Standarddistanzwerte sind in "typedef struct audioDistance" für jeden AudioStruct "type" definiert.

Alle in einem Step auftretetenden audioStruct's werden in einer std::list audioevents gesammelt (game.h) und über die Methode update() in jedem Step der Klasse Audiocontrol übergeben. Audiocontrol steuert den richtigen Abspieltyp jedes audioStruct. Nach jedem Step wird die Liste gelöscht und wieder neu mit audioStructs gefüllt. Audio Events welche in der GameLogik nur einmal auftreten, wie ein Powerup aufnehmen, werden mit einem Cooldown Timer zusätzlich länger an die Liste audioevents angehängt um ein weiteres Abspielen zu garantieren. Die Dauer des Cooldown Timers ist in "typedef struct audioCooldown" für jeden AudioStruct "type" definiert.

Ist ein Event mit zu erfolgender Audioausgabe vorhanden wird ein audioStruct mit Eventname und aktueller Distanz des Audio-Events vom Spieler zum Event erstellt. Dieses Audiostruct wird an die Liste audioevents mit allen im Step stattfinden audioStructs angehängt. Ist ein Objekt / Event nachwievor aktiv in der Szene wird das Struct im nächsten Step wieder an die Liste audioevents angehängt und die audioStruct "id" konstant gehalten. Ist ein Objekt nicht mehr in der Szene zu sehen, so muss kein audioStruct übergeben werden. Die audioStruct "id" diese Objekts wird im weiteren Spielverlauf nicht mehr verwendet.

Befindet sich z.B. ein Bier mit "id = ..." in der Szene, so ist der "type = scene_beer". In jedem Step muss in der AudioStruktur die "distance" des Biers zum Spieler aktualisiert werden und an die Liste audioevents angehängt werden. Verschwindet des Bierobjekt so wird das audioStruct nicht mehr übergeben und die "id" nicht mehr verwendet. Gibt

es mehrere Bierobjekte so wird das Struct mit Gruppen "type=scene_beer" mit verschiednen "id"'s an die Liste angehängt.

Läuft der Spieler im aktuellen Step so wird das audioStruct "player_walk" erstellt("distance" stets 0). Läuft er im nächsten Step nachwievor (hat also seine Position geändert) wird das Audiostruct wieder an die audioevents Liste angehängt. Läuft er nicht mehr wird es nicht mehr an die audioevents liste angehängt.

Ist gerade das Level 1 aktiv so wird in jedem Step ein audioStruct mit "ID=..." und "type=background_level1" an die Liste angehängt. Bei Background Musik ist "distance=0.5". Dies bewirkt dass sie leiser als Playersounds (distance = 0) abgespielt wird.

Autor

Felix Pfreundtner

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.7 collisionStruct Strukturreferenz

Struktur für die Events Enthält affectedObject als Objekt, aus dessen Sicht die Kollision berechnet wurde. affectedObject ist immer ein MovingObject, causingObject kann beides sein. Die Art und Richtung der Kollision werden mit gespeichert.

`#include <game.h>`

Öffentliche Attribute

- GameObject * **affectedObject**
- GameObject * **causingObject**
- enum collisionDirection **direction**

5.7.1 Ausführliche Beschreibung

Struktur für die Events Enthält affectedObject als Objekt, aus dessen Sicht die Kollision berechnet wurde. affectedObject ist immer ein MovingObject, causingObject kann beides sein. Die Art und Richtung der Kollision werden mit gespeichert.

Autor

Simon, Johann(15.6)

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.h

5.8 compareGameObjects Strukturreferenz

Vergleich zweier GameObjects bezüglich der X-Position Die Methode std::list::sort benötig ein struct mit einem boolschen Operator zur Sortierung. Diese Implementierung des Operators sortiert aufsteigend.

Öffentliche Methoden

- bool **operator()** (GameObject *objA, GameObject *objB)

5.8.1 Ausführliche Beschreibung

Vergleich zweier GameObjects bezüglich der X-Position Die Methode std::list::sort benötig ein struct mit einem boolschen Operator zur Sortierung. Diese Implementierung des Operators sortiert aufsteigend.

Parameter

1.Objekt	
2.Objekt	

Rückgabe

true, wenn 1.Objekt weiter links als 2.Objekt

Autor

Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.cpp

5.9 compareScores Strukturreferenz

Vergleich zweier Scores Der Vergleich findet über die Summe der Punkte in den einzelnen Kategorien statt. Der Operator im struct ist mit größer (>) programmiert, da die Liste absteigend sortiert werden soll.

Öffentliche Methoden

- bool **operator()** (scoreStruct scoreA, scoreStruct scoreB)

5.9.1 Ausführliche Beschreibung

Vergleich zweier Scores Der Vergleich findet über die Summe der Punkte in den einzelnen Kategorien statt. Der Operator im struct ist mit größer (>) programmiert, da die Liste absteigend sortiert werden soll.

Autor

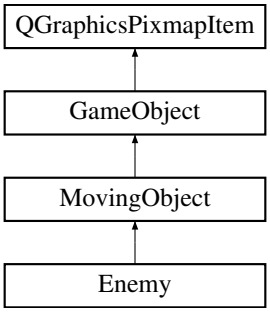
Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.cpp

5.10 Enemy Klassenreferenz

Klassendiagramm für Enemy:



Öffentliche Methoden

- Enemy (int posX, int posY, int speedX, objectType enemy)

  Konstruktor für ein Enemy-Objekt.
- int getHealth () const

  Enemy::getHealth Gibt Lebensstand zurück.
- void setHealth (int health)

  Enemy::setHealth Lebensstand wird gesetzt.
- bool **receiveDamage** (int damage)
- int getInflictedDamage () const

  Enemy::getInflictedDamage gibt Schaden zurück, den der gegner zufügt.
- int getFireCooldown () const

  Enemy::getFireCooldown.
- bool getDeath () const

  Enemy::getDeath Gibt an ob der Gegner Tot ist.
- void setDeath (bool death)

  Enemy::setDeath Zustand-TOT wird gesetzt.
- int getDeathCooldown () const

  Enemy::getDeathCooldown.
- virtual void update ()

  Enemy::update führt Bewegungen des Gegners aus.

Weitere Geerbte Elemente

5.10.1 Beschreibung der Konstruktoren und Destruktoren

5.10.1.1 Enemy::Enemy (int *posX*, int *posY*, int *speedX*, objectType *enemy*)

Konstruktor für ein Enemy-Objekt.

Class Enemy lastUpdate: update() 10.6 Johann

Parameter

posX	: X-Position
posY	: Y-Position
speedX	: Geschwindigkeit in X-Richtung

Noch zu erledigen Skalieren der Werte und fireCooldown erhöhen

5.10.2 Dokumentation der Elementfunktionen

5.10.2.1 bool Enemy::getDeath ( ) const

Enemy::getDeath Gibt an ob der Gegner Tot ist.

Rückgabe

: Zustand - TOT

5.10.2.2 int Enemy::getDeathCooldown ( ) const

Enemy::getDeathCooldown.

Rückgabe

deathCooldown

5.10.2.3 int Enemy::getFireCooldown ( ) const

Enemy::getFireCooldown.

Rückgabe

fireCooldown

5.10.2.4 int Enemy::getHealth ( ) const

Enemy::getHealth Gibt Lebensstand zurück.

Rückgabe

: Lebensstand

5.10.2.5 int Enemy::getInflictedDamage ( ) const

Enemy::getInflictedDamage gibt Schaden zurück, den der gegner zufügt.

Rückgabe

: Schaden

5.10.2.6 void Enemy::setDeath (bool *death*)

Enemy::setDeath Zustand-TOT wird gesetzt.

Parameter

death	: Zustand-TOT

5.10.2.7 void Enemy::setHealth (int *health*)

Enemy::setHealth Lebensstand wird gesetzt.

Parameter

health	: Lebensstand

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

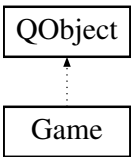
- Wiesn-Run/src/enemy.h
- Wiesn-Run/src/enemy.cpp

5.11 Game Klassenreferenz

Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels. Innerhalb der main.cpp wird eine Instanz dieser Klasse angelegt, aus der heraus das gesamte Spiel läuft. Die einzelnen Methoden werden in der game.cpp jeweils erklärt.

`#include <game.h>`

Klassendiagramm für Game:



Öffentliche Methoden

- Game (int argc, char ∗argv[])

  Konstruktor und Destruktor.
- int step ()

  Game-Loop Diese Funktion wird von timerEvent() aufgerufen und ist für den kompletten Ablauf des Spiels verantwortlich. grober Ablauf: LOOP:
- int **run** (QApplication &app)
- int start ()

  Starten der Applikation.
- void setState (enum gameState newState)

  Hilfsfunktion.

Öffentliche Attribute

- struct stateStruct **gameStats**
- std::list< struct collisionStruct > **collisionsToHandle**

Geschützte Methoden

- void timerEvent (QTimerEvent ∗event)

  wird regelmäßig aufgerufen event muss drinstehen, damit der Timer die Funktion aufruft

5.11.1 Ausführliche Beschreibung

Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels. Innerhalb der main.cpp wird eine Instanz dieser Klasse angelegt, aus der heraus das gesamte Spiel läuft. Die einzelnen Methoden werden in der game.cpp jeweils erklärt.

funtion handleCollisions hinzugefügt

Autor

Simon, Johann, Felix

5.11.2 Beschreibung der Konstruktoren und Destruktoren

5.11.2.1 Game::Game (int *argc,* char ∗ *argv[]*)

Konstruktor und Destruktor.

Konstruktor Initialisiert den appPointer.

Parameter

argc	
argv	

Autor

Rupert

Initialisiert den appPointer mit der QApplication

5.11.3 Dokumentation der Elementfunktionen

5.11.3.1 void Game::setState (enum gameState *newState*)

Hilfsfunktion.

setzt den Spielstatus

Parameter

newState	

Autor

Rupert

5.11.3.2 int Game::start ( )

Starten der Applikation.

Die Startfunktion, erstellt Fenster und Menüs, wird von main() aufgerufen Grafik und Input (Flo,Felix): Erstelle Q-Application app mit QGraphicsView Widget window (Eventfilter installiert) und Zeiger input auf Input Objekt. Um Funktionen der Tastatur Eingabe entwickeln zu können ist ein Qt Widget Fenster nötig. Auf dem Widget wird ein Eventfilter installiert welcher kontinuierlich Tastatureingaben mitloggt. Die Eingaben werden in dem Objekt der Input Klasse gespeichert und können über getKeyactions() abgerufen werden.

Logik (Rupert): Außerdem wird ein Timer gestartet, der in jedem Intervall timerEvent(...) aufruft, wo dann step() aufgerufen wird. Das ist dann unsere Game-Loop. Der Timer funktioniert auch bei 5ms Intervall noch genau. Menüs (Rupert): Alle Menüs werden angelegt

gameState wird auf gameMenuStart gesetzt, dh das Spiel startet im Startmenü

Rückgabe

Rückgabewert von app.exec()

Autor

Rupert, Felix

Erstelle Audiocontrol Objekt zum Einlesen der Audiodatein und speichern der Ausgabeparameter

Erstelle einen neuen Thread portaudiothread. Initialisiere dort PortAudio und beginne eine Audioausgabe zu erzeugen.

Installiere Event Filter zum Überwachen der Keyboard Eingabe und des QT Fenster Schließ-Buttons (x)

5.11.3.3 int Game::step ( )

Game-Loop Diese Funktion wird von timerEvent() aufgerufen und ist für den kompletten Ablauf des Spiels verantwortlich. grober Ablauf: LOOP:

- Timer starten
- Neue Objekte zur Welt hinzufügen
- alte Objekte löschen
- Input auslesesn
- Bewegungen berechnen
- Kollisionskontrolle
- Bewegungen korrigieren
- Events behandeln (Treffer..)
- Grafik rendern und ausgeben
- Audio ausgeben
- verbleibende Zeit im Slot berechnen (Timer auslesen)
- entsprechend warten goto LOOP

Rückgabe

0 bei fehlerfreiem Beenden

Autor

Rupert, Felix

Tasten abfragen

Noch zu erledigen Erfolgreich Schriftzug einfügen

Noch zu erledigen GameOver schriftzug einfügen

delete List audioStruct elements in list and fill it in the next step again

5.11.3.4 void Game::timerEvent (QTimerEvent * *event*) [protected]

wird regelmäßig aufgerufen event muss drinstehen, damit der Timer die Funktion aufruft

Parameter

event	

Autor

Rupert, Felix

falls QT Schließ-Button (x) gedrückt wurde (Game::eventFilter()) oder im Hautpmenü exit ausgewählt wurde (Game::step()) beende Spiel

beende Audio Thread und lösche Variablen

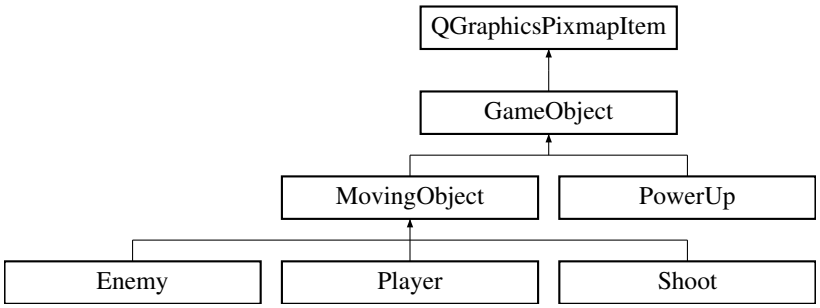
Beende Spiel

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/game.h
- Wiesn-Run/src/game.cpp

5.12 GameObject Klassenreferenz

Klassendiagramm für GameObject:



Öffentliche Methoden

- GameObject (int posX, int posY, int length, int height, objectType type)

  GameObject::GameObject Konstruktor.
- **GameObject** (int posX, int posY, objectType type)
- int **getPosX** () const
- int **getPosY** () const
- int **getLength** () const
- int **getHeight** () const
- objectType **getType** () const
- void **setAudioID** (int audioID)
- int **getAudioID** () const

Geschützte Attribute

- int **posX**
- int **posY**

5.12.1 Beschreibung der Konstruktoren und Destruktoren

5.12.1.1 GameObject::GameObject (int *posX,* int *posY,* int *length,* int *height,* objectType *type*)

GameObject::GameObject Konstruktor.

Parameter

length	: Länge
height	: Höhe
type	: Typ
posX	: X-Position
posY	: Y-Position
colType	: Kollisionstyp

Autor

Johann

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/gameobject.h
- Wiesn-Run/src/gameobject.cpp

5.13 Input Klassenreferenz

Input-Klasse Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben. Eine Instanz dieser Klasse wir innerhalb der game.h angelegt. Die einzelnen Methoden werden in der input.cpp erklärt.

```
#include <input.h>
```

Öffentliche Typen

- enum **Keyaction** {
  noKeyaction, **Right**, **Up**, **Down**,
  Jump_Right, **Shoot**, **Exit**, **Enter** }
- enum **Keyletter** {
  noKeyletter, **a** = (int)'a', **b** = (int)'b', **c** = (int)'c',
  d = (int)'d', **e** = (int)'e', **f** = (int)'f', **g** = (int)'g',
  h = (int)'h', **i** = (int)'i', **j** = (int)'f', **k** = (int)'k',
  l = (int)'l', **m** = (int)'m', **n** = (int)'n', **o** = (int)'o',
  p = (int)'p', **q** = (int)'q', **r** = (int)'r', **s** = (int)'s',
  t = (int)'t', **u** = (int)'u', **v** = (int)'v', **w** = (int)'w',
  x = (int)'x', **y** = (int)'y', **z** = (int)'z', **A** = (int)'A',
  B = (int)'B', **C** = (int)'C', **D** = (int)'D', **E** = (int)'E',
  F = (int)'F', **G** = (int)'G', **H** = (int)'G', **I** = (int)'I',
  J = (int)'J', **K** = (int)'K', **L** = (int)'L', **M** = (int)'M',
  N = (int)'N', **O** = (int)'O', **P** = (int)'P', **Q** = (int)'Q',
  R = (int)'R', **S** = (int)'S', **T** = (int)'T', **U** = (int)'U',
  V = (int)'V', **W** = (int)'W', **X** = (int)'X', **Y** = (int)'Y',
  Z = (int)'Z', **Backspace** = (int)'\b' }

Öffentliche Methoden

- Input ()

  Input::Input Konstruktor instanziert ein Objekt der Klasse Input.
- ~Input ()

  Input::~Input Destruktor löscht ein Objekt der Klasse Input.
- void evaluatekeyEvent (QEvent *event)

Input::evaluatekeyEvent Nach Aufruf über Game::eventFilter wertet evaluatekeyEvent alle im Momment gleichzeitig gepressten Tastatur Eingaben aus und speichert die zugehörigen enum ids in der Instanzvariable keyevents. wird eine Taste nicht mehr gedrück wird die enum id in keyevents gelöscht wird eine Taste neu gedrückt wird die enum id in keyevents hinzugefügt.

- QSet< int > getKeyactions ()

  Input::getKeyactions getKeyactions gibt bei Aufruf das QSet keyactions zurück, welches alle im Moment gedrückten Spielaktionen als Enum beinhaltet. Jeder Tastaturkombination wird eine Integer ID zugeordnet welche im QSet keyactions gespeichert ist. Über die Enumeration Input::Keyaction ist jeder Spielbefehl mit dem zugehörigen Indize in keyactions verknüft. Möchte man nun bespielsweise abfragen ob der Spieler im Moment schießt so überprüft man: input->getKeyactions().contains(Input::Keyaction::Shoot) == True.

- std::set< char > getKeyletters ()

  Input::getKeyletters getKeyletters gibt bei Aufruf das QSet keyletters zurück, welches alle im Moment gedrückten Buchstaben als Enum beinhaltet. Jeder Buchststaben Taste wird ein String Buchstaben zugeordnet, welcher im QSet keyletters gespeichert ist. Über die Enumeration Input::Keyletter ist jeder Buchstabe mit dem zugehörigen Indize in keyletters verknüft. Möchte man nun bespielsweise abfragen ob der Spieler im Moment die "a" Taste drückt so überprüft man: input->getKeyletters().find(Input::Keyletter::a) != getKeyletters().end(). Möchte man abfragen ob der Spieler im Moment die "A" Taste drückt so überprüft man: input->getKeyletters().find(Input::Keyletter::A) != getKeyletters().end(). Ist die Taste gedrückt so kann aus dem Enum Keyletter über eine Typenumwandlung der Char berechnet werden: 'a' = (char)Keyletter::a.

- Keyaction getLastKeyaction ()

  Input::getLastKeyaction Gibt letzte gedrücke Spielaktion als Enum Keyaction zurück und setzt die Variable lastKeyaction auf noKeyaction. Wird für die Menüführung gebraucht, da ein dauerhaftes Auswerten der Tasten dort zu Sprüngen beim Auswählen der Menü Einträge führt.

- Keyletter getLastKeyletter ()

  Input::getLastKeyletter Gibt letzten gedrücken Buchstaben als enum Keyletter zurück und setzt die Variable lastKeyletter auf noKeyletter. Wurde eine Taste gedrückt (lastKeyletter_return != noKeyletter) so kann aus dem Enum Keyletter über eine Typenumwandlung der zugehörige Char berechnet werden: a = (char)lastKeyletter_return Verwendung findet die Funktion bei der Eingabe des Highscore Namens.

5.13.1 Ausführliche Beschreibung

Input-Klasse Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben. Eine Instanz dieser Klasse wir innerhalb der game.h angelegt. Die einzelnen Methoden werden in der input.cpp erklärt.

Autor

Felix Pfreundtner

5.13.2 Beschreibung der Konstruktoren und Destruktoren

5.13.2.1 Input::Input ( )

Input::Input Konstruktor instanziert ein Objekt der Klasse Input.

Autor

Felix Pfreundtner

5.13.2.2 Input::∼Input ( )

Input::∼Input Destruktor löscht ein Objekt der Klasse Input.

Autor

Felix Pfreundtner

5.13.3 Dokumentation der Elementfunktionen

5.13.3.1 void Input::evaluatekeyEvent (QEvent * *event*)

Input::evaluatekeyEvent Nach Aufruf über Game::eventFilter wertet evaluatekeyEvent alle im Momment gleichzeitig gepressten Tastatur Eingaben aus und speichert die zugehörigen enum ids in der Instanzvariable keyevents. wird eine Taste nicht mehr gedrück wird die enum id in keyevents gelöscht wird eine Taste neu gedrückt wird die enum id in keyevents hinzugefügt.

Parameter

QEvent	*event

Rückgabe

Boolean

Autor

Felix Pfreundtner

5.13.3.2 QSet< int > Input::getKeyactions ( )

Input::getKeyactions getKeyactions gibt bei Aufruf das QSet keyactions zurück, welches alle im Moment gedrückten Spielaktionen als Enum beinhaltet. Jeder Tastaturkombination wird eine Integer ID zugeordnet welche im QSet keyactions gespeichert ist. Über die Enumeration Input::Keyaction ist jeder Spielbefehl mit dem zugehörigen Indize in keyactions verknüft. Möchte man nun bespielsweise abfragen ob der Spieler im Moment schießt so überprüft man: input->getKeyactions().contains(Input::Keyaction::Shoot) == True.

Rückgabe

QSet Instanzvariable keyactions

Autor

Felix Pfreundtner

5.13.3.3 std::set< char > Input::getKeyletters ( )

Input::getKeyletters getKeyletters gibt bei Aufruf das QSet keyletters zurück, welches alle im Moment gedrückten Buchstaben als Enum beinhaltet. Jeder Buchststaben Taste wird ein String Buchstaben zugeordnet, welcher im QSet keyletters gespeichert ist. Über die Enumeration Input::Keyletter ist jeder Buchstabe mit dem zugehörigen Indize in keyletters verknüft. Möchte man nun bespielsweise abfragen ob der Spieler im Moment die "a" Taste drückt so überprüft man: input->getKeyletters().find(Input::Keyletter::a) != getKeyletters().end(). Möchte man abfragen ob der Spieler im Moment die "A" Taste drückt so überprüft man: input->getKeyletters().find(Input::Keyletter::A) != getKeyletters().end(). Ist die Taste gedrückt so kann aus dem Enum Keyletter über eine Typenumwandlung der Char berechnet werden: 'a' = (char)Keyletter::a.

Rückgabe

Std::Set Instanzvariable keyactions

Autor

Felix Pfreundtner

5.13.3.4 Input::Keyaction Input::getLastKeyaction ( )

Input::getLastKeyaction Gibt letzte gedrücke Spielaktion als Enum Keyaction zurück und setzt die Variable lastKeyaction auf noKeyaction. Wird für die Menüführung gebraucht, da ein dauerhaftes Auswerten der Tasten dort zu Sprüngen beim Auswählen der Menü Einträge führt.

Rückgabe

Enum Keyaction

Autor

Rupert, Felix

5.13.3.5 Input::Keyletter Input::getLastKeyletter ( )

Input::getLastKeyletter Gibt letzten gedrücken Buchstaben als enum Keyletter zurück und setzt die Variable lastKeyletter auf noKeyletter. Wurde eine Taste gedrückt (lastKeyletter_return != noKeyletter) so kann aus dem Enum Keyletter über eine Typenumwandlung der zugehörige Char berechnet werden: a = (char)lastKeyletter_return Verwendung findet die Funktion bei der Eingabe des Highscore Namens.

Rückgabe

Enum Keyletter

Autor

Felix

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/input.h
- Wiesn-Run/src/input.cpp

5.14 Menu Klassenreferenz

Menü-Klasse eine Instanz repräsentiert ein Menü mit diesen Funktionen:

```
#include <menu.h>
```

Klassen

- struct menuEntry

  Struct zur Beschreibung eines Menü-Eintrags.

Öffentliche Typen

- enum menuSelectionChange { **up**, **down** }

  wird von der Menu-Klasse zur Auswahl-Änderung benötigt
- enum menuType { **normal**, **highscore** }

  für verschiedene Menü-Typen (für Background-Musik)

Öffentliche Methoden

- Menu (std::string ∗menuTitle, menuType type=normal)

  Menu-Konstruktor.
- void clear ()

  löscht alle Einträge bis auf den ersten
- menuType getType ()

  gibt den Typ zurück
- std::string ∗ getTitle ()

  gibt den Titel zurück
- int displayInit ()

  Initialisiert das angezeigt Menü
- int displayUpdate ()

  Aktualisiert das angezeigt Menü
- int addEntry (std::string name, int id, bool clickable=false, gameState stateOnClick=(gameState) NULL)

  Neuen Eintrag hinzufügen (evtl private -> Einträge nur im Konstruktor erstellen -> unterschiedlich viele Argumente)
- int changeSelection (menuSelectionChange changeType)

  wird nach Tastendruck aufgerufen
- Menu::menuEntry ∗ getSelection ()

  Zeiger auf aktuelle gewählten Menüeintrag, sollte nach Enter aufgerufen werden.
- Menu::menuEntry ∗ getEntry (int position)

  Gibt Menü-Eintrag an der entsprechenden Position zurück.

Öffentliche Attribute

- QGraphicsPixmapItem ∗ background

  Zeiger auf die Menü-Scene und das Menü-Hintergrundbild.
- QGraphicsScene ∗ **menuScene**

5.14.1 Ausführliche Beschreibung

Menü-Klasse eine Instanz repräsentiert ein Menü mit diesen Funktionen:

- Einträge hinzufügen
- aktuelle Auswahl ändern (nach Tastendruck)
- anzeigen

5.14.2 Beschreibung der Konstruktoren und Destruktoren

5.14.2.1 Menu::Menu (std::string ∗ *menuTitle,* menuType *type* = `normal`)

Menu-Konstruktor.

Parameter

Zeiger	auf String mit Menu-Titel

Autor

Rupert

5.14.3 Dokumentation der Elementfunktionen

5.14.3.1 int Menu::addEntry (std::string *name,* int *id,* bool *clickable =* `false`*,* gameState *stateOnClick =* `(gameState)NULL`)

Neuen Eintrag hinzufügen (evtl private -> Einträge nur im Konstruktor erstellen -> unterschiedlich viele Argumente)

Neuen Eintrag hinzufügen.

Parameter

name	String, der angezeigt wird
id	zur eindeutigen Identifizierung, kann zB aus enum gecastet werden

Rückgabe

0 bei Erfolg

Autor

Rupert

5.14.3.2 int Menu::changeSelection (menuSelectionChange *changeType*)

wird nach Tastendruck aufgerufen

Parameter

changeType	entweder up oder down

Rückgabe

0 bei Erfolg, -1 wenn kein klickbarer Eintrag vorhanden

Autor

Rupert

5.14.3.3 void Menu::clear ( )

löscht alle Einträge bis auf den ersten

entfernt alle Einträge aus dem Menü Titel wird danach wieder hinzugefügt wird für Statistik und Highscore benötigt

5.14.3.4 int Menu::displayInit ( )

Initialisiert das angezeigt Menü

Initialisiert das sichtbare Menü , muss immer nach anlegen der Menü Entrys aufgerufen werden.

Rückgabe

0 bei Erfolg

Autor

Flo

5.14.3.5 int Menu::displayUpdate ( )

Aktualisiert das angezeigt Menü

aktualisiert das sichtbare Menü

Rückgabe

0 bei Erfolg

Autor

Flo

5.14.3.6 struct Menu::menuEntry * Menu::getEntry (int *position*)

Gibt Menü-Eintrag an der entsprechenden Position zurück.

gibt Eintrag an der gesuchten Position zurück

Parameter

position	

Rückgabe

Zeiger auf gefundenen Eintrag, sonst NULL

Autor

Rupert

Schleife startet beim ersten Element und geht bis zum letzen Element durch

5.14.3.7 struct Menu::menuEntry * Menu::getSelection ( )

Zeiger auf aktuelle gewählten Menüeintrag, sollte nach Enter aufgerufen werden.

gibt den gewählten Eintrag zurück sollte nach Enter aufgerufen werden

Rückgabe

Zeiger auf menuEntry des aktuellen Eintrags, NULL bei Fehler

Autor

Rupert

5.14.3.8 std::string * Menu::getTitle ( )

gibt den Titel zurück

gibt den Menü-Titel zurück

Rückgabe

Zeiger auf String

Autor

Rupert

5.14.3.9 Menu::menuType Menu::getType ( )

gibt den Typ zurück

gibt den Menü-Typ zurück

Rückgabe

enum menuType

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/menu.h
- Wiesn-Run/src/menu.cpp

5.15 Menu::menuEntry Strukturreferenz

Struct zur Beschreibung eines Menü-Eintrags.

`#include <menu.h>`

Öffentliche Attribute

- std::string **name**
- int **id**
- int **position**
- bool **isClickable**
- bool **menuOnEnter**
- gameState **stateOnClick**
- QGraphicsTextItem **showEntry**

5.15.1 Ausführliche Beschreibung

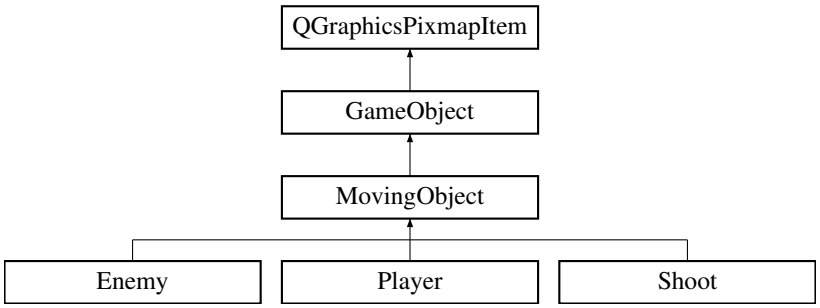
Struct zur Beschreibung eines Menü-Eintrags.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/menu.h

5.16 MovingObject Klassenreferenz

Klassendiagramm für MovingObject:



Öffentliche Methoden

- **MovingObject** (int posX, int posY, objectType type, int speedX, int speedY)
- void **setPosX** (int posX)
- void **setPosY** (int posY)
- int **getSpeedX** () const
- int **getSpeedY** () const
- void **setSpeedX** (int speedX)
- void **setSpeedY** (int speedY)
- void **setFramesDirection** (int framesDirection)
- int **getFramesDirection** ()
- virtual void **update** ()=0
- void flipHorizontal ()

  spiegelt Grafiken an der Y-Achse
- void **swapImage** ()

Geschützte Methoden

- void updatePosition ()

  überschreibt die X und Y Position gemäß SpeedXY.

Weitere Geerbte Elemente

5.16.1 Dokumentation der Elementfunktionen

5.16.1.1 void MovingObject::flipHorizontal ( )

spiegelt Grafiken an der Y-Achse

Autor

Flo

5.16.1.2 void MovingObject::updatePosition ( ) [protected]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/movingobject.h
- Wiesn-Run/src/movingobject.cpp

5.17 AudioControl::paData Strukturreferenz

Öffentliche Attribute

- float **left_phase**
- float **right_phase**

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/audiocontrol.h

5.18 PaDeviceInfo Strukturreferenz

`#include <portaudio.h>`

Öffentliche Attribute

- int **structVersion**
- const char * **name**
- PaHostApiIndex hostApi
- int **maxInputChannels**
- int **maxOutputChannels**
- PaTime defaultLowInputLatency
- PaTime **defaultLowOutputLatency**
- PaTime defaultHighInputLatency
- PaTime **defaultHighOutputLatency**
- double **defaultSampleRate**

5.18.1 Ausführliche Beschreibung

A structure providing information and capabilities of PortAudio devices. Devices may support input, output or both input and output.

5.18.2 Dokumentation der Datenelemente

5.18.2.1 PaTime PaDeviceInfo::defaultHighInputLatency

Default latency values for robust non-interactive applications (eg. playing sound files).

5.18.2.2 PaTime PaDeviceInfo::defaultLowInputLatency

Default latency values for interactive performance.

5.18.2.3 PaHostApiIndex PaDeviceInfo::hostApi

note this is a host API index, not a type id

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/portaudio.h

5.19 PaHostApiInfo Strukturreferenz

`#include <portaudio.h>`

Öffentliche Attribute

- int structVersion
- PaHostApiTypeId type
- const char * name
- int deviceCount
- PaDeviceIndex defaultInputDevice
- PaDeviceIndex defaultOutputDevice

5.19.1 Ausführliche Beschreibung

A structure containing information about a particular host API.

5.19.2 Dokumentation der Datenelemente

5.19.2.1 PaDeviceIndex PaHostApiInfo::defaultInputDevice

The default input device for this host API. The value will be a device index ranging from 0 to (Pa_GetDeviceCount()-1), or paNoDevice if no default input device is available.

5.19.2.2 PaDeviceIndex PaHostApiInfo::defaultOutputDevice

The default output device for this host API. The value will be a device index ranging from 0 to (Pa_GetDeviceCount()-1), or paNoDevice if no default output device is available.

5.19.2.3 int PaHostApiInfo::deviceCount

The number of devices belonging to this host API. This field may be used in conjunction with Pa_HostApiDeviceIndexToDeviceIndex() to enumerate all devices for this host API.

Siehe auch

Pa_HostApiDeviceIndexToDeviceIndex

5.19.2.4 const char* PaHostApiInfo::name

A textual description of the host API for display on user interfaces.

5.19.2.5 int PaHostApiInfo::structVersion

this is struct version 1

5.19.2.6 PaHostApiTypeId PaHostApiInfo::type

The well known unique identifier of this host API

Siehe auch

PaHostApiTypeId

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/portaudio.h

5.20 PaHostErrorInfo Strukturreferenz

`#include <portaudio.h>`

Öffentliche Attribute

- PaHostApiTypeId hostApiType
- long errorCode
- const char * errorText

5.20.1 Ausführliche Beschreibung

Structure used to return information about a host error condition.

5.20.2 Dokumentation der Datenelemente

5.20.2.1 long PaHostErrorInfo::errorCode

the error code returned

5.20.2.2 const char* PaHostErrorInfo::errorText

a textual description of the error if available, otherwise a zero-length string

5.20.2.3 PaHostApiTypeId PaHostErrorInfo::hostApiType

the host API which returned the error code

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/portaudio.h

5.21 PaStreamCallbackTimeInfo Strukturreferenz

`#include <portaudio.h>`

Öffentliche Attribute

- PaTime inputBufferAdcTime
- PaTime currentTime
- PaTime outputBufferDacTime

5.21.1 Ausführliche Beschreibung

Timing information for the buffers passed to the stream callback.

Time values are expressed in seconds and are synchronised with the time base used by Pa_GetStreamTime() for the associated stream.

Siehe auch

PaStreamCallback, Pa_GetStreamTime

5.21.2 Dokumentation der Datenelemente

5.21.2.1 PaTime PaStreamCallbackTimeInfo::currentTime

The time when the stream callback was invoked

5.21.2.2 PaTime PaStreamCallbackTimeInfo::inputBufferAdcTime

The time when the first sample of the input buffer was captured at the ADC input

5.21.2.3 PaTime PaStreamCallbackTimeInfo::outputBufferDacTime

The time when the first sample of the output buffer will output the DAC

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/portaudio.h

5.22 PaStreamInfo Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- int structVersion
- PaTime inputLatency
- PaTime outputLatency
- double sampleRate

5.22.1 Ausführliche Beschreibung

A structure containing unchanging information about an open stream.

Siehe auch

Pa_GetStreamInfo

5.22.2 Dokumentation der Datenelemente

5.22.2.1 PaTime PaStreamInfo::inputLatency

The input latency of the stream in seconds. This value provides the most accurate estimate of input latency available to the implementation. It may differ significantly from the suggestedLatency value passed to Pa_OpenStream(). The value of this field will be zero (0.) for output-only streams.

Siehe auch

PaTime

5.22.2.2 PaTime PaStreamInfo::outputLatency

The output latency of the stream in seconds. This value provides the most accurate estimate of output latency available to the implementation. It may differ significantly from the suggestedLatency value passed to Pa_OpenStream(). The value of this field will be zero (0.) for input-only streams.

Siehe auch

PaTime

5.22.2.3 double PaStreamInfo::sampleRate

The sample rate of the stream in Hertz (samples per second). In cases where the hardware sample rate is inaccurate and PortAudio is aware of it, the value of this field may be different from the sampleRate parameter passed to Pa_OpenStream(). If information about the actual hardware sample rate is not available, this field will have the same value as the sampleRate parameter passed to Pa_OpenStream().

5.22.2.4 int PaStreamInfo::structVersion

this is struct version 1

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/portaudio.h

5.23 PaStreamParameters Strukturreferenz

```
#include <portaudio.h>
```

Öffentliche Attribute

- PaDeviceIndex device
- int channelCount
- PaSampleFormat sampleFormat
- PaTime suggestedLatency
- void * hostApiSpecificStreamInfo

5.23.1 Ausführliche Beschreibung

Parameters for one direction (input or output) of a stream.

5.23.2 Dokumentation der Datenelemente

5.23.2.1 int PaStreamParameters::channelCount

The number of channels of sound to be delivered to the stream callback or accessed by Pa_ReadStream() or Pa_WriteStream(). It can range from 1 to the value of maxInputChannels in the PaDeviceInfo record for the device specified by the device parameter.

5.23.2.2 PaDeviceIndex PaStreamParameters::device

A valid device index in the range 0 to (Pa_GetDeviceCount()-1) specifying the device to be used or the special constant paUseHostApiSpecificDeviceSpecification which indicates that the actual device(s) to use are specified in hostApiSpecificStreamInfo. This field must not be set to paNoDevice.

5.23.2.3 void∗ PaStreamParameters::hostApiSpecificStreamInfo

An optional pointer to a host api specific data structure containing additional information for device setup and/or stream processing. hostApiSpecificStreamInfo is never required for correct operation, if not used it should be set to NULL.

5.23.2.4 PaSampleFormat PaStreamParameters::sampleFormat

The sample format of the buffer provided to the stream callback, a_ReadStream() or Pa_WriteStream(). It may be any of the formats described by the PaSampleFormat enumeration.

5.23.2.5 PaTime PaStreamParameters::suggestedLatency

The desired latency in seconds. Where practical, implementations should configure their latency based on these parameters, otherwise they may choose the closest viable latency instead. Unless the suggested latency is greater than the absolute upper limit for the device implementations should round the suggestedLatency up to the next practical value - ie to provide an equal or higher latency than suggestedLatency wherever possible. Actual latency values for an open stream may be retrieved using the inputLatency and outputLatency fields of the PaStreamInfo structure returned by Pa_GetStreamInfo().

Siehe auch

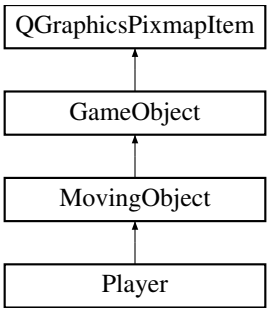
default∗Latency in PaDeviceInfo, ∗Latency in PaStreamInfo

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/portaudio.h

5.24 Player Klassenreferenz

Klassendiagramm für Player:



Öffentliche Methoden

- Player (int posX, int posY, int speedX)
- int getHealth () const

  Player::getHealth Gibt aktuellen Lebensstand zurück.
- void setHealth (int health)

  Player::setHealth Lebensstand des Spielers wird gesetzt.
- void **increaseHealth** (int health)
- bool receiveDamage (int damage)

  Player::receiveDamage.
- int getAlcoholLevel () const

  Player::getAlcoholLevel Gibt den Pegel des Spielers zurück.
- void increaseAlcoholLevel (int additionalAlcohol)

  Player::increaseAlcoholLevel AlkoholPegel wird verändert. Durch einen negativen Wert im Argument wird der Pegel gesenkt.
- void decreaseAlcoholLevel (int decreaseLevel)

  Player::decreaseAlcoholLevel verringert den Pegel des Spielers.
- int getAmmunatiuon () const

  Player::getAmmunatiuon Gibt verbleibende Munition zurück.
- void increaseAmmunation (int ammunationBonus)

  Player::increaseAmmunation erhöht die verbleibende Munition des Spielers um 1.
- void decreaseAmmunation ()

  Player::decreaseAmmunation verringert die verbleibende Munition des Spielers um 1.
- void **setFireCooldown** ()
- int getFireCooldown ()

  Player::getFireCooldown.
- int getInflictedDamage () const

  Player::getInflictedDamage.
- int getImmunityCooldown () const

  Player::getImmunityCooldown.
- void setImmunityCooldown (int remainingTime)

  Wird nicht benutzt 23.6.
- void startJump ()

  beginnt einen Sprung Nur wenn der Spieler sich nicht in der Luft befindet
- bool inJump () const

  gibt den Sprung-Zustande des Spielers zurück
- void resetJumpState ()

  Gibt an dass der Spieler nicht in einem Sprung ist.
- void abortJump ()

  Methode wird aufgerufen, wenn der Spieler bei einem Sprung mit einem Hinderniss zusammengestoßen ist.
- int getEnemiesKilled ()

Player::getEnemiesKilled Übergibt die Zahl getöteter Gegner.

- void increaseEnemiesKilled ()

  Perhöht die Anzahl der getöteten Gegner um 1.

- virtual void update ()

  Player::update führt die Bewegung des Spielers aus (über updatePosition) und verringert Cooldown-Variable.

Weitere Geerbte Elemente

5.24.1 Beschreibung der Konstruktoren und Destruktoren

5.24.1.1 Player::Player (int *posX,* int *posY,* int *speedX*)

Class Player lastUpdate: update() 10.6 Johann

5.24.2 Dokumentation der Elementfunktionen

5.24.2.1 void Player::decreaseAlcoholLevel (int *decreaseLevel*)

Player::decreaseAlcoholLevel verringert den Pegel des Spielers.

Noch zu erledigen Überflüssig, da nie aufgerufen. Auch wenn der Name es nicht vermuten lässt: increaseAlcoholLevel kann den Level auch verringern und wird benutzt.

Parameter

decreaseLevel	Wert um den der Pegel verringert wird

5.24.2.2 int Player::getAlcoholLevel ( ) const

Player::getAlcoholLevel Gibt den Pegel des Spielers zurück.

Rückgabe

: Alkoholpegel

5.24.2.3 int Player::getAmmunatiuon ( ) const

Player::getAmmunatiuon Gibt verbleibende Munition zurück.

Rückgabe

: verbleibende Munition

5.24.2.4 int Player::getFireCooldown ( )

Player::getFireCooldown.

Rückgabe

verbleibende Zeit bs nächster schuss möglich ist

5.24.2.5 int Player::getHealth ( ) const

Player::getHealth Gibt aktuellen Lebensstand zurück.

Rückgabe

: Lebensstand

5.24.2.6 int Player::getImmunityCooldown ( ) const

Player::getImmunityCooldown.

Rückgabe

5.24.2.7 int Player::getInflictedDamage ( ) const

Player::getInflictedDamage.

Rückgabe

Schaden den der Spieler zufügt

5.24.2.8 void Player::increaseAlcoholLevel (int *additionalAlcohol*)

Player::increaseAlcoholLevel AlkoholPegel wird verändert. Durch einen negativen Wert im Argument wird der Pegel gesenkt.

Parameter

additionalAlcohol	Wert um den erhöht wird

5.24.2.9 bool Player::inJump ( ) const

gibt den Sprung-Zustande des Spielers zurück

Rückgabe

5.24.2.10 bool Player::receiveDamage (int *damage*)

Player::receiveDamage.

Rückgabe

Lebenszustand des Spielers: true = tot

5.24.2.11 void Player::setHealth (int *health*)

Player::setHealth Lebensstand des Spielers wird gesetzt.

Parameter

health	Lebensstand auf den der Spieler gesetzt wird

5.24.2.12 void Player::setImmunityCooldown (int *remainingTime*)

Wird nicht benutzt 23.6.

Player::setImmunityCooldown Zahl der Frames für Unverwundbarkeit wird gesetzt.

Parameter

immunity-Cooldown	Zahl der Frames

5.24.2.13 void Player::update ( ) [virtual]

Player::update führt die Bewegung des Spielers aus (über updatePosition) und verringert Cooldown-Variable.

Autor

Johann

Implementiert MovingObject.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/player.h
- Wiesn-Run/src/player.cpp

5.25 AudioControl::playStruct Strukturreferenz

Öffentliche Attribute

- int id

  id des playStruct
- audioType type

  type des playStruct
- float volume

  Lautstärke des playStruct.
- bool playnext

  variable welche angibt ob sound im moment abgespielt wird
- Audio * audioobject

  Zeiger auf das (Audio-)object des playStruct, welches Eventgruppe "type" zugeordnet ist.
- int position

  aktuelle Abspielposition in Audiobjekt in Samples (Beginn des Abspielblockes mit Länge 1024 Samples

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

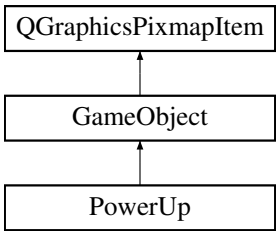
- Wiesn-Run/src/audiocontrol.h

5.26 PowerUp Klassenreferenz

Klasse für Power-Ups.

`#include <powerup.h>`

Klassendiagramm für PowerUp:



Öffentliche Methoden

- PowerUp (int posX, int posY, int healthBonus, int alcoholLevelBonus, int ammunationBonus, int immunityCooldownBonus, powerUpType type)

  Konstruktor.
- ~PowerUp ()

  Destruktor.
- int getHealthBonus () const

  Get-Methoden für die Objekteigenschaften.
- int getAlcoholLevelBonus () const

  Gibt den Bonus auf Alcohollevel zurück.
- int getAmmunationBonus () const

  Gibt den Bonus auf Munnition zurück.
- int getImmunityCooldownBonus () const

  Gibt den Bonus auf Immunität zurück.
- powerUpType getPowerUPType () const

  PowerUp::getPowerUPType.

Weitere Geerbte Elemente

5.26.1 Ausführliche Beschreibung

Klasse für Power-Ups.

Autor

Johann

5.26.2 Beschreibung der Konstruktoren und Destruktoren

5.26.2.1 PowerUp::PowerUp (int *posX,* int *posY,* int *healthBonus,* int *alcoholLevelBonus,* int *ammunationBonus,* int *immunityCooldownBonus,* powerUpType *type*)

Konstruktor.

Parameter

posX	
posY	
length	
height	
healthBonus	
alcoholLevel-Bonus	
ammunation-Bonus	
immunity-CooldownBonus	

Autor

Johann

5.26.2.2 PowerUp::~PowerUp ( )

Destruktor.

Autor

Johann

5.26.3 Dokumentation der Elementfunktionen

5.26.3.1 int PowerUp::getAlcoholLevelBonus ( ) const

Gibt den Bonus auf Alcohollevel zurück.

Autor

Johann

5.26.3.2 int PowerUp::getAmmunationBonus ( ) const

Gibt den Bonus auf Munnition zurück.

Autor

Johann

5.26.3.3 int PowerUp::getHealthBonus ( ) const

Get-Methoden für die Objekteigenschaften.

Gibt den Bonus auf Leben zurück.

Autor

Johann

5.26.3.4 int PowerUp::getImmunityCooldownBonus ( ) const

Gibt den Bonus auf Immunität zurück.

Autor

Johann

5.26.3.5 powerUpType PowerUp::getPowerUPType ( ) const

PowerUp::getPowerUPType.

Rückgabe

Art des powerups

Autor

Johann

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/powerup.h
- Wiesn-Run/src/powerup.cpp

5.27 RenderBackground Klassenreferenz

Hintergrund-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt. Die Klasse initialisiert alle Hintergrundgrafiken und aktualisiert deren Positionen im laufendem Spiel. Auch die Bewegungsparallaxe wird hier berechnet. Jede Hintergrundebene besteht immer aus zwei nebeneinander stehenden Bildern. Ist eines davon, bedingt durch die Vorwärtsbewegung des Spielers nicht mehr sichtbar, so wird es wieder am zweiten Bild vorbei, nach vorne geschoben. So wird gewährleistet das der Spieler nicht an den Bildern "vorbeiläuft".

`#include <renderbackground.h>`

Öffentliche Methoden

- RenderBackground (QGraphicsScene *scene, int level)

  Konstruktor für alle Hintergrundgrafiken Hintergrundgrafiken werden initialisiert, positioniert und der Scene hinzugefügt.
- void setPos (int x, QGraphicsPixmapItem *background)

  RenderBackground::setPos Funktion positioniert Hintergrundgrafiken neu.(nur "x" ändert sich, "y" ist immer 0)
- void updateParallaxe (int x)

  RenderBackground::updateParallaxe Die Position der hinteren Hintergrundebene wird laufend so aktualisiert. Und zwar so dass sie sich mit halber Geschwindigkeit des Spielers bewegt und eine Parallaxeeffekt entsteht.
- void updateBackgroundPos (int x)

  RenderBackground::updateBackgroundPos Immer wenn eine Hintergrundgrafik durch Spieler-Vorwärtsbewegung nicht mehr sichtbar ist wird sie wieder nach vorne, vor den Spieler versetzt. So ist ein ständig sichtbarer Hintergrund gewährleistet.

5.27.1 Ausführliche Beschreibung

Hintergrund-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt. Die Klasse initialisiert alle Hintergrundgrafiken und aktualisiert deren Positionen im laufendem Spiel. Auch die Bewegungsparallaxe wird hier berechnet. Jede Hintergrundebene besteht immer aus zwei nebeneinander stehenden Bildern. Ist eines davon, bedingt durch die Vorwärtsbewegung des Spielers nicht mehr sichtbar, so wird es wieder am zweiten Bild vorbei, nach vorne geschoben. So wird gewährleistet das der Spieler nicht an den Bildern "vorbeiläuft".

Autor

Flo

5.27.2 Beschreibung der Konstruktoren und Destruktoren

5.27.2.1 RenderBackground::RenderBackground (QGraphicsScene * *scene,* int *level*)

Konstruktor für alle Hintergrundgrafiken Hintergrundgrafiken werden initialisiert, positioniert und der Scene hinzugefügt.

Parameter

scene	: levelScene
level	: aktuelles Level

Autor

Flo

5.27.3 Dokumentation der Elementfunktionen

5.27.3.1 void RenderBackground::setPos (int *x,* QGraphicsPixmapItem * *background*)

RenderBackground::setPos Funktion positioniert Hintergrundgrafiken neu.(nur "x" ändert sich, "y" ist immer 0)

Parameter

x	: x-Position
background	: Hintergrundgrafikitem

Autor

Flo

5.27.3.2 void RenderBackground::updateBackgroundPos (int *x*)

RenderBackground::updateBackgroundPos Immer wenn eine Hintergrundgrafik durch Spieler-Vorwärtsbewegung nicht mehr sichtbar ist wird sie wieder nach vorne, vor den Spieler versetzt. So ist ein ständig sichtbarer Hintergrund gewährleistet.

Parameter

x	: x-Position des linken Bildrandes im Level

Autor

Flo

5.27.3.3 void RenderBackground::updateParallaxe (int *x*)

RenderBackground::updateParallaxe Die Position der hinteren Hintergrundebene wird laufend so aktualisiert. Und zwar so dass sie sich mit halber Geschwindigkeit des Spielers bewegt und eine Parallaxeeffekt entsteht.

Parameter

x	: x-Wert der Positionsänderung des Spielers im aktuellen Step

Autor

Flo

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/renderbackground.h
- Wiesn-Run/src/renderbackground.cpp

5.28 RenderGUI Klassenreferenz

Anzeigen der Spielerwerte-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt. Die Klasse initialisiert alle Grafikelemente die mit der Anzeige von Spielerwerten zu tun hat (Gesundheit, Alkoholpegel, Munitionsvorrat, Punkte). Außerdem werden hier auch die angezeigten Werte im Spiel fortlaufend aktualisiert. Alle Elemente sind "Kinder" der Gesundheitsanzeige um Positionsaktualisierungen zu vereinfachen (Kindelemente verhalten sich immer relativ um Elternobjekt und werden auch automatisch mit diesem der Scene hinzugefügt bzw. auch wieder entfernt)

`#include <renderGUI.h>`

Öffentliche Methoden

- RenderGUI (QGraphicsScene ∗scene)

  Konstruktor für alle Spielerwert Anzeigen Die Grafikelemente der Anzeigen werden initialisiert, eingestellt und der Scene hinzugefügt.
- void setPos (int x)

  RenderGUI::setPos sorgt für eine Positionsänderung identisch mit der des Spielers auf der X-Achse (Anzeigen bleiben auf den Spieler zentriert)
- void setValues (int health, int alcohol, int ammo, int score)

  RenderGUI::setValues Aktualisierung aller angezeigten Wert, Gesundheits- und Pegelbalken sind immer auf die maximal möglichen Werte normiert.

5.28.1 Ausführliche Beschreibung

Anzeigen der Spielerwerte-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt. Die Klasse initialisiert alle Grafikelemente die mit der Anzeige von Spielerwerten zu tun hat (Gesundheit, Alkoholpegel, Munitionsvorrat, Punkte). Außerdem werden hier auch die angezeigten Werte im Spiel fortlaufend aktualisiert. Alle Elemente sind "Kinder" der Gesundheitsanzeige um Positionsaktualisierungen zu vereinfachen (Kindelemente verhalten sich immer relativ um Elternobjekt und werden auch automatisch mit diesem der Scene hinzugefügt bzw. auch wieder entfernt)

Autor

Flo

5.28.2 Beschreibung der Konstruktoren und Destruktoren

5.28.2.1 RenderGUI::RenderGUI (QGraphicsScene ∗ *scene*)

Konstruktor für alle Spielerwert Anzeigen Die Grafikelemente der Anzeigen werden initialisiert, eingestellt und der Scene hinzugefügt.

Parameter

scene	: levelScene

Autor

Flo

5.28.3 Dokumentation der Elementfunktionen

5.28.3.1 void RenderGUI::setPos (int *x*)

RenderGUI::setPos sorgt für eine Positionsänderung identisch mit der des Spielers auf der X-Achse (Anzeigen bleiben auf den Spieler zentriert)

Parameter

x	: x-Wert der Positionsänderung des Spielers im aktuellen Step

Autor

Flo

5.28.3.2 void RenderGUI::setValues (int *health,* int *alcohol,* int *ammo,* int *score*)

RenderGUI::setValues Aktualisierung aller angezeigten Wert, Gesundheits- und Pegelbalken sind immer auf die maximal möglichen Werte normiert.

Parameter

health	: aktueller Gesundheitswert
alcohol	: altueller Alkoholpegelwert
ammo	: aktueller Munitionsstand
score	: aktueller Punktestad

Autor

Flo

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/renderGUI.h
- Wiesn-Run/src/renderGUI.cpp

5.29 scoreStruct Strukturreferenz

Struktur für die Score des Spielers In dieser Struktur werden Name des Spielers, getötete Gegner, zurückgelegte Entfernung und Alkohol-Punkte gespeichert. Alkohol-Punkte erhält der Spieler für einen gewissen Pegel in einem Zeitabschnitt.

`#include <definitions.h>`

Öffentliche Attribute

- std::string **name**

- int **totalPoints**
- int **distanceCovered**
- int **alcoholPoints**
- int **enemiesKilled**

5.29.1 Ausführliche Beschreibung

Struktur für die Score des Spielers In dieser Struktur werden Name des Spielers, getötete Gegner, zurückgelegte Entfernung und Alkohol-Punkte gespeichert. Alkohol-Punkte erhält der Spieler für einen gewissen Pegel in einem Zeitabschnitt.

Noch zu erledigen Das Konzept der Alkohol-Punkte muss noch ausgearbeitet werden.

Autor

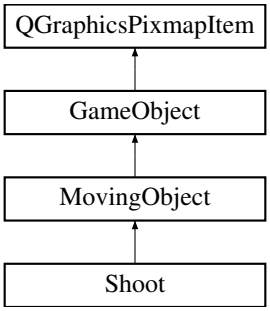
Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

5.30 Shoot Klassenreferenz

Klassendiagramm für Shoot:



Öffentliche Methoden

- Shoot (int posX, int posY, int direction, objectType origin)

  Konstruktor für einen Schuss(Bierkrug)
- int getInflictedDamage () const

  Shoot::getInflictedDamage gibt den Schaden den der Schuss zufügt zurück.
- objectType getOrigin ()

  Shoot::getOrigin gibt den Ursprung des Bierkrugs zurück, Wer hat ihn geworfen (Player/Enemy)
- virtual void **update** ()

Weitere Geerbte Elemente

5.30.1 Beschreibung der Konstruktoren und Destruktoren

5.30.1.1 Shoot::Shoot (int *posX*, int *posY*, int *direction*, objectType *origin*)

Konstruktor für einen Schuss(Bierkrug)

Parameter

posX	: x-Position
posY	: y-Position
origin	: Schuss Erzeuger

Schuss bewegt sich dreimal so schnell wie der spieler Größe des Bierkruges festgesetzt (erste idee)

Autor

Johann

5.30.2 Dokumentation der Elementfunktionen

5.30.2.1 int Shoot::getInflictedDamage ( ) const

Shoot::getInflictedDamage gibt den Schaden den der Schuss zufügt zurück.

Rückgabe

Schaden

Autor

Johann

5.30.2.2 objectType Shoot::getOrigin ( )

Shoot::getOrigin gibt den Ursprung des Bierkrugs zurück, Wer hat ihn geworfen (Player/Enemy)

Rückgabe

Ursprung des Bierkruges

Autor

Johann

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesn-Run/src/shoot.h
- Wiesn-Run/src/shoot.cpp

5.31 stateStruct Strukturreferenz

Struktur für die States des Spiels Sowohl Sound- als auch Grafik-Ausgabe erhalten aus den States Informationen darüber, was gerade im Spiel passiert, z.B. dass gerade der Spieler angreift, ein Gegner stribt etc.

`#include <definitions.h>`

Öffentliche Attribute

- bool **gameOver** = false
- int **actLevel** = 0
- int **audioID_Background** = 0
- bool **beerCollected** = 0
- bool **chickenCollected** = 0

5.31.1 Ausführliche Beschreibung

Struktur für die States des Spiels Sowohl Sound- als auch Grafik-Ausgabe erhalten aus den States Informationen darüber, was gerade im Spiel passiert, z.B. dass gerade der Spieler angreift, ein Gegner stribt etc.

Noch zu erledigen Diese Struktur ist vermutlich überflüssig.

Autor

Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/definitions.h

Kapitel 6

Datei-Dokumentation

6.1 Wiesn-Run/src/portaudio.h-Dateireferenz

The portable PortAudio API.

Klassen

- struct PaHostApiInfo
- struct PaHostErrorInfo
- struct PaDeviceInfo
- struct PaStreamParameters
- struct PaStreamCallbackTimeInfo
- struct PaStreamInfo

Makrodefinitionen

- #define paNoDevice ((PaDeviceIndex)-1)
- #define paUseHostApiSpecificDeviceSpecification ((PaDeviceIndex)-2)
- #define paFloat32 ((PaSampleFormat) 0x00000001)
- #define paInt32 ((PaSampleFormat) 0x00000002)
- #define paInt24 ((PaSampleFormat) 0x00000004)
- #define paInt16 ((PaSampleFormat) 0x00000008)
- #define paInt8 ((PaSampleFormat) 0x00000010)
- #define paUInt8 ((PaSampleFormat) 0x00000020)
- #define paCustomFormat ((PaSampleFormat) 0x00010000)
- #define paNonInterleaved ((PaSampleFormat) 0x80000000)
- #define paFormatIsSupported (0)
- #define paFramesPerBufferUnspecified (0)
- #define paNoFlag ((PaStreamFlags) 0)
- #define paClipOff ((PaStreamFlags) 0x00000001)
- #define paDitherOff ((PaStreamFlags) 0x00000002)
- #define paNeverDropInput ((PaStreamFlags) 0x00000004)
- #define paPrimeOutputBuffersUsingStreamCallback ((PaStreamFlags) 0x00000008)
- #define paPlatformSpecificFlags ((PaStreamFlags)0xFFFF0000)
- #define paInputUnderflow ((PaStreamCallbackFlags) 0x00000001)
- #define paInputOverflow ((PaStreamCallbackFlags) 0x00000002)
- #define paOutputUnderflow ((PaStreamCallbackFlags) 0x00000004)
- #define paOutputOverflow ((PaStreamCallbackFlags) 0x00000008)
- #define paPrimingOutput ((PaStreamCallbackFlags) 0x00000010)

Typdefinitionen

- typedef int PaError
- typedef enum PaErrorCode **PaErrorCode**
- typedef int PaDeviceIndex
- typedef int PaHostApiIndex
- typedef enum PaHostApiTypeId PaHostApiTypeId
- typedef struct PaHostApiInfo PaHostApiInfo
- typedef struct PaHostErrorInfo PaHostErrorInfo
- typedef double PaTime
- typedef unsigned long PaSampleFormat
- typedef struct PaDeviceInfo PaDeviceInfo
- typedef struct PaStreamParameters PaStreamParameters
- typedef void PaStream
- typedef unsigned long PaStreamFlags
- typedef struct PaStreamCallbackTimeInfo PaStreamCallbackTimeInfo
- typedef unsigned long PaStreamCallbackFlags
- typedef enum PaStreamCallbackResult PaStreamCallbackResult
- typedef int PaStreamCallback (const void ∗input, void ∗output, unsigned long frameCount, const PaStreamCallbackTimeInfo ∗timeInfo, PaStreamCallbackFlags statusFlags, void ∗userData)
- typedef void PaStreamFinishedCallback (void ∗userData)
- typedef struct PaStreamInfo PaStreamInfo

Aufzählungen

- enum **PaErrorCode** {
  paNoError = 0, **paNotInitialized** = -10000, **paUnanticipatedHostError**, **paInvalidChannelCount**, **paInvalidSampleRate**, **paInvalidDevice**, **paInvalidFlag**, **paSampleFormatNotSupported**, **paBadIODeviceCombination**, **paInsufficientMemory**, **paBufferTooBig**, **paBufferTooSmall**, **paNullCallback**, **paBadStreamPtr**, **paTimedOut**, **paInternalError**, **paDeviceUnavailable**, **paIncompatibleHostApiSpecificStreamInfo**, **paStreamIsStopped**, **paStreamIsNotStopped**, **paInputOverflowed**, **paOutputUnderflowed**, **paHostApiNotFound**, **paInvalidHostApi**, **paCanNotReadFromACallbackStream**, **paCanNotWriteToACallbackStream**, **paCanNotReadFromAnOutputOnlyStream**, **paCanNotWriteToAnInputOnlyStream**, **paIncompatibleStreamHostApi**, **paBadBufferPtr** }
- enum PaHostApiTypeId {
  paInDevelopment =0, **paDirectSound** =1, **paMME** =2, **paASIO** =3, **paSoundManager** =4, **paCoreAudio** =5, **paOSS** =7, **paALSA** =8, **paAL** =9, **paBeOS** =10, **paWDMKS** =11, **paJACK** =12, **paWASAPI** =13, **paAudioScienceHPI** =14 }
- enum PaStreamCallbackResult { paContinue =0, paComplete =1, paAbort =2 }

Funktionen

- int Pa_GetVersion (void)
- const char ∗ Pa_GetVersionText (void)
- const char ∗ Pa_GetErrorText (PaError errorCode)
- PaError Pa_Initialize (void)
- PaError Pa_Terminate (void)
- PaHostApiIndex Pa_GetHostApiCount (void)
- PaHostApiIndex Pa_GetDefaultHostApi (void)
- const PaHostApiInfo ∗ Pa_GetHostApiInfo (PaHostApiIndex hostApi)
- PaHostApiIndex Pa_HostApiTypeIdToHostApiIndex (PaHostApiTypeId type)

- PaDeviceIndex Pa_HostApiDeviceIndexToDeviceIndex (PaHostApiIndex hostApi, int hostApiDeviceIndex)
- const PaHostErrorInfo * Pa_GetLastHostErrorInfo (void)
- PaDeviceIndex Pa_GetDeviceCount (void)
- PaDeviceIndex Pa_GetDefaultInputDevice (void)
- PaDeviceIndex Pa_GetDefaultOutputDevice (void)
- const PaDeviceInfo * Pa_GetDeviceInfo (PaDeviceIndex device)
- PaError Pa_IsFormatSupported (const PaStreamParameters *inputParameters, const PaStreamParameters *outputParameters, double sampleRate)
- PaError Pa_OpenStream (PaStream **stream, const PaStreamParameters *inputParameters, const PaStreamParameters *outputParameters, double sampleRate, unsigned long framesPerBuffer, PaStreamFlags streamFlags, PaStreamCallback *streamCallback, void *userData)
- PaError Pa_OpenDefaultStream (PaStream **stream, int numInputChannels, int numOutputChannels, PaSampleFormat sampleFormat, double sampleRate, unsigned long framesPerBuffer, PaStreamCallback *streamCallback, void *userData)
- PaError Pa_CloseStream (PaStream *stream)
- PaError Pa_SetStreamFinishedCallback (PaStream *stream, PaStreamFinishedCallback *streamFinishedCallback)
- PaError Pa_StartStream (PaStream *stream)
- PaError Pa_StopStream (PaStream *stream)
- PaError Pa_AbortStream (PaStream *stream)
- PaError Pa_IsStreamStopped (PaStream *stream)
- PaError Pa_IsStreamActive (PaStream *stream)
- const PaStreamInfo * Pa_GetStreamInfo (PaStream *stream)
- PaTime Pa_GetStreamTime (PaStream *stream)
- double Pa_GetStreamCpuLoad (PaStream *stream)
- PaError Pa_ReadStream (PaStream *stream, void *buffer, unsigned long frames)
- PaError Pa_WriteStream (PaStream *stream, const void *buffer, unsigned long frames)
- signed long Pa_GetStreamReadAvailable (PaStream *stream)
- signed long Pa_GetStreamWriteAvailable (PaStream *stream)
- PaHostApiTypeId Pa_GetStreamHostApiType (PaStream *stream)
- PaError Pa_GetSampleSize (PaSampleFormat format)
- void Pa_Sleep (long msec)

6.1.1 Ausführliche Beschreibung

The portable PortAudio API.

6.1.2 Makro-Dokumentation

6.1.2.1 #define paClipOff ((PaStreamFlags) 0x00000001)

Disable default clipping of out of range samples.

Siehe auch

PaStreamFlags

6.1.2.2 #define paCustomFormat ((PaSampleFormat) 0x00010000)

Siehe auch

PaSampleFormat

6.1.2.3 #define paDitherOff ((PaStreamFlags) 0x00000002)

Disable default dithering.

Siehe auch

PaStreamFlags

6.1.2.4 #define paFloat32 ((PaSampleFormat) 0x00000001)

Siehe auch

PaSampleFormat

6.1.2.5 #define paFormatIsSupported (0)

Return code for Pa_IsFormatSupported indicating success.

6.1.2.6 #define paFramesPerBufferUnspecified (0)

Can be passed as the framesPerBuffer parameter to Pa_OpenStream() or Pa_OpenDefaultStream() to indicate that the stream callback will accept buffers of any size.

6.1.2.7 #define paInputOverflow ((PaStreamCallbackFlags) 0x00000002)

In a stream opened with paFramesPerBufferUnspecified, indicates that data prior to the first sample of the input buffer was discarded due to an overflow, possibly because the stream callback is using too much CPU time. Otherwise indicates that data prior to one or more samples in the input buffer was discarded.

Siehe auch

PaStreamCallbackFlags

6.1.2.8 #define paInputUnderflow ((PaStreamCallbackFlags) 0x00000001)

In a stream opened with paFramesPerBufferUnspecified, indicates that input data is all silence (zeros) because no real data is available. In a stream opened without paFramesPerBufferUnspecified, it indicates that one or more zero samples have been inserted into the input buffer to compensate for an input underflow.

Siehe auch

PaStreamCallbackFlags

6.1.2.9 #define paInt16 ((PaSampleFormat) 0x00000008)

Siehe auch

PaSampleFormat

6.1.2.10 #define paInt24 ((PaSampleFormat) 0x00000004)

Packed 24 bit format.

Siehe auch

PaSampleFormat

6.1.2.11 #define paInt32 ((PaSampleFormat) 0x00000002)

Siehe auch

PaSampleFormat

6.1.2.12 #define paInt8 ((PaSampleFormat) 0x00000010)

Siehe auch

PaSampleFormat

6.1.2.13 #define paNeverDropInput ((PaStreamFlags) 0x00000004)

Flag requests that where possible a full duplex stream will not discard overflowed input samples without calling the stream callback. This flag is only valid for full duplex callback streams and only when used in combination with the paFramesPerBufferUnspecified (0) framesPerBuffer parameter. Using this flag incorrectly results in a paInvalidFlag error being returned from Pa_OpenStream and Pa_OpenDefaultStream.

Siehe auch

PaStreamFlags, paFramesPerBufferUnspecified

6.1.2.14 #define paNoDevice ((PaDeviceIndex)-1)

A special PaDeviceIndex value indicating that no device is available, or should be used.

Siehe auch

PaDeviceIndex

6.1.2.15 #define paNoFlag ((PaStreamFlags) 0)

Siehe auch

PaStreamFlags

6.1.2.16 #define paNonInterleaved ((PaSampleFormat) 0x80000000)

Siehe auch

PaSampleFormat

6.1.2.17 #define paOutputOverflow ((PaStreamCallbackFlags) 0x00000008)

Indicates that output data will be discarded because no room is available.

Siehe auch

PaStreamCallbackFlags

6.1.2.18 #define paOutputUnderflow ((PaStreamCallbackFlags) 0x00000004)

Indicates that output data (or a gap) was inserted, possibly because the stream callback is using too much CPU time.

Siehe auch

PaStreamCallbackFlags

6.1.2.19 #define paPlatformSpecificFlags ((PaStreamFlags)0xFFFF0000)

A mask specifying the platform specific bits.

Siehe auch

PaStreamFlags

6.1.2.20 #define paPrimeOutputBuffersUsingStreamCallback ((PaStreamFlags) 0x00000008)

Call the stream callback to fill initial output buffers, rather than the default behavior of priming the buffers with zeros (silence). This flag has no effect for input-only and blocking read/write streams.

Siehe auch

PaStreamFlags

6.1.2.21 #define paPrimingOutput ((PaStreamCallbackFlags) 0x00000010)

Some of all of the output data will be used to prime the stream, input data may be zero.

Siehe auch

PaStreamCallbackFlags

6.1.2.22 #define paUInt8 ((PaSampleFormat) 0x00000020)

Siehe auch

PaSampleFormat

6.1.2.23 #define paUseHostApiSpecificDeviceSpecification ((PaDeviceIndex)-2)

A special PaDeviceIndex value indicating that the device(s) to be used are specified in the host api specific stream info structure.

Siehe auch

PaDeviceIndex

6.1.3 Dokumentation der benutzerdefinierten Typen

6.1.3.1 typedef int PaDeviceIndex

The type used to refer to audio devices. Values of this type usually range from 0 to (Pa_GetDeviceCount()-1), and may also take on the PaNoDevice and paUseHostApiSpecificDeviceSpecification values.

Siehe auch

Pa_GetDeviceCount, paNoDevice, paUseHostApiSpecificDeviceSpecification

6.1.3.2 typedef struct PaDeviceInfo PaDeviceInfo

A structure providing information and capabilities of PortAudio devices. Devices may support input, output or both input and output.

6.1.3.3 typedef int PaError

Error codes returned by PortAudio functions. Note that with the exception of paNoError, all PaErrorCodes are negative.

6.1.3.4 typedef int PaHostApiIndex

The type used to enumerate to host APIs at runtime. Values of this type range from 0 to (Pa_GetHostApiCount()-1).

Siehe auch

Pa_GetHostApiCount

6.1.3.5 typedef struct PaHostApiInfo PaHostApiInfo

A structure containing information about a particular host API.

6.1.3.6 typedef enum PaHostApiTypeId PaHostApiTypeId

Unchanging unique identifiers for each supported host API. This type is used in the PaHostApiInfo structure. The values are guaranteed to be unique and to never change, thus allowing code to be written that conditionally uses host API specific extensions.

New type ids will be allocated when support for a host API reaches "public alpha" status, prior to that developers should use the paInDevelopment type id.

Siehe auch

PaHostApiInfo

6.1.3.7 typedef struct PaHostErrorInfo PaHostErrorInfo

Structure used to return information about a host error condition.

6.1.3.8 typedef unsigned long PaSampleFormat

A type used to specify one or more sample formats. Each value indicates a possible format for sound data passed to and from the stream callback, Pa_ReadStream and Pa_WriteStream.

The standard formats paFloat32, paInt16, paInt32, paInt24, paInt8 and aUInt8 are usually implemented by all implementations.

The floating point representation (paFloat32) uses +1.0 and -1.0 as the maximum and minimum respectively.

paUInt8 is an unsigned 8 bit format where 128 is considered "ground"

The paNonInterleaved flag indicates that audio data is passed as an array of pointers to separate buffers, one buffer for each channel. Usually, when this flag is not used, audio data is passed as a single buffer with all channels interleaved.

Siehe auch

Pa_OpenStream, Pa_OpenDefaultStream, PaDeviceInfo
paFloat32, paInt16, paInt32, paInt24, paInt8
paUInt8, paCustomFormat, paNonInterleaved

6.1.3.9 typedef void PaStream

A single PaStream can provide multiple channels of real-time streaming audio input and output to a client application. A stream provides access to audio hardware represented by one or more PaDevices. Depending on the underlying Host API, it may be possible to open multiple streams using the same device, however this behavior is implementation defined. Portable applications should assume that a PaDevice may be simultaneously used by at most one PaStream.

Pointers to PaStream objects are passed between PortAudio functions that operate on streams.

Siehe auch

Pa_OpenStream, Pa_OpenDefaultStream, Pa_OpenDefaultStream, Pa_CloseStream, Pa_StartStream, Pa_StopStream, Pa_AbortStream, Pa_IsStreamActive, Pa_GetStreamTime, Pa_GetStreamCpuLoad

6.1.3.10 typedef int PaStreamCallback(const void *input, void *output, unsigned long frameCount, const PaStreamCallbackTimeInfo *timeInfo, PaStreamCallbackFlags statusFlags, void *userData)

Functions of type PaStreamCallback are implemented by PortAudio clients. They consume, process or generate audio in response to requests from an active PortAudio stream.

When a stream is running, PortAudio calls the stream callback periodically. The callback function is responsible for processing buffers of audio samples passed via the input and output parameters.

The PortAudio stream callback runs at very high or real-time priority. It is required to consistently meet its time deadlines. Do not allocate memory, access the file system, call library functions or call other functions from the stream callback that may block or take an unpredictable amount of time to complete.

In order for a stream to maintain glitch-free operation the callback must consume and return audio data faster than it is recorded and/or played. PortAudio anticipates that each callback invocation may execute for a duration approaching the duration of frameCount audio frames at the stream sample rate. It is reasonable to expect to be able to utilise 70% or more of the available CPU time in the PortAudio callback. However, due to buffer size adaption and other factors, not all host APIs are able to guarantee audio stability under heavy CPU load with arbitrary fixed callback buffer sizes. When high callback CPU utilisation is required the most robust behavior can be achieved by using paFramesPerBufferUnspecified as the Pa_OpenStream() framesPerBuffer parameter.

Parameter

input	and
output	are either arrays of interleaved samples or; if non-interleaved samples were requested using the paNonInterleaved sample format flag, an array of buffer pointers, one non-interleaved buffer for each channel.

The format, packing and number of channels used by the buffers are determined by parameters to Pa_OpenStream().

Parameter

frameCount	The number of sample frames to be processed by the stream callback.
timeInfo	Timestamps indicating the ADC capture time of the first sample in the input buffer, the DAC output time of the first sample in the output buffer and the time the callback was invoked. See PaStreamCallbackTimeInfo and Pa_GetStreamTime()
statusFlags	Flags indicating whether input and/or output buffers have been inserted or will be dropped to overcome underflow or overflow conditions.
userData	The value of a user supplied pointer passed to Pa_OpenStream() intended for storing synthesis data etc.

Rückgabe

The stream callback should return one of the values in the PaStreamCallbackResult enumeration. To ensure that the callback continues to be called, it should return paContinue (0). Either paComplete or paAbort can be returned to finish stream processing, after either of these values is returned the callback will not be called again. If paAbort is returned the stream will finish as soon as possible. If paComplete is returned, the stream will continue until all buffers generated by the callback have been played. This may be useful in applications such as soundfile players where a specific duration of output is required. However, it is not necessary to utilize this mechanism as Pa_StopStream(), Pa_AbortStream() or Pa_CloseStream() can also be used to stop the stream. The callback must always fill the entire output buffer irrespective of its return value.

Siehe auch

Pa_OpenStream, Pa_OpenDefaultStream

Zu beachten

With the exception of Pa_GetStreamCpuLoad() it is not permissible to call PortAudio API functions from within the stream callback.

6.1.3.11 typedef unsigned long PaStreamCallbackFlags

Flag bit constants for the statusFlags to PaStreamCallback.

Siehe auch

paInputUnderflow, paInputOverflow, paOutputUnderflow, paOutputOverflow, paPrimingOutput

6.1.3.12 typedef enum PaStreamCallbackResult PaStreamCallbackResult

Allowable return values for the PaStreamCallback.

Siehe auch

PaStreamCallback

6.1.3.13 typedef struct PaStreamCallbackTimeInfo PaStreamCallbackTimeInfo

Timing information for the buffers passed to the stream callback.

Time values are expressed in seconds and are synchronised with the time base used by Pa_GetStreamTime() for the associated stream.

Siehe auch

PaStreamCallback, Pa_GetStreamTime

6.1.3.14 typedef void PaStreamFinishedCallback(void *userData)

Functions of type PaStreamFinishedCallback are implemented by PortAudio clients. They can be registered with a stream using the Pa_SetStreamFinishedCallback function. Once registered they are called when the stream becomes inactive (ie once a call to Pa_StopStream() will not block). A stream will become inactive after the stream callback returns non-zero, or when Pa_StopStream or Pa_AbortStream is called. For a stream providing audio output, if the stream callback returns paComplete, or Pa_StopStream is called, the stream finished callback will not be called until all generated sample data has been played.

Parameter

userData	The userData parameter supplied to Pa_OpenStream()

Siehe auch

Pa_SetStreamFinishedCallback

6.1.3.15 typedef unsigned long PaStreamFlags

Flags used to control the behavior of a stream. They are passed as parameters to Pa_OpenStream or Pa_OpenDefaultStream. Multiple flags may be ORed together.

Siehe auch

Pa_OpenStream, Pa_OpenDefaultStream
paNoFlag, paClipOff, paDitherOff, paNeverDropInput, paPrimeOutputBuffersUsingStreamCallback, paPlatformSpecificFlags

6.1.3.16 typedef struct PaStreamInfo PaStreamInfo

A structure containing unchanging information about an open stream.

Siehe auch

Pa_GetStreamInfo

6.1.3.17 typedef struct PaStreamParameters PaStreamParameters

Parameters for one direction (input or output) of a stream.

6.1.3.18 typedef double PaTime

The type used to represent monotonic time in seconds. PaTime is used for the fields of the PaStreamCallbackTime-Info argument to the PaStreamCallback and as the result of Pa_GetStreamTime().

PaTime values have unspecified origin.

Siehe auch

PaStreamCallback, PaStreamCallbackTimeInfo, Pa_GetStreamTime

6.1.4 Dokumentation der Aufzählungstypen

6.1.4.1 enum PaHostApiTypeId

Unchanging unique identifiers for each supported host API. This type is used in the PaHostApiInfo structure. The values are guaranteed to be unique and to never change, thus allowing code to be written that conditionally uses host API specific extensions.

New type ids will be allocated when support for a host API reaches "public alpha" status, prior to that developers should use the paInDevelopment type id.

Siehe auch

PaHostApiInfo

6.1.4.2 enum PaStreamCallbackResult

Allowable return values for the PaStreamCallback.

Siehe auch

PaStreamCallback

Aufzählungswerte

paContinue Signal that the stream should continue invoking the callback and processing audio.

paComplete Signal that the stream should stop invoking the callback and finish once all output samples have played.

paAbort Signal that the stream should stop invoking the callback and finish as soon as possible.

6.1.5 Dokumentation der Funktionen

6.1.5.1 PaError Pa_AbortStream (PaStream * *stream*)

Terminates audio processing immediately without waiting for pending buffers to complete.

6.1.5.2 PaError Pa_CloseStream (PaStream * *stream*)

Closes an audio stream. If the audio stream is active it discards any pending buffers as if Pa_AbortStream() had been called.

6.1.5.3 PaHostApiIndex Pa_GetDefaultHostApi (void)

Retrieve the index of the default host API. The default host API will be the lowest common denominator host API on the current platform and is unlikely to provide the best performance.

Rückgabe

A non-negative value ranging from 0 to (Pa_GetHostApiCount()-1) indicating the default host API index or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.4 PaDeviceIndex Pa_GetDefaultInputDevice (void)

Retrieve the index of the default input device. The result can be used in the inputDevice parameter to Pa_OpenStream().

Rückgabe

The default input device index for the default host API, or paNoDevice if no default input device is available or an error was encountered.

6.1.5.5 PaDeviceIndex Pa_GetDefaultOutputDevice (void)

Retrieve the index of the default output device. The result can be used in the outputDevice parameter to Pa_OpenStream().

Rückgabe

The default output device index for the default host API, or paNoDevice if no default output device is available or an error was encountered.

Zu beachten

On the PC, the user can specify a default device by setting an environment variable. For example, to use device #1.

```
set PA_RECOMMENDED_OUTPUT_DEVICE=1
```

The user should first determine the available device ids by using the supplied application "pa_devs".

6.1.5.6 PaDeviceIndex Pa_GetDeviceCount (void)

Retrieve the number of available devices. The number of available devices may be zero.

Rückgabe

A non-negative value indicating the number of available devices or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.7 const PaDeviceInfo* Pa_GetDeviceInfo (PaDeviceIndex *device*)

Retrieve a pointer to a PaDeviceInfo structure containing information about the specified device.

Rückgabe

A pointer to an immutable PaDeviceInfo structure. If the device parameter is out of range the function returns NULL.

Parameter

device	A valid device index in the range 0 to (Pa_GetDeviceCount()-1)

Zu beachten

PortAudio manages the memory referenced by the returned pointer, the client must not manipulate or free the memory. The pointer is only guaranteed to be valid between calls to Pa_Initialize() and Pa_Terminate().

Siehe auch

PaDeviceInfo, PaDeviceIndex

6.1.5.8 const char* Pa_GetErrorText (PaError *errorCode*)

Translate the supplied PortAudio error code into a human readable message.

6.1.5.9 PaHostApiIndex Pa_GetHostApiCount (void)

Retrieve the number of available host APIs. Even if a host API is available it may have no devices available.

Rückgabe

A non-negative value indicating the number of available host APIs or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

Siehe auch

PaHostApiIndex

6.1.5.10 const PaHostApiInfo* Pa_GetHostApiInfo (PaHostApiIndex *hostApi*)

Retrieve a pointer to a structure containing information about a specific host Api.

Parameter

hostApi	A valid host API index ranging from 0 to (Pa_GetHostApiCount()-1)

Rückgabe

A pointer to an immutable PaHostApiInfo structure describing a specific host API. If the hostApi parameter is out of range or an error is encountered, the function returns NULL.

The returned structure is owned by the PortAudio implementation and must not be manipulated or freed. The pointer is only guaranteed to be valid between calls to Pa_Initialize() and Pa_Terminate().

6.1.5.11 const PaHostErrorInfo* Pa_GetLastHostErrorInfo (void)

Return information about the last host error encountered. The error information returned by Pa_GetLastHostErrorInfo() will never be modified asynchronously by errors occurring in other PortAudio owned threads (such as the thread that manages the stream callback.)

This function is provided as a last resort, primarily to enhance debugging by providing clients with access to all available error information.

Rückgabe

A pointer to an immutable structure constraining information about the host error. The values in this structure will only be valid if a PortAudio function has previously returned the paUnanticipatedHostError error code.

6.1.5.12 PaError Pa_GetSampleSize (PaSampleFormat *format*)

Retrieve the size of a given sample format in bytes.

Rückgabe

The size in bytes of a single sample in the specified format, or paSampleFormatNotSupported if the format is not supported.

6.1.5.13 double Pa_GetStreamCpuLoad (PaStream * *stream*)

Retrieve CPU usage information for the specified stream. The "CPU Load" is a fraction of total CPU time consumed by a callback stream's audio processing routines including, but not limited to the client supplied stream callback. This function does not work with blocking read/write streams.

This function may be called from the stream callback function or the application.

Rückgabe

A floating point value, typically between 0.0 and 1.0, where 1.0 indicates that the stream callback is consuming the maximum number of CPU cycles possible to maintain real-time operation. A value of 0.5 would imply that PortAudio and the stream callback was consuming roughly 50% of the available CPU time. The return value may exceed 1.0. A value of 0.0 will always be returned for a blocking read/write stream, or if an error occurs.

6.1.5.14 PaHostApiTypeId Pa_GetStreamHostApiType (PaStream * *stream*)

Retrieve the host type handling an open stream.

Rückgabe

Returns a non-negative value representing the host API type handling an open stream or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.15 const PaStreamInfo* Pa_GetStreamInfo (PaStream * *stream*)

Retrieve a pointer to a PaStreamInfo structure containing information about the specified stream.

Rückgabe

A pointer to an immutable PaStreamInfo structure. If the stream parameter invalid, or an error is encountered, the function returns NULL.

Parameter

stream	A pointer to an open stream previously created with Pa_OpenStream.

Zu beachten

PortAudio manages the memory referenced by the returned pointer, the client must not manipulate or free the memory. The pointer is only guaranteed to be valid until the specified stream is closed.

Siehe auch

PaStreamInfo

6.1.5.16 signed long Pa_GetStreamReadAvailable (PaStream * *stream*)

Retrieve the number of frames that can be read from the stream without waiting.

Rückgabe

Returns a non-negative value representing the maximum number of frames that can be read from the stream without blocking or busy waiting or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.17 PaTime Pa_GetStreamTime (PaStream * *stream*)

Returns the current time in seconds for a stream according to the same clock used to generate callback PaStreamCallbackTimeInfo timestamps. The time values are monotonically increasing and have unspecified origin.

Pa_GetStreamTime returns valid time values for the entire life of the stream, from when the stream is opened until it is closed. Starting and stopping the stream does not affect the passage of time returned by Pa_GetStreamTime.

This time may be used for synchronizing other events to the audio stream, for example synchronizing audio to MIDI.

Rückgabe

The stream's current time in seconds, or 0 if an error occurred.

Siehe auch

PaTime, PaStreamCallback, PaStreamCallbackTimeInfo

6.1.5.18 signed long Pa_GetStreamWriteAvailable (PaStream * *stream*)

Retrieve the number of frames that can be written to the stream without waiting.

Rückgabe

Returns a non-negative value representing the maximum number of frames that can be written to the stream without blocking or busy waiting or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

6.1.5.19 int Pa_GetVersion (void)

Retrieve the release number of the currently running PortAudio build, eg 1900.

6.1.5.20 const char* Pa_GetVersionText (void)

Retrieve a textual description of the current PortAudio build, eg "PortAudio V19-devel 13 October 2002".

6.1.5.21 PaDeviceIndex Pa_HostApiDeviceIndexToDeviceIndex (PaHostApiIndex *hostApi*, int *hostApiDeviceIndex*)

Convert a host-API-specific device index to standard PortAudio device index. This function may be used in conjunction with the deviceCount field of PaHostApiInfo to enumerate all devices for the specified host API.

Parameter

hostApi	A valid host API index ranging from 0 to (Pa_GetHostApiCount()-1)
hostApiDevice-Index	A valid per-host device index in the range 0 to (Pa_GetHostApiInfo(hostApi)->deviceCount-1)

Rückgabe

A non-negative PaDeviceIndex ranging from 0 to (Pa_GetDeviceCount()-1) or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

A paInvalidHostApi error code indicates that the host API index specified by the hostApi parameter is out of range.

A paInvalidDevice error code indicates that the hostApiDeviceIndex parameter is out of range.

Siehe auch

PaHostApiInfo

6.1.5.22 PaHostApiIndex Pa_HostApiTypeIdToHostApiIndex (PaHostApiTypeId *type*)

Convert a static host API unique identifier, into a runtime host API index.

Parameter

type	A unique host API identifier belonging to the PaHostApiTypeId enumeration.

Rückgabe

A valid PaHostApiIndex ranging from 0 to (Pa_GetHostApiCount()-1) or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

The paHostApiNotFound error code indicates that the host API specified by the type parameter is not available.

Siehe auch

PaHostApiTypeId

6.1.5.23 PaError Pa_Initialize (void)

Library initialization function - call this before using PortAudio. This function initializes internal data structures and prepares underlying host APIs for use. With the exception of Pa_GetVersion(), Pa_GetVersionText(), and Pa_GetErrorText(), this function MUST be called before using any other PortAudio API functions.

If Pa_Initialize() is called multiple times, each successful call must be matched with a corresponding call to Pa_Terminate(). Pairs of calls to Pa_Initialize()/Pa_Terminate() may overlap, and are not required to be fully nested.

Note that if Pa_Initialize() returns an error code, Pa_Terminate() should NOT be called.

Rückgabe

paNoError if successful, otherwise an error code indicating the cause of failure.

Siehe auch

Pa_Terminate

6.1.5.24 PaError Pa_IsFormatSupported (const PaStreamParameters * *inputParameters*, const PaStreamParameters * *outputParameters*, double *sampleRate*)

Determine whether it would be possible to open a stream with the specified parameters.

Parameter

inputParameters	A structure that describes the input parameters used to open a stream. The suggestedLatency field is ignored. See PaStreamParameters for a description of these parameters. inputParameters must be NULL for output-only streams.
outputParameters	A structure that describes the output parameters used to open a stream. The suggestedLatency field is ignored. See PaStreamParameters for a description of these parameters. outputParameters must be NULL for input-only streams.
sampleRate	The required sampleRate. For full-duplex streams it is the sample rate for both input and output

Rückgabe

Returns 0 if the format is supported, and an error code indicating why the format is not supported otherwise. The constant paFormatIsSupported is provided to compare with the return value for success.

Siehe auch

paFormatIsSupported, PaStreamParameters

6.1.5.25 PaError Pa_IsStreamActive (PaStream * *stream*)

Determine whether the stream is active. A stream is active after a successful call to Pa_StartStream(), until it becomes inactive either as a result of a call to Pa_StopStream() or Pa_AbortStream(), or as a result of a return value other than paContinue from the stream callback. In the latter case, the stream is considered inactive after the last buffer has finished playing.

Rückgabe

Returns one (1) when the stream is active (ie playing or recording audio), zero (0) when not playing or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

Siehe auch

Pa_StopStream, Pa_AbortStream, Pa_IsStreamStopped

6.1.5.26 PaError Pa_IsStreamStopped (PaStream * *stream*)

Determine whether the stream is stopped. A stream is considered to be stopped prior to a successful call to Pa_StartStream and after a successful call to Pa_StopStream or Pa_AbortStream. If a stream callback returns a value other than paContinue the stream is NOT considered to be stopped.

Rückgabe

Returns one (1) when the stream is stopped, zero (0) when the stream is running or, a PaErrorCode (which are always negative) if PortAudio is not initialized or an error is encountered.

Siehe auch

Pa_StopStream, Pa_AbortStream, Pa_IsStreamActive

6.1.5.27 PaError Pa_OpenDefaultStream (PaStream ** *stream*, int *numInputChannels*, int *numOutputChannels*, PaSampleFormat *sampleFormat*, double *sampleRate*, unsigned long *framesPerBuffer*, PaStreamCallback * *streamCallback*, void * *userData*)

A simplified version of Pa_OpenStream() that opens the default input and/or output devices.

Parameter

stream	The address of a PaStream pointer which will receive a pointer to the newly opened stream.
numInput-Channels	The number of channels of sound that will be supplied to the stream callback or returned by Pa_ReadStream. It can range from 1 to the value of maxInputChannels in the PaDeviceInfo record for the default input device. If 0 the stream is opened as an output-only stream.
numOutput-Channels	The number of channels of sound to be delivered to the stream callback or passed to Pa-_WriteStream. It can range from 1 to the value of maxOutputChannels in the PaDeviceInfo record for the default output device. If 0 the stream is opened as an output-only stream.
sampleFormat	The sample format of both the input and output buffers provided to the callback or passed to and from Pa_ReadStream and Pa_WriteStream. sampleFormat may be any of the formats described by the PaSampleFormat enumeration.
sampleRate	Same as Pa_OpenStream parameter of the same name.
framesPerBuffer	Same as Pa_OpenStream parameter of the same name.
streamCallback	Same as Pa_OpenStream parameter of the same name.
userData	Same as Pa_OpenStream parameter of the same name.

Rückgabe

As for Pa_OpenStream

Siehe auch

Pa_OpenStream, PaStreamCallback

6.1.5.28 PaError Pa_OpenStream (PaStream ** *stream*, const PaStreamParameters * *inputParameters*, const PaStreamParameters * *outputParameters*, double *sampleRate*, unsigned long *framesPerBuffer*, PaStreamFlags *streamFlags*, PaStreamCallback * *streamCallback*, void * *userData*)

Opens a stream for either input, output or both.

Parameter

stream	The address of a PaStream pointer which will receive a pointer to the newly opened stream.
inputParameters	A structure that describes the input parameters used by the opened stream. See PaStream-Parameters for a description of these parameters. inputParameters must be NULL for output-only streams.
output-Parameters	A structure that describes the output parameters used by the opened stream. See PaStream-Parameters for a description of these parameters. outputParameters must be NULL for input-only streams.
sampleRate	The desired sampleRate. For full-duplex streams it is the sample rate for both input and output
framesPerBuffer	The number of frames passed to the stream callback function, or the preferred block granularity for a blocking read/write stream. The special value paFramesPerBufferUnspecified (0) may be used to request that the stream callback will receive an optimal (and possibly varying) number of frames based on host requirements and the requested latency settings. Note: With some host APIs, the use of non-zero framesPerBuffer for a callback stream may introduce an additional layer of buffering which could introduce additional latency. PortAudio guarantees that the additional latency will be kept to the theoretical minimum however, it is strongly recommended that a non-zero framesPerBuffer value only be used when your algorithm requires a fixed number of frames per stream callback.

streamFlags	Flags which modify the behavior of the streaming process. This parameter may contain a combination of flags ORed together. Some flags may only be relevant to certain buffer formats.
streamCallback	A pointer to a client supplied function that is responsible for processing and filling input and output buffers. If this parameter is NULL the stream will be opened in 'blocking read/write' mode. In blocking mode, the client can receive sample data using Pa_ReadStream and write sample data using Pa_WriteStream, the number of samples that may be read or written without blocking is returned by Pa_GetStreamReadAvailable and Pa_GetStreamWriteAvailable respectively.
userData	A client supplied pointer which is passed to the stream callback function. It could for example, contain a pointer to instance data necessary for processing the audio buffers. This parameter is ignored if streamCallback is NULL.

Rückgabe

Upon success Pa_OpenStream() returns paNoError and places a pointer to a valid PaStream in the stream argument. The stream is inactive (stopped). If a call to Pa_OpenStream() fails, a non-zero error code is returned (see PaError for possible error codes) and the value of stream is invalid.

Siehe auch

PaStreamParameters, PaStreamCallback, Pa_ReadStream, Pa_WriteStream, Pa_GetStreamReadAvailable, Pa_GetStreamWriteAvailable

6.1.5.29 PaError Pa_ReadStream (PaStream * *stream*, void * *buffer*, unsigned long *frames*)

Read samples from an input stream. The function doesn't return until the entire buffer has been filled - this may involve waiting for the operating system to supply the data.

Parameter

stream	A pointer to an open stream previously created with Pa_OpenStream.
buffer	A pointer to a buffer of sample frames. The buffer contains samples in the format specified by the inputParameters->sampleFormat field used to open the stream, and the number of channels specified by inputParameters->numChannels. If non-interleaved samples were requested using the paNonInterleaved sample format flag, buffer is a pointer to the first element of an array of buffer pointers, one non-interleaved buffer for each channel.
frames	The number of frames to be read into buffer. This parameter is not constrained to a specific range, however high performance applications will want to match this parameter to the framesPerBuffer parameter used when opening the stream.

Rückgabe

On success PaNoError will be returned, or PaInputOverflowed if input data was discarded by PortAudio after the previous call and before this call.

6.1.5.30 PaError Pa_SetStreamFinishedCallback (PaStream * *stream*, PaStreamFinishedCallback * *streamFinishedCallback*)

Register a stream finished callback function which will be called when the stream becomes inactive. See the description of PaStreamFinishedCallback for further details about when the callback will be called.

Parameter

stream	a pointer to a PaStream that is in the stopped state - if the stream is not stopped, the stream's finished callback will remain unchanged and an error code will be returned.
streamFinished-Callback	a pointer to a function with the same signature as PaStreamFinishedCallback, that will be called when the stream becomes inactive. Passing NULL for this parameter will un-register a previously registered stream finished callback function.

Rückgabe

on success returns paNoError, otherwise an error code indicating the cause of the error.

Siehe auch

PaStreamFinishedCallback

6.1.5.31 void Pa_Sleep (long *msec*)

Put the caller to sleep for at least 'msec' milliseconds. This function is provided only as a convenience for authors of portable code (such as the tests and examples in the PortAudio distribution.)

The function may sleep longer than requested so don't rely on this for accurate musical timing.

6.1.5.32 PaError Pa_StartStream (PaStream * *stream*)

Commences audio processing.

6.1.5.33 PaError Pa_StopStream (PaStream * *stream*)

Terminates audio processing. It waits until all pending audio buffers have been played before it returns.

6.1.5.34 PaError Pa_Terminate (void)

Library termination function - call this when finished using PortAudio. This function deallocates all resources allocated by PortAudio since it was initialized by a call to Pa_Initialize(). In cases where Pa_Initialise() has been called multiple times, each call must be matched with a corresponding call to Pa_Terminate(). The final matching call to Pa_Terminate() will automatically close any PortAudio streams that are still open.

Pa_Terminate() MUST be called before exiting a program which uses PortAudio. Failure to do so may result in serious resource leaks, such as audio devices not being available until the next reboot.

Rückgabe

paNoError if successful, otherwise an error code indicating the cause of failure.

Siehe auch

Pa_Initialize

6.1.5.35 PaError Pa_WriteStream (PaStream * *stream*, const void * *buffer*, unsigned long *frames*)

Write samples to an output stream. This function doesn't return until the entire buffer has been consumed - this may involve waiting for the operating system to consume the data.

Parameter

stream	A pointer to an open stream previously created with Pa_OpenStream.
buffer	A pointer to a buffer of sample frames. The buffer contains samples in the format specified by the outputParameters->sampleFormat field used to open the stream, and the number of channels specified by outputParameters->numChannels. If non-interleaved samples were requested using the paNonInterleaved sample format flag, buffer is a pointer to the first element of an array of buffer pointers, one non-interleaved buffer for each channel.
frames	The number of frames to be written from buffer. This parameter is not constrained to a specific range, however high performance applications will want to match this parameter to the framesPerBuffer parameter used when opening the stream.

Rückgabe

On success PaNoError will be returned, or paOutputUnderflowed if additional output data was inserted after the previous call and before this call.