

Wiesn-Run

Erzeugt von Doxygen 1.8.6

Mon Jul 13 2015 15:38:48

Inhaltsverzeichnis

1	Ausstehende Aufgaben	1
2	Hierarchie-Verzeichnis	3
2.1	Klassenhierarchie	3
3	Klassen-Verzeichnis	5
3.1	Auflistung der Klassen	5
4	Datei-Verzeichnis	7
4.1	Auflistung der Dateien	7
5	Klassen-Dokumentation	9
5.1	Audio Klassenreferenz	9
5.1.1	Ausführliche Beschreibung	10
5.1.2	Beschreibung der Konstruktoren und Destruktoren	10
5.1.2.1	Audio	10
5.1.2.2	~Audio	10
5.1.3	Dokumentation der Elementfunktionen	10
5.1.3.1	getSource	10
5.1.3.2	getSample	10
5.1.3.3	getSamplenumber	11
5.1.3.4	readSamples	11
5.1.3.5	to16bitSample	11
5.1.3.6	normalize	12
5.2	AudioControl Klassenreferenz	12
5.2.1	Ausführliche Beschreibung	13
5.2.2	Klassen-Dokumentation	13
5.2.2.1	struct AudioControl::playStruct	13
5.2.3	Beschreibung der Konstruktoren und Destruktoren	14
5.2.3.1	AudioControl	14
5.2.3.2	~AudioControl	15
5.2.4	Dokumentation der Elementfunktionen	15
5.2.4.1	playInitialize	15

5.2.4.2	playTerminate	15
5.2.4.3	updatePlayevents	15
5.2.4.4	instancepaCallback	15
5.2.4.5	staticpaCallback	16
5.2.5	Dokumentation der Datenelemente	16
5.2.5.1	mtx	16
5.2.5.2	playeventsnumber	17
5.2.5.3	status_filter	17
5.3	collisionStruct Strukturreferenz	17
5.3.1	Ausführliche Beschreibung	17
5.3.2	Dokumentation der Datenelemente	17
5.3.2.1	affectedObject	17
5.3.2.2	causingObject	17
5.3.2.3	direction	17
5.4	compareGameObjects Strukturreferenz	18
5.4.1	Ausführliche Beschreibung	18
5.5	compareScores Strukturreferenz	18
5.5.1	Ausführliche Beschreibung	18
5.6	Enemy Klassenreferenz	19
5.6.1	Beschreibung der Konstruktoren und Destruktoren	20
5.6.1.1	Enemy	20
5.6.2	Dokumentation der Elementfunktionen	20
5.6.2.1	getHealth	20
5.6.2.2	setHealth	20
5.6.2.3	getInflictedDamage	21
5.6.2.4	getFireCooldown	21
5.6.2.5	getDeath	21
5.6.2.6	setDeath	21
5.6.2.7	getDeathCooldown	21
5.6.2.8	updateFramesDirection	21
5.6.2.9	flipHorizontal	22
5.6.2.10	swapImage	22
5.6.2.11	updatePosition	22
5.7	Game Klassenreferenz	22
5.7.1	Ausführliche Beschreibung	25
5.7.2	Beschreibung der Konstruktoren und Destruktoren	26
5.7.2.1	Game	26
5.7.2.2	~Game	26
5.7.3	Dokumentation der Elementfunktionen	26
5.7.3.1	step	26

5.7.3.2	start	27
5.7.3.3	setState	27
5.7.3.4	timerEvent	27
5.7.3.5	startNewGame	27
5.7.3.6	loadLevelFile	28
5.7.3.7	updateHighScore	28
5.7.3.8	displayStatistics	28
5.7.3.9	endGame	28
5.7.3.10	appendWorldObjects	29
5.7.3.11	reduceWorldObjects	29
5.7.3.12	evaluateInput	29
5.7.3.13	calculateMovement	30
5.7.3.14	detectCollision	30
5.7.3.15	handleCollisions	30
5.7.3.16	updateScore	30
5.7.3.17	updateAudioevents	30
5.7.3.18	renderGraphics	31
5.7.3.19	menuInit	31
5.7.3.20	exitGame	31
5.7.3.21	eventFilter	31
5.7.3.22	getStepIntervall	32
5.7.4	Dokumentation der Datenelemente	32
5.7.4.1	levelSpawn	32
5.8	GameObject Klassenreferenz	32
5.8.1	Beschreibung der Konstruktoren und Destruktoren	33
5.8.1.1	GameObject	33
5.8.1.2	GameObject	33
5.9	Input Klassenreferenz	33
5.9.1	Ausführliche Beschreibung	35
5.9.2	Beschreibung der Konstruktoren und Destruktoren	35
5.9.2.1	Input	35
5.9.2.2	~Input	35
5.9.3	Dokumentation der Elementfunktionen	35
5.9.3.1	evaluatekeyEvent	35
5.9.3.2	getKeyactions	36
5.9.3.3	getKeyletters	36
5.9.3.4	getLastKeyaction	36
5.9.3.5	getLastKeyletter	37
5.9.3.6	updateKeys	37
5.10	Menu Klassenreferenz	37

5.10.1	Ausführliche Beschreibung	39
5.10.2	Klassen-Dokumentation	39
5.10.2.1	struct Menu::menuEntry	39
5.10.3	Beschreibung der Konstruktoren und Destruktoren	39
5.10.3.1	Menu	39
5.10.4	Dokumentation der Elementfunktionen	39
5.10.4.1	clear	39
5.10.4.2	getType	40
5.10.4.3	getTitle	40
5.10.4.4	displayInit	40
5.10.4.5	displayUpdate	40
5.10.4.6	addEntry	40
5.10.4.7	changeSelection	41
5.10.4.8	getSelection	41
5.10.4.9	getEntry	41
5.10.4.10	selectFirstEntry	41
5.11	MovingObject Klassenreferenz	42
5.11.1	Dokumentation der Elementfunktionen	43
5.11.1.1	updateFramesDirection	43
5.11.1.2	flipHorizontal	43
5.11.1.3	swapImage	43
5.11.1.4	updatePosition	43
5.12	Player Klassenreferenz	44
5.12.1	Dokumentation der Elementfunktionen	46
5.12.1.1	getHealth	46
5.12.1.2	setHealth	46
5.12.1.3	receiveDamage	46
5.12.1.4	getAlcoholLevel	46
5.12.1.5	increaseAlcoholLevel	46
5.12.1.6	decreaseAlcoholLevel	46
5.12.1.7	getAmmunatiuon	47
5.12.1.8	getFireCooldown	47
5.12.1.9	getInflictedDamage	47
5.12.1.10	getImmunityCooldown	47
5.12.1.11	setImmunityCooldown	47
5.12.1.12	inJump	47
5.12.1.13	update	48
5.12.1.14	updateFramesDirection	48
5.12.1.15	flipHorizontal	48
5.12.1.16	swapImage	48

5.12.1.17	updatePosition	48
5.13	PowerUp Klassenreferenz	49
5.13.1	Ausführliche Beschreibung	50
5.13.2	Beschreibung der Konstruktoren und Destruktoren	50
5.13.2.1	PowerUp	50
5.13.2.2	~PowerUp	50
5.13.3	Dokumentation der Elementfunktionen	50
5.13.3.1	getHealthBonus	50
5.13.3.2	getAlcoholLevelBonus	50
5.13.3.3	getAmmunationBonus	51
5.13.3.4	getImmunityCooldownBonus	51
5.13.3.5	getPowerUPType	51
5.14	RenderBackground Klassenreferenz	51
5.14.1	Ausführliche Beschreibung	52
5.14.2	Beschreibung der Konstruktoren und Destruktoren	52
5.14.2.1	RenderBackground	52
5.14.3	Dokumentation der Elementfunktionen	52
5.14.3.1	setPos	52
5.14.3.2	updateParallaxe	53
5.14.3.3	updateBackgroundPos	54
5.15	RenderGUI Klassenreferenz	54
5.15.1	Ausführliche Beschreibung	55
5.15.2	Beschreibung der Konstruktoren und Destruktoren	55
5.15.2.1	RenderGUI	55
5.15.3	Dokumentation der Elementfunktionen	55
5.15.3.1	setPos	55
5.15.3.2	setValues	55
5.16	Shoot Klassenreferenz	56
5.16.1	Beschreibung der Konstruktoren und Destruktoren	57
5.16.1.1	Shoot	57
5.16.2	Dokumentation der Elementfunktionen	57
5.16.2.1	getInflictedDamage	57
5.16.2.2	getOrigin	58
5.16.2.3	updateFramesDirection	58
5.16.2.4	flipHorizontal	58
5.16.2.5	swapImage	58
5.16.2.6	updatePosition	58
6	Datei-Dokumentation	59
6.1	Wiesn-Run/src/definitions.h-Dateireferenz	59

6.1.1	Ausführliche Beschreibung	60
6.1.2	Klassen-Dokumentation	61
6.1.2.1	struct scoreStruct	61
6.1.2.2	struct audioCooldownStruct	61
6.1.2.3	struct audioDistanceStruct	62
6.1.2.4	struct audioStruct	63
6.1.2.5	struct audioCooldownstruct	63
6.1.2.6	struct stateStruct	63
6.1.3	Dokumentation der Aufzählungstypen	64
6.1.3.1	gameState	64
6.1.3.2	objectType	64
6.1.3.3	collisionDirection	64
6.1.3.4	audioType	64
6.1.4	Variablen-Dokumentation	65
6.1.4.1	spawnDistance	65
6.2	Wiesn-Run/src/portaudio.h-Dateireferenz	65
6.2.1	Ausführliche Beschreibung	69
6.2.2	Klassen-Dokumentation	69
6.2.2.1	struct PaHostApiInfo	69
6.2.2.2	struct PaHostErrorInfo	70
6.2.2.3	struct PaDeviceInfo	70
6.2.2.4	struct PaStreamParameters	71
6.2.2.5	struct PaStreamCallbackTimeInfo	72
6.2.2.6	struct PaStreamInfo	72
6.2.3	Makro-Dokumentation	73
6.2.3.1	paNoDevice	73
6.2.3.2	paUseHostApiSpecificDeviceSpecification	73
6.2.3.3	paFloat32	73
6.2.3.4	paInt32	73
6.2.3.5	paInt24	73
6.2.3.6	paInt16	73
6.2.3.7	paInt8	74
6.2.3.8	paUInt8	74
6.2.3.9	paCustomFormat	74
6.2.3.10	paNonInterleaved	74
6.2.3.11	paFormatIsSupported	74
6.2.3.12	paNoFlag	74
6.2.3.13	paClipOff	74
6.2.3.14	paDitherOff	74
6.2.3.15	paNeverDropInput	75

6.2.3.16	paPrimeOutputBuffersUsingStreamCallback	75
6.2.3.17	paPlatformSpecificFlags	75
6.2.3.18	paInputUnderflow	75
6.2.3.19	paInputOverflow	75
6.2.3.20	paOutputUnderflow	76
6.2.3.21	paOutputOverflow	76
6.2.3.22	paPrimingOutput	76
6.2.4	Dokumentation der benutzerdefinierten Typen	76
6.2.4.1	PaError	76
6.2.4.2	PaDeviceIndex	76
6.2.4.3	PaHostApiIndex	76
6.2.4.4	PaHostApiTypeId	77
6.2.4.5	PaHostApiInfo	77
6.2.4.6	PaTime	77
6.2.4.7	PaSampleFormat	77
6.2.4.8	PaDeviceInfo	77
6.2.4.9	PaStream	78
6.2.4.10	PaStreamFlags	78
6.2.4.11	PaStreamCallbackTimeInfo	78
6.2.4.12	PaStreamCallbackFlags	78
6.2.4.13	PaStreamCallbackResult	78
6.2.4.14	PaStreamCallback	79
6.2.4.15	PaStreamFinishedCallback	80
6.2.4.16	PaStreamInfo	80
6.2.5	Dokumentation der Aufzählungstypen	80
6.2.5.1	PaHostApiTypeId	80
6.2.5.2	PaStreamCallbackResult	80
6.2.6	Dokumentation der Funktionen	81
6.2.6.1	Pa_Initialize	81
6.2.6.2	Pa_Terminate	81
6.2.6.3	Pa_GetHostApiCount	81
6.2.6.4	Pa_GetDefaultHostApi	82
6.2.6.5	Pa_GetHostApiInfo	82
6.2.6.6	Pa_HostApiTypeIdToHostApiIndex	82
6.2.6.7	Pa_HostApiDeviceIndexToDeviceIndex	82
6.2.6.8	Pa_GetLastHostErrorInfo	83
6.2.6.9	Pa_GetDeviceCount	83
6.2.6.10	Pa_GetDefaultInputDevice	83
6.2.6.11	Pa_GetDefaultOutputDevice	84
6.2.6.12	Pa_GetDeviceInfo	84

6.2.6.13	Pa_IsFormatSupported	84
6.2.6.14	Pa_OpenStream	85
6.2.6.15	Pa_OpenDefaultStream	86
6.2.6.16	Pa_CloseStream	86
6.2.6.17	Pa_SetStreamFinishedCallback	86
6.2.6.18	Pa_StopStream	87
6.2.6.19	Pa_IsStreamStopped	87
6.2.6.20	Pa_IsStreamActive	87
6.2.6.21	Pa_GetStreamInfo	88
6.2.6.22	Pa_GetStreamTime	88
6.2.6.23	Pa_GetStreamCpuLoad	88
6.2.6.24	Pa_ReadStream	89
6.2.6.25	Pa_WriteStream	89
6.2.6.26	Pa_GetStreamReadAvailable	89
6.2.6.27	Pa_GetStreamWriteAvailable	90
6.2.6.28	Pa_GetStreamHostApiType	90
6.2.6.29	Pa_GetSampleSize	90
6.2.6.30	Pa_Sleep	90

Index	91
--------------	-----------

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:

Audio	9
AudioControl	12
audioCooldownstruct	59
audioCooldownStruct	59
audioDistanceStruct	59
audioStruct	59
collisionStruct	17
compareGameObjects	18
compareScores	18
Input	33
Menu	37
Menu::menuEntry	37
PaDeviceInfo	65
PaHostApiInfo	65
PaHostErrorInfo	65
PaStreamCallbackTimeInfo	65
PaStreamInfo	65
PaStreamParameters	65
AudioControl::playStruct	12
QGraphicsPixmapItem	
GameObject	32
MovingObject	42
Enemy	19
Player	44
Shoot	56
PowerUp	49
QObject	
Game	22
RenderBackground	51
RenderGUI	54
scoreStruct	59
stateStruct	59

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

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

Audio	Die Audio-Klasse erzeugt Audioobjekte	9
AudioControl	Die AudioControl-Klasse synchronisiert alle Audioausgabeanweisungen und spielt passende Audioobjekte ab	12
collisionStruct	Struktur für die Events Enthält affectedObject als Objekt, aus dessen Sicht die Kollision berechnet wurde	17
compareGameObjects	Vergleich zweier GameObjects bezüglich der X-Position	18
compareScores	Vergleich zweier Scores	18
Enemy	19
Game	Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels	22
GameObject	32
Input	Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben	33
Menu	Klasse zum Erzeugen und Anzeigen von Spielmenüs	37
MovingObject	42
Player	44
PowerUp	Klasse für Power-Ups	49
RenderBackground	Hintergrund-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt	51
RenderGUI	Anzeigen der Spielerwerte-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt	54
Shoot	56

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Wiesn-Run/src/ audio.h	??
Wiesn-Run/src/ audiocontrol.h	??
Wiesn-Run/src/ definitions.h	
Definitions beinhaltet Datentyp Definitionen	59
Wiesn-Run/src/ enemy.h	??
Wiesn-Run/src/ game.h	??
Wiesn-Run/src/ gameobject.h	??
Wiesn-Run/src/ input.h	??
Wiesn-Run/src/ menu.h	??
Wiesn-Run/src/ movingobject.h	??
Wiesn-Run/src/ player.h	??
Wiesn-Run/src/ portaudio.h	
The portable PortAudio API	65
Wiesn-Run/src/ powerup.h	??
Wiesn-Run/src/ renderbackground.h	??
Wiesn-Run/src/ renderGUI.h	??
Wiesn-Run/src/ shoot.h	??

Kapitel 5

Klassen-Dokumentation

5.1 Audio Klassenreferenz

Die Audio-Klasse erzeugt Audioobjekte.

```
#include <audio.h>
```

Öffentliche Methoden

- [Audio](#) (std::string type_name)
Konstruktor instanziert ein Objekt der Klasse [Audio](#).
- [~Audio](#) ()
Destruktor löscht ein Objekt der Klasse [Audio](#).
- std::string [getSource](#) ()
getSource gibt bei Aufruf den Namen des Objektes zurück welcher dem Pfad in der Ressourcendatenbank entspricht.
- float [getSample](#) (int pos)
getSample gibt bei Aufruf das Sample an Position = pos der zu Audioobjekt gehörigen Wave Datei mit Bittiefe 16 bit zurück.
- int [getSamplenumber](#) ()
getSamplenumber gibt bei Aufruf die Anzahl an Samples der zu Audioobjekt gehörigen Wave Datei zurück.

Private Methoden

- void [readSamples](#) ()
readSamples liest bei Aufruf alle Samples der zu Audioobjekt gehörigen Wave Datei in die Variable "samples" ein.
- quint16 [to16bitSample](#) (quint8 sample8bit)
to16bitSample konvertiert einen 8 bit integer Sample in einen 16 bit Integer Sample.
- void [normalize](#) ()
normalize normalisiert den 16 bit Integer QVector samples.

Private Attribute

- std::string [source](#)
source speichert den Namen des Audioobjekts als string welcher dem Dateinamen der zugehörigen Wave Datei entspricht.
- std::vector< float > [samples](#)
samples speichert die normalisierten samples des [Audio](#) Objekts als QVector mit 32 bit float Werten.

- int [samplenumber](#)

samplenumber speichert die Anzahl an Samples in der gesamten [Audio](#) Datei des [Audio](#) Objekts als Integer.

5.1.1 Ausführliche Beschreibung

Die Audio-Klasse erzeugt Audioobjekte.

Für jeden Audioobjekt Typ mit Name `type_name` wird zu Beginn eine Instanz der Klasse erstellt und in [AudioControl](#) an die Liste `audioobjects` angehängt. Jedes Audioobjekt liest die zum Objekt Typ gehörigen Audiosamples aus einer WAVE Datei ein und übergibt das Sample an Position `pos` per Aufruf mit [getSample\(int pos\)](#) an den Aufrufer.

Autor

Felix Pfreundtner

5.1.2 Beschreibung der Konstruktoren und Destrukturen

5.1.2.1 `Audio::Audio (std::string type_name)`

Konstruktor instanziert ein Objekt der Klasse [Audio](#).

Autor

Felix Pfreundtner

5.1.2.2 `Audio::~~Audio ()`

Destruktor löscht ein Objekt der Klasse [Audio](#).

Autor

Felix Pfreundtner

5.1.3 Dokumentation der Elementfunktionen

5.1.3.1 `std::string 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

5.1.3.2 `float Audio::getSample (int pos)`

`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

5.1.3.3 int Audio::getSamplenumber ()

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

Rückgabe

int Instanzvariable samplenumber

Autor

Felix Pfreundtner

5.1.3.4 void Audio::readSamples () [private]

readSamples liest bei Aufruf alle Samples der zu Audioobjekt gehörigen Wave Datei in die Variable "samples" ein. Eingelesen werden sollen RIFF Mono Wave Dateien mit 44100Hz Samplerate. Die Bittiefe ist hierbei variabel 8 oder 16bit. Es greift hierfür auf die zum Objekt gehörige, in der Ressourcendatenbank gespeicherte Wave Datei mit Pfadnamen "source" zurück. Die Funktion wertet den fmt Header des Wave File aus und liest im Anschluss den data Chunk ein. Die Bittiefe wird in float konvertiert um eine Weiterbearbeitung der Samples ohne Dynamikverlust durchführen zu können.

Autor

Felix Pfreundtner

lese den Namen des Headers des nächsten Chunks aus

5.1.3.5 qint16 Audio::to16bitSample (quint8 *sample8bit*) [private]

to16bitSample konvertiert einen 8 bit integer Sample in einen 16 bit Integer Sample.

Ziel ist eine einheitlich Bearbeitung der Samples verschiedener Audioobjekte vornehmen zu können.

Parameter

<i>quint8</i>	sample8bit
---------------	------------

Rückgabe

qint16 sample16bit

Autor

Felix Pfreundtner

5.1.3.6 void Audio::normalize () [private]

normalize normalisiert den 16 bit Integer QVector samples.

Es wird hierfür die größte Betrag-Amplitude eines Sample in samples bestimmt. Diese Amplitude wird auf den maximalen signed Integer 16 Bit Wert gesetzt. Alle anderen Samples werden entsprechend ihres Verhältnisses zur größten Betrag-Amplitude skaliert.

Autor

Felix Pfreundtner

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.2 AudioControl Klassenreferenz

Die AudioControl-Klasse synchronisiert alle Audioausgabeanweisungen und spielt passende Audioobjekte ab.

```
#include <audiocontrol.h>
```

Klassen

- struct [playStruct](#)

Öffentliche Methoden

- [AudioControl](#) ()
Konstruktor instanziert ein Objekt der Klasse [AudioControl](#).
- [~AudioControl](#) ()
Destruktor löscht ein Objekt der Klasse [AudioControl](#).
- void [playInitialize](#) ()
playInitialize initialisiert die Abspielbibliothek Portaudio, öffnet den PortAudio Stream pastream und startet eine Callback Audiowiedergabe
- void [playTerminate](#) ()
playTerminate stoppt bei Aufruf die PortAudio Audioausgabe, beendet im Anschluss den Portaudio Stream und beendet zuletzt PortAudio.
- void [updatePlayevents](#) (std::list< struct [audioStruct](#) > *audioevents)
updatePlayevents aktualisiert nach Aufruf über [Game::step](#) alle im Moment abgespielten, in der Liste "playevents" gespeicherten [playStruct](#)'s mit aktuellen [audioStruct](#)'s aus der übergebenen Liste audioevents.

Private Typen

- enum [statusFilter](#) { **no**, **alcohol**, **lifecritical** }
statusFilter definiert alle [Audio](#) Filter Status Optionen

Private Methoden

- int [instancepaCallback](#) (const void *input, void *output, unsigned long frameCount, const [PaStreamCallback-TimelInfo](#) *timeInfo, [PaStreamCallbackFlags](#) statusFlags)
instancepaCallback wird von Portaudio aufgerufen wenn nahezu letzter Audioblock abgespielt wurde und neu Audio-samples benötigt werden.

Private, statische Methoden

- static int [staticpaCallback](#) (const void *input, void *output, unsigned long frameCount, const [PaStreamCallbackTimeInfo](#) *timeInfo, [PaStreamCallbackFlags](#) statusFlags, void *userData)

staticpaCallback ist die Statische Callback Funktion der [AudioControl](#) Klasse.

Private Attribute

- std::mutex [mtx](#)
mtx ist eine Mutex, welche zwischen dem [Game](#) Thread und dem PortAudio Ausgabe Thread vermittelt.
- std::list< [playStruct](#) > [playevents](#)
playevents beinhaltet eine Liste mit allen im Moment abgespielten playStructs.
- std::vector< [Audio](#) > [audioobjects](#)
audioobjects beinhaltet eine Array mit allen vorhandenen Objekten der Klasse [Audio](#)(beispielsweise deren Samples als QVector).
- int [waitinms](#)
waitinms speichert die wartezeit bis zum Beenden von PortAudio in Millisekunden.
- [PaError](#) [playinitializeerror](#)
playinitializeerror speichert eventuell auftretende Error beim Öffnen und Schließen des PortAudio Streams.
- int [max_playevents](#)
max_playevents definiert die maximale Anzahl an abgespielten playEvents ohne Clipping Effekte.
- int [blockcounter](#)
blockcounter zählt die bereits abgespielten [Audio](#) Ausgabe Blöcke.
- float [mixed_sample](#)
mixed_sample beinhaltet das aktuell von mixSample() gemixte Sample aller audioEvents.
- int [playeventsnumber](#)
playeventsnumber beinhaltet die Anzahl an aktuelle abzuspielenden audioEvents.
- [PaStream](#) * [pastream](#)
pastream ist ein Zeiger auf den PortAudio Stream.
- [PaError](#) [paerror](#)
paerror speichert einen eventuellen PortAudio Error.
- int [status_filter](#)
status_filter gibt den Filterstatus an.

5.2.1 Ausführliche Beschreibung

Die AudioControl-Klasse synchronisiert alle Audioausgabeeinstellungen und spielt passende Audioobjekte ab.

Eine Instanz dieser Klasse wird innerhalb der [game.h](#) angelegt.

Autor

Felix Pfreundtner

5.2.2 Klassen-Dokumentation

5.2.2.1 struct AudioControl::playStruct

Klassen-Elemente

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

5.2.3 Beschreibung der Konstruktoren und Destruktoren

5.2.3.1 [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

Quelle scene_flyingbeer: <http://soundbible.com/1247-Wind.html>

Quelle scene_enemy_tourist: <http://www.freesound.org/people/Reitanna/sounds/241215/>

Quelle scene_enemy_security: <http://www.freesound.org/people/Robinhood76/sounds/195414/>

Quelle scene_enemy_boss: <http://soundbible.com/1501-Buzzer.html>

Quelle scene_collision_obstacle: <http://soundbible.com/1522-Balloon-Popping.html>

Quelle scene_collision_enemy: <http://www.freesound.org/people/qubodup/sounds/169725/>

Quelle scene_collision_player: <http://www.freesound.org/people/thecheeseman/sounds/44430/>

Quelle scene_collision_flyingbeer: <http://helios.augustana.edu/~dr/105/wav/glasbk.wav>

Quelle powerup_beer: <http://www.freesound.org/people/edhutschek/sounds/215634/>

Quelle powerup_food: <https://www.freesound.org/people/bassboybg/sounds/264544/>

Quelle status_alcohol: <http://www.freesound.org/people/afleetingspeck/sounds/151180/>

Quelle status_life: <http://soundbible.com/1612-Slow-HeartBeat.html>

Quelle status_lifecritical: <http://soundbible.com/1612-Slow-HeartBeat.html>

Quelle status_dead: <http://www.freesound.org/people/Robinhood76/sounds/256469/>

Quelle player_walk: http://www.arts.rpi.edu/public_html/ruiz/VES01/sebram/final/walk-_crop.wav

Quelle player_jump: <http://soundbible.com/266-Boing-Cartoonish.html>

Quelle background_menu: <http://www.theholidayspot.com/oktoberfest/music/Octoberfest%20–%20Beerdrinking%20song%28Bavarian%29.wma>

Quelle background_highscore: <http://soundbible.com/1563-Pacman-Introduction-Music.-html>

Quelle background_level1: <http://soundbible.com/1763-Ambience-Casino.html>

Quelle background_level2: <http://www.freesound.org/people/Kyster/sounds/122789/>

Quelle background_level3: <http://www.freesound.org/people/Westmed/sounds/239538/>

Quelle background_startgame: <http://www.freesound.org/people/Harbour11/sounds/194625/>

Quelle background_levelfinished: <http://soundbible.com/1823-Winning-Triumphal-Fanfare.-html>

5.2.3.2 AudioControl::~~AudioControl ()

Destruktor löscht ein Objekt der Klasse [AudioControl](#).

Autor

Felix Pfreundtner

5.2.4 Dokumentation der Elementfunktionen

5.2.4.1 void AudioControl::playInitialize ()

playInitialize initialisiert die Abspielbibliothek Portaudio, öffnet den PortAudio Stream pstream und startet eine Callback Audiowiedergabe

Autor

Felix Pfreundtner

5.2.4.2 void AudioControl::playTerminate ()

playTerminate stoppt bei Aufruf die PortAudio Audioausgabe, beendet im Anschluss den Portaudio Stream und beendet zuletzt PortAudio.

Autor

Felix Pfreundtner

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

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

Parameter

<i>std::list<struct</i>	audioStruct > *audioevents
----------------------------	--

Autor

Felix Pfreundtner

5.2.4.4 int AudioControl::instancepaCallback (const void * inputBuffer, void * outputBuffer, unsigned long framesPerBuffer, const PaStreamCallbackTimeInfo * timeInfo, PaStreamCallbackFlags statusFlags) [private]

instancepaCallback wird von Portaudio aufgerufen wenn nahezu letzter Audioblock abgespielt wurde und neu Audiosamples benötigt werden.

Parameter

<i>const</i>	void *inputBuffer
<i>void</i>	*outputBuffer
<i>unsigned</i>	long framesPerBuffer,
<i>const</i>	PaStreamCallbackTimeInfo* timeInfo,
<i>PaStream-CallbackFlags</i>	statusFlags

Rückgabe

int returncode

Autor

Felix Pfreundtner

5.2.4.5 static int AudioControl::staticpaCallback (const void * *input*, void * *output*, unsigned long *frameCount*, const PaStreamCallbackTimeInfo * *timeInfo*, PaStreamCallbackFlags *statusFlags*, void * *userData*)
[inline], [static], [private]

staticpaCallback ist die Statische Callback Funktion der [AudioControl](#) Klasse.

Die Funktion wird immer dann aufgerufen, wenn der PortAudio Stream einen neuen Ausgabeblock benötigt, da der letzte abgespielt wurde. Die Funktion ruft die Funktion instancepaCallback auf, welche nicht statisch ist und auf alle instance variablen und Funktionen (des von [Game](#) erzeugten [AudioControl](#) Objektes audioOutput) zugreifen kann. Dies ermöglicht einen Einfachen Austausch von [Audio](#) Blöcken zwischen [Game](#) Thread und Portaudio Wiedergabethread.

Parameter

<i>const</i>	void *inputBuffer
<i>void</i>	*outputBuffer
<i>unsigned</i>	long framesPerBuffer,
<i>const</i>	PaStreamCallbackTimeInfo* timeInfo,
<i>PaStream-CallbackFlags</i>	statusFlags

Rückgabe

((AudioControl*)userData) ->instancepaCallback(input, output, frameCount, timeInfo, statusFlags)

Autor

Felix Pfreundtner

5.2.5 Dokumentation der Datenelemente

5.2.5.1 std::mutex AudioControl::mtx [private]

mtx ist eine Mutex, welche zwischen dem [Game](#) Thread und dem PortAudio Ausgabe Thread vermittelt.

Es muss die gleichzeitig von [Game](#) über [updatePlayevents\(\)](#) beschriebene und PortAudio über [instancepaCallback\(\)](#) gelesene Liste playevents gelockt werden.

5.2.5.2 int AudioControl::playeventsnumber [private]

playeventsnumber beinhaltet die Anzahl an aktuelle abzuspielenden audioEvents.

Float Format da mit diesem Wert in mixsamples effizient gerechnet werden muss ohne Castumwandlung Integer in Float.

5.2.5.3 AudioControl::status_filter [private]

status_filter gibt den Filterstatus an.

Wenn kein Audioevent in der audiovents List den Type status_alcohol hat -> enum none-> 0. Wenn mindestens ein Audioevent in der audiovents List den Type status_alcohol hat -> enum alcohol-> 1. Wenn mindestens ein Audioevent in der audiovents List den Type status_life hat -> enum alcohol-> 2. Wenn mindestens ein Audioevent in der audiovents List den Type status_lifecritical hat -> enum alcohol-> 3.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.3 collisionStruct Strukturreferenz

Struktur für die Events Enthält affectedObject als Objekt, aus dessen Sicht die Kollision berechnet wurde.

```
#include <game.h>
```

Öffentliche Attribute

- [GameObject](#) * **affectedObject**
- [GameObject](#) * **causingObject**
- enum [collisionDirection](#) **direction**

5.3.1 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

5.3.2 Dokumentation der Datenelemente

5.3.2.1 GameObject* collisionStruct::affectedObject

5.3.2.2 GameObject* collisionStruct::causingObject

5.3.2.3 enum collisionDirection collisionStruct::direction

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.h

5.4 compareGameObjects Strukturreferenz

Vergleich zweier GameObjects bezüglich der X-Position.

Öffentliche Methoden

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

5.4.1 Ausführliche Beschreibung

Vergleich zweier GameObjects bezüglich der X-Position.

Die Methode std::list::sort benötigt 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

5.5 compareScores Strukturreferenz

Vergleich zweier Scores.

Öffentliche Methoden

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

5.5.1 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

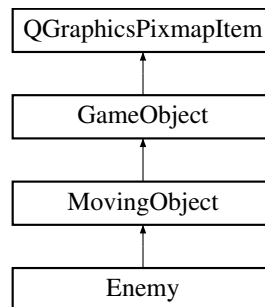
Simon

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- Wiesn-Run/src/game.cpp

5.6 Enemy Klassenreferenz

Klassendiagramm für Enemy:



Öffentliche Methoden

- **Enemy** (int posX, int posY, int speedX, objectType enemy)
 - Class Enemy lastUpdate: update() 10.6 Johann.*
- 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.*
- void **setPosX** (int posX)
- void **setPosY** (int posY)
- int **getSpeedX** () const
- int **getSpeedY** () const
- void **setSpeedX** (int speedX)
- void **setSpeedY** (int speedY)
- void **updateFramesDirection** ()
 - aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).*
- void **flipHorizontal** ()
 - spiegelt Grafiken an der Y-Achse kopiert von "https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2" und angepasst.*
- void **swapImage** ()
 - MovingObject::swapImage Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.*
- int **getPosX** () const

- int **getPosY** () const
- int **getLength** () const
- int **getHeight** () const
- [objectType](#) **getType** () const
- void **setAudioID** (int audioID)
- int **getAudioID** () const

Geschützte Methoden

- void [updatePosition](#) ()
überschreibt die X und Y Position gemäß SpeedXY.

Geschützte Attribute

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

Private Attribute

- int **health**
- int **fireRate**
- int **fireCooldown**
- int **inflictedDamage**
- bool **death**
- int **DeathCooldown**

5.6.1 Beschreibung der Konstruktoren und Destruktoren

5.6.1.1 Enemy::Enemy (int *posX*, int *posY*, int *speedX*, [objectType](#) *enemy*)

Class [Enemy](#) lastUpdate: [update\(\)](#) 10.6 Johann.

Konstruktor für ein Enemy-Objekt

Parameter

<i>posX</i>	: X-Position
<i>posY</i>	: Y-Position
<i>speedX</i>	: Geschwindigkeit in X-Richtung

[Noch zu erledigen](#) Skalieren der Werte und fireCooldown erhöhen

5.6.2 Dokumentation der Elementfunktionen

5.6.2.1 int Enemy::getHealth () const

[Enemy::getHealth](#) Gibt Lebensstand zurück.

Rückgabe

: Lebensstand

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

[Enemy::setHealth](#) Lebensstand wird gesetzt.

Parameter

<i>health</i>	: Lebensstand
---------------	---------------

5.6.2.3 int Enemy::getInflictedDamage () const

[Enemy::getInflictedDamage](#) gibt Schaden zurück, den der gegner zufügt.

Rückgabe

: Schaden

5.6.2.4 int Enemy::getFireCooldown () const

[Enemy::getFireCooldown](#).

Rückgabe

fireCooldown

5.6.2.5 bool Enemy::getDeath () const

[Enemy::getDeath](#) Gibt an ob der Gegner Tot ist.

Rückgabe

: Zustand - TOT

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

[Enemy::setDeath](#) Zustand-TOT wird gesetzt.

Parameter

<i>death</i>	: Zustand-TOT
--------------	---------------

5.6.2.7 int Enemy::getDeathCooldown () const

[Enemy::getDeathCooldown](#).

Rückgabe

deathCooldown

5.6.2.8 void MovingObject::updateFramesDirection () [inherited]

aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).

So lässt sich auch die Richtung abfragen (> || < als 0) und mit dem aktuellen speedX-Wert ein Richtungswechsel feststellen

Autor

Flo

5.6.2.9 void MovingObject::flipHorizontal () [inherited]

spiegelt Grafiken an der Y-Achse kopiert von "https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2" und angepasst.

Ermöglicht das Spiegeln von Bildern über eine Transformationsmatrix. Am Anfang wird getestet ob ein Richtungswechsel statt gefunden hat.

Autor

Flo

5.6.2.10 void MovingObject::swapImage () [inherited]

[MovingObject::swapImage](#) Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.

Es wird alle framRate/2 Frames gewechselt und sofort beim loslaufen. Wenn der Spieler in der Luft ist bzw. springt setzt die Animation aus Flo

5.6.2.11 void MovingObject::updatePosition () [protected],[inherited]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

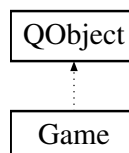
- Wiesn-Run/src/enemy.h
- Wiesn-Run/src/enemy.cpp

5.7 Game Klassenreferenz

Game-Klasse Die Game-Klasse bündelt alle Kern-Funktionalitäten des Spiels.

```
#include <game.h>
```

Klassendiagramm für Game:



Öffentliche Methoden

- [Game](#) (int argc, char *argv[])

Konstruktor und Destruktor.

- `~Game ()`

Destruktor.

- `int step ()`

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

- `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`

Liste von Kollisionen.

Geschützte Methoden

- `void timerEvent (QTimerEvent *event)`

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

Private Typen

- `enum menuIds {
 menuId_NonClickable, menuStartId_NewGame, menuStartId_EndGame, menuStartId_Help,
 menuStartId_Credits, menuCreditsId_Back, menuLevelId_Back, menuLevelId_Demo,
 menuLevelId_Level1, menuLevelId_Level2, menuLevelId_Level3, menuLevelId_StartGame,
 menuBreakId_Resume, menuBreakId_EarlyEnd, menuBreakId_EndGame, menuStatisticsId_Next,
 menuNameId_Next, menuHighscoreId_Next, menuHelpId_Back }`

zur Unterscheidung und Identifizierung der Menü-Einträge

Private Methoden

- `void startNewGame (QString levelFileName, int levelNum)`
Level starten und beenden.
- `void loadLevelFile (QString fileSpecifier)`
Level-Datei auslesen.
- `void updateHighScore (std::string mode)`
Game::updateHighScore Diese Funktion liest und aktualisiert die Highscore des Spiels.
- `void displayStatistics ()`
füllt das Statistik- und HighscoreMenü löscht das Statistik- und Highscore-Menü und füllt es mit aktuellen Werten
- `void endGame ()`
Game::endGame Diese Funktion löscht nicht mehr nötige Variablen und Objekte wenn vom Spiel in das Statistikmenü gewechselt wird.
- `void appendWorldObjects (Player *playerPointer)`
Funktionen in der Loop.
- `void reduceWorldObjects (Player *playerPointer)`
Game::reduceWorldObjects.
- `void evaluateInput ()`

- Checkt welche Tasten für die Spielkontrolle gedrückt sind mögliche Tasten:*

 - void `calculateMovement` ()

Geht die worldObjects durch und aktualisiert bei jedem die Position, Gegner bei denen der DeathCooldown abgelaufen ist, werden zum löschen vorgemerkt, Gegner bei denen der FireCooldown abgelaufen ist feuern.
 - void `detectCollision` (std::list< `GameObject` * > *objectsToCalculate)

`Game::detectCollision` Diese Funktion berechnet, ob Kollisionen zwischen benachbarten Objekten auftreten und falls ja, aus welcher Richtung diese stattfinden.
 - void `handleCollisions` ()

Kollisionen in der Liste collisionsToHandle werden der Reihe nach aus Sicht des affectedObjects bearbeitet.
 - void `updateScore` ()

`Game::updateScore` Aktualisiert die Score des Spielers.
 - void `updateAudioevents` ()

`Game::updateAudioevents` Übergibt vom Spiel erzeugte Audioevents an den Output Audioevents der Hintergrundmusik für das entsprechende Level werden übergeben, Überprüfen, ob sich der Spieler in einem Kritischen Zustand befindet und entsprechende Audioevents übergeben.
 - void `renderGraphics` (std::list< `GameObject` * > *objectList, `Player` *playerPointer)

`Game::renderGraphics`.
 - void `menuInit` ()

Funktionen zu Start und Ende der Applikation.
 - void `exitGame` ()

Diese Funktion wird aufgerufen wenn das Programm beendet werden soll.
 - bool `eventFilter` (QObject *obj, QEvent *event)

eventFilter wird aufgerufen, wenn ein neues QEvent auftritt.
 - int `getStepIntervall` ()

Hilfsfunktion.
 - void `timeNeeded` (string name)

Private Attribute

- std::list< `GameObject` * > `worldObjects`

In der Welt befindliche Objekte.
- std::list< `GameObject` * > `levelSpawn`

Statische Objekte, die zu Anfang gespawnt werden.
- std::list< `GameObject` * > `objectsToDelete`

Zu löschende Schüsse.
- `AudioControl` * `audioOutput`

Audiocontrol Objekt welches aktuelle Audioevents auswertet.
- std::thread `portaudiothread`

`Audio` Wiedergabe Thread welcher Portaudio Callback Funktion ausführt und Audioevents Blockweise abspielt.
- std::list< struct `audioStruct` > `audioevents`

Liste audioevents mit allen im Step stattfindenden AudioStructs.
- std::list< struct `audioCooldownstruct` > `audioStorage`

Liste mit den Audioevents die einmal aufgerufen werden aber eine Längere Spielzeit haben.
- int `sceneWidth`

Breite der Szene.
- int `levelLength` = 0

Länge des Levels.
- std::list< struct `scoreStruct` > `scoreList`
- struct `scoreStruct` `playerScore`
- int `stepIntervall`

Länge eines Steps.

- bool `exitGameevent`
Spiel Beenden gedrückt.
- bool `levelStartevent`
Spiel Starten gedrückt.
- `Player` * `playerObjPointer`
- `QGraphicsView` * `window`
für das Ausgabefenster QGraphicsView und QGraphicsScene der Level
- `QGraphicsScene` * `levelScene`
- `RenderGUI` * `showGUI`
für alle Anzeigen wie Leben,Alkohol,Score,..
- `RenderBackground` * `showBackground`
für die Hintergrundgrafiken
- `QApplication` * `appPointer`
Zeiger auf QApplication.
- `std::chrono::high_resolution_clock::time_point` `letzterAufruf`
für Zeitmessung
- `Input` * `keyInput` = new `Input()`
Erstelle Input Objekt zum Aufzeichnen der Keyboard Inputs.
- enum `gameState` `state` = `gameMenuStart`
- `Menu` * `aktMenu` = `menuStart`
aktueller Spielzustand
- `Menu` * `menuStart`
aktuell aktives Menü, null während das Spiel läuft; wird in setState gesetzt
- `Menu` * `menuCredits`
- `Menu` * `menuLevel`
- `Menu` * `menuBreak`
- `Menu` * `menuStatistics`
- `Menu` * `menuName`
- `Menu` * `menuHighscore`
- `Menu` * `menuHelp`
- int `stepCount` = 0
*stepCount wird mit jedem Step um ein erhöht Auslesen der vergangenen Zeit: stepCount * `getStepIntervall()`*
- int `audioIDs`
- `audioCooldownStruct` `audioCooldown`
- `audioDistanceStruct` `audioDistance`
- `chrono::high_resolution_clock::time_point` `thisStep`
- `chrono::high_resolution_clock::time_point` `testStep`

5.7.1 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.

funktion handleCollisions hinzugefügt

Autor

Simon, Johann, Felix, Rupert

5.7.2 Beschreibung der Konstruktoren und Destruktoren

5.7.2.1 `Game::Game (int argc, char * argv[])`

Konstruktor und Destruktor.

Konstruktor: Initialisiert den appPointer.

Autor

Rupert

5.7.2.2 `Game::~~Game ()`

Destruktor.

- Gibt verwendeten Heap-Speicher wieder frei.

5.7.3 Dokumentation der Elementfunktionen

5.7.3.1 `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` auslesen
- 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

Noch zu erledigen Erfolgreich Schriftzug einfügen

Noch zu erledigen GameOver schriftzug einfügen

5.7.3.2 int Game::start ()

Starten der Applikation.

Die Startfunktion, erstellt Fenster und Menüs, wird von main() aufgerufen.

Grafik (Flo): Es wird ein QGraphicsView Widget "window" angelegt in der Größe 1024x768 angelegt welches Das Spiel visualisiert. Verschiedene Einstellungen werden vorgenommen wie zb. das deaktivieren der Scrollbars.

Input (Felix): Erstelle QApplication 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("windwo") 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, Flo

5.7.3.3 void Game::setState (enum gameState newState)

Hilfsfunktion.

setzt den Spielstatus

Parameter

<i>newState</i>	
-----------------	--

Autor

Rupert

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

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

Parameter

<i>event</i>	
--------------	--

Autor

Rupert, Felix

5.7.3.5 void Game::startNewGame (QString levelFileName, int levelNum) [private]

Level starten und beenden.

Startet neues Spiel.

- lädt Leveldatei
- füllt worldobjects
- LevelScene wird eingestellt und aktiv geschaltet
- verschiedene Grafikinitialisierungen

5.7.3.6 void Game::loadLevelFile (QString *fileSpecifier*) [private]

Level-Datei auslesen.

Parameter

<i>fileSpecifier</i>	Diese Funktion liest Level-Dateien aus. In der Leveldatei werden Keywords für die anzulegen- den Objekte verwendet. Nach den Objekten stehen durch Kommata getrennt die benötigten Parameter. Ein Player-Eintrag enthält posX und posY. Ein Enemy-Eintrag enthält posX, posY und speedX. Ein Obstacle-Eintrag enthält posX und posY. Ein Plane-Eintrag (Zwischene- bene) enthält posX und posY. Ein PowerUp-Eintrag enthält posX, posY und die jeweiligen Boni.
----------------------	--

Autor

Simon

5.7.3.7 void Game::updateHighScore (std::string *mode*) [private]

[Game::updateHighScore](#) Diese Funktion liest und aktualisiert die Highscore des Spiels.

Als Parameter wird ein std::string mode erwartet. Ist der mode = "write", so wird die aktuelle Highscore unter Berücksichtigung der aktuellen playerScore neu geschrieben. Alle anderen Werte für mode lesen nur die alte Highscore und die des Spielers in die Liste ein, um sie z.B. im Highscore-Menü anzuzeigen. Dazu wird versucht, die Datei "wiesnHighscore.txt" auszulesen. Ist dies nicht möglich, so wurde das Spiel in dem aktuellen Verzeichnis noch nie gestartet. Falls die Datei gefunden und gelesen werden kann, so wird jeder Highscore-Eintrag in die scoreList aufgenommen. Anschließend wird die Liste nach der Summe der Punkte absteigend sortiert, und nur die 10 besten Elemente werden gespeichert. Wurde für das aktuelle Spiel eine Score angelegt und in der scoreList gespeichert, so wird dieser Eintrag eingeordnet und gegebenenfalls auch abgespeichert.

Autor

Simon

5.7.3.8 void Game::displayStatistics () [private]

füllt das Statistik- und HighscoreMenü löscht das Statistik- und Highscore-Menü und füllt es mit aktuellen Werten

Autor

Rupert

5.7.3.9 void Game::endGame () [private]

[Game::endGame](#) Diese Funktion löscht nicht mehr nötige Variablen und Objekte wenn vom Spiel in das Statistikmenü gewechselt wird.

es werden auch die Statistik und Highscoremenüs aktualisiert

Autor

: Felix, Johann

5.7.3.10 void Game::appendWorldObjects (Player * *playerPointer*) [private]

Funktionen in der Loop.

[Game::appendWorldObjects.](#)

Parameter

<i>playerPointer</i>	Diese Funktion fügt der Spielwelt dynamisch Gegner hinzu. In jedem Zeitschritt wird die sortierte Liste levelSpawn vom Anfang her durchlaufen. Ist die Distanz des Spielers zum Gegner kleiner als die Distanz levelSpawn, so wird das Objekt den worldObjects hinzugefügt und aus levelSpawn gelöscht. Die for-Schleife läuft solange, bis das erste Mal ein Objekt weiter als levelSpawn vom Spieler entfernt ist. Dann wird abgebrochen, da alle folgenden Objekte auf Grund der Sortierung noch weiter entfernt sein werden. Hier werden auch die Objekte der levelScene hinzugefügt.
----------------------	---

Autor

Simon

5.7.3.11 void Game::reduceWorldObjects (Player * *playerPointer*) [private]

[Game::reduceWorldObjects.](#)

Parameter

<i>playerPointer</i>	Alle Objekte aus der Liste objectsToDelete werden in der worldObjects gesucht und entfernt. Ihr Speicher wird wieder freigegeben. Die Funktion reduceWorldObjects löscht die Game-Objects und gibt den Speicher wieder frei, von denen der Spieler bereits weiter rechts als die spawnDistance entfernt ist.
----------------------	--

Autor

Simon, Johann

5.7.3.12 void Game::evaluateInput () [private]

Checkt welche Tasten für die Spielkontrolle gedrückt sind mögliche Tasten:

- Pfeil rechts zum laufen
- Pfeil hoch zum springen
- Leertaste zum schießen
- ESC für Menü

Autor

Rupert

5.7.3.13 void Game::calculateMovement () [private]

Geht die worldObjects durch und aktualisiert bei jedem die Position, Gegner bei denen der DeathCooldown abgelaufen ist, werden zum löschen vorgemerkt, Gegner bei denen der FireCooldown abgelaufen ist feuern.

wird momentan auch über Debug ausgegeben

Autor

Rupert, Johann

5.7.3.14 void Game::detectCollision (std::list< GameObject * > * objectsToCalculate) [private]

[Game::detectCollision](#) Diese Funktion berechnet, ob Kollisionen zwischen benachbarten Objekten auftreten und falls ja, aus welcher Richtung diese stattfinden.

Da die Liste worldObjects in jedem Zeitschritt sortiert wird, müssen die Kollisionen nur für die nächsten Nachbarn berechnet werden. Allerdings können durch ungünstige Lage auch Objekte kollidieren, die nicht direkt nebeneinander in der Liste liegen. Dafür werden die fünf Nachbarn links und rechts jedes MovingObjects geprüft, falls vorhanden.

Autor

Simon

5.7.3.15 void Game::handleCollisions () [private]

Kollisionen in der Liste collisionsToHandle werden der Reihe nach aus Sicht des affectedObjects bearbeitet.

In einer Schleife wird das jeweils erst CollisionEvent bearbeitet. Dabei werden nur an dem Objekt affectedObject Änderungen vorgenommen. Mögliche Objekte: Spieler(player), Gegner(enemy), Bierkrug(shot) mögliche Kollision mit Spieler(player), Hindernis(obstacle), Gegner(enemy), Bierkrug(shot), Power-Up(powerUp)

Autor

Johann (15.6.15)

5.7.3.16 void Game::updateScore () [private]

[Game::updateScore](#) Aktualisiert die Score des Spielers.

Diese Score wird von der Grafik während des Spiels ausgegeben und am Ende des Spiels in die Highscore aufgenommen.

Autor

Simon

5.7.3.17 void Game::updateAudioevents () [private]

[Game::updateAudioevents](#) Übergibt vom Spiel erzeugte Audioevents an den Output Audioevents der Hintergrundmusik für das entsprechende Level werden übergeben, Überprüfen, ob sich der Spieler in einem Kritischen Zustand befindet und entsprechende Audioevents übergeben.

Für Gegner und fliegende Bierkrüge Audioevents übergeben. Bei einmaligen Audioevents die Restspielzeit aktualisieren und das Event übergeben

Autor

Johann, Felix

5.7.3.18 `void Game::renderGraphics (std::list< GameObject * > * objectList, Player * playerPointer)` [private]

[Game::renderGraphics](#).

Parameter

<i>objectList</i>	
<i>playerPointer</i>	

Autor

Flo

5.7.3.19 `void Game::menuInit ()` [private]

Funktionen zu Start und Ende der Applikation.

Initialisierung der Menüs.

wird in [start\(\)](#) aufgerufen

Logik: Startmenü Credits Levelauswahl spielen... Pause Name eingeben Spielstatistik Highscore Von vorne

Autor

Rupert

5.7.3.20 `void Game::exitGame ()` [private]

Diese Funktion wird aufgerufen wenn das Programm beendet werden soll.

Autor

: Felix

5.7.3.21 `bool Game::eventFilter (QObject * obj, QEvent * event)` [private]

eventFilter wird aufgerufen, wenn ein neues QEvent auftritt.

Diese Funktion überwacht die Betätigung von Tastatur Eingaben und handelt den Aufruf des QT Schließ-Button (x) im Spielfenster. Die Tastatureingaben werden über das keyInput Objekt ausgewertet. Der Aufruf des QT Schließ-Button (x) ist neben dem Aufruf des Hauptmenüeintrags Exit die 2. Möglichkeit das Spiel zu beenden. Wird ein CloseEvent festgestellt wird die Variable exitGameevent auf False gesetzt und das Spiel zum Ende des aktuellen Steps in [Game::timerEvent](#) beendet.

Parameter

	QObject * <i>obj</i>
	QEvent * <i>event</i>

Rückgabe

: **QObject**::eventFilter(*obj*, *event*)

Autor

: Felix

5.7.3.22 `int Game::getStepIntervall () [private]`

Hilfsfunktion.

gibt stepIntervall zurück wird für Zeit auslesen gebraucht

Rückgabe

int Stepintervall in ms

Autor

Rupert

5.7.4 Dokumentation der Datenelemente

5.7.4.1 `std::list<GameObject*> Game::levelSpawn [private]`

Statische Objekte, die zu Anfang gespawnt werden.

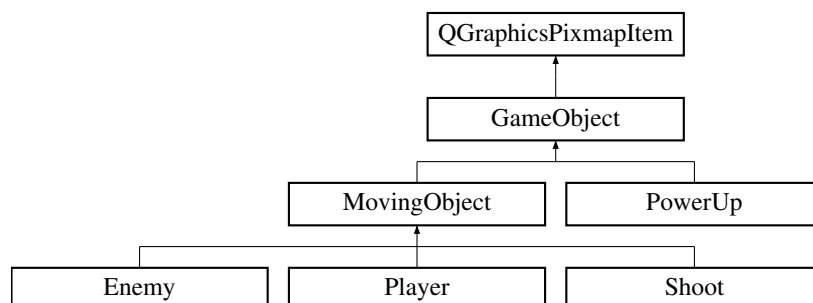
Objekte die zur Laufzeit dynamisch gespawnt werden

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.8 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)`
GameObject Konstruktor Jedes Objekt bekommt ihr seine Grafik zugewiesen und die "Startposition" wird in Szenen-koordinaten errechnet.
- `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**

Private Attribute

- int **length**
- int **height**
- **objectType** type
- int **audioID**

5.8.1 Beschreibung der Konstruktoren und Destruktoren

5.8.1.1 `GameObject::GameObject (int posX, int posY, int length, int height, objectType type)`

`GameObject::GameObject` Konstruktor.

Parameter

<i>length</i>	: Länge
<i>height</i>	: Höhe
<i>type</i>	: Typ
<i>posX</i>	: X-Position
<i>posY</i>	: Y-Position
<i>colType</i>	: Kollisionstyp

Autor

Johann

5.8.1.2 `GameObject::GameObject (int posX, int posY, objectType type)`

`GameObject` Konstruktor Jedes Objekt bekommt ihr seine Grafik zugewiesen und die "Startposition" wird in Szenenkoordinaten errechnet.

Parameter

<i>posX</i>	: X-Position
<i>posY</i>	: Y-Position
<i>type</i>	: Typ

Autor

Johann, Flo

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.9 Input Klassenreferenz

Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben.

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

Öffentliche Typen

- enum [Keyaction](#) {
noKeyaction, Right, Up, Down,
Jump_Right, Shoot, Exit, Enter }
Keyaction definiert alle auszuwertenden Tastenkombinationen Bezeichner.
- 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)'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', 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)'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' }
Keyletter definiert alle auszuwertenden Tastatur Buchstaben.

Öffentliche Methoden

- [Input](#) ()
Konstruktor instanziert ein Objekt der Klasse [Input](#).
- [~Input](#) ()
Destruktor löscht ein Objekt der Klasse [Input](#).
- void [evaluateKeyEvent](#) (QEvent *event)
Nach Aufruf über [Game::eventFilter](#) wertet evaluateKeyEvent alle im Moment gleichzeitig gepressten Tastatur Eingaben aus und speichert die zugehörigen enum ids in der Instanzvariable keyevents.
- QSet< int > [getKeyactions](#) ()
getKeyactions gibt bei Aufruf das QSet keyactions zurück, welches alle im Moment gedrückten Spielaktionen als Enum beinhaltet.
- std::set< char > [getKeyletters](#) ()
getKeyletters gibt bei Aufruf das QSet keyletters zurück, welches alle im Moment gedrückten Buchstaben als Enum beinhaltet.
- [Keyaction](#) [getLastKeyaction](#) ()
Gibt letzte gedrückte Spielaktion als Enum Keyaction zurück und setzt die Variable lastKeyaction auf noKeyaction.
- [Keyletter](#) [getLastKeyletter](#) ()
Gibt letzten gedrückten Buchstaben als enum Keyletter zurück und setzt die Variable lastKeyletter auf noKeyletter.

Private Methoden

- void [updateKeys](#) ()
updateKeyactions berechnet aus allen in keyevents gespeicherten Tastatureingaben die für das Spiel relevanten Kombinationen und speichert diese in keyactions.

Private Attribute

- QSet< int > [keyevents](#)
keyevents speichert die id aller im Moment gepressten Tasten.

- `QSet< int > keyactions`

Die Variable keyactions speichert die id aller im Moment gepressten Tastenkombinationen, welche für das Spiel relevant sind.

- `std::set< char > keyletters`

Die Variable keyletters speichert die Buchstababen als "strings" aller im Moment gepressten Buchstaben Tasten.

- `Keyaction lastKeyaction`

Die Variable lastKeyaction speichert die letzte gedrückte Tastenkombination als Enum Keyaction.

- `Keyletter lastKeyletter`

Die Variable lastKeyletter speichert den letzten gedrückten Buchstaben als Enum Keyletter.

5.9.1 Ausführliche Beschreibung

Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben.

Eine Instanz dieser Klasse wird innerhalb der `game.h` angelegt.

Autor

Felix Pfreundtner

5.9.2 Beschreibung der Konstruktoren und Destruktoren

5.9.2.1 `Input::Input ()`

Konstruktor instanziert ein Objekt der Klasse `Input`.

Autor

Felix Pfreundtner

5.9.2.2 `Input::~~Input ()`

Destruktor löscht ein Objekt der Klasse `Input`.

Autor

Felix Pfreundtner

5.9.3 Dokumentation der Elementfunktionen

5.9.3.1 `void Input::evaluateKeyEvent (QEvent * event)`

Nach Aufruf über `Game::eventFilter` wertet `evaluateKeyEvent` alle im Moment gleichzeitig gepressten Tastatur Eingaben aus und speichert die zugehörigen enum ids in der Instanzvariable `keyevents`.

Wird eine Taste nicht mehr gedrückt wird die enum id in `keyevents` gelöscht. Wird eine Taste neu gedrückt wird die enum id in `keyevents` hinzugefügt

Parameter

<code>QEvent</code>	<code>*event</code>
---------------------	---------------------

Autor

Felix Pfreundtner

5.9.3.2 QSet< int > 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 verknüpft. Möchte man nun beispielsweise abfragen ob der Spieler im Moment schießt so überprüft man: `input->getKeyactions().contains(-Input::Keyaction::Shoot) == True`.

Rückgabe

QSet<int> Instanzvariable keyactions

Autor

Felix Pfreundtner

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

getKeyletters gibt bei Aufruf das QSet keyletters zurück, welches alle im Moment gedrückten Buchstaben als Enum beinhaltet.

Jeder Buchstaben 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 verknüpft. Möchte man nun beispielsweise 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<char> Instanzvariable keyletters

Autor

Felix Pfreundtner

5.9.3.4 Input::Keyaction Input::getLastKeyaction ()

Gibt letzte gedrückte 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 Instanzvariable lastKeyaction

Autor

Rupert, Felix

5.9.3.5 Input::Keyletter Input::getLastKeyletter ()

Gibt letzten gedrückten 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 Instanzvariable lastKeyletter

Autor

Felix

5.9.3.6 void Input::updateKeys () [private]

updateKeyactions berechnet aus allen in keyevents gespeicherten Tastatureingaben die für das Spiel relevanten Kombinationen und speichert diese in keyactions.

Jede Aktionen ist im QSet keyactions als Integer gespeichert, welche über die enumeration Keyaction adressiert wird. Wird durch die Funktion eventFilter ein KeyRelease oder KeyPress Event aufgezeichnet, so wird der QSet keyactions gelöscht und mit den aktualisierten Werten im Qset keyevents abgeglichen. Sind Tasten oder Tastenkombinationen gedrückt worden, welche für das Spiel relevant sind so wird die zur Aktion gehörige integer ID im QSet keyactions hinzugefügt.

Autor

Felix Pfreundtner

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.10 Menu Klassenreferenz

Klasse zum Erzeugen und Anzeigen von Spielmenüs.

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

Klassen

- struct [menuEntry](#)
Strukt zur Beschreibung eines Menü-Eintrags. [Mehr ...](#)

Öffentliche Typen

- enum [menuSelectionChange](#) { **up**, **down** }
wird von changeSelection benötigt
- enum [menuType](#) { **normal**, **highscore** }
verschiedene Menü-Typen (für Background-Musik)

Öffentliche Methoden

- **Menu** (std::string *menuTitle, **menuType** type=normal)
Konstruktor: Erzeugt ein neues Menü, Titel und Type werden festgelegt.
- **~Menu** ()
Menu-Destruktor: Gibt verwendeten Heap-Speicher frei.
- void **clear** ()
entfernt alle Einträge aus dem Menü außer den Titel.
- **menuType** **getType** ()
gibt den Menü-Typ zurück normal/highscore
- std::string * **getTitle** ()
gibt den Menü-Titel zurück
- int **displayInit** ()
Initialisiert das sichtbare Menü.
- int **displayUpdate** ()
aktualisiert das sichtbare Menü.
- int **addEntry** (std::string name, int id, bool clickable=false, **gameState** stateOnClick=(**gameState**) NULL)
Neuen Eintrag hinzufügen.
- int **changeSelection** (**menuSelectionChange** changeType)
wird nach Tastendruck aufgerufen
- **Menu::menuEntry** * **getSelection** ()
gibt den gewählten Eintrag zurück sollte nach Enter aufgerufen werden
- **Menu::menuEntry** * **getEntry** (int position)
gibt Eintrag an der gesuchten Position zurück

Öffentliche Attribute

- QGraphicsPixmapItem **background**
Zeiger auf das Menü-Hintergrundbild.
- QGraphicsScene * **menuScene**
Zeiger auf die Menü-Scene.
- QGraphicsPixmapItem **beerMug**
Bierkrug im Menü

Private Methoden

- int **selectFirstEntry** ()
aktiviert ersten klickbaren Eintrag

Private Attribute

- std::list< struct **menuEntry** * > **menuEntrys**
Liste, die die Menü-Einträge enthält.
- int **currentPosition** = 0
Zeiger auf gewählten Menüpunkt.
- int **numberOfEntrys** = 0
Anzahl der Einträge.
- std::string * **title**
Zeiger auf String, in dem der Titel des Menüs steht. Wird automatisch als erster Eintrag angezeigt.
- **menuType** type
Menü-Typ.

5.10.1 Ausführliche Beschreibung

Klasse zum Erzeugen und Anzeigen von Spielmenüs.

Eine Instanz repräsentiert ein Menü, die wichtigsten Funktionen sind folgende:

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

Die Interaktion mit dem Benutzer wird nicht in der Klasse behandelt, z.B. werden Tastendrucke in `step()` interpretiert und entsprechend `changeSelection()` aufgerufen.

Autor

Rupert

5.10.2 Klassen-Dokumentation

5.10.2.1 struct Menu::menuEntry

Struct zur Beschreibung eines Menü-Eintrags.

Klassen-Elemente

string	name	
int	id	Name, der angezeigt wird.
int	position	ID des Eintrags. Wird mittels menuIds aus game.h eindeutig belegt und in <code>step()</code> zur Unterscheidung der Einträge verwendet.
bool	isClickable	Position im Menü. 0=ganz oben, wird automatisch beim Anlegen gesetzt, d.h. die Reihenfolge ist die Reihenfolge, in der die Einträge erzeugt werden, sie kann später nicht mehr geändert werden.
bool	menuOnEnter	true = Eintrag kann ausgewählt werden, Einträge mit false werden in <code>changeSelection()</code> übersprungen.
gameState	stateOnClick	Ob auf diesen Eintrag ein weiteres Menü folgt. true = Dieser Eintrag ruft ein anderes Menü auf, macht die Auswertung in <code>step()</code> einfacher.
QGraphicsText-Item	showEntry	nächstes Menü. Zusammen mit menuOnEnter, wird in <code>step()</code> ausgewertet.

5.10.3 Beschreibung der Konstruktoren und Destruktoren

5.10.3.1 Menu::Menu (std::string * menuTitle, menuType type = normal)

Konstruktor: Erzeugt ein neues Menü, Titel und Type werden festgelegt.

Danach können Einträge hinzugefügt werden.

Parameter

<i>menuTitle</i>	Zeiger auf String mit Menu-Titel
<i>type</i>	normal/highscore, für Hintergrundmusik

5.10.4 Dokumentation der Elementfunktionen

5.10.4.1 void Menu::clear ()

entfernt alle Einträge aus dem Menü außer den Titel.

Wird für Statistik und Highscore benötigt, nur so können Menüeinträge verändert werden

5.10.4.2 **Menu::menuType** Menu::getType ()

gibt den Menü-Typ zurück normal/highscore

Rückgabe

enum menuType

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

gibt den Menü-Titel zurück

Rückgabe

Zeiger auf std::tring

5.10.4.4 **int Menu::displayInit ()**

Initialisiert das sichtbare Menü.

Muss immer nach anlegen der Menü Entrys aufgerufen werden. Jeder Menüeintrag hat auch QGraphicsTextItem welches hier eingestellt entsprechend eingestellt wird

Rückgabe

0 bei Erfolg

Autor

Flo

5.10.4.5 **int Menu::displayUpdate ()**

aktualisiert das sichtbare Menü.

Je nach Userinput wird immer der aktuell ausgewählte Menüeintrag rot dargestellt und der Bierkrug wird links daneben angezeigt.

Rückgabe

0 bei Erfolg

Autor

Flo

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

Neuen Eintrag hinzufügen.

Parameter

<i>name</i>	String, der angezeigt wird
<i>id</i>	zur eindeutigen Identifizierung, kann zB aus enum menuIds gecastet werden
<i>clickable</i>	Eintrag auswählbar?
<i>stateOnClick</i>	nächstes Menü

Legt einen neuen [menuEntry](#) an und speichert darin die Informationen

Rückgabe

0 bei Erfolg

5.10.4.7 int Menu::changeSelection (menuSelectionChange changeType)

wird nach Tastendruck aufgerufen

Parameter

<i>changeType</i>	up/down
-------------------	---------

Rückgabe

0 bei Erfolg, -1 wenn kein klickbarer Eintrag gefunden

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

gibt den gewählten Eintrag zurück sollte nach Enter aufgerufen werden

Rückgabe

Zeiger auf [menuEntry](#) des aktuellen Eintrags, NULL bei Fehler

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

gibt Eintrag an der gesuchten Position zurück

Parameter

<i>position</i>	
-----------------	--

Rückgabe

Zeiger auf gefundenen Eintrag, sonst NULL

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

5.10.4.10 int Menu::selectFirstEntry () [private]

aktiviert ersten klickbaren Eintrag

Rückgabe

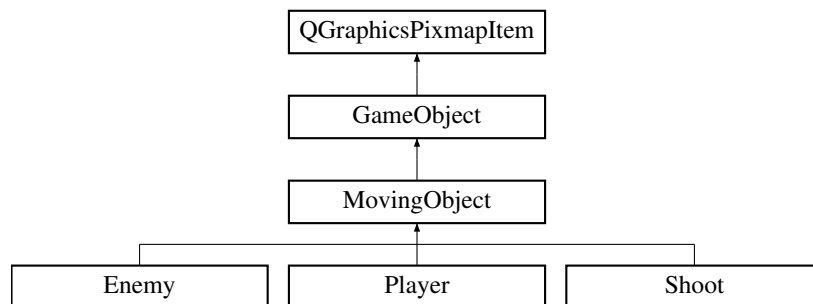
int 0 bei Erfolg, -1 sonst

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.11 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)
- virtual void **update** ()=0
- void **updateFramesDirection** ()
aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).
- void **flipHorizontal** ()
spiegelt Grafiken an der Y-Achse kopiert von "<https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2>" und angepasst.
- void **swapImage** ()
***MovingObject::swapImage** Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.*
- int **getPosX** () const
- int **getPosY** () const
- int **getLength** () const
- int **getHeight** () const
- **objectType** **getType** () const
- void **setAudioID** (int audioID)
- int **getAudioID** () const

Geschützte Methoden

- void **updatePosition** ()
überschreibt die X und Y Position gemäß SpeedXY.

Geschützte Attribute

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

Private Attribute

- int **speedX**
- int **speedY**
- int **framesDirection** = 0
- bool **imageState** = true

5.11.1 Dokumentation der Elementfunktionen

5.11.1.1 void MovingObject::updateFramesDirection ()

aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).

So lässt sich auch die Richtung abfragen ($> || <$ als 0) und mit dem aktuellen speedX-Wert ein Richtungswechsel feststellen

Autor

Flo

5.11.1.2 void MovingObject::flipHorizontal ()

spiegelt Grafiken an der Y-Achse kopiert von "<https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2>" und angepasst.

Ermöglicht das Spiegeln von Bildern über eine Transformationsmatrix. Am Anfang wird getestet ob ein Richtungswechsel statt gefunden hat.

Autor

Flo

5.11.1.3 void MovingObject::swapImage ()

[MovingObject::swapImage](#) Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.

Es wird alle framRate/2 Frames gewechselt und sofort beim loslaufen. Wenn der Spieler in der Luft ist bzw. springt setzt die Animation aus Flo

5.11.1.4 void MovingObject::updatePosition () [protected]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

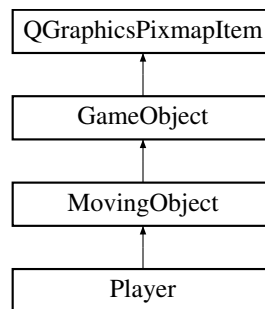
Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.12 Player Klassenreferenz

Klassendiagramm für Player:



Öffentliche Methoden

- **Player** (int posX, int posY, int speedX)
*Class **Player** lastUpdate: [update\(\)](#) 10.6 Johann.*
- 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.
- void **decreaseAlcoholLevel** (int decreaseLevel)
[Player::decreaseAlcoholLevel](#) verringert den Pegel des Spielers.
- int **getAmmunatiuon** () const
[Player::getAmmunatiuon](#) Gibt verbleibende Munition zurück.
- void **increaseAmmunation** (int ammunatationBonus)
[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.
- void **setPosX** (int posX)
- void **setPosY** (int posY)
- int **getSpeedX** () const
- int **getSpeedY** () const
- void **setSpeedX** (int speedX)
- void **setSpeedY** (int speedY)
- void **updateFramesDirection** ()
aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).
- void **flipHorizontal** ()
spiegelt Grafiken an der Y-Achse kopiert von "https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2" und angepasst.
- void **swapImage** ()
MovingObject::swapImage Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.
- int **getPosX** () const
- int **getPosY** () const
- int **getLength** () const
- int **getHeight** () const
- **objectType** **getType** () const
- void **setAudioID** (int audioID)
- int **getAudioID** () const

Geschützte Methoden

- void **updatePosition** ()
überschreibt die X und Y Position gemäß SpeedXY.

Geschützte Attribute

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

Private Attribute

- int **health**
- int **alcoholLevel**
- int **ammunation**
- int **inflictedDamage**
- int **immunityCooldown**
- int **fireCooldown**

- int **fireRate**
- bool **jumpState**
- int **jumpCooldown**
- int **enemiesKilled**

5.12.1 Dokumentation der Elementfunktionen

5.12.1.1 int Player::getHealth () const

[Player::getHealth](#) Gibt aktuellen Lebensstand zurück.

Rückgabe

: Lebensstand

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

[Player::setHealth](#) Lebensstand des Spielers wird gesetzt.

Parameter

<i>health</i>	Lebensstand auf den der Spieler gesetzt wird
---------------	--

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

[Player::receiveDamage](#).

Rückgabe

Lebenszustand des Spielers: true = tot

5.12.1.4 int Player::getAlcoholLevel () const

[Player::getAlcoholLevel](#) Gibt den Pegel des Spielers zurück.

Rückgabe

: Alkoholpegel

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

[Player::increaseAlcoholLevel](#) AlkoholPegel wird verändert.

Durch einen negativen Wert im Argument wird der Pegel gesenkt

Parameter

<i>additionalAlcohol</i>	Wert um den erhöht wird
--------------------------	-------------------------

5.12.1.6 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

<i>decreaseLevel</i>	Wert um den der Pegel verringert wird
----------------------	---------------------------------------

5.12.1.7 int Player::getAmmunatiuon () const

[Player::getAmmunatiuon](#) Gibt verbleibende Munition zurück.

Rückgabe

: verbleibende Munition

5.12.1.8 int Player::getFireCooldown ()

[Player::getFireCooldown](#).

Rückgabe

verbleibende Zeit bs nächster schuss möglich ist

5.12.1.9 int Player::getInflictedDamage () const

[Player::getInflictedDamage](#).

Rückgabe

Schaden den der Spieler zufügt

5.12.1.10 int Player::getImmunityCooldown () const

[Player::getImmunityCooldown](#).

Rückgabe

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

Wird nicht benutzt 23.6.

[Player::setImmunityCooldown](#) Zahl der Frames für Unverwundbarkeit wird gesetzt.

Parameter

<i>immunity-Cooldown</i>	Zahl der Frames
--------------------------	-----------------

5.12.1.12 bool Player::inJump () const

gibt den Sprung-Zustande des Spielers zurück

Rückgabe

5.12.1.13 void Player::update () [virtual]

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

Autor

Johann

Implementiert [MovingObject](#).

5.12.1.14 void MovingObject::updateFramesDirection () [inherited]

aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).

So lässt sich auch die Richtung abfragen ($> || <$ als 0) und mit dem aktuellen speedX-Wert ein Richtungswechsel feststellen

Autor

Flo

5.12.1.15 void MovingObject::flipHorizontal () [inherited]

spiegelt Grafiken an der Y-Achse kopiert von "<https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2>" und angepasst.

Ermöglicht das Spiegeln von Bildern über eine Transformationsmatrix. Am Anfang wird getestet ob ein Richtungswechsel statt gefunden hat.

Autor

Flo

5.12.1.16 void MovingObject::swapImage () [inherited]

[MovingObject::swapImage](#) Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.

Es wird alle framRate/2 Frames gewechselt und sofort beim loslaufen. Wenn der Spieler in der Luft ist bzw. springt setzt die Animation aus Flo

5.12.1.17 void MovingObject::updatePosition () [protected],[inherited]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

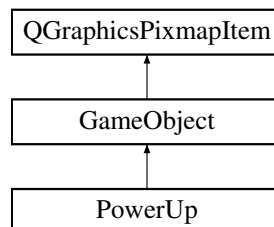
- Wiesn-Run/src/player.h
- Wiesn-Run/src/player.cpp

5.13 PowerUp Klassenreferenz

Klasse für Power-Ups.

```
#include <powerup.h>
```

Klassendiagramm für PowerUp:



Öffentliche Methoden

- **PowerUp** (int posX, int posY, int healthBonus, int alcoholLevelBonus, int ammunitionBonus, 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.
- 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**

Private Attribute

- int **healthBonus**
- int **alcoholLevelBonus**
- int **ammunationBonus**
- int **immunityCooldownBonus**
- powerUpType **powType**

5.13.1 Ausführliche Beschreibung

Klasse für Power-Ups.

Autor

Johann

5.13.2 Beschreibung der Konstruktoren und Destruktoren

5.13.2.1 `PowerUp::PowerUp (int posX, int posY, int healthBonus, int alcoholLevelBonus, int ammunitionBonus, int immunityCooldownBonus, powerUpType type)`

Konstruktor.

Parameter

<i>posX</i>	
<i>posY</i>	
<i>length</i>	
<i>height</i>	
<i>healthBonus</i>	
<i>alcoholLevel-</i> <i>Bonus</i>	
<i>ammunition-</i> <i>Bonus</i>	
<i>immunity-</i> <i>CooldownBonus</i>	

Autor

Johann

5.13.2.2 `PowerUp::~~PowerUp ()`

Destruktor.

Autor

Johann

5.13.3 Dokumentation der Elementfunktionen

5.13.3.1 `int PowerUp::getHealthBonus () const`

Get-Methoden für die Objekteigenschaften.

Gibt den Bonus auf Leben zurück.

Autor

Johann

5.13.3.2 `int PowerUp::getAlcoholLevelBonus () const`

Gibt den Bonus auf Alcohollevel zurück.

Autor

Johann

5.13.3.3 int PowerUp::getAmmunationBonus () const

Gibt den Bonus auf Munition zurück.

Autor

Johann

5.13.3.4 int PowerUp::getImmunityCooldownBonus () const

Gibt den Bonus auf Immunität zurück.

Autor

Johann

5.13.3.5 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

5.14 RenderBackground Klassenreferenz

Hintergrund-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion [Game::startNewGame](#) angelegt.

```
#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.
- void [updateParallaxe](#) (int x)
[RenderBackground::updateParallaxe](#) Die Position der hinteren Hintergrundebene wird laufend so aktualisiert.
- 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.

Private Attribute

- QGraphicsPixmapItem **backgroundOne**
- QGraphicsPixmapItem **backgroundTwo**
- QGraphicsPixmapItem **backgroundThree**
- QGraphicsPixmapItem **backgroundFour**
- int **imageLength** = 2560

5.14.1 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

5.14.2 Beschreibung der Konstruktoren und Destruktoren

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

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

Parameter

<i>scene</i>	: levelScene
<i>level</i>	: aktuelles Level

Autor

Flo

5.14.3 Dokumentation der Elementfunktionen

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

[RenderBackground::setPos](#) Funktion positioniert Hintergrundgrafiken neu.

(nur "x" ändert sich, "y" ist immer 0)

Parameter

<i>x</i>	: x-Position
<i>background</i>	: Hintergrundgrafikitem

Autor

Flo

5.14.3.2 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

<code>x</code>	: x-Wert der Positionsänderung des Spielers im aktuellen Step
----------------	---

Autor

Flo

5.14.3.3 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

<code>x</code>	: x-Position des linken Bildrandes im Level
----------------	---

Autor

Flo

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.15 RenderGUI Klassenreferenz

Anzeigen der Spielerwerte-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion [Game::startNewGame](#) angelegt.

```
#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.

Private Attribute

- QGraphicsTextItem **showHealth**
- QGraphicsRectItem **showHealthBar** [2]
- QGraphicsTextItem **showScore**
- QGraphicsTextItem **showAmmo**
- QGraphicsTextItem **showAlcohol**
- QGraphicsRectItem **showAlcoholBar** [2]

5.15.1 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

5.15.2 Beschreibung der Konstruktoren und Destruktoren

5.15.2.1 RenderGUI::RenderGUI (QGraphicsScene * scene)

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

Parameter

<i>scene</i>	: levelScene
--------------	--------------

Autor

Flo

5.15.3 Dokumentation der Elementfunktionen

5.15.3.1 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

<i>x</i>	: x-Wert der Positionsänderung des Spielers im aktuellen Step
----------	---

Autor

Flo

5.15.3.2 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

<i>health</i>	: aktueller Gesundheitswert
<i>alcohol</i>	: aktueller Alkoholpegelwert

<i>ammo</i>	: aktueller Munitionsstand
<i>score</i>	: aktueller Punktestad

Autor

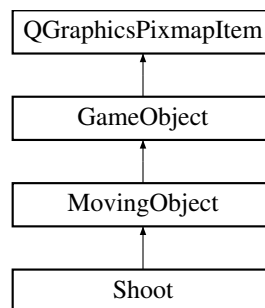
Flo

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

5.16 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** ()
- void **setPosX** (int posX)
- void **setPosY** (int posY)
- int **getSpeedX** () const
- int **getSpeedY** () const
- void **setSpeedX** (int speedX)
- void **setSpeedY** (int speedY)
- void **updateFramesDirection** ()
aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Objekt steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).
- void **flipHorizontal** ()
spiegelt Grafiken an der Y-Achse kopiert von "https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2" und angepasst.
- void **swapImage** ()
MovingObject::swapImage Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.
- int **getPosX** () const

- int **getPosY** () const
- int **getLength** () const
- int **getHeight** () const
- **objectType** **getType** () const
- void **setAudioID** (int audioID)
- int **getAudioID** () const

Geschützte Methoden

- void **updatePosition** ()
überschreibt die X und Y Position gemäß SpeedXY.

Geschützte Attribute

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

Private Attribute

- int **inflictedDamage**
- **objectType** **origin**

5.16.1 Beschreibung der Konstruktoren und Destruktoren

5.16.1.1 Shoot::Shoot (int posX, int posY, int direction, objectType origin)

Konstruktor für einen Schuss(Bierkrug)

Parameter

<i>posX</i>	: x-Position
<i>posY</i>	: y-Position
<i>origin</i>	: Schuss Erzeuger

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

Autor

Johann

5.16.2 Dokumentation der Elementfunktionen

5.16.2.1 int Shoot::getInflictedDamage () const

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

Rückgabe

Schaden

Autor

Johann

5.16.2.2 objectType Shoot::getOrigin ()

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

Rückgabe

Ursprung des Bierkruges

Autor

Johann

5.16.2.3 void MovingObject::updateFramesDirection () [inherited]

aktualisiert die Anzahl der Frames für die ein Object ununterbrochen in eine Richtung gelaufen ist Wenn das Object steht oder die Richtung wechselt wird FramesDirection auf 0 gesetzt, ansonsten je nach Richtung um eins erhöht (vorwärts) oder um eins erniedrigt (rückwärts).

So lässt sich auch die Richtung abfragen ($> || <$ als 0) und mit dem aktuellen speedX-Wert ein Richtungswechsel feststellen

Autor

Flo

5.16.2.4 void MovingObject::flipHorizontal () [inherited]

spiegelt Grafiken an der Y-Achse kopiert von "<https://forum.qt.io/topic/18131/solved-flip-a-qgraphicssvgitem-on-its-center-point/2>" und angepasst.

Ermöglicht das Spiegeln von Bildern über eine Transformationsmatrix. Am Anfang wird getestet ob ein Richtungswechsel statt gefunden hat.

Autor

Flo

5.16.2.5 void MovingObject::swapImage () [inherited]

[MovingObject::swapImage](#) Die Funktion testet mit Hilfe von "imageState" welches Bild gerade aktiv ist und wechselt dann jeweils auf das andere Bild für die Bewegungsanimation.

Es wird alle framRate/2 Frames gewechselt und sofort beim loslaufen. Wenn der Spieler in der Luft ist bzw. springt setzt die Animation aus Flo

5.16.2.6 void MovingObject::updatePosition () [protected],[inherited]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

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

Kapitel 6

Datei-Dokumentation

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

definitions beinhaltet Datentyp Definitionen.

```
#include <iostream>
#include <list>
#include <chrono>
```

Klassen

- struct [scoreStruct](#)
Struktur für die Score des Spielers In dieser Struktur werden Name des Spielers, getötete Gegner, zurückgelegte Entfernung und Alkohol-Punkte gespeichert. [Mehr ...](#)
- struct [audioCooldownStruct](#)
- struct [audioDistanceStruct](#)
typedef struct [audioDistanceStruct](#) Mehr ...
- struct [audioStruct](#)
- struct [audioCooldownstruct](#)
- struct [stateStruct](#)
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. [Mehr ...](#)

Aufzählungen

- enum [gameState](#) {
gameIsRunning, gameMenuStart, gameMenuCredits, gameMenuLevel,
gameMenuBreak, gameMenuStatistics, gameMenuName, gameMenuHighscore,
gameMenuHelp }
Enumerator für den aktuellen Spielstatus ähnlich zu einer StateMachine wird in step() über switch abgefragt.
- enum [objectType](#) {
player, enemy_tourist, enemy_security, obstacle,
plane, shot, powerUp, BOSS }
Enumerator für den Objekt-Typ um welche Art von Objekt handelt es sich.
- enum [collisionDirection](#) { **fromLeft, fromRight, fromAbove, fromBelow** }
Enumerator für die Kollisions-Richtung Zur Bewegungskorrektur muss klar sein, ob der Spieler ein anderes Objekt von der Seite oder von Oben/Unten berührt hat.

- enum `audioType` {
`scene_flyingbeer`, `scene_enemy_tourist`, `scene_enemy_security`, `scene_enemy_boss`,
`scene_collision_obstacle`, `scene_collision_enemy`, `scene_collision_player`, `scene_collision_flyingbeer`,
`powerup_beer`, `powerup_food`, `status_alcohol`, `status_life`,
`status_lifecritical`, `status_dead`, `player_walk`, `player_jump`,
`background_menu`, `background_highscore`, `background_level1`, `background_level2`,
`background_level3`, `background_startgame`, `background_levelfinished` }

Enum für den Audiotype.

- enum `powerUpType` { `beer`, `food` }

Variablen

- const int `frameRate` = 30
Anzahl gameloop-Durchläufe pro Sekunde wird in allen Klassen für die CooldownParameter benutzt.
- const int `minusAlcoholPerSecond` = 30
Alkohol, der pro Sekunde abgebaut wird.
- const int `playerScale` = 60
*Skalierungsfaktor für die Breite des Spielerobjekts bei 1024 Bildschirmbreite: Breite:Höhe Spieler, Gegner: 1:2
Hindernisse: (3/2):2 , 2:(1/3) dabei ist das erste das Standardhindernis Power-Up: (2/3):(2/3) , Krug: (1/3):(2/3)*
- const int `yOffset` = 668
*Offsets Spieler <-> linker Fensterrand und Spielebene <-> oberer Fensterrand yOffset: Distanz zwischen oberem Rand (QT Koordinatensystem läuft von oben nach unten) und unterster Spielebene => Fensterhöhe(768px) - yOffset = 100px
playerOffset: Distanz zwischen linkem Rand und Spieler.*
- const int `playerOffset` = 100
- const int `maxSpeed` = 3 * (`playerScale` / `frameRate`)
Geschwindigkeit mit der sich die beweglichen Objekte durch die Welt bewegen.
- const int `playerSpeed` = `maxSpeed` + 1
normale Player-Speed
- const int `maxSpeedY` = 3 * `maxSpeed` / 2
Fall- / Sprunggeschwindigkeit.
- const int `maxHealth` = 5
maximales Leben
- const int `maxAlcohol` = 1000
- const int `beerAlcohol` = 400
PowerUp-Konstanten Hier werden die Konstanten gesetzt, die beim Einsammeln eines PowerUps hinzugefügt werden.
- const int `beerHealth` = 1
- const int `beerAmmo` = 1
- const int `hendlHealth` = 1
- const int `hendlAlcoholMalus` = -500
- const int `spawnDistance` = 1024
Distanzen spawnDistance ist die Distanz vom Spieler zum Objekt, ab der Objekte von levelSpawn nach worldObjects verlegt werden.
- const int `deleteDistance` = 200

6.1.1 Ausführliche Beschreibung

definitions beinhaltet Datentyp Definitionen.

Autor

Johann, Simon, Felix

6.1.2 Klassen-Dokumentation

6.1.2.1 struct scoreStruct

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

Klassen-Elemente

string	name	
int	totalPoints	
int	distanceCovered	
int	alcoholPoints	
int	enemiesKilled	

6.1.2.2 struct audioCooldownStruct

Klassen-Elemente

duration< int, milli >	scene_flyingbeer	
duration< int, milli >	scene_enemy_- security	
duration< int, milli >	scene_enemy_- tourist	
duration< int, milli >	scene_enemy_- boss	
duration< int, milli >	scene_collision- _obstacle	
duration< int, milli >	scene_collision- _enemy	
duration< int, milli >	scene_collision- _player	
duration< int, milli >	scene_collision- _flyingbeer	
duration< int, milli >	powerup_beer	
duration< int, milli >	powerup_food	
duration< int, milli >	status_alcohol	
duration< int, milli >	status_life	
duration< int, milli >	status_lifecritical	

duration< int, milli >	status_dead	
duration< int, milli >	player_walk	
duration< int, milli >	player_jump	
duration< int, milli >	background_- menu	
duration< int, milli >	background_- highscore	
duration< int, milli >	background_- level1	
duration< int, milli >	background_- level2	
duration< int, milli >	background_- level3	
duration< int, milli >	background_- startgame	
duration< int, milli >	background_- levelfinished	

6.1.2.3 struct audioDistanceStruct

typedef struct [audioDistanceStruct](#)

Typdef Struct mit Konstanten für die Distance jedes Audiotypes. In diesen Konstanten wird festgelegt wie weit entfernt ein Event (mit id=...) eines audioTypes vom Spieler standardmäßig auftritt [Werbereich 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

Klassen-Elemente

float	scene_flyingbeer	
float	scene_enemy_- tourist	
float	scene_enemy_- security	
float	scene_enemy_- boss	
float	scene_collision- _obstacle	
float	scene_collision- _enemy	
float	scene_collision- _player	
float	scene_collision- _flyingbeer	
float	powerup_beer	
float	powerup_food	

float	status_alcohol	
float	status_life	
float	status_lifecritical	
float	status_dead	
float	player_walk	
float	player_jump	
float	background_- menu	
float	background_- highscore	
float	background_- level1	
float	background_- level2	
float	background_- level3	
float	background_- startgame	
float	background_- levelfinished	

6.1.2.4 struct audioStruct

Klassen-Elemente

int	id	
audioType	type	
float	distance	

6.1.2.5 struct audioCooldownstruct

Klassen-Elemente

struct audioStruct	audioEvent	
duration< int, milli >	cooldown	

6.1.2.6 struct stateStruct

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 stirbt etc.

Noch zu erledigen Diese Struktur ist vermutlich überflüssig.

Autor

Simon

Klassen-Elemente

bool	gameOver	
int	actLevel	
int	audioID_- Background	
bool	beerCollected	
bool	chickenCollected	

6.1.3 Dokumentation der Aufzählungstypen

6.1.3.1 enum gameState

Enumerator für den aktuellen Spielstatus ähnlich zu einer StateMachine wird in step() über switch abgefragt.

Autor

Rupert

6.1.3.2 enum objectType

Enumerator für den Objekt-Typ um welche Art von Objekt handelt es sich.

Autor

Johann

6.1.3.3 enum collisionDirection

Enumerator für die Kollisions-Richtung Zur Bewegungskorrektur muss klar sein, ob der Spieler ein anderes Objekt von der Seite oder von Oben/Unten berührt hat.

Da auch aus Gegner-Sicht die Kollision berechnet wird, gibt es auch Kollisionen von rechts.

Autor

Simon

6.1.3.4 enum enum audioType

Enum für den Audiotype.

In diesen Enum wird der Audiotype in einen Integer gewandelt. Jede ID wird dabei einem Audiotype zugeordnet welcher angibt welches Audiofile zur ID abgespielt werden soll.

Autor

Felix Pfreundtner

Aufzählungswerte

scene_flyingbeer fliegendes Bier: wird solange gesendet wie Bier in der Luft fliegt

scene_enemy_tourist auftretender Tourist Gegner: wird gesendet solange Gegner lebt

scene_enemy_security auftretender Security Gegner: wird gesendet solange Gegner lebt

scene_enemy_boss lebender Boss Gegner: wird gesendet solange Bossgegner lebt

- scene_collision_obstacle** Kollision mit Hinderniss aufgetreten: wird einmal gesendet wenn eine Kollision mit einem Hindernis auftritt (cooldown)
- scene_collision_enemy** Spieler hat Schaden am Gegner bezweckt: wird einmal gesendet wenn Schaden auftritt (cooldown)
- scene_collision_player** Spieler hat Schaden genommen: wird einmal gesendet wenn Schaden auftritt (cooldown)
- scene_collision_flyingbeer** Kollision mit geworfenen Bier aufgetreten: wird einmal gesendet wenn eine Kollision mit einem geworfenen Bier auftritt (cooldown)
- powerup_beer** Bier Powerup aufgenommen: wird einmal gesendet wenn Powerup aufgenommen wird (cooldown)
- powerup_food** Essens Powerup aufgenommen: wird einmal gesendet wenn Powerup aufgenommen wird (cooldown)
- status_alcohol** Alkohol Status höher als 60%: wird solange gesendet wie Alkoholstatus höher als 60% ist.
- status_life** Leben Status 2 Lebenspunkt3: wird solange gesendet wie Spieler 2 Lebenspunkt3 hat 40% ist.
- status_lifecritical** Leben Status 1 Lebenspunkt: wird solange gesendet wie Spieler 1 Lebenspunkt hat.
- status_dead** Gameover des Spielers: wird gesendet wenn der Spieler 0% Lebenstatus hat (cooldown)
- player_walk** Laufbewegung des Spielers: wird solange gesendet wie Spieler sich bewegt.
- player_jump** Springbewegung des Spielers: wird einmal gesendet wenn der Spieler losspringt (cooldown)
- background_menu** Hintergrund Musik des Hauptmenüs: wird solange gesendet wie Hauptmenü aktiv ist.
- background_highscore** Hintergrund Musik des Highscoremenüs: wird solange gesendet wie Highscoremenü aktiv ist.
- background_level1** Hintergrund Musik des Levels 1: wird solange gesendet wie Level 1 aktiv ist.
- background_level2** Hintergrund Musik des Levels 2: wird solange gesendet wie Level 2 aktiv ist.
- background_level3** Hintergrund Musik des Levels 3: wird solange gesendet wie Level 3 aktiv ist.
- background_startgame** Startton wenn Spiel begonnen wird: wird einmal zu Beginn des Level 1 gesendet (cooldown)
- background_levelfinished** Gewinnton wenn Level erfolgreich beendet wurde: wird einmal an jedem Levelende gesendet (cooldown)

6.1.4 Variablen-Dokumentation

6.1.4.1 `const int spawnDistance = 1024`

Distanzen `spawnDistance` ist die Distanz vom Spieler zum Objekt, ab der Objekte von `levelSpawn` nach `worldObjects` verlegt werden.

`deleteDistance` ist die Distanz von einem Objekt zum Spieler, ab welcher das Objekt gelöscht wird.

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

The portable PortAudio API.

Klassen

- struct [PaHostApiInfo](#)
A structure containing information about a particular host API. [Mehr ...](#)
- struct [PaHostErrorInfo](#)
Structure used to return information about a host error condition. [Mehr ...](#)
- struct [PaDeviceInfo](#)

- *A structure providing information and capabilities of PortAudio devices. [Mehr ...](#)*
- struct [PaStreamParameters](#)
Parameters for one direction (input or output) of a stream. [Mehr ...](#)
- struct [PaStreamCallbackTimeInfo](#)
Timing information for the buffers passed to the stream callback. [Mehr ...](#)
- struct [PaStreamInfo](#)
A structure containing unchanging information about an open stream. [Mehr ...](#)

Makrodefinitionen

- #define [paNoDevice](#) ((PaDeviceIndex)-1)
A special PaDeviceIndex value indicating that no device is available, or should be used.
- #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.
- #define [paFloat32](#) ((PaSampleFormat) 0x00000001)
- #define [paInt32](#) ((PaSampleFormat) 0x00000002)
- #define [paInt24](#) ((PaSampleFormat) 0x00000004)
Packed 24 bit format.
- #define [paInt16](#) ((PaSampleFormat) 0x00000008)
- #define [paInt8](#) ((PaSampleFormat) 0x00000010)
- #define [paUInt8](#) ((PaSampleFormat) 0x00000020)
- #define [paCustomFormat](#) ((PaSampleFormat) 0x00010000)
- #define [paNonInterleaved](#) ((PaSampleFormat) 0x80000000)
- #define [paFormatIsSupported](#) (0)
Return code for Pa_IsFormatSupported indicating success.
- #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.
- #define [paNoFlag](#) ((PaStreamFlags) 0)
- #define [paClipOff](#) ((PaStreamFlags) 0x00000001)
Disable default clipping of out of range samples.
- #define [paDitherOff](#) ((PaStreamFlags) 0x00000002)
Disable default dithering.
- #define [paNeverDropInput](#) ((PaStreamFlags) 0x00000004)
Flag requests that where possible a full duplex stream will not discard overflowed input samples without calling the stream callback.
- #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).
- #define [paPlatformSpecificFlags](#) ((PaStreamFlags) 0xFFFF0000)
A mask specifying the platform specific bits.
- #define [paInputUnderflow](#) ((PaStreamCallbackFlags) 0x00000001)
In a stream opened with paFramesPerBufferUnspecified, indicates that input data is all silence (zeros) because no real data is available.
- #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.
- #define [paOutputUnderflow](#) ((PaStreamCallbackFlags) 0x00000004)
Indicates that output data (or a gap) was inserted, possibly because the stream callback is using too much CPU time.
- #define [paOutputOverflow](#) ((PaStreamCallbackFlags) 0x00000008)
Indicates that output data will be discarded because no room is available.
- #define [paPrimingOutput](#) ((PaStreamCallbackFlags) 0x00000010)
Some of all of the output data will be used to prime the stream, input data may be zero.

Typdefinitionen

- typedef int [PaError](#)
Error codes returned by PortAudio functions.
- typedef enum PaErrorCode **PaErrorCode**
- typedef int [PaDeviceIndex](#)
The type used to refer to audio devices.
- typedef int [PaHostApiIndex](#)
The type used to enumerate to host APIs at runtime.
- typedef enum [PaHostApiType](#) [PaHostApiType](#)
Unchanging unique identifiers for each supported host API.
- typedef struct [PaHostApiInfo](#) [PaHostApiInfo](#)
A structure containing information about a particular host API.
- typedef struct [PaHostErrorInfo](#) [PaHostErrorInfo](#)
Structure used to return information about a host error condition.
- typedef double [PaTime](#)
The type used to represent monotonic time in seconds.
- typedef unsigned long [PaSampleFormat](#)
A type used to specify one or more sample formats.
- typedef struct [PaDeviceInfo](#) [PaDeviceInfo](#)
A structure providing information and capabilities of PortAudio devices.
- typedef struct [PaStreamParameters](#) [PaStreamParameters](#)
Parameters for one direction (input or output) of a stream.
- typedef void [PaStream](#)
A single PaStream can provide multiple channels of real-time streaming audio input and output to a client application.
- typedef unsigned long [PaStreamFlags](#)
Flags used to control the behavior of a stream.
- typedef struct [PaStreamCallbackTimeInfo](#) [PaStreamCallbackTimeInfo](#)
Timing information for the buffers passed to the stream callback.
- typedef unsigned long [PaStreamCallbackFlags](#)
Flag bit constants for the statusFlags to PaStreamCallback.
- typedef enum [PaStreamCallbackResult](#) [PaStreamCallbackResult](#)
Allowable return values for the PaStreamCallback.
- 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.
- typedef void [PaStreamFinishedCallback](#) (void *userData)
Functions of type PaStreamFinishedCallback are implemented by PortAudio clients.
- typedef struct [PaStreamInfo](#) [PaStreamInfo](#)
A structure containing unchanging information about an open stream.

Aufzählungen

- enum **PaErrorCode** {
paNoError = 0, **paNotInitialized** = -10000, **paUnanticipatedHostError**, **paInvalidChannelCount**,
paInvalidSampleRate, **paInvalidDevice**, **paInvalidFlag**, **paSampleFormatNotSupported**,
paBadIODeviceCombination, **paInsufficientMemory**, **paBufferTooBig**, **paBufferTooSmall**,
paNullCallback, **paBadStreamPtr**, **paTimedOut**, **paInternalError**,
paDeviceUnavailable, **paIncompatibleHostApiSpecificStreamInfo**, **paStreamsStopped**, **paStreamsNotStopped**,
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 }
Unchanging unique identifiers for each supported host API.
- enum [PaStreamCallbackResult](#) { [paContinue](#) =0, [paComplete](#) =1, [paAbort](#) =2 }
Allowable return values for the PaStreamCallback.

Funktionen

- int [Pa_GetVersion](#) (void)
Retrieve the release number of the currently running PortAudio build, eg 1900.
- const char * [Pa_GetVersionText](#) (void)
Retrieve a textual description of the current PortAudio build, eg "PortAudio V19-devel 13 October 2002".
- const char * [Pa_GetErrorText](#) ([PaError](#) errorCode)
Translate the supplied PortAudio error code into a human readable message.
- [PaError](#) [Pa_Initialize](#) (void)
Library initialization function - call this before using PortAudio.
- [PaError](#) [Pa_Terminate](#) (void)
Library termination function - call this when finished using PortAudio.
- [PaHostApiIndex](#) [Pa_GetHostApiCount](#) (void)
Retrieve the number of available host APIs.
- [PaHostApiIndex](#) [Pa_GetDefaultHostApi](#) (void)
Retrieve the index of the default host API.
- const [PaHostApiInfo](#) * [Pa_GetHostApiInfo](#) ([PaHostApiIndex](#) hostApi)
Retrieve a pointer to a structure containing information about a specific host Api.
- [PaHostApiIndex](#) [Pa_HostApiTypeIdToHostApiIndex](#) ([PaHostApiTypeId](#) type)
Convert a static host API unique identifier, into a runtime host API index.
- [PaDeviceIndex](#) [Pa_HostApiDeviceIndexToDeviceIndex](#) ([PaHostApiIndex](#) hostApi, int hostApiDeviceIndex)
Convert a host-API-specific device index to standard PortAudio device index.
- const [PaHostErrorInfo](#) * [Pa_GetLastHostErrorInfo](#) (void)
Return information about the last host error encountered.
- [PaDeviceIndex](#) [Pa_GetDeviceCount](#) (void)
Retrieve the number of available devices.
- [PaDeviceIndex](#) [Pa_GetDefaultInputDevice](#) (void)
Retrieve the index of the default input device.
- [PaDeviceIndex](#) [Pa_GetDefaultOutputDevice](#) (void)
Retrieve the index of the default output device.
- const [PaDeviceInfo](#) * [Pa_GetDeviceInfo](#) ([PaDeviceIndex](#) device)
Retrieve a pointer to a [PaDeviceInfo](#) structure containing information about the specified device.
- [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.
- [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.
- [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.*
- [PaError Pa_CloseStream](#) ([PaStream](#) *stream)
Closes an audio stream.
- [PaError Pa_SetStreamFinishedCallback](#) ([PaStream](#) *stream, [PaStreamFinishedCallback](#) *streamFinished-Callback)
Register a stream finished callback function which will be called when the stream becomes inactive.
- [PaError Pa_StartStream](#) ([PaStream](#) *stream)
Commences audio processing.
- [PaError Pa_StopStream](#) ([PaStream](#) *stream)
Terminates audio processing.
- [PaError Pa_AbortStream](#) ([PaStream](#) *stream)
Terminates audio processing immediately without waiting for pending buffers to complete.
- [PaError Pa_IsStreamStopped](#) ([PaStream](#) *stream)
Determine whether the stream is stopped.
- [PaError Pa_IsStreamActive](#) ([PaStream](#) *stream)
Determine whether the stream is active.
- const [PaStreamInfo](#) * [Pa_GetStreamInfo](#) ([PaStream](#) *stream)
Retrieve a pointer to a [PaStreamInfo](#) structure containing information about the specified stream.
- [PaTime Pa_GetStreamTime](#) ([PaStream](#) *stream)
Returns the current time in seconds for a stream according to the same clock used to generate callback [PaStream-CallbackTimeInfo](#) timestamps.
- double [Pa_GetStreamCpuLoad](#) ([PaStream](#) *stream)
Retrieve CPU usage information for the specified stream.
- [PaError Pa_ReadStream](#) ([PaStream](#) *stream, void *buffer, unsigned long frames)
Read samples from an input stream.
- [PaError Pa_WriteStream](#) ([PaStream](#) *stream, const void *buffer, unsigned long frames)
Write samples to an output stream.
- signed long [Pa_GetStreamReadAvailable](#) ([PaStream](#) *stream)
Retrieve the number of frames that can be read from the stream without waiting.
- signed long [Pa_GetStreamWriteAvailable](#) ([PaStream](#) *stream)
Retrieve the number of frames that can be written to the stream without waiting.
- [PaHostApiTypeId Pa_GetStreamHostApiType](#) ([PaStream](#) *stream)
Retrieve the host type handling an open stream.
- [PaError Pa_GetSampleSize](#) ([PaSampleFormat](#) format)
Retrieve the size of a given sample format in bytes.
- void [Pa_Sleep](#) (long msec)
Put the caller to sleep for at least 'msec' milliseconds.

6.2.1 Ausführliche Beschreibung

The portable PortAudio API.

6.2.2 Klassen-Dokumentation

6.2.2.1 struct PaHostApiInfo

A structure containing information about a particular host API.

Klassen-Elemente

int	structVersion	this is struct version 1
PaHostApiTypeId	type	The well known unique identifier of this host API. Siehe auch PaHostApiTypeId
const char *	name	A textual description of the host API for display on user interfaces.
int	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
PaDeviceIndex	defaultInput-Device	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.
PaDeviceIndex	defaultOutput-Device	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.

6.2.2.2 struct PaHostErrorInfo

Structure used to return information about a host error condition.

Klassen-Elemente

PaHostApiTypeId	hostApiType	the host API which returned the error code
long	errorCode	the error code returned
const char *	errorText	a textual description of the error if available, otherwise a zero-length string

6.2.2.3 struct PaDeviceInfo

A structure providing information and capabilities of PortAudio devices.

Devices may support input, output or both input and output.

Klassen-Elemente

int	structVersion	
const char *	name	
PaHostApiIndex	hostApi	note this is a host API index, not a type id
int	maxInput-Channels	
int	maxOutput-Channels	
PaTime	defaultLowInput-Latency	Default latency values for interactive performance.

PaTime	defaultLow-OutputLatency	
PaTime	defaultHighInput-Latency	Default latency values for robust non-interactive applications (eg. playing sound files).
PaTime	defaultHigh-OutputLatency	
double	defaultSample-Rate	

6.2.2.4 struct PaStreamParameters

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

Klassen-Elemente

PaDeviceIndex	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.
int	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.
PaSample-Format	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.
PaTime	suggested-Latency	<p>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().</p> <p>Siehe auch</p> <p>default*Latency in PaDeviceInfo, *Latency in PaStreamInfo</p>

void *	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.
--------	---------------------------	---

6.2.2.5 struct 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](#)

Klassen-Elemente

PaTime	inputBufferAdcTime	The time when the first sample of the input buffer was captured at the ADC input.
PaTime	currentTime	The time when the stream callback was invoked.
PaTime	outputBufferDacTime	The time when the first sample of the output buffer will output the DAC.

6.2.2.6 struct PaStreamInfo

A structure containing unchanging information about an open stream.

Siehe auch

[Pa_GetStreamInfo](#)

Klassen-Elemente

int	structVersion	this is struct version 1
PaTime	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
PaTime	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

double	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() .
--------	------------	--

6.2.3 Makro-Dokumentation

6.2.3.1 #define paNoDevice ((PaDeviceIndex)-1)

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

Siehe auch

[PaDeviceIndex](#)

6.2.3.2 #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](#)

6.2.3.3 #define paFloat32 ((PaSampleFormat) 0x00000001)

Siehe auch

[PaSampleFormat](#)

6.2.3.4 #define paInt32 ((PaSampleFormat) 0x00000002)

Siehe auch

[PaSampleFormat](#)

6.2.3.5 #define paInt24 ((PaSampleFormat) 0x00000004)

Packed 24 bit format.

Siehe auch

[PaSampleFormat](#)

6.2.3.6 #define paInt16 ((PaSampleFormat) 0x00000008)

Siehe auch

[PaSampleFormat](#)

6.2.3.7 **#define paInt8 ((PaSampleFormat) 0x00000010)**

Siehe auch

[PaSampleFormat](#)

6.2.3.8 **#define paUInt8 ((PaSampleFormat) 0x00000020)**

Siehe auch

[PaSampleFormat](#)

6.2.3.9 **#define paCustomFormat ((PaSampleFormat) 0x00010000)**

Siehe auch

[PaSampleFormat](#)

6.2.3.10 **#define paNonInterleaved ((PaSampleFormat) 0x80000000)**

Siehe auch

[PaSampleFormat](#)

6.2.3.11 **#define paFormatIsSupported (0)**

Return code for Pa_IsFormatSupported indicating success.

6.2.3.12 **#define paNoFlag ((PaStreamFlags) 0)**

Siehe auch

[PaStreamFlags](#)

6.2.3.13 **#define paClipOff ((PaStreamFlags) 0x00000001)**

Disable default clipping of out of range samples.

Siehe auch

[PaStreamFlags](#)

6.2.3.14 **#define paDitherOff ((PaStreamFlags) 0x00000002)**

Disable default dithering.

Siehe auch

[PaStreamFlags](#)

6.2.3.15 #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](#)

6.2.3.16 #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](#)

6.2.3.17 #define paPlatformSpecificFlags ((PaStreamFlags)0xFFFF0000)

A mask specifying the platform specific bits.

Siehe auch

[PaStreamFlags](#)

6.2.3.18 #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](#)

6.2.3.19 #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](#)

6.2.3.20 `#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](#)

6.2.3.21 `#define paOutputOverflow ((PaStreamCallbackFlags) 0x00000008)`

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

Siehe auch

[PaStreamCallbackFlags](#)

6.2.3.22 `#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](#)

6.2.4 Dokumentation der benutzerdefinierten Typen

6.2.4.1 `typedef int PaError`

Error codes returned by PortAudio functions.

Note that with the exception of `paNoError`, all `PaErrorCodes` are negative.

6.2.4.2 `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](#)

6.2.4.3 `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](#)

6.2.4.4 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](#)

6.2.4.5 typedef struct PaHostApiInfo PaHostApiInfo

A structure containing information about a particular host API.

6.2.4.6 typedef double PaTime

The type used to represent monotonic time in seconds.

PaTime is used for the fields of the [PaStreamCallbackTimeInfo](#) argument to the PaStreamCallback and as the result of [Pa_GetStreamTime\(\)](#).

PaTime values have unspecified origin.

Siehe auch

[PaStreamCallback](#), [PaStreamCallbackTimeInfo](#), [Pa_GetStreamTime](#)

6.2.4.7 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](#)

6.2.4.8 typedef struct PaDeviceInfo PaDeviceInfo

A structure providing information and capabilities of PortAudio devices.

Devices may support input, output or both input and output.

6.2.4.9 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](#)

6.2.4.10 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](#)

6.2.4.11 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](#)

6.2.4.12 typedef unsigned long PaStreamCallbackFlags

Flag bit constants for the statusFlags to PaStreamCallback.

Siehe auch

[paInputUnderflow](#), [paInputOverflow](#), [paOutputUnderflow](#), [paOutputOverflow](#), [paPrimingOutput](#)

6.2.4.13 typedef enum PaStreamCallbackResult PaStreamCallbackResult

Allowable return values for the PaStreamCallback.

Siehe auch

[PaStreamCallback](#)

6.2.4.14 `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

<i>input</i>	and
<i>output</i>	are either arrays of interleaved samples or; if non-interleaved samples were requested using the <code>paNonInterleaved</code> 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

<i>frameCount</i>	The number of sample frames to be processed by the stream callback.
<i>timeInfo</i>	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 <code>PaStreamCallbackTimeInfo</code> and <code>Pa_GetStreamTime()</code>
<i>statusFlags</i>	Flags indicating whether input and/or output buffers have been inserted or will be dropped to overcome underflow or overflow conditions.
<i>userData</i>	The value of a user supplied pointer passed to <code>Pa_OpenStream()</code> 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.

6.2.4.15 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

<i>userData</i>	The userData parameter supplied to Pa_OpenStream()
-----------------	--

Siehe auch

[Pa_SetStreamFinishedCallback](#)

6.2.4.16 typedef struct PaStreamInfo PaStreamInfo

A structure containing unchanging information about an open stream.

Siehe auch

[Pa_GetStreamInfo](#)

6.2.5 Dokumentation der Aufzählungstypen

6.2.5.1 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](#)

6.2.5.2 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.

6.2.6 Dokumentation der Funktionen

6.2.6.1 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](#)

6.2.6.2 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_Initialize\(\)](#) 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](#)

6.2.6.3 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](#)

6.2.6.4 **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.

6.2.6.5 **const PaHostApiInfo*** Pa_GetHostApiInfo (**PaHostApiIndex** *hostApi*)

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

Parameter

<i>hostApi</i>	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\(\)](#).

6.2.6.6 **PaHostApiIndex** Pa_HostApiTypeIdToHostApiIndex (**PaHostApiTypeId** *type*)

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

Parameter

<i>type</i>	A unique host API identifier belonging to the <code>PaHostApiTypeId</code> 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](#)

6.2.6.7 **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

<i>hostApi</i>	A valid host API index ranging from 0 to (Pa_GetHostApiCount() -1)
<i>hostApiDevice-Index</i>	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](#)

6.2.6.8 `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.

6.2.6.9 `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.

6.2.6.10 `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.

6.2.6.11 **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".

6.2.6.12 **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

<i>device</i>	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](#)

6.2.6.13 **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

<i>inputParameters</i>	A structure that describes the input parameters used to open a stream. The suggested-Latency field is ignored. See PaStreamParameters for a description of these parameters. <i>inputParameters</i> must be <code>NULL</code> for output-only streams.
------------------------	--

<i>outputParameters</i>	A structure that describes the output parameters used to open a stream. The suggested-Latency field is ignored. See PaStreamParameters for a description of these parameters. outputParameters must be NULL for input-only streams.
<i>sampleRate</i>	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](#)

6.2.6.14 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

<i>stream</i>	The address of a PaStream pointer which will receive a pointer to the newly opened stream.
<i>inputParameters</i>	A structure that describes the input parameters used by the opened stream. See PaStreamParameters for a description of these parameters. inputParameters must be NULL for output-only streams.
<i>outputParameters</i>	A structure that describes the output parameters used by the opened stream. See PaStreamParameters for a description of these parameters. outputParameters must be NULL for input-only streams.
<i>sampleRate</i>	The desired sampleRate. For full-duplex streams it is the sample rate for both input and output
<i>framesPerBuffer</i>	The number of frames passed to the stream callback function, or the preferred block granularity for a blocking read/write stream. The special value <code>paFramesPerBufferUnspecified</code> (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.
<i>streamFlags</i>	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.
<i>streamCallback</i>	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 <code>Pa_ReadStream</code> and write sample data using <code>Pa_WriteStream</code> , the number of samples that may be read or written without blocking is returned by <code>Pa_GetStreamReadAvailable</code> and <code>Pa_GetStreamWriteAvailable</code> respectively.

<i>userData</i>	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](#)

6.2.6.15 `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

<i>stream</i>	The address of a PaStream pointer which will receive a pointer to the newly opened stream.
<i>numInputChannels</i>	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.
<i>numOutputChannels</i>	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.
<i>sampleFormat</i>	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.
<i>sampleRate</i>	Same as Pa_OpenStream parameter of the same name.
<i>framesPerBuffer</i>	Same as Pa_OpenStream parameter of the same name.
<i>streamCallback</i>	Same as Pa_OpenStream parameter of the same name.
<i>userData</i>	Same as Pa_OpenStream parameter of the same name.

Rückgabe

As for [Pa_OpenStream](#)

Siehe auch

[Pa_OpenStream](#), [PaStreamCallback](#)

6.2.6.16 `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.

6.2.6.17 `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

<i>stream</i>	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.
<i>streamFinished-Callback</i>	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](#)

6.2.6.18 PaError Pa_StopStream (PaStream * *stream*)

Terminates audio processing.

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

6.2.6.19 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](#)

6.2.6.20 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](#)

6.2.6.21 `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

<i>stream</i>	A pointer to an open stream previously created with <code>Pa_OpenStream</code> .
---------------	--

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](#)

6.2.6.22 `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](#)

6.2.6.23 `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.

6.2.6.24 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

<i>stream</i>	A pointer to an open stream previously created with Pa_OpenStream.
<i>buffer</i>	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.
<i>frames</i>	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.

6.2.6.25 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

<i>stream</i>	A pointer to an open stream previously created with Pa_OpenStream.
<i>buffer</i>	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.
<i>frames</i>	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.

6.2.6.26 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.

6.2.6.27 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.

6.2.6.28 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.

6.2.6.29 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.

6.2.6.30 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.

Index

- ~Audio
 - Audio, [10](#)
- ~AudioControl
 - AudioControl, [15](#)
- ~Game
 - Game, [26](#)
- ~Input
 - Input, [35](#)
- ~PowerUp
 - PowerUp, [50](#)
- addEntry
 - Menu, [40](#)
- affectedObject
 - collisionStruct, [17](#)
- appendWorldObjects
 - Game, [29](#)
- Audio, [9](#)
 - ~Audio, [10](#)
 - Audio, [10](#)
 - getSample, [10](#)
 - getSampelenumber, [11](#)
 - getSource, [10](#)
 - normalize, [11](#)
 - readSamples, [11](#)
 - to16bitSample, [11](#)
- AudioControl::playStruct, [13](#)
- audioCooldownStruct, [61](#)
- audioCooldownstruct, [63](#)
- audioDistanceStruct, [62](#)
- audioStruct, [63](#)
- AudioControl, [12](#)
 - ~AudioControl, [15](#)
 - AudioControl, [14](#)
 - AudioControl, [14](#)
 - instancepaCallback, [15](#)
 - mtx, [16](#)
 - playInitialize, [15](#)
 - playTerminate, [15](#)
 - playeventsnumber, [16](#)
 - staticpaCallback, [16](#)
 - status_filter, [17](#)
 - updatePlayevents, [15](#)
- audioType
 - definitions.h, [64](#)
- background_highscore
 - definitions.h, [65](#)
- background_level1
 - definitions.h, [65](#)
- background_level2
 - definitions.h, [65](#)
- background_level3
 - definitions.h, [65](#)
- background_levelfinished
 - definitions.h, [65](#)
- background_menu
 - definitions.h, [65](#)
- background_startgame
 - definitions.h, [65](#)
- calculateMovement
 - Game, [29](#)
- causingObject
 - collisionStruct, [17](#)
- changeSelection
 - Menu, [41](#)
- clear
 - Menu, [39](#)
- collisionDirection
 - definitions.h, [64](#)
- collisionStruct, [17](#)
 - affectedObject, [17](#)
 - causingObject, [17](#)
 - direction, [17](#)
- compareGameObjects, [18](#)
- compareScores, [18](#)
- decreaseAlcoholLevel
 - Player, [46](#)
- definitions.h
 - background_highscore, [65](#)
 - background_level1, [65](#)
 - background_level2, [65](#)
 - background_level3, [65](#)
 - background_levelfinished, [65](#)
 - background_menu, [65](#)
 - background_startgame, [65](#)
 - player_jump, [65](#)
 - player_walk, [65](#)
 - powerup_beer, [65](#)
 - powerup_food, [65](#)
 - scene_collision_enemy, [65](#)
 - scene_collision_flyingbeer, [65](#)
 - scene_collision_obstacle, [64](#)
 - scene_collision_player, [65](#)
 - scene_enemy_boss, [64](#)
 - scene_enemy_security, [64](#)
 - scene_enemy_tourist, [64](#)
 - scene_flyingbeer, [64](#)

- status_alcohol, 65
- status_dead, 65
- status_life, 65
- status_lifecritical, 65
- definitions.h
 - audioType, 64
 - collisionDirection, 64
 - gameState, 64
 - objectType, 64
 - spawnDistance, 65
- detectCollision
 - Game, 30
- direction
 - collisionStruct, 17
- displayInit
 - Menu, 40
- displayStatistics
 - Game, 28
- displayUpdate
 - Menu, 40
- endGame
 - Game, 28
- Enemy, 19
 - Enemy, 20
 - flipHorizontal, 22
 - getDeath, 21
 - getDeathCooldown, 21
 - getFireCooldown, 21
 - getHealth, 20
 - getInflictedDamage, 21
 - setDeath, 21
 - setHealth, 20
 - swapImage, 22
 - updateFramesDirection, 21
 - updatePosition, 22
- evaluateInput
 - Game, 29
- evaluateKeyEvent
 - Input, 35
- eventFilter
 - Game, 31
- exitGame
 - Game, 31
- flipHorizontal
 - Enemy, 22
 - MovingObject, 43
 - Player, 48
 - Shoot, 58
- Game, 22
 - ~Game, 26
 - appendWorldObjects, 29
 - calculateMovement, 29
 - detectCollision, 30
 - displayStatistics, 28
 - endGame, 28
 - evaluateInput, 29
 - eventFilter, 31
 - exitGame, 31
 - Game, 26
 - getStepIntervall, 31
 - handleCollisions, 30
 - levelSpawn, 32
 - loadLevelFile, 28
 - menuInit, 31
 - reduceWorldObjects, 29
 - renderGraphics, 30
 - setState, 27
 - start, 26
 - startNewGame, 27
 - step, 26
 - timerEvent, 27
 - updateAudioevents, 30
 - updateHighScore, 28
 - updateScore, 30
- GameObject, 32
 - GameObject, 33
 - GameObject, 33
- gameState
 - definitions.h, 64
- getAlcoholLevel
 - Player, 46
- getAlcoholLevelBonus
 - PowerUp, 50
- getAmmunationBonus
 - PowerUp, 51
- getAmmunatiuon
 - Player, 47
- getDeath
 - Enemy, 21
- getDeathCooldown
 - Enemy, 21
- getEntry
 - Menu, 41
- getFireCooldown
 - Enemy, 21
 - Player, 47
- getHealth
 - Enemy, 20
 - Player, 46
- getHealthBonus
 - PowerUp, 50
- getImmunityCooldown
 - Player, 47
- getImmunityCooldownBonus
 - PowerUp, 51
- getInflictedDamage
 - Enemy, 21
 - Player, 47
 - Shoot, 57
- getKeyactions
 - Input, 35
- getKeyletters
 - Input, 36
- getLastKeyaction

- Input, [36](#)
- getLastKeyLetter
 - Input, [36](#)
- getOrigin
 - Shoot, [57](#)
- getPowerUPType
 - PowerUp, [51](#)
- getSample
 - Audio, [10](#)
- getSamplenummer
 - Audio, [11](#)
- getSelection
 - Menu, [41](#)
- getSource
 - Audio, [10](#)
- getStepIntervall
 - Game, [31](#)
- getTitle
 - Menu, [40](#)
- getType
 - Menu, [40](#)
- handleCollisions
 - Game, [30](#)
- inJump
 - Player, [47](#)
- increaseAlcoholLevel
 - Player, [46](#)
- Input, [33](#)
 - ~Input, [35](#)
 - evaluateKeyEvent, [35](#)
 - getKeyactions, [35](#)
 - getKeyletters, [36](#)
 - getLastKeyaction, [36](#)
 - getLastKeyLetter, [36](#)
 - Input, [35](#)
 - updateKeys, [37](#)
- instancepaCallback
 - AudioControl, [15](#)
- levelSpawn
 - Game, [32](#)
- loadLevelFile
 - Game, [28](#)
- Menu, [37](#)
 - addEntry, [40](#)
 - changeSelection, [41](#)
 - clear, [39](#)
 - displayInit, [40](#)
 - displayUpdate, [40](#)
 - getEntry, [41](#)
 - getSelection, [41](#)
 - getTitle, [40](#)
 - getType, [40](#)
 - Menu, [39](#)
 - selectFirstEntry, [41](#)
- Menu::menuEntry, [39](#)
- menuInit
 - Game, [31](#)
- MovingObject, [42](#)
 - flipHorizontal, [43](#)
 - swapImage, [43](#)
 - updateFramesDirection, [43](#)
 - updatePosition, [43](#)
- mtx
 - AudioControl, [16](#)
- normalize
 - Audio, [11](#)
- objectType
 - definitions.h, [64](#)
- paAbort
 - portaudio.h, [80](#)
- paComplete
 - portaudio.h, [80](#)
- paContinue
 - portaudio.h, [80](#)
- PaDeviceInfo, [70](#)
- PaHostApiInfo, [69](#)
- PaHostErrorInfo, [70](#)
- PaStreamCallbackTimeInfo, [72](#)
- PaStreamInfo, [72](#)
- PaStreamParameters, [71](#)
- Pa_CloseStream
 - portaudio.h, [86](#)
- Pa_GetDefaultHostApi
 - portaudio.h, [81](#)
- Pa_GetDefaultInputDevice
 - portaudio.h, [83](#)
- Pa_GetDefaultOutputDevice
 - portaudio.h, [83](#)
- Pa_GetDeviceCount
 - portaudio.h, [83](#)
- Pa_GetDeviceInfo
 - portaudio.h, [84](#)
- Pa_GetHostApiCount
 - portaudio.h, [81](#)
- Pa_GetHostApiInfo
 - portaudio.h, [82](#)
- Pa_GetLastHostErrorInfo
 - portaudio.h, [83](#)
- Pa_GetSampleSize
 - portaudio.h, [90](#)
- Pa_GetStreamCpuLoad
 - portaudio.h, [88](#)
- Pa_GetStreamHostApiType
 - portaudio.h, [90](#)
- Pa_GetStreamInfo
 - portaudio.h, [87](#)
- Pa_GetStreamReadAvailable
 - portaudio.h, [89](#)
- Pa_GetStreamTime
 - portaudio.h, [88](#)
- Pa_GetStreamWriteAvailable

- portaudio.h, [89](#)
- Pa_HostApiDeviceIndexToDeviceIndex
 - portaudio.h, [82](#)
- Pa_HostApiTypeIdToHostApiIndex
 - portaudio.h, [82](#)
- Pa_Initialize
 - portaudio.h, [81](#)
- Pa_IsFormatSupported
 - portaudio.h, [84](#)
- Pa_IsStreamActive
 - portaudio.h, [87](#)
- Pa_IsStreamStopped
 - portaudio.h, [87](#)
- Pa_OpenDefaultStream
 - portaudio.h, [86](#)
- Pa_OpenStream
 - portaudio.h, [85](#)
- Pa_ReadStream
 - portaudio.h, [88](#)
- Pa_SetStreamFinishedCallback
 - portaudio.h, [86](#)
- Pa_Sleep
 - portaudio.h, [90](#)
- Pa_StopStream
 - portaudio.h, [87](#)
- Pa_Terminate
 - portaudio.h, [81](#)
- Pa_WriteStream
 - portaudio.h, [89](#)
- paClipOff
 - portaudio.h, [74](#)
- paCustomFormat
 - portaudio.h, [74](#)
- PaDeviceIndex
 - portaudio.h, [76](#)
- PaDeviceInfo
 - portaudio.h, [77](#)
- paDitherOff
 - portaudio.h, [74](#)
- PaError
 - portaudio.h, [76](#)
- paFloat32
 - portaudio.h, [73](#)
- paFormatIsSupported
 - portaudio.h, [74](#)
- PaHostApiIndex
 - portaudio.h, [76](#)
- PaHostApiInfo
 - portaudio.h, [77](#)
- PaHostApiTypeId
 - portaudio.h, [76](#), [80](#)
- paInputOverflow
 - portaudio.h, [75](#)
- paInputUnderflow
 - portaudio.h, [75](#)
- paInt16
 - portaudio.h, [73](#)
- paInt24
 - portaudio.h, [73](#)
- paInt32
 - portaudio.h, [73](#)
- paInt8
 - portaudio.h, [73](#)
- paNeverDropInput
 - portaudio.h, [74](#)
- paNoDevice
 - portaudio.h, [73](#)
- paNoFlag
 - portaudio.h, [74](#)
- paNonInterleaved
 - portaudio.h, [74](#)
- paOutputOverflow
 - portaudio.h, [76](#)
- paOutputUnderflow
 - portaudio.h, [75](#)
- paPlatformSpecificFlags
 - portaudio.h, [75](#)
- paPrimeOutputBuffersUsingStreamCallback
 - portaudio.h, [75](#)
- paPrimingOutput
 - portaudio.h, [76](#)
- PaSampleFormat
 - portaudio.h, [77](#)
- PaStream
 - portaudio.h, [77](#)
- PaStreamCallback
 - portaudio.h, [78](#)
- PaStreamCallbackFlags
 - portaudio.h, [78](#)
- PaStreamCallbackResult
 - portaudio.h, [78](#), [80](#)
- PaStreamCallbackTimeInfo
 - portaudio.h, [78](#)
- PaStreamFinishedCallback
 - portaudio.h, [79](#)
- PaStreamFlags
 - portaudio.h, [78](#)
- PaStreamInfo
 - portaudio.h, [80](#)
- PaTime
 - portaudio.h, [77](#)
- paUInt8
 - portaudio.h, [74](#)
- paUseHostApiSpecificDeviceSpecification
 - portaudio.h, [73](#)
- playInitialize
 - AudioControl, [15](#)
- playTerminate
 - AudioControl, [15](#)
- Player, [44](#)
 - decreaseAlcoholLevel, [46](#)
 - flipHorizontal, [48](#)
 - getAlcoholLevel, [46](#)
 - getAmmunatiuon, [47](#)
 - getFireCooldown, [47](#)
 - getHealth, [46](#)

- getImmunityCooldown, 47
- getInflictedDamage, 47
- inJump, 47
- increaseAlcoholLevel, 46
- receiveDamage, 46
- setHealth, 46
- setImmunityCooldown, 47
- swapImage, 48
- update, 47
- updateFramesDirection, 48
- updatePosition, 48
- player_jump
 - definitions.h, 65
- player_walk
 - definitions.h, 65
- playeventsnumber
 - AudioControl, 16
- portaudio.h
 - paAbort, 80
 - paComplete, 80
 - paContinue, 80
- portaudio.h
 - Pa_CloseStream, 86
 - Pa_GetDefaultHostApi, 81
 - Pa_GetDefaultInputDevice, 83
 - Pa_GetDefaultOutputDevice, 83
 - Pa_GetDeviceCount, 83
 - Pa_GetDeviceInfo, 84
 - Pa_GetHostApiCount, 81
 - Pa_GetHostApiInfo, 82
 - Pa_GetLastHostErrorInfo, 83
 - Pa_GetSampleSize, 90
 - Pa_GetStreamCpuLoad, 88
 - Pa_GetStreamHostApiType, 90
 - Pa_GetStreamInfo, 87
 - Pa_GetStreamReadAvailable, 89
 - Pa_GetStreamTime, 88
 - Pa_GetStreamWriteAvailable, 89
 - Pa_HostApiDeviceIndexToDeviceIndex, 82
 - Pa_HostApiTypeIdToHostApiIndex, 82
 - Pa_Initialize, 81
 - Pa_IsFormatSupported, 84
 - Pa_IsStreamActive, 87
 - Pa_IsStreamStopped, 87
 - Pa_OpenDefaultStream, 86
 - Pa_OpenStream, 85
 - Pa_ReadStream, 88
 - Pa_SetStreamFinishedCallback, 86
 - Pa_Sleep, 90
 - Pa_StopStream, 87
 - Pa_Terminate, 81
 - Pa_WriteStream, 89
 - paClipOff, 74
 - paCustomFormat, 74
 - PaDeviceIndex, 76
 - PaDeviceInfo, 77
 - paDitherOff, 74
 - PaError, 76
 - paFloat32, 73
 - paFormatIsSupported, 74
 - PaHostApiIndex, 76
 - PaHostApiInfo, 77
 - PaHostApiTypeId, 76, 80
 - paInputOverflow, 75
 - paInputUnderflow, 75
 - paInt16, 73
 - paInt24, 73
 - paInt32, 73
 - paInt8, 73
 - paNeverDropInput, 74
 - paNoDevice, 73
 - paNoFlag, 74
 - paNonInterleaved, 74
 - paOutputOverflow, 76
 - paOutputUnderflow, 75
 - paPlatformSpecificFlags, 75
 - paPrimeOutputBuffersUsingStreamCallback, 75
 - paPrimingOutput, 76
 - PaSampleFormat, 77
 - PaStream, 77
 - PaStreamCallback, 78
 - PaStreamCallbackFlags, 78
 - PaStreamCallbackResult, 78, 80
 - PaStreamCallbackTimeInfo, 78
 - PaStreamFinishedCallback, 79
 - PaStreamFlags, 78
 - PaStreamInfo, 80
 - PaTime, 77
 - paUInt8, 74
 - paUseHostApiSpecificDeviceSpecification, 73
- PowerUp, 49
 - ~PowerUp, 50
 - getAlcoholLevelBonus, 50
 - getAmmunationBonus, 51
 - getHealthBonus, 50
 - getImmunityCooldownBonus, 51
 - getPowerUPType, 51
 - PowerUp, 50
 - PowerUp, 50
- powerup_beer
 - definitions.h, 65
- powerup_food
 - definitions.h, 65
- readSamples
 - Audio, 11
- receiveDamage
 - Player, 46
- reduceWorldObjects
 - Game, 29
- RenderBackground, 51
 - RenderBackground, 52
 - RenderBackground, 52
 - setPos, 52
 - updateBackgroundPos, 54
 - updateParallaxe, 52
- RenderGUI, 54

- RenderGUI, 55
- RenderGUI, 55
- setPos, 55
- setValues, 55
- renderGraphics
 - Game, 30
- scene_collision_enemy
 - definitions.h, 65
- scene_collision_flyingbeer
 - definitions.h, 65
- scene_collision_obstacle
 - definitions.h, 64
- scene_collision_player
 - definitions.h, 65
- scene_enemy_boss
 - definitions.h, 64
- scene_enemy_security
 - definitions.h, 64
- scene_enemy_tourist
 - definitions.h, 64
- scene_flyingbeer
 - definitions.h, 64
- scoreStruct, 61
- selectFirstEntry
 - Menu, 41
- setDeath
 - Enemy, 21
- setHealth
 - Enemy, 20
 - Player, 46
- setImmunityCooldown
 - Player, 47
- setPos
 - RenderBackground, 52
 - RenderGUI, 55
- setState
 - Game, 27
- setValues
 - RenderGUI, 55
- Shoot, 56
 - flipHorizontal, 58
 - getInflictedDamage, 57
 - getOrigin, 57
 - Shoot, 57
 - swapImage, 58
 - updateFramesDirection, 58
 - updatePosition, 58
- spawnDistance
 - definitions.h, 65
- start
 - Game, 26
- startNewGame
 - Game, 27
- stateStruct, 63
- staticpaCallback
 - AudioControl, 16
- status_alcohol
 - definitions.h, 65
- status_dead
 - definitions.h, 65
- status_life
 - definitions.h, 65
- status_lifecritical
 - definitions.h, 65
- status_filter
 - AudioControl, 17
- step
 - Game, 26
- swapImage
 - Enemy, 22
 - MovingObject, 43
 - Player, 48
 - Shoot, 58
- timerEvent
 - Game, 27
- to16bitSample
 - Audio, 11
- update
 - Player, 47
- updateAudioevents
 - Game, 30
- updateBackgroundPos
 - RenderBackground, 54
- updateFramesDirection
 - Enemy, 21
 - MovingObject, 43
 - Player, 48
 - Shoot, 58
- updateHighScore
 - Game, 28
- updateKeys
 - Input, 37
- updateParallaxe
 - RenderBackground, 52
- updatePlayevents
 - AudioControl, 15
- updatePosition
 - Enemy, 22
 - MovingObject, 43
 - Player, 48
 - Shoot, 58
- updateScore
 - Game, 30
- Wiesn-Run/src/definitions.h, 59
- Wiesn-Run/src/portaudio.h, 65