

Wiesn-Run

Erzeugt von Doxygen 1.8.6

Mit Jul 15 2015 09:29:52

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	11
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	14
5.2.2.1	struct AudioControl::playStruct	14
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	16
5.2.4.3	updatePlayevents	16
5.2.4.4	instancepaCallback	16
5.2.4.5	staticpaCallback	16
5.2.5	Dokumentation der Datenelemente	18
5.2.5.1	mtx	18
5.2.5.2	status_filter	18
5.3	collisionStruct Strukturreferenz	18
5.3.1	Ausführliche Beschreibung	19
5.3.2	Dokumentation der Datenelemente	19
5.3.2.1	affectedObject	19
5.3.2.2	causingObject	19
5.3.2.3	direction	19
5.4	compareGameObjects Strukturreferenz	19
5.4.1	Ausführliche Beschreibung	19
5.5	compareScores Strukturreferenz	20
5.5.1	Ausführliche Beschreibung	20
5.6	Enemy Klassenreferenz	20
5.6.1	Ausführliche Beschreibung	22
5.6.2	Beschreibung der Konstruktoren und Destruktoren	22
5.6.2.1	Enemy	22
5.6.3	Dokumentation der Elementfunktionen	23
5.6.3.1	getHealth	23
5.6.3.2	setHealth	23
5.6.3.3	receiveDamage	23
5.6.3.4	getInflictedDamage	23
5.6.3.5	getFireCooldown	23
5.6.3.6	getDeath	24
5.6.3.7	setDeath	24
5.6.3.8	getDeathCooldown	24
5.6.3.9	update	24
5.6.3.10	setPosX	24
5.6.3.11	setPosY	24
5.6.3.12	getSpeedX	25
5.6.3.13	getSpeedY	25
5.6.3.14	setSpeedX	25
5.6.3.15	setSpeedY	25
5.6.3.16	updateFramesDirection	25
5.6.3.17	flipHorizontal	25
5.6.3.18	swapImage	26

5.6.3.19	updatePosition	26
5.6.3.20	getPosX	26
5.6.3.21	getPosY	26
5.6.3.22	getLength	26
5.6.3.23	getHeight	26
5.6.3.24	getType	27
5.6.3.25	setAudioID	27
5.6.3.26	getAudioID	27
5.7	Game Klassenreferenz	27
5.7.1	Ausführliche Beschreibung	31
5.7.2	Dokumentation der Elementfunktionen	31
5.7.2.1	step	31
5.7.2.2	start	32
5.7.2.3	setState	32
5.7.2.4	timerEvent	32
5.7.2.5	startNewGame	33
5.7.2.6	loadLevelFile	33
5.7.2.7	updateHighScore	33
5.7.2.8	displayStatistics	33
5.7.2.9	endGame	34
5.7.2.10	appendWorldObjects	34
5.7.2.11	reduceWorldObjects	34
5.7.2.12	evaluateInput	34
5.7.2.13	calculateMovement	35
5.7.2.14	detectCollision	35
5.7.2.15	handleCollisions	35
5.7.2.16	updateScore	35
5.7.2.17	updateAudioevents	36
5.7.2.18	renderGraphics	36
5.7.2.19	menuInit	36
5.7.2.20	exitGame	36
5.7.2.21	eventFilter	36
5.7.2.22	getStepIntervall	38
5.7.2.23	timeNeeded	38
5.8	GameObject Klassenreferenz	38
5.8.1	Ausführliche Beschreibung	39
5.8.2	Beschreibung der Konstruktoren und Destruktoren	40
5.8.2.1	GameObject	40
5.8.2.2	GameObject	40
5.8.3	Dokumentation der Elementfunktionen	40

5.8.3.1	getPosX	40
5.8.3.2	getPosY	40
5.8.3.3	getLength	41
5.8.3.4	getHeight	41
5.8.3.5	getType	41
5.8.3.6	setAudioID	41
5.8.3.7	getAudioID	41
5.9	Input Klassenreferenz	41
5.9.1	Ausführliche Beschreibung	43
5.9.2	Beschreibung der Konstruktoren und Destrukturen	43
5.9.2.1	Input	43
5.9.2.2	~Input	43
5.9.3	Dokumentation der Elementfunktionen	43
5.9.3.1	evaluateKeyEvent	43
5.9.3.2	getKeyactions	45
5.9.3.3	getKeyletters	45
5.9.3.4	getLastKeyaction	45
5.9.3.5	getLastKeyletter	46
5.9.3.6	updateKeys	46
5.10	Menu Klassenreferenz	46
5.10.1	Ausführliche Beschreibung	48
5.10.2	Klassen-Dokumentation	48
5.10.2.1	struct Menu::menuEntry	48
5.10.3	Beschreibung der Konstruktoren und Destrukturen	48
5.10.3.1	Menu	48
5.10.4	Dokumentation der Elementfunktionen	48
5.10.4.1	clear	48
5.10.4.2	getType	49
5.10.4.3	getTitle	49
5.10.4.4	displayInit	49
5.10.4.5	displayUpdate	49
5.10.4.6	addEntry	49
5.10.4.7	changeSelection	50
5.10.4.8	getSelection	50
5.10.4.9	getEntry	50
5.10.4.10	selectFirstEntry	50
5.11	MovingObject Klassenreferenz	51
5.11.1	Ausführliche Beschreibung	52
5.11.2	Beschreibung der Konstruktoren und Destrukturen	52
5.11.2.1	MovingObject	52

5.11.3	Dokumentation der Elementfunktionen	53
5.11.3.1	setPosX	53
5.11.3.2	setPosY	53
5.11.3.3	getSpeedX	53
5.11.3.4	getSpeedY	53
5.11.3.5	setSpeedX	53
5.11.3.6	setSpeedY	54
5.11.3.7	updateFramesDirection	54
5.11.3.8	flipHorizontal	54
5.11.3.9	swapImage	54
5.11.3.10	updatePosition	54
5.11.3.11	getPosX	55
5.11.3.12	getPosY	55
5.11.3.13	getLength	55
5.11.3.14	getHeight	55
5.11.3.15	getType	55
5.11.3.16	setAudioID	55
5.11.3.17	getAudioID	56
5.12	Player Klassenreferenz	56
5.12.1	Ausführliche Beschreibung	58
5.12.2	Beschreibung der Konstruktoren und Destruktoren	59
5.12.2.1	Player	59
5.12.3	Dokumentation der Elementfunktionen	59
5.12.3.1	getHealth	59
5.12.3.2	setHealth	59
5.12.3.3	receiveDamage	59
5.12.3.4	increaseAlcoholLevel	60
5.12.3.5	decreaseAlcoholLevel	60
5.12.3.6	getAmmunition	60
5.12.3.7	increaseAmmunition	60
5.12.3.8	getInflictedDamage	60
5.12.3.9	getFireCooldown	61
5.12.3.10	getImmunityCooldown	61
5.12.3.11	setImmunityCooldown	61
5.12.3.12	startJump	61
5.12.3.13	inJump	61
5.12.3.14	getEnemiesKilled	61
5.12.3.15	getSpeedScale	61
5.12.3.16	update	62
5.12.3.17	setPosX	62

5.12.3.18 setPosY	62
5.12.3.19 getSpeedX	62
5.12.3.20 getSpeedY	62
5.12.3.21 setSpeedX	63
5.12.3.22 setSpeedY	63
5.12.3.23 updateFramesDirection	63
5.12.3.24 flipHorizontal	63
5.12.3.25 swapImage	63
5.12.3.26 updatePosition	64
5.12.3.27 getPosX	64
5.12.3.28 getPosY	64
5.12.3.29 getLength	64
5.12.3.30 getHeight	64
5.12.3.31 getType	64
5.12.3.32 setAudioID	64
5.12.3.33 getAudioID	65
5.13 PowerUp Klassenreferenz	65
5.13.1 Ausführliche Beschreibung	66
5.13.2 Beschreibung der Konstruktoren und Destruktoren	66
5.13.2.1 PowerUp	66
5.13.2.2 ~PowerUp	67
5.13.3 Dokumentation der Elementfunktionen	67
5.13.3.1 getHealthBonus	67
5.13.3.2 getAlcoholLevelBonus	67
5.13.3.3 getAmmunationBonus	67
5.13.3.4 getImmunityCooldownBonus	67
5.13.3.5 getPowerUPType	68
5.13.3.6 getPosX	68
5.13.3.7 getPosY	68
5.13.3.8 getLength	68
5.13.3.9 getHeight	68
5.13.3.10 getType	68
5.13.3.11 setAudioID	69
5.13.3.12 getAudioID	70
5.14 RenderBackground Klassenreferenz	70
5.14.1 Ausführliche Beschreibung	70
5.14.2 Beschreibung der Konstruktoren und Destruktoren	71
5.14.2.1 RenderBackground	71
5.14.3 Dokumentation der Elementfunktionen	71
5.14.3.1 setPos	71

5.14.3.2	updateParallaxe	71
5.14.3.3	updateBackgroundPos	71
5.15	RenderGUI Klassenreferenz	72
5.15.1	Ausführliche Beschreibung	72
5.15.2	Beschreibung der Konstruktoren und Destruktoren	73
5.15.2.1	RenderGUI	73
5.15.3	Dokumentation der Elementfunktionen	73
5.15.3.1	setPos	73
5.15.3.2	setValues	73
5.16	Shoot Klassenreferenz	74
5.16.1	Ausführliche Beschreibung	75
5.16.2	Beschreibung der Konstruktoren und Destruktoren	76
5.16.2.1	Shoot	76
5.16.3	Dokumentation der Elementfunktionen	76
5.16.3.1	getInflictedDamage	76
5.16.3.2	getOrigin	76
5.16.3.3	getHarming	76
5.16.3.4	setToDelete	76
5.16.3.5	setPosX	76
5.16.3.6	setPosY	77
5.16.3.7	getSpeedX	77
5.16.3.8	getSpeedY	77
5.16.3.9	setSpeedX	77
5.16.3.10	setSpeedY	77
5.16.3.11	updateFramesDirection	77
5.16.3.12	flipHorizontal	78
5.16.3.13	swapImage	78
5.16.3.14	updatePosition	78
5.16.3.15	getPosX	78
5.16.3.16	getPosY	78
5.16.3.17	getLength	79
5.16.3.18	getHeight	79
5.16.3.19	getType	79
5.16.3.20	setAudioID	79
5.16.3.21	getAudioID	79
6	Datei-Dokumentation	81
6.1	Wiesn-Run/src/definitions.h-Dateireferenz	81
6.1.1	Ausführliche Beschreibung	82
6.1.2	Klassen-Dokumentation	83

6.1.2.1	struct scoreStruct	83
6.1.2.2	struct audioCooldownStruct	83
6.1.2.3	struct audioDistanceStruct	84
6.1.2.4	struct audioStruct	85
6.1.2.5	struct audioCooldownstruct	86
6.1.2.6	struct stateStruct	86
6.1.3	Dokumentation der Aufzählungstypen	86
6.1.3.1	gameState	86
6.1.3.2	objectType	87
6.1.3.3	collisionDirection	87
6.1.3.4	audioType	87
6.1.4	Variablen-Dokumentation	88
6.1.4.1	spawnDistance	88
Index		89

Kapitel 1

Ausstehende Aufgaben

Element `Game::detectCollision` (`std::list< GameObject * > *objectsToCalculate`)

Simon, willst du hier die normalen Kommentare noch einarbeiten und die Funktion genauer erklären? - Rupl

Autor

Simon

Element `Game::renderGraphics` (`std::list< GameObject * > *objectList`, `Player *playerPointer`)

Flo, willst du hier die normalen Kommentare noch einarbeiten und die Funktion genauer erklären? - Rupl

Element `Game::step` ()

Erfolgreich Schriftzug einfügen

GameOver schriftzug einfügen

Klasse `scoreStruct`

Das Konzept der Alkohol-Punkte muss noch ausgearbeitet werden.

Autor

Simon

Kapitel 2

Hierarchie-Verzeichnis

2.1 Klassenhierarchie

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

Audio	9
AudioControl	12
audioCooldownstruct	81
audioCooldownStruct	81
audioDistanceStruct	81
audioStruct	81
collisionStruct	18
compareGameObjects	19
compareScores	20
Input	41
Menu	46
Menu::menuEntry	46
AudioControl::playStruct	12
QGraphicsPixmapItem	
GameObject	38
MovingObject	51
Enemy	20
Player	56
Shoot	74
PowerUp	65
QObject	
Game	27
RenderBackground	70
RenderGUI	72
scoreStruct	81
stateStruct	81

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, welche Wave Samples und deren Meta Information speichern	9
AudioControl	Die AudioControl Klasse synchronisiert alle aktuellen Audioausgabeanweisungen des Game Objekt mit dem PortAudio Wiedergabe Thread und ermöglicht das blockweise Abspielen von Samples unter Einbezug der Distanzinformationen Spieler zum audioEvent (2D Audio)	12
collisionStruct	Struktur für die Events	18
compareGameObjects	Vergleich zweier GameObjects bezüglich der X-Position	19
compareScores	Vergleich zweier Scores	20
Enemy	Das Gegner Objekt	20
Game	Kern-Funktionalität des Spiels	27
GameObject	Das Spieler-Objekt	38
Input	Die Input-Klasse aktualisiert die für das Spiel relevanten Tastatureingaben	41
Menu	Klasse zum Erzeugen und Anzeigen von Spielmenüs	46
MovingObject	Das Moving-Object	51
Player	Das Spieler-Objekt	56
PowerUp	Klasse für Power-Ups	65
RenderBackground	Hintergrund-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt	70
RenderGUI	Anzeigen der Spielerwerte-Klasse Eine Instanz wird bei jedem Levelstart in der Funktion Game::startNewGame angelegt	72
Shoot	Das Schuss Objekt	74

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	81
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/ 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, welche Wave Samples und deren Meta Information speichern.

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

Öffentliche Methoden

- [Audio](#) (std::string type_name)
Konstruktor instanziiert 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 Dateinamen der zugehörigen WAVE Datei entspricht.
- float [getSample](#) (int pos)
getSample gibt bei Aufruf das Sample an Position = pos der zu Audioobjekt gehörigen Wave Datei mit Bittiefe 32 bit float 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 float vector 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 QVektor 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, welche Wave Samples und deren Meta Information speichern.

Für jedes [Audio](#) Wave File im Ordner [Audio](#) (AudioEventgruppe [audioType](#)) wird zum Spielstart in [Game](#) über den Konstruktor des [AudioControl](#) Objekt [audioOutput](#) [Audio\(std::string type_name\)](#) eine Instanz der Klasse erstellt und das erstellte Objekt in [audioOutput](#) an eine Liste [audioobjects](#) angehängt. Bei der Erstellung liest das Audioobjekt die zum [Audio](#) Objekt gehörigen Samples byteweise aus der gespeicherten WAVE Datei in den Vektor [samples](#) ein, wobei die Samples normalisiert werden. Das Eingabeformat sind Einkanal little-endian RIFF Wave Dateien mit 16 bit oder 8 bit Bittiefe und einer Samplerate von 44100 Hz. Das [audioControl](#) Objekt [audioOutput](#) kann auf ein bestimmtes Sample über die Methode [getSample\(int pos\)](#) zugreifen. Die Anzahl aller Samples im Vektor [samples](#) kann über [getSamplenumber\(\)](#) abgefragt werden.

Autor

Felix Pfreundtner

5.1.2 Beschreibung der Konstruktoren und Destruktoren

5.1.2.1 [Audio::Audio \(std::string type_name \)](#)

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

Wird mehrmals zum Spielstart von dem [audioControl](#) Objekt [audioOutput](#) aufgerufen.

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 dem Dateinamen der zugehörigen WAVE Datei 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 32 bit float 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 Methode 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 float vector 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:

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

5.2 AudioControl Klassenreferenz

Die [AudioControl](#) Klasse synchronisiert alle aktuellen Audioausgabeanweisungen des [Game](#) Objekt mit dem Port-Audio Wiedergabe Thread und ermöglicht das blockweise Abspielen von Samples unter Einbezug der Distanzinformationen Spieler zum audioEvent (2D [Audio](#)).

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

Klassen

- struct [playStruct](#)
[playStruct](#) definiert die Struktur eines playEvents [Mehr ...](#)

Öffentliche Methoden

- [AudioControl](#) ()
Konstruktor instanziiert ein Objekt der Klasse [AudioControl](#).
- [~AudioControl](#) ()
Destruktor löscht ein Objekt der Klasse [AudioControl](#).
- void [playInitialize](#) ()
[playInitialize](#) initialisiert die Abspielbibliothek Portaudio, öffnet einen PortAudio Stream 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 audioStructs 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 *timelInfo, PaStreamCallbackFlags statusFlags)

instancepaCallback wird von Portaudio aufgerufen wenn der letzte Audioblock abgespielt wurde und ein neuer Ausgabe Audioblock geniert werden muss.

Private, statische Methoden

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

staticpaCallback ist die statische Callback Methode 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 vector mit allen vorhandenen Objekten der Klasse [Audio](#).
- int [waitinms](#)
waitinms speichert die Wartezeit bis zum Beenden von PortAudio in Millisekunden.
- int [max_playevents](#)
- int [blockcounter](#)
blockcounter zählt die bereits abgespielten [Audio](#) Ausgabe Blöcke.
- float [mixed_sample](#)
mixed_sample beinhaltet das aktuell gemischte 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 aktuellen Audioausgabeanweisungen des [Game](#) Objekt mit dem Port-Audio Wiedergabe Thread und ermöglicht das blockweise Abspielen von Samples unter Einbezug der Distanzinformationen Spieler zum audioEvent (2D [Audio](#)).

Eine Instanz der Klasse [AudioControl](#) [audioOutput](#) wird zum Spielstart im [game](#) Objekt erstellt. Beim Erstellen wird über den Konstruktor für alle im [Audio](#) Ordner gespeicherten WAVE Dateien (WAVE Datei -> [audioEventgruppe](#) [audioType](#)) ein Objekt der Klasse [Audio](#) erstellt und an die Liste [audioobjects](#) angehängt. Der Datentyp eines [audioEvents](#) ist ein [audioStruct](#), welches eine ID, einen [audioType](#) und eine Distanzinformation besitzt, genaueres siehe [Definitions.h](#). Der Datentyp der zugehörigen Abspielinformation ist ein [playStruct](#), welches eine id, einen [audioType](#), eine Lautstärkeinformation, einen Zeiger auf das passende [Audio](#) Objekt mit den Samples und eine aktuelle Abspielposition in Samples besitzt. In jedem Spiel-Step erzeugt das [Game](#) Objekt alle im Moment abzuspielenden [audioEvents](#) über die Funktion [updateaudioEvents\(\)](#) und übergibt die Liste [audioEvents](#) der [updatePlayevents\(\)](#) Methode. Jedes [audioStruct](#) wird aus der Liste entnommen und dessen id mit allen ids, der im Moment in der

Liste playEvents stehenden playStructs, verglichen. Die Distanzinformation "|Spieler - Ort audioEvent|" des [audioStruct](#) wird linear in eine Volumeninformation umgerechnet. Ist das zum [audioStruct](#) zugehörige [playStruct](#) bereits vorhanden, so wird nur die Distanzinformation aktualisiert. Ist kein passendes [playStruct](#) vorhanden wird ein neues [playStruct](#) aus der [audioStruct](#) Information erzeugt und der Liste playevents angehängt. Wird ein vorher abgespieltes [playStruct](#) nun nicht mehr vom [Game](#) Objekt als [audioStruct](#) gesendet, wird das [playStruct](#) aus der Liste playevents entfernt und das audioEvent nicht weiter abgespielt.

Für die [Audio](#) Wiedergabe wurde die Bibliothek PortAudio gewählt. Sie ermöglicht eine Low Level Audioausgabe auf Sampleebene. PortAudio wird zum Programmstart über die Methode playInitialize als Thread gestartet, was eine gleichzeitige Erstellung von audioEvents im Spielablauf durch das [Game](#) Objekt und Rückgriff auf die aktuellen, in der Instanzvariable playevents, stehenden Ausgabeinformationen über eine Instanz Callbackfunktion [instancepaCallback\(\)](#) ermöglicht. Das Abspielen erfolgt unabhängig vom Step Takt des [Game](#) Objekts, was die Stabilität der Wiedergabe garantiert. Die Audioausgabe wird hierbei blockweise mit Blockgröße 1024 Samples (44100 Hz Samplingfrequenz) erstellt. Die statische Callback Methode der Klasse [staticpaCallback\(\)](#) wird von PortAudio automatisch immer dann aufgerufen, wenn der PortAudio Stream einen neuen Ausgabeblock benötigt, da der letzte abgespielt wurde. Die Methode ruft im Anschluss die Instanz Callback Funktion [instancepaCallback\(\)](#) des [AudioControl](#) Objekts audioOutput auf, welche über eine Mutex auf die Instanzvariable playevents zugreift. Bei jedem Aufruf wird in [instancepaCallback\(\)](#) die Liste playevents ausgewertet und für jedes [playStruct](#) über den Zeiger auf das zugehörige [Audio](#) Objekt mit [getSample\(\)](#) ein Sample eingelesen. Durch Multiplikation mit der aktuellen im [playStruct](#) stehenden Volumeninformation wird die Distanz des audioEvents zum Spieler in jedem Block neu berücksichtigt. Nach Auswerten eines Samples wird im [playStruct](#) der Positionswert um 1 erhöht. Ist der Positionswert größer als [getSampleNumber\(\)](#) wird die Position auf 0 gesetzt und das audioEvent wieder von Beginn an abgespielt (Loop). Das Gleiche wird für die anderen playStructs in der Liste playevents durchgeführt. Durch Summation aller Samples wird der aktuelle Ausgabeblock gemischt, welcher von PortAudio wiedergegeben wird. Es wurden zudem zwei Filter Effekte programmiert, welche auftreten wenn der Spieler betrunken ist (Audiowiedergabe ausgewählter audioTypes wird pro Sample um ein Sample verzögert) und der Spieler nur noch einen Lebenspunkt hat (Audiowiedergabe ausgewählter audioTypes wird pro Sample um ein Sample erhöht).

Autor

Felix Pfreundtner

5.2.2 Klassen-Dokumentation

5.2.2.1 struct AudioControl::playStruct

[playStruct](#) definiert die Struktur eines playEvents

Klassen-Elemente

int	id	id des playStruct
audioType	type	audioType des playStruct
float	volume	Lautstärke des playStruct .
bool	playnext	variable welche angibt ob das playEvent im Moment abgespielt wird
Audio *	audioobject	Zeiger auf das Audio Object des playEvents, welches der audio-Eventgruppe type zugeordnet ist.
int	position	aktuelle Abspielposition des Audiobjekt in Samples

5.2.3 Beschreibung der Konstruktoren und Destruktoren

5.2.3.1 AudioControl::AudioControl ()

Konstruktor instanziiert ein Objekt der Klasse [AudioControl](#).

Wird einmal zum Spielstart von dem game Objekt aufgerufen, welche ein Instanz audioOutput erstellt. Es wird für jede audioEventgruppe type ein audio Objekt erstellt, welches unter anderem die Samples der zugehörigen WAVE Datei beinhaltet.

Autor

Felix Pfreundtner

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 einen PortAudio Stream 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 audioStructs 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 der letzte Audioblock abgespielt wurde und ein neuer Ausgabe Audioblock geniert werden muss.

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 Methode der [AudioControl](#) Klasse.

Die Methode wird immer dann aufgerufen, wenn der PortAudio Stream einen neuen Ausgabeblock benötigt, da der letzte abgespielt wurde. Die Methode ruft die Methode `instancepaCallback` auf, welche nicht statisch ist und auf alle Instanz Variablen und Methoden (des von [Game](#) erzeugten [AudioControl](#) Objektes `audioOutput`) zugreifen kann. Dies ermöglicht einen einfachen Austausch von audiobezogenen Informationen 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 AudioControl::status_filter [private]

status_filter gibt den Filterstatus an.

Wenn kein Audioevent in der audiovents List den Type status_alcohol hat -> enum statusFilter no-> 0. Wenn mindestens ein Audioevent in der audiovents List den Type status_alcohol hat -> enum statusFilter alcohol-> 1. Wenn mindestens ein Audioevent in der audiovents Liste den Type status_lifecritical hat -> enum statusFilter lifecritical-> 2.

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.

```
#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

<i>1.Objekt</i>	
<i>2.Objekt</i>	

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:

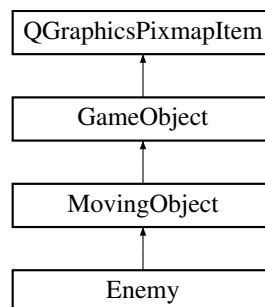
- Wiesn-Run/src/game.cpp

5.6 Enemy Klassenreferenz

Das Gegner Objekt.

```
#include <enemy.h>
```

Klassendiagramm für Enemy:



Öffentliche Methoden

- [Enemy](#) (int posX, int posY, int speedX, [objectType](#) enemy)
Konstruktor für ein Enemy-Objekt Erzeugt einen neuen Gegner, dabei werden Startwerte für die einzelnen Attribute festgelegt, je nachdem, um welchen Gegner es sich handelt.
- [~Enemy](#) ()
Destruktor.
- int [getHealth](#) () const
Gibt Leben des Gegners zurück.
- void [setHealth](#) (int health)
Setzt das Leben des Gegners.
- bool [receiveDamage](#) (int damage)

- Fügt dem Gegner Schaden zu.*

 - int `getInflictedDamage` () const

Gibt den Schaden zurück, den der Gegner zufügt.
- int `getFireCooldown` () const

Gibt fir verbleibende Nachladezeit zurück.
- bool `getDeath` () const

gibt den Todeszustand des Gegners zurück
- void `setDeath` (bool death)

setzt den Todeszustand des Gegners
- int `getDeathCooldown` () const

Gibt die verbleibende Zeit zurück, die der Gegner noch angezeigt werden soll.
- virtual void `update` ()

Hier werden alle framespezifischen Aktualisierungen durchgeführt.
- void `setPosX` (int posX)

Setzt die X-Position des Objekts.
- void `setPosY` (int posY)

Setzt die Y-Position des Objekts.
- int `getSpeedX` () const

Gibt die horizontale Geschwindigkeit zurück.
- int `getSpeedY` () const

Gibt die vertikale Geschwindigkeit zurück.
- void `setSpeedX` (int speedX)

Setzt die horizontale Geschwindigkeit.
- void `setSpeedY` (int speedY)

Setzt die vertikale Geschwindigkeit.
- 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

Gibt die X-Position des Objekts zurück.
- int `getPosY` () const

Gibt die Y-Position des Objekts zurück.
- int `getLength` () const

Gibt die Länge des Objekts zurück.
- int `getHeight` () const

Gibt die Höhe des Objekts zurück.
- `objectType` `getType` () const

Gibt den Objekt-Typ des Objekts zurück.
- void `setAudioID` (int audioID)

Setzt die Audio-ID fest.
- int `getAudioID` () const

Gibt die Audio-ID des Objekts zurück.

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 Ausführliche Beschreibung

Das Gegner Objekt.

Dieses Objekt repräsentiert den Gegner. Das Objekt erbt von MovingObjekt und die wichtigsten Funktionen sind:

- Automatische Aktualisierung
- Schaden erhalten die wichtigsten Attribute sind:
- Leben
- Schaden Die Gegner-Objekte führen alle Bewegungen selbstständig aus nur das Bierkrugwerfen wird von außen geregelt.

Autor

Johann, Simon

5.6.2 Beschreibung der Konstruktoren und Destruktoren

5.6.2.1 `Enemy::Enemy (int posX, int posY, int speedX, objectType enemy)`

Konstruktor für ein Enemy-Objekt Erzeugt einen neuen Gegner, dabei werden Startwerte für die einzelnen Attribute festgelegt, je nachdem, um welchen Gegner es sich handelt.

Mögliche Gegnertypen: BOSS, Tourist, Security Attribute in denen sich die Gegner unterscheiden: Leben, Feuerrate

Parameter

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

<i>enemy</i>	Gegnertyp
--------------	-----------

Autor

Johann, Simon

5.6.3 Dokumentation der Elementfunktionen**5.6.3.1 int Enemy::getHealth () const**

Gibt Leben des Gegners zurück.

Rückgabe

int

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

Setzt das Leben des Gegners.

Parameter

<i>Wert,auf</i>	den das Leben gesetzt werden soll
-----------------	-----------------------------------

5.6.3.3 bool Enemy::receiveDamage (int *damage*)

Fügt dem Gegner Schaden zu.

Und gibt zurück, ob der Gegner danach tot ist.

Parameter

<i>Wert</i>	des Schadens, der zugefügt werden soll
-------------	--

Rückgabe

true, wenn der Gegner tot ist

5.6.3.4 int Enemy::getInflictedDamage () const

Gibt den Schaden zurück, den der Gegner zufügt.

Rückgabe

int

5.6.3.5 int Enemy::getFireCooldown () const

Gibt fir verbleibende Nachladezeit zurück.

Rückgabe

int

5.6.3.6 `bool Enemy::getDeath () const`

gibt den Todeszustand des Gegners zurück

Rückgabe

true, wenn der Gegner tot ist

5.6.3.7 `void Enemy::setDeath (bool death)`

setzt den Todeszustand des Gegners

Parameter

<i>Todeszustand</i>	
---------------------	--

5.6.3.8 `int Enemy::getDeathCooldown () const`

Gibt die verbleibende Zeit zurück, die der Gegner noch angezeigt werden soll.

In Frames

Rückgabe

int

5.6.3.9 `void Enemy::update () [virtual]`

Hier werden alle framespezifischen Aktualisierungen durchgeführt.

Autor

Johann

Wenn der Gegner tot ist, wird die verbleibende Zeit der Anzeige um 1 verringert,
sonst wird jedesmal, wenn die Nachladezeit abgelaufen ist, mit einem Bierkrug geworfen
zuletzt wird die Position des Gegners aktualisiert, damit im Falle des Todes die Gegner nach unten Fallen.
Implementiert [MovingObject](#).

5.6.3.10 `void MovingObject::setPosX (int posX) [inherited]`

Setzt die X-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.6.3.11 `void MovingObject::setPosY (int posY) [inherited]`

Setzt die Y-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.6.3.12 `int MovingObject::getSpeedX () const` [inherited]

Gibt die horizontale Geschwindigkeit zurück.

Rückgabe

int

5.6.3.13 `int MovingObject::getSpeedY () const` [inherited]

Gibt die vertikale Geschwindigkeit zurück.

Rückgabe

int

5.6.3.14 `void MovingObject::setSpeedX (int speedX)` [inherited]

Setzt die horizontale Geschwindigkeit.

Parameter

<i>speedX</i>	horizontale Geschwindigkeit
---------------	-----------------------------

5.6.3.15 `void MovingObject::setSpeedY (int speedY)` [inherited]

Setzt die vertikale Geschwindigkeit.

Parameter

<i>speedY</i>	vertikale Geschwindigkeit
---------------	---------------------------

5.6.3.16 `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

5.6.3.17 `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

5.6.3.18 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, wenn er nur noch ein Leben hat läuft sie doppelt so schnell ab.

Autor

Flo

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

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

5.6.3.20 int GameObject::getPosX () const [inherited]

Gibt die X-Position des Objekts zurück.

Rückgabe

int

5.6.3.21 int GameObject::getPosY () const [inherited]

Gibt die Y-Position des Objekts zurück.

Rückgabe

int

5.6.3.22 int GameObject::getLength () const [inherited]

Gibt die Länge des Objekts zurück.

Rückgabe

int

5.6.3.23 int GameObject::getHeight () const [inherited]

Gibt die Höhe des Objekts zurück.

Rückgabe

int

5.6.3.24 objectType GameObject::getType () const [inherited]

Gibt den Objekt-Typ des Objekts zurück.

Rückgabe

objectType

5.6.3.25 void GameObject::setAudioID (int audioID) [inherited]

Setzt die Audio-ID fest.

Diese wird in der game.cpp benötigt, um objektspezifische Sounds wiederzugeben.

Parameter

<i>audioID</i>	Audio-ID
----------------	----------

5.6.3.26 int GameObject::getAudioID () const [inherited]

Gibt die Audio-ID des Objekts zurück.

Rückgabe

int

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

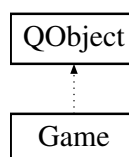
- Wiesn-Run/src/enemy.h
- Wiesn-Run/src/enemy.cpp

5.7 Game Klassenreferenz

Kern-Funktionalität des Spiels.

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

Klassendiagramm für Game:



Öffentliche Methoden

- **Game** (int argc, char *argv[])
Konstruktor: Initialisiert den appPointer.
- **~Game** ()
Destruktor: Gibt verwendeten Heap-Speicher wieder frei.
- int **step** ()
Game-Loop.
- int **start** ()

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

- void `setState` (enum `gameState` newState)
setzt den Spielstatus.

Öffentliche Attribute

- struct `stateStruct` `gameStats`
States des Spiels.
- std::list< struct `collisionStruct` > `collisionsToHandle`
Liste von Kollisionen.

Geschützte Methoden

- void `timerEvent` (QTimerEvent *event)
Wird vom Timer in jedem Intervall aufgerufen.

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)
Startet neues Level.
- void `loadLevelFile` (QString fileSpecifier)
Level-Datei auslesen.
- void `updateHighScore` (std::string mode)
Diese Funktion liest und aktualisiert die Highscore des Spiels.
- void `displayStatistics` ()
füllt das Statistik- und HighscoreMenü.
- void `endGame` ()
Wird beim Beenden des Levels aufgerufen.
- void `appendWorldObjects` (Player *playerPointer)
Diese Funktion fügt der Spielwelt dynamisch Gegner hinzu.
- void `reduceWorldObjects` (Player *playerPointer)
Diese Funktion löscht nicht mehr benötigte Objecte.
- void `evaluateInput` ()
Checkt, welche Tasten für die Spielkontrolle gedrückt sind.
- void `calculateMovement` ()
Geht die worldObjects-Liste durch und aktualisiert bei jedem Object die Position.
- void `detectCollision` (std::list< GameObject * > *objectsToCalculate)
Kollisionsdetektion.
- void `handleCollisions` ()
Kollisionen in der Liste collisionsToHandle werden der Reihe nach aus Sicht des affectedObjects bearbeitet.

- void `updateScore ()`
Aktualisiert die Score des Spielers.
- void `updateAudioevents ()`
Übergibt vom Spiel erzeugte Audioevents an den Output.
- void `renderGraphics (std::list< GameObject * > *objectList, Player *playerPointer)`
Grafik wird ausgegeben.
- void `menuInit ()`
Initialisierung der Menüs.
- void `exitGame ()`
Wird zum Spielende aufgerufen.
- bool `eventFilter (QObject *obj, QEvent *event)`
eventFilter wird aufgerufen, wenn ein neues QEvent auftritt.
- int `getStepIntervall ()`
Gibt stepIntervall zurück.
- void `timeNeeded (string name)`
Misst die Zeit, die zwischen dem letzten Aufruf und dem aktuellen Aufruf vergangen ist und gibt diese in der Konsole mit dem übergebenen String aus.

Private Attribute

- std::list< `GameObject` * > `worldObjects`
In der Welt befindliche Objekte.
- std::list< `GameObject` * > `levelSpawn`
Objekte die zur Laufzeit dynamisch 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 aktiven 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`
Score-Liste.
- struct `scoreStruct` `playerScore`
Scores.
- int `stepIntervall`
Länge eines Steps.
- bool `exitGameevent`
Spiel Beenden gedrückt.
- bool `levelStartevent`
Spiel Starten gedrückt.
- `Player` * `playerObjPointer`
Pointer auf Spieler-Objekt.

- `QGraphicsView * window`
für das Ausgabefenster QGraphicsView
- `QGraphicsScene * levelScene`
QGraphicsScene des Levels.
- `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`
aktueller Spielzustand
- `Menu * aktMenu = menuStart`
aktuell aktives Menü, null, während das Spiel läuft; wird in setState gesetzt
- `Menu * menuStart`
Startmenü
- `Menu * menuCredits`
Credits-Menü
- `Menu * menuLevel`
Levelauswahl-Menü
- `Menu * menuBreak`
Pause-Menü
- `Menu * menuStatistics`
Statistik-Menü
- `Menu * menuName`
Namenseingabe-Menü
- `Menu * menuHighscore`
Highscore-Menü
- `Menu * menuHelp`
Hilfemenü
- `int stepCount = 0`
*stepCount wird mit jedem Step um ein erhöht Auslesen der vergangenen Zeit: stepCount * getStepIntervall()*
- `int audioIDs`
audioIDs wird mit jedem AudioEvent erhöht. Jedes AudioEvent erhält eine feste ID
- `audioCooldownStruct audioCooldown`
Struktur zur Erstellung von Audio-Cooldown-Events.
- `audioDistanceStruct audioDistance`
Struktur zur Erstellung von AudioEvents, gibt die Lautstärke des AudioEvents an.
- `chrono::high_resolution_clock::time_point thisStep`
Zeitpunkt zur Bestimmung der Dauer eines "Steps". Wird in updateAudioevents benutzt um den Cooldown von Audioevents zu verringern.
- `chrono::high_resolution_clock::time_point testStep`
Zeitpunkt zur Bestimmung der Dauer zwischen zwei timeNeeded aufrufen.

5.7.1 Ausführliche Beschreibung

Kern-Funktionalität 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.

Autor

Simon, Johann, Felix, Rupert, Florian

5.7.2 Dokumentation der Elementfunktionen

5.7.2.1 `int Game::step ()`

Game-Loop.

Diese Funktion wird von `timerEvent()` aufgerufen und ist für den kompletten Ablauf des Spiels verantwortlich. Der Ablauf sieht so aus:

- `Input` abfragen
- Läuft das Spiel oder ist ein Menü aktiv?

falls Menü:

- Hintergrundmusik
- Up/Down auswerten
- Enter auswerten: Hier wird im aktuellen Menü die Eigenschaft `menuOnEnter` überprüft, also ob wieder ein Menü folgt. Wenn ja, wird die Eigenschaft `stateOnClick` aus dem Menü Eintrag als neues Menü gesetzt.
- Die Einträge, auf die kein weiteres Menü folgt, werden einzeln ausgewertet, z.B. Level starten/Spiel beenden.
- Namen-Menü: Hier erfolgt die Auswertung der Namenseingabe

falls Spiel:

- auf ESC testen
- Neue Objekte zur Welt hinzufügen
- alte Objekte löschen
- Audioevents updaten
- `Input` auslesen
- Bewegungen berechnen
- Kollisionskontrolle und Bewegungen korrigieren
- Events behandeln (Treffer..)
- Score aktualisieren
- Grafik rendern und ausgeben
- Level zu Ende?
- Spieler tot?

Rückgabe

0 bei fehlerfreiem Beenden

Autor

Rupert, Felix, Johann

Noch zu erledigen Erfolgreich Schriftzug einfügen

Noch zu erledigen GameOver schriftzug einfügen

5.7.2.2 int Game::start ()

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

5.7.2.3 void Game::setState (enum gameState newState)

setzt den Spielstatus.

Hier wird der aktuelle Spielzustand geändert, z.B. von Menü-Anzeige zu laufendem Level.

Parameter

<i>newState</i>	neuer Status aus der enum gameState
-----------------	-------------------------------------

Autor

Rupert

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

Wird vom Timer in jedem Intervall aufgerufen.

Hier wird dann wiederum **step()** aufgerufen. Außerdem wird überprüft ob das Fenster geschlossen wurde und gegebenenfalls **exitGame()** aufgerufen.

Autor

Rupert, Felix

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

Startet neues Level.

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

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

Level-Datei auslesen.

Diese Funktion liest Level-Dateien aus. In der Leveldatei werden Keywords für die anzulegenden 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 (Zwischenebene) enthält posX und posY. Ein PowerUp-Eintrag enthält posX, posY und die jeweiligen Boni.

Parameter

<i>fileSpecifier</i>	String mit Dateinamen der Leveldatei
----------------------	--------------------------------------

Autor

Simon

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

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

5.7.2.8 void Game::displayStatistics () [private]

füllt das Statistik- und HighscoreMenü.

Diese Funktion löscht das Statistik- und Highscore-Menü und füllt es mit aktuellen Werten.

Autor

Rupert

5.7.2.9 void Game::endGame () [private]

Wird beim Beenden des Levels aufgerufen.

Diese Funktion löscht nicht mehr nötige Variablen und Objekte wenn vom Spiel in das Statistikmenü gewechselt wird. Es wird ein zufälliger Name aus einer Liste gewählt, der dann vom Spieler abgeändert werden kann. Dazu wird das Namen-Menü neu geschrieben und dorthin gewechselt.

Autor

: Felix, Johann, Rupert

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

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.

Parameter

<i>playerPointer</i>	
----------------------	--

Autor

Simon

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

Diese Funktion löscht nicht mehr benötigte Objekte.

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

Parameter

<i>playerPointer</i>	
----------------------	--

Autor

Simon, Johann

5.7.2.12 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

es wird entsprechend die Spielergeschwindigkeit geändert, ein Sprung oder Schuss initialisiert und Audioevents erzeugt.

Autor

Rupert

5.7.2.13 void Game::calculateMovement () [private]

Geht die worldObjects-Liste durch und aktualisiert bei jedem Object die Position.

Gegner, bei denen der DeathCooldown abgelaufen ist, werden zum löschen vorgemerkt, Gegner, bei denen der FireCooldown abgelaufen ist, feuern. Wird auch über Debug ausgegeben.

Autor

Rupert, Johann

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

Kollisionsdetektion.

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.

Noch zu erledigen Simon, willst du hier die normalen Kommentare noch einarbeiten und die Funktion genauer erklären? - Rupl

Autor

Simon

5.7.2.15 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

5.7.2.16 void Game::updateScore () [private]

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

5.7.2.17 void Game::updateAudioevents () [private]

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 Ausioevents übergeben. Bei einmaligen Audioevents die Restspielzeit aktualisieren und das Event übergeben

Autor

Johann, Felix

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

Grafik wird ausgegeben.

Noch zu erledigen Flo, willst du hier die normalen Kommentare noch einarbeiten und die Funktion genauer erklären? - Rupl

Parameter

<i>objectList</i>	Liste der Objekte (worldObjects)
<i>playerPointer</i>	Pointer auf den Spieler, wird für Positionsabfrage gebraucht

Autor

Florian

5.7.2.19 void Game::menuInit () [private]

Initialisierung der Menüs.

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

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

Autor

Rupert

5.7.2.20 void Game::exitGame () [private]

Wird zum Spielende aufgerufen.

Diese Funktion wird aufgerufen wenn das Programm beendet werden soll. Hier werden alle Objekte gelöscht und der Speicher wieder freigegeben.

Autor

: Felix, Johann

5.7.2.21 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 *obj
	QEvent *event

Rückgabe

: QObject::eventFilter(obj, event)

Autor

: Felix

5.7.2.22 int Game::getStepIntervall () [private]

Gibt stepIntervall zurück.

Wird zum Auslesen der Zeit gebraucht.

Rückgabe

int Stepintervall in ms

Autor

Rupert

5.7.2.23 void Game::timeNeeded (string name) [private]

Misst die Zeit, die zwischen dem letzten Aufruf und dem aktuellen Aufruf vergangen ist und gibt diese in der Konsole mit dem übergebenen String aus.

Parameter

<i>name</i>	String für die Ausgabe
-------------	------------------------

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

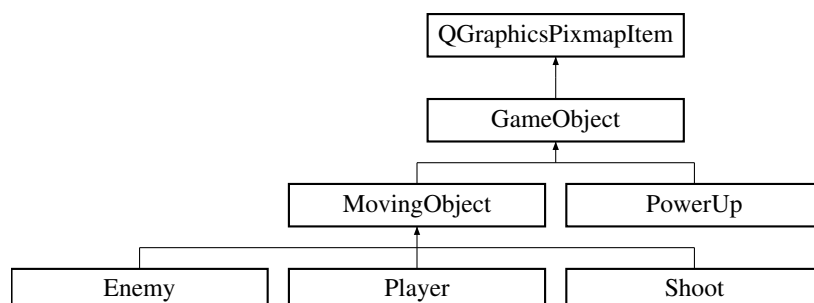
- Wiesn-Run/src/game.h
- Wiesn-Run/src/game.cpp

5.8 GameObject Klassenreferenz

Das Spieler-Objekt.

```
#include <gameobject.h>
```

Klassendiagramm für GameObject:



Öffentliche Methoden

- **GameObject** (int posX, int posY, int length, int height, **objectType** type)
*Konstruktor für ein **GameObject**.*
- **GameObject** (int posX, int posY, **objectType** type)
***GameObject** Konstruktor Je nach Objekt-Typ bekommt hier jedes Objekt Abmessungen und eine Grafik zugewiesen und die "Startposition" wird in Szenenkoordinaten errechnet.*
- virtual **~GameObject** ()
Destruktor.
- int **getPosX** () const
Gibt die X-Position des Objekts zurück.
- int **getPosY** () const
Gibt die Y-Position des Objekts zurück.
- int **getLength** () const
Gibt die Länge des Objekts zurück.
- int **getHeight** () const
Gibt die Höhe des Objekts zurück.
- **objectType** **getType** () const
Gibt den Objekt-Typ des Objekts zurück.
- void **setAudioID** (int audioID)
Setzt die Audio-ID fest.
- int **getAudioID** () const
Gibt die Audio-ID des Objekts zurück.

Geschützte Attribute

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

Private Attribute

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

5.8.1 Ausführliche Beschreibung

Das Spieler-Objekt.

Dieses Objekt repräsentiert das Grundobjekt. Das Objekt beschreibt ein einfaches Rechteck, mit den Attributen:

- X/Y-Position
- Länge/Höhe
- Objekt-Typ
- Audio-ID

Autor

Johann, Flo

5.8.2 Beschreibung der Konstruktoren und Destruktoren

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

Konstruktor für ein [GameObject](#).

Die Attribute Länge, Höhe und Objekt-Typ können nicht mehr geändert werden und X/Y-Position nur durch erbende Klassen.

Parameter

<i>posX</i>	X-Position
<i>posY</i>	Y-Position
<i>length</i>	Länge
<i>height</i>	Breite
<i>type</i>	Objekt-Typ

Autor

Johann

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

[GameObject](#) Konstruktor Je nach Objekt-Typ bekommt hier jedes Objekt Abmessungen und eine Grafik zugewiesen und die "Startposition" wird in Szenenkoordinaten errechnet.

Die Attribute Länge, Höhe und Objekt-Typ können nicht mehr geändert werden und X/Y-Position nur durch erbende Klassen.

Parameter

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

Autor

Johann, Flo

5.8.3 Dokumentation der Elementfunktionen

5.8.3.1 `int GameObject::getPosX () const`

Gibt die X-Position des Objekts zurück.

Rückgabe

int

5.8.3.2 `int GameObject::getPosY () const`

Gibt die Y-Position des Objekts zurück.

Rückgabe

int

5.8.3.3 `int GameObject::getLength () const`

Gibt die Länge des Objekts zurück.

Rückgabe

`int`

5.8.3.4 `int GameObject::getHeight () const`

Gibt die Höhe des Objekts zurück.

Rückgabe

`int`

5.8.3.5 `objectType GameObject::getType () const`

Gibt den Objekt-Typ des Objekts zurück.

Rückgabe

`objectType`

5.8.3.6 `void GameObject::setAudioID (int audioID)`

Setzt die Audio-ID fest.

Diese wird in der game.cpp benötigt, um objektspezifische Sounds wiederzugeben.

Parameter

<i>audioID</i>	Audio-ID
----------------	----------

5.8.3.7 `int GameObject::getAudioID () const`

Gibt die Audio-ID des Objekts zurück.

Rückgabe

`int`

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.
- 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 instanziiert 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 set keyletters zurück, welches alle im Moment gedrückten Buchstaben als Enum beinhaltet.
- [Keyaction](#) [getLastKeyaction](#) ()
Gibt die 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](#) ()
updateKeys berechnet aus allen in keyevents gespeicherten Tastatureingaben die für das Spiel relevanten Kombinationen und speichert diese im QSet keyactions oder set keyletters.

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 Buchstaben als char 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 keyInput dieser Klasse wird zum Programmstart im game Objekt angelegt. Im Objekt der `Game` Klasse wird über die Methode `eventFilter()` ein `QEvent` Filter installiert. Dieser ruft bei neuen Events die Methode `evaluateKeyEvent()` auf, welche prüft ob das Event ein `KeyEvent` war. Ist dies der Fall wird die aktuelle Tastatureingabe als zusätzlicher Enum in dem `QSet` `keyevents` gespeichert oder bei Loslassen der Taste gelöscht. Im Anschluss wird über die Methode `updateKeys()` überprüft ob mit allen aktuell gedrückten Eingaben (gespeichert in `keyevents`) eine Tastaturkombination erfolgt, welche für das Spiel relevant ist. Ist dies der Fall wird die aktuelle Tastatureingabe-Aktion als enum `Keyaction` in dem `QSet` `keyactions` gespeichert. Sind Buchstaben in `keyevents` gespeichert so werden diese im `std::set` `keyletters` als `char` gespeichert. Das `Game` Objekt kann über die Methoden `getKeyactions()`, `getKeyletters()`, `getLastKeyaction()` und `getLastKeyletter()` alle, für das Spiel relevanten, Eingaben aus den Variablen auslesen.

Autor

Felix Pfreundtner

5.9.2 Beschreibung der Konstruktoren und Destruktoren

5.9.2.1 `Input::Input ()`

Konstruktor instanziiert ein Objekt der Klasse `Input`.

Wird einmal zum Spielstart von dem game Objekt aufgerufen und ein Objekt `keyInput` erstellt.

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. Um die relevanten Tastaturkombinationen auszuwerten wird die Methode `updateKeys` aufgerufen.

Parameter

<i>QEvent</i>	*event
---------------	--------

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 Index 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 set keyletters zurück, welches alle im Moment gedrückten Buchstaben als Enum beinhaltet.

Jeder Buchstaben Taste wird ein String Buchstaben zugeordnet, welcher im set keyletters gespeichert ist. Über die Enumeration [Input::Keyletter](#) ist jeder Buchstabe mit dem zugehörigen Index in keyletters verknüpft.

Rückgabe

std::set<char> Instanzvariable keyletters

Autor

Felix Pfreundtner

5.9.3.4 Input::Keyaction Input::getLastKeyaction ()

Gibt die 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 Methode bei der Eingabe des Highscore Namens.

Rückgabe

Enum Keyletter Instanzvariable lastKeyletter

Autor

Felix

5.9.3.6 void Input::updateKeys () [private]

updateKeys berechnet aus allen in keyevents gespeicherten Tastatureingaben die für das Spiel relevanten Kombinationen und speichert diese im QSet keyactions oder set keyletters.

Jede aktuell gedrückte Tastaturkombination ist im QSet keyactions als Integer gespeichert, welche über die enumeration Keyaction adressiert wird. Jeder aktuell gedrückte Buchstabe ist im set keyletters als char gespeichert, welche über die enumeration Keyletter und einen int to char Typecast adressiert wird. Wird durch die Methode evaluatekey-Event ein KeyRelease oder KeyPress Event aufgezeichnet, so nach Aufruf der Methode das QSet keyactions und set keyletters gelöscht. Im Anschluss wird geprüft ob in der neuen Situation in keyevents relevante Tastaturkombinationen vorhanden sind. Sind Tasten oder Tastenkombinationen gedrückt worden, welche für das Spiel relevant sind so wird die zur Aktion gehörige enum Keyaction integer ID im QSet keyactions hinzugefügt. Ist ein Buchstabe in keyevents gespeichert, so wird der zugehörige char im set keyletters gespeichert.

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::string

5.10.4.4 `int Menu::displayInit ()`

Initialisiert das sichtbare Menü.

Muss immer nach anlegen der Menü Eintrys 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:

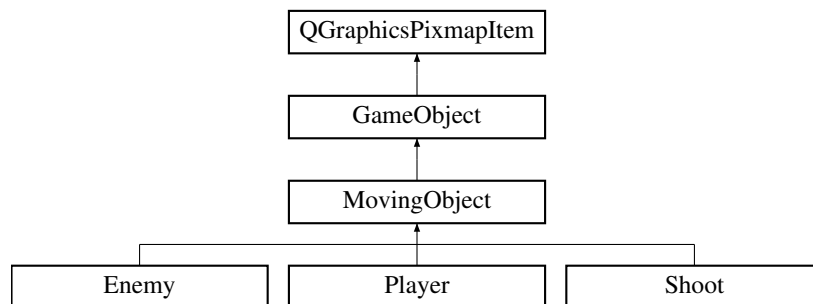
- Wiesn-Run/src/menu.h
- Wiesn-Run/src/menu.cpp

5.11 MovingObject Klassenreferenz

Das Moving-Object.

```
#include <movingobject.h>
```

Klassendiagramm für MovingObject:



Öffentliche Methoden

- **MovingObject** (int posX, int posY, objectType type, int speedX, int speedY)
Konstruktor für ein MovingObject.
- **~MovingObject** ()
Destruktor.
- void **setPosX** (int posX)
Setzt die X-Position des Objekts.
- void **setPosY** (int posY)
Setzt die Y-Position des Objekts.
- int **getSpeedX** () const
Gibt die horizontale Geschwindigkeit zurück.
- int **getSpeedY** () const
Gibt die vertikale Geschwindigkeit zurück.
- void **setSpeedX** (int speedX)
Setzt die horizontale Geschwindigkeit.
- void **setSpeedY** (int speedY)
Setzt die vertikale Geschwindigkeit.
- virtual void **update** ()=0
- void **updateFramesDirection** ()
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).
- 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
Gibt die X-Position des Objekts zurück.
- int **getPosY** () const
Gibt die Y-Position des Objekts zurück.
- int **getLength** () const
Gibt die Länge des Objekts zurück.

- int `getHeight ()` const
Gibt die Höhe des Objekts zurück.
- `objectType getType ()` const
Gibt den Objekt-Typ des Objekts zurück.
- void `setAudioID (int audioID)`
Setzt die Audio-ID fest.
- int `getAudioID ()` const
Gibt die Audio-ID des Objekts zurück.

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 Ausführliche Beschreibung

Das Moving-Object.

Hierbei handelt es sich um eine abstrakte Klasse, die nicht instanziiert werden kann. Die Klasse erbt von [Game-Object](#). Die wichtigsten Funktionen sind:

- Bewegungsausführung
- Graphik

Autor

Simon, Rupert, Johann, Flo

5.11.2 Beschreibung der Konstruktoren und Destruktoren

5.11.2.1 `MovingObject::MovingObject (int posX, int posY, objectType type, int speedX, int speedY)`

Konstruktor für ein [MovingObject](#).

Als abstrakte Klasse kann [MovingObject](#) nicht instanziiert werden. Alle Attribute für diese Klasse müssen von den erbenden Klassen übergeben werden

Parameter

<i>posX</i>	X-Position im Spiel
<i>posY</i>	Y-Position im Spiel
<i>type</i>	Objekt-Typ
<i>speedX</i>	horizontale Geschwindigkeit, >0 entspricht einer Bewegung nach rechts
<i>speedY</i>	vertikale Geschwindigkeit, >0 entspricht einer Bewegung nach oben

Autor

Johann, Simon

5.11.3 Dokumentation der Elementfunktionen

5.11.3.1 void MovingObject::setPosX (int *posX*)

Setzt die X-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.11.3.2 void MovingObject::setPosY (int *posY*)

Setzt die Y-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.11.3.3 int MovingObject::getSpeedX () const

Gibt die horizontale Geschwindigkeit zurück.

Rückgabe

int

5.11.3.4 int MovingObject::getSpeedY () const

Gibt die vertikale Geschwindigkeit zurück.

Rückgabe

int

5.11.3.5 void MovingObject::setSpeedX (int *speedX*)

Setzt die horizontale Geschwindigkeit.

Parameter

<i>speedX</i>	horizontale Geschwindigkeit
---------------	-----------------------------

5.11.3.6 void MovingObject::setSpeedY (int *speedY*)

Setzt die vertikale Geschwindigkeit.

Parameter

<i>speedY</i>	vertikale Geschwindigkeit
---------------	---------------------------

5.11.3.7 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

5.11.3.8 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

5.11.3.9 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, wenn er nur noch ein Leben hat läuft sie doppelt so schnell ab.

Autor

Flo

5.11.3.10 void MovingObject::updatePosition () [protected]

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

5.11.3.11 int GameObject::getPosX () const [inherited]

Gibt die X-Position des Objekts zurück.

Rückgabe

int

5.11.3.12 int GameObject::getPosY () const [inherited]

Gibt die Y-Position des Objekts zurück.

Rückgabe

int

5.11.3.13 int GameObject::getLength () const [inherited]

Gibt die Länge des Objekts zurück.

Rückgabe

int

5.11.3.14 int GameObject::getHeight () const [inherited]

Gibt die Höhe des Objekts zurück.

Rückgabe

int

5.11.3.15 objectType GameObject::getType () const [inherited]

Gibt den Objekt-Typ des Objekts zurück.

Rückgabe

objectType

5.11.3.16 void GameObject::setAudioID (int *audioID*) [inherited]

Setzt die Audio-ID fest.

Diese wird in der game.cpp benötigt, um objektspezifische Sounds wiederzugeben.

Parameter

<i>audioID</i>	Audio-ID
----------------	----------

5.11.3.17 int GameObject::getAudioID () const [inherited]

Gibt die Audio-ID des Objekts zurück.

Rückgabe

int

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

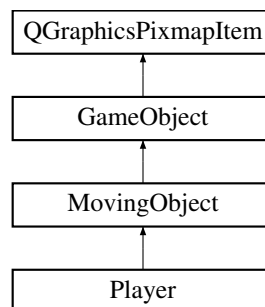
- Wiesn-Run/src/movingobject.h
- Wiesn-Run/src/movingobject.cpp

5.12 Player Klassenreferenz

Das Spieler-Objekt.

```
#include <player.h>
```

Klassendiagramm für Player:



Öffentliche Methoden

- **Player** (int posX, int posY, int speedX)
Konstruktor.
- **~Player** ()
Destruktor.
- int **getHealth** () const
Gibt Leben des Spielers zurück.
- void **setHealth** (int health)
Leben wird erhöht.
- void **increaseHealth** (int health)
- bool **receiveDamage** (int damage)
Fügt dem Spieler Schaden zu, wenn er nicht immun ist, lässt ihn für eine Sekunde immun sein und gibt zurück, ob er gestorben ist.
- int **getAlcoholLevel** () const
- void **increaseAlcoholLevel** (int additionalAlcohol)
Player::increaseAlcoholLevel AlkoholPegel wird verändert.
- void **decreaseAlcoholLevel** (int decreaseLevel)
Verringert den Alkoholpegel des Spielers und den Wert decreaseLevel.
- int **getAmmunatiuon** () const
Gibt die Munition des Spielers zurück.
- void **increaseAmmunation** (int ammunationBonus)

- Erhöht die Munition des Spielers.*

 - void `decreaseAmmunation` ()
- Verringert die Munition des Spielers um 1.*

 - int `getInflictedDamage` () const

Gibt den Schaden zurück, den der Spieler zufügt.
- Setzt die verbleibende Nachladezeit auf den default-Wert, fireRate.*

 - void `setFireCooldown` ()
- gibt die verbleibende Nachladezeit zurück.*

 - int `getFireCooldown` ()
- Gibt zurück, wie lange der Spieler noch Immun ist.*

 - int `getImmunityCooldown` () const
- Zeit für Unverwundbarkeit wird gesetzt, in Frames.*

 - void `setImmunityCooldown` (int remainingTime)
- Beginnt einen Sprung.*

 - void `startJump` ()
- Gibt zurück, ob sich der Spieler in der Luft befindet.*

 - bool `inJump` () const
- Setzt den Wert, dass sich der Spieler auf festem Grund befindet.*

 - void `resetJumpState` ()
- bricht einen Sprung ab und initiiert einen Fall.*

 - void `abortJump` ()
- Gibt zurück, wieviele Gegner der Spieler schon besiegt hat.*

 - int `getEnemiesKilled` ()
- Erhöht die Anzahl der besiegten Gegner um 1.*

 - void `increaseEnemiesKilled` ()
- Gibt den Skalierungsfaktor für die Spielergeschwindigkeit wieder.*

 - int `getSpeedScale` () const
- Hier werden alle framespezifischen Aktualisierungen durchgeführt.*

 - virtual void `update` ()
- Setzt die X-Position des Objekts.*

 - void `setPosX` (int posX)
- Setzt die Y-Position des Objekts.*

 - void `setPosY` (int posY)
- Gibt die horizontale Geschwindigkeit zurück.*

 - int `getSpeedX` () const
- Gibt die vertikale Geschwindigkeit zurück.*

 - int `getSpeedY` () const
- Setzt die horizontale Geschwindigkeit.*

 - void `setSpeedX` (int speedX)
- Setzt die vertikale Geschwindigkeit.*

 - void `setSpeedY` (int speedY)
- 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 `updateFramesDirection` ()
- 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 `flipHorizontal` ()
- 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.*

 - void `swapImage` ()

- int `getPosX ()` const
Gibt die X-Position des Objekts zurück.
- int `getPosY ()` const
Gibt die Y-Position des Objekts zurück.
- int `getLength ()` const
Gibt die Länge des Objekts zurück.
- int `getHeight ()` const
Gibt die Höhe des Objekts zurück.
- `objectType getType ()` const
Gibt den Objekt-Typ des Objekts zurück.
- void `setAudioID (int audioID)`
Setzt die Audio-ID fest.
- int `getAudioID ()` const
Gibt die Audio-ID des Objekts zurück.

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 `ammunition`
- int `inflictedDamage`
- int `fireCooldown`
- int `fireRate`
- int `immunityCooldown`
- bool `jumpState`
- int `jumpCooldown`
- int `speedScale`
- int `alcoholDamageCooldown`
- int `enemiesKilled`

5.12.1 Ausführliche Beschreibung

Das Spieler-Objekt.

Dieses Objekt repräsentiert den Spieler. Das Objekt erbt von `MovingObject` und die wichtigsten Funktionen sind:

- Automatische Aktualisierung
- Schaden erhalten
- Springen des weiteren sind die wichtigsten Attribute:
- Leben

- Alkoholpegel
- Munition
- Schaden Die Bewegungen des Spielers über die Eingabe erfolgt in der `step()`-Methode der `game.cpp`. Es werden hierbei nur die Attribute `speedx/y` gesetzt.

Autor

Johann, Simon

5.12.2 Beschreibung der Konstruktoren und Destruktoren

5.12.2.1 `Player::Player (int posX, int posY, int speedX)`

Konstruktor.

Erzeugt einen neuen Spieler. Dabei werden Startwerte für alle Attribute festgelegt.

Parameter

<i>posX</i>	X-Position im Level
<i>posY</i>	Y-Position im Level
<i>speedX</i>	HorizontalGeschwindigkeit

Autor

Johann, Simon

5.12.3 Dokumentation der Elementfunktionen

5.12.3.1 `int Player::getHealth () const`

Gibt Leben des Spielers zurück.

Rückgabe

`int health`

5.12.3.2 `void Player::setHealth (int health)`

Leben wird erhöht.

Parameter

<i>health</i>	Wert, um den das Leben erhöht wird
---------------	------------------------------------

5.12.3.3 `bool Player::receiveDamage (int damage)`

Fügt dem Spieler Schaden zu, wenn er nicht immun ist, lässt ihn für eine Sekunde immun sein und gibt zurück, ob er gestorben ist.

Parameter

<i>Schaden</i>	der dem Spieler zugefügt werden soll
----------------	--------------------------------------

Rückgabe

true, wenn der Spieler gestorben ist

5.12.3.4 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
--------------------------	-------------------------

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

Verringert den Alkoholpegel des Spielers und den Wert decreaseLevel.

Falls der Wert unter 0 fällt wird er auf 0 gesetzt

Parameter

<i>Wert</i>	um den der Alkoholpegel verringert werden soll
-------------	--

5.12.3.6 int Player::getAmmunition () const

Gibt die Munition des Spielers zurück.

Rückgabe

int

5.12.3.7 void Player::increaseAmmunition (int *ammunitionBonus*)

Erhöht die Munition des Spielers.

Parameter

<i>Der</i>	Wert um den die Munition erhöht werden soll
------------	---

5.12.3.8 int Player::getInflictedDamage () const

Gibt den Schaden zurück, den der Spieler zufügt.

Rückgabe

int

5.12.3.9 int Player::getFireCooldown ()

gibt die verbleibende Nachladezeit zurück.

Rückgabe

int

5.12.3.10 int Player::getImmunityCooldown () const

Gibt zurück, wie lange der Spieler noch Immun ist.

Rückgabe

int

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

Zeit für Unverwundbarkeit wird gesetzt, in Frames.

Parameter

<i>Anzahl</i>	der Frames, für die der Spieler unverwundbar sein soll
---------------	--

5.12.3.12 void Player::startJump ()

Beginnt einen Sprung.

Nur wenn der Spieler sich nicht in der Luft befindet

5.12.3.13 bool Player::inJump () const

Gibt zurück, ob sich der Spieler in der Luft befindet.

Rückgabe

true, falls der Spieler in der Luft ist

5.12.3.14 int Player::getEnemiesKilled ()

Gibt zurück, wieviele Gegner der Spieler schon besiegt hat.

Rückgabe

int

5.12.3.15 int Player::getSpeedScale () const

Gibt den Skalierungsfaktor für die Spielergeschwindigkeit wieder.

Rückgabe

int

5.12.3.16 void Player::update () [virtual]

Hier werden alle framespezifischen Aktualisierungen durchgeführt.

Autor

Johann

Bewegung ausführen

Sprungverlauf:

befindet sich der Spieler in der Luft und hat noch verbleibende Sprungdauer, so wird die verbleibende Sprungdauer um eins verringert, sonst wird ein Fall initiiert

Falls sich der Spieler über der Nullebene befindet, entspricht der Spieler ist in der Luft, wird jumpState auf true gesetzt, um einen Sprung zu verhindern

Alkoholpegel wird über die Zeit abgebaut

Immunität wird über die Zeit abgebaut

restliche Nachladezeit wird verkürzt

Der Skalierungsfaktor für die Spielergeschwindigkeit wird aktualisiert: Spieler kann sich doppelt so schnell bewegen, wenn er nur noch ein Leben hat.

Hat der Spieler zu viel Alkohol im Blut, so verliert er alle 4 Sekunden ein Leben.

Implementiert [MovingObject](#).

5.12.3.17 void MovingObject::setPosX (int posX) [inherited]

Setzt die X-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.12.3.18 void MovingObject::setPosY (int posY) [inherited]

Setzt die Y-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.12.3.19 int MovingObject::getSpeedX () const [inherited]

Gibt die horizontale Geschwindigkeit zurück.

Rückgabe

int

5.12.3.20 int MovingObject::getSpeedY () const [inherited]

Gibt die vertikale Geschwindigkeit zurück.

Rückgabe

int

5.12.3.21 void MovingObject::setSpeedX (int *speedX*) [inherited]

Setzt die horizontale Geschwindigkeit.

Parameter

<i>speedX</i>	horizontale Geschwindigkeit
---------------	-----------------------------

5.12.3.22 void MovingObject::setSpeedY (int *speedY*) [inherited]

Setzt die vertikale Geschwindigkeit.

Parameter

<i>speedY</i>	vertikale Geschwindigkeit
---------------	---------------------------

5.12.3.23 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

5.12.3.24 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

5.12.3.25 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, wenn er nur noch ein Leben hat läuft sie doppelt so schnell ab.

Autor

Flo

5.12.3.26 `void MovingObject::updatePosition () [protected],[inherited]`

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

5.12.3.27 `int GameObject::getPosX () const [inherited]`

Gibt die X-Position des Objekts zurück.

Rückgabe

int

5.12.3.28 `int GameObject::getPosY () const [inherited]`

Gibt die Y-Position des Objekts zurück.

Rückgabe

int

5.12.3.29 `int GameObject::getLength () const [inherited]`

Gibt die Länge des Objekts zurück.

Rückgabe

int

5.12.3.30 `int GameObject::getHeight () const [inherited]`

Gibt die Höhe des Objekts zurück.

Rückgabe

int

5.12.3.31 `objectType GameObject::getType () const [inherited]`

Gibt den Objekt-Typ des Objekts zurück.

Rückgabe

objectType

5.12.3.32 `void GameObject::setAudioID (int audioID) [inherited]`

Setzt die Audio-ID fest.

Diese wird in der game.cpp benötigt, um objektspezifische Sounds wiederzugeben.

Parameter

<i>audioID</i>	Audio-ID
----------------	----------

5.12.3.33 int GameObject::getAudioID () const [inherited]

Gibt die Audio-ID des Objekts zurück.

Rückgabe

int

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

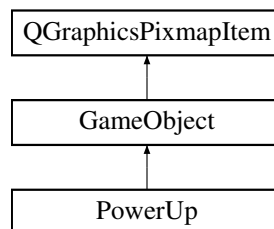
- Wiesn-Run/src/player.h
- Wiesn-Run/src/player.cpp

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 ammurationBonus, int immunity-CooldownBonus, **powerUpType** type)
Konstruktor.
- **~PowerUp** ()
Destruktor.
- int **getHealthBonus** () const
Get-Methoden für die Objekteigenschaften.
- int **getAlcoholLevelBonus** () const
Gibt den Bonus auf Alcohollevel zurück.
- int **getAmmurationBonus** () const
Gibt den Bonus auf Munnition zurück.
- int **getImmunityCooldownBonus** () const
Gibt den Bonus auf Immunität zurück.
- **powerUpType** **getPowerUPType** () const
PowerUp::getPowerUPType.
- int **getPosX** () const
Gibt die X-Position des Objekts zurück.

- `int getPosY () const`
Gibt die Y-Position des Objekts zurück.
- `int getLength () const`
Gibt die Länge des Objekts zurück.
- `int getHeight () const`
Gibt die Höhe des Objekts zurück.
- `objectType getType () const`
Gibt den Objekt-Typ des Objekts zurück.
- `void setAudioID (int audioID)`
Setzt die Audio-ID fest.
- `int getAudioID () const`
Gibt die Audio-ID des Objekts zurück.

Geschützte Attribute

- `int posX`
- `int posY`

Private Attribute

- `int healthBonus`
- `int alcoholLevelBonus`
- `int ammurationBonus`
- `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 ammurationBonus, 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>ammunation-Bonus</i>	
<i>immunity-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 Munnition 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

5.13.3.6 **int** GameObject::getPosX () const [inherited]

Gibt die X-Position des Objekts zurück.

Rückgabe

int

5.13.3.7 **int** GameObject::getPosY () const [inherited]

Gibt die Y-Position des Objekts zurück.

Rückgabe

int

5.13.3.8 **int** GameObject::getLength () const [inherited]

Gibt die Länge des Objekts zurück.

Rückgabe

int

5.13.3.9 **int** GameObject::getHeight () const [inherited]

Gibt die Höhe des Objekts zurück.

Rückgabe

int

5.13.3.10 **objectType** GameObject::getType () const [inherited]

Gibt den Objekt-Typ des Objekts zurück.

Rückgabe

objectType

5.13.3.11 void GameObject::setAudioID (int *audioID*) [inherited]

Setzt die Audio-ID fest.

Diese wird in der game.cpp benötigt, um objektspezifische Sounds wiederzugeben.

Parameter

<i>audioID</i>	Audio-ID
----------------	----------

5.13.3.12 `int GameObject::getAudioID () const` [inherited]

Gibt die Audio-ID des Objekts zurück.

Rückgabe

int

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesen-Run/src/powerup.h
- Wiesen-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

<i>x</i>	: 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

x	: 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, int stepCount)
[RenderGUI::setValues](#) Aktualisierung aller angezeigten Wert, Gesundheits- und Pegelbalken sind immer auf die maximal möglichen Werte normiert.

Private Attribute

- QGraphicsPixmapItem **showHealth**
- QGraphicsRectItem **showHealthBar** [2]
- QGraphicsPixmapItem **showScore**
- QGraphicsTextItem **showScoreValue**
- QGraphicsPixmapItem **showAmmo**
- QGraphicsTextItem **showAmmoValue**
- QGraphicsPixmapItem **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*, int *stepCount*)

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

Wird der maximale Alkoholwert überschritten blinkt der Balken Rot da der Spieler Schaden bekommt.

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:

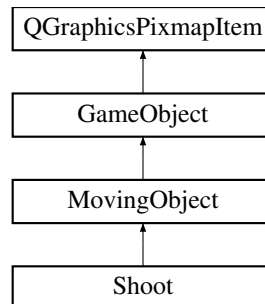
- Wiesn-Run/src/renderGUI.h
- Wiesn-Run/src/renderGUI.cpp

5.16 Shoot Klassenreferenz

Das Schuss Objekt.

```
#include <shoot.h>
```

Klassendiagramm für Shoot:



Öffentliche Methoden

- **Shoot** (int posX, int posY, int direction, objectType origin)
Konstruktor für einen Schuss(Bierkrug).
- **~Shoot** ()
Destruktor.
- int **getInflictedDamage** () const
Gibt den Schaden zurück, den der Bierkrug zufügt.
- objectType **getOrigin** ()
Gibt den Ursprung des Bierkrugs zurück.
- bool **getHarming** () const
Gibt zurück, ob der Bierkrug noch Schaden zufügt.
- void **setToDelete** ()
Setzt den Wert, damit der Schuss keinen Schaden mehr zufügt.
- virtual void **update** ()
Die Funktion aktualisiert die Position des Bierkruges.
- void **setPosX** (int posX)
Setzt die X-Position des Objekts.
- void **setPosY** (int posY)
Setzt die Y-Position des Objekts.
- int **getSpeedX** () const
Gibt die horizontale Geschwindigkeit zurück.
- int **getSpeedY** () const
Gibt die vertikale Geschwindigkeit zurück.
- void **setSpeedX** (int speedX)
Setzt die horizontale Geschwindigkeit.
- void **setSpeedY** (int speedY)
Setzt die vertikale Geschwindigkeit.
- 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
Gibt die X-Position des Objekts zurück.
- int `getPosY ()` const
Gibt die Y-Position des Objekts zurück.
- int `getLength ()` const
Gibt die Länge des Objekts zurück.
- int `getHeight ()` const
Gibt die Höhe des Objekts zurück.
- `objectType getType ()` const
Gibt den Objekt-Typ des Objekts zurück.
- void `setAudioID (int audioID)`
Setzt die Audio-ID fest.
- int `getAudioID ()` const
Gibt die Audio-ID des Objekts zurück.

Geschützte Methoden

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

Geschützte Attribute

- int `posX`
- int `posY`

Private Attribute

- int `inflictedDamage`
- `objectType` `origin`
- bool `harming`

5.16.1 Ausführliche Beschreibung

Das Schuss Objekt.

Dieses Objekt repräsentiert einen geworfenen Bierkrug. Das Objekt erbt von MovingObject und die wichtigsten Funktionen sind:

- Automatische Aktualisierung die wichtigsten Attribute sind:
- Ursprung
- Schaden Die Bierkrüge führen alle Bewegungen selbständig aus

Autor

Johann, Simon

5.16.2 Beschreibung der Konstruktoren und Destruktoren

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

Konstruktor für einen Schuss(Bierkrug).

Erzeugt einen fliegenden Bierkrug, dabei werden alle Werte gesetzt. Diese können später nur noch ausgelesen und nicht mehr geändert werden. Es gilt, dass ein Bierkrug dreimal so schnell fliegt wie sich der Spieler bewegen kann. Bei der Erzeugung eines Schusses nur die Richtung entscheidend ist. >0 bedeutet nach Rechts

Parameter

<i>posX</i>	X-Position im Level
<i>posY</i>	Y-Position im Level
<i>direction</i>	Richtung, in die der Bierkrug fliegen soll
<i>origin</i>	Schuss-Erzeuger

Autor

Johann, Simon

5.16.3 Dokumentation der Elementfunktionen

5.16.3.1 int Shoot::getInflictedDamage () const

Gibt den Schaden zurück, den der Bierkrug zufügt.

Rückgabe

int

5.16.3.2 objectType Shoot::getOrigin ()

Gibt den Ursprung des Bierkrugs zurück.

Rückgabe

objectType

5.16.3.3 bool Shoot::getHarming () const

Gibt zurück, ob der Bierkrug noch Schaden zufügt.

Rückgabe

true, wenn der Bierkrug noch nichts getroffen hat und Schaden zufügt

5.16.3.4 void Shoot::setToDelete ()

Setzt den Wert, damit der Schuss keinen Schaden mehr zufügt.

Dies ist wichtig, damit man nicht zwei Gegner mit einem Bierkrug besiegen kann.

5.16.3.5 void MovingObject::setPosX (int *posX*) [inherited]

Setzt die X-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.16.3.6 void MovingObject::setPosY (int *posY*) [inherited]

Setzt die Y-Position des Objekts.

Parameter

<i>Position</i>	
-----------------	--

5.16.3.7 int MovingObject::getSpeedX () const [inherited]

Gibt die horizontale Geschwindigkeit zurück.

Rückgabe

int

5.16.3.8 int MovingObject::getSpeedY () const [inherited]

Gibt die vertikale Geschwindigkeit zurück.

Rückgabe

int

5.16.3.9 void MovingObject::setSpeedX (int *speedX*) [inherited]

Setzt die horizontale Geschwindigkeit.

Parameter

<i>speedX</i>	horizontale Geschwindigkeit
---------------	-----------------------------

5.16.3.10 void MovingObject::setSpeedY (int *speedY*) [inherited]

Setzt die vertikale Geschwindigkeit.

Parameter

<i>speedY</i>	vertikale Geschwindigkeit
---------------	---------------------------

5.16.3.11 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 ($> || < 0$) und mit dem aktuellen speedX-Wert ein Richtungswechsel feststellen

Autor

Flo

5.16.3.12 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

5.16.3.13 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, wenn er nur noch ein Leben hat läuft sie doppelt so schnell ab.

Autor

Flo

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

überschreibt die X und Y Position gemäß SpeedXY.

Autor

Rupert

5.16.3.15 int GameObject::getPosX () const [inherited]

Gibt die X-Position des Objekts zurück.

Rückgabe

int

5.16.3.16 int GameObject::getPosY () const [inherited]

Gibt die Y-Position des Objekts zurück.

Rückgabe

int

5.16.3.17 `int GameObject::getLength () const` [inherited]

Gibt die Länge des Objekts zurück.

Rückgabe

`int`

5.16.3.18 `int GameObject::getHeight () const` [inherited]

Gibt die Höhe des Objekts zurück.

Rückgabe

`int`

5.16.3.19 `objectType GameObject::getType () const` [inherited]

Gibt den Objekt-Typ des Objekts zurück.

Rückgabe

`objectType`

5.16.3.20 `void GameObject::setAudioID (int audioID)` [inherited]

Setzt die Audio-ID fest.

Diese wird in der game.cpp benötigt, um objektspezifische Sounds wiederzugeben.

Parameter

<i>audioID</i>	Audio-ID
----------------	----------

5.16.3.21 `int GameObject::getAudioID () const` [inherited]

Gibt die Audio-ID des Objekts zurück.

Rückgabe

`int`

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- Wiesen-Run/src/shoot.h
- Wiesen-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](#)
Typdef Struct mit Konstanten für den Audiocooldown jedes Audiotypes. [Mehr ...](#)
- struct [audioDistanceStruct](#)
Typdef Struct mit Konstanten für die Distance jedes Audiotypes. [Mehr ...](#)
- struct [audioStruct](#)
Struktur für einzelne [Audio](#) Events. [Mehr ...](#)
- struct [audioCooldownstruct](#)
Struktur für audioevents mit ihrer Abspielzeit als Cooldown. [Mehr ...](#)
- struct [stateStruct](#)
Struktur für die AudioLevels und Spielzustand. [Mehr ...](#)

Aufzählungen

- enum [gameState](#) {
gamelsRunning, gameMenuStart, gameMenuCredits, gameMenuLevel,
gameMenuBreak, gameMenuStatistcs, 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, welcher eine audioEventgruppe definiert und in audioControl angibt welches WAVE File zum audioType abgespielt werden soll.

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

Enum für die Powerup Typen.

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**)
*Geschwindigkeitskonstanten maxSpeed: Grundgeschwindigkeit, von dieser leiten sich alle Geschwindigkeiten des Spiels ab
playerSpeed: Grundgeschwindigkeit mit der sich der Spieler horizontal bewegt
maxSpeedY: Fallgeschwindigkeit/Sprunggeschwindigkeit.*
- const int **playerSpeed** = **maxSpeed** + 1
- const int **maxSpeedY** = 3 * **maxSpeed** / 2
- 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

Typdef Struct mit Konstanten für den Audiocooldown jedes Audiotypes.

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

Autor

Felix, Johann

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

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 [Wertebereich 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 dem [Game](#) Objekt in jedem Step über updateAudioevents() neu bestimmt werden (2D [Audio](#)).

Autor

Felix, Johann

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

Struktur für einzelne [Audio](#) Events.

Das [AudioControl](#) Objekt `audioOutput` arbeitet Events von dieser Struktur ab. Jedes `audioEvent` hat eine eindeutige `int id`, einen `enum` Audiogruppen `type` und eine `float distance` Information und ordnet somit jedem Objekt einen Sound zu, wobei sich die Distanzinformation des Sounds bei Veränderung der relativen Position Spieler - Audio-Event ändert. Ein Distanzwert beträgt dabei minimal 0 und maximal 1 (größte Entfernung x-Achse im Gamefenster). Die Standarddistanzwerte sind in `typedef struct audioDistance` für jeden Audiogruppen `type` definiert.

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

Ist ein Event mit zu erfolgreicher Audioausgabe vorhanden wird ein [audioStruct](#) mit neuer `id`, Audiogruppen `type` und aktueller Distanz des Audio-Events vom Spieler zum Event erstellt. Dieses `Audiostruct` wird an die Liste `audioevents` mit allen im Step stattfinden `audioStructs` angehängt. Ist ein Event nachwievor aktiv in der Szene wird das Struct im nächsten Step wieder an die Liste `audioevents` angehängt und die [audioStruct](#) `id` konstant gehalten. Ist ein Event nicht mehr in der Szene zu sehen, so muss kein [audioStruct](#) übergeben werden. Die [audioStruct](#) `id` des Events wird im weiteren Spielverlauf nicht mehr verwendet.

Befindet sich z.B. ein Bier mit `id = ...` in der Szene, so ist der `type = scene_beer` (zugehörige WAVE Datei siehe [Audio](#) Ordner). In jedem Step muss in der Audio-Struktur die `distance` des Biers zum Spieler aktualisiert werden und an die Liste `audioevents` angehängt werden. Verschwindet des Bierobjekt so wird das [audioStruct](#) nicht mehr übergeben

und die id nicht mehr verwendet. Gibt es mehrere Bierobjekte so wird das Struct mit Gruppen type=scene_beer mit verschiedenen ids an die Liste angehängt.

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

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

Autor

Felix Pfreundtner

Klassen-Elemente

int	id	
audioType	type	
float	distance	

6.1.2.5 struct audioCooldownstruct

Struktur für audioevents mit ihrer Abspielzeit als Cooldown.

Autor

Johann, Felix

Klassen-Elemente

struct audioStruct	audioEvent	
duration< int, milli >	cooldown	

6.1.2.6 struct stateStruct

Struktur für die AudioLevels und Spielzustand.

Autor

Johann

Klassen-Elemente

bool	gameOver	
int	actLevel	

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 audioType

Enum für den audioType, welcher eine audioEventgruppe definiert und in audioControl angibt welches WAVE File zum audioType 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 Spiel 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 `world-Objects` verlegt werden.

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

Index

- ~Audio
 - Audio, [10](#)
- ~AudioControl
 - AudioControl, [15](#)
- ~Input
 - Input, [43](#)
- ~PowerUp
 - PowerUp, [67](#)
- addEntry
 - Menu, [49](#)
- affectedObject
 - collisionStruct, [19](#)
- appendWorldObjects
 - Game, [34](#)
- Audio, [9](#)
 - ~Audio, [10](#)
 - Audio, [10](#)
 - getSample, [10](#)
 - getSampLenumber, [11](#)
 - getSource, [10](#)
 - normalize, [11](#)
 - readSamples, [11](#)
 - to16bitSample, [11](#)
- AudioControl::playStruct, [14](#)
- audioCooldownStruct, [83](#)
- audioCooldownstruct, [86](#)
- audioDistanceStruct, [84](#)
- audioStruct, [85](#)
- AudioControl, [12](#)
 - ~AudioControl, [15](#)
 - AudioControl, [14](#)
 - AudioControl, [14](#)
 - instancepaCallback, [16](#)
 - mtx, [18](#)
 - playInitialize, [15](#)
 - playTerminate, [16](#)
 - staticpaCallback, [16](#)
 - status_filter, [18](#)
 - updatePlayevents, [16](#)
- audioType
 - definitions.h, [87](#)
- background_highscore
 - definitions.h, [87](#)
- background_level1
 - definitions.h, [88](#)
- background_level2
 - definitions.h, [88](#)
- background_level3
 - definitions.h, [88](#)
- background_levelfinished
 - definitions.h, [88](#)
- background_menu
 - definitions.h, [87](#)
- background_startgame
 - definitions.h, [88](#)
- calculateMovement
 - Game, [35](#)
- causingObject
 - collisionStruct, [19](#)
- changeSelection
 - Menu, [50](#)
- clear
 - Menu, [48](#)
- collisionDirection
 - definitions.h, [87](#)
- collisionStruct, [18](#)
 - affectedObject, [19](#)
 - causingObject, [19](#)
 - direction, [19](#)
- compareGameObjects, [19](#)
- compareScores, [20](#)
- decreaseAlcoholLevel
 - Player, [60](#)
- definitions.h
 - background_highscore, [87](#)
 - background_level1, [88](#)
 - background_level2, [88](#)
 - background_level3, [88](#)
 - background_levelfinished, [88](#)
 - background_menu, [87](#)
 - background_startgame, [88](#)
 - player_jump, [87](#)
 - player_walk, [87](#)
 - powerup_beer, [87](#)
 - powerup_food, [87](#)
 - scene_collision_enemy, [87](#)
 - scene_collision_flyingbeer, [87](#)
 - scene_collision_obstacle, [87](#)
 - scene_collision_player, [87](#)
 - scene_enemy_boss, [87](#)
 - scene_enemy_security, [87](#)
 - scene_enemy_tourist, [87](#)
 - scene_flyingbeer, [87](#)
 - status_alcohol, [87](#)
 - status_dead, [87](#)
 - status_life, [87](#)

- status_lifecritical, 87
- definitions.h
 - audioType, 87
 - collisionDirection, 87
 - gameState, 86
 - objectType, 86
 - spawnDistance, 88
- detectCollision
 - Game, 35
- direction
 - collisionStruct, 19
- displayInit
 - Menu, 49
- displayStatistics
 - Game, 33
- displayUpdate
 - Menu, 49
- endGame
 - Game, 33
- Enemy, 20
 - Enemy, 22
 - flipHorizontal, 25
 - getAudioID, 27
 - getDeath, 23
 - getDeathCooldown, 24
 - getFireCooldown, 23
 - getHealth, 23
 - getHeight, 26
 - getInflictedDamage, 23
 - getLength, 26
 - getPosX, 26
 - getPosY, 26
 - getSpeedX, 25
 - getSpeedY, 25
 - getType, 26
 - receiveDamage, 23
 - setAudioID, 27
 - setDeath, 24
 - setHealth, 23
 - setPosX, 24
 - setPosY, 24
 - setSpeedX, 25
 - setSpeedY, 25
 - swapImage, 26
 - update, 24
 - updateFramesDirection, 25
 - updatePosition, 26
- evaluateInput
 - Game, 34
- evaluateKeyEvent
 - Input, 43
- eventFilter
 - Game, 36
- exitGame
 - Game, 36
- flipHorizontal
 - Enemy, 25
- MovingObject, 54
- Player, 63
- Shoot, 78
- Game, 27
 - appendWorldObjects, 34
 - calculateMovement, 35
 - detectCollision, 35
 - displayStatistics, 33
 - endGame, 33
 - evaluateInput, 34
 - eventFilter, 36
 - exitGame, 36
 - getStepIntervall, 38
 - handleCollisions, 35
 - loadLevelFile, 33
 - menuInit, 36
 - reduceWorldObjects, 34
 - renderGraphics, 36
 - setState, 32
 - start, 32
 - startNewGame, 33
 - step, 31
 - timeNeeded, 38
 - timerEvent, 32
 - updateAudioevents, 35
 - updateHighScore, 33
 - updateScore, 35
- GameObject, 38
 - GameObject, 40
 - GameObject, 40
 - getAudioID, 41
 - getHeight, 41
 - getLength, 40
 - getPosX, 40
 - getPosY, 40
 - getType, 41
 - setAudioID, 41
- gameState
 - definitions.h, 86
- getAlcoholLevelBonus
 - PowerUp, 67
- getAmmunationBonus
 - PowerUp, 67
- getAmmunatiuon
 - Player, 60
- getAudioID
 - Enemy, 27
 - GameObject, 41
 - MovingObject, 55
 - Player, 65
 - PowerUp, 70
 - Shoot, 79
- getDeath
 - Enemy, 23
- getDeathCooldown
 - Enemy, 24
- getEnemiesKilled
 - Player, 61

- getEntry
 - Menu, [50](#)
- getFireCooldown
 - Enemy, [23](#)
 - Player, [60](#)
- getHarming
 - Shoot, [76](#)
- getHealth
 - Enemy, [23](#)
 - Player, [59](#)
- getHealthBonus
 - PowerUp, [67](#)
- getHeight
 - Enemy, [26](#)
 - GameObject, [41](#)
 - MovingObject, [55](#)
 - Player, [64](#)
 - PowerUp, [68](#)
 - Shoot, [79](#)
- getImmunityCooldown
 - Player, [61](#)
- getImmunityCooldownBonus
 - PowerUp, [67](#)
- getInflictedDamage
 - Enemy, [23](#)
 - Player, [60](#)
 - Shoot, [76](#)
- getKeyactions
 - Input, [45](#)
- getKeyletters
 - Input, [45](#)
- getLastKeyaction
 - Input, [45](#)
- getLastKeyletter
 - Input, [45](#)
- getLength
 - Enemy, [26](#)
 - GameObject, [40](#)
 - MovingObject, [55](#)
 - Player, [64](#)
 - PowerUp, [68](#)
 - Shoot, [78](#)
- getOrigin
 - Shoot, [76](#)
- getPosX
 - Enemy, [26](#)
 - GameObject, [40](#)
 - MovingObject, [54](#)
 - Player, [64](#)
 - PowerUp, [68](#)
 - Shoot, [78](#)
- getPosY
 - Enemy, [26](#)
 - GameObject, [40](#)
 - MovingObject, [55](#)
 - Player, [64](#)
 - PowerUp, [68](#)
 - Shoot, [78](#)
- getPowerUPType
 - PowerUp, [67](#)
- getSample
 - Audio, [10](#)
- getSamplenumber
 - Audio, [11](#)
- getSelection
 - Menu, [50](#)
- getSource
 - Audio, [10](#)
- getSpeedScale
 - Player, [61](#)
- getSpeedX
 - Enemy, [25](#)
 - MovingObject, [53](#)
 - Player, [62](#)
 - Shoot, [77](#)
- getSpeedY
 - Enemy, [25](#)
 - MovingObject, [53](#)
 - Player, [62](#)
 - Shoot, [77](#)
- getStepIntervall
 - Game, [38](#)
- getTitle
 - Menu, [49](#)
- getType
 - Enemy, [26](#)
 - GameObject, [41](#)
 - Menu, [49](#)
 - MovingObject, [55](#)
 - Player, [64](#)
 - PowerUp, [68](#)
 - Shoot, [79](#)
- handleCollisions
 - Game, [35](#)
- inJump
 - Player, [61](#)
- increaseAlcoholLevel
 - Player, [60](#)
- increaseAmmunation
 - Player, [60](#)
- Input, [41](#)
 - ~Input, [43](#)
 - evaluateKeyEvent, [43](#)
 - getKeyactions, [45](#)
 - getKeyletters, [45](#)
 - getLastKeyaction, [45](#)
 - getLastKeyletter, [45](#)
 - Input, [43](#)
 - updateKeys, [46](#)
- instancepaCallback
 - AudioControl, [16](#)
- loadLevelFile
 - Game, [33](#)

- Menu, 46
 - addEntry, 49
 - changeSelection, 50
 - clear, 48
 - displayInit, 49
 - displayUpdate, 49
 - getEntry, 50
 - getSelection, 50
 - getTitle, 49
 - getType, 49
 - Menu, 48
 - selectFirstEntry, 50
- Menu::menuEntry, 48
- menuInit
 - Game, 36
- MovingObject, 51
 - flipHorizontal, 54
 - getAudioID, 55
 - getHeight, 55
 - getLength, 55
 - getPosX, 54
 - getPosY, 55
 - getSpeedX, 53
 - getSpeedY, 53
 - getType, 55
 - MovingObject, 52
 - MovingObject, 52
 - setAudioID, 55
 - setPosX, 53
 - setPosY, 53
 - setSpeedX, 53
 - setSpeedY, 54
 - swapImage, 54
 - updateFramesDirection, 54
 - updatePosition, 54
- mtx
 - AudioControl, 18
- normalize
 - Audio, 11
- objectType
 - definitions.h, 86
- playInitialize
 - AudioControl, 15
- playTerminate
 - AudioControl, 16
- Player, 56
 - decreaseAlcoholLevel, 60
 - flipHorizontal, 63
 - getAmmunatiuon, 60
 - getAudioID, 65
 - getEnemiesKilled, 61
 - getFireCooldown, 60
 - getHealth, 59
 - getHeight, 64
 - getImmunityCooldown, 61
 - getInflictedDamage, 60
 - getLength, 64
 - getPosX, 64
 - getPosY, 64
 - getSpeedScale, 61
 - getSpeedX, 62
 - getSpeedY, 62
 - getType, 64
 - inJump, 61
 - increaseAlcoholLevel, 60
 - increaseAmmunation, 60
 - Player, 59
 - receiveDamage, 59
 - setAudioID, 64
 - setHealth, 59
 - setImmunityCooldown, 61
 - setPosX, 62
 - setPosY, 62
 - setSpeedX, 63
 - setSpeedY, 63
 - startJump, 61
 - swapImage, 63
 - update, 61
 - updateFramesDirection, 63
 - updatePosition, 63
- player_jump
 - definitions.h, 87
- player_walk
 - definitions.h, 87
- PowerUp, 65
 - ~PowerUp, 67
 - getAlcoholLevelBonus, 67
 - getAmmunationBonus, 67
 - getAudioID, 70
 - getHealthBonus, 67
 - getHeight, 68
 - getImmunityCooldownBonus, 67
 - getLength, 68
 - getPosX, 68
 - getPosY, 68
 - getPowerUPType, 67
 - getType, 68
 - PowerUp, 66
 - PowerUp, 66
 - setAudioID, 68
- powerup_beer
 - definitions.h, 87
- powerup_food
 - definitions.h, 87
- readSamples
 - Audio, 11
- receiveDamage
 - Enemy, 23
 - Player, 59
- reduceWorldObjects
 - Game, 34
- RenderBackground, 70
 - RenderBackground, 71
 - RenderBackground, 71

- setPos, 71
- updateBackgroundPos, 71
- updateParallaxe, 71
- RenderGUI, 72
 - RenderGUI, 73
 - RenderGUI, 73
 - setPos, 73
 - setValues, 73
- renderGraphics
 - Game, 36
- scene_collision_enemy
 - definitions.h, 87
- scene_collision_flyingbeer
 - definitions.h, 87
- scene_collision_obstacle
 - definitions.h, 87
- scene_collision_player
 - definitions.h, 87
- scene_enemy_boss
 - definitions.h, 87
- scene_enemy_security
 - definitions.h, 87
- scene_enemy_tourist
 - definitions.h, 87
- scene_flyingbeer
 - definitions.h, 87
- scoreStruct, 83
- selectFirstEntry
 - Menu, 50
- setAudioID
 - Enemy, 27
 - GameObject, 41
 - MovingObject, 55
 - Player, 64
 - PowerUp, 68
 - Shoot, 79
- setDeath
 - Enemy, 24
- setHealth
 - Enemy, 23
 - Player, 59
- setImmunityCooldown
 - Player, 61
- setPos
 - RenderBackground, 71
 - RenderGUI, 73
- setPosX
 - Enemy, 24
 - MovingObject, 53
 - Player, 62
 - Shoot, 76
- setPosY
 - Enemy, 24
 - MovingObject, 53
 - Player, 62
 - Shoot, 77
- setSpeedX
 - Enemy, 25
- MovingObject, 53
- Player, 63
- Shoot, 77
- setSpeedY
 - Enemy, 25
 - MovingObject, 54
 - Player, 63
 - Shoot, 77
- setState
 - Game, 32
- setToDelete
 - Shoot, 76
- setValues
 - RenderGUI, 73
- Shoot, 74
 - flipHorizontal, 78
 - getAudioID, 79
 - getHarming, 76
 - getHeight, 79
 - getInflictedDamage, 76
 - getLength, 78
 - getOrigin, 76
 - getPosX, 78
 - getPosY, 78
 - getSpeedX, 77
 - getSpeedY, 77
 - getType, 79
 - setAudioID, 79
 - setPosX, 76
 - setPosY, 77
 - setSpeedX, 77
 - setSpeedY, 77
 - setToDelete, 76
 - Shoot, 76
 - swapImage, 78
 - updateFramesDirection, 77
 - updatePosition, 78
- spawnDistance
 - definitions.h, 88
- start
 - Game, 32
- startJump
 - Player, 61
- startNewGame
 - Game, 33
- stateStruct, 86
- staticpaCallback
 - AudioControl, 16
- status_alcohol
 - definitions.h, 87
- status_dead
 - definitions.h, 87
- status_life
 - definitions.h, 87
- status_lifecritical
 - definitions.h, 87
- status_filter
 - AudioControl, 18

- step
 - Game, [31](#)
- swapImage
 - Enemy, [26](#)
 - MovingObject, [54](#)
 - Player, [63](#)
 - Shoot, [78](#)
- timeNeeded
 - Game, [38](#)
- timerEvent
 - Game, [32](#)
- to16bitSample
 - Audio, [11](#)
- update
 - Enemy, [24](#)
 - Player, [61](#)
- updateAudioevents
 - Game, [35](#)
- updateBackgroundPos
 - RenderBackground, [71](#)
- updateFramesDirection
 - Enemy, [25](#)
 - MovingObject, [54](#)
 - Player, [63](#)
 - Shoot, [77](#)
- updateHighScore
 - Game, [33](#)
- updateKeys
 - Input, [46](#)
- updateParallaxe
 - RenderBackground, [71](#)
- updatePlayevents
 - AudioControl, [16](#)
- updatePosition
 - Enemy, [26](#)
 - MovingObject, [54](#)
 - Player, [63](#)
 - Shoot, [78](#)
- updateScore
 - Game, [35](#)
- Wiesn-Run/src/definitions.h, [81](#)