Programozás alapjai 1 NHF - NQEHDL - Nagy Bendegúz

Generated by Doxygen 1.8.7

Sat Dec 5 2015 18:48:43

Chapter 1

Programozás alapjai 1 NHF - NQEHDL - Nagy Bendegúz

1.1 Global workflow.

The global state is handled by function pointers. Each state can define an init, flow and deinit function. When the program enters a state, it's init function is called, then it's flow function is called again and again. When a state is switched from, it's deinit function will be called.

Init and deinit function are for resource allocation and return whether it was successful, if not then the program will exit with an error. (Side note: it would be wise to define 2 more lifecycle functions, namely lazy first init and last deinit so as not to allocated resources over and over again, but i deemed it unnecessary at this scale.)

The program spends most of it's time in the main.c cycle which gets the passed time, updates input and timers and calls the flow function. If the exit flag is set, the program will exit (Main_setExitFlag()).

States can be swapped with SwapGlobalState().

1.2 Program init.

Program init is done with Main_init(), which is called once that the start of the program. Which set's up SDL, the input, timer and ttf modules and creates the function state table.

1.3 Program deinit.

Program deinit is don with Main_deinit(), which is called after the exit flag has been set and the program is about to exit. Calls deinit functions.

1.4 Brief usage.

Use the arrow keys to navigate the menu, use the numpad 5,1,2,3 and arrow key to control the first player and use wasd+fcvb keys for the second player.

1.5 Documentation.

For implementation documentation, check source code and it's comments. Each .c files has a corresponding .h file. Bigger picture is in the .h files, function documentation is in the .c files. Documentation for stuff that is defined only in the .h is in the .h file.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 AABB Struct Reference

Represents a rectangle by half width, height and the center co-ordinates.

`#include <AABB.h>`

Data Fields

- Vector2D center
- float hWidth
- float hHeight

5.1.1 Detailed Description

Represents a rectangle by half width, height and the center co-ordinates.

5.1.2 Field Documentation

5.1.2.1 Vector2D AABB::center

5.1.2.2 float AABB::hHeight

5.1.2.3 float AABB::hWidth

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/AABB.h

5.2 AttackData Struct Reference

Each player has one of these, hold data for the attacking state.

`#include <player.h>`

Data Fields

- int isLive

- int usedUp
- Vector2D relPos
- Object ∗ box
- int attCD

5.2.1 Detailed Description

Each player has one of these, hold data for the attacking state.

5.2.2 Field Documentation

5.2.2.1 int AttackData::attCD

5.2.2.2 Object∗ AttackData::box

5.2.2.3 int AttackData::isLive

5.2.2.4 Vector2D AttackData::relPos

5.2.2.5 int AttackData::usedUp

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/HEAD/player.h

5.3 Bag Struct Reference

Holds a dynamically growing array.

```
#include <bag.h>
```

Data Fields

- void ∗∗ vector
- int elemCount
- int maxSize
- freeData freeDataPtr

5.3.1 Detailed Description

Holds a dynamically growing array.

5.3.2 Field Documentation

5.3.2.1 int Bag::elemCount

5.3.2.2 freeData Bag::freeDataPtr

5.3.2.3 int Bag::maxSize

5.3.2.4 void** Bag::vector

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Utility/HEAD/bag.h

5.4 DashData Struct Reference

Each player has one of these, hold data for the dashing state.

`#include <player.h>`

Data Fields

- int dashCD
- int isLive
- Vector2D dir

5.4.1 Detailed Description

Each player has one of these, hold data for the dashing state.

5.4.2 Field Documentation

5.4.2.1 int DashData::dashCD

5.4.2.2 Vector2D DashData::dir

5.4.2.3 int DashData::isLive

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/HEAD/player.h

5.5 Fonts Struct Reference

Internal representation of the global font in a given size.

Data Fields

- int size
- TTF_Font * font

5.5.1 Detailed Description

Internal representation of the global font in a given size.

These are made by means of lazy initialization.

5.5.2 Field Documentation

5.5.2.1 TTF_Font∗ Fonts::font

5.5.2.2 int Fonts::size

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Graphics/SRC/textsprite.c

5.6 LevelSel_data Struct Reference

Data Fields

- int enterDown
- int escDown
- LevelSel_Options currSel

5.6.1 Field Documentation

5.6.1.1 LevelSel_Options LevelSel_data::currSel

5.6.1.2 int LevelSel_data::enterDown

5.6.1.3 int LevelSel_data::escDown

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/SRC/LevelSelState.c

5.7 MAIN_DATA Struct Reference

Main specific data.

Data Fields

- int keepRunning

  The exit flag.
- uint32_t delta

  The the passed since the last main cycle run.

5.7.1 Detailed Description

Main specific data.

5.7.2 Field Documentation

5.7.2.1 uint32_t MAIN_DATA::delta

The the passed since the last main cycle run.

5.7.2.2 int MAIN_DATA::keepRunning

The exit flag.

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/SRC/main.c

5.8 Menu_data Struct Reference

Main menu specific data.

Data Fields

- int enterDown
- Menu_options currSel

    Currently selected menu option.

5.8.1 Detailed Description

Main menu specific data.

5.8.2 Field Documentation

5.8.2.1 Menu_options Menu_data::currSel

Currently selected menu option.

5.8.2.2 int Menu_data::enterDown

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/SRC/MenuState.c

5.9 Object Struct Reference

```
#include <physics.h>
```

Data Fields

- World ∗ world

    The world this object belongs to.
- int oHandle

Do not modify, index at which this object is stored in the World.

- PH_OBJ_TYPE type
- Vector2D forceSum
- Vector2D velocity
- float velCapX

  Maximum velocity on the X axis.
- float velCapY

  Maximum velocity on the Y axis.
- float invMass
- AABB aabb

  The dimension of the object.
- Vector2D lastPos

  Position of the object before the last position integration.
- UserData userData
- PH_callback callBack

  Collision callback.
- void ∗ cbState

  Callback state passed to the callback function.
- SDL_Color color

  This does not belong here, used for convenient rendering.

5.9.1 Field Documentation

5.9.1.1 AABB Object::aabb

The dimension of the object.

5.9.1.2 PH_callback Object::callBack

Collision callback.

5.9.1.3 void∗ Object::cbState

Callback state passed to the callback function.

5.9.1.4 SDL_Color Object::color

This does not belong here, used for convenient rendering.

5.9.1.5 Vector2D Object::forceSum

5.9.1.6 float Object::invMass

5.9.1.7 Vector2D Object::lastPos

Position of the object before the last position integration.

5.9.1.8 int Object::oHandle

Do not modify, index at which this object is stored in the World.

5.9.1.9 PH_OBJ_TYPE Object::type

5.9.1.10 UserData Object::userData

5.9.1.11 float Object::velCapX

Maximum velocity on the X axis.

5.9.1.12 float Object::velCapY

Maximum velocity on the Y axis.

5.9.1.13 Vector2D Object::velocity

5.9.1.14 World∗ Object::world

The world this object belongs to.

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/physics.h

5.10 PH_Manifold Struct Reference

```
#include <physics.h>
```

Data Fields

- Object ∗ A
- Object ∗ B
- PH_COLL_TYPE type
- Vector2D n

    Normal vector of the collision.
- float depth

    Penetration depth.

5.10.1 Field Documentation

5.10.1.1 Object∗ PH_Manifold::A

5.10.1.2 Object∗ PH_Manifold::B

5.10.1.3 float PH_Manifold::depth

Penetration depth.

5.10.1.4 Vector2D PH_Manifold::n

Normal vector of the collision.

5.10.1.5 PH_COLL_TYPE PH_Manifold::type

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/physics.h

5.11 Player Struct Reference

Holds every data defining a player.

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

Data Fields

- World ∗ world
- Object ∗ phObj
- SDL_Keycode keys [8]
- stateFunc state
- stateFunc movState
- uint32_t contKeyDown
- uint32_t keyDown
- uint32_t flags
- int score
- AttackData attData
- DashData dashData
- ShootData shData

5.11.1 Detailed Description

Holds every data defining a player.

5.11.2 Field Documentation

5.11.2.1 AttackData Player::attData

5.11.2.2 uint32_t Player::contKeyDown

5.11.2.3 DashData Player::dashData

5.11.2.4 uint32_t Player::flags

5.11.2.5 uint32_t Player::keyDown

5.11.2.6 SDL_Keycode Player::keys[8]

5.11.2.7 stateFunc Player::movState

5.11.2.8 Object∗ Player::phObj

5.11.2.9 int Player::score

5.11.2.10 ShootData Player::shData

5.11.2.11 stateFunc Player::state

5.11.2.12 World∗ Player::world

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/HEAD/player.h

5.12 ShootData Struct Reference

Each player has one of these, hold data for the shooting state.

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

Data Fields

- int shootCD
- int shootCount
- Bag ∗ bag

5.12.1 Detailed Description

Each player has one of these, hold data for the shooting state.

5.12.2 Field Documentation

5.12.2.1 Bag∗ ShootData::bag

5.12.2.2 int ShootData::shootCD

5.12.2.3 int ShootData::shootCount

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Game/HEAD/player.h

5.13 Subscriber Struct Reference

Internal representation of subscribed functions.

Data Fields

- inputConsumer consFuc
- void ∗ state

5.13.1 Detailed Description

Internal representation of subscribed functions.

5.13.2 Field Documentation

5.13.2.1 inputConsumer Subscriber::consFuc

5.13.2.2 void∗ Subscriber::state

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Events/SRC/input.c

5.14 TextSprite Struct Reference

Internal representation of a text object.

Data Fields

- SDL_Texture ∗ textTure
- SDL_Rect dest

5.14.1 Detailed Description

Internal representation of a text object.

5.14.2 Field Documentation

5.14.2.1 SDL_Rect TextSprite::dest

5.14.2.2 SDL_Texture∗ TextSprite::textTure

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Graphics/SRC/textsprite.c

5.15 Timed_event Struct Reference

```
#include <Timer_man.h>
```

Data Fields

- int id
- Timer ∗ timer
- Timer_callBack callBack
- void ∗ state

5.15.1 Field Documentation

5.15.1.1 Timer_callBack Timed_event::callBack

5.15.1.2 int Timed_event::id

5.15.1.3 void∗ Timed_event::state

5.15.1.4 Timer∗ Timed_event::timer

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Events/HEAD/Timer_man.h

5.16 Timer Struct Reference

Internal representation of time-keeping objects.

Data Fields

- Uint32 ticksRunning
- int isStarted

5.16.1 Detailed Description

Internal representation of time-keeping objects.

5.16.2 Field Documentation

5.16.2.1 int Timer::isStarted

5.16.2.2 Uint32 Timer::ticksRunning

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Events/SRC/timer.c

5.17 UserData Struct Reference

Each object has one, can be set freely, type identifies the void∗ type.

```
#include <physics.h>
```

Data Fields

- UserDataType type
- void ∗ data

5.17.1 Detailed Description

Each object has one, can be set freely, type identifies the void∗ type.

5.17.2 Field Documentation

5.17.2.1 void∗ UserData::data

5.17.2.2 UserDataType UserData::type

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/physics.h

5.18 Vector2D Struct Reference

Represents two vectors in floats, for performance reasons.

```
#include <vector.h>
```

Data Fields

- float x
- float y

5.18.1 Detailed Description

Represents two vectors in floats, for performance reasons.

5.18.2 Field Documentation

5.18.2.1 float Vector2D::x

5.18.2.2 float Vector2D::y

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Utility/HEAD/vector.h

5.19 World Struct Reference

```
#include <physics.h>
```

Data Fields

- Bag ∗ dynObjBag
- Bag ∗ stObjBag
- Bag ∗ hybObjBag
- Vector2D gravity
- double stepTime
- double deltaLeftover

5.19.1 Field Documentation

5.19.1.1 double World::deltaLeftover

5.19.1.2 Bag∗ World::dynObjBag

5.19.1.3 Vector2D World::gravity

5.19.1.4 Bag∗ World::hybObjBag

5.19.1.5 double World::stepTime

5.19.1.6 Bag∗ World::stObjBag

The documentation for this struct was generated from the following file:

- /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/physics.h

Chapter 6

File Documentation

6.1 /home/bendeguz/ClionProjects/Dummy/cmake/readme.md File Reference

6.2 /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/AABB.h File Reference

Module for representing rectangles by means of half widths and heights and the center co-ordinates.

```
#include <SDL2/SDL.h>
#include "../../Utility/HEAD/vector.h"
```

Data Structures

- struct AABB

  Represents a rectangle by half width, height and the center co-ordinates.

Typedefs

- typedef struct AABB AABB

  Represents a rectangle by half width, height and the center co-ordinates.

Functions

- int AABB_vs_AABB (AABB ∗a, AABB ∗b)

  Checks two AABBs for overlap.
- int AABB_vs_Point (AABB ∗a, float x, float y)

  Checks if the AABB contains a given point.
- void AABB_renderColor (AABB ∗a, SDL_Color c)

  Convenience function for rendering an AABB with a given colour, relies on global gRenderer reference.

6.2.1 Detailed Description

Module for representing rectangles by means of half widths and heights and the center co-ordinates.

Author

Bendegúz Nagy

Create new AABBs on the stack or by means of dynamic memory allocation.

6.2.2 Typedef Documentation

6.2.2.1 typedef struct AABB AABB

Represents a rectangle by half width, height and the center co-ordinates.

6.2.3 Function Documentation

6.2.3.1 void AABB_renderColor (AABB ∗ *a*, SDL_Color *c*)

Convenience function for rendering an AABB with a given colour, relies on global gRenderer reference.

Parameters

|>p0.15|p0.805|

a	the AABB to be drawn.
c	the colour the AABB should be drawn with.

6.2.3.2 int AABB_vs_AABB (AABB ∗ *a*, AABB ∗ *b*)

Checks two AABBs for overlap.

Parameters

|>p0.15|p0.805|

a	First AABB.
b	Other AABB.

Returns

0 if they overlap, non-zero otherwise.

6.2.3.3 int AABB_vs_Point (AABB ∗ *a*, float *x*, float *y*)

Checks if the AABB contains a given point.

Parameters

|>p0.15|p0.805|

a	the AABB to be checked.
x	the X co-ordinate of the point.
y	the Y co-ordinate of the point.

Returns

0 zero if not, non-zero otherwise

6.3 /home/bendeguz/ClionProjects/Dummy/Collision/HEAD/physics.h File Reference

Primitive physics engine.

```
#include <stdint.h>
#include "../../Utility/HEAD/vector.h"
#include "../../Utility/HEAD/bag.h"
#include "AABB.h"
```

Data Structures

- struct UserData

  Each object has one, can be set freely, type identifies the void∗ type.
- struct World
- struct Object
- struct PH_Manifold

Typedefs

- typedef enum PH_OBJ_TYPE PH_OBJ_TYPE

  Defines the type of an object, static, dynamic, hybrid.
- typedef struct World World

  Defines the context for the physics world, objects can collide with other objects from the same World.
- typedef struct Object Object

  Representation of an object in the world, only AABBs are supported currently.
- typedef enum PH_COLL_TYPE PH_COLL_TYPE

  Used in the generated PH_manifolds to identify collision types.
- typedef struct PH_Manifold PH_Manifold

  Holds data about a collision, used be the collision resolution function.
- typedef int(∗ PH_callback)(PH_Manifold ∗m, Object ∗callObj, Object ∗collObj, void ∗state)

  Objects can have collision callbacks set to them, they have to adhere to this signature.
- typedef enum UserDataType UserDataType

  Each object can have a void∗ userData and a userDataType bind data to objects.
- typedef struct UserData UserData

  Each object has one, can be set freely, type identifies the void∗ type.

Enumerations

- enum UserDataType {
  NONE, BLOCK, PLAYER, WALL,
  ATTACKBOX, BULLET }

  Each object can have a void∗ userData and a userDataType bind data to objects.
- enum PH_OBJ_TYPE { STATIC = 1, HYBRID = 2, DYNAMIC = 4 }
- enum PH_COLL_TYPE { HYBRID_HYBRID, STATIC_DYNAMIC, HYBRID_DYNAMIC, DYNAMIC_DYNA↩
  MIC }

Functions

- World ∗ PH_createWorld ()

  Creates an empty world.
- Object ∗ PH_createBox (int x, int y, int width, int height, float mass, PH_OBJ_TYPE type, World ∗world)

  Creates an objects at x,y co-ord with given heigh, width, type and mass in the given world.
- void PH_setStepTime (double delta, World ∗world)

  Sets the time by which the world will be stepped.
- void PH_setGravity (float gravityX, float gravityY, World ∗world)

  Sets the gravity of a world, will only have apply after the next PH_stepWorld().
- void PH_stepWorld (double delta, World ∗world)

  Asks the world to update the objects with an amount of passed time. Collisions will be resolved and callbacks called.
- void PH_impulse (Vector2D ∗impulse, Object ∗obj)

  Applies an impulse to an object.
- void PH_force (Vector2D ∗force, Object ∗obj)

  Applies a force to an object, forces are cleared after a PH_stepWorld().
- void PH_setVelCap (float capX, float capY, Object ∗obj)

  Sets the velocity cap, calling this during a callback is undefined.
- void PH_setPosition (Vector2D vec, Object ∗obj)

  Sets a the position of an object, calling this during a callback is undefined.
- void PH_setUData (void ∗data, UserDataType type, Object ∗obj)

  Sets the userdata and it's type for an object.
- void PH_destroyObject (Object ∗o)

  Deletes an object from the world, doing this during a callback will result in undefined behaviour.
- void PH_destroyWorld (World ∗world)

  Free up all the memory the objects and the world take up.
- void PH_setCallback (PH_callback callBack, void ∗state, Object ∗obj)

  Set the callback function for an object and it's related state pointer.
- void PH_queryPoint (Vector2D point, PH_OBJ_TYPE types, int cap, Bag ∗bag, World ∗world)

  Clear and fill the passed Bag∗ with the Objects that contain the point.
- void PH_renderObjects (World ∗world)

  Render the objects by their colours.
- void PH_setColor (Uint8 r, Uint8 g, Uint8 b, Uint8 a, Object ∗o)

  Set the color of the object, can be used for convenient rendering.

6.3.1 Detailed Description

Primitive physics engine.

Author

Bendegúz Nagy

This is a primitive physics engine implementation. I could have like used convex polygons and the separating axis theorem as a basis, but I'm not smart enough for that and it would be an overkill anyways.

It has three types of objects, and all of them are AABBs. Dynamic object, which can be acted upon by forces and gravity applies to them, they can collide with STATIC and HYBRID objects, collision resolution applies to them exclusively.

Hybrid objects which can collide with static and hybrid objects for the sole purpose of collision callback. Gravity does not apply to them, but they can be moved by forces.

Static objects, which do not move.

It works by first integrating their velocity according to the forces applied to the objects then integrating their position, then checking each possible combination of objects for overlap. DYNAMIC vs HYBRID DYNAMIC vs STATIC HYBRID vs STATIC HYBRID vs HYBRID

6.3.2 Typedef Documentation

6.3.2.1 typedef struct Object Object

Representation of an object in the world, only AABBs are supported currently.

6.3.2.2 typedef int(∗ PH_callback)(PH_Manifold ∗m, Object ∗callObj, Object ∗collObj, void ∗state)

Objects can have collision callbacks set to them, they have to adhere to this signature.

Object collision callback function. Return value indicates whether to collision should be allowed. The second argument is the callback object, the third is the object with which it collided. The fourth argument is an optional state pointer, set at callback registration.

6.3.2.3 typedef enum PH_COLL_TYPE PH_COLL_TYPE

Used in the generated PH_manifolds to identify collision types.

Can be: DYNAMIC vs HYBRID DYNAMIC vs STATIC HYBRID vs STATIC HYBRID vs HYBRID

6.3.2.4 typedef struct PH_Manifold PH_Manifold

Holds data about a collision, used be the collision resolution function.

6.3.2.5 typedef enum PH_OBJ_TYPE PH_OBJ_TYPE

Defines the type of an object, static, dynamic, hybrid.

Defines the type of an object, static, dynamic, hybrid. Currently only these combinations are able to collide: DYN↩
AMIC vs HYBRID DYNAMIC vs STATIC HYBRID vs STATIC HYBRID vs HYBRID Collision resolution only applies to dynamic objects in the form of correcting their position for no overlap with any other object. No impulse is applied. Collision callbacks apply to all of the above. Forces can only act on hybrid and dynamic objects. Gravity only applies to dynamic objects.

6.3.2.6 typedef struct UserData UserData

Each object has one, can be set freely, type identifies the void∗ type.

6.3.2.7 typedef enum UserDataType UserDataType

Each object can have a void∗ userData and a userDataType bind data to objects.

6.3.2.8 typedef struct World World

Defines the context for the physics world, objects can collide with other objects from the same World.

6.3.3 Enumeration Type Documentation

6.3.3.1 enum typedef enum PH_COLL_TYPE PH_COLL_TYPE

Enumerator

HYBRID_HYBRID

STATIC_DYNAMIC

HYBRID_DYNAMIC

DYNAMIC_DYNAMIC

6.3.3.2 enum typedef enum PH_OBJ_TYPE PH_OBJ_TYPE

Enumerator

STATIC

HYBRID

DYNAMIC

6.3.3.3 enum UserDataType

Each object can have a void∗ userData and a userDataType bind data to objects.

Enumerator

NONE Default.

BLOCK Destroyable piece of wall.

PLAYER Object representing a player.

WALL Indestructible wall blocks.

ATTACKBOX Attackbox spawned when a player attacks, used for collision callbacks.

BULLET Bullet object spawned when the player shoots, used for collision callbacks.

6.3.4 Function Documentation

6.3.4.1 Object∗ PH_createBox (int *x,* int *y,* int *width,* int *height,* float *mass,* PH_OBJ_TYPE *type,* World ∗ *world*)

Creates an objects at x,y co-ord with given heigh, width, type and mass in the given world.

6.3.4.2 World∗ PH_createWorld ( )

Creates an empty world.

6.3.4.3 void PH_destroyObject (Object ∗ *o*)

Deletes an object from the world, doing this during a callback will result in undefined behaviour.

6.3.4.4 void PH_destroyWorld (World ∗ *world*)

Free up all the memory the objects and the world take up.

6.3.4.5 void PH_force (Vector2D ∗ *force,* Object ∗ *obj*)

Applies a force to an object, forces are cleared after a PH_stepWorld().

6.3.4.6 void PH_impulse (Vector2D ∗ *impulse,* Object ∗ *obj*)

Applies an impulse to an object.

6.3.4.7 void PH_queryPoint (Vector2D *point,* PH_OBJ_TYPE *types,* int *cap,* Bag ∗ *bag,* World ∗ *world*)

Clear and fill the passed Bag∗ with the Objects that contain the point.

Parameters

|>p0.15|p0.805|

PH_OBJ_TYPE can be used for type specification, OR'ing together types

cap can be used for specifing max find count, negative if no cap

6.3.4.8 void PH_renderObjects (World ∗ *world*)

Render the objects by their colours.

6.3.4.9 void PH_setCallback (PH_callback *callBack,* void ∗ *state,* Object ∗ *obj*)

Set the callback function for an object and it's related state pointer.

6.3.4.10 void PH_setColor (Uint8 *r,* Uint8 *g,* Uint8 *b,* Uint8 *a,* Object ∗ *o*)

Set the color of the object, can be used for convenient rendering.

6.3.4.11 void PH_setGravity (float *gravityX,* float *gravityY,* World ∗ *world*)

Sets the gravity of a world, will only have apply after the next PH_stepWorld().

6.3.4.12 void PH_setPosition (Vector2D *vec,* Object ∗ *obj*)

Sets a the position of an object, calling this during a callback is undefined.

6.3.4.13 void PH_setStepTime (double *delta,* World ∗ *world*)

Sets the time by which the world will be stepped.

6.3.4.14 void PH_setUData (void ∗ *data,* UserDataType *type,* Object ∗ *obj*)

Sets the userdata and it's type for an object.

6.3.4.15 void PH_setVelCap (float *capX,* float *capY,* Object ∗ *obj*)

Sets the velocity cap, calling this during a callback is undefined.

6.3.4.16 void PH_stepWorld (double *delta,* World ∗ *world*)

Asks the world to update the objects with an amount of passed time. Collisions will be resolved and callbacks called.

6.4 /home/bendeguz/ClionProjects/Dummy/Collision/SRC/AABB.c File Reference

```
#include <stdlib.h>
#include <math.h>
#include "../HEAD/AABB.h"
#include "../../Graphics/HEAD/graphics_man.h"
```

Functions

- int AABB_vs_AABB (AABB ∗a, AABB ∗b)

  Checks two AABBs for overlap.
- int AABB_vs_Point (AABB ∗a, float x, float y)

  Checks if the AABB contains a given point.
- void AABB_renderColor (AABB ∗a, SDL_Color c)

  Convenience function for rendering an AABB with a given colour, relies on global gRenderer reference.

6.4.1 Function Documentation

6.4.1.1 void AABB_renderColor (AABB ∗ *a*, SDL_Color *c*)

Convenience function for rendering an AABB with a given colour, relies on global gRenderer reference.

Parameters

|>p0.15|p0.805|

a	the AABB to be drawn.
c	the colour the AABB should be drawn with.

6.4.1.2 int AABB_vs_AABB (AABB ∗ *a*, AABB ∗ *b*)

Checks two AABBs for overlap.

Parameters

|>p0.15|p0.805|

a	First AABB.
b	Other AABB.

Returns

0 if they overlap, non-zero otherwise.

6.4.1.3 int AABB_vs_Point (AABB ∗ *a*, float *x*, float *y*)

Checks if the AABB contains a given point.

Parameters

|>p0.15|p0.805|

a the AABB to be checked.

x the X co-ordinate of the point.

y the Y co-ordinate of the point.

Returns

0 zero if not, non-zero otherwise

6.5 /home/bendeguz/ClionProjects/Dummy/Collision/SRC/physics.c File Reference

```
#include <float.h>
#include "../HEAD/physics.h"
```

Macros

- #define PH_DEF_STEPTIME (1.0/60.0)

  No matter how much time we pass to PH_stepWorld(), it will chunk it up into this length.
- #define PH_SPIRAL_OF_DEATH_CAP (0.25)

  Prevents a sprial of death.

Functions

- void PH_integrate (double delta, World ∗world)

  Private, integrates the position of the objects, called from PH_stepWorld(),.
- void PH_capVelocity (Object ∗obj)

  Private, caps the velocities, called from PH_integrate(),.
- void PH_resetForces (World ∗world)

  Private, resets the forces to gravity or zero depending on the object type, called from PH_stepWorld(),.
- void PH_testAndResolve (World ∗world)

  Private, called from PH_stepWorld(), detects and resolves collisions.
- void PH_testTwoObjects (Object ∗A, Object ∗B, PH_COLL_TYPE type, PH_Manifold ∗m)

  Private, used in PH_testAndResolve(), tests two objects for collision and resolves it, call callbacks.
- int PH_testOverlap (Object ∗A, Object ∗B)

  Private, used in PH_testTwoObjects(), tests by overlap.
- void PH_generateManifold (Object ∗objA, Object ∗objB, PH_COLL_TYPE type, PH_Manifold ∗manifold)

  Private, used in Ph_testTwoObjects(), generates collision manifold.
- int PH_testCallback (Object ∗A, Object ∗B, PH_Manifold ∗m)

  Private, used in PH_testTwoObjects(), return whether the callbacks allow for collision, if they don't exit then they allow.
- void PH_resolveCollision (PH_Manifold ∗m)

  Private, used in PH_testTwoObjects(), resolves overlap for DYNAMIC vs DYNAMIC, does nothing for anything else.
- World ∗ PH_createWorld ()

  Creates an empty world.
- Object ∗ PH_createBox (int x, int y, int width, int height, float mass, PH_OBJ_TYPE type, World ∗world)

  Creates an objects at x,y co-ord with given heigh, width, type and mass in the given world.
- void PH_stepWorld (double delta, World ∗world)

Asks the world to update the objects with an amount of passed time. Collisions will be resolved and callbacks called.

- void PH_setStepTime (double delta, World ∗world)

    Sets the time by which the world will be stepped.
- void PH_setGravity (float gravityX, float gravityY, World ∗world)

    Sets the gravity of a world, will only have apply after the next PH_stepWorld().
- void PH_impulse (Vector2D ∗impulse, Object ∗obj)

    Applies an impulse to an object.
- void PH_force (Vector2D ∗force, Object ∗obj)

    Applies a force to an object, forces are cleared after a PH_stepWorld().
- void PH_destroyObject (Object ∗o)

    Deletes an object from the world, doing this during a callback will result in undefined behaviour.
- void PH_destroyWorld (World ∗world)

    Free up all the memory the objects and the world take up.
- void PH_setVelCap (float capX, float capY, Object ∗obj)

    Sets the velocity cap, calling this during a callback is undefined.
- void PH_setPosition (Vector2D vec, Object ∗obj)

    Sets a the position of an object, calling this during a callback is undefined.
- void PH_setUData (void ∗data, UserDataType type, Object ∗obj)

    Sets the userdata and it's type for an object.
- void PH_setCallback (PH_callback callBack, void ∗state, Object ∗obj)

    Set the callback function for an object and it's related state pointer.
- void PH_setColor (Uint8 r, Uint8 g, Uint8 b, Uint8 a, Object ∗o)

    Set the color of the object, can be used for convenient rendering.
- void PH_renderObjects (World ∗world)

    Render the objects by their colours.
- void PH_queryPoint (Vector2D point, PH_OBJ_TYPE types, int cap, Bag ∗bag, World ∗world)

    Clear and fill the passed Bag∗ with the Objects that contain the point.

6.5.1 Macro Definition Documentation

6.5.1.1 #define PH_DEF_STEPTIME (1.0/60.0)

No matter how much time we pass to PH_stepWorld(), it will chunk it up into this length.

6.5.1.2 #define PH_SPIRAL_OF_DEATH_CAP (0.25)

Prevents a sprial of death.

6.5.2 Function Documentation

6.5.2.1 void PH_capVelocity (Object ∗ *obj*)

Private, caps the velocities, called from PH_integrate(),.

6.5.2.2 Object∗ PH_createBox (int *x,* int *y,* int *width,* int *height,* float *mass,* PH_OBJ_TYPE *type,* World ∗ *world*)

Creates an objects at x,y co-ord with given heigh, width, type and mass in the given world.

6.5.2.3 World∗ PH_createWorld ( )

Creates an empty world.

6.5.2.4 void PH_destroyObject (Object ∗ *o*)

Deletes an object from the world, doing this during a callback will result in undefined behaviour.

6.5.2.5 void PH_destroyWorld (World ∗ *world*)

Free up all the memory the objects and the world take up.

6.5.2.6 void PH_force (Vector2D ∗ *force,* Object ∗ *obj*)

Applies a force to an object, forces are cleared after a PH_stepWorld().

6.5.2.7 void PH_generateManifold (Object ∗ *objA,* Object ∗ *objB,* PH_COLL_TYPE *type,* PH_Manifold ∗ *manifold*)

Private, used in Ph_testTwoObjects(), generates collision manifold.

6.5.2.8 void PH_impulse (Vector2D ∗ *impulse,* Object ∗ *obj*)

Applies an impulse to an object.

6.5.2.9 void PH_integrate (double *delta,* World ∗ *world*)

Private, integrates the position of the objects, called from PH_stepWorld(),.

6.5.2.10 void PH_queryPoint (Vector2D *point,* PH_OBJ_TYPE *types,* int *cap,* Bag ∗ *bag,* World ∗ *world*)

Clear and fill the passed Bag∗ with the Objects that contain the point.

Parameters

|>p0.15|p0.805|

PH_OBJ_TYPE	can be used for type specification, OR'ing together types
cap	can be used for specifing max find count, negative if no cap

6.5.2.11 void PH_renderObjects (World ∗ *world*)

Render the objects by their colours.

6.5.2.12 void PH_resetForces (World ∗ *world*)

Private, resets the forces to gravity or zero depending on the object type, called from PH_stepWorld(),.

6.5.2.13 void PH_resolveCollision (PH_Manifold ∗ *m*)

Private, used in PH_testTwoObjects(), resolves overlap for DYNAMIC vs DYNAMIC, does nothing for anything else.

6.5.2.14 void PH_setCallback (PH_callback *callBack,* void ∗ *state,* Object ∗ *obj*)

Set the callback function for an object and it's related state pointer.

6.5.2.15 void PH_setColor (Uint8 *r,* Uint8 *g,* Uint8 *b,* Uint8 *a,* Object ∗ *o*)

Set the color of the object, can be used for convenient rendering.

6.5.2.16 void PH_setGravity (float *gravityX,* float *gravityY,* World ∗ *world*)

Sets the gravity of a world, will only have apply after the next PH_stepWorld().

6.5.2.17 void PH_setPosition (Vector2D *vec,* Object ∗ *obj*)

Sets a the position of an object, calling this during a callback is undefined.

6.5.2.18 void PH_setStepTime (double *delta,* World ∗ *world*)

Sets the time by which the world will be stepped.

6.5.2.19 void PH_setUData (void ∗ *data,* UserDataType *type,* Object ∗ *obj*)

Sets the userdata and it's type for an object.

6.5.2.20 void PH_setVelCap (float *capX,* float *capY,* Object ∗ *obj*)

Sets the velocity cap, calling this during a callback is undefined.

6.5.2.21 void PH_stepWorld (double *delta,* World ∗ *world*)

Asks the world to update the objects with an amount of passed time. Collisions will be resolved and callbacks called.

6.5.2.22 void PH_testAndResolve (World ∗ *world*)

Private, called from PH_stepWorld(), detects and resolves collisions.

6.5.2.23 int PH_testCallback (Object ∗ *A,* Object ∗ *B,* PH_Manifold ∗ *m*)

Private, used in PH_testTwoObjects(), return whether the callbacks allow for collision, if they don't exit then they allow.

6.5.2.24 int PH_testOverlap (Object ∗ *A,* Object ∗ *B*)

Private, used in PH_testTwoObjects(), tests by overlap.

6.5.2.25 void PH_testTwoObjects (Object ∗ *A,* Object ∗ *B,* PH_COLL_TYPE *type,* PH_Manifold ∗ *m*)

Private, used in PH_testAndResolve(), tests two objects for collision and resolves it, call callbacks.

6.6 /home/bendeguz/ClionProjects/Dummy/Events/HEAD/input.h File Reference

Module for handling SDL input with push-down implementation.

Typedefs

- typedef int(∗ inputConsumer)(SDL_Event ∗e, void ∗state)

  Registered functions have to adhere to this signature.

Functions

- void Input_init ()

  Initializes the module.
- void Input_deinit ()

  Deinitializes the module.
- void Input_subscribe (inputConsumer cons, void ∗state)

  Subscribe functions and their state pointers with this.
- void Input_process ()

  Process accumulated SDL_Events by pushing them down the inputConsumer queue.
- void Input_clear ()

  Resets the list of subscribed functions.

6.6.1 Detailed Description

Module for handling SDL input with push-down implementation.

Author

Bendegúz Nagy

This module maintains an ordered list of inputConsumer functions. When Input_process() is called, the module calls SDL_PollEvent() until there are no events left and feeds the SDL_Events to the registered inputConsumers. If one of those functions returns non-zero, then the SDL_Event is considered consumed and will not be passed to successive functions in the list.

Initialize the module with Input_init(), deinitialize with Input_deinit(). When you wish to process the events accumulated in SDL, call Input_process(). Register inputConsumers with Input_subscribe(). To empty the maintained list, call Input_clear().

6.6.2 Typedef Documentation

6.6.2.1 typedef int(∗ inputConsumer)(SDL_Event ∗e, void ∗state)

Registered functions have to adhere to this signature.

Parameters

|>p0.15|p0.805|

e	Pointer to the SDL_Event to be processed.
state	Optional state pointer set at function registration to be passed to the function with each call

Returns

0 if the event should be considered consumed, non-zero otherwise

6.6.3 Function Documentation

6.6.3.1 void Input_clear ( )

Resets the list of subscribed functions.

6.6.3.2 void Input_deinit ( )

Deinitializes the module.

6.6.3.3 void Input_init ( )

Initializes the module.

Initializes the module. It is mandatory to call this before the module is put to use in any way. Calling this more than once without calling Input_deinit() in between will cause a memory leak.

6.6.3.4 void Input_process ( )

Process accumulated SDL_Events by pushing them down the inputConsumer queue.

6.6.3.5 void Input_subscribe (inputConsumer *cons,* void ∗ *state*)

Subscribe functions and their state pointers with this.

Parameters

|>p0.15|p0.805|

cons	Function pointer to the inputConsumer function.
state	Optional state pointer to be passed to the function with each call.

Subscribe functions and their state pointers with this. The processing will be done in the order the functions are registered.

6.7 /home/bendeguz/ClionProjects/Dummy/Events/HEAD/timer.h File Reference

Module for object which track time.

```
#include <SDL2/SDL.h>
```

Typedefs

- typedef struct Timer Timer

Functions

- Uint32 getDelta ()

    Returns the time between now and the last time this function was called.
- Timer ∗ Timer_new ()

    Creates a paused Timer object with zero ticks.
- void Timer_free (Timer ∗ptr)

    Deallocates the memory allocated by Timer_new()
- void Timer_start (Timer ∗ptr)

    Sets the isStarted flag of the Timer object to true.
- void Timer_updateDelta (Uint32 delta, Timer ∗ptr)

    Updates a Timer with a given number of ms.
- Uint32 Timer_getTicks (Timer ∗ptr)

    Returns the number of ms held in the Timer object.

6.7.1 Detailed Description

Module for object which track time.

Author

Bendegúz Nagy

This s a module for time-keeping objects. They can be started, then updated with the number of ms passed. The implementation is minimal, because this project had no use for more.

It also features a function getDelta(), which will return the number of ticks passed since it was called.

Create the objects with Timer_new(), destroy them with Timer_free(). Start them with Timer_start(), they won't accept ticks if they hadn't been started. Update them with Timer_updateDelta(), retrieve their time with Timer_get↩ Ticks().

6.7.2 Typedef Documentation

6.7.2.1 typedef struct Timer Timer

6.7.3 Function Documentation

6.7.3.1 Uint32 getDelta ( )

Returns the time between now and the last time this function was called.

Returns

The time between now and the last time this function was called.

6.7.3.2 void Timer_free (Timer ∗ *ptr*)

Deallocates the memory allocated by Timer_new()

Parameters

|>p0.15|p0.805|

ptr The Timer object to be destroyed.

6.7.3.3 Uint32 Timer_getTicks (Timer ∗ *ptr*)

Returns the number of ms held in the Timer object.

Parameters

|>p0.15|p0.805|

ptr The Timer from which the elapsed time should be extracted.

Returns

Time in ms held by this Timer object.

6.7.3.4 Timer∗ Timer_new ( )

Creates a paused Timer object with zero ticks.

Returns

Returns the newly allocated Timer object.

6.7.3.5 void Timer_start (Timer ∗ *ptr*)

Sets the isStarted flag of the Timer object to true.

Parameters

|>p0.15|p0.805|

ptr The timer object to start.

Sets the isStarted flag of the Timer object to true. It is necessary to call this for a Timer objects, otherwise it won't accept passed time updates.

6.7.3.6 void Timer_updateDelta (Uint32 *delta,* Timer ∗ *ptr*)

Updates a Timer with a given number of ms.

Parameters

|>p0.15|p0.805|

delta The amount of time passed.

ptr The Timer to be updated.

Updates a Timer with a given number of ms. The Timer has to be started for this to take any effect

6.8 /home/bendeguz/ClionProjects/Dummy/Events/HEAD/Timer_man.h File Reference

Module for handling timed events, built upon Timer objects.

```
#include "../HEAD/Timer_man.h"
#include "timer.h"
```

Data Structures

- struct Timed_event

Typedefs

- typedef int(∗ Timer_callBack)(Uint32 delta, Timer ∗timer, void ∗state)

  Timed_event function pointers have to adhere to this signature.
- typedef struct Timed_event Timed_event

Functions

- void TM_init ()

  Initialize the module.
- void TM_deinit ()

  Deinitialize the module.
- Timed_event ∗ TM_new (Timer_callBack callBack, void ∗state)

  Create a new timed event and register it.
- void TM_freeTimed_event (Timed_event ∗e)

  Deallocate a Timed_event, this can also be done automatically.
- void TM_process (Uint32 delta)

  Process the Timed_events with the elapsed time passed in ms.
- void TM_clear ()

  Destroy each Timed_event.

6.8.1 Detailed Description

Module for handling timed events, built upon Timer objects.

Author

Bendegúz Nagy

This module maintains an ordered list of timed events, which consists of Timer_callBack function pointers, their state pointers. Also each one is assigned a Timer object.

Initialize and deinitialize the module with TM_init() and TM_deinit() respectively. Register Timed_events with TM_↩ new(), free them with TM_freeTimed_event(). Note: A Timed_event is destroyed automatically, if during a callback it returns a non-zero constant. Call TM_process() with the elapsed time in ms, which calls each registered Timed↩ _event with the elapsed time. Destroy every Timed_event with TM_clear().

6.8.2 Typedef Documentation

6.8.2.1 typedef struct Timed_event Timed_event

6.8.2.2 typedef int(∗ Timer_callBack)(Uint32 delta, Timer ∗timer, void ∗state)

Timed_event function pointers have to adhere to this signature.

Parameters

|>p0.15|p0.805|

delta	Elapsed time.
timer	The event's own timer.
state	The state pointer set during subscription.

Returns

non-zero if the Timed_event should be deleted.

6.8.3 Function Documentation

6.8.3.1 void TM_clear ( )

Destroy each Timed_event.

6.8.3.2 void TM_deinit ( )

Deinitialize the module.

6.8.3.3 void TM_freeTimed_event (Timed_event ∗ *e*)

Deallocate a Timed_event, this can also be done automatically.

Parameters

|>p0.15|p0.805|

e	The event to be destroyed.

Deallocate a Timed_event, note that if during a TM_process() an event's callback function returns non-zero, it will be automatically deleted.

6.8.3.4 void TM_init ( )

Initialize the module.

Initialize the module, this has to be called before th module is put to use. Calling this more than once without calling TM_deinit() in between will cause memory leak.

6.8.3.5 Timed_event∗ TM_new (Timer_callBack *callBack,* void ∗ *state*)

Create a new timed event and register it.

Parameters

|>p0.15|p0.805|

callBack The callback function.

state The state function which will be passed to the function with each function.

Returns

Returns the newly allocated Timed_event.

6.8.3.6 void TM_process (Uint32 *delta*)

Process the Timed_events with the elapsed time passed in ms.

Parameters

|>p0.15|p0.805|

delta Elapsed time since the last call in ms.

6.9 /home/bendeguz/ClionProjects/Dummy/Events/SRC/input.c File Reference

```
#include <SDL_events.h>
#include "../HEAD/input.h"
#include "../../Utility/HEAD/bag.h"
```

Data Structures

- struct Subscriber

  Internal representation of subscribed functions.

Typedefs

- typedef struct Subscriber Subscriber

  Internal representation of subscribed functions.

Functions

- void Input_init ()

  Initializes the module.
- void Input_deinit ()

  Deinitializes the module.
- void Input_clear ()

  Resets the list of subscribed functions.
- void Input_subscribe (inputConsumer cons, void ∗state)

  Subscribe functions and their state pointers with this.
- void Input_process ()

  Process accumulated SDL_Events by pushing them down the inputConsumer queue.

6.9.1 Typedef Documentation

6.9.1.1 typedef struct Subscriber Subscriber

Internal representation of subscribed functions.

6.9.2 Function Documentation

6.9.2.1 void Input_clear ( )

Resets the list of subscribed functions.

6.9.2.2 void Input_deinit ( )

Deinitializes the module.

6.9.2.3 void Input_init ( )

Initializes the module.

Initializes the module. It is mandatory to call this before the module is put to use in any way. Calling this more than once without calling Input_deinit() in between will cause a memory leak.

6.9.2.4 void Input_process ( )

Process accumulated SDL_Events by pushing them down the inputConsumer queue.

6.9.2.5 void Input_subscribe (inputConsumer *cons*, void ∗ *state*)

Subscribe functions and their state pointers with this.

Parameters

|>p0.15|p0.805|

cons	Function pointer to the inputConsumer function.
state	Optional state pointer to be passed to the function with each call.

Subscribe functions and their state pointers with this. The processing will be done in the order the functions are registered.

6.10 /home/bendeguz/ClionProjects/Dummy/Events/SRC/timer.c File Reference

```
#include <stdlib.h>
#include "../HEAD/timer.h"
```

Data Structures

- struct Timer

  Internal representation of time-keeping objects.

Functions

- Uint32 getDelta ()

    Returns the time between now and the last time this function was called.
- Timer ∗ Timer_new ()

    Creates a paused Timer object with zero ticks.
- void Timer_free (Timer ∗ptr)

    Deallocates the memory allocated by Timer_new()
- void Timer_start (Timer ∗ptr)

    Sets the isStarted flag of the Timer object to true.
- void Timer_updateDelta (Uint32 delta, Timer ∗ptr)

    Updates a Timer with a given number of ms.
- Uint32 Timer_getTicks (Timer ∗ptr)

    Returns the number of ms held in the Timer object.

6.10.1 Function Documentation

6.10.1.1 Uint32 getDelta ( )

Returns the time between now and the last time this function was called.

Returns

The time between now and the last time this function was called.

6.10.1.2 void Timer_free (Timer ∗ *ptr*)

Deallocates the memory allocated by Timer_new()

Parameters

|>p0.15|p0.805|

ptr The Timer object to be destroyed.

6.10.1.3 Uint32 Timer_getTicks (Timer ∗ *ptr*)

Returns the number of ms held in the Timer object.

Parameters

|>p0.15|p0.805|

ptr The Timer from which the elapsed time should be extracted.

Returns

Time in ms held by this Timer object.

6.10.1.4 Timer∗ Timer_new ( )

Creates a paused Timer object with zero ticks.

Returns

Returns the newly allocated Timer object.

6.10.1.5 void Timer_start (Timer ∗ *ptr*)

Sets the isStarted flag of the Timer object to true.

Parameters

|>p0.15|p0.805|

ptr The timer object to start.

Sets the isStarted flag of the Timer object to true. It is necessary to call this for a Timer objects, otherwise it won't accept passed time updates.

6.10.1.6 void Timer_updateDelta (Uint32 *delta,* Timer ∗ *ptr*)

Updates a Timer with a given number of ms.

Parameters

|>p0.15|p0.805|

delta The amount of time passed.

ptr The Timer to be updated.

Updates a Timer with a given number of ms. The Timer has to be started for this to take any effect

6.11 /home/bendeguz/ClionProjects/Dummy/Events/SRC/Timer_man.c File Reference

```
#include "../HEAD/Timer_man.h"
#include "../../Utility/HEAD/bag.h"
```

Functions

- Timed_event ∗ TM_new (Timer_callBack callBack, void ∗state)

  Create a new timed event and register it.
- void TM_freeTimed_event (Timed_event ∗e)

  Deallocate a Timed_event, this can also be done automatically.
- void TM_init ()

  Initialize the module.
- void TM_deinit ()

  Deinitialize the module.
- void TM_clear ()

  Destroy each Timed_event.
- void TM_process (Uint32 delta)

  Process the Timed_events with the elapsed time passed in ms.

6.11.1 Function Documentation

6.11.1.1 void TM_clear ( )

Destroy each Timed_event.

6.11.1.2 void TM_deinit ( )

Deinitialize the module.

6.11.1.3 void TM_freeTimed_event (Timed_event ∗ *e*)

Deallocate a Timed_event, this can also be done automatically.

Parameters

|>p0.15|p0.805|

e The event to be destroyed.

Deallocate a Timed_event, note that if during a TM_process() an event's callback function returns non-zero, it will be automatically deleted.

6.11.1.4 void TM_init ( )

Initialize the module.

Initialize the module, this has to be called before th module is put to use. Calling this more than once without calling TM_deinit() in between will cause memory leak.

6.11.1.5 Timed_event∗ TM_new (Timer_callBack *callBack,* void ∗ *state*)

Create a new timed event and register it.

Parameters

|>p0.15|p0.805|

callBack The callback function.

state The state function which will be passed to the function with each function.

Returns

Returns the newly allocated Timed_event.

6.11.1.6 void TM_process (Uint32 *delta*)

Process the Timed_events with the elapsed time passed in ms.

Parameters

|>p0.15|p0.805|

delta Elapsed time since the last call in ms.

6.12 /home/bendeguz/ClionProjects/Dummy/Game/HEAD/GameState.h File Reference

Actual game state and it's defining functions.

```
#include <stdint.h>
```

Functions

- int Game_start ()
- void Game_func (uint32_t delta)
- int Game_end ()

6.12.1 Detailed Description

Actual game state and it's defining functions.

Author

Bendegúz Nagy

6.12.2 Function Documentation

6.12.2.1 int Game_end ( )

6.12.2.2 void Game_func (uint32_t *delta*)

6.12.2.3 int Game_start ( )

6.13 /home/bendeguz/ClionProjects/Dummy/Game/HEAD/LevelSelState.h File Reference

Level select menu state and it's defining functions.

```
#include <stdint.h>
```

Functions

- int LevelSel_start ()
- void LevelSel_func (uint32_t delta)
- int LevelSel_end ()

6.13.1 Detailed Description

Level select menu state and it's defining functions.

Author

Bendegúz Nagy

This is almost exact copy of the MenuState state. For information on how this works, check that out.

6.13.2 Function Documentation

6.13.2.1 int LevelSel_end ( )

6.13.2.2 void LevelSel_func (uint32_t *delta*)

6.13.2.3 int LevelSel_start ( )

6.14 /home/bendeguz/ClionProjects/Dummy/Game/HEAD/main.h File Reference

Main holding together the whole program.

```
#include <stdint.h>
```

Typedefs

- typedef enum GlobalState GlobalState

  Enumeration of possible states.
- typedef int(∗ StateStart)(void)

  State init function signature.
- typedef void(∗ StateFunc)(uint32_t delta)

  State flow function signature.
- typedef int(∗ StateEnd)(void)

  State deinit function signature.

Enumerations

- enum GlobalState { MAIN_MENU, LEVEL_SELECT_MENU, GAME, GLOBAL_STATE_TOTAL }

  Enumeration of possible states.

Functions

- void SwapGlobalState (GlobalState stateTo)
- void Main_setExitFlag ()

Variables

- char ∗ currMapPath

  Ad hoc solution for selecting a map, this shouldn't be here.

6.14.1 Detailed Description

Main holding together the whole program.

Author

Bendegúz Nagy

6.14.2 Typedef Documentation

6.14.2.1 typedef enum GlobalState GlobalState

Enumeration of possible states.

6.14.2.2 typedef int(∗ StateEnd)(void)

State deinit function signature.

6.14.2.3 typedef void(∗ StateFunc)(uint32_t delta)

State flow function signature.

6.14.2.4 typedef int(∗ StateStart)(void)

State init function signature.

6.14.3 Enumeration Type Documentation

6.14.3.1 enum GlobalState

Enumeration of possible states.

Enumerator

MAIN_MENU

LEVEL_SELECT_MENU

GAME

GLOBAL_STATE_TOTAL

6.14.4 Function Documentation

6.14.4.1 void Main_setExitFlag ( )

6.14.4.2 void SwapGlobalState (GlobalState *stateTo*)

6.14.5 Variable Documentation

6.14.5.1 char∗ currMapPath

Ad hoc solution for selecting a map, this shouldn't be here.

6.15 /home/bendeguz/ClionProjects/Dummy/Game/HEAD/MenuState.h File Reference

Main menu state and it's defining functions.

```
#include <stdint.h>
```

Functions

- int Menu_start (void)
- void Menu_func (uint32_t delta)
- int Menu_end (void)

6.15.1 Detailed Description

Main menu state and it's defining functions.

Author

Bendegúz Nagy

6.15.2 Function Documentation

6.15.2.1 int Menu_end (void)

6.15.2.2 void Menu_func (uint32_t *delta*)

6.15.2.3 int Menu_start (void)

6.16 /home/bendeguz/ClionProjects/Dummy/Game/HEAD/player.h File Reference

Module for handling player related stuff.

```
#include "../../Collision/HEAD/physics.h"
#include "../HEAD/player.h"
```

Data Structures

- struct AttackData

  Each player has one of these, hold data for the attacking state.
- struct DashData

  Each player has one of these, hold data for the dashing state.
- struct ShootData

  Each player has one of these, hold data for the shooting state.
- struct Player

  Holds every data defining a player.

Typedefs

- typedef struct Player Player

  Holds every data defining a player.
- typedef enum PLAYER_KEYFLAGS PLAYER_KEYFLAGS

  Enumeration of not continuous player keys, stored in an int by OR-ing together.
- typedef enum PLAYER_CONTKEYFLAGS PLAYER_CONTKEYFLAGS

  Enumeration of continuous player keys, stored in an int by OR-ing together.
- typedef enum PLAYER_FLAGS PLAYER_FLAGS

  Player state flags, stored in an int by OR-ing together.
- typedef enum PLAYER_STATE PLAYER_STATE

  The player can only be a single state, namely one of these.
- typedef enum PLAYER_MOV_STATE PLAYER_MOV_STATE

  The player can only be in one of these movement states.
- typedef void(∗ stateFunc)(Player ∗p)

  Movement and logic state function must adhere to this function.
- typedef struct AttackData AttackData

  Each player has one of these, hold data for the attacking state.
- typedef struct DashData DashData

  Each player has one of these, hold data for the dashing state.
- typedef struct ShootData ShootData

  Each player has one of these, hold data for the shooting state.

Enumerations

- enum PLAYER_KEYFLAGS { ATT_KEY = 1, JUMP_KEY = 2, DASH_KEY = 4, SHOOT_KEY = 8 }

  Enumeration of not continuous player keys, stored in an int by OR-ing together.
- enum PLAYER_CONTKEYFLAGS { MOV_UP = 1, MOV_DOWN = 2, MOV_LEFT = 4, MOV_RIGHT = 8 }

  Enumeration of continuous player keys, stored in an int by OR-ing together.
- enum PLAYER_FLAGS { ON_THE_GROUND = 1, DAMAGED = 2, STATE_INIT = 4, MOV_STATE_INIT = 8 }

  Player state flags, stored in an int by OR-ing together.
- enum PLAYER_STATE {
  STILL = 0, WALKING = 1, GOING_UP = 2, GOING_DOWN = 3,
  ATTACKING = 4, DEAD = 5, DASHING = 6, SHOOTING = 7 }

  The player can only be a single state, namely one of these.
- enum PLAYER_MOV_STATE { NO_MOV = 0, FLY = 1, GROUND = 2 }

  The player can only be in one of these movement states.

Functions

- void Player_initModule ()

  This has to be called before the player module is put to use. Calling this multiple times without calling Player_deinit↩ Module() in between will cause a memory leak.
- void Player_deinitModule ()

  Deinitializes the player module.
- Player ∗ Player_new (int x, int y, World ∗world)

  Create a new player with no color or controlling keys.
- void Player_free (Player ∗player)

  Deallocates a player.
- void Player_reset (Player ∗p)

Resets a player, can be used like after a respawn.
- int Player_feedInput (SDL_Event ∗e, Player ∗p)

  Input consumer for a player.
- void Player_update (Player ∗p, Uint32 delta)

  Updates a player, does like calling the state and movement function.
- void Player_postRender (Uint32 delta)

  Call this function after rendering has been compelted.
- void Player_setState (PLAYER_STATE state, Player ∗p)

  Use this for swapping a player's state.
- int Player_compState (PLAYER_STATE state, Player ∗p)

  Use this to compare the player's state to a state enum.
- void Player_setMovState (PLAYER_MOV_STATE state, Player ∗p)

  Use this for setting the movement state.
- void Player_setControl (SDL_Keycode up, SDL_Keycode down, SDL_Keycode left, SDL_Keycode right, S↩ DL_Keycode attack, SDL_Keycode jump, SDL_Keycode dash, SDL_Keycode shoot, Player ∗player)

  Fill's the player's arrays with the SDL_Keycodes.

6.16.1 Detailed Description

Module for handling player related stuff.

Author

Bendegúz Nagy

The player module uses two concurrent function table FSMs. One for movement and one for logic. State init is done with the STATE_INIT flag in state functions. Movement state init is done with MOV_STATE_INIT flag in the movement state functions.

Each state function follows the same pattern. Check the STATE_INIT flag, do some logic, check if transition should occur to another state.

6.16.2 Typedef Documentation

6.16.2.1 typedef struct AttackData AttackData

Each player has one of these, hold data for the attacking state.

6.16.2.2 typedef struct DashData DashData

Each player has one of these, hold data for the dashing state.

6.16.2.3 typedef struct Player Player

Holds every data defining a player.

6.16.2.4 typedef enum PLAYER_CONTKEYFLAGS PLAYER_CONTKEYFLAGS

Enumeration of continuous player keys, stored in an int by OR-ing together.

6.16.2.5 typedef enum PLAYER_FLAGS PLAYER_FLAGS

Player state flags, stored in an int by OR-ing together.

6.16.2.6 typedef enum PLAYER_KEYFLAGS PLAYER_KEYFLAGS

Enumeration of not continuous player keys, stored in an int by OR-ing together.

6.16.2.7 typedef enum PLAYER_MOV_STATE PLAYER_MOV_STATE

The player can only be in one of these movement states.

6.16.2.8 typedef enum PLAYER_STATE PLAYER_STATE

The player can only be a single state, namely one of these.

6.16.2.9 typedef struct ShootData ShootData

Each player has one of these, hold data for the shooting state.

6.16.2.10 typedef void(∗ stateFunc)(Player ∗p)

Movement and logic state function must adhere to this function.

6.16.3 Enumeration Type Documentation

6.16.3.1 enum PLAYER_CONTKEYFLAGS

Enumeration of continuous player keys, stored in an int by OR-ing together.

Enumerator

MOV_UP

MOV_DOWN

MOV_LEFT

MOV_RIGHT

6.16.3.2 enum PLAYER_FLAGS

Player state flags, stored in an int by OR-ing together.

Enumerator

ON_THE_GROUND

DAMAGED

STATE_INIT

MOV_STATE_INIT

6.16.3.3 enum PLAYER_KEYFLAGS

Enumeration of not continuous player keys, stored in an int by OR-ing together.

Enumerator

ATT_KEY

JUMP_KEY

DASH_KEY

SHOOT_KEY

6.16.3.4 enum PLAYER_MOV_STATE

The player can only be in one of these movement states.

Enumerator

NO_MOV

FLY

GROUND

6.16.3.5 enum PLAYER_STATE

The player can only be a single state, namely one of these.

Enumerator

STILL

WALKING

GOING_UP

GOING_DOWN

ATTACKING

DEAD

DASHING

SHOOTING

6.16.4 Function Documentation

6.16.4.1 int Player_compState (PLAYER_STATE *state*, Player ∗ *p*)

Use this to compare the player's state to a state enum.

6.16.4.2 void Player_deinitModule ( )

Deinitializes the player module.

6.16.4.3 int Player_feedInput (SDL_Event ∗ *e*, Player ∗ *p*)

Input consumer for a player.

6.16.4.4 void Player_free (Player ∗ *player*)

Deallocates a player.

6.16.4.5 void Player_initModule ( )

This has to be called before the player module is put to use. Calling this multiple times without calling Player_↩ deinitModule() in between will cause a memory leak.

6.16.4.6 Player∗ Player_new (int *x*, int *y*, World ∗ *world*)

Create a new player with no color or controlling keys.

6.16.4.7 void Player_postRender (Uint32 *delta*)

Call this function after rendering has been compelted.

6.16.4.8 void Player_reset (Player ∗ *p*)

Resets a player, can be used like after a respawn.

6.16.4.9 void Player_setControl (SDL_Keycode *up*, SDL_Keycode *down*, SDL_Keycode *left*, SDL_Keycode *right*, SDL_Keycode *attack*, SDL_Keycode *jump*, SDL_Keycode *dash*, SDL_Keycode *shoot*, Player ∗ *player*)

Fill's the player's arrays with the SDL_Keycodes.

6.16.4.10 void Player_setMovState (PLAYER_MOV_STATE *state*, Player ∗ *p*)

Use this for setting the movement state.

6.16.4.11 void Player_setState (PLAYER_STATE *state*, Player ∗ *p*)

Use this for swapping a player's state.

6.16.4.12 void Player_update (Player ∗ *p*, Uint32 *delta*)

Updates a player, does like calling the state and movement function.

6.17 /home/bendeguz/ClionProjects/Dummy/Game/SRC/GameState.c File Reference

```
#include "../HEAD/GameState.h"
#include "../../Collision/HEAD/physics.h"
#include "../HEAD/player.h"
#include "../HEAD/main.h"
#include "../../Events/HEAD/input.h"
#include "../../Events/HEAD/timer.h"
#include "../../Events/HEAD/Timer_man.h"
#include "../../Graphics/HEAD/graphics_man.h"
#include "../../Graphics/HEAD/textsprite.h"
```

Macros

- #define PLAYER_COUNT 2
- #define GRAVITY -1700
- #define RESPAWN_TIME 1000
- #define WIN_SCORE 5

Functions

- int Game_respawn (uint32_t delta, Timer ∗timer, Player ∗p)

  Respawns a player.
- int Game_escapeInputProc (SDL_Event ∗e, void ∗null)

  Input consumer used at the end of a game to process the ESC key.
- int Game_start ()
- void Game_func (uint32_t delta)
- int Game_end ()

Variables

- World ∗ world

  The physics world singleton used for the game.
- Player ∗ players [PLAYER_COUNT]

  Array holding player objects, currently hardcoded for 2.
- Bag ∗ spawnPos = NULL

  Bag holding 2D vectors of possible spawn positions, read from map file.
- int Game_paused

  Flag for "is game paused?" question.
- SDL_Texture ∗ youreWinner = NULL

  End game picture.
- SDL_Rect youreWinnerRect = {490, 200, 221, 237}

  Rectangle for rendering end game picture.
- TextSprite ∗ winText = NULL

  Text object to be displayed at the end of a game.

6.17.1 Macro Definition Documentation

6.17.1.1 #define GRAVITY -1700

6.17.1.2 #define PLAYER_COUNT 2

6.17.1.3 #define RESPAWN_TIME 1000

6.17.1.4 #define WIN_SCORE 5

6.17.2 Function Documentation

6.17.2.1 int Game_end ( )

6.17.2.2 int Game_escapeInputProc (SDL_Event ∗ *e*, void ∗ *null*)

Input consumer used at the end of a game to process the ESC key.

6.17.2.3 void Game_func (uint32_t *delta*)

6.17.2.4 int Game_respawn (uint32_t *delta,* Timer ∗ *timer,* Player ∗ *p*)

Respawns a player.

6.17.2.5 int Game_start ( )

6.17.3 Variable Documentation

6.17.3.1 int Game_paused

Flag for "is game paused?" question.

6.17.3.2 Player∗ players[PLAYER_COUNT]

Array holding player objects, currently hardcoded for 2.

6.17.3.3 Bag∗ spawnPos = NULL

Bag holding 2D vectors of possible spawn positions, read from map file.

6.17.3.4 TextSprite∗ winText = NULL

Text object to be displayed at the end of a game.

6.17.3.5 World∗ world

The physics world singleton used for the game.

6.17.3.6 SDL_Texture∗ youreWinner = NULL

End game picture.

6.17.3.7 SDL_Rect youreWinnerRect = {490, 200, 221, 237}

Rectangle for rendering end game picture.

6.18 /home/bendeguz/ClionProjects/Dummy/Game/SRC/LevelSelState.c File Reference

```
#include "../HEAD/LevelSelState.h"
#include "../../Graphics/HEAD/textsprite.h"
#include "../../Graphics/HEAD/graphics_man.h"
#include "../../Events/HEAD/input.h"
#include "../HEAD/main.h"
```

Data Structures

- struct LevelSel_data

Typedefs

- typedef enum LevelSel_Options LevelSel_Options
- typedef struct LevelSel_data LevelSel_data

Enumerations

- enum LevelSel_Options {
  LOWER_BOUND = -1, MAP1, MAP2, MAP3,
  MAP4, MAP5, UPPER_BOUND }

Functions

- int LevelSelInputProc (SDL_Event ∗e, void ∗null)
- int LevelSel_start ()
- void LevelSel_func (uint32_t delta)
- int LevelSel_end ()

Variables

- TextSprite ∗ LevelSel_selected [UPPER_BOUND]
- TextSprite ∗ LevelSel_unselected [UPPER_BOUND]
- SDL_Texture ∗ levelSelBackground = NULL
- LevelSel_data levelSel

6.18.1 Typedef Documentation

6.18.1.1 typedef struct LevelSel_data LevelSel_data

6.18.1.2 typedef enum LevelSel_Options LevelSel_Options

6.18.2 Enumeration Type Documentation

6.18.2.1 enum LevelSel_Options

Enumerator

LOWER_BOUND

MAP1

MAP2

MAP3

MAP4

MAP5

UPPER_BOUND

6.18.3 Function Documentation

6.18.3.1 int LevelSel_end ( )

6.18.3.2 void LevelSel_func (uint32_t *delta*)

6.18.3.3 int LevelSel_start ( )

6.18.3.4 int LevelSelInputProc (SDL_Event ∗ *e,* void ∗ *null*)

6.18.4 Variable Documentation

6.18.4.1 LevelSel_data levelSel

6.18.4.2 TextSprite∗ LevelSel_selected[UPPER_BOUND]

6.18.4.3 TextSprite∗ LevelSel_unselected[UPPER_BOUND]

6.18.4.4 SDL_Texture∗ levelSelBackground = NULL

6.19 /home/bendeguz/ClionProjects/Dummy/Game/SRC/main.c File Reference

```
#include <stdio.h>
#include <SDL_image.h>
#include <time.h>
#include "../HEAD/main.h"
#include "../HEAD/MenuState.h"
#include "../../Graphics/HEAD/graphics_man.h"
#include "../../Events/HEAD/timer.h"
#include "../../Events/HEAD/Timer_man.h"
#include "../../Events/HEAD/input.h"
#include "../HEAD/LevelSelState.h"
#include "../HEAD/GameState.h"
#include "../../Graphics/HEAD/textsprite.h"
```

Data Structures

- struct MAIN_DATA

  Main specific data.

Typedefs

- typedef struct MAIN_DATA MAIN_DATA

  Main specific data.

Functions

- int Main_init ()
- void Main_deinit ()
- int Main_quitInputCB (SDL_Event ∗e, void ∗null)

  Input consumer for Window X clicks.

- int main (int argc, char ∗args[])
- void SwapGlobalState (GlobalState stateTo)
- void Main_setExitFlag ()

Variables

- MAIN_DATA mData

  Singleton for main specific data.
- StateStart stStart [GLOBAL_STATE_TOTAL]

  Table holding the init functions.
- StateFunc stFunc [GLOBAL_STATE_TOTAL]

  Table holding the flow functions.
- StateEnd stEnd [GLOBAL_STATE_TOTAL]

  Table holding the deinit functions.
- GlobalState currState = MAIN_MENU

  Current state.
- char ∗ currMapPath = "res/maps/map1.dat"

  Ad hoc solution for selecting a map, this shouldn't be here.

6.19.1 Typedef Documentation

6.19.1.1 typedef struct MAIN_DATA MAIN_DATA

Main specific data.

6.19.2 Function Documentation

6.19.2.1 int main (int *argc*, char ∗ *args[]*)

6.19.2.2 void Main_deinit ( )

6.19.2.3 int Main_init ( )

6.19.2.4 int Main_quitInputCB (SDL_Event ∗ *e*, void ∗ *null*)

Input consumer for Window X clicks.

6.19.2.5 void Main_setExitFlag ( )

6.19.2.6 void SwapGlobalState (GlobalState *stateTo*)

6.19.3 Variable Documentation

6.19.3.1 char∗ currMapPath = "res/maps/map1.dat"

Ad hoc solution for selecting a map, this shouldn't be here.

6.19.3.2 GlobalState currState = MAIN_MENU

Current state.

6.19.3.3 MAIN_DATA mData

Singleton for main specific data.

6.19.3.4 StateEnd stEnd[GLOBAL_STATE_TOTAL]

Table holding the deinit functions.

6.19.3.5 StateFunc stFunc[GLOBAL_STATE_TOTAL]

Table holding the flow functions.

6.19.3.6 StateStart stStart[GLOBAL_STATE_TOTAL]

Table holding the init functions.

6.20 /home/bendeguz/ClionProjects/Dummy/Game/SRC/MenuState.c File Reference

```
#include <SDL_render.h>
#include "../HEAD/MenuState.h"
#include "../../Graphics/HEAD/graphics_man.h"
#include "../../Graphics/HEAD/textsprite.h"
#include "../../Events/HEAD/input.h"
#include "../HEAD/main.h"
```

Data Structures

- struct Menu_data

  Main menu specific data.

Typedefs

- typedef enum Menu_options Menu_options

  Possible menu options.
- typedef struct Menu_data Menu_data

  Main menu specific data.

Enumerations

- enum Menu_options {
  LOWER_BOUND = -1, START_GAME, SELECT_LEVEL, EXIT,
  UPPER_BOUND }

  Possible menu options.

Functions

- int MenuInputProc (SDL_Event ∗e, void ∗null)

  Input consumer for the menu, manages up, down arrows and the enter key.
- int Menu_start (void)
- void Menu_func (uint32_t delta)
- int Menu_end (void)

Variables

- TextSprite ∗ Menu_selected [UPPER_BOUND]

  Array holding text objects of menu options in selected colour.
- TextSprite ∗ Menu_unselected [UPPER_BOUND]
- SDL_Texture ∗ menuBackground = NULL

  Menu background picture.
- Menu_data menuData

  Singleton for Menu specific data.

6.20.1 Typedef Documentation

6.20.1.1 typedef struct Menu_data Menu_data

Main menu specific data.

6.20.1.2 typedef enum Menu_options Menu_options

Possible menu options.

6.20.2 Enumeration Type Documentation

6.20.2.1 enum Menu_options

Possible menu options.

Enumerator

LOWER_BOUND	Used to identify loop around in menu select.
START_GAME	
SELECT_LEVEL	
EXIT	
UPPER_BOUND	Used to identify loop around in menu select.

6.20.3 Function Documentation

6.20.3.1 int Menu_end (void)

6.20.3.2 void Menu_func (uint32_t *delta*)

6.20.3.3 int Menu_start (void)

6.20.3.4 int MenuInputProc (SDL_Event ∗ *e,* void ∗ *null*)

Input consumer for the menu, manages up, down arrows and the enter key.

6.20.4 Variable Documentation

6.20.4.1 TextSprite∗ Menu_selected[UPPER_BOUND]

Array holding text objects of menu options in selected colour.

6.20.4.2 TextSprite∗ Menu_unselected[UPPER_BOUND]

Array holding text objects of menu options in unselected colour.

6.20.4.3 SDL_Texture∗ menuBackground = NULL

Menu background picture.

6.20.4.4 Menu_data menuData

Singleton for Menu specific data.

6.21 /home/bendeguz/ClionProjects/Dummy/Game/SRC/player.c File Reference

```
#include "../HEAD/player.h"
#include "../../Events/HEAD/Timer_man.h"
```

Macros

- #define XCAP 350
- #define YCAP 1000
- #define WALK_FORCE 2100
- #define JUMP_SPEED 700
- #define FLY_FORCE 1600
- #define SLIDE_MAX 85
- #define ATTACK_DURR 150
- #define ATTACK_CD 500
- #define CLING_IMPULSE 700
- #define DASH_CD 1000
- #define DASH_SPEED 1500
- #define DASH_DURR 50
- #define SHOOT_CD 300
- #define BULLET_SPEED 1000
- #define SHOOT_COUNT 3000

Functions

- void Player_initModule ()

  This has to be called before the player module is put to use. Calling this multiple times without calling Player_deinit↩ Module() in between will cause a memory leak.
- void Player_deinitModule ()

  Deinitializes the player module.

- int Player_attackRemoveTimer (Uint32 delta, Timer ∗timer, Player ∗p)

  Defines a timer callback function for pulling the player out of the attacking state and destroying it's attackbox.
- int Player_attackBoxColl (PH_Manifold ∗m, Object ∗callObj, Object ∗collObj, Player ∗p)

  PH_callback for the attackbox collisions.
- int Player_collCallBack (PH_Manifold ∗m, Object ∗A, Object ∗B, Player ∗player)

  PH_callback for player collisions.
- int Player_dashTimer (Uint32 delta, Timer ∗timer, Player ∗p)

  Defines a timer callback function for pulling the player out of the dashing state.
- int Player_bulletCB (PH_Manifold ∗m, Object ∗A, Object ∗B, Player ∗p)

  PH_callback for bullets.
- void Player_still (Player ∗p)

  When the player is on the ground and is not moving.
- void Player_walking (Player ∗p)

  When the player is on the ground and is walking.
- void Player_goingUp (Player ∗p)

  When the player is in the air and is going upwards.
- void Player_goingDown (Player ∗p)

  When the player is in the air and is going downwards.
- void Player_attack (Player ∗p)

  When the player is spawning an attackbox.
- void Player_dead (Player ∗p)

  When the player is dead.
- void Player_dash (Player ∗p)

  When the player is currently dashing.
- void Player_shoot (Player ∗p)

  When the player is shooting a bulett.
- void Player_groundMov (Player ∗p)

  Movement when the player is on the ground.
- void Player_flyMov (Player ∗p)

  Movement state, when the player is in the air.
- Player ∗ Player_new (int x, int y, World ∗world)

  Create a new player with no color or controlling keys.
- void Player_reset (Player ∗p)

  Resets a player, can be used like after a respawn.
- void Player_setState (PLAYER_STATE state, Player ∗p)

  Use this for swapping a player's state.
- int Player_compState (PLAYER_STATE state, Player ∗p)

  Use this to compare the player's state to a state enum.
- void Player_setMovState (PLAYER_MOV_STATE state, Player ∗p)

  Use this for setting the movement state.
- void Player_setControl (SDL_Keycode up, SDL_Keycode down, SDL_Keycode left, SDL_Keycode right, S↩ DL_Keycode attack, SDL_Keycode jump, SDL_Keycode dash, SDL_Keycode shoot, Player ∗player)

  Fill's the player's arrays with the SDL_Keycodes.
- void Player_free (Player ∗player)

  Deallocates a player.
- int Player_feedInput (SDL_Event ∗e, Player ∗p)

  Input consumer for a player.
- void Player_postRender (Uint32 delta)

  Call this function after rendering has been compelted.
- void Player_update (Player ∗p, Uint32 delta)

  Updates a player, does like calling the state and movement function.

Variables

- Bag ∗ queryBag = NULL

  Helper bag fro querying stuff, do not assume it's contents will remain the same in between function calls.
- Bag ∗ destroyBag = NULL

  Only push and only objects onto this. Holds PH_Objects which could not be deleted during a callback.
- const stateFunc states [8]

  Array holding the state functions.
- const stateFunc movStates [3]

  Array for holding movement state functions.

6.21.1 Macro Definition Documentation

6.21.1.1 #define ATTACK_CD 500

6.21.1.2 #define ATTACK_DURR 150

6.21.1.3 #define BULLET_SPEED 1000

6.21.1.4 #define CLING_IMPULSE 700

6.21.1.5 #define DASH_CD 1000

6.21.1.6 #define DASH_DURR 50

6.21.1.7 #define DASH_SPEED 1500

6.21.1.8 #define FLY_FORCE 1600

6.21.1.9 #define JUMP_SPEED 700

6.21.1.10 #define SHOOT_CD 300

6.21.1.11 #define SHOOT_COUNT 3000

6.21.1.12 #define SLIDE_MAX 85

6.21.1.13 #define WALK_FORCE 2100

6.21.1.14 #define XCAP 350

6.21.1.15 #define YCAP 1000

6.21.2 Function Documentation

6.21.2.1 void Player_attack (Player ∗ *p*)

When the player is spawning an attackbox.

6.21.2.2 int Player_attackBoxColl (PH_Manifold ∗ *m,* Object ∗ *callObj,* Object ∗ *collObj,* Player ∗ *p*)

PH_callback for the attackbox collisions.

6.21.2.3 int Player_attackRemoveTimer (Uint32 *delta,* Timer ∗ *timer,* Player ∗ *p*)

Defines a timer callback function for pulling the player out of the attacking state and destroying it's attackbox.

6.21.2.4 int Player_bulletCB (PH_Manifold ∗ *m,* Object ∗ *A,* Object ∗ *B,* Player ∗ *p*)

PH_callback for bullets.

6.21.2.5 int Player_collCallBack (PH_Manifold ∗ *m,* Object ∗ *A,* Object ∗ *B,* Player ∗ *player*)

PH_callback for player collisions.

6.21.2.6 int Player_compState (PLAYER_STATE *state,* Player ∗ *p*)

Use this to compare the player's state to a state enum.

6.21.2.7 void Player_dash (Player ∗ *p*)

When the player is currently dashing.

6.21.2.8 int Player_dashTimer (Uint32 *delta,* Timer ∗ *timer,* Player ∗ *p*)

Defines a timer callback function for pulling the player out of the dashing state.

6.21.2.9 void Player_dead (Player ∗ *p*)

When the player is dead.

6.21.2.10 void Player_deinitModule ( )

Deinitializes the player module.

6.21.2.11 int Player_feedInput (SDL_Event ∗ *e,* Player ∗ *p*)

Input consumer for a player.

6.21.2.12 void Player_flyMov (Player ∗ *p*)

Movement state, when the player is in the air.

6.21.2.13 void Player_free (Player ∗ *player*)

Deallocates a player.

6.21.2.14 void Player_goingDown (Player ∗ *p*)

When the player is in the air and is going downwards.

6.21.2.15 void Player_goingUp (Player ∗ *p*)

When the player is in the air and is going upwards.

6.21.2.16 void Player_groundMov (Player ∗ *p*)

Movement when the player is on the ground.

6.21.2.17 void Player_initModule ( )

This has to be called before the player module is put to use. Calling this multiple times without calling Player_↩ deinitModule() in between will cause a memory leak.

6.21.2.18 Player∗ Player_new (int *x,* int *y,* World ∗ *world*)

Create a new player with no color or controlling keys.

6.21.2.19 void Player_postRender (Uint32 *delta*)

Call this function after rendering has been compelted.

6.21.2.20 void Player_reset (Player ∗ *p*)

Resets a player, can be used like after a respawn.

6.21.2.21 void Player_setControl (SDL_Keycode *up,* SDL_Keycode *down,* SDL_Keycode *left,* SDL_Keycode *right,* SDL_Keycode *attack,* SDL_Keycode *jump,* SDL_Keycode *dash,* SDL_Keycode *shoot,* Player ∗ *player*)

Fill's the player's arrays with the SDL_Keycodes.

6.21.2.22 void Player_setMovState (PLAYER_MOV_STATE *state,* Player ∗ *p*)

Use this for setting the movement state.

6.21.2.23 void Player_setState (PLAYER_STATE *state,* Player ∗ *p*)

Use this for swapping a player's state.

6.21.2.24 void Player_shoot (Player ∗ *p*)

When the player is shooting a bulett.

6.21.2.25 void Player_still (Player ∗ *p*)

When the player is on the ground and is not moving.

6.21.2.26 void Player_update (Player ∗ *p,* Uint32 *delta*)

Updates a player, does like calling the state and movement function.

6.21.2.27 void Player_walking (Player ∗ *p*)

When the player is on the ground and is walking.

6.21.3 Variable Documentation

6.21.3.1 Bag∗ destroyBag = NULL

Only push and only objects onto this. Holds PH_Objects which could not be deleted during a callback.

6.21.3.2 const stateFunc movStates[3]

Initial value:

```
= {
        NULL,
        &Player_flyMov,
        &Player_groundMov
}
```

Array for holding movement state functions.

6.21.3.3 Bag∗ queryBag = NULL

Helper bag fro querying stuff, do not assume it's contents will remain the same in between function calls.

6.21.3.4 const stateFunc states[8]

Initial value:

```
= {
    &Player_still,
    &Player_walking,
    &Player_goingUp,
    &Player_goingDown,
    &Player_attack,
    &Player_dead,
    &Player_dash,
    &Player_shoot
}
```

Array holding the state functions.

6.22 /home/bendeguz/ClionProjects/Dummy/Graphics/HEAD/graphics_↩ man.h File Reference

Handles the initialization of SDL, creation of the window and loading of png files.

```
#include "SDL2/SDL.h"
```

Functions

- int GM_init (int SDL_FlAGS, int IMG_FLAGS)

  Initializes the ttf, image and core sdl library, creates a window.
- void GM_deinit ()

Destroys everything GM_init() has initialized.

- SDL_Texture ∗ GM_loadPngFromFile (char ∗path, int r, int g, int b)

  Loads a png image from a file and creates an SDL_Texture from it.

Variables

- const int SCREEN_WIDTH

  Window width.
- const int SCREEN_HEIGHT

  Window height.
- SDL_Window ∗ gWindow

  Pointer to the single window this project has.
- SDL_Renderer ∗ gRenderer

  Pointer the single render object this project has.

6.22.1 Detailed Description

Handles the initialization of SDL, creation of the window and loading of png files.

Author

Bendegúz Nagy

Initialize SDL and create window with GM_init(), deinitialize with GM_deinit(). Load png files into SDL_Textures with GM_loadPngFromFile().

6.22.2 Function Documentation

6.22.2.1 void GM_deinit ( )

Destroys everything GM_init() has initialized.

6.22.2.2 int GM_init (int *SDL_FlAGS,* int *IMG_FLAGS*)

Initializes the ttf, image and core sdl library, creates a window.

Parameters

|>p0.15|p0.805|

SDL_FLAGS	the subsystems the SDL library should init.
IMG_FLAGS	the subsystems the IMG library should init.

6.22.2.3 SDL_Texture∗ GM_loadPngFromFile (char ∗ *path,* int *r,* int *g,* int *b*)

Loads a png image from a file and creates an SDL_Texture from it.

Takes 3 colour variables {r,g,b} to set the color keying of the SDL_texture. If either of them is negative, no colorkeying will be set.

6.22.3 Variable Documentation

6.22.3.1 SDL_Renderer∗ gRenderer

Pointer the single render object this project has.

6.22.3.2 SDL_Window∗ gWindow

Pointer to the single window this project has.

6.22.3.3 const int SCREEN_HEIGHT

Window height.

6.22.3.4 const int SCREEN_WIDTH

Window width.

6.23 /home/bendeguz/ClionProjects/Dummy/Graphics/HEAD/textsprite.h File Reference

Module for rendering text.

```
#include "SDL2/SDL.h"
#include "SDL2/SDL_ttf.h"
```

Typedefs

- typedef struct TextSprite TextSprite

Functions

- void TS_init (char ∗fontPath)

  Initializes the module.
- void TS_deinit ()

  Deinitializes the module.
- TextSprite ∗ TS_new ()

  Creates a new empty text object.
- void TS_free (TextSprite ∗ptr)

  Deallocates a text object allocated by TS_new().
- int TS_setText (char ∗text, SDL_Color ∗color, int size, TextSprite ∗ptr)

  Set the text of the text object with a given colour and size.
- void TS_setPos (int x, int y, TextSprite ∗ptr)

  Sets the position of a text object.
- void TS_render (TextSprite ∗ptr)

  Render the text held by te text object at the previously set position.
- int TS_getWidth (TextSprite ∗ptr)

  Get the width of the SDL_Texture held by this text object.
- int TS_getHeight (TextSprite ∗ptr)

  Get the height of the SDL_Texture held by this text object.

6.23.1 Detailed Description

Module for rendering text.

Author

Bendegúz Nagy

This module uses a single global font type and lazy initialization for requested text sizes. Initialize the module with TS_init(), deinit with TS_deinit(). Create new empty text objects with TS_new(), free them with TS_free(). Set their text any number of times with TS_setText(). Set their position with TS_setPos(), render them with TS_render().

6.23.2 Typedef Documentation

6.23.2.1 typedef struct TextSprite TextSprite

6.23.3 Function Documentation

6.23.3.1 void TS_deinit ( )

Deinitializes the module.

6.23.3.2 void TS_free (TextSprite ∗ *ptr*)

Deallocates a text object allocated by TS_new().

6.23.3.3 int TS_getHeight (TextSprite ∗ *ptr*)

Get the height of the SDL_Texture held by this text object.

6.23.3.4 int TS_getWidth (TextSprite ∗ *ptr*)

Get the width of the SDL_Texture held by this text object.

6.23.3.5 void TS_init (char ∗ *fontPath*)

Initializes the module.

Initializes the module, this has to be called before the module is put to use. Calling this more than once, without calling TS_deinit in between will cause a memory leak.

6.23.3.6 TextSprite∗ TS_new ( )

Creates a new empty text object.

Returns

the newly allocated text object.

6.23.3.7 void TS_render (TextSprite ∗ *ptr*)

Render the text held by te text object at the previously set position.

6.23.3.8 void TS_setPos (int *x*, int *y*, TextSprite ∗ *ptr*)

Sets the position of a text object.

6.23.3.9 int TS_setText (char ∗ *text*, SDL_Color ∗ *color*, int *size*, TextSprite ∗ *ptr*)

Set the text of the text object with a given colour and size.

This set the text of the text object by rendering an SDL_Texture.

6.24 /home/bendeguz/ClionProjects/Dummy/Graphics/SRC/graphics_↩ man.c File Reference

```
#include <SDL2/SDL_image.h>
#include "SDL2/SDL_ttf.h"
#include "../HEAD/textsprite.h"
```

Functions

- int GM_init (int SDL_FIAGS, int IMG_FLAGS)

  Initializes the ttf, image and core sdl library, creates a window.
- void GM_deinit ()

  Destroys everything GM_init() has initialized.
- SDL_Texture ∗ GM_loadPngFromFile (char ∗path, int r, int g, int b)

  Loads a png image from a file and creates an SDL_Texture from it.

Variables

- const int SCREEN_WIDTH = 1200

  Window width.
- const int SCREEN_HEIGHT = 700

  Window height.
- SDL_Window ∗ gWindow = NULL

  Pointer to the single window this project has.
- SDL_Renderer ∗ gRenderer = NULL

  Pointer the single render object this project has.

6.24.1 Function Documentation

6.24.1.1 void GM_deinit ( )

Destroys everything GM_init() has initialized.

6.24.1.2 int GM_init (int *SDL_FIAGS*, int *IMG_FLAGS*)

Initializes the ttf, image and core sdl library, creates a window.

Parameters

|>p0.15|p0.805|

SDL_FLAGS	the subsystems the SDL library should init.
IMG_FLAGS	the subsystems the IMG library should init.

6.24.1.3 SDL_Texture* GM_loadPngFromFile (char * *path,* int *r,* int *g,* int *b*)

Loads a png image from a file and creates an SDL_Texture from it.

Takes 3 colour variables {r,g,b} to set the color keying of the SDL_texture. If either of them is negative, no colorkeying will be set.

6.24.2 Variable Documentation

6.24.2.1 SDL_Renderer* gRenderer = NULL

Pointer the single render object this project has.

6.24.2.2 SDL_Window* gWindow = NULL

Pointer to the single window this project has.

6.24.2.3 const int SCREEN_HEIGHT = 700

Window height.

6.24.2.4 const int SCREEN_WIDTH = 1200

Window width.

6.25 /home/bendeguz/ClionProjects/Dummy/Graphics/SRC/textsprite.c File Reference

```
#include <stdlib.h>
#include "../HEAD/textsprite.h"
#include "../HEAD/graphics_man.h"
#include "../../Utility/HEAD/bag.h"
```

Data Structures

- struct Fonts

  Internal representation of the global font in a given size.
- struct TextSprite

  Internal representation of a text object.

Typedefs

- typedef struct Fonts Fonts

  Internal representation of the global font in a given size.

- typedef struct TextSprite TextSprite

    Internal representation of a text object.

Functions

- void TS_freeFont (Fonts ∗font)

    deallocates a Fonts type allocated by TS_new().
- void TS_init (char ∗fontPath)

    Initializes the module.
- void TS_deinit ()

    Deinitializes the module.
- TextSprite ∗ TS_new ()

    Creates a new empty text object.
- void TS_free (TextSprite ∗ptr)

    Deallocates a text object allocated by TS_new().
- int TS_setText (char ∗text, SDL_Color ∗color, int size, TextSprite ∗ptr)

    Set the text of the text object with a given colour and size.
- void TS_setPos (int x, int y, TextSprite ∗ptr)

    Sets the position of a text object.
- void TS_render (TextSprite ∗ptr)

    Render the text held by te text object at the previously set position.
- int TS_getWidth (TextSprite ∗ptr)

    Get the width of the SDL_Texture held by this text object.
- int TS_getHeight (TextSprite ∗ptr)

    Get the height of the SDL_Texture held by this text object.

Variables

- Bag ∗ fontsBag = NULL

    Holds lazily initialized Fonts objects.
- char ∗ fontsPath = NULL

    Holds the file path of the global font.

6.25.1 Typedef Documentation

6.25.1.1 typedef struct Fonts Fonts

Internal representation of the global font in a given size.

These are made by means of lazy initialization.

6.25.1.2 typedef struct TextSprite TextSprite

Internal representation of a text object.

6.25.2 Function Documentation

6.25.2.1 void TS_deinit ( )

Deinitializes the module.

6.25.2.2 void TS_free (TextSprite * *ptr*)

Deallocates a text object allocated by TS_new().

6.25.2.3 void TS_freeFont (Fonts * *font*)

deallocates a Fonts type allocated by TS_new().

6.25.2.4 int TS_getHeight (TextSprite * *ptr*)

Get the height of the SDL_Texture held by this text object.

6.25.2.5 int TS_getWidth (TextSprite * *ptr*)

Get the width of the SDL_Texture held by this text object.

6.25.2.6 void TS_init (char * *fontPath*)

Initializes the module.

Initializes the module, this has to be called before the module is put to use. Calling this more than once, without calling TS_deinit in between will cause a memory leak.

6.25.2.7 TextSprite* TS_new ( )

Creates a new empty text object.

Returns

the newly allocated text object.

6.25.2.8 void TS_render (TextSprite * *ptr*)

Render the text held by te text object at the previously set position.

6.25.2.9 void TS_setPos (int *x,* int *y,* TextSprite * *ptr*)

Sets the position of a text object.

6.25.2.10 int TS_setText (char * *text,* SDL_Color * *color,* int *size,* TextSprite * *ptr*)

Set the text of the text object with a given colour and size.

This set the text of the text object by rendering an SDL_Texture.

6.25.3 Variable Documentation

6.25.3.1 Bag* fontsBag = NULL

Holds lazily initialized Fonts objects.

6.25.3.2 char∗ fontsPath = NULL

Holds the file path of the global font.

6.26 /home/bendeguz/ClionProjects/Dummy/Utility/HEAD/bag.h File Reference

Dynamically growing array implementation.

Data Structures

- struct Bag

  Holds a dynamically growing array.

Typedefs

- typedef void(∗ freeData)(void ∗ptr)

  Bags can have a data freeing function which have to adhere to this signature.
- typedef struct Bag Bag

  Holds a dynamically growing array.

Functions

- Bag ∗ Bag_new (freeData freeDatPtr)

  Allocates a new Bag.
- void Bag_free (Bag ∗bag, int freeData)

  Deallocate a bag allocated by Bag_new().
- int Bag_push (void ∗data, Bag ∗bag)

  Push a new element to the end of the Bag.
- void ∗ Bag_unorderedRemove (int index, Bag ∗bag)

  Remove an element at an index, place the last element in it's place.
- int Bag_search (void ∗data, Bag ∗bag)

  Linear search by pointer equality.
- void Bag_slowClear (Bag ∗bag, int free)

  Clears the bag and overwrites the backing array with zeros.
- void Bag_fastClear (Bag ∗bag)

  Sets the element count to zero.

6.26.1 Detailed Description

Dynamically growing array implementation.

Author

Bendegúz Nagy

Create a new dynamic array with Bag_new(), add element with Bag_push(), remove with Bag_unordered() remove. Iterate by accessing the implementation. Search by pointer equality with Bag_search()

6.26.2 Typedef Documentation

6.26.2.1 typedef struct Bag Bag

Holds a dynamically growing array.

6.26.2.2 typedef void(∗ freeData)(void ∗ptr)

Bags can have a data freeing function which have to adhere to this signature.

6.26.3 Function Documentation

6.26.3.1 void Bag_fastClear (Bag ∗ *bag*)

Sets the element count to zero.

6.26.3.2 void Bag_free (struct Bag ∗ *bag,* int *freeDataPtr*)

Deallocate a bag allocated by Bag_new().

Parameters

|>p0.15|p0.805|

bag	the bag to be destroyed.
freeDataPtr	non-zero if the bag should call the registered free function for each contained data pointer.

6.26.3.3 Bag∗ Bag_new (freeData *freeDataPtr*)

Allocates a new Bag.

Parameters

|>p0.15|p0.805|

freeDataPtr	a function that will be used to free held data when deleting elements.

Returns

the newly allocated Bag.

6.26.3.4 int Bag_push (void ∗ *data,* struct Bag ∗ *bag*)

Push a new element to the end of the Bag.

Returns

the index at which the new data will be stored, can change when elements are deleted.

6.26.3.5 int Bag_search (void ∗ *data,* Bag ∗ *bag*)

Linear search by pointer equality.

Returns

the index at which the data is stored, -1 if it could not be found.

6.26.3.6 void Bag_slowClear (Bag ∗ *bag,* int *free*)

Clears the bag and overwrites the backing array with zeros.

Parameters

|>p0.15|p0.805|

free	if the registered free function should be called for each contained data pointer.

6.26.3.7 void∗ Bag_unorderedRemove (int *index,* Bag ∗ *bag*)

Remove an element at an index, place the last element in it's place.

Returns

the removed data pointer.

6.27 /home/bendeguz/ClionProjects/Dummy/Utility/HEAD/vector.h File Reference

Basic library for handling 2D vectors represented by float co-ordinates.

Data Structures

- struct Vector2D

  Represents two vectors in floats, for performance reasons.

Functions

- Vector2D ∗ VEC2D_new (float x, float y)

  Allocates a new Vector.
- Vector2D ∗ VEC2D_Pnew (float angle, float length)

  Allocates a new vector.
- void VEC2D_free (Vector2D ∗ptr)

  Deallocates a vector allocated by either VEC2D_Pnew() or VEC2D_new().
- Vector2D VEC2D_add (const Vector2D ∗srcA, const Vector2D ∗srcB)

  Adds two vectors together.
- Vector2D VEC2D_sub (const Vector2D ∗srcA, const Vector2D ∗srcB)

  Subtract the second vector from the first vector.
- Vector2D VEC2D_scale (Vector2D ∗srcA, float scale)

  Scale a vector with a given scalar.
- Vector2D VEC2D_rotate (Vector2D ∗srcA, float angle)

  Rotate a vector by a given angle.
- Vector2D VEC2D_normalize (Vector2D ∗srcA)

  Normalize a vector.
- float VEC2D_distance (Vector2D ∗a, Vector2D ∗b)

Calculate the distance between two points defined by two vectors.

- float VEC2D_scalar (Vector2D ∗a, Vector2D ∗b)

  Calculate the dot product of from two vectors.
- float VEC2D_angle (Vector2D ∗src)

  Get the angle between a vector and the X axis.
- float VEC2D_length (Vector2D ∗src)

  Get the length of a vecotor.

6.27.1 Detailed Description

Basic library for handling 2D vectors represented by float co-ordinates.

Author

Bendegúz Nagy

6.27.2 Function Documentation

6.27.2.1 Vector2D VEC2D_add (const Vector2D ∗ *srcA,* const Vector2D ∗ *srcB*)

Adds two vectors together.

Returns

the resulting vector.

6.27.2.2 float VEC2D_angle (Vector2D ∗ *src*)

Get the angle between a vector and the X axis.

Returns

the angle of a vector.

6.27.2.3 float VEC2D_distance (Vector2D ∗ *a,* Vector2D ∗ *b*)

Calculate the distance between two points defined by two vectors.

Returns

the distance.

6.27.2.4 void VEC2D_free (Vector2D ∗ *ptr*)

Deallocates a vector allocated by either VEC2D_Pnew() or VEC2D_new().

Parameters

|>p0.15|p0.805|

ptr	the vector to be freed.

6.27.2.5 float VEC2D_length (Vector2D ∗ *src*)

Get the length of a vecotor.

Returns

the length of a vector.

6.27.2.6 Vector2D∗ VEC2D_new (float *x,* float *y*)

Allocates a new Vector.

Parameters

|>p0.15|p0.805|

x	initial x coordinate.
y	initial y coordinate.

Returns

pointer to the allocated vector.

6.27.2.7 Vector2D VEC2D_normalize (Vector2D ∗ *srcA*)

Normalize a vector.

Returns

the resulting vector.

6.27.2.8 Vector2D∗ VEC2D_Pnew (float *angle,* float *length*)

Allocates a new vector.

Parameters

|>p0.15|p0.805|

angle	the initial angle.
length	the initial length.

Returns

pointer to the allocated vector.

6.27.2.9 Vector2D VEC2D_rotate (Vector2D ∗ *srcA,* float *angle*)

Rotate a vector by a given angle.

Returns

the resulting vector.

6.27.2.10 float VEC2D_scalar (Vector2D ∗ *a,* Vector2D ∗ *b*)

Calculate the dot product of from two vectors.

Returns

the calculated dot product.

6.27.2.11 Vector2D VEC2D_scale (Vector2D ∗ *srcA,* float *scale*)

Scale a vector with a given scalar.

Returns

the resulting vector.

6.27.2.12 Vector2D VEC2D_sub (const Vector2D ∗ *srcA,* const Vector2D ∗ *srcB*)

Subtract the second vector from the first vector.

Parameters

|>p0.15|p0.805|

srcA	the second to be subtracted from.

Returns

the resulting vector.

6.28 /home/bendeguz/ClionProjects/Dummy/Utility/SRC/bag.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include "../HEAD/bag.h"
```

Macros

- #define BAG_INIT_SIZE (16)

  The initial size of a bag.
- #define BAG_GROW_RATE (7.0/4.0)

  Scale at which the bag will grow if required.

Functions

- void AS_grow (Bag ∗stack)

  Internal function to increase the size of a Bag.
- Bag ∗ Bag_new (freeData freeDataPtr)

  Allocates a new Bag.
- void Bag_free (struct Bag ∗bag, int freeDataPtr)

Deallocate a bag allocated by Bag_new().

- int Bag_push (void ∗data, struct Bag ∗bag)

  Push a new element to the end of the Bag.
- void ∗ Bag_unorderedRemove (int index, Bag ∗bag)

  Remove an element at an index, place the last element in it's place.
- int Bag_search (void ∗data, Bag ∗bag)

  Linear search by pointer equality.
- void Bag_slowClear (Bag ∗bag, int free)

  Clears the bag and overwrites the backing array with zeros.
- void Bag_fastClear (Bag ∗bag)

  Sets the element count to zero.

6.28.1 Macro Definition Documentation

6.28.1.1 #define BAG_GROW_RATE (7.0/4.0)

Scale at which the bag will grow if required.

6.28.1.2 #define BAG_INIT_SIZE (16)

The initial size of a bag.

6.28.2 Function Documentation

6.28.2.1 void AS_grow (Bag ∗ *stack*)

Internal function to increase the size of a Bag.

6.28.2.2 void Bag_fastClear (Bag ∗ *bag*)

Sets the element count to zero.

6.28.2.3 void Bag_free (struct Bag ∗ *bag,* int *freeDataPtr*)

Deallocate a bag allocated by Bag_new().

Parameters

|>p0.15|p0.805|

bag	the bag to be destroyed.
freeDataPtr	non-zero if the bag should call the registered free function for each contained data pointer.

6.28.2.4 Bag∗ Bag_new (freeData *freeDataPtr*)

Allocates a new Bag.

Parameters

|>p0.15|p0.805|

freeDataPtr	a function that will be used to free held data when deleting elements.

Returns

the newly allocated Bag.

6.28.2.5 int Bag_push (void ∗ *data,* struct Bag ∗ *bag*)

Push a new element to the end of the Bag.

Returns

the index at which the new data will be stored, can change when elements are deleted.

6.28.2.6 int Bag_search (void ∗ *data,* Bag ∗ *bag*)

Linear search by pointer equality.

Returns

the index at which the data is stored, -1 if it could not be found.

6.28.2.7 void Bag_slowClear (Bag ∗ *bag,* int *free*)

Clears the bag and overwrites the backing array with zeros.

Parameters

|>p0.15|p0.805|

free if the registered free function should be called for each contained data pointer.

6.28.2.8 void∗ Bag_unorderedRemove (int *index,* Bag ∗ *bag*)

Remove an element at an index, place the last element in it's place.

Returns

the removed data pointer.

6.29 /home/bendeguz/ClionProjects/Dummy/Utility/SRC/vector.c File Reference

```
#include "../HEAD/vector.h"
#include <stdlib.h>
#include <math.h>
```

Functions

- Vector2D ∗ VEC2D_new (float x, float y)

  Allocates a new Vector.
- Vector2D ∗ VEC2D_Pnew (float angle, float length)

  Allocates a new vector.

- void VEC2D_free (Vector2D ∗ptr)

  Deallocates a vector allocated by either VEC2D_Pnew() or VEC2D_new().
- Vector2D VEC2D_add (const Vector2D ∗srcA, const Vector2D ∗srcB)

  Adds two vectors together.
- Vector2D VEC2D_sub (const Vector2D ∗srcA, const Vector2D ∗srcB)

  Subtract the second vector from the first vector.
- Vector2D VEC2D_scale (Vector2D ∗srcA, float scale)

  Scale a vector with a given scalar.
- Vector2D VEC2D_rotate (Vector2D ∗srcA, float angle)

  Rotate a vector by a given angle.
- Vector2D VEC2D_normalize (Vector2D ∗srcA)

  Normalize a vector.
- Vector2D VEC2d_integrate (Vector2D ∗srcA, Vector2D ∗srcInteg, float scale)
- float VEC2D_distance (Vector2D ∗a, Vector2D ∗b)

  Calculate the distance between two points defined by two vectors.
- float VEC2D_scalar (Vector2D ∗a, Vector2D ∗b)

  Calculate the dot product of from two vectors.
- float VEC2D_angle (Vector2D ∗src)

  Get the angle between a vector and the X axis.
- float VEC2D_length (Vector2D ∗src)

  Get the length of a vecotor.
- float VEC2d_lSquared (Vector2D ∗src)

6.29.1 Function Documentation

6.29.1.1 Vector2D VEC2D_add (const Vector2D ∗ *srcA*, const Vector2D ∗ *srcB*)

Adds two vectors together.

Returns

the resulting vector.

6.29.1.2 float VEC2D_angle (Vector2D ∗ *src*)

Get the angle between a vector and the X axis.

Returns

the angle of a vector.

6.29.1.3 float VEC2D_distance (Vector2D ∗ *a*, Vector2D ∗ *b*)

Calculate the distance between two points defined by two vectors.

Returns

the distance.

6.29.1.4 void VEC2D_free (Vector2D * *ptr*)

Deallocates a vector allocated by either VEC2D_Pnew() or VEC2D_new().

Parameters

|>p0.15|p0.805|

ptr the vector to be freed.

6.29.1.5 Vector2D VEC2d_integrate (Vector2D * *srcA,* Vector2D * *srcInteg,* float *scale*)

6.29.1.6 float VEC2D_length (Vector2D * *src*)

Get the length of a vecotor.

Returns

the length of a vector.

6.29.1.7 float VEC2d_lSquared (Vector2D * *src*)

6.29.1.8 Vector2D* VEC2D_new (float *x,* float *y*)

Allocates a new Vector.

Parameters

|>p0.15|p0.805|

x initial x coordinate.

y initial y coordinate.

Returns

pointer to the allocated vector.

6.29.1.9 Vector2D VEC2D_normalize (Vector2D * *srcA*)

Normalize a vector.

Returns

the resulting vector.

6.29.1.10 Vector2D* VEC2D_Pnew (float *angle,* float *length*)

Allocates a new vector.

Parameters

|>p0.15|p0.805|

angle the initial angle.

length the initial length.

Returns

pointer to the allocated vector.

6.29.1.11 Vector2D VEC2D_rotate (Vector2D ∗ *srcA,* float *angle*)

Rotate a vector by a given angle.

Returns

the resulting vector.

6.29.1.12 float VEC2D_scalar (Vector2D ∗ *a,* Vector2D ∗ *b*)

Calculate the dot product of from two vectors.

Returns

the calculated dot product.

6.29.1.13 Vector2D VEC2D_scale (Vector2D ∗ *srcA,* float *scale*)

Scale a vector with a given scalar.

Returns

the resulting vector.

6.29.1.14 Vector2D VEC2D_sub (const Vector2D ∗ *srcA,* const Vector2D ∗ *srcB*)

Subtract the second vector from the first vector.

Parameters

|>p0.15|p0.805|

srcA the second to be subtracted from.

Returns

the resulting vector.