2PII

Generated by Doxygen 1.9.1

1 2PII 2

1.1 Compilation

Some dependencies are required to compile, they can be installed with `sudo apt-get install libsdl2-dev libomp-dev libsdl2-ttf-dev libsdl2-mixer-dev` on Debian based distributions.

1.2 Doxygen

The use of `make doc` requires doxygen, LateX and graphviz.

The use of the command `make htmldoc` requires firefox

The html documentation is also available at https://uwu-segfault.eu/2p2doc/

2 resources

Here is a list of documents used during the making of this project

Websites and pages :

- https://web.archive.org/web/19990903133921/http://www.concentric.net/~Ttwang/tech/primehash.htm
- https://courses.csail.mit.edu/6.006/spring11/rec/rec07.pdf
- https://wiki.libsdl.orgs/SDL2
- https://en.cppreference.com
- https://ianjk.com/ecs-in-rust/
- https://austinmorlan.com/posts/entity_component_system/
- https://www.david-colson.com/2020/02/09/making-a-simple-ecs.html

- https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function
- https://www.libsdl.org/release/SDL-1.2.15/docs/html/
- https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf
- https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html#work-sharing-direct
- http://www.gameaipro.com/
- https://valgrind.org/docs/manual/index.html
- https://lazyfoo.net/tutorials/SDL/21_sound_effects_and_music/index.php

Books and articles :

- Game AI Pro 360: Guide to Movement and Pathfinding - Steve Rabin - 2019
- Steering Behaviors For Autonomous Characters - Craig W. Reynolds - 1999
- Game Engine Architecture, 3rd edition - Jason Gregory - 2018

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Data Structure Documentation

5.1 Animator Struct Reference

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

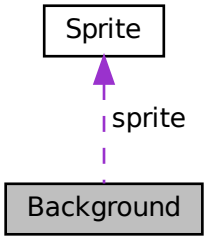
- anim.h

5.2 Background Struct Reference

Entities with this component are the background of the user interface.

`#include <ui.h>`

Collaboration diagram for Background:



Data Fields

- Sprite * **sprite**
- SDL_Rect * **rect**

5.2.1 Detailed Description

Entities with this component are the background of the user interface.

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

- ui.h

5.3 Camera Struct Reference

`#include <camera.h>`

Data Fields

- float **x**
- float **y**
- float zoom

5.3.1 Detailed Description

The `Camera` struct is not a component, it is meant to have exactly one instance and serves as the base for screenspace<->worldspace calculations

5.3.2 Field Documentation

5.3.2.1 zoom `float Camera::zoom`

`zoom` is such that if `zoom==1`, one pixel in screenspace is one pixel in worldspace, while if `zoom==2`, one pixel in screenspace is two pixels in worldspace

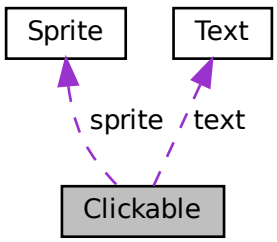
The documentation for this struct was generated from the following file:

- camera.h

5.4 Clickable Struct Reference

`#include <ui.h>`

Collaboration diagram for Clickable:



Data Fields

- Sprite * **sprite**
- SDL_Rect * **rect**
- Text * **text**
- Uint8 **is_clicked**
- void(* **click_event**)()

5.4.1 Detailed Description

Entities with this component start an action when clicked on. The value of is_clicked depends if and how it is clicked on, when is_clicked is equal to one it means the left click is pressed on over the clickable and that either it was already equal to one before or that it was being clicked on. If it is equal to two, it means the value was one and the click was released while over it. It activates the clickable's click_event.

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

- ui.h

5.5 ComponentWrapper Struct Reference

Used to store the component, its type and its id.

`#include <ecs.h>`

Data Fields

- uint64_t id
  The component id.
- int type_id
  The id refering to the component type.
- void * component
  A pointer to the component itself.

5.5.1 Detailed Description

Used to store the component, its type and its id.

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

- ecs.h

5.6 Entity Struct Reference

The entity structure for the ECS.

`#include <ecs.h>`

Public Member Functions

- VEC (uint64_t) components
  A vector of `ComponentWrapper` containing the entity's components.

Data Fields

- uint64_t id
  The entity's id.

5.6.1 Detailed Description

The entity structure for the ECS.

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

- ecs.h

5.7 Entry Struct Reference

Data Fields

- int **ind**
- int **from**

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

- pathfinding.c

5.8 HashMap Struct Reference

A hash map.

`#include <hash_map.h>`

Public Member Functions

- VEC (LinkedList) bucket
  The vector that stores the entries.

Data Fields

- uint64_t(∗ hash_function)(void ∗)
  The function used for hashing the values stored in the `HashMap`
- char(∗ comp_function)(void ∗, void ∗)
  The function used to compare values in the `HashMap`
- uint length
  Length of the bucket.
- uint size
  Numberb of elements in the hashmap.

5.8.1 Detailed Description

A hash map.

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

- hash_map.h

5.9 HashMapEntry Struct Reference

An entry in a `HashMap`, i.e. a key-value pair.

`#include <hash_map.h>`

Data Fields

- void * **key**
- void * **value**
- uint64_t hash

  The hash of `value`

5.9.1 Detailed Description

An entry in a `HashMap`, i.e. a key-value pair.

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

- hash_map.h

5.10 Hoverable Struct Reference

Entities with this component show text when hovered.

`#include <ui.h>`

Data Fields

- SDL_Rect * **rect**
- char * **text**

5.10.1 Detailed Description

Entities with this component show text when hovered.

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

- ui.h

5.11 Inputs Struct Reference

stores keys and mouse buttons

`#include <input.h>`

Data Fields

- int * keys
  uses SDL Scancodes as indices
- Uint64 key_nb
  number of keys currently in
- char mouse
  1st bit = mb_left; 2nd bit = mb_middle; 3rd bit = mb_right

5.11.1 Detailed Description

stores keys and mouse buttons

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

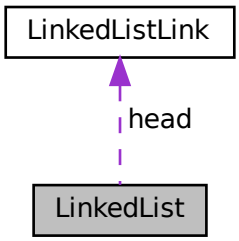
- input.h

5.12 LinkedList Struct Reference

A singly linked list.

`#include <linked_list.h>`

Collaboration diagram for LinkedList:



Data Fields

- LinkedListLink * head
  Pointer to the the first link of the list. `NULL` if empty.

5.12.1 Detailed Description

A singly linked list.

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

- linked_list.h

5.13 LinkedListLink Struct Reference

A link of `LinkedList`

`#include <linked_list.h>`

Data Fields

- void * data
- struct _Lk * next

  Pointer to the next link in the list. `NULL` if last.

5.13.1 Detailed Description

A link of `LinkedList`

5.13.2 Field Documentation

5.13.2.1 data `void* LinkedListLink::data`

Pointer to this link's data. Figuring out which type it is is up to the user.

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

- linked_list.h

5.14 MapComponent Struct Reference

`#include <map.h>`

Data Fields

- Map **map**

5.14.1 Detailed Description

A component that contains a `Map`, for rendering purposes. Having more than one such component is undefined behabior.

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

- map.h

5.15 Minimap Struct Reference

Component that corresponds to the minimap.

`#include <ui.h>`

Data Fields

- SDL_Rect * **rect**

5.15.1 Detailed Description

Component that corresponds to the minimap.

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

- ui.h

5.16 Position Struct Reference

A component that contains the world space coordinates of an entity.

`#include <camera.h>`

Data Fields

- float **x**
- float **y**

5.16.1 Detailed Description

A component that contains the world space coordinates of an entity.

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

- camera.h

5.17 PQueueEntry Struct Reference

an entry within a `PQueue`

`#include <pqueue.h>`

Data Fields

- void * **value**
- double **weight**

5.17.1 Detailed Description

an entry within a `PQueue`

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

- pqueue.h

5.18 Rc Struct Reference

Data Fields

- uintptr_t **counter**
- void * **ref**
- AssetKind **kd**
- char **locked**

5.18.1 Detailed Description

A reference counter for the assets. Should only be used through the API available in `asset_manager.h`

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

- asset_manager.c

5.19 Sprite Struct Reference

Data Fields

- SDL_Texture * **texture**
- SDL_Rect * **rect**

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

- sprite.h

5.20 Text Struct Reference

Data Fields

- char * **str**
- SDL_Color * **color**

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

- ui.h

5.21 TilePosition Struct Reference

stores the position of a tile

`#include <pathfinding.h>`

Data Fields

- int **x**
- int **y**

5.21.1 Detailed Description

stores the position of a tile

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

- pathfinding.h

5.22 Vec2 Struct Reference

a 2d vector for use in units movement

`#include <util.h>`

Data Fields

- float **x**
- float **y**

5.22.1 Detailed Description

a 2d vector for use in units movement

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

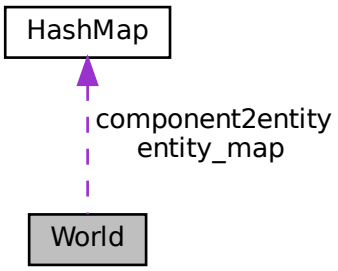
- util.h

5.23 World Struct Reference

The world structure used to store the different parts of the ECS.

`#include <ecs.h>`

Collaboration diagram for World:



Public Member Functions

- VEC (uint) component_sizes
- void (VEC() *component_free)(void *)
- VEC (ComponentWrapper) components

  A vector of `ComponentWrapper` containing all the components.
- VEC (Entity) entities

  A vector of `Entity` containing all the entities.
- VEC (uint) component_sparsity

  Stores the available spaces in `components` that entity deletion created.
- VEC (uint) entity_sparsity

  Stores the available spaces in `entities` that entity deletion created.

Data Fields

- HashMap entity_map
- HashMap component2entity
- uint last_component

  Indicates the id the next component to be added should take.

5.23.1 Detailed Description

The world structure used to store the different parts of the ECS.

5.23.2 Member Function Documentation

5.23.2.1 VEC() `World::VEC (`
`uint )`

A vector containing all the sizes corresponding to each of the components' types

5.23.2.2 void() `World::void (`
`VEC() * component_free )`

A vector of functions used to free each of the compontents (one function per type)

5.23.3 Field Documentation

5.23.3.1 component2entity `HashMap World::component2entity`

A `HashMap` with `uint64_t` as keys and `uint64_t` as values, the keys are components'ids and the values are entities'ids. It establishes for each component the list of the entities currently linked to it

5.23.3.2 entity_map `HashMap World::entity_map`

A `HashMap` with `Bitflag` as keys and `VEC(uint64_t)` as values, the map is used to easily access the list of entities corresponding to the system represented by the `Bitflag` key

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

- ecs.h

6 File Documentation

6.1 anim.h File Reference

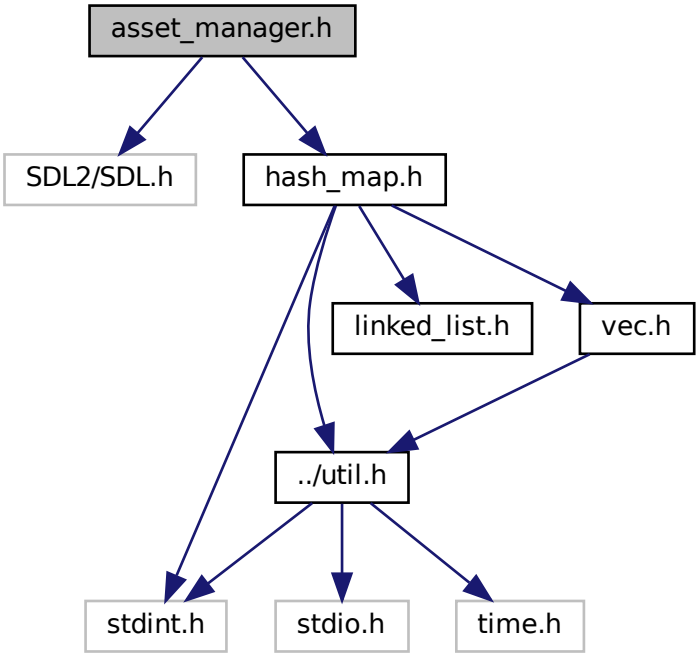
Data Structures

- struct Animator

6.2 asset_manager.h File Reference

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

Include dependency graph for asset_manager.h:



Functions

- void init_asset_manager ()
- Error **lock_asset** (char *t, char locked)
- char **is_asset_locked** (char *t)
- void * get_texture (char *t, SDL_Renderer *renderer, SDL_Window *window)
- void * load_texture (char *t, SDL_Renderer *renderer, SDL_Window *window)
- int drop_texture (char *t)
- void * **get_audio** (char *t, char is_mus)
- void * load_audio (char *t, char is_mus)
- int **drop_audio** (char *t)
- void * **load_font** (char *t, Uint8 size)
- void * get_font (char *t, Uint8 size)
- int **drop_font** (char *font, Uint8 size)
- void **free_asset_store** ()

Variables

- HashMap ASSET_STORE

  Stores and manages the textures used in the game.

6.2.1 Function Documentation

6.2.1.1 drop_texture()

```
int drop_texture (
        char * t )
```

Returns a pointer to the audio from file `t`. Will had it to the `ASSET_STORE` if it is not in it yet

6.2.1.2 get_font()

```
void* get_font (
        char * t,
        Uint8 size )
```

Returns a pointer to the font from file `t`. Will add it to the `ASSET_STORE` if it is not in it yet.

6.2.1.3 get_texture()

```
void* get_texture (
        char * t,
        SDL_Renderer * renderer,
        SDL_Window * window )
```

Returns a pointer to the texture from file `t`. Will had it to the `ASSET_STORE` if it is not in it yet

6.2.1.4 init_asset_manager()

```
void init_asset_manager ( )
```

Initializes the `ASSET_STORE`; must be called before any call to `get_texture` or `load_texture`

6.2.1.5 load_audio()

```
void* load_audio (
        char * t,
        char is_mus )
```

Loads the audio from file `t` in the `ASSET_STORE` While calling it multiple times with the same `t` shouldn't fail, it is unadvisable as slow. Crashes on invalid file path or audio creation.

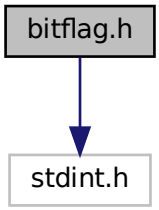
6.2.1.6 load_texture()

```
void* load_texture (
        char * t,
        SDL_Renderer * renderer,
        SDL_Window * window )
```

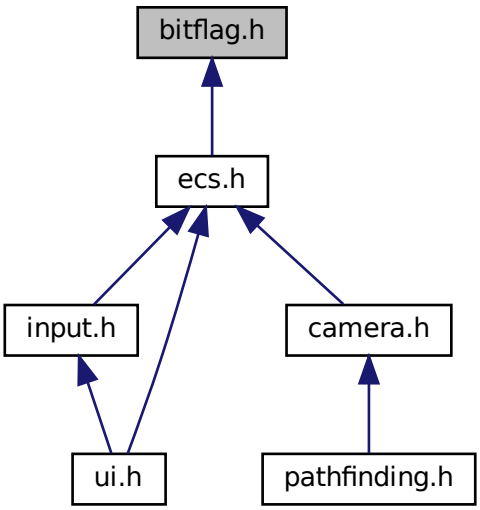
Loads the texture from file `t` in the `ASSET_STORE` While calling it multiple times with the same `t` shouldn't fail, it is unadvisable as slow. Crashes on invalid file path or texture creation.

6.3 bitflag.h File Reference

`#include <stdint.h>`
Include dependency graph for bitflag.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define bitflag_get(b, r) (((b) >> (r)) & 1)
  expands to the `r`th least significant bit of `b`
- #define bitflag_set(b, r, v) ((v) ? (1 << (r)) | (b) : (~(1 << (r))) & (b))
  expands to the value of `b` with its `r`th least significant bit set to `v`

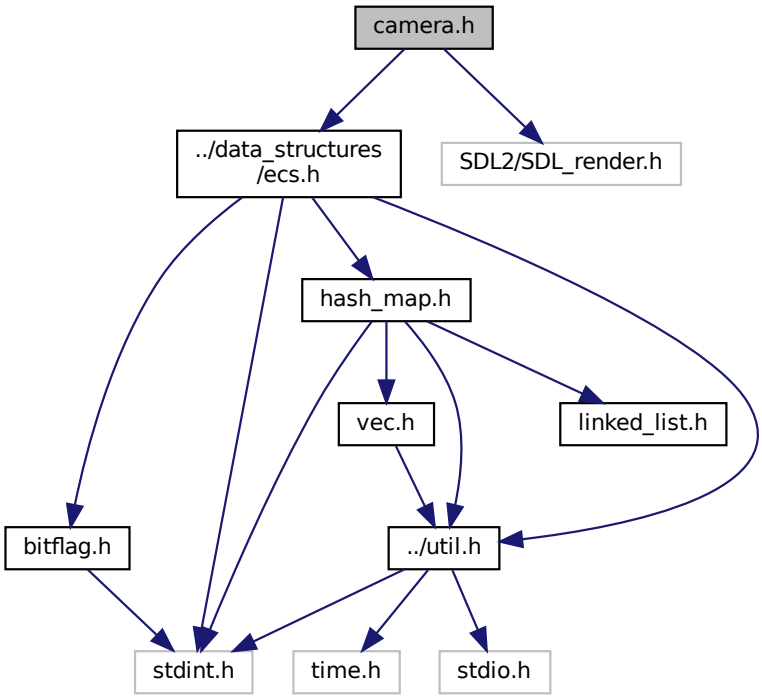
Typedefs

- typedef uint64_t **Bitflag**

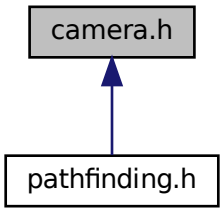
6.4 camera.h File Reference

```
#include "../data_structures/ecs.h"
#include <SDL2/SDL_render.h>
```

Include dependency graph for camera.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Camera
- struct Position

  A component that contains the world space coordinates of an entity.

Functions

- Position world2screenspace (Position *p, Camera *cam)

  Transfers `p` *to screenspace, according to* `cam`
- Position screen2worldspace (Position *p, Camera *cam)

  Transfers `p` *to worldspace, according to* `cam`
- void render (World *w, SDL_Renderer *rdr, Camera *cam, SDL_Window *window)

6.4.1 Function Documentation

6.4.1.1 render()

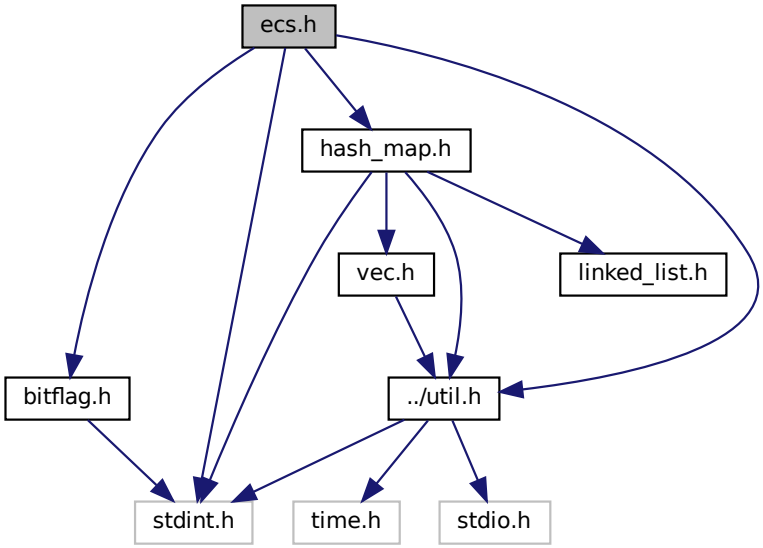
```
void render (
        World * w,
        SDL_Renderer * rdr,
        Camera * cam,
        SDL_Window * window )
```

Renders any entity with a `Position` and a `Sprite`, according to `cam`. Said position must be in worldspace coordinates. Also renders the map if found.

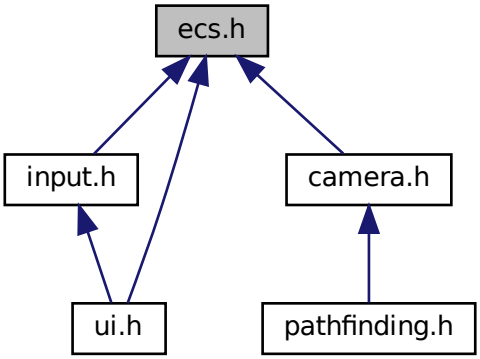
6.5 ecs.h File Reference

```
#include "../util.h"
#include "bitflag.h"
#include "hash_map.h"
#include <stdint.h>
```

Include dependency graph for ecs.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ComponentWrapper

  Used to store the component, its type and its id.
- struct Entity

  The entity structure for the ECS.
- struct World

  The world structure used to store the different parts of the ECS.

Macros

- #define register_component(w, tp) register_component_inner_callback((w), sizeof(tp), free)
- #define register_component_callback(w, tp, callback) register_component_inner_callback((w), sizeof(tp), (callback))
- #define parallelize_query(erefs, commands)

Typedefs

- typedef uint64_t EntityRef

Functions

- char eq_u64 (void *a, void *b)
- World world_new ()

  Returns a new initialized `World` structure.
- void world_free (World *)

  Frees a `World` structure created using `world_new`
- int register_component_inner_callback (World *w, int size, void(*callback)(void *))
- void register_system_requirement (World *w, Bitflag b)
- Entity * spawn_entity (World *w)

Spawns an `Entity` into the world and returns a pointer to it.

- void ecs_add_component (World *w, Entity *e, int cid, void *c)
- void despawn_entity (World *w, Entity *e)

  Despawns an `Entity`
- void despawn_from_component (World *w, Bitflag b)

  Despawns every `Entity` with this `Bitflag`
- Entity * get_entity (World *w, EntityRef ref)

  Returns an `Entity` pointer corresponding to the passed reference.
- VEC (EntityRef) world_query(World *w
- void * entity_get_component (World *w, Entity *e, int type)

Variables

- Bitflag * **b**

6.5.1 Macro Definition Documentation

6.5.1.1 parallelize_query `#define parallelize_query(`

erefs,
commands)

Value:

```
{                                                                    \
  _Pragma("omp parallel") {                                          \
    _Pragma("omp for") {                                             \
      for (uint i = 0; i < vec_len(erefs); i++) {                    \
        EntityRef ei = erefs[i];                                     \
        commands;                                                    \
      }                                                              \
    }                                                                \
  }                                                                  \
}
```

Expands to a parallel query on the elements of `erefs`. `erefs` is expected to be the return value of `world_query`, and must be a glvalue. Commands are executed with the understanding that they can access the element they work on with `ei`. Note that spawning the threads is a significant overhead. For trivial cases, using the sequential method can be faster. If unsure, use `TIME` to benchmark both usecases. Note that Valgrind will detect some "possibly lost memory". This is intended behavior, see https://gcc.gnu.org/bugzilla/show_bug.cgi?id=36298

6.5.1.2 register_component `#define register_component(`

w,
tp) register_component_inner_callback((w), sizeof(tp), free)

`register_component(World*, type)` where type is the type of the component. Registers a new component that uses `free` as a way to free it

6.5.1.3 register_component_callback `#define register_component_callback(`

w,
tp,
callback) register_component_inner_callback((w), sizeof(tp), (callback))

`register_component(World*, type, void (*callback)(void *))` where type is the type of the component. Registers a new component using a callback function to free it

6.5.2 Typedef Documentation

6.5.2.1 EntityRef `typedef uint64_t EntityRef`

Note that this reference is only valid until the number of entities decreases

6.5.3 Function Documentation

6.5.3.1 ecs_add_component() `void ecs_add_component (`

```
World * w,
Entity * e,
int cid,
void * c )
```

Links a component to an `Entity`. The component itself need to live as long as the world does (beware of scopes)

6.5.3.2 entity_get_component() `void* entity_get_component (`

```
World * w,
Entity * e,
int type )
```

Returns a pointer to the component of type `type` linked to the `Entity`, if no component of this type is linked the the `Entity` the NULL pointer is returned

6.5.3.3 eq_u64() `char eq_u64 (`

```
void * a,
void * b )
```

Returns a normalized boolean (0 or 1) indicating if the two arguments are equal when both interpreted as `uint64_t`

6.5.3.4 register_component_inner_callback() `int register_component_inner_callback (`

```
World * w,
int size,
void(*)(void *) callback )
```

Registers a new component using a callback function to free it, the size of the component's type needs to be passed instead of the type itself

6.5.3.5 register_system_requirement() `void register_system_requirement (`

```
World * w,
Bitflag b )
```

Updates the entity_map of the world to take into account the system represented by the `Bitflag` argument. Please not that single-component requirements SHOULD NOT be registered. This is considered undefined behavior, as well as registering the same requirements more than once.

6.5.3.6 VEC() VEC (EntityRef)

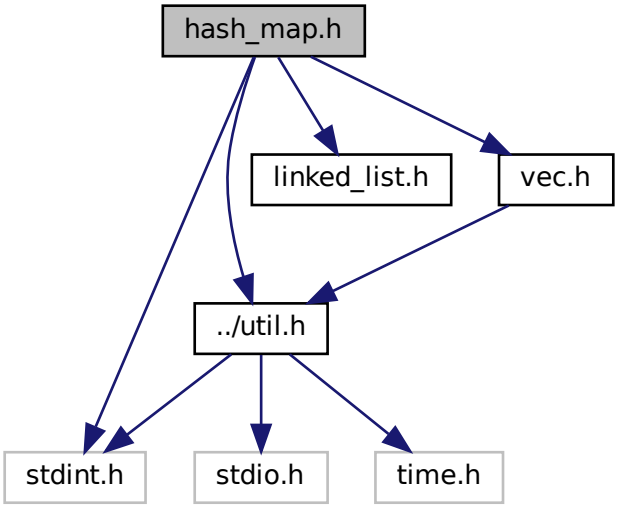
Returns a vector of EntityRef referencing entities corresponding to the system described by the Bitflag argument. If you want to modify the World based on the return value of this function, use world_query_mut instead. The system needs to be registered using register_system_requirement before using this function

Returns a pointer to a vector of EntityRef referencing entities corresponding to the system described by the Bitflag argument. The system needs to be registered using register_system_requirement before using this function

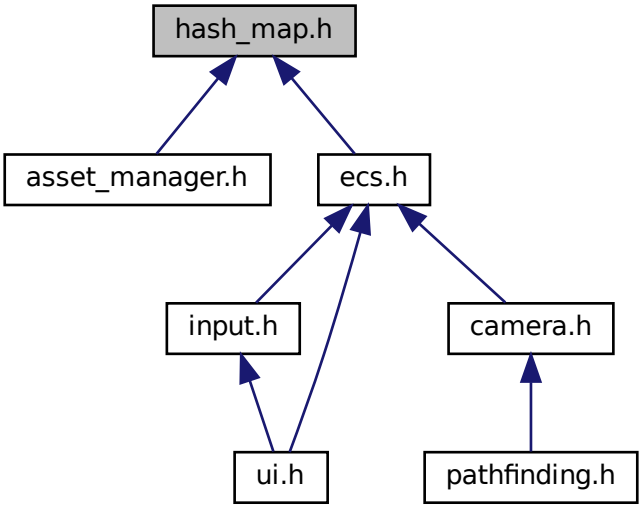
6.6 hash_map.h File Reference

```
#include <stdint.h>
#include "../util.h"
#include "linked_list.h"
#include "vec.h"
```

Include dependency graph for hash_map.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct HashMapEntry

  An entry in a `HashMap`, i.e. a key-value pair.
- struct HashMap

  A hash map.

Macros

- #define HASHMAP_DEFAULT_LENGTH 32

  The initial length of the internal array of a `HashMap`
- #define HASHMAP_OCCUP_MAX 0.7

  The occupation ratio of a `HashMap` over which it grows.
- #define HASHMAP_OCCUP_MIN 0.3

  The occupation ratio of a `HashMap` below which it shrinks.

Functions

- uint64_t hash_str (void ∗)

  A polynomial rolling hash for strings.
- uint64_t hash_u64 (void ∗)

  A FNV hash function for 64 bit integers.
- uint64_t hash_u8 (void ∗)

  A FNV hash function for 8 bit integers.
- HashMap hash_map_create (uint64_t(∗hash)(void ∗), char(∗cmp)(void ∗, void ∗))
- void hash_map_free_callback (HashMap ∗h, void(∗callback)(void ∗))

Frees h, calling callback on each entry to free it.

- void hash_map_free (HashMap *h)

  Same as hash_map_free_callback but uses hash_map_entry_free as callback.

- void hash_map_free_void (void *h)

  Same as hash_map_free, deprecated.

- int hash_map_insert_callback (HashMap *h, void *k, void *v, void(*callback)(void *))
- int hash_map_insert (HashMap *h, void *k, void *v)
- int hash_map_delete_callback (HashMap *h, void *k, void(*callback)(void *))

  deletes the entry with key k using callback

- int hash_map_delete (HashMap *h, void *k)

  Same as hash_map_delete_callback but uses hash_map_entry_free as callback.

- void * hash_map_get (HashMap *h, void *k)

6.6.1 Function Documentation

6.6.1.1 hash_map_create()

```
HashMap hash_map_create (
        uint64_t(*)(void *) hash,
        char(*)(void *, void *) cmp )
```

Creates and returns a new HashMap that uses hash as the hash function and cmp as the comparison function

6.6.1.2 hash_map_get()

```
void* hash_map_get (
        HashMap * h,
        void * k )
```

Returns the value associated with key k, or a null pointer if there is no such pair

6.6.1.3 hash_map_insert()

```
int hash_map_insert (
        HashMap * h,
        void * k,
        void * v )
```

Same as hash_map_insert_callback but uses hash_map_entry_free as callback

6.6.1.4 hash_map_insert_callback()

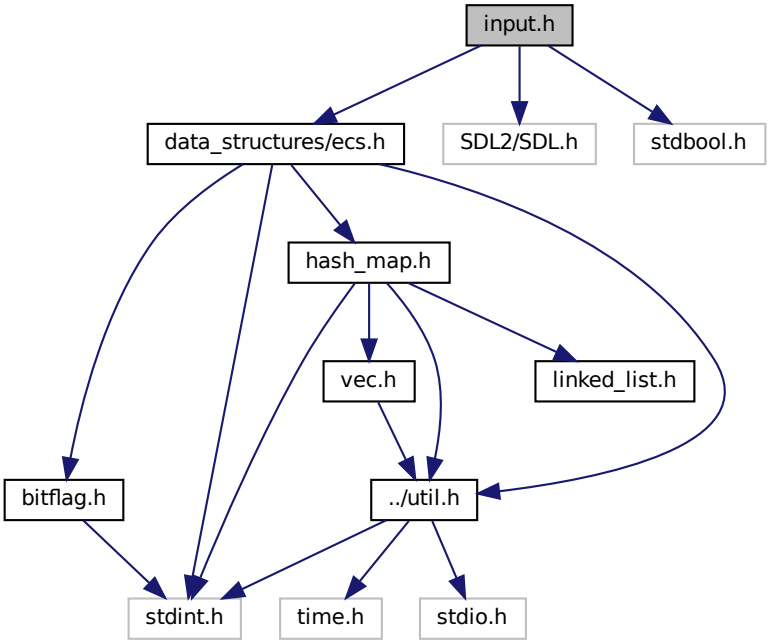
```
int hash_map_insert_callback (
        HashMap * h,
        void * k,
        void * v,
        void(*)(void *) callback )
```

Inserts the key-value pair k,v in h, deleting any previous entry of key k with callback

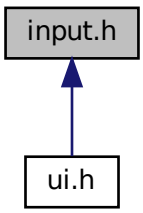
6.7 input.h File Reference

```
#include "data_structures/ecs.h"
#include <SDL2/SDL.h>
#include <stdbool.h>
```

Include dependency graph for input.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Inputs

  stores keys and mouse buttons

Macros

- #define KEY_PRESSED 0

  the instant the key is pressed
- #define KEY_RELEASED 1

  the instant the key is released
- #define KEY_DOWN 2

  starts on press (included), ends on release (not included)
- #define inputs_is_key_in_from_scancode(inputs, scancode) ((inputs)->keys[(scancode)])
- #define inputs_is_key_in(inputs, key) ((inputs)->keys[SDL_GetScancodeFromKey(key)])
- #define inputs_is_mouse_button_in(inputs, button) (((inputs)->mouse >> ((button)-1)) & 1)
- #define inputs_update_key_in(inputs, key, new_val)
- #define inputs_update_mouse_button_in(inputs, button, new_val)

Typedefs

- typedef Uint8 **KeyState**
- typedef Uint8 MouseButton
- typedef void(* KeyEvent) (World *, SDL_Renderer *, Entity *, Inputs *, KeyState)

  type of callback functions for the key events

Functions

- Inputs * inputs_new ()

  creates a new Inputs instance
- void inputs_free (Inputs *)

  frees the Inputs instance
- void inputs_update_key_in_from_scancode (Inputs *inputs, SDL_Scancode scancode, bool new_val)
- void inputs_run_callbacks (World *, SDL_Renderer *rdr, Inputs *, KeyState)

  calls all the callbacks for the keyevent
- Uint8 mouse_in_rect (SDL_Renderer *rdr, SDL_Rect *rect)

  Checks if the mouse is in the rectangle.

6.7.1 Macro Definition Documentation

6.7.1.1 inputs_is_key_in

```
#define inputs_is_key_in(
        inputs,
        key )   ((inputs)->keys[SDL_GetScancodeFromKey(key)])
```

the state of a key accessed using SDL_KeyCode bool inputs_is_key_in(Inputs*, SDL_KeyCode)

6.7.1.2 inputs_is_key_in_from_scancode

```
#define inputs_is_key_in_from_scancode(
        inputs,
        scancode )   ((inputs)->keys[(scancode)])
```

the state of a key accessed using SDL_Scancode !!!!!!!!!! this does not take into account non QWERTY keyboards / remaps !!!!!!!!!! bool inputs_is_key_in_from_scancode(Input*,SDL_Scancode)

6.7.1.3 inputs_is_mouse_button_in

```
#define inputs_is_mouse_button_in(
        inputs,
        button )  (((inputs)->mouse >> ((button)-1)) & 1)
```

the state of a mouse button bool inputs_is_mouse_button_in(Inputs*,MouseButton)

6.7.1.4 inputs_update_key_in

```
#define inputs_update_key_in(
        inputs,
        key,
        new_val )
```

Value:

```
(inputs_update_key_in_from_scancode(inputs, SDL_GetScancodeFromKey(key),     \
                                    new_val))
```

updates the state of a key using SDL_KeyCode void inputs_update_key_in(Input*,SDL_KeyCode,bool)

6.7.1.5 inputs_update_mouse_button_in

```
#define inputs_update_mouse_button_in(
        inputs,
        button,
        new_val )
```

Value:

```
((inputs)->mouse = ((((!!(new_val)) « ((button)-1)) |                        \
                    ((inputs)->mouse & (~(1 « ((button)-1))))))
```

updates the state of a mouse button MouseButton inputs_update_mouse_button_in(Input*,MouseButton,bool)

6.7.2 Typedef Documentation

6.7.2.1 MouseButton

```
typedef Uint8 MouseButton
```

describes any of the following: SDL_BUTTON_LEFT,SDL_BUTTON_MIDDLE, SDL_BUTTON_RIGHT

6.7.3 Function Documentation

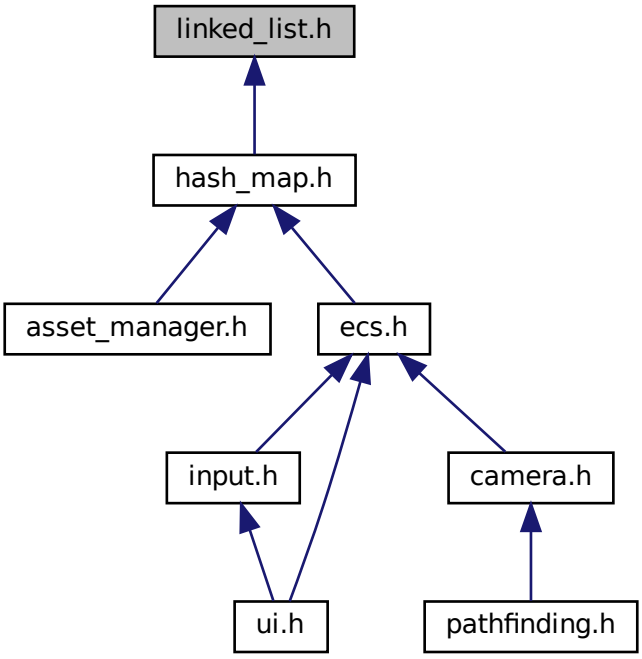
6.7.3.1 inputs_update_key_in_from_scancode()

```
void inputs_update_key_in_from_scancode (
        Inputs * inputs,
        SDL_Scancode scancode,
        bool new_val )
```

updates the state of a key using SDL_Scancode !!!!!!!!!! this does not take into account non QWERTY keyboards / remaps !!!!!!!!!! void inputs_update_key_in_from_scancode(Input*,SDL_Scancode,bool)

6.8 linked_list.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct LinkedListLink

  A link of `LinkedList`
- struct LinkedList

  A singly linked list.

Functions

- LinkedList linked_list_create ()

  Creates a `LinkedList`
- int linked_list_insert (LinkedList ∗l, void ∗e, int i)
- int linked_list_remove (LinkedList ∗l, int i)

  Same as `linked_list_remove_callback`*, with* `free` *as the* `callback`
- int linked_list_remove_callback (LinkedList ∗l, int i, void(∗callback)(void ∗))
- void linked_list_free (LinkedList ∗)

  Same as `linked_list_free`*, with* `free` *as the* `callback`
- void linked_list_free_callback (LinkedList ∗l, void(∗callback)(void ∗))
- void ∗ linked_list_get (LinkedList ∗l, int i)

  Returns the `data` *field of the* `i`*th element of* `l`

6.8.1 Function Documentation

6.8.1.1 linked_list_free_callback()

```
void linked_list_free_callback (
        LinkedList * l,
        void(*)(void *) callback )
```

Frees `l`, calling `callback` on the `data` fields of each link as a way to free them

6.8.1.2 linked_list_insert()

```
int linked_list_insert (
        LinkedList * l,
        void * e,
        int i )
```

Add `e` as an element of `l` at index `i` Returns `0` on success, `-1` on allocation error and `-2` if i is out of range

6.8.1.3 linked_list_remove_callback()

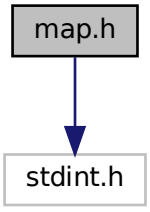
```
int linked_list_remove_callback (
        LinkedList * l,
        int i,
        void(*)(void *) callback )
```

Removes element at index `i` in `l`, running `callback` on its `data` as a way to free it

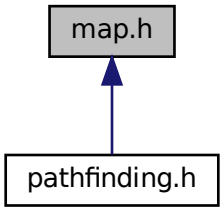
6.9 map.h File Reference

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

Include dependency graph for map.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct MapComponent

Macros

- #define TILE_SIZE 16

  `TILE_SIZE` is the ingame length of a tile's edge (tiles being squares)
- #define map_width(m) ((int *)(m))[-2]

  returns the width of a `Map`
- #define map_height(m) ((int *)(m))[-1]

  returns the height of a `Map`

Typedefs

- typedef TileTypes ** Map

Enumerations

- enum TileTypes { **TILE_PLAIN** , **TILE_FOREST** , **TILE_NUMBER** }

  an `enum` containing all the tiles for the game

Functions

- void map_component_free (void *a)

  frees a `MapComponent`, for use in the ecs.
- Map map_create (int w, int h)

  returns a new `Map` initialized at 0 with size `w*h`
- void map_free (Map m)

  frees a `Map` created with `map_create`
- Map load_map_from_bmp (char *path)
- char * get_tile_file_name (int8_t id)

6.9.1 Typedef Documentation

6.9.1.1 Map `typedef TileTypes** Map`

used to store a map as a matrix of `TileTypes` (each value designates a specific type of tile, ex: water, plain...)

6.9.2 Function Documentation

6.9.2.1 get_tile_file_name() `char* get_tile_file_name ( int8_t id )`

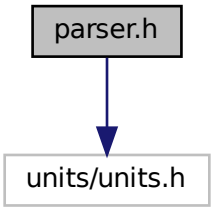
Returns the file name associated with a certain color. Return value should be freed.

6.9.2.2 load_map_from_bmp() `Map load_map_from_bmp ( char * path )`

Creates a `Map` from the bitmap pointed to by `path`. Said bitmap sould be single channel, with 8 bit per color.

6.10 parser.h File Reference

`#include "units/units.h"`
Include dependency graph for parser.h:



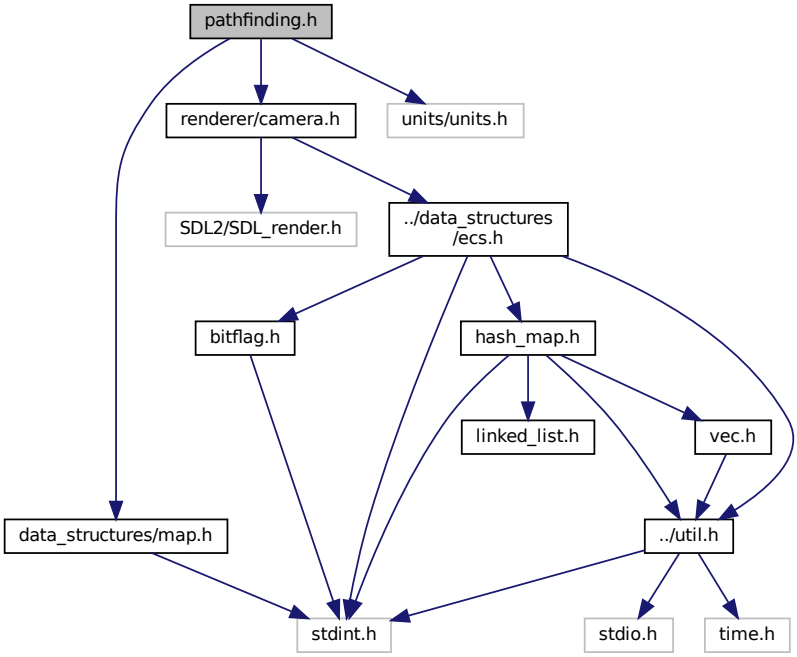
Functions

- Unit * **parse** (char *path, SDL_Renderer *renderer, SDL_Window *window)

6.11 pathfinding.h File Reference

```
#include "data_structures/map.h"
#include "renderer/camera.h"
#include "units/units.h"
```

Include dependency graph for pathfinding.h:



Data Structures

- struct TilePosition

  stores the position of a tile

Functions

- typedef VEC (TilePosition ∗) Path
- void **path_free** (Path p)
- Path pathfind_astar (Map m, UnitTypes u, TilePosition ∗src, TilePosition ∗dest)

  returns a minimal `Path` using the A∗ algorithm
- double pathfind_astar_heuristic (UnitTypes u, TilePosition ∗src, TilePosition ∗dest)

6.11.1 Function Documentation

6.11.1.1 pathfind_astar_heuristic()

```
double pathfind_astar_heuristic (
    UnitTypes u,
    TilePosition * src,
    TilePosition * dest )
```

returns the distance between `src` and `dest` times the lowest cost for crossing a tile (cost = 1/speed) (currently using the euclidean distance)

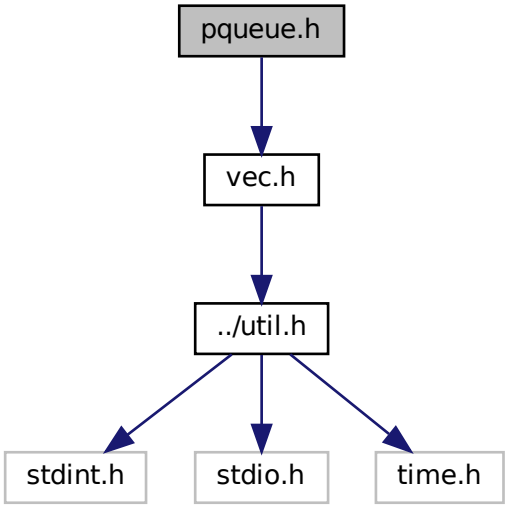
6.11.1.2 VEC()

```
typedef VEC (
    TilePosition *  )
```

a `Path` on the global map is succession of tile positions, first index being the start of the path

6.12 pqueue.h File Reference

`#include "vec.h"`
Include dependency graph for pqueue.h:



Data Structures

- struct PQueueEntry

  an entry within a `PQueue`

Macros

- #define pqueue_new() vec_new(PQueueEntry *)

  creates an empty `PQueue`
- #define pqueue_len(p) vec_len(p)

  returns the number of elements currently into the queue
- #define **pqueue_push**(p, val, weight) (p = pqueue_push_inner(p, val, weight))

Functions

- typedef VEC (PQueueEntry ∗) PQueue

  a priority queue
- void pqueue_free (PQueue p)

  frees the queue (does not free the elements still within the queue)
- void pqueue_free_callback (PQueue p, void(∗callback)(void ∗))

  frees the queue and call `callback` on each element still in the queue
- PQueueEntry ∗ pqueue_pop (PQueue p)

  removes and returns the element with the smallest weight from the queue
- PQueueEntry ∗ pqueue_get (PQueue p)

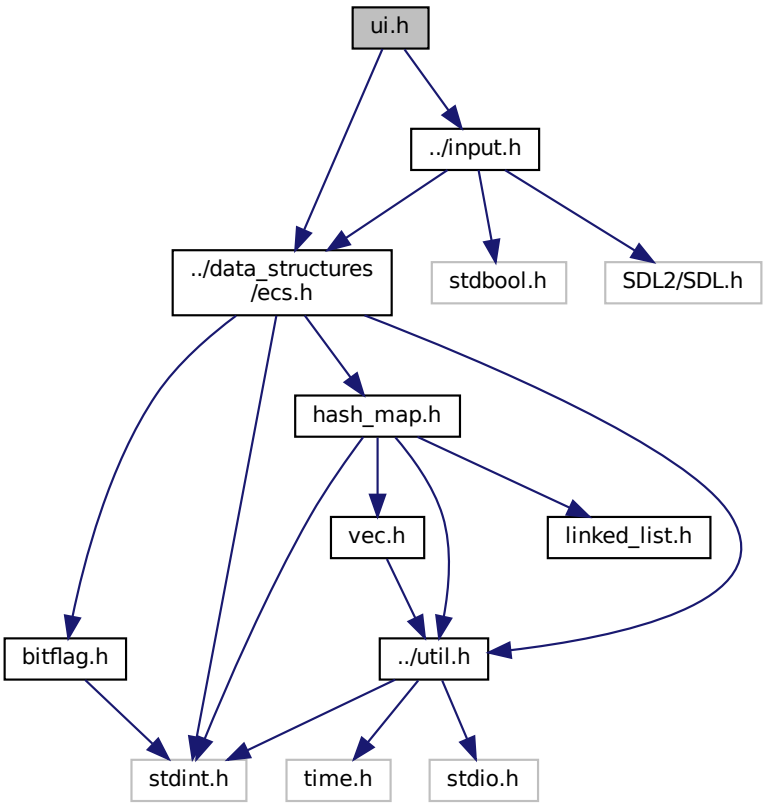
  returns the element with the smallest weight in the queue
- PQueue pqueue_push_inner (PQueue p, void ∗val, double weight)

  puts an element in the queue

6.13 ui.h File Reference

```
#include "../data_structures/ecs.h"
#include "../input.h"
#include "sprite.h"
```

Include dependency graph for ui.h:



Data Structures

- struct Text
- struct Background

  Entities with this component are the background of the user interface.
- struct Clickable
- struct Minimap

  Component that corresponds to the minimap.
- struct Hoverable

  Entities with this component show text when hovered.

Functions

- void render_ui (World *w, SDL_Renderer *rdr)

  Renders any entity that has user interface related component.
- Entity * spawn_clickable (World *w, Clickable *object, KeyEvent *event)

  Adds a clickable to the world.
- void clickable_event (World *w, SDL_Renderer *rdr, Entity *entity, Inputs *in, KeyState keystate)
- void render_hoverable (SDL_Rect *rect, char *text)
- void **hoverable_component_free** (void *tmp)
- void **minimap_component_free** (void *temp)
- void **background_component_free** (void *temp)
- void **clickable_component_free** (void *temp)
- void **text_component_free** (void *temp)
- void **null_function** ()

6.13.1 Function Documentation

6.13.1.1 clickable_event()

```
void clickable_event (
        World * w,
        SDL_Renderer * rdr,
        Entity * entity,
        Inputs * in,
        KeyState keystate )
```

The `KeyEvent` of the entities associated with a clickable component, there are different cases, if the mouse is out of the sprite, it set `is_clicked` to 0 as for doing nothing, if the left click is pressed on the sprite, it will be set to 1 and if it is set to 1 and the click is released then it will be set to 2. The idea is that if set to 1 there will be a visual change by darkening the sprite and if it set to 2 it will start the action linked to the sprite. It must be noted that if you click on the sprite, mouve your mouse out and then release the click it will do nothing as a way to correct missclicks.

6.13.1.2 render_hoverable()

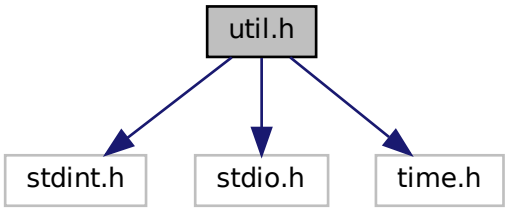
```
void render_hoverable (
        SDL_Rect * rect,
        char * text )
```

This function is used to render the entities associated with a hoverable component

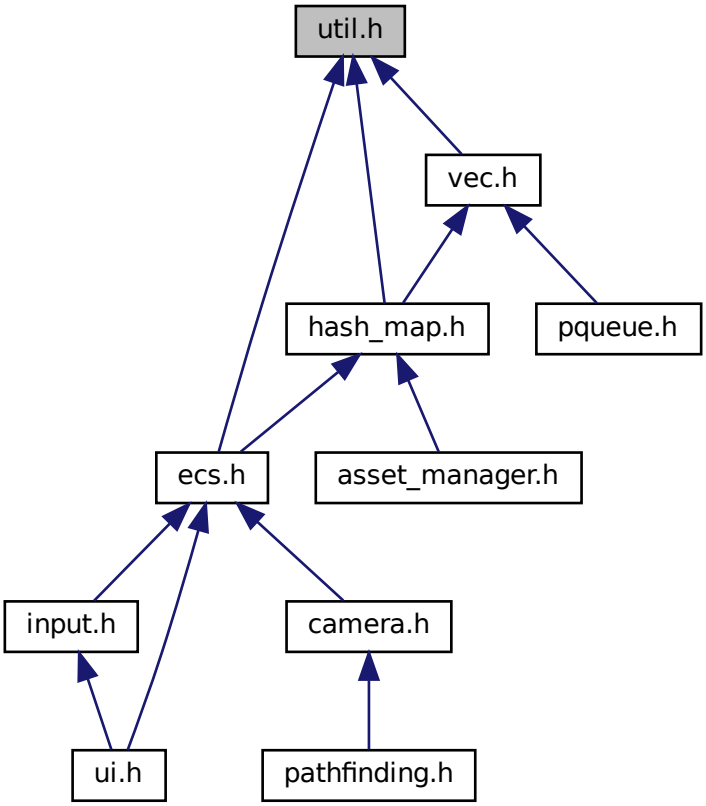
6.14 util.h File Reference

```
#include "errors.h"
#include <stdint.h>
#include <stdio.h>
#include <time.h>
```

Include dependency graph for util.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Vec2

  a 2d vector for use in units movement

Macros

- #define WARN(message)

  prints `message` as a warning
- #define HANDLE_ERROR(err, message, callback)

  prints `message` when `err != 0`, and then runs `callback`
- #define TARGET_FRAMETIME (1000.0 / 60.0)

  The frametime that the game should try to maintain, in milliseconds.
- #define WIN_H 360

  The main window's logical height.
- #define WIN_W 640

  The main window's logical width.
- #define ASSERT(a)
- #define TIME(label, commands)

  Benchmarks `commands`
- #define **max**(a, b) ((a > b) ? (a) : (b))
- #define **min**(a, b) ((a < b) ? (a) : (b))
- #define **v2op_dec**(name) Vec2 v2##name(Vec2 a, Vec2 b)

Typedefs

- typedef unsigned int **uint**

Functions

- void free_nothing (void *)

  Does nothing. Used when a callback is necessary but nothing is to be done.
- char not_strcmp (void *a, void *b)

  Strictly equivalent to `!strcmp(a,b)`. Used as a callback.
- void sleep_nano (uint64_t n)

  Sleeps the calling thread for n nanoseconds. Uses GNU extensions.
- v2op_dec (sub)

  substracts two `Vec2`
- v2op_dec (add)

  adds two `Vec2`
- Vec2 v2normalize (Vec2 a)

  normalizes a `Vec2`
- Vec2 v2mul (float a, Vec2 b)

  performs a scalar product between `Vec2 b` and `a`
- Vec2 v2div (Vec2 a, float b)

  performs a scalar product between `Vec2 a` and `1/b`
- float v2angle (Vec2 a)

  returns the angle (in radian) between `a` and the `(1,0)` vector
- float v2len (Vec2 a)

  returns the length of a `Vec2`
- Vec2 v2truncate (Vec2 a, float b)

  returns a vector of same direction than `a` and of length `max(v2len(a), b)`
- float v2dot (Vec2 a, Vec2 b)

  performs a dot product between two `Vec2`

6.14.1 Macro Definition Documentation

6.14.1.1 ASSERT `#define ASSERT(`

a)

Value:

```
{                                                                                   \
  if (!(a)) {                                                                       \
    fprintf(stderr, "[%s:%d] [\x1b[31mE\x1b[0m] assertion '%s' failed\n",           \
            __FILE__, __LINE__, #a);                                                \
    return ASSERTION_FAILED;                                                        \
  }                                                                                 \
}
```

Verify that `a != 0`. Otherwise, prints an error and exits the current function with error `-1`

6.14.1.2 HANDLE_ERROR `#define HANDLE_ERROR(`

err,
message,
callback)

Value:

```
{                                                                                   \
  if (err) {                                                                        \
    fprintf(stderr, "[%s:%d] [\x1b[31mE\x1b[0m] %s\n", __FILE__, __LINE__,         \
            message);                                                               \
    callback;                                                                       \
  }                                                                                 \
}
```

prints `message` when `err != 0`, and then runs `callback`

6.14.1.3 TIME `#define TIME(`

label,
commands)

Value:

```
{                                                                                   \
  struct timespec _beg;                                                             \
  clock_gettime(CLOCK_MONOTONIC, &_beg);                                            \
  commands;                                                                         \
  struct timespec _end;                                                             \
  clock_gettime(CLOCK_MONOTONIC, &_end);                                            \
  uint64_t elapsed = (uint64_t)(_end.tv_sec - _beg.tv_sec) * 1000000000 +           \
                     (uint64_t)(_end.tv_nsec - _beg.tv_nsec);                       \
  printf("[\033[38;5;96mBENCHMARKING\033[0m] %s took %fs\n", label,                 \
         (double)elapsed / 1000000000);                                             \
}
```

Benchmarks `commands`

6.14.1.4 WARN `#define WARN(`

message)

Value:

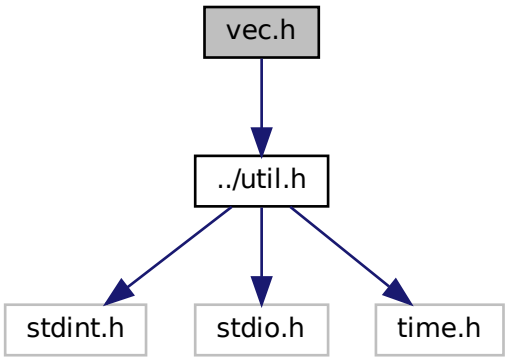
```
fprintf(stderr, "[%s:%d] [\x1b[33mW\x1b[0m] %s\n", __FILE__, __LINE__,          \
        message);
```

prints `message` as a warning

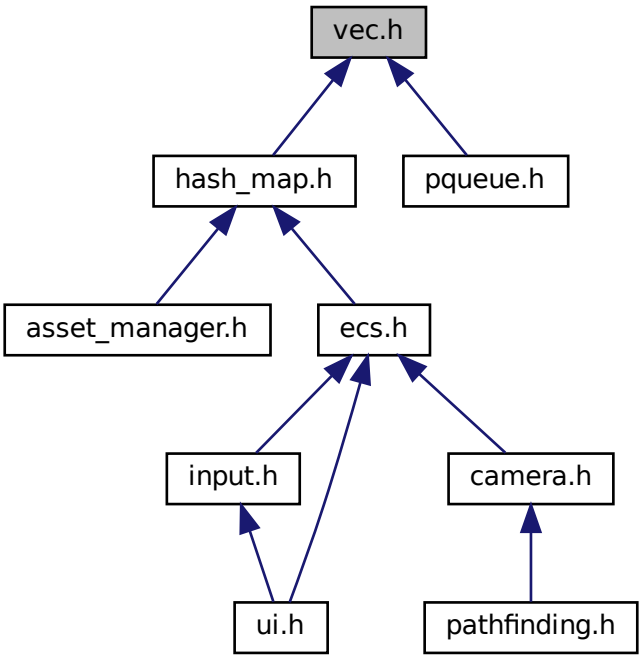
6.15 vec.h File Reference

```
#include "../util.h"
```

Include dependency graph for vec.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VEC(x) x *
- #define VEC_INIT_CAPACITY 16

  The length of a `vec` at creation.
- #define vec_new(type) (vec_new_inner(sizeof(type)))

  Creates a new `vec` for type `type`
- #define vec_push(vec, obj) vec = (vec_push_inner(((void *)(vec)), (void *)&(obj)))

  adds a copy of `obj` at the end of `vec`
- #define vec_last(a) (a)[vec_len((a)) - 1]

  expands to the last element of the `vec`

Functions

- VEC (void) vec_copy(VEC(void) vec)

  copies `vec` and returns the copy
- void vec_free (VEC(void) vec)

  frees a `vec`. This should always be used instead of `free(vec)`
- void vec_pop (VEC(void) vec)
- uint vec_len (VEC(void) vec)

  returns the length of `vec`. This is a `O(1)` operation.
- void vec_sort (VEC(void) vec, char(*gt)(void *a, void *b))
- void vec_swap (VEC(void) vec, int a, int b)

  swaps the elements at index `a` and `b` in `vec`
- char u64_gt (void *a, void *b)
- void vec_remove (void *vec, int a)

  removes element at index `a` in `vec`
- void vec_reverse (void *vec)

Variables

- void * **obj**

6.15.1 Detailed Description

file This file defines a redimensionnable array, hereafter reffered to as `vec`. Relevent informations about the content of the `vec` are stored just before the pointer that the user manipulates

6.15.2 Macro Definition Documentation

6.15.2.1 VEC `#define VEC( x ) x *`

A macro that extends to a pointer to `x`, to differentiate vectors from arbitrary pointers

6.15.3 Function Documentation

6.15.3.1 u64_gt() `char u64_gt (`

```
void * a,
void * b )
```

`a` and `b` are assumed to be `uint64_t`. returns true iff `&(uint64_t*)a>=&(uint64_t*)b`. Used for `vec_sort`

6.15.3.2 VEC() `VEC (`

```
void  )
```

copies `vec` and returns the copy

adds a copy of what `obj` points to at the end of `vec`. returns a potentially new pointer to the vec

6.15.3.3 vec_pop() `void vec_pop (`

```
VEC(void) vec )
```

removes the last element of the `vec`. Doesn't return it for optimisation purposes

6.15.3.4 vec_reverse() `void vec_reverse (`

```
void * vec )
```

reverses`vec` (i.e. old_vec[i] = new_vec[n-1-i] where n is the length of vec)

6.15.3.5 vec_sort() `void vec_sort (`

```
VEC(void) vec,
char(*)(void *a, void *b) gt )
```

sorts `vec` in place, using `gt` as a way to compare elements. `gt`'s parameters are pointers to the actually compared data, and `gt` returns true iff `a>=b`. `vec_sort` uses merge sort and is consequentially in `O(n*log(n))`

Index