# 2PII

Generated by Doxygen 1.9.1

# 1 Age of Sources

Age of Sources is a real time strategy game spiritually in line with games like Dune 2, Command and Conquer: Red Alert, and Warhammer 40k: Dawn of War II. It depicts a war between the tanuki civilization and U.N.E.C.T.A.S., a dangerous foreign entity which is after their resources.

To fight Unectas (the leader of U.N.E.C.T.A.S.), the player will have to build a town and a varied army of cute tanukis helped by their allies (including notably mounted penguins).

## 1.1 How to play

Use the arrow keys to move on the map, select your units with box selection, move them to your mouse's position with right click. Use a unit's ability when it's selected by left-clicking on the grid in the bottom-right of the screen. To win, destroy the ennemy's forum.

## 1.2 Design

### 1.2.1 Economy

Each player must pile up resources in the form of clay and water to build their town and army. Production of resources originates from wells and furnaces and is passive. Buildings can be constructed by beavers, that can be hired from the forum, which is the only building the players originally have.

### 1.2.2 Combat

Combat is based on the famous "Rock-Paper-Scissors" system, where each kind of unit is strong against one other type, and weak against one.

### 1.2.3 Find out more

More informations on the game's design can be found (in French) on: https://docs.google.com/document/d/1IPbxiTLj0bLq1hyJecttDWtiTuiY5Pz58Ut8i5y4QXE/edit?usp=sharing

## 1.3 Technical details

This game's code is heavily documented in the rest of this document. That being said, it still seems relevant to discuss here one of our technical choices:

### 1.3.1 On the use of an ECS

An Entity Component System (thereafter refered to as ECS) is a data structures commonly used in game engines, that stores entities and link them to components.
A Component can be any data structure, and an entity is just an abstraction to which is associated a number of components.
Not all entities have an instance of each component, and the main benefit of the ECS is that one can get all the entities that have a specific set of components in linear time.

We decided to use an ECS as it is a great tool to structure a game, and as some of us had previous experience implementing them and working with them.

### 1.4 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.

### 1.5 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/. If the site isn't available for some reason (Microsoft Azure), please contact Louis Buisson ( louis.buisson@telecomnancy.eu)

### 1.6 Acknowledgement

A game by Eliott Huet, Ghislain Mounier, Louis Buisson and Maxime Soldatov
Main menu artwork by Sacha Banak
Uses Fira Code Nerd Font provided under the SIL OPEN FONT LICENSE Version 1.1

## 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 Actionnable Struct Reference

`#include <actionnable.h>`

### Data Fields

- Action **act**
- EntityRef **target**

### 5.1.1 Detailed Description

A component that holds a unit's current action, i.e. what it's attacking/building

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

- actionnable.h

## 5.2 ActualisedText Struct Reference

`#include <ui.h>`

### Data Fields

- char *(* **get_text** )(World *w, Entity *e)
- SDL_Rect * **rect**
- SDL_Color * **color**

### 5.2.1 Detailed Description

Type used to render text that is not constant such as hp, sound volume or ressources.

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

- ui.h

## 5.3 Animator Struct Reference

A component that marks an entity as being subject to animations.

`#include <anim.h>`

### Data Fields

- uint **frame**
- SDL_Rect **current**
- AnimState state
- int max [3]

  *`max` indicates the last frame for each animation that is still valid*
- char flipped

  *wether the sprite must be vertically flipped*

### 5.3.1 Detailed Description

A component that marks an entity as being subject to animations.

#### 5.3.2 Field Documentation

##### 5.3.2.1 state `AnimState Animator::state`

The current state of the animation, `Noop` leads to undefined behavior (note however that this is an unreachable state unless you modify the component directly)

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

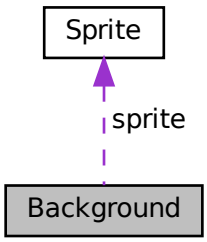
- anim.h

### 5.4 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**

#### 5.4.1 Detailed Description

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

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

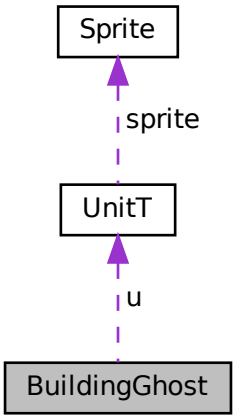
- ui.h

## 5.5 BuildingGhost Struct Reference

A component that serves as a building currently being built.

`#include <construction.h>`

Collaboration diagram for BuildingGhost:




### Data Fields

- UnitT * **u**
- int **progress**
- int **max**
- char **construction_done**
- UnitTypes **unit_type**

### 5.5.1 Detailed Description

A component that serves as a building currently being built.

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

- construction.h

## 5.6 Camera Struct Reference

`#include <camera.h>`

**Data Fields**

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

### 5.6.1 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

### 5.6.2 Field Documentation

#### 5.6.2.1 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

## 5.7 ClaySource Struct Reference

A component that flags an entity as a source of clay for its owner.

```
#include <players.h>
```

### 5.7.1 Detailed Description

A component that flags an entity as a source of clay for its owner.

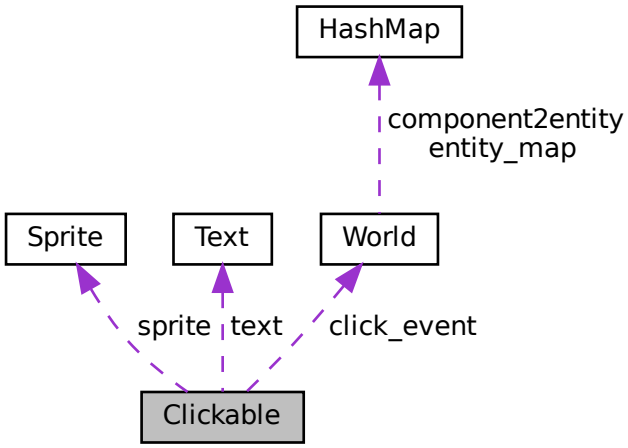
The documentation for this struct was generated from the following file:

- players.h

## 5.8 Clickable Struct Reference

`#include <ui.h>`

Collaboration diagram for Clickable:




### Data Fields

- Sprite * **sprite**
- SDL_Rect * **rect**
- Text * **text**
- Uint8 **is_clicked**
- ClickEvent **click_event**

### 5.8.1 Detailed Description

Entities with this type start an action when clicked on. The value of `is_clicked` depends on 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

## 5.9 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.*

### 5.9.1 Detailed Description

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

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

- src/data_structures/ecs.h

## 5.10 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.*

### 5.10.1 Detailed Description

The entity structure for the ECS.

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

- src/data_structures/ecs.h

## 5.11 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.*

### 5.11.1 Detailed Description

A hash map.

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

- src/data_structures/hash_map.h

## 5.12 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`*

### 5.12.1 Detailed Description

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

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

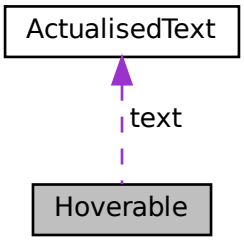
- src/data_structures/hash_map.h

## 5.13 Hoverable Struct Reference

Entities with this component show text when hovered.

`#include <ui.h>`

Collaboration diagram for Hoverable:




### Data Fields

- SDL_Rect * **rect**
- ActualisedText * **text**
- UnitTypes **t**

### 5.13.1 Detailed Description

Entities with this component show text when hovered.

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

- ui.h

## 5.14 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*

### 5.14.1 Detailed Description

stores keys and mouse buttons

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

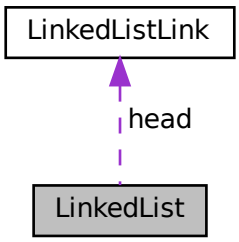
- src/input.h

## 5.15 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.*

### 5.15.1 Detailed Description

A singly linked list.

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

- src/data_structures/linked_list.h

## 5.16 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.*

### 5.16.1 Detailed Description

A link of `LinkedList`

### 5.16.2 Field Documentation

#### 5.16.2.1 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:

- src/data_structures/linked_list.h

## 5.17 MapComponent Struct Reference

`#include <map.h>`

#### Data Fields

- Map **map**

### 5.17.1 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

## 5.18 Minimap Struct Reference

Type that corresponds to the minimap.

`#include <ui.h>`

#### Data Fields

- SDL_Rect ∗ **rect**

### 5.18.1 Detailed Description

Type that corresponds to the minimap.

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

- ui.h

## 5.19 Ownership Struct Reference

A component indicating to which player a unit is belonging.

`#include <units.h>`

#### Data Fields

- char **owner**

### 5.19.1 Detailed Description

A component indicating to which player a unit is belonging.

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

- units.h

## 5.20 PlayerManager Struct Reference

A component that stores the current status of one of the players.

`#include <players.h>`

#### Data Fields

- int **id**
- int **water**
- int **dwater**
- int **clay**
- int **dclay**
- float **damage_multiplier**
- float **construct_multiplier**
- float **clay_multiplier**
- float **water_multiplier**

### 5.20.1 Detailed Description

A component that stores the current status of one of the players.

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

- players.h

## 5.21 Position Struct Reference

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

`#include <camera.h>`

#### Data Fields

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

### 5.21.1 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

## 5.22 PQueueEntry Struct Reference

an entry within a `PQueue`

`#include <pqueue.h>`

#### Data Fields

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

### 5.22.1 Detailed Description

an entry within a `PQueue`

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

- src/data_structures/pqueue.h

## 5.23 Renderer Struct Reference

`#include <util.h>`

### Data Fields

- SDL_Renderer * **r**

### 5.23.1 Detailed Description

This is a type that was created to be able to get the renderer from the world using the ecs.

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

- util.h

## 5.24 Selectable Struct Reference

`#include <selection.h>`

### Data Fields

- char **is_ghost**

### 5.24.1 Detailed Description

This component is a flag that marks something as selectable. A `Position` and a `Sprite` is still required so that we can now where and how big it is.

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

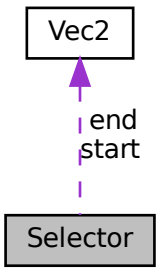
- selection.h

## 5.25 Selector Struct Reference

A component that manages selection.

`#include <selection.h>`

Collaboration diagram for Selector:



#### Public Member Functions

- **VEC** (EntityRef) selected

#### Data Fields

- SelectionType **type**
- Vec2 **start**
- Vec2 **end**
- char **is_selecting**
- char * **building**
- UnitTypes **building_utype**
- int **water_cost**
- int **clay_cost**

### 5.25.1 Detailed Description

A component that manages selection.

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

- selection.h

## 5.26 Sprite Struct Reference

A component that stores an entity's sprite.

`#include <sprite.h>`

#### Data Fields

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

### 5.26.1 Detailed Description

A component that stores an entity's sprite.

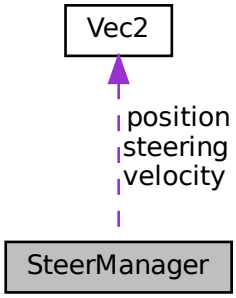
The documentation for this struct was generated from the following file:

- sprite.h

## 5.27 SteerManager Struct Reference

`#include <steering_behaviors.h>`

Collaboration diagram for SteerManager:




### Data Fields

- float **max_speed**
- float **max_force**
- float **mass**
- float **awareness**
- float **bounding_circle**
- float **rotation**
- Vec2 **velocity**
- Vec2 **position**
- Vec2 **steering**
- Path **current_path**

### 5.27.1 Detailed Description

The `SteerManager` is a componenent that manages movement for an entity according to steering behaviors as defined in *Steering Behaviors For Autonomous Characters* (see resources.md). `max_force` can be set to `INFINITY` (available in `math.h`) if steering should be instantaneous.

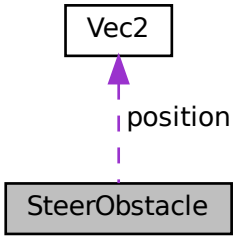
The documentation for this struct was generated from the following file:

- steering_behaviors.h

## 5.28 SteerObstacle Struct Reference

`#include <steering_behaviors.h>`

Collaboration diagram for SteerObstacle:




### Data Fields

- float bounding_circle
- Vec2 **position**

### 5.28.1 Detailed Description

The `SteerObstacle` is a component that indicates that an entity should be avoided when computing obstacle avoidance behaviors.

### 5.28.2 Field Documentation

#### 5.28.2.1 bounding_circle `float SteerObstacle::bounding_circle`

the radius of the circle that's considered as an obstacle. Should be the incircle of the sprite for units and the arithmetic mean of the incircle and the circumcircle for buildings.

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

- steering_behaviors.h

## 5.29 Text Struct Reference

`#include <ui.h>`

#### Data Fields

- char * **str**
- SDL_Color * **color**
- int **padding**

### 5.29.1 Detailed Description

Type that corresponds to the text that should be rendered on entities such as `Clickable`and `Hoverable`.

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

- ui.h

## 5.30 TilePosition Struct Reference

Stores the position of a tile.

`#include <pathfinding.h>`

#### Data Fields

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

### 5.30.1 Detailed Description

Stores the position of a tile.

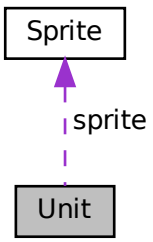
The documentation for this struct was generated from the following file:

- src/pathfinding.h

## 5.31 Unit Struct Reference

`#include <units.h>`

Collaboration diagram for Unit:



**Data Fields**

- char * name
- double hp

  *hp corresponds to the unit's current Health Points.*
- uint16_t **max_hp**
- uint16_t b_dam

  *b_dam corresponds to the unit's Blunt Damage.*
- uint16_t p_dam

  *p_dam corresponds to the unit's Piercing Damage.*
- uint16_t s_dam

  *s_dam corresponds to the unit's Special Defence*
- uint16_t b_def

  *b_def corresponds to the unit's Blunt Defence.*
- uint16_t p_def

  *p_def corresponds to the unit's Piercing Defence.*
- uint16_t s_def

  *s_def corresponds to the unit's Special defence.*
- uint16_t rg

  *rg corresponds to the unit's Range.*
- uint16_t sp

  *sp corresponds to the unit's Speed.*
- Sprite * sprite

  *sprite corresponds to the unit's Sprite*
- char * path_to_sprite

  *Self explanatory.*
- char * descr
- UnitTypes **t**

### 5.31.1 Detailed Description

The Unit type describes every unit and building in the game. Every number type data must be smaller or equal than 65535. A building is differentiated from a soldier/builder because a building's sp is 0.

### 5.31.2 Field Documentation

#### 5.31.2.1 descr `char* Unit::descr`

descr matches the unit's description, it must not be more than 65535 characters long not including '\0', it must not contain '*'.

#### 5.31.2.2 name `char* Unit::name`

name matches the unit's name, it must not be more than 255 characters long not including '\0', it must not contain '*'.

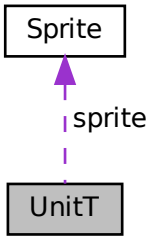
The documentation for this struct was generated from the following file:

- units.h

## 5.32 UnitT Struct Reference

`#include <units.h>`

Collaboration diagram for UnitT:




### Data Fields

- char * **name**
- uint16_t **hp**
- uint16_t **b_dam**
- uint16_t **p_dam**
- uint16_t **s_dam**
- uint16_t **b_def**
- uint16_t **p_def**
- uint16_t **s_def**
- uint16_t **rg**
- uint16_t **sp**
- Sprite * **sprite**
- char * **path_to_sprite**
- char * **descr**
- UnitTypes **t**

#### 5.32.1 Detailed Description

A type similar to `Unit`, it's use is to be stored in the asset manager and be copied to create a `Unit`.

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

- units.h

## 5.33 Vec2 Struct Reference

A 2d vector used for the units'movement.

`#include <util.h>`

#### Data Fields

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

### 5.33.1 Detailed Description

A 2d vector used for the units'movement.

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

- util.h

## 5.34 WaterSource Struct Reference

A component that flags an entity as a source of water for its owner.

`#include <players.h>`

### 5.34.1 Detailed Description

A component that flags an entity as a source of water for its owner.

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

- players.h

## 5.35 Window Struct Reference

`#include <util.h>`

#### Data Fields

- SDL_Window * **w**

### 5.35.1 Detailed Description

This is a type that was created to be able to get the window from the world using the ecs.

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

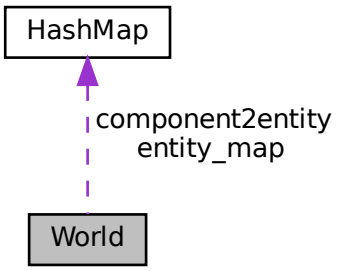
- util.h

## 5.36 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 (uint64_t) component_sparsity

  *Stores the available spaces in `components` that entity deletion created.*
- VEC (uint64_t) 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.*

### 5.36.1 Detailed Description

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

### 5.36.2 Member Function Documentation

#### 5.36.2.1 VEC() `World::VEC (`

`uint )`

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

#### 5.36.2.2 void() `World::void (`

`VEC() * component_free )`

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

### 5.36.3 Field Documentation

#### 5.36.3.1 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

#### 5.36.3.2 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

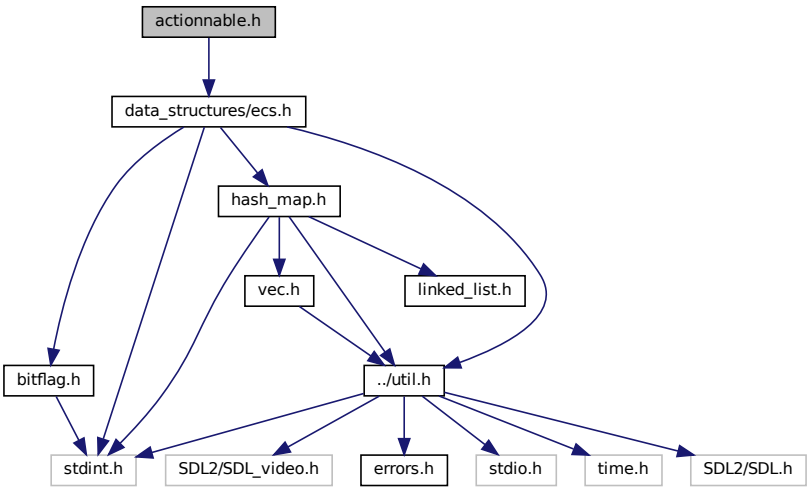
The documentation for this struct was generated from the following file:

- src/data_structures/ecs.h

# 6 File Documentation

## 6.1 actionnable.h File Reference

`#include "data_structures/ecs.h"`
Include dependency graph for actionnable.h:



### Data Structures

- struct Actionnable

### Enumerations

- enum Action { Lazy , Build , Attack , Produce }

  *The action performed by a unit.*

### Functions

- char actionnate (World ∗w, Actionnable ∗ac, Entity ∗se)

### 6.1.1 Enumeration Type Documentation

#### 6.1.1.1 Action `enum Action`

The action performed by a unit.

**Enumerator**

| | |
|---|---|
| Lazy | The unit does nothing. |
| Build | The unit is building something. |
| Attack | The unit is attacking something. |
| Produce | The unit is producing something. |

### 6.1.2 Function Documentation

#### 6.1.2.1 actionnate() `char actionnate (`

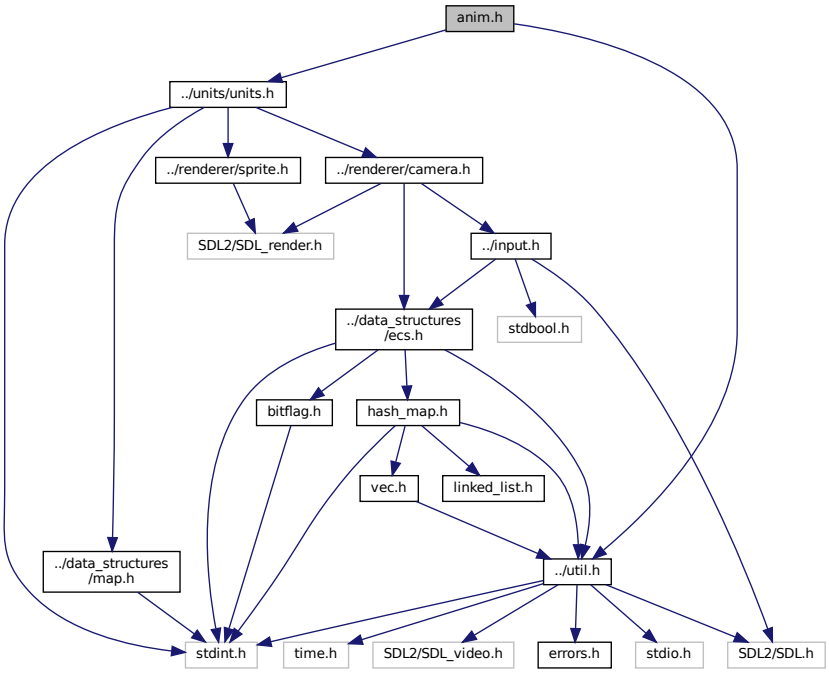
```
World * w,
Actionnable * ac,
Entity * se )
```

Performs the action specified in `ac` on the target. Returns 1 if an action is being performed, 0 otherwise

## 6.2 anim.h File Reference

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

Include dependency graph for anim.h:




### Data Structures

- struct Animator

  *A component that marks an entity as being subject to animations.*

### Macros

- #define ANIM_STEP 20

  *The number of renderer frames for which each animation frame is displayed.*

### Enumerations

- enum AnimState { **Idle** , **Moving** , **Attacking** , **Noop** }

  *The state of an animation, i.e. the kind of movement that is being animated.*

### Functions

- void advance_anim_state (Animator ∗a, AnimState as, char flipped)
- Animator animator_new (Unit ∗unit_kind)

  *Builds a new animator for a specific unit.*

#### 6.2.1 Function Documentation

#### 6.2.1.1 advance_anim_state()

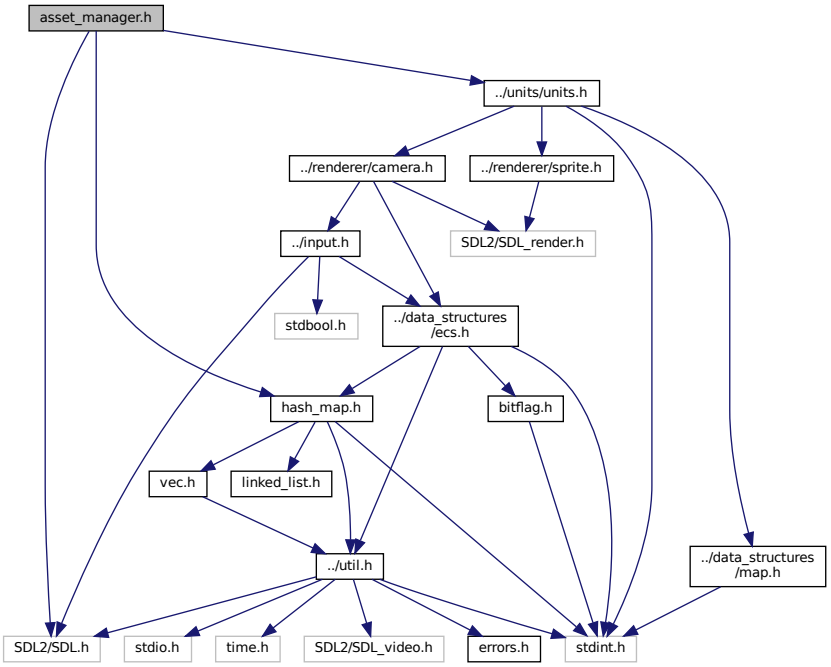
```
void advance_anim_state (
        Animator * a,
        AnimState as,
        char flipped )
```

Runs an animation step on the animator, changing the frame if required, and changing the state if `as` isn't `Noop`. Also changes the flipped status of the Animator if `flipped != -1`

## 6.3 asset_manager.h File Reference

```
#include <SDL2/SDL.h>
#include "../units/units.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)

  *Returns true iff an asset is locked. See `lock_asset` for more details.*
- 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)

  *Remove a texture from the `ASSET_STORE` and free it.*
- void * get_audio (char *t, char is_mus)
- void * load_audio (char *t, char is_mus)
- int drop_audio (char *t)

*Frees the audio of the file `t` in the `ASSET_STORE`.*

- void * load_font (char *t, Uint8 size)
- void * get_font (char *t, Uint8 size)
- int drop_font (char *font, Uint8 size)

  *Remove a font from the `ASSET_STORE` and free it.*
- void * load_unit (UnitTypes t, SDL_Renderer *renderer, SDL_Window *window)
- void * get_unit (UnitTypes t, SDL_Renderer *renderer, SDL_Window *window)
- int drop_unit (UnitTypes *t)

  *Remove a unit from the `ASSET_STORE` and free it.*
- void free_asset_store ()

  *Self explanatory.*

### Variables

- HashMap ASSET_STORE

  *Stores and manages the textures used in the game.*

## 6.3.1 Function Documentation

### 6.3.1.1 get_audio()

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

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

### 6.3.1.2 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.

### 6.3.1.3 get_texture()

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

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

### 6.3.1.4 get_unit()

```
void* get_unit (
        UnitTypes t,
        SDL_Renderer * renderer,
        SDL_Window * window )
```

Returns a pointer to the UnitT of the UnitTypes `t`. Will add it to the `ASSET_STORE` if it is not in it yet.

#### 6.3.1.5 init_asset_manager() `void init_asset_manager ( )`

Initializes the `ASSET_STORE`; must be called before any call to a `get_*` function or a `load_*` function.

#### 6.3.1.6 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.

#### 6.3.1.7 load_font()

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

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

#### 6.3.1.8 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.

#### 6.3.1.9 load_unit()

```
void* load_unit (
        UnitTypes t,
        SDL_Renderer * renderer,
        SDL_Window * window )
```

Loads the unit of the UnitTypes `t` in the `ASSET_STORE` by parsing the file that matches it. 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

#### 6.3.1.10 lock_asset()

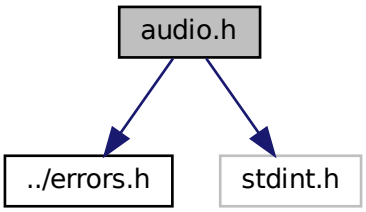
```
Error lock_asset (
        char * t,
        char locked )
```

Change the locked status of an asset to `locked`. A locked asset isn't dropped by the `drop_*` functions even when it is not used anymore.

## 6.4 audio.h File Reference

```
#include "../errors.h"
#include <stdint.h>
```

Include dependency graph for audio.h:




### Functions

- Error play_audio (char *path, char is_music)

  *Plays sound* `path`
- void set_volume (uint8_t volume)
- uint8_t get_volume ()

  *Returns the current volume of the audio.*
- void toggle_music ()

  *Starts and stops the music.*

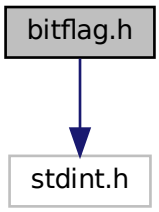
### 6.4.1 Function Documentation

#### 6.4.1.1 set_volume() `void set_volume ( uint8_t volume )`

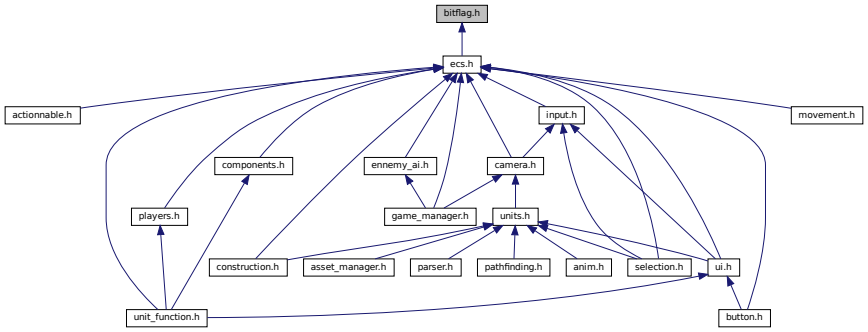
Sets the audio volume. Volume must be between 0 and 128. Anything higher than that is clamped.

## 6.5 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`*

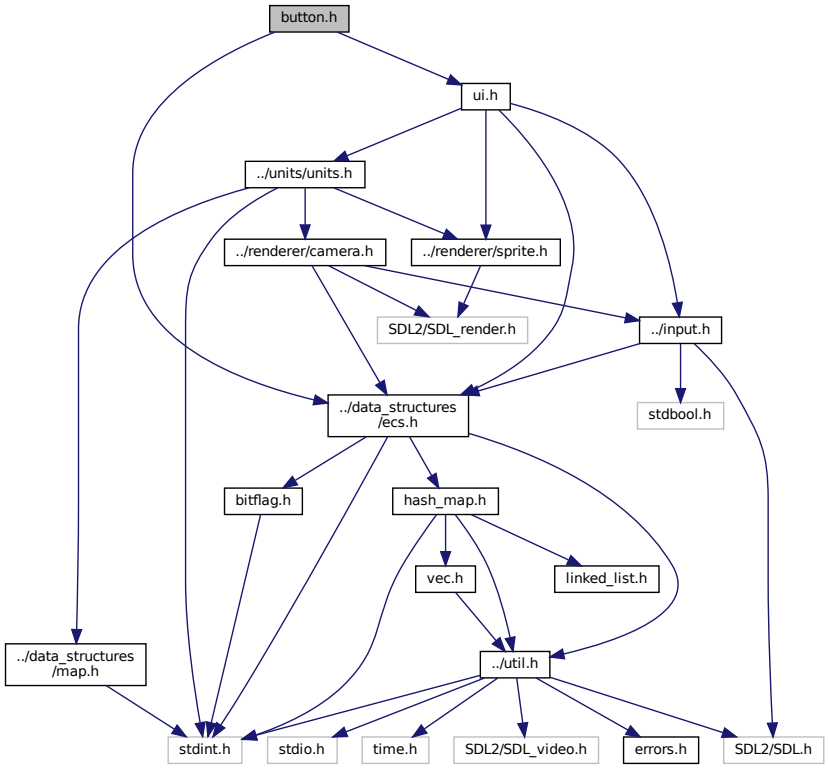
### Typedefs

- typedef uint64_t **Bitflag**

## 6.6 button.h File Reference

```
#include "../data_structures/ecs.h"
#include "ui.h"
```

Include dependency graph for button.h:




### Functions

- Clickable ∗ spawn_button (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window, void(∗event)(World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window), char ∗t, int xp, int yp)
- void spawn_main_menu (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window)

  *Creates the main menu.*
- void key_event_escape (World ∗w, SDL_Renderer ∗rdr, Entity ∗entity, Inputs ∗in, KeyState keystate)

  *Key_event that manages the escape button.*
- char ∗ str_sound_level (World ∗w, Entity ∗e)
- void spawn_victory (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window)

  *Displays the victory screen.*
- void spawn_defeat (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window)

  *Displays the defeat screen.*

#### 6.6.1 Function Documentation

#### 6.6.1.1 spawn_button()

```
Clickable* spawn_button (
        World * w,
        SDL_Renderer * renderer,
        SDL_Window * window,
        void(*)(World *w, SDL_Renderer *renderer, SDL_Window *window) event,
        char * t,
        int xp,
        int yp )
```

This function is used to add clickable in the world with the `click_event event`, has the text `t` and it's upper left corner is at the coordinates `xp yp`.

#### 6.6.1.2 str_sound_level()

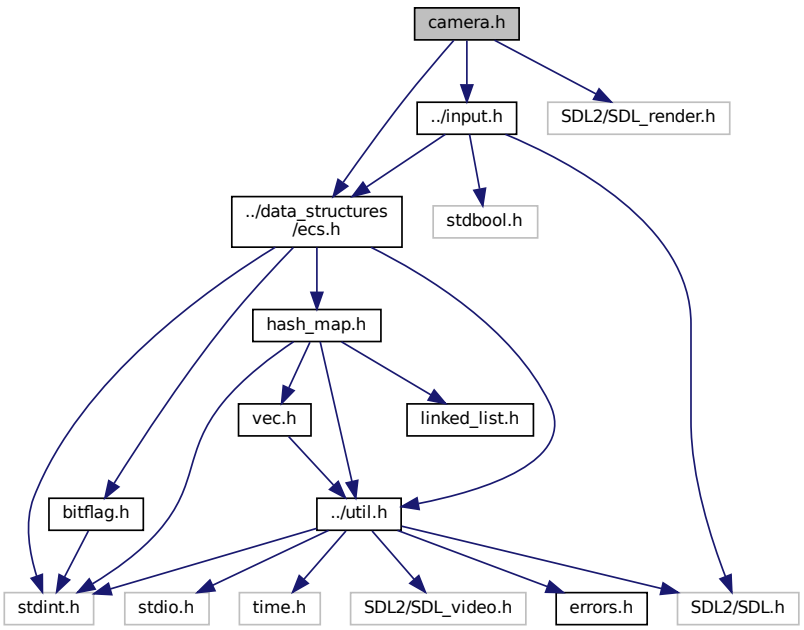
```
char* str_sound_level (
        World * w,
        Entity * e )
```

Function that returns the sound level as a string of the format "`sound`/128", for a volume of 37, it returns "37/128".

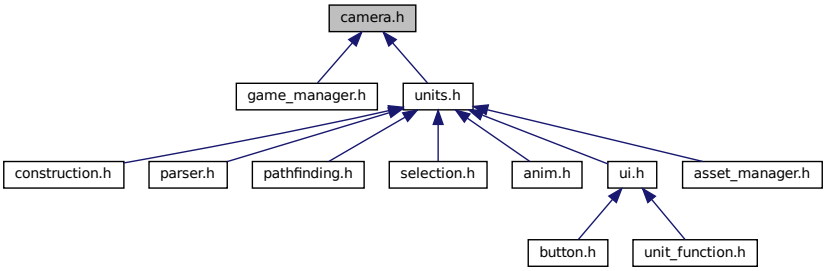
## 6.7 camera.h File Reference

```
#include "../data_structures/ecs.h"
#include "../input.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)
- void map_movement (World *w, SDL_Renderer *, Entity *e, Inputs *i, KeyState st)

  *A* `KeyEvent` *that moves the camera around.*

### 6.7.1 Function Documentation

#### 6.7.1.1 render()

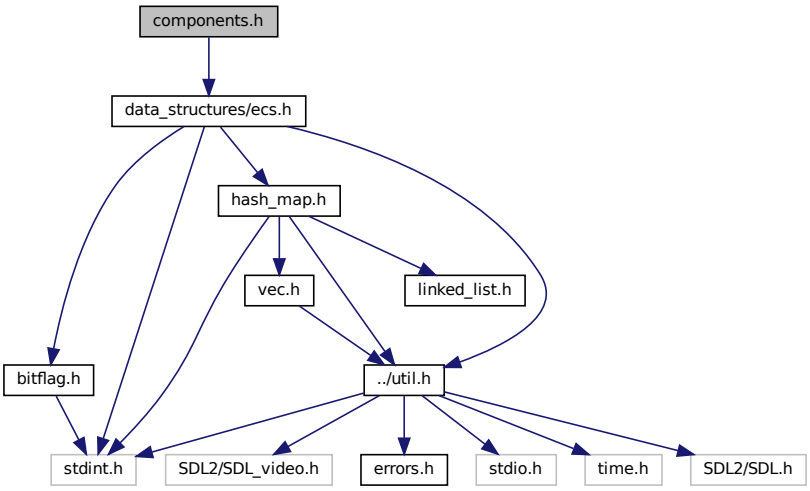
```
void render (
        World * w,
        SDL_Renderer * rdr,
        Camera * cam,
        SDL_Window * window )
```

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.

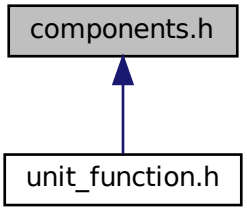
## 6.8 components.h File Reference

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

Include dependency graph for components.h:




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




### Macros

- #define **COMPF_POSITION** (1 << 0)
- #define **COMP_POSITION** 0
- #define **COMPF_SPRITE** (1 << 1)
- #define **COMP_SPRITE** 1
- #define **COMPF_KEY_EVENT** (1 << 2)
- #define **COMP_KEY_EVENT** 2
- #define **COMPF_BACKGROUND** (1 << 3)
- #define **COMP_BACKGROUND** 3
- #define **COMPF_CLICKABLE** (1 << 4)
- #define **COMP_CLICKABLE** 4

- #define **COMPF_MINIMAP** (1 << 5)
- #define **COMP_MINIMAP** 5
- #define **COMPF_HOVERABLE** (1 << 6)
- #define **COMP_HOVERABLE** 6
- #define **COMPF_MAPCOMPONENT** (1 << 7)
- #define **COMP_MAPCOMPONENT** 7
- #define **COMPF_STEERMANAGER** (1 << 8)
- #define **COMP_STEERMANAGER** 8
- #define **COMPF_STEEROBSTACLE** (1 << 9)
- #define **COMP_STEEROBSTACLE** 9
- #define **COMPF_UNIT** (1 << 10)
- #define **COMP_UNIT** 10
- #define **COMPF_TEXT** (1 << 11)
- #define **COMP_TEXT** 11
- #define **COMPF_ANIMATOR** (1 << 12)
- #define **COMP_ANIMATOR** 12
- #define **COMPF_CAMERA** (1 << 13)
- #define **COMP_CAMERA** 13
- #define **COMPF_SELECTABLE** (1 << 14)
- #define **COMP_SELECTABLE** 14
- #define **COMPF_SELECTOR** (1 << 15)
- #define **COMP_SELECTOR** 15
- #define **COMPF_PLAYERMANAGER** (1 << 16)
- #define **COMP_PLAYERMANAGER** 16
- #define **COMPF_WINDOW** (1 << 17)
- #define **COMP_WINDOW** 17
- #define **COMPF_ACTUALISEDTEXT** (1 << 18)
- #define **COMP_ACTUALISEDTEXT** 18
- #define **COMPF_BUILDINGGHOST** (1 << 19)
- #define **COMP_BUILDINGGHOST** 19
- #define **COMPF_OWNERSHIP** (1 << 20)
- #define **COMP_OWNERSHIP** 20
- #define **COMPF_ACTIONNABLE** (1 << 21)
- #define **COMP_ACTIONNABLE** 21
- #define **COMPF_WATERSOURCE** (1 << 22)
- #define **COMP_WATERSOURCE** 22
- #define **COMPF_CLAYSOURCE** (1 << 23)
- #define **COMP_CLAYSOURCE** 23
- #define **COMP_RENDERER** 24
- #define **COMPF_RENDERER** (1 << 24)

### Functions

- int init_world (World *w)

### 6.8.1 Function Documentation

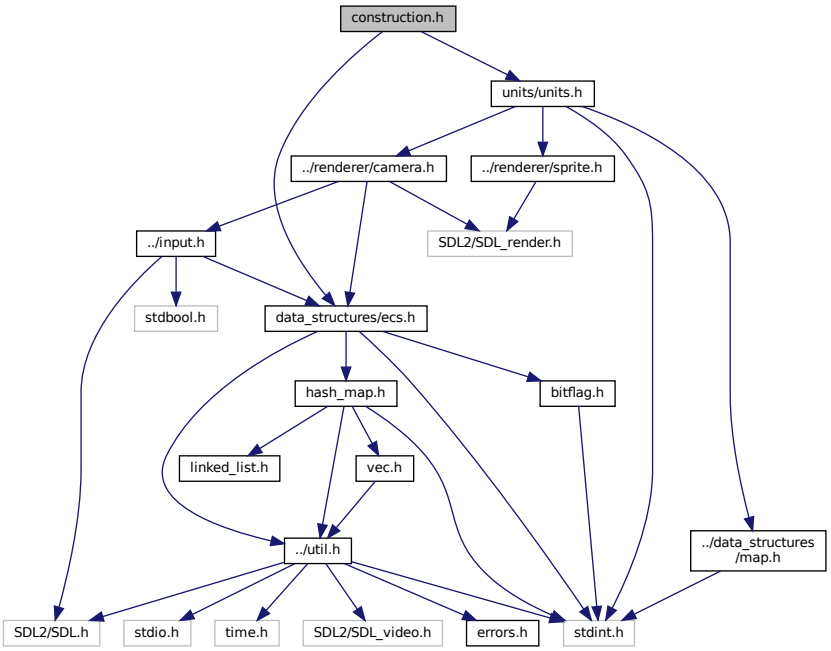
#### 6.8.1.1 init_world() `int init_world ( World * w )`

Initializes the components required by the game. Must be called exactly once before using the world.

## 6.9 construction.h File Reference

```
#include "data_structures/ecs.h"
#include "units/units.h"
```

Include dependency graph for construction.h:




### Data Structures

- struct BuildingGhost

  *A component that serves as a building currently being built.*

### Functions

- void building_ghost_component_free (void *)

  *Self explanatory.*
- void finish_construction (World *w, Entity *e)

### 6.9.1 Function Documentation

#### 6.9.1.1 finish_construction()

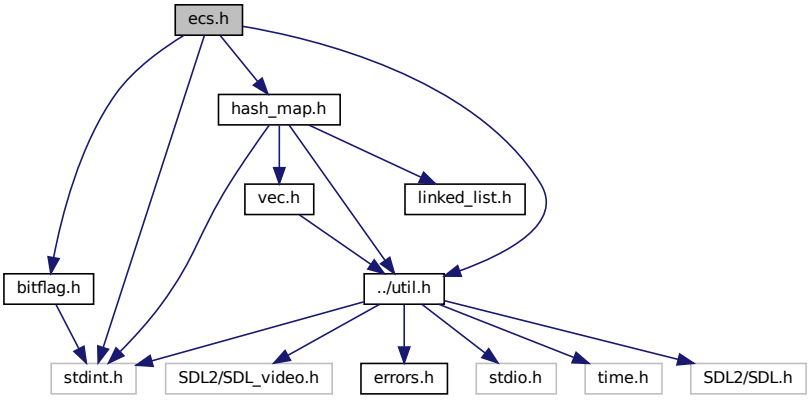
```
void finish_construction (
        World * w,
        Entity * e )
```

Transforms a building's ghost into a proper building (typically, would be called when it's finished being constructed)

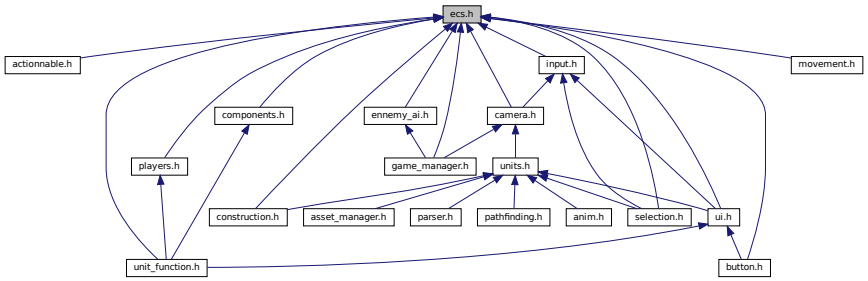
## 6.10 ecs.h File Reference

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

Include dependency graph for src/data_structures/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)
- EntityRef get_next_entity_ref (World *w)

### Variables

- Bitflag * **b**

## 6.10.1 Macro Definition Documentation

### 6.10.1.1 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

#### 6.10.1.2 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

#### 6.10.1.3 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

### 6.10.2 Typedef Documentation

#### 6.10.2.1 EntityRef `typedef uint64_t EntityRef`

A representation of an `Entity`. this note was left in its implementation : "Note that this reference is only valid until the number of entities decreases", but I'm pretty sure it's not true

### 6.10.3 Function Documentation

#### 6.10.3.1 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)

#### 6.10.3.2 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

#### 6.10.3.3 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`

#### 6.10.3.4 get_next_entity_ref()

```
EntityRef get_next_entity_ref (
        World * w )
```

Returns the `EntityRef` of an entity that would be created just after the call to this function

#### 6.10.3.5 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

#### 6.10.3.6 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.

#### 6.10.3.7 VEC()

```
VEC (
        EntityRef  )
```

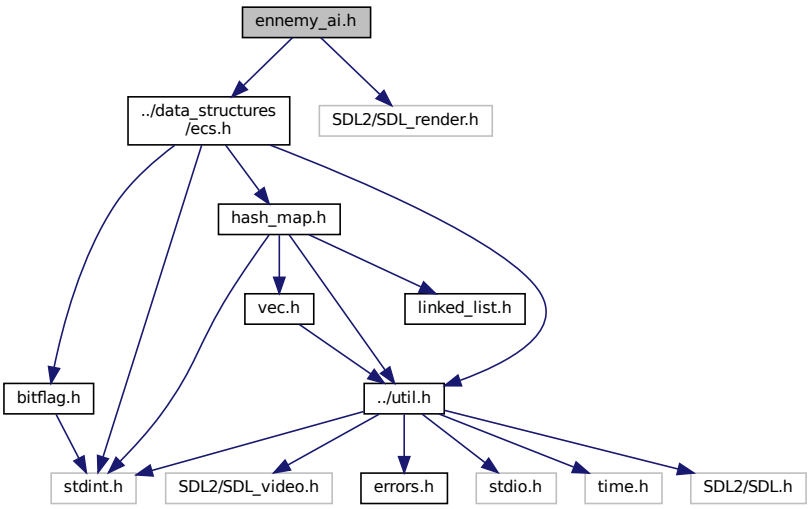
Returns a vector of `EntityRef` referencing entities corresponding to the system described by the `Bitflag` argument. If you want to modify the `World` based on the return value of this function, use `world_query_mut` instead. The system needs to be registered using `register_system_requirement` before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

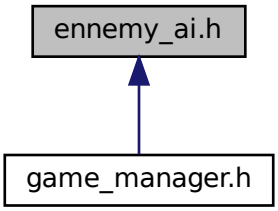
## 6.11 ennemy_ai.h File Reference

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

Include dependency graph for ennemy_ai.h:



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




### Enumerations

- enum AiState { Eco , Offense , Defense }

### Functions

- void reconsider_ai_state (World *w, AiState *ais)

  *Reconsiders the current `AiState` depending on the current state of the game.*
- char is_ai_attacked (World *w)

  *Returns true iff ai is being attacked.*
- void take_ai_action (World *w, AiState *ais, SDL_Renderer *renderer, SDL_Window *window)

  *Cause the ai to act.*
- void ai_defends_itself (World *w)

  *Hints the ai to fight imminent threats to its units.*
- void deghost (World *w)

  *Hints the ai to delete unused ghost buildings.*

## 6.11.1 Enumeration Type Documentation

### 6.11.1.1 AiState `enum AiState`

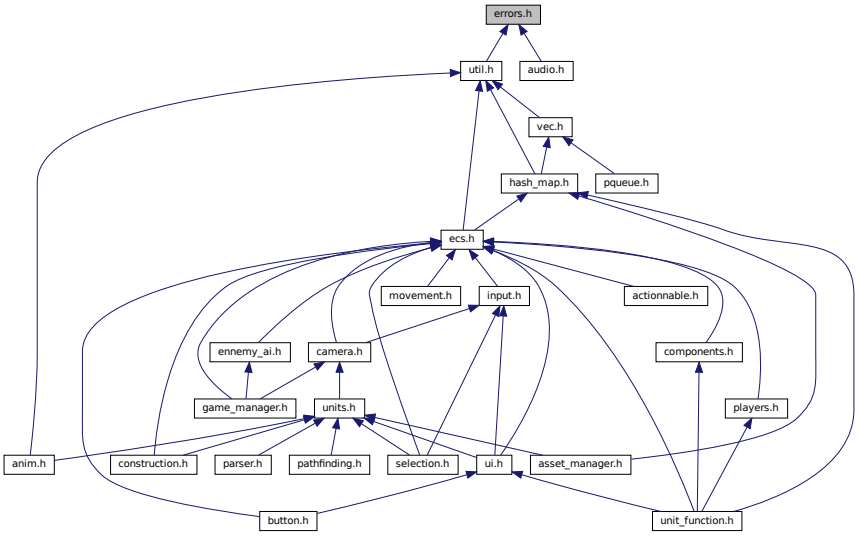
The current state the ai is in, i.e., which behavior it should adopt. This is essentially a finite state machine

**Enumerator**

| | |
|---|---|
| Eco | The ai focuses on strengthening its economy. |
| Offense | The ai focuses on attacking its opponent. |
| Defense | The ai focuses on repsonding to an immediate threat. |

## 6.12 errors.h File Reference

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




### Enumerations

- enum Error {
  SUCCESS = 0 , OUT_OF_MEMORY , INDEX_OUT_OF_RANGE , ASSET_NOT_FOUND ,
  ASSERTION_FAILED , COULD_NOT_MIX_SOUND }

  *a general error return type*

### 6.12.1 Enumeration Type Documentation

#### 6.12.1.1 Error `enum Error`

a general error return type

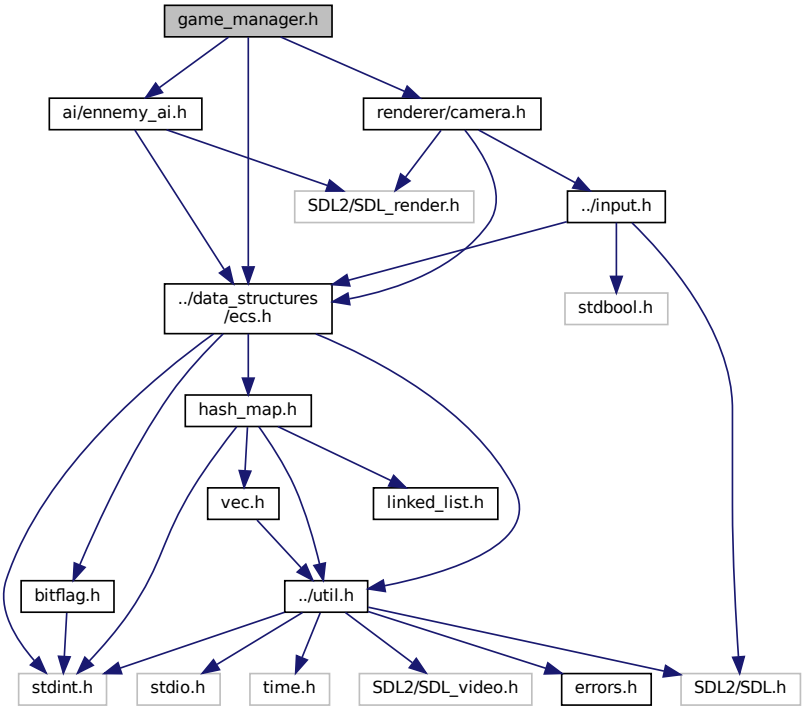
**Enumerator**

| | |
|---|---|
| SUCCESS | self explanatory |
| OUT_OF_MEMORY | function call couldn't succeed because of insufficiant memory in the heap |
| INDEX_OUT_OF_RANGE | function was called with invalid parameters leading to access out of the allowed range |
| ASSET_NOT_FOUND | asset doesn't exist, i.e. it is not loaded in the asset manager and there is no file corresponding to the requested asset |
| ASSERTION_FAILED | self explanatory |
| COULD_NOT_MIX_SOUND | self explanatory |

## 6.13 game_manager.h File Reference

```
#include "ai/ennemy_ai.h"
#include "data_structures/ecs.h"
#include "renderer/camera.h"
```

Include dependency graph for game_manager.h:




### Functions

- void new_game (World *w, SDL_Renderer *renderer, SDL_Window *window, Camera *cam, AiState *ais)

  *Initiates a game (i.e. creates the map, places the required units, ...)*
- void revert_game (World *w)

### 6.13.1 Function Documentation

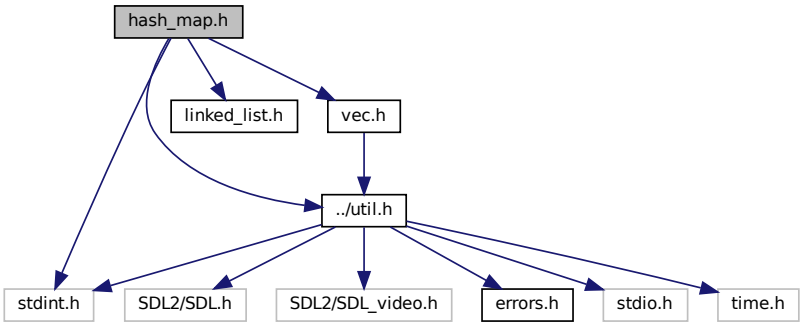
#### 6.13.1.1 revert_game() `void revert_game ( World * w )`

Reverts any changes to the world made during a game so a new one can be created

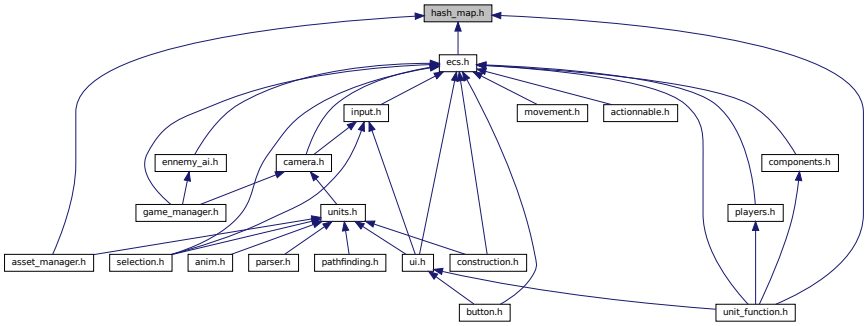
## 6.14 hash_map.h File Reference

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

Include dependency graph for src/data_structures/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)
- void hash_map_entry_free_keys (void *u)

  *A `hash_map_free_callback` callback that dosn't free the values.*

#### 6.14.1 Function Documentation

##### 6.14.1.1 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

##### 6.14.1.2 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

##### 6.14.1.3 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

#### 6.14.1.4 hash_map_insert_callback()

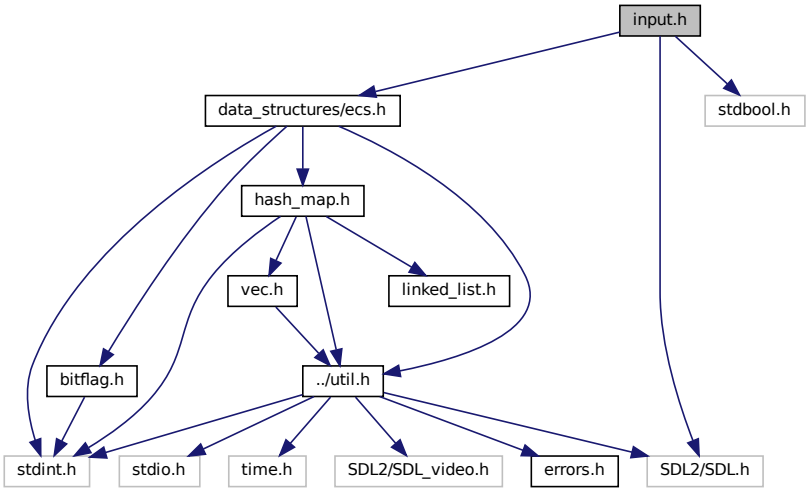
```
int hash_map_insert_callback (
        HashMap * h,
        void * k,
        void * v,
        void(*)(void *) callback )
```

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

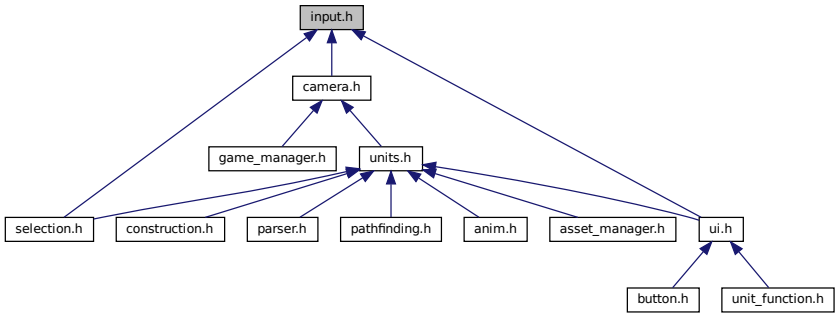
## 6.15 input.h File Reference

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

Include dependency graph for src/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*
- SDL_Point get_mouse_position (SDL_Renderer *rdr)
- Uint8 mouse_in_rect (SDL_Renderer *rdr, SDL_Rect *rect)

  *Checks if the mouse is in the rectangle.*

## 6.15.1 Macro Definition Documentation

### 6.15.1.1 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)

### 6.15.1.2 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)

#### 6.15.1.3 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)

#### 6.15.1.4 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)

#### 6.15.1.5 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)

### 6.15.2 Typedef Documentation

#### 6.15.2.1 MouseButton

```
typedef Uint8 MouseButton
```

describes any of the following: SDL_BUTTON_LEFT,SDL_BUTTON_MIDDLE, SDL_BUTTON_RIGHT

### 6.15.3 Function Documentation

#### 6.15.3.1 get_mouse_position()

```
SDL_Point get_mouse_position (
        SDL_Renderer * rdr )
```

Returns the mouse position in the scaled coordinates of the integer scaled window

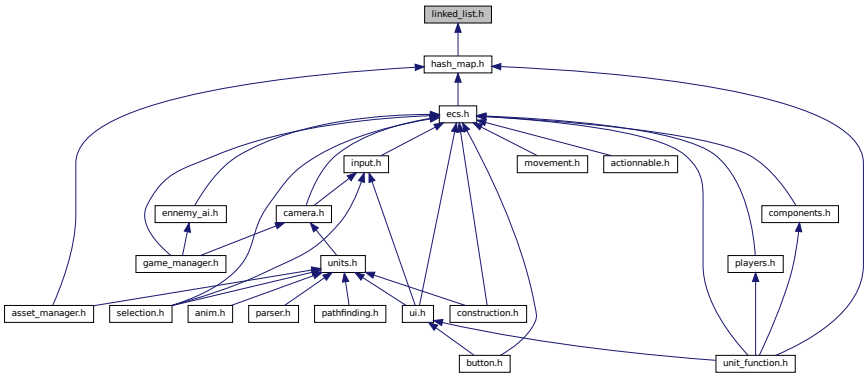
#### 6.15.3.2 inputs_update_key_in_from_scancode()

```
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)

## 6.16 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`

### 6.16.1 Function Documentation

#### 6.16.1.1 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

#### 6.16.1.2 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

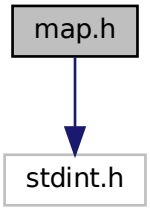
#### 6.16.1.3 linked_list_remove_callback()

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

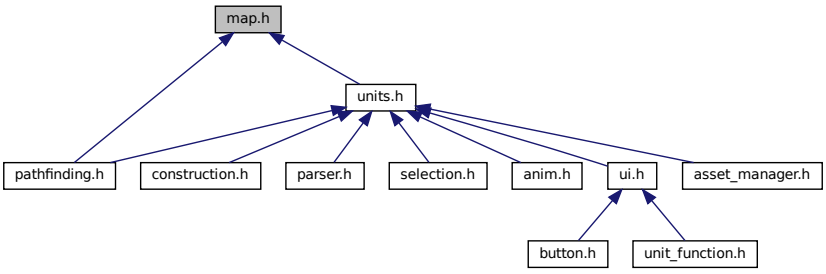
Removes element at index `i` in `l`, running `callback` on its `data` as a way to free it

## 6.17 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_SWAMP** , **TILE_WATER** ,
  **TILE_MOUNTAIN** , **TILE_GIGEMENT** , **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 (TileTypes tt)
- TileTypes pixel2tiletype (int8_t id)

  *Takes a pixel `id` and return the TileTypes that matches it.*
- int8_t tiletype2pixel (TileTypes id)

  *Takes a TileTypes `id` and return a pixel that matches it.*

## 6.17.1 Typedef Documentation

### 6.17.1.1 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...)

## 6.17.2 Function Documentation

#### 6.17.2.1 get_tile_file_name() `char* get_tile_file_name (TileTypes tt )`

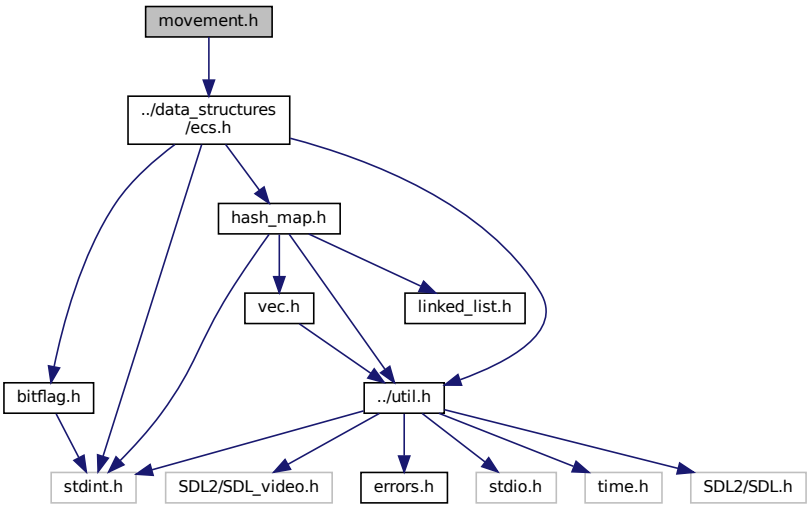
Returns the file name associated with a certain color. Return value should be freed.

#### 6.17.2.2 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.

## 6.18 movement.h File Reference

`#include "../data_structures/ecs.h"`
Include dependency graph for movement.h:




### Functions

- void move_units (World *w)

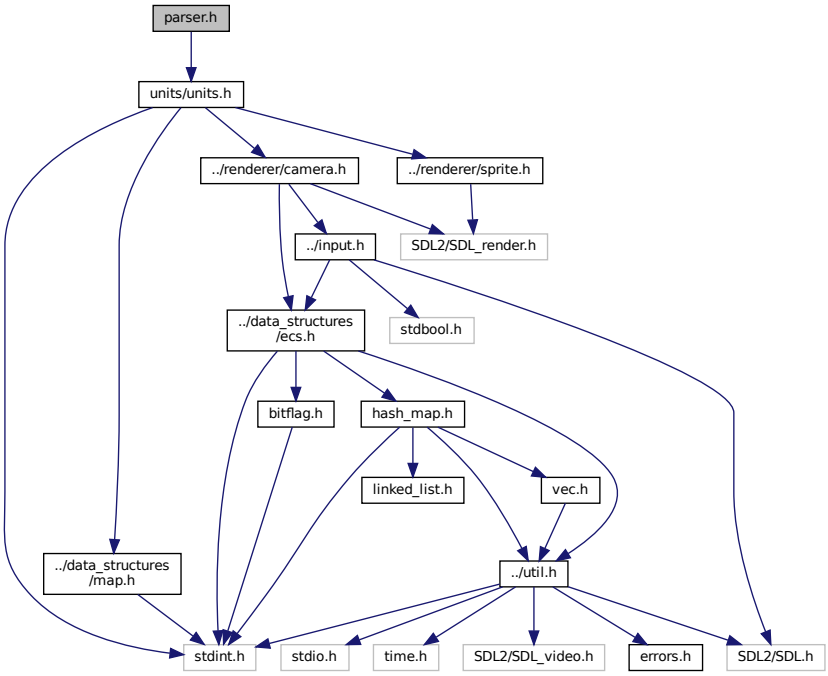
### 6.18.1 Function Documentation

#### 6.18.1.1 move_units() `void move_units (World * w )`

apply the required steering behaviors to move all the units in `w` that have a `SteeringManager` and a `Position`

## 6.19 parser.h File Reference

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




### Functions

- UnitT ∗ parse (char ∗path, SDL_Renderer ∗renderer, SDL_Window ∗window)

### 6.19.1 Function Documentation

#### 6.19.1.1 parse()

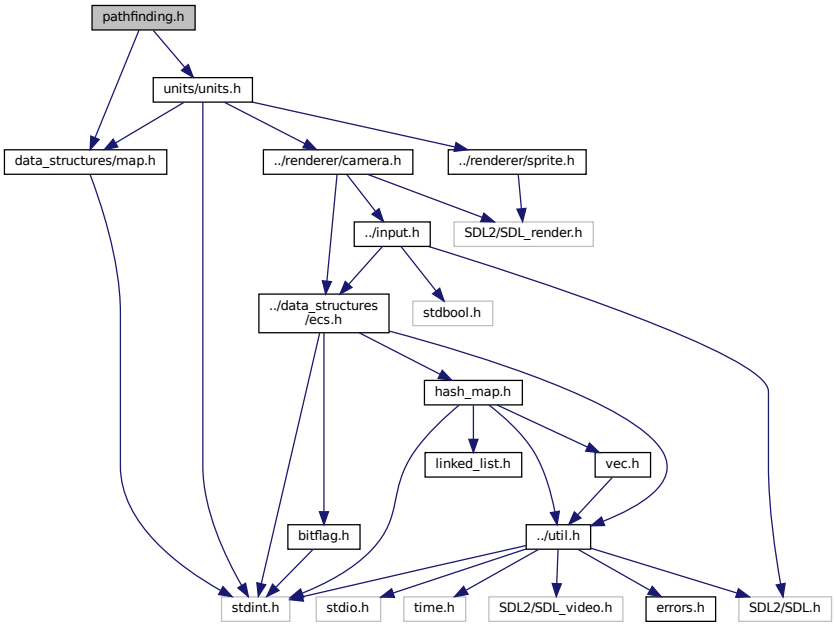
```
UnitT* parse (
        char * path,
        SDL_Renderer * renderer,
        SDL_Window * window )
```

This function create and UnitT by parsing the file at `path`, the file must follow the model of `unit_template.h`

## 6.20 pathfinding.h File Reference

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

Include dependency graph for src/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)

### 6.20.1 Function Documentation

#### 6.20.1.1 pathfind_astar_heuristic()

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

Returns the distance between `src` and `dest` times the cost for crossing a tile (cost = `TILE_NUMBER`/sum(speeds)) (currently using the euclidean distance)

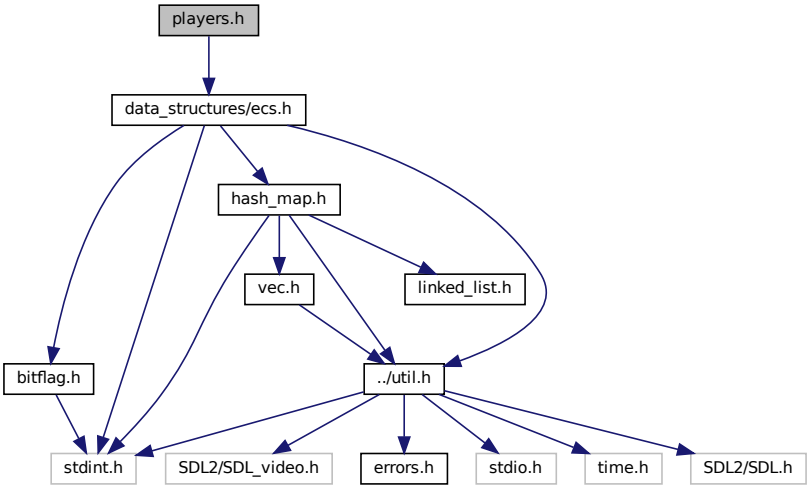
#### 6.20.1.2 VEC() typedef VEC (

TilePosition * )

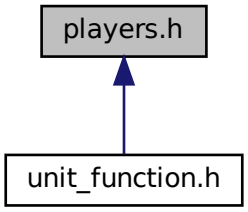
A Path on the global map is succession of tile positions, first index being the start of the path

## 6.21 players.h File Reference

#include "data_structures/ecs.h"
Include dependency graph for players.h:




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




### Data Structures

- struct PlayerManager

  *A component that stores the current status of one of the players.*
- struct WaterSource

  *A component that flags an entity as a source of water for its owner.*
- struct ClaySource

  *A component that flags an entity as a source of clay for its owner.*

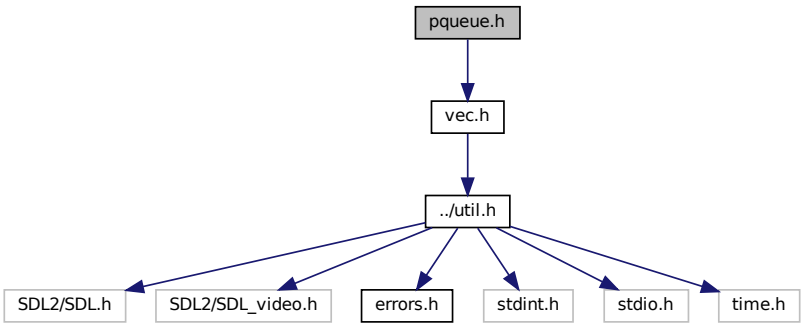
### Functions

- void update_ressources (World *w)

  *Self explanatory.*

## 6.22 pqueue.h File Reference

`#include "vec.h"`
Include dependency graph for src/data_structures/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)

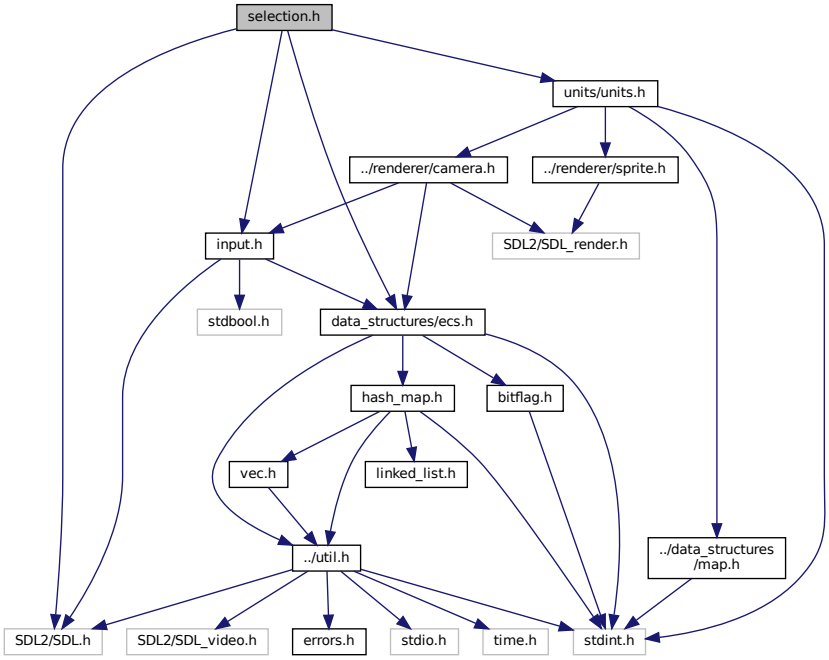
  *returns the element with the smallest weight in the queue*
- PQueue pqueue_push_inner (PQueue p, void *val, double weight)

  *puts an element in the queue*

## 6.23 selection.h File Reference

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

Include dependency graph for selection.h:




### Data Structures

- struct Selectable
- struct Selector

  *A component that manages selection.*

### Enumerations

- enum SelectionType { **Normal** , **Building** }

### Functions

- void selection_event (World ∗w, SDL_Renderer ∗r, Entity ∗e, Inputs ∗i, KeyState st)

  *A `KeyEvent` that manages selections.*
- void draw_selection (World ∗w, SDL_Renderer ∗rdr, SDL_Window ∗window)

  *draws the selection box when required*
- void set_building_selection (World ∗w, char ∗building, UnitTypes but, int water, int clay)
- void selector_free (void ∗s)

  *Self explanatory.*
- void render_unit_grid (World ∗w, EntityRef e)

  *Self explanatory.*
- void unselect (Selector ∗s)

  *Advances one step in the unit unselection process.*

### 6.23.1 Enumeration Type Documentation

#### 6.23.1.1 SelectionType `enum SelectionType`

The type of the ongoing selection, i.e wether it is used for unit selection or for placement

### 6.23.2 Function Documentation

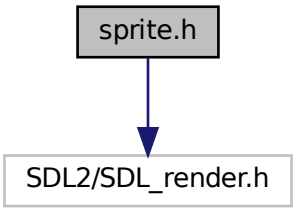
#### 6.23.2.1 set_building_selection() `void set_building_selection (`

```
World * w,
char * building,
UnitTypes but,
int water,
int clay )
```

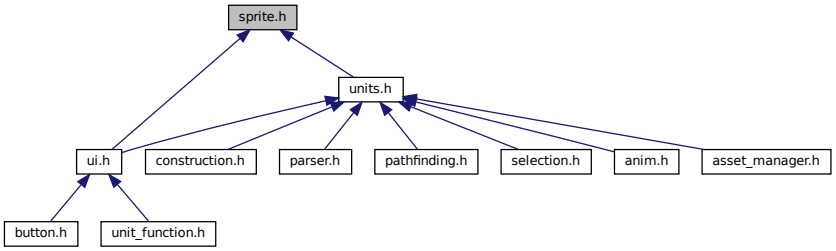
Switches the selector to `Building` type for a specified `building` and its UnitTypes `but`

## 6.24 sprite.h File Reference

`#include <SDL2/SDL_render.h>`
Include dependency graph for sprite.h:




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



### Data Structures

- struct Sprite

  *A component that stores an entity's sprite.*

### Functions

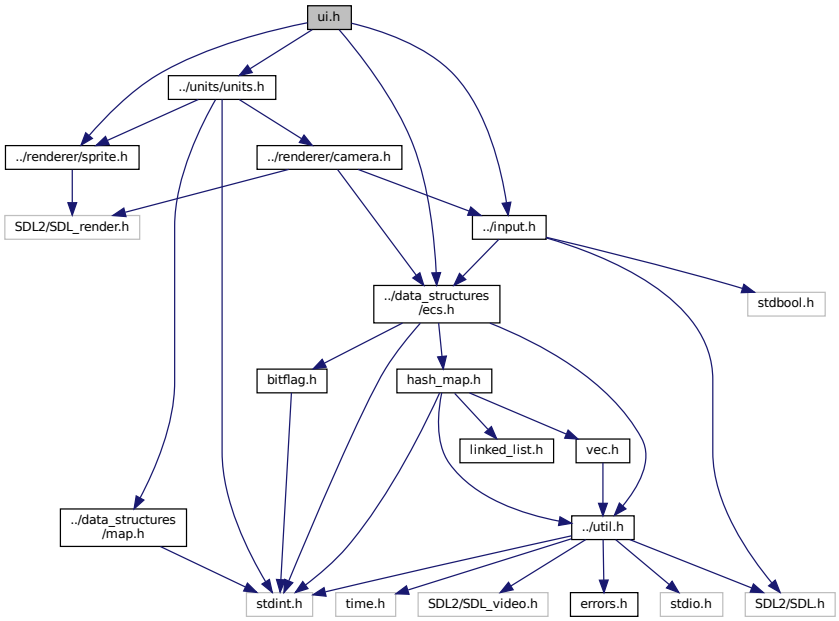
- void sprite_component_free (void *temp)

  *Frees a `Sprite`. Generally, should only be called by the ecs.*

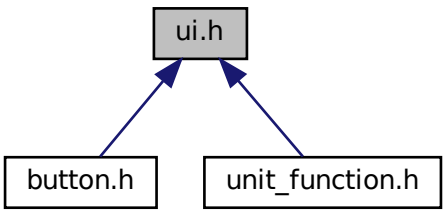
## 6.25 ui.h File Reference

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

Include dependency graph for ui.h:




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



### Data Structures

- struct Text
- struct Background

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

  *Type that corresponds to the minimap.*
- struct ActualisedText
- struct Hoverable

  *Entities with this component show text when hovered.*

### Typedefs

- typedef void(∗ ClickEvent) (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window)

  *The type of the callback function called when a `Clickable` is clicked.*

### Functions

- void render_ui (World ∗w, SDL_Renderer ∗rdr, SDL_Window ∗wi)

  *Renders any entity that has user interface related components.*
- 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)

  *Self explanatory.*
- void minimap_component_free (void ∗temp)

  *Self explanatory.*
- void background_component_free (void ∗temp)

  *Self explanatory.*
- void clickable_component_free (void ∗temp)

  *Self explanatory.*
- void text_component_free (void ∗temp)

  *Self explanatory.*
- void actualised_text_component_free (void ∗temp)

  *Self explanatory.*
- Background ∗ spawn_backbackground (SDL_Renderer ∗rdr, SDL_Window ∗window)

  *Creates a black background that will be rendered before everything else.*
- void null_click_event (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window)
- void biggest_possible_rectangle_centered (SDL_Rect ∗outer, SDL_Rect ∗inner, int padding)
- void biggest_possible_rectangle (SDL_Rect ∗outer, SDL_Rect ∗inner, int padding)
- ActualisedText ∗ render_game_state (World ∗w)
- char ∗ running_to_str (World ∗w, Entity ∗e)
- SDL_Renderer ∗ get_renderer (World ∗w)

  *Self explanatory.*
- SDL_Window ∗ get_window (World ∗w)

  *Self explanatory.*
- char ∗ unit_hover_text (World ∗w, Entity ∗e)

### 6.25.1 Function Documentation

#### 6.25.1.1 biggest_possible_rectangle()

```
void biggest_possible_rectangle (
        SDL_Rect * outer,
        SDL_Rect * inner,
        int padding )
```

Changes `inner` so that it becomes the biggest rectangle of same ratio that can fit into `outer` padded by `padding` pixels.

#### 6.25.1.2 biggest_possible_rectangle_centered()

```
void biggest_possible_rectangle_centered (
        SDL_Rect * outer,
        SDL_Rect * inner,
        int padding )
```

Changes `inner` so that it becomes the biggest rectangle of same ratio that can fit into `outer` padded by `padding` pixels and centers it.

#### 6.25.1.3 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 it is set to 1 there will be a visual change by darkening the sprite and if it is set to 2 it will start the action linked to the sprite. It must be noted that if you click on the sprite, move your mouse out and then release the click it will do nothing as a way to correct missclicks.

#### 6.25.1.4 null_click_event()

```
void null_click_event (
        World * w,
        SDL_Renderer * renderer,
        SDL_Window * window )
```

A click_event that must be used when a clickable shouldn't do anything. This function ought to be useless outside of debug.

#### 6.25.1.5 render_game_state()

```
ActualisedText* render_game_state (
        World * w )
```

This function adds an `Actualised_Text`to the world that will show the game state in the upper left corner of the game.

#### 6.25.1.6 render_hoverable()

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

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

#### 6.25.1.7 running_to_str()

```
char* running_to_str (
        World * w,
        Entity * e )
```

This function returns the string that represents a specific value of `RUNNING`. The argument are not used but they are there for type consistency. The string is padded with spaces at the end so that when the text is rendered they all begin at the same place

#### 6.25.1.8 unit_hover_text()

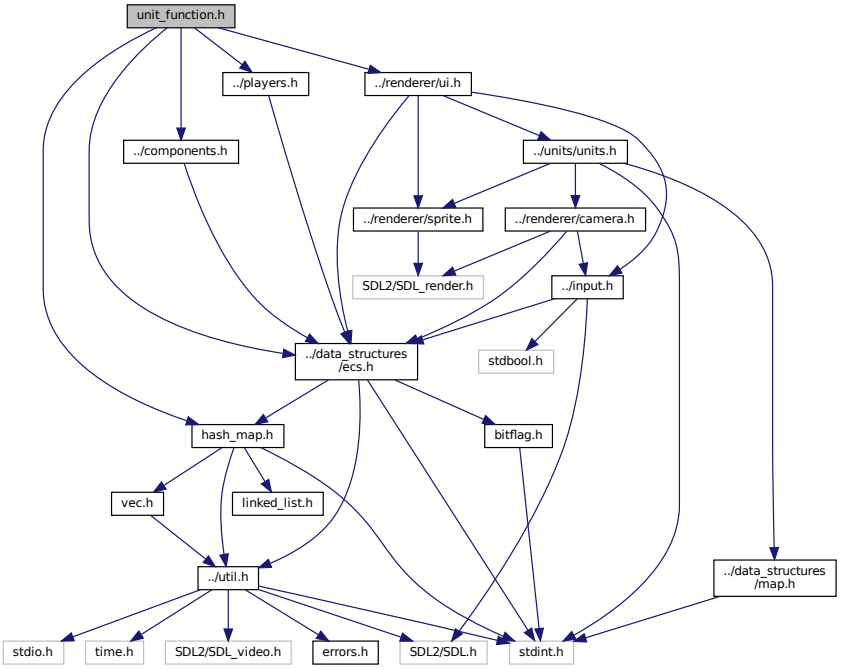
```
char* unit_hover_text (
        World * w,
        Entity * e )
```

This function is used to get the text that will appear when the button to create an unit is hovered.

## 6.26 unit_function.h File Reference

```
#include "../data_structures/ecs.h"
#include "../data_structures/hash_map.h"
#include "../renderer/ui.h"
#include "../components.h"
#include "../players.h"
```

Include dependency graph for unit_function.h:




### Macros

- #define slot_spawn_unit(func, unit_name, wat, cla, entity, unit)

### Typedefs

- typedef ClickEvent(∗ GridFunction) (World ∗, int, Entity ∗)

### Functions

- void empty_click_event (World ∗w, SDL_Renderer ∗renderer, SDL_Window ∗window)

  *A dummy `ClickEvent` that does nothing.*
- ClickEvent **tanuki_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **well_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **fish_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **frog_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **forum_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **debug_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **debug2_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **fort_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **beaver_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **casern_grid** (World ∗w, int slot, Entity ∗e)
- ClickEvent **konbini_grid** (World ∗w, int slot, Entity ∗e)
- PlayerManager ∗ get_player_0 (World ∗w)

  *Returns the `PlayerManager` of player 0, i.e. the human player.*
- void set_grid_functions ()

  *Initializes `GRID_FUNCTION_MAP`*
- void free_grid_functions ()

  *Frees `GRID_FUNCTION_MAP`*

### Variables

- HashMap GRID_FUNCTION_MAP

### 6.26.1 Macro Definition Documentation

#### 6.26.1.1 slot_spawn_unit `#define slot_spawn_unit(`

*func,*
*unit_name,*
*wat,*
*cla,*
*entity,*
*unit )*

**Value:**

```
  void func(World *w, SDL_Renderer *renderer, SDL_Window *window) {            \
    PlayerManager *pm0 = get_player_0(w);                                      \
    if (pm0->water >= wat && pm0->clay >= cla) {                               \
      pm0->water -= wat;                                                       \
      pm0->clay -= cla;                                                        \
      char *c = malloc(sizeof(char) *                                          \
                       (strlen("src/units/unit_" #unit_name ".c") + 1));       \
      strcpy(c, "src/units/unit_" #unit_name ".c");                            \
      Position *p = entity_get_component(w, entity, COMP_POSITION);            \
      spawn_unit(                                                              \
          w, unit, renderer, window,                                           \
          (Position){p->x + rand() % 200 - 100, p->y + rand() % 200 - 100},    \
          0);                                                                  \
    }                                                                          \
  }
```

A macro that extends to the declation of a `ClickEvent` that spawns a unit of name `unit_name` for `wat` water and `cla` clay, around the position of `entity`, and with `unit` their name in the `Unit` enum

### 6.26.2 Typedef Documentation

#### 6.26.2.1 GridFunction `typedef ClickEvent(* GridFunction) (World *, int, Entity *)`

The type of the callback function called when a button in a unit's action grid is pressed

### 6.26.3 Variable Documentation

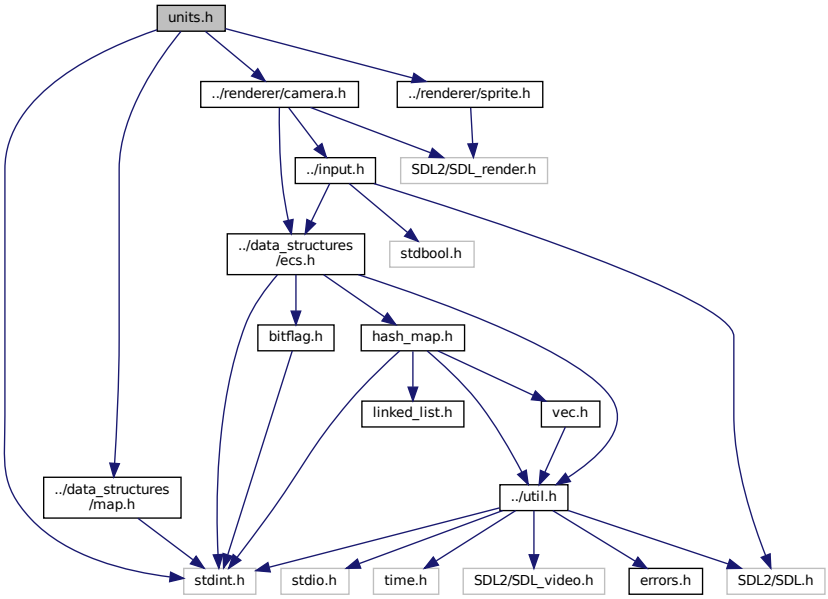
#### 6.26.3.1 GRID_FUNCTION_MAP `HashMap GRID_FUNCTION_MAP [extern]`

A global `HashMap` that stores the actions to do when a unit's grid is clicked

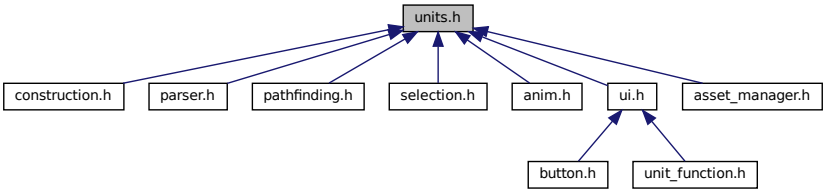
## 6.27 units.h File Reference

```
#include <stdint.h>
#include "../data_structures/map.h"
#include "../renderer/camera.h"
#include "../renderer/sprite.h"
```

Include dependency graph for units.h:



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




### Data Structures

- struct Unit
- struct UnitT
- struct Ownership

  *A component indicating to which player a unit is belonging.*

### Enumerations

- enum UnitTypes {
  **UNIT_TEST** , **WELL** , **FURNACE** , **CASERN** ,
  **TOWER** , **KONBINI** , **HOUSE** , **FORT** ,
  **FORUM** , **UWELL** , **UFURNACE** , **UCASERN** ,
  **UTOWER** , **UKONBINI** , **UHOUSE** , **UFORT** ,
  **UFORUM** , **BEAVER** , **UBEAVER** , **BASE_SOLDIER** ,
  **BASE_FISH** , **MAID** , **FROG** , **SHRIMP** ,
  **HIPPO** , **NARVAL** , **PINGU** , **SAMURAI** ,
  **SECU** , **T34** , **DEBUG** , **DEBUG2** ,
  **UNIT_NUMBER** }

  *an `enum` containing all the units for the game*

### Functions

- double units_get_tile_speed (UnitTypes u, TileTypes t)
- void unit_component_free (void *uni)

  *Self explanatory.*
- void unitt_free (UnitT *u)

  *Self explanatory.*
- Entity * spawn_unit (World *w, UnitTypes t, SDL_Renderer *renderer, SDL_Window *window, Position p, char owner)

  *Creates an Unit of the UnitTypes `t` at the position `p` owned by `owner`.*

#### 6.27.1 Function Documentation

#### 6.27.1.1 units_get_tile_speed()

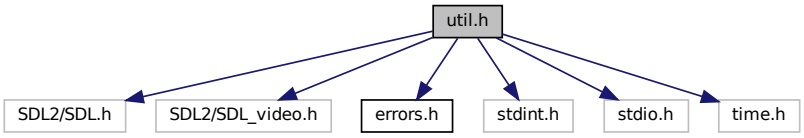
```
double units_get_tile_speed (
        UnitTypes u,
        TileTypes t )
```

returns the speed at which a unit of type `unit` should go on a tile of type `tile`

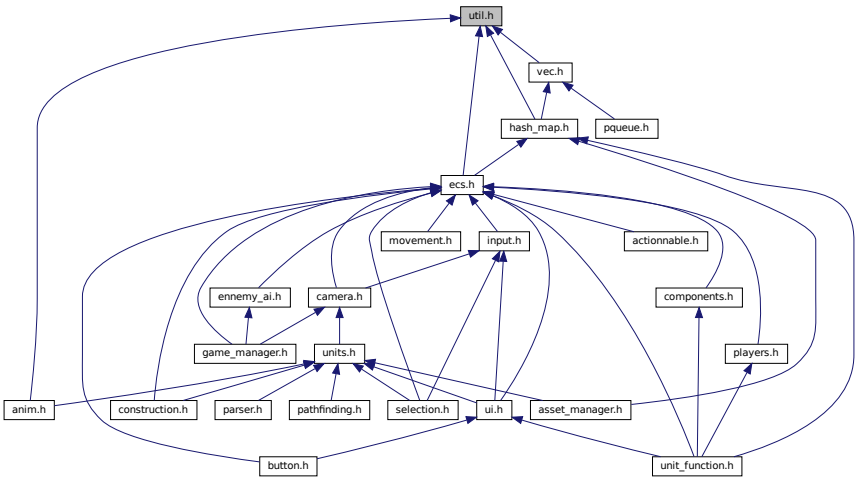
## 6.28 util.h File Reference

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_video.h>
#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 used for the units'movement.*
- struct Window
- struct Renderer

### 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))

  *Self explanatory.*
- #define min(a, b) ((a < b) ? (a) : (b))

  *Self explanatory.*
- #define **v2op_dec**(name) Vec2 v2##name(Vec2 a, Vec2 b)

### Typedefs

- typedef unsigned int **uint**

### Enumerations

- enum Running {
  **STOP** , **MAIN** , **OPTIONMAIN** , **IN_GAME** ,
  **IN_GAMEMENU** , **IN_GAMEOPTION** , **VICTORY** , **DEFEAT** }

  *This enum is used to know the state of the game.*

### 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 product between `Vec2 a` and the scalar `b`*
- Vec2 v2div (Vec2 a, float b)

  *performs a product between `Vec2 a` and the scalar `1/b`*

- float v2angle (Vec2 a)

  *returns the angle (in radian) between* `a` *and the* `(0,1)` *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`

### 6.28.1 Macro Definition Documentation

#### 6.28.1.1 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 `ASSERTION_FAILED`

#### 6.28.1.2 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`

#### 6.28.1.3 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`

#### 6.28.1.4 WARN `#define WARN( message )`

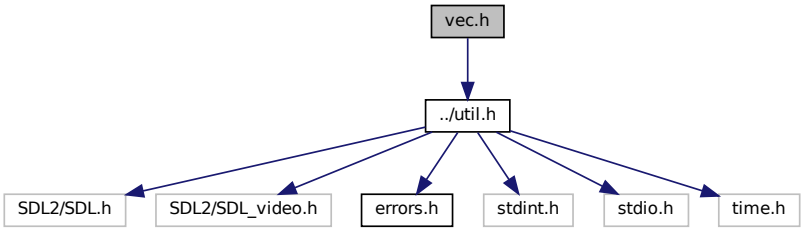
**Value:**

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

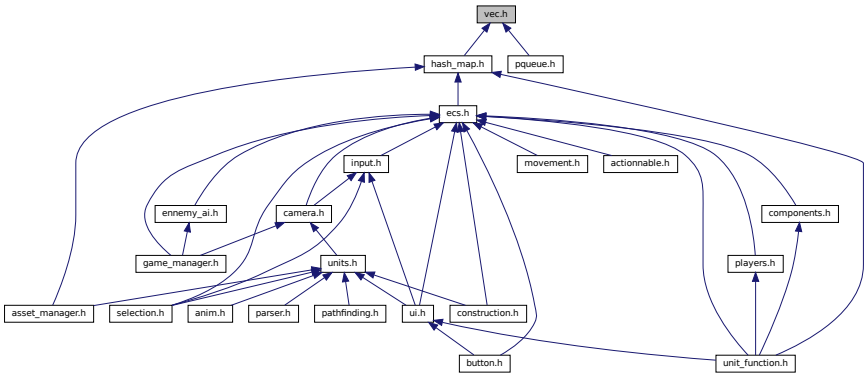
prints `message` as a warning

## 6.29 vec.h File Reference

`#include "../util.h"`
Include dependency graph for src/data_structures/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)
- void vec_clear (VEC(void) vec)

  *Removes every elements of a `Vec`.*

### Variables

- void * **obj**

#### 6.29.1 Detailed Description

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

#### 6.29.2 Macro Definition Documentation

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

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

#### 6.29.3 Function Documentation

##### 6.29.3.1 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`

#### 6.29.3.2 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

#### 6.29.3.3 vec_pop() `void vec_pop (`

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

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

#### 6.29.3.4 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)

#### 6.29.3.5 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