R-Type

Generated by Doxygen 1.9.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 network::client Class Reference

Public Member Functions

- **client** (asio::ip::udp::endpoint &&endp)
- std::string **get_ip** () const
- unsigned short **get_port** ()
- const asio::ip::udp::endpoint & **get_endpoint** () const noexcept
- std::vector< packet > & **get_packets** () noexcept
- std::vector< packet > **pop_packets** () noexcept
- bool & **alive** ()

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

- include/Network/Client.hpp

4.2 Core Class Reference

Public Member Functions

- void **loop** ()

  this function is the main loop that calls menu, lobby and game

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

- include/UI/Core.hpp

4.3 DyLib Class Reference

Classes

- class exception
- class handle_error
- class symbol_error

Public Member Functions

- constexpr DyLib () noexcept=default
- **DyLib** (const DyLib &)=delete
- DyLib & **operator=** (const DyLib &)=delete
- DyLib (DyLib &&other) noexcept
- DyLib & **operator=** (DyLib &&other) noexcept
- DyLib (const char ∗path)
- **DyLib** (const std::string &path)
- **DyLib** (std::string path, const char ∗ext)
- void open (const char ∗path)
- void **open** (const std::string &path)
- void **open** (std::string path, const char ∗ext)
- template< typename T >
  std::function< T > getFunction (const char ∗name) const
- template< typename T >
  std::function< T > **getFunction** (const std::string &name) const
- template< typename T >
  T & getVariable (const char ∗name) const
- template< typename T >
  T & **getVariable** (const std::string &name) const
- void close () noexcept

Static Public Attributes

- static constexpr auto **extension** = ".so"

4.3.1 Constructor & Destructor Documentation

4.3.1.1 DyLib() [1/3]

```
constexpr DyLib::DyLib ( )  [constexpr], [default], [noexcept]
```

Creates a dynamic library object

4.3.1.2 DyLib() [2/3]

```
DyLib::DyLib (
            DyLib && other )  [inline], [noexcept]
```

Move constructor : move a dynamic library instance to build this object

Parameters

other	ref on rvalue of the other DyLib (use std::move)

4.3.1.3 DyLib() [3/3]

```
DyLib::DyLib (
            const char * path )  [inline], [explicit]
```

Creates a dynamic library instance

Parameters

path	path to the dynamic library to load
ext	use DyLib::extension to specify the os extension (optional parameter)

4.3.2 Member Function Documentation

4.3.2.1 close()

```
void DyLib::close ( )  [inline], [noexcept]
```

Close the dynamic library currently loaded in the object. This function will be automatically called by the class destructor

4.3.2.2 getFunction()

```
template<typename T >
std::function< T > DyLib::getFunction (
            const char * name ) const  [inline]
```

Get a function from the dynamic library currently loaded in the object

Parameters

T	the template argument must be the function prototype. it must be the same pattern as the template of std::function
name	symbol name of the function to get from the dynamic library

Returns

std::function<T> that contains the function

4.3.2.3 getVariable()

```
template<typename T >
T & DyLib::getVariable (
            const char * name ) const  [inline]
```

Get a global variable from the dynamic library currently loaded in the object

Parameters

T	type of the global variable
name	name of the global variable to get from the dynamic library

Returns

global variable of type <T>

4.3.2.4 open()

```
void DyLib::open (
            const char * path )  [inline]
```

Load a dynamic library into the object. If a dynamic library was already opened, it will be unload and replaced

Parameters

path	path to the dynamic library to load
ext	use DyLib::extension to specify the os extension (optional parameter)

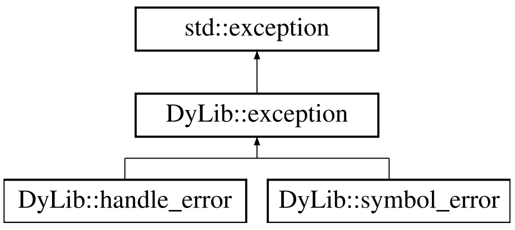
The documentation for this class was generated from the following file:

- include/DyLib.hpp

4.4 DyLib::exception Class Reference

```
#include <DyLib.hpp>
```

Inheritance diagram for DyLib::exception:



Public Member Functions

- **exception** (std::string &&message)
- const char ∗ **what** () const noexcept override

Protected Attributes

- const std::string **m_error**

4.4.1 Detailed Description

This exception is thrown when the DyLib class encountered an error.

Returns

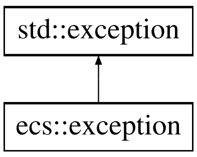
error message by calling what() member function

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

- include/DyLib.hpp

4.5 ecs::exception Class Reference

Inheritance diagram for ecs::exception:



Public Member Functions

- **exception** (std::string err)
- template<typename ... T>
  exception (T &&...err)
- const char $*$ **what** () const noexcept override

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

- include/ECS/Exception.hpp

4.6 Game Class Reference

Public Member Functions

- **Game** (sf::RenderWindow &window, network::socket &socket, uint8_t &status)
- void **gameScreen** ()

  this function is the game loop that calls all utils fonctions
- void **getServState** ()

  Get the Serv State object.
- void parseData (const std::string &str)

  this function is used to parse date received from serv updating game entities status
- void removeKilled (std::vector< size_t > id)

  function that remove dead ids in m_display
- void **checkInput** ()

  check user inputs and send them to serv
- void **gameEvents** ()

  this function is managing all events during the game
- void **eventsButton** ()

  this function is animating buttons when player is dead
- void **eventsPressedButton** ()

  this function is sending request to serv when player click on a button
- void **animSprites** ()

  this function is animating sprites
- void **displayGame** ()

  this function is drawing the game on the screen

4.6.1 Member Function Documentation

4.6.1.1 parseData()

```
void Game::parseData (
            const std::string & str )
```

this function is used to parse date received from serv updating game entities status

Parameters

data	const std::string & (received from serv)

4.6.1.2 removeKilled()

```
void Game::removeKilled (
            std::vector< size_t > id )
```

function that remove dead ids in m_display

Parameters

id	std::vector<size_t>

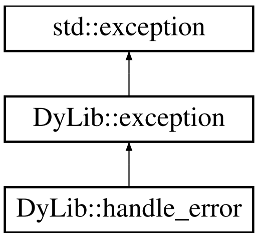
The documentation for this class was generated from the following file:

- include/UI/Game.hpp

4.7 DyLib::handle_error Class Reference

```
#include <DyLib.hpp>
```

Inheritance diagram for DyLib::handle_error:



Public Member Functions

- **handle_error** (std::string &&message)

Additional Inherited Members

4.7.1 Detailed Description

This exception is thrown when the library failed to load or the library encountered symbol resolution issues

Parameters

message	error message

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

- include/DyLib.hpp

4.8 network::header Struct Reference

Public Attributes

- uint32_t **magicValue**

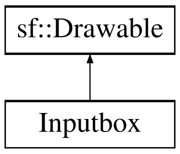
- uint8_t **type**
- uint32_t **size**

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

- include/Network/Header.hpp

4.9 Inputbox Class Reference

Inheritance diagram for Inputbox:



Public Member Functions

- **Inputbox** (int size=15, sf::Color color=sf::Color::White, bool select=false, size_t lim=0)
- void inputLogic (char c)

  this function is adding char to the end of a sf::String
- void setFont (const sf::Font &font)

  Set the Font object.
- void setPosition (sf::Vector2f pos)

  Set the Position object.
- void setLimit (size_t lim)

  Set the Limit object (set the maximum size of the String object)
- void setSelected (bool sel)

  Set the Selected object.
- void setString (sf::String str)

  Set the String object.
- sf::String & getText ()

  Get the Text object.
- const sf::String & getText () const

  Get the Text object.
- bool & getSelected ()

  Get the Selected object.
- sf::Text & getInputbox ()

  Get the Inputbox object.
- void **updateString** ()

  this function is updating the sf::Text with the sf::String
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Drawing the class on the window.

4.9.1 Member Function Documentation

4.9.1.1 draw()

```
void Inputbox::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

Drawing the class on the window.

Parameters

target	sf::RenderTarget window
states	sf::RenderStates Define the states used for drawing to a RenderTarget

4.9.1.2 getInputbox()

```
sf::Text & Inputbox::getInputbox ( )
```

Get the Inputbox object.

Returns

sf::Text&

4.9.1.3 getSelected()

```
bool & Inputbox::getSelected ( )
```

Get the Selected object.

Returns

true

false

4.9.1.4 getText() [1/2]

```
sf::String & Inputbox::getText ( )
```

Get the Text object.

Returns

sf::String&

4.9.1.5 getText() [2/2]

```
const sf::String & Inputbox::getText ( ) const
```

Get the Text object.

Returns

const sf::String&

4.9.1.6 inputLogic()

```
void Inputbox::inputLogic (
            char c )
```

this function is adding char to the end of a sf::String

Parameters

c	char to add at the end of the string

4.9.1.7 setFont()

```
void Inputbox::setFont (
            const sf::Font & font )
```

Set the Font object.

Parameters

font	sf::Font to set

4.9.1.8 setLimit()

```
void Inputbox::setLimit (
            size_t lim )
```

Set the Limit object (set the maximum size of the String object)

Parameters

lim	size_t to set

4.9.1.9 setPosition()

```
void Inputbox::setPosition (
             sf::Vector2f pos )
```

Set the Position object.

Parameters

pos	sf::Vector2f to set

4.9.1.10 setSelected()

```
void Inputbox::setSelected (
             bool sel )
```

Set the Selected object.

Parameters

sel	bool to set

4.9.1.11 setString()

```
void Inputbox::setString (
             sf::String str )
```

Set the String object.

Parameters

str	sf::String to set

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

- include/UI/Inputbox.hpp

4.10 Lobby Class Reference

Public Member Functions

- **Lobby** (sf::RenderWindow &window, network::socket &socket, uint8_t &status)

- void **initLobby** ()

    this function init all lobby requirements to start from 0
- void **endLobby** ()

    function called at the end of lobby to stop music, reset the cursor...
- void **lobbyScreen** ()

    this function is the lobby loop that calls utils functions
- void **eventsLobby** ()

    this function is managing all lobby events
- void **eventsReady** ()

    this function is managing ready button animations
- void **eventsDisconnect** ()

    this function is managing disconnect button animations
- void **eventsPressButton** ()

    this function is managing all buttons events
- void **reqLobby** ()

    this function is communicating with the serv getting infos of all other clients connected to the serv: ready or not getting info when game start
- void parseData (const std::string &data)

    this function is used to parse date received from serv setting a vector of profile (profile = client)
- void **displayLobby** ()

    function that draw the lobby on the window

4.10.1 Member Function Documentation

4.10.1.1 parseData()

```
void Lobby::parseData (
            const std::string & data )
```

this function is used to parse date received from serv setting a vector of profile (profile = client)

Parameters

data	const std::string & (received from serv)

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

- include/UI/Lobby.hpp

4.11 Menu Class Reference

Public Member Functions

- **Menu** (sf::RenderWindow &window, network::socket &socket, uint8_t &status)

- void **initMenu** ()

  this function init all menu requirements to start from 0
- void **loginScreen** ()

  this function is the menu loop that calls all utils functions
- void **eventsInputbox** ()

  this function is updating cursor texture also select the inputbox selected with cursor
- void **checkInputboxSelected** ()

  this function select the next inputbox when user press tab
- void **eventsMenu** ()

  this function is managing all menu events
- void **eventsLogin** ()

  this function update login button texture
- void **eventsUser** ()

  this function udpate user button texture
- void **tryLogin** ()

  this function send to serv a login request set an error if connection failed go to username stage if connection is setup
- void **tryUsername** ()

  this function send username entered by user to the serv set an error if username is already used go to lobby if username is good
- void **displayMenu** ()

  this function draw the menu on window
- void **drawUser** ()

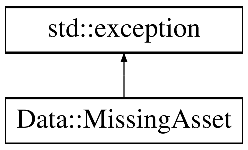
  this function draw the username stage and errors

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

- include/UI/Menu.hpp

4.12 Data::MissingAsset Class Reference

Inheritance diagram for Data::MissingAsset:



Public Member Functions

- **MissingAsset** (const std::string &path)
- const char ∗ **what** () const noexcept override

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

- include/UI/Data.hpp

4.13 ecs::module Class Reference

Public Member Functions

- virtual ecs::entity spawn (registry &)=0
- virtual std::function< void(registry &)> get_system ()=0
- virtual std::string get_label ()=0

4.13.1 Member Function Documentation

4.13.1.1 get_label()

```
virtual std::string ecs::module::get_label ( )  [pure virtual]
```

Returns

the entity's label

4.13.1.2 get_system()

```
virtual std::function< void(registry &)> ecs::module::get_system ( )  [pure virtual]
```

get the system to manage the entity

Parameters

registry	the registry to use with the system

Returns

the system function as a std::function<void(registry &)>

4.13.1.3 spawn()

```
virtual ecs::entity ecs::module::spawn (
            registry &  )  [pure virtual]
```

spawn a new entity

Parameters

registry	the registry to put the new entity on

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

- include/ECS/Module.hpp

4.14 network::packet Struct Reference

Public Attributes

- uint8_t **type**
- std::string **data**

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

- include/Network/Packet.hpp

4.15 ecs::component::position Struct Reference

Public Attributes

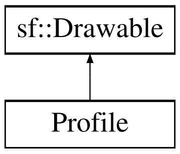
- int **x**
- int **y**
- int **vx**
- int **vy**
- int **cur**

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

- include/ECS/Components.hpp

4.16 Profile Class Reference

Inheritance diagram for Profile:



Public Member Functions

- **Profile** (std::string username, bool rdy, size_t nb)
- void setPosition (sf::Vector2f pos)

  Set the Position object set position of all objects.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Drawing the class on the window.

4.16.1 Member Function Documentation

4.16.1.1 draw()

```
void Profile::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const  [override]
```

Drawing the class on the window.

Parameters

target	sf::RenderTarget window
states	sf::RenderStates Define the states used for drawing to a RenderTarget

4.16.1.2 setPosition()

```
void Profile::setPosition (
             sf::Vector2f pos )
```

Set the Position object set position of all objects.

Parameters

pos	sf::Vector2f pos

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

- include/UI/Profile.hpp

4.17 ecs::registry Class Reference

Public Member Functions

- template<typename T >
  sparse_array< T > & register_component ()

- template<class T >
  bool has_component ()
- template<typename T >
  sparse_array< T > & get_components ()
- template<typename T >
  const sparse_array< T > & get_components () const
- template<typename T >
  sparse_array< T >::reference_type insert_component (const entity &ent, T &&comp)
- template<typename T , typename ... Params>
  sparse_array< T >::reference_type emplace_component (const entity &ent, Params &&...params)
- template<typename T >
  void remove_component (const entity &ent)
- entity spawn_entity ()
- void kill_entity (entity e)
- const std::vector< entity > & get_entities ()
- template<typename Function >
  void add_system (Function &&f)
- void run_systems ()
- void destroy ()

4.17.1 Member Function Documentation

4.17.1.1 add_system()

```
template<typename Function >
void ecs::registry::add_system (
            Function && f )  [inline]
```

This function is used to add a system

Parameters

f	function representing a system

4.17.1.2 destroy()

```
void ecs::registry::destroy ( )  [inline]
```

This function is used to destroy all data from the registry

4.17.1.3 emplace_component()

```
template<typename T , typename ...  Params>
sparse_array< T >::reference_type ecs::registry::emplace_component (
            const entity & ent,
            Params &&...  params )  [inline]
```

emplace a component at a position

Parameters

T	type of the component
ent	position to emplace the component
params	parameters to forward to build a component

Returns

sparse_array of components

4.17.1.4 get_components() [1/2]

```
template<typename T >
sparse_array< T > & ecs::registry::get_components ( )  [inline]
```

get a sparse_array of components from the registry

Parameters

T	type of the component to get

Returns

sparse_array of components

4.17.1.5 get_components() [2/2]

```
template<typename T >
const sparse_array< T > & ecs::registry::get_components ( ) const  [inline]
```

get a sparse_array of components from the registry

Parameters

T	type of the component to get

Returns

sparse_array of components

4.17.1.6 get_entities()

```
const std::vector< entity > & ecs::registry::get_entities ( )  [inline]
```

get entities from the registry

Returns

the entity list

4.17.1.7 has_component()

```
template<class T >
bool ecs::registry::has_component ( )  [inline]
```

check if the registry contains a component

Parameters

T	type of the component to check

Returns

true if the component is registered, false otherwise

4.17.1.8 insert_component()

```
template<typename T >
sparse_array< T >::reference_type ecs::registry::insert_component (
             const entity & ent,
             T && comp )  [inline]
```

insert a component at a position

Parameters

T	type of the component
ent	position to insert the component
comp	component to be insert

Returns

sparse_array of components

4.17.1.9 kill_entity()

```
void ecs::registry::kill_entity (
             entity e )  [inline]
```

This function is used to kill an entity

Parameters

e	entity to kill

4.17.1.10 register_component()

```
template<typename T >
sparse_array< T > & ecs::registry::register_component ( )  [inline]
```

register a new component to the registry

Parameters

T	type of the new component

Returns

the new component as sparse_array<T>

4.17.1.11 remove_component()

```
template<typename T >
void ecs::registry::remove_component (
             const entity & ent )  [inline]
```

remove a component from a sparse_array of component

Parameters

T	type of the component
ent	position to remove the component

4.17.1.12 run_systems()

```
void ecs::registry::run_systems ( )  [inline]
```

This function is used to run all loaded systems

4.17.1.13 spawn_entity()

```
entity ecs::registry::spawn_entity ( )  [inline]
```

This function is used to spawn a new entity

Returns

a new entity

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

- include/ECS/Registry.hpp

4.18 network::server Class Reference

Public Member Functions

- **server** (unsigned short port, size_t max_clients, uint8_t login_handshake, uint8_t logout_handshake, uint8↩ _t alive_handshake)
- **server** (server &)=delete
- server & **operator=** (server &)=delete
- **server** (server &&)=delete
- server & **operator=** (server &&)=delete
- void send (const client &c, uint8_t type, const std::string &buffer="", unsigned short repeat=1)
- void receive ()
- std::vector< client > & get_clients () noexcept
- void allow_new_connections (bool value)

4.18.1 Member Function Documentation

4.18.1.1 allow_new_connections()

```
void network::server::allow_new_connections (
            bool value )  [inline]
```

this function is used to lock or unlock new connections

Parameters

value	choice

4.18.1.2 get_clients()

```
std::vector< client > & network::server::get_clients ( )  [inline], [noexcept]
```

this function is used to get the clients

Returns

client list

4.18.1.3 receive()

```
void network::server::receive ( )  [inline]
```

this function is used to receive pending packets from clients

4.18.1.4 send()

```
void network::server::send (
             const client & c,
             uint8_t type,
             const std::string & buffer = "",
             unsigned short repeat = 1 )  [inline]
```

this function is used to send packets to a client

Parameters

c	client to send the packet
type	type of the request to the client as a enum
buffer	data to send to client
repeat	number of times the packet will be send

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

- include/Network/Server.hpp

4.19 network::socket Class Reference

Public Member Functions

- **socket** (unsigned short port, uint8_t login_handshake, uint8_t logout_handshake, uint8_t alive_handshake)

- **socket** (socket &)=delete
- socket & **operator=** (socket &)=delete
- **socket** (socket &&)=delete
- socket & **operator=** (socket &&)=delete
- void connect (std::string ip, unsigned short port, long login_timeout=3)
- void disconnect () noexcept
- void send (uint8_t type, const std::string &buffer="", unsigned short repeat=1)
- std::string get_remote_ip () const
- unsigned short get_remote_port () const
- std::vector< packet > get_packets () noexcept
- std::vector< packet > pop_packets () noexcept

4.19.1 Member Function Documentation

4.19.1.1 connect()

```
void network::socket::connect (
            std::string ip,
            unsigned short port,
            long login_timeout = 3 )  [inline]
```

this function is used to send a request to the server to connect

Parameters

ip	the host ip
port	is the port for the ip
login_timeout	login timeout in sec, the function will throw after that delay

4.19.1.2 disconnect()

```
void network::socket::disconnect ( )  [inline], [noexcept]
```

this function is used to disconnect from the host and clear the buffer

4.19.1.3 get_packets()

```
std::vector< packet > network::socket::get_packets ( )  [inline], [noexcept]
```

this function is used to get the pending packets

Returns

the pending packets contained in the socket

4.19.1.4 get_remote_ip()

```
std::string network::socket::get_remote_ip ( ) const  [inline]
```

this function is used to get the remote ip

Returns

the remote address as a std::string

4.19.1.5 get_remote_port()

```
unsigned short network::socket::get_remote_port ( ) const  [inline]
```

this function is used to get the remote port

Returns

the remote port as a unsigned short

4.19.1.6 pop_packets()

```
std::vector< packet > network::socket::pop_packets ( )  [inline], [noexcept]
```

this function is used to move the pending packets and get them

Returns

the pending packets contained in the socket and empty them on the socket class

4.19.1.7 send()

```
void network::socket::send (
             uint8_t type,
             const std::string & buffer = "",
             unsigned short repeat = 1 )  [inline]
```

this function is used to send packets to the server

Parameters

type	type of the packet
buffer	data to send to the server
repeat	number of times to send the packets

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

- include/Network/Socket.hpp

4.20 ecs::sparse_array< T > Class Template Reference

Public Types

- using **value_type** = std::optional< T >
- using **reference_type** = value_type &
- using **const_reference_type** = const value_type &
- using **container_t** = std::vector< value_type >
- using **size_type** = typename container_t::size_type
- using **iterator** = typename container_t::iterator
- using **const_iterator** = typename container_t::const_iterator

Public Member Functions

- **sparse_array** (sparse_array const &)=default
- **sparse_array** (sparse_array &&) noexcept=default
- sparse_array & **operator=** (sparse_array const &)=default
- sparse_array & **operator=** (sparse_array &&) noexcept=default
- iterator **begin** ()
- const_iterator **begin** () const
- const_iterator **cbegin** () const
- iterator **end** ()
- const_iterator **end** () const
- const_iterator **cend** () const
- size_type **size** () const
- reference_type **operator[ ]** (size_t idx)
- const_reference_type **operator[ ]** (size_t idx) const
- reference_type insert_at (size_type pos, const T &comp)
- reference_type insert_at (size_type pos, T &&comp)
- template<class ... Params>
  reference_type emplace_at (size_type pos, Params &&...params)
- void erase (size_type pos)
- size_type get_index (value_type const &val) const

4.20.1 Member Function Documentation

4.20.1.1 emplace_at()

```
template<typename T >
template<class ...  Params>
reference_type ecs::sparse_array< T >::emplace_at (
             size_type pos,
             Params &&...   params )   [inline]
```

emplace the component at the position pos

Parameters

pos	position to emplace the component
params	parameters to forward to build the component

Returns

the component just emplaced

4.20.1.2 erase()

```
template<typename T >
void ecs::sparse_array< T >::erase (
             size_type pos )  [inline]
```

erase a component at a position

Parameters

pos	position where to erease a component

4.20.1.3 get_index()

```
template<typename T >
size_type ecs::sparse_array< T >::get_index (
             value_type const & val ) const  [inline]
```

get the index from a value

Parameters

val	value

Returns

the index matching the value

4.20.1.4 insert_at() [1/2]

```
template<typename T >
reference_type ecs::sparse_array< T >::insert_at (
             size_type pos,
             const T & comp )  [inline]
```

insert the component at the position pos

Parameters

pos	position to insert the component
comp	component to insert

Returns

the component just inserted

4.20.1.5 insert_at() [2/2]

```
template<typename T >
reference_type ecs::sparse_array< T >::insert_at (
             size_type pos,
             T && comp )  [inline]
```

insert the component at the position pos

Parameters

pos	position to insert the component
comp	component to insert

Returns

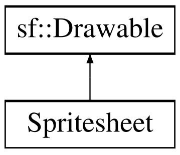
the component just inserted

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

- include/ECS/SparseArray.hpp

4.21 Spritesheet Class Reference

Inheritance diagram for Spritesheet:



Public Member Functions

- **Spritesheet** (sf::Texture &text, sf::Vector2f pos={0, 0}, size_t nb=1, sf::Vector2f dim={1920, 1080}, sf::IntRect rect={0, 0, 1920, 1080}, sf::Vector2f scale={1, 1}, int space=0)
- void setNbSprite (size_t nb)

  Set the Nb Sprite object.
- void setSizeImage (sf::Vector2f size)

  Set the Size Image object.
- void setTexture (const sf::Texture &text)

  Set the Texture object.
- void setPosition (sf::Vector2f pos)

  Set the Position object.
- void setRectSize (sf::IntRect rect)

  Set the Rect object (only width and height)
- void setRect (sf::IntRect rect)

  Set the Rect object.
- void setSpace (int space)

  Set the Space object.
- void setScale (sf::Vector2f scale)

  Set the Scale object.
- size_t & getCurrentImage ()

  Get the Current Image object.
- size_t & getNbSprite ()

  Get the Nb Sprite object.
- sf::Vector2f & getScale ()

  Get the Scale object.
- sf::IntRect & getRect ()

  Get the Rect object.
- sf::Vector2f & getSize ()

  Get the Size object.
- int & getSpace ()

  Get the Space object.
- sf::Vector2f & getPosition ()

  Get the Position object.
- const sf::Texture & getTexture ()

  Get the Texture object.
- bool mouseIsInSprite (sf::Vector2i pmouse)

  function that check if the cursor is in the sprite return true if the cursor is in the sprite, false otherwise
- void playImage (size_t nb)

  function that play an exact sprite
- void **animSprite** ()

  function that call play image with the next current sprite
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Drawing the class on the window.

4.21.1 Member Function Documentation

4.21.1.1 draw()

```
void Spritesheet::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

Drawing the class on the window.

Parameters

target	sf::RenderTarget window
states	sf::RenderStates Define the states used for drawing to a RenderTarget

4.21.1.2 getCurrentImage()

```
size_t & Spritesheet::getCurrentImage ( )
```

Get the Current Image object.

Returns

size_t&

4.21.1.3 getNbSprite()

```
size_t & Spritesheet::getNbSprite ( )
```

Get the Nb Sprite object.

Returns

size_t&

4.21.1.4 getPosition()

```
sf::Vector2f & Spritesheet::getPosition ( )
```

Get the Position object.

Returns

sf::Vector2f&

4.21.1.5 getRect()

```
sf::IntRect & Spritesheet::getRect ( )
```

Get the Rect object.

Returns

sf::IntRect&

4.21.1.6 getScale()

```
sf::Vector2f & Spritesheet::getScale ( )
```

Get the Scale object.

Returns

sf::Vector2f&

4.21.1.7 getSize()

```
sf::Vector2f & Spritesheet::getSize ( )
```

Get the Size object.

Returns

sf::Vector2f&

4.21.1.8 getSpace()

```
int & Spritesheet::getSpace ( )
```

Get the Space object.

Returns

int&

4.21.1.9 getTexture()

```
const sf::Texture & Spritesheet::getTexture ( )
```

Get the Texture object.

Returns

sf::Texture&

4.21.1.10 mouseIsInSprite()

```
bool Spritesheet::mouseIsInSprite (
            sf::Vector2i pmouse )
```

function that check if the cursor is in the sprite return true if the cursor is in the sprite, false otherwise

Parameters

pmouse	sf::Vector2i (pos of cursor on windows)

Returns

true

false

4.21.1.11 playImage()

```
void Spritesheet::playImage (
            size_t nb )
```

function that play an exact sprite

Parameters

nb	size_t (sprite nb wanted to display on window)

4.21.1.12 setNbSprite()

```
void Spritesheet::setNbSprite (
            size_t nb )
```

Set the Nb Sprite object.

Parameters

nb	size_t (nb of sprites on image)

4.21.1.13 setPosition()

```
void Spritesheet::setPosition (
            sf::Vector2f pos )
```

Set the Position object.

Parameters

pos	sf::Vector2f (pos of texture on window)

4.21.1.14 setRect()

```
void Spritesheet::setRect (
            sf::IntRect rect )
```

Set the Rect object.

Parameters

rect	sf::IntRect (rect of the sprite on image)

4.21.1.15 setRectSize()

```
void Spritesheet::setRectSize (
            sf::IntRect rect )
```

Set the Rect object (only width and height)

Parameters

rect	

4.21.1.16 setScale()

```
void Spritesheet::setScale (
             sf::Vector2f scale )
```

Set the Scale object.

Parameters

scale	sf::Vector2f (scale of sprite on window)

4.21.1.17 setSizeImage()

```
void Spritesheet::setSizeImage (
             sf::Vector2f size )
```

Set the Size Image object.

Parameters

size	sf::Vector2f (size of image)

4.21.1.18 setSpace()

```
void Spritesheet::setSpace (
             int space )
```

Set the Space object.

Parameters

space	int (space between all sprites on image)

4.21.1.19 setTexture()

```
void Spritesheet::setTexture (
             const sf::Texture & text )
```

Set the Texture object.

Parameters

text	const sf::Texture & (with a load texture)

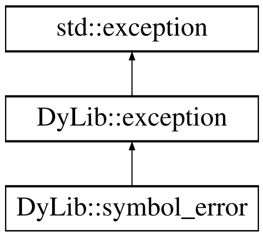
The documentation for this class was generated from the following file:

- include/UI/Spritesheet.hpp

4.22 DyLib::symbol_error Class Reference

```
#include <DyLib.hpp>
```

Inheritance diagram for DyLib::symbol_error:



Public Member Functions

- **symbol_error** (std::string &&message)

Additional Inherited Members

4.22.1 Detailed Description

This exception is thrown when the library failed to load a symbol. This usually happens when you forgot to mark a library function or variable as extern "C"

Parameters

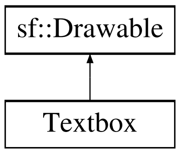
message	error message

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

- include/DyLib.hpp

4.23 Textbox Class Reference

Inheritance diagram for Textbox:



Public Member Functions

- **Textbox** (size_t size=20, sf::String str="", sf::Color color=sf::Color::White, sf::Vector2f pos={0, 0})
- void setSize (size_t size)

  Set the Size object.
- void setString (sf::String str)

  Set the String object.
- void setColor (sf::Color color)

  Set the Color object.
- void setPosition (sf::Vector2f pos)

  Set the Position object.
- void setOutlineColor (sf::Color color, float size)

  Set the Outline Color object.
- void setFont (const sf::Font &font)

  Set the Font object.
- sf::String & getString (void)

  Get the String object.
- sf::Vector2f & getPosition (void)

  Get the Position object.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Drawing the class on the window.

4.23.1 Member Function Documentation

4.23.1.1 draw()

```
void Textbox::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const  [override]
```

Drawing the class on the window.

Parameters

target	sf::RenderTarget window
states	sf::RenderStates Define the states used for drawing to a RenderTarget

4.23.1.2 getPosition()

```
sf::Vector2f & Textbox::getPosition (
             void  )
```

Get the Position object.

Returns

sf::Vector2f&

4.23.1.3 getString()

```
sf::String & Textbox::getString (
             void  )
```

Get the String object.

Returns

sf::String&

4.23.1.4 setColor()

```
void Textbox::setColor (
             sf::Color color )
```

Set the Color object.

Parameters

color	sf::Color (color of the text)

4.23.1.5 setFont()

```
void Textbox::setFont (
             const sf::Font & font )
```

Set the Font object.

Parameters

font	const sf::Font & (font of text)

4.23.1.6 setOutlineColor()

```
void Textbox::setOutlineColor (
            sf::Color color,
            float size )
```

Set the Outline Color object.

Parameters

color	sf::Color (outline color)
size	float (size of outline)

4.23.1.7 setPosition()

```
void Textbox::setPosition (
            sf::Vector2f pos )
```

Set the Position object.

Parameters

pos	sf::Vector2f (position of the text on window)

4.23.1.8 setSize()

```
void Textbox::setSize (
            size_t size )
```

Set the Size object.

Parameters

size	size_t (set size of text)

4.23.1.9 setString()

```
void Textbox::setString (
            sf::String str )
```

Set the String object.

Parameters

str	sf::String (string to draw)

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

- include/UI/Textbox.hpp

4.24 thread_pool Class Reference

Public Member Functions

- **thread_pool** (const thread_pool &)=delete
- **thread_pool** (thread_pool &&)=delete
- thread_pool & **operator=** (const thread_pool &)=delete
- thread_pool & **operator=** (thread_pool &&)=delete
- template<typename Function , typename ... Args>
  void add (Function &&fn, Args &&...args)
- void remove (size_t position)
- void release ()

4.24.1 Member Function Documentation

4.24.1.1 add()

```
template<typename Function , typename ...  Args>
void thread_pool::add (
            Function && fn,
            Args &&...  args )  [inline]
```

Start a function on another thread

Parameters

fn	function to be run on a new thread
args	arguments that will be forward to run the function

4.24.1.2 release()

```
void thread_pool::release ( )  [inline]
```

Release all threads from thread pool This function will be called by the class destructor

4.24.1.3 remove()

```
void thread_pool::remove (
             size_t position )  [inline]
```

Remove a thread from the thread pool

Parameters

position	position of the thread to be remove

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

- include/ThreadPool.hpp

4.25 vec2float Struct Reference

Public Attributes

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

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

- include/UI/SpriteSize.hpp

Chapter 5

File Documentation

5.1 include/DyLib.hpp File Reference

Cross-platform Dynamic Library Loader.

```
#include <string>
#include <functional>
#include <exception>
#include <utility>
#include <dlfcn.h>
```

Classes

- class DyLib
- class DyLib::exception
- class DyLib::handle_error
- class DyLib::symbol_error

5.1.1 Detailed Description

Cross-platform Dynamic Library Loader.

Author

Martin Olivier

Version

1.6.1

5.2 DyLib.hpp

Go to the documentation of this file.

```cpp

#pragma once

#include <string>
#include <functional>
#include <exception>
#include <utility>
#if defined(_WIN32) || defined(_WIN64)
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#else
#include <dlfcn.h>
#endif

class DyLib
{
private:
#if defined(_WIN32) || defined(_WIN64)
    HINSTANCE m_handle{nullptr};
    static HINSTANCE openLib(const char *path) noexcept
    {
        return LoadLibraryA(path);
    }
    FARPROC getSymbol(const char *name) const noexcept
    {
        return GetProcAddress(m_handle, name);
    }
    void closeLib() noexcept
    {
        FreeLibrary(m_handle);
    }
    static char *getErrorMessage() noexcept
    {
        constexpr size_t bufSize = 512;
        auto errorCode = GetLastError();
        if (!errorCode)
            return nullptr;
        static char msg[bufSize];
        auto lang = MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US);
        const DWORD len = FormatMessageA(FORMAT_MESSAGE_FROM_SYSTEM, nullptr, errorCode, lang, msg,
    bufSize, nullptr);
        if (len > 0)
            return msg;
        return nullptr;
    }
#else
    void *m_handle{nullptr};
    static void *openLib(const char *path) noexcept
    {
        return dlopen(path, RTLD_NOW | RTLD_LOCAL);
    }
    void *getSymbol(const char *name) const noexcept
    {
        return dlsym(m_handle, name);
    }
    void closeLib() noexcept
    {
        dlclose(m_handle);
    }
    static char *getErrorMessage() noexcept
    {
        return dlerror();
    }
#endif
    static std::string getHandleError(const std::string &name)
    {
        auto err = getErrorMessage();
        if (!err)
            return "error while loading dynamic library \"" + name + "\"";
        return err;
    }
    static std::string getSymbolError(const std::string &name)
    {
        auto err = getErrorMessage();
        if (!err)
            return "error while loading symbol \"" + name + "\"";
        return err;
    }

public:

#if defined(_WIN32) || defined(_WIN64)
```

```
    static constexpr auto extension = ".dll";
#elif defined(__APPLE__)
    static constexpr auto extension = ".dylib";
#else
    static constexpr auto extension = ".so";
#endif

    class exception : public std::exception
    {
    protected:
        const std::string m_error;
    public:
        explicit exception(std::string &&message) : m_error(std::move(message)) {}
        const char *what() const noexcept override {return m_error.c_str();}
    };

    class handle_error : public exception
    {
    public:
        explicit handle_error(std::string &&message) : exception(std::move(message)) {}
    };

    class symbol_error : public exception
    {
    public:
        explicit symbol_error(std::string &&message) : exception(std::move(message)) {}
    };

    constexpr DyLib() noexcept = default;

    DyLib(const DyLib&) = delete;
    DyLib& operator=(const DyLib&) = delete;

    DyLib(DyLib &&other) noexcept
    {
        m_handle = other.m_handle;
        other.m_handle = nullptr;
    }

    DyLib& operator=(DyLib &&other) noexcept
    {
        if (this != &other) {
            close();
            m_handle = other.m_handle;
            other.m_handle = nullptr;
        }
        return *this;
    }

    explicit DyLib(const char *path)
    {
        open(path);
    }

    explicit DyLib(const std::string &path)
    {
        open(path.c_str());
    }

    DyLib(std::string path, const char *ext)
    {
        open(std::move(path), ext);
    }

    ~DyLib()
    {
        close();
    }

    void open(const char *path)
    {
        close();
        if (!path)
            throw handle_error(getHandleError("(nullptr)"));
        m_handle = openLib(path);
        if (!m_handle)
            throw handle_error(getHandleError(path));
    }

    void open(const std::string &path)
    {
        open(path.c_str());
    }

    void open(std::string path, const char *ext)
    {
        close();
        if (!ext)
            throw handle_error("bad extension : (nullptr)");
        path += ext;
        m_handle = openLib(path.c_str());
        if (!m_handle)
            throw handle_error(getHandleError(path));
    }

    template<typename T>
    std::function<T> getFunction(const char *name) const
    {
        if (!m_handle)
            throw handle_error("error : no dynamic library loaded");
        if (!name)
            throw symbol_error(getSymbolError("(nullptr)"));
        auto sym = getSymbol(name);
        if (!sym)
            throw symbol_error(getSymbolError(name));
        return reinterpret_cast<T *>(sym);
    }

    template<typename T>
    std::function<T> getFunction(const std::string &name) const
    {
        return getFunction<T>(name.c_str());
    }

    template<typename T>
    T &getVariable(const char *name) const
    {
        if (!m_handle)
            throw handle_error("error : no dynamic library loaded");
        if (!name)
            throw symbol_error(getSymbolError("(nullptr)"));
        auto sym = getSymbol(name);
        if (!sym)
            throw symbol_error(getSymbolError(name));
        return *reinterpret_cast<T *>(sym);
    }

    template<typename T>
    T &getVariable(const std::string &name) const
    {
        return getVariable<T>(name.c_str());
    }

    void close() noexcept
    {
        if (m_handle) {
            closeLib();
            m_handle = nullptr;
        }
    }
};
```

5.3 Components.hpp

```
#pragma once

#include <string>

namespace ecs {
namespace component {
    struct position {
        int x, y, vx, vy, cur;
    };
    using pv = int;
    using label = std::string;
}
}
```

5.4 Entity.hpp

```
#pragma once

#include <cstddef>

namespace ecs {
    using entity = size_t;
}
```

5.5 Exception.hpp

```
#pragma once

#include <exception>
#include <string>
#include <utility>

namespace ecs {
class exception : public std::exception
{
private:
    std::string m_err;
public:
    explicit exception(std::string err) : m_err(std::move(err)) {}
    template<typename ...T>
    exception(T &&...err) : m_err() {m_err.append(std::forward<T>(err)...);}
    ~exception() override = default;
    [[nodiscard]] inline const char *what() const noexcept override {return m_err.c_str();}
};
}
```

5.6 Module.hpp

```
#pragma once

#include <string>
#include "Registry.hpp"

namespace ecs {
class module
{
public:
    virtual ~module() = default;
    virtual ecs::entity spawn(registry &) = 0;
    virtual std::function<void(registry &)> get_system() = 0;
    virtual std::string get_label() = 0;
};
}
```

5.7 Registry.hpp

```
#pragma once

#include <map>
#include <any>
#include <typeindex>
#include <functional>

#include "Entity.hpp"
#include "SparseArray.hpp"
#include "Exception.hpp"

namespace ecs {
class registry {
public:

    template<typename T>
    sparse_array<T> &register_component() {
        if (!has_component<T>())
            m_map[std::type_index(typeid(T))] = std::make_any<sparse_array<T>>();
        m_deleter.push_back([&](entity ent) {
            std::any_cast<sparse_array<T> &>(m_map[std::type_index(typeid(T))]).erase(ent);
        });
        return std::any_cast<sparse_array<T> &>(m_map[std::type_index(typeid(T))]);
    }

    template<class T>
    bool has_component() {
        if (m_map.contains(std::type_index(typeid(T))))
            return true;
        return false;
    }

    template <typename T>
    sparse_array<T> &get_components() {
        if (!has_component<T>())
            throw exception("registry::get_components");
        return std::any_cast<sparse_array<T> &>(m_map[std::type_index(typeid(T))]);
    }

    template <typename T>
    const sparse_array<T> &get_components() const {
        if (!has_component<T>())
            throw exception("registry::get_components");
        return std::any_cast<const sparse_array<T> &>(m_map.at(std::type_index(typeid(T))));
    }

    template <typename T>
    typename sparse_array<T>::reference_type insert_component(const entity &ent, T &&comp) {
        if (!has_component<T>())
            throw exception("registry::insert_component");
        return std::any_cast<sparse_array<T> &>(m_map[std::type_index(typeid(T))]).insert_at(ent,
    std::forward<T>(comp));
    }

    template <typename T, typename ...Params>
    typename sparse_array<T>::reference_type emplace_component(const entity &ent, Params && ...params) {
        if (!has_component<T>())
            throw exception("registry::emplace_component");
        return std::any_cast<sparse_array<T> &>(m_map[std::type_index(typeid(T))]).emplace_at(ent,
    std::forward<Params>(params)...);
    }

    template <typename T>
    void remove_component(const entity &ent) {
        if (!has_component<T>())
            throw exception("registry::remove_component");
        std::any_cast<sparse_array<T> &>(m_map[std::type_index(typeid(T))]).erease(ent);
    }


    entity spawn_entity() {
        if (m_ents_available.empty()) {
            m_ent_iter++;
            m_pool.push_back(m_ent_iter);
            return m_ent_iter;
        }
        auto it = m_ents_available.front();
        m_ents_available.erase(m_ents_available.begin());
        m_pool.push_back(it);
        return it;
    }

    void kill_entity(entity e) {
        auto res = std::find(m_pool.begin(), m_pool.end(), e);
        if (res == std::end(m_pool))
            throw exception("entity_pool::kill_entity");
        m_pool.erase(res);
        m_ents_available.push_back(e);
        for (auto &del : m_deleter)
            del(e);
    }

    const std::vector<entity> &get_entities() {
        return m_pool;
    }

    template <typename Function>
    void add_system(Function && f) {
        m_systems.emplace_back([&]() {
            std::forward<Function>(f)(*this);
        });
    }

    void run_systems() {
        for (auto &it : m_systems)
            it();
    }

    void destroy() {
        m_map.clear();
        m_pool.clear();
        m_ents_available.clear();
        m_deleter.clear();
        m_systems.clear();
        m_ent_iter = 0;
    }

private:
    std::map<std::type_index, std::any> m_map{};
    std::vector<entity> m_pool{};
    std::vector<entity> m_ents_available{};
    std::vector<std::function<void(entity)» m_deleter{};
    std::vector<std::function<void()» m_systems{};
    entity m_ent_iter{};
```

```cpp
};
}
```

5.8 SparseArray.hpp

```cpp
#pragma once

#include <optional>
#include <vector>
#include <utility>

namespace ecs {
template <typename T>
class sparse_array {
public:
    using value_type = std::optional<T>;
    using reference_type = value_type &;
    using const_reference_type = const value_type &;
    using container_t = std::vector<value_type>;
    using size_type = typename container_t::size_type;
    using iterator = typename container_t::iterator;
    using const_iterator = typename container_t::const_iterator;

    sparse_array() = default;
    sparse_array(sparse_array const &) = default;
    sparse_array(sparse_array &&) noexcept = default;
    ~sparse_array() = default;
    sparse_array &operator=(sparse_array const &) = default;
    sparse_array &operator=(sparse_array &&) noexcept = default;
    iterator begin() {return m_data.begin();}
    const_iterator begin() const {return m_data.begin();}
    const_iterator cbegin() const {return m_data.cbegin();}
    iterator end() {return m_data.end();}
    const_iterator end() const {return m_data.end();}
    const_iterator cend() const {return m_data.cend();}
    size_type size() const {return m_data.size();}
    reference_type operator[](size_t idx) {
        if (idx >= size())
            m_data.resize(idx + 1);
        return m_data[idx];
    }
    const_reference_type operator[](size_t idx) const {
        if (idx >= size())
            m_data.resize(idx + 1);
        return m_data[idx];
    }

    reference_type insert_at(size_type pos, const T &comp) {
        if (pos >= size())
            m_data.resize(pos + 1);
        m_data[pos] = std::make_optional<T>(comp);
        return m_data[pos];
    }

    reference_type insert_at(size_type pos, T &&comp) {
        if (pos >= size())
            m_data.resize(pos + 1);
        m_data[pos] = std::make_optional<T>(std::forward<T>(comp));
        return m_data[pos];
    }

    template <class ...Params>
    reference_type emplace_at(size_type pos, Params &&...params) {
        if (pos >= size())
            m_data.resize(pos + 1);
        m_data[pos] = std::make_optional<T>(std::forward<T>(params)...);
        return m_data[pos];
    }

    void erase(size_type pos) {
        if (pos < m_data.size())
            m_data[pos] = std::nullopt;
    }

    size_type get_index(value_type const &val) const {
        for (size_t i = 0; i < m_data.size(); i++) {
            if (m_data[i] == val)
                return i;
        }
        return -1;
    }
private:
    mutable container_t m_data{};
};
}
```

5.9 Systems.hpp

```
#pragma once

#include "Registry.hpp"
#include "Components.hpp"
#include "UI/SpriteSize.hpp"
#include <iostream>

namespace ecs {
namespace system {
    inline void create_players(registry &reg, size_t client_size) {
        for (size_t i = 0; i < client_size; i++) {
            auto ent = reg.spawn_entity();
            reg.insert_component<ecs::component::label>(ent, "player" + std::to_string(i));
            reg.insert_component<ecs::component::position>(ent, {0, 100 + static_cast<int>(100 * i), 0, 0, 0});
            reg.insert_component<ecs::component::pv>(ent, 1);
        }
    }

    inline void move_player(registry &reg, size_t client_pos, int x, int y) {
        auto &positions = reg.get_components<ecs::component::position>();
        auto &tags = reg.get_components<ecs::component::label>();
        for (auto &e : reg.get_entities()) {
            if (tags[e].value() == "player" + std::to_string(client_pos)) {
                auto &val = positions[e].value();
                if (y > 0)
                    val.vy = 1;
                else if (y < 0)
                    val.vy = -1;
                else
                    val.vy = 0;
                val.x += x * 15;
                val.y += y * 15;
                if (val.x < 0)
                    val.x = 0;
                if (val.y < 0)
                    val.y = 0;
                if (val.x > 1856)
                    val.x = 1856;
                if (val.y > 1052)
                    val.y = 1052;
                return;
            }
        }
    }

    inline void create_friend_shoot(registry &reg, size_t client_pos) {
        auto &tags = reg.get_components<ecs::component::label>();
        auto &positions = reg.get_components<ecs::component::position>();
        for (auto player : reg.get_entities()) {
            if (tags[player].value() == "player" + std::to_string(client_pos)) {
                auto ent = reg.spawn_entity();
                auto player_pos = positions[player].value();
                reg.insert_component<ecs::component::label>(ent, "friend_shoot");
                reg.insert_component<ecs::component::position>(ent,
                    {player_pos.x + static_cast<int>(Data::sprite_data["player"].first.x * Data::sprite_data["player"].second.x),
                    player_pos.y + static_cast<int>((Data::sprite_data["player"].first.y * Data::sprite_data["player"].second.y) / 2), 20, 0, 0});
                reg.insert_component<ecs::component::pv>(ent, 1);
                return;
            }
        }
    }

    inline void create_enemy_shoot(registry &reg, size_t enemy_ent, int speed = -5) {
        auto &label = reg.get_components<ecs::component::label>();
        auto &positions = reg.get_components<ecs::component::position>();

        auto ent = reg.spawn_entity();
        auto enemy_pos = positions[enemy_ent].value();
        reg.insert_component<ecs::component::label>(ent, "enemy_shoot");
        reg.insert_component<ecs::component::position>(ent,
        {enemy_pos.x, enemy_pos.y + static_cast<int>((Data::sprite_data[label[enemy_ent].value()].first.y
        * Data::sprite_data[label[enemy_ent].value()].second.y) / 2), speed, 0, 0});
        reg.insert_component<ecs::component::pv>(ent, 1);
    }

```

```
inline void update_velocity(registry &reg) {
    auto &positions = reg.get_components<ecs::component::position>();
    auto &tags = reg.get_components<ecs::component::label>();
    auto &pvs = reg.get_components<ecs::component::pv>();
    for (auto &e : reg.get_entities()) {
        if (tags[e].value().find("player") != 0) {
            positions[e].value().x += positions[e].value().vx;
            positions[e].value().y += positions[e].value().vy;
            positions[e].value().cur += positions[e].value().vy;
            if (positions[e].value().x > 1920 or positions[e].value().x < -100)
                pvs[e].value() = 0;
        }
    }
}

inline void entity_killer(registry &reg) {
    auto &pvs = reg.get_components<ecs::component::pv>();
    for (auto &e : reg.get_entities()) {
        if (pvs[e].value() <= 0) {
            reg.kill_entity(e);
            return;
        }
    }
}

 bool is_collision(registry &reg, entity e1, entity e2) {
     auto &label = reg.get_components<ecs::component::label>();
     auto &l1 = label[e1].value();
     auto &l2 = label[e2].value();
     auto &positions = reg.get_components<ecs::component::position>();
     auto &pos1 = positions[e1].value();
     auto &pos2 = positions[e2].value();
     float rect1[2] = {Data::sprite_data[l1].first.x * Data::sprite_data[l1].second.x,
                     Data::sprite_data[l1].first.y * Data::sprite_data[l1].second.y};
     float rect2[2] = {Data::sprite_data[l2].first.x * Data::sprite_data[l2].second.x,
                     Data::sprite_data[l2].first.y * Data::sprite_data[l2].second.y};

     if (pos1.x >= pos2.x && pos1.x <= pos2.x + rect2[0] && pos1.y >= pos2.y && pos1.y <= pos2.y +
rect2[1])
         return true;
     if (pos1.x + rect1[0] >= pos2.x && pos1.x + rect1[0] <= pos2.x + rect2[0] && pos1.y >= pos2.y &&
pos1.y <= pos2.y + rect2[1])
         return true;
     if (pos1.x >= pos2.x && pos1.x <= pos2.x + rect2[0] && pos1.y + rect1[1] >= pos2.y && pos1.y +
rect1[1] <= pos2.y + rect2[1])
         return true;
     if (pos1.x + rect1[0] >= pos2.x && pos1.x + rect1[0] <= pos2.x + rect2[0] && pos1.y + rect1[1]
>= pos2.y && pos1.y + rect1[1] <= pos2.y + rect2[1])
         return true;
     if (pos2.x >= pos1.x && pos2.x <= pos1.x + rect1[0] && pos2.y >= pos1.y && pos2.y <= pos1.y +
rect1[1])
         return true;
     if (pos2.x + rect2[0] >= pos1.x && pos2.x + rect2[0] <= pos1.x + rect1[0] && pos2.y >= pos1.y &&
pos2.y <= pos1.y + rect1[1])
         return true;
     if (pos2.x >= pos1.x && pos2.x <= pos1.x + rect1[0] && pos2.y + rect2[1] >= pos1.y && pos2.y +
rect2[1] <= pos1.y + rect1[1])
         return true;
     if (pos2.x + rect2[0] >= pos1.x && pos2.x + rect2[0] <= pos1.x + rect1[0] && pos2.y + rect2[1]
>= pos1.y && pos2.y + rect2[1] <= pos1.y + rect1[1])
         return true;
     return false;
 }

 inline void collision_system(registry &reg) {
     auto &labels = reg.get_components<ecs::component::label>();
     auto &pvs = reg.get_components<ecs::component::pv>();
     for (auto e1 : reg.get_entities()) {
         for (auto e2 : reg.get_entities()) {
             if (e1 != e2 and is_collision(reg, e1, e2)) {
                 if ((labels[e1].value().find("enemy") == 0 and labels[e2].value().find("player") ==
0) and
                     labels[e1].value().find("enemy_shoot") != 0) {
                     pvs[e1].value() -= 1;
                     pvs[e2].value() -= 1;
                 }
                 if ((labels[e1].value().find("enemy_shoot") == 0 and
labels[e2].value().find("player") == 0) and
                     labels[e2].value().find("player_shoot") != 0) {
                     pvs[e1].value() -= 1;
                     pvs[e2].value() -= 1;
                 }
                 if ((labels[e1].value().find("friend_shoot") == 0 and
labels[e2].value().find("enemy") == 0) and
                     labels[e2].value().find("enemy_shoot") != 0) {
                     pvs[e1].value() -= 1;
                     pvs[e2].value() -= 1;
```

```
                    }
                }
            }
        }
    }
}
}
```

5.10 Client.hpp

```
#pragma once

#include <asio.hpp>
#include "Packet.hpp"

namespace network {
class client {
private:
    asio::ip::udp::endpoint endpoint;
     std::vector<packet> buffer{};
     bool m_alive = true;
public:
     client(asio::ip::udp::endpoint &&endp) : endpoint(std::move(endp)) {}
     ~client() = default;
     std::string get_ip() const {return endpoint.address().to_string();}
     unsigned short get_port() {return endpoint.port();}
     const asio::ip::udp::endpoint &get_endpoint() const noexcept {return endpoint;}
     std::vector<packet> &get_packets() noexcept {return buffer;}
     std::vector<packet> pop_packets() noexcept {return std::move(buffer);}
     bool &alive() {return m_alive;}
};
}
```

5.11 Header.hpp

```
#pragma once

#include <cstdint>

namespace network {
struct header {
    uint32_t magicValue;
    uint8_t type;
    uint32_t size;
};
}
```

5.12 Packet.hpp

```
#pragma once

#include <cstdint>
#include <string>

namespace network {
struct packet {
    uint8_t type;
    std::string data;
};
}
```

5.13 Server.hpp

```
#pragma once

#include <asio.hpp>
#include <iostream>
#include <memory>
#include <vector>
#include <chrono>

```

```cpp
#include "Header.hpp"
#include "Packet.hpp"
#include "Client.hpp"

namespace network {
class server {
private:
    asio::io_context m_io_context{};
    asio::ip::udp::socket m_socket;
    std::vector<client> m_clients{};
    uint8_t m_login_handshake;
    uint8_t m_logout_handshake;
    uint8_t m_alive_handshake;
    size_t m_max_clients;
    bool m_allow_new_clients = true;
    std::chrono::system_clock::time_point m_check_alive = std::chrono::system_clock::now();
public:
    server(unsigned short port, size_t max_clients, uint8_t login_handshake, uint8_t logout_handshake, uint8_t alive_handshake)
        : m_socket(m_io_context, asio::ip::udp::endpoint(asio::ip::udp::v4(), port)),
        m_login_handshake(login_handshake), m_logout_handshake(logout_handshake), m_alive_handshake(alive_handshake), m_max_clients(max_clients) {
        m_socket.non_blocking(true);
    }
    ~server() = default;
    server(server &) = delete;
    server &operator=(server &) = delete;
    server(server &&) = delete;
    server &operator=(server &&) = delete;

    void send(const client &c, uint8_t type, const std::string &buffer = "", unsigned short repeat = 1) {
        size_t size = buffer.size();
        auto p = reinterpret_cast<header *>(::operator new (sizeof(header) + size));
        p->magicValue = 0x42dead42;
        p->type = type;
        p->size = buffer.size();
        std::memcpy(reinterpret_cast<uint8_t *>(p) + sizeof(header), buffer.data(), size);
        try {
            while (repeat != 0) {
                m_socket.send_to(asio::buffer(reinterpret_cast<const uint8_t *>(p), sizeof(header) + size), c.get_endpoint());
                repeat--;
            }
        } catch (...) {}
        delete p;
    }
    void receive() {
        if (std::chrono::system_clock::now() > m_check_alive + std::chrono::seconds(10)) {
            m_check_alive = std::chrono::system_clock::now();
            for (size_t i = 0; i < m_clients.size(); i++) {
                if (m_clients[i].alive() == false) {
                    std::cout « "[USER] " « m_clients[i].get_ip() « " : DISCONNECTED" « std::endl;
                    m_clients.erase(m_clients.begin() + i);
                    i--;
                } else {
                    m_clients[i].alive() = false;
                }
            }
        }
        while (true) {
            asio::ip::udp::endpoint endpoint;
            uint8_t recv_str[4800];
            asio::error_code error;
            auto len = m_socket.receive_from(asio::buffer(recv_str, 4800), endpoint, 0, error);
            if (error == asio::error::would_block)
                return;
            std::string data{};
            auto ret = reinterpret_cast<const header *>(recv_str);
            if (ret->magicValue != 0x42dead42 or len != sizeof(header) + (ret->size))
                return;
            for (size_t i = 0; i < ret->size; i++)
                data.push_back(((reinterpret_cast<const uint8_t *>(ret) + sizeof(header)))[i]);
            for (size_t i = 0; i < m_clients.size(); i++) {
                if (m_clients[i].get_ip() == endpoint.address().to_string() and m_clients[i].get_port() == endpoint.port()) {
                    if (ret->type == m_alive_handshake) {
                        m_clients[i].alive() = true;
                        continue;
                    } else if (ret->type == m_logout_handshake) {
                        std::cout « "[USER] " « m_clients[i].get_ip() « " : DISCONNECTED" « std::endl;
                        m_clients.erase(m_clients.begin() + i);
                        return;
                    }
                    m_clients[i].get_packets().push_back({ret->type, std::move(data)});
                    return;
                }
            }
            if (ret->type == m_login_handshake and m_clients.size() < m_max_clients and m_allow_new_clients == true) {
                m_clients.push_back(std::move(endpoint));
                std::cout « "[USER] " « m_clients.back().get_ip() « " : CONNECTED" « std::endl;
                m_check_alive = std::chrono::system_clock::now();
                send(m_clients.back(), m_login_handshake, "", 10);
            }
        }
    }
    std::vector<client> &get_clients() noexcept {
        return m_clients;
    }
    void allow_new_connections(bool value) {m_allow_new_clients = value;}
};
}
```

5.14 Socket.hpp

```
#pragma once

#include <asio.hpp>
#include <thread>
#include <memory>
#include <vector>
#include <chrono>
#include <mutex>
#include <exception>

#include "Header.hpp"
#include "Packet.hpp"

namespace network {
class socket {
private:
    asio::io_context m_io_context{};
    asio::ip::udp::socket m_socket;
    std::unique_ptr<asio::ip::udp::endpoint> m_server{};
    std::unique_ptr<std::thread> m_receiver{};
    std::mutex m_mutex{};
    std::vector<packet> m_buffer{};
    uint8_t m_login_handshake;
    uint8_t m_logout_handshake;
    uint8_t m_alive_handshake;
    bool m_connected{};
public:
    socket(unsigned short port, uint8_t login_handshake, uint8_t logout_handshake, uint8_t alive_handshake)
        : m_socket(m_io_context, asio::ip::udp::endpoint(asio::ip::udp::v4(), port)),
        m_login_handshake(login_handshake), m_logout_handshake(logout_handshake), m_alive_handshake(alive_handshake) {
        m_socket.non_blocking(true);
    }
    ~socket() {disconnect();}
    socket(socket &) = delete;
    socket &operator=(socket &) = delete;
    socket(socket &&) = delete;
    socket &operator=(socket &&) = delete;

    void connect(std::string ip, unsigned short port, long login_timeout = 3) {
        disconnect();
        std::this_thread::sleep_for(std::chrono::seconds(1));
        try {
            std::chrono::system_clock::time_point deadline = std::chrono::system_clock::now() + std::chrono::seconds(login_timeout);
            m_server = std::make_unique<asio::ip::udp::endpoint>(asio::ip::make_address(std::move(ip)), port);
            m_connected = true;
            m_receiver = std::make_unique<std::thread>(receive, std::ref(*this));
            send(m_login_handshake, "", 10);
            while (std::chrono::system_clock::now() < deadline) {
                auto packets = pop_packets();
                for (auto &p : packets) {
                    if (p.type == m_login_handshake)
                        return;
                }
                std::this_thread::yield();
            }
            throw std::exception();
        } catch (...) {
            disconnect();
            throw std::runtime_error("network::client::connect : could not connect to " + ip + ":" + std::to_string(port));
        }
```

```cpp
    }

    void disconnect() noexcept {
        if (m_receiver) {
            m_connected = false;
            m_receiver->join();
        }
        try {
            send(m_logout_handshake, "", 10);
        } catch (...) {}
        m_server = nullptr;
        m_receiver = nullptr;
        m_buffer.clear();
    }

    void send(uint8_t type, const std::string &buffer = "", unsigned short repeat = 1) {
        if (!m_server)
            throw std::runtime_error("network::client::send : not connected to any host");
        size_t size = buffer.size();
        auto p = reinterpret_cast<header *>(::operator new (sizeof(header) + size));
        p->magicValue = 0x42dead42;
        p->type = type;
        p->size = buffer.size();
        std::memcpy(reinterpret_cast<uint8_t *>(p) + sizeof(header), buffer.data(), size);
        try {
            while (repeat != 0) {
                m_socket.send_to(asio::buffer(reinterpret_cast<const uint8_t *>(p), sizeof(header) + size), *m_server);
                repeat--;
            }
        } catch (...) {}
        delete p;
    }

    std::string get_remote_ip() const {
        if (!m_server)
            return "";
        return m_server->address().to_string();
    }

    unsigned short get_remote_port() const {
        if (!m_server)
            return 0;
        return m_server->port();
    }

    std::vector<packet> get_packets() noexcept {
        std::lock_guard lock(m_mutex);
        return m_buffer;
    }

    std::vector<packet> pop_packets() noexcept {
        std::lock_guard lock(m_mutex);
        return std::move(m_buffer);
    }
private:
    static void receive(socket &self) {
        std::chrono::system_clock::time_point chrono = std::chrono::system_clock::now();

        if (!self.m_server)
            throw std::runtime_error("network::client::receive : not connected to any host");

        while (self.m_connected) {
            if (std::chrono::system_clock::now() > chrono + std::chrono::seconds(1)) {
                chrono = std::chrono::system_clock::now();
                self.send(self.m_alive_handshake, "", 1);
            }
            asio::ip::udp::endpoint endpoint;
            uint8_t recv_str[4800];
            asio::error_code error;
            std::string data{};
            auto len = self.m_socket.receive_from(asio::buffer(recv_str, 4800), endpoint, 0, error);
            if (error == asio::error::would_block)
                continue;
            if (self.get_remote_ip() != endpoint.address().to_string() or self.get_remote_port() != endpoint.port())
                continue;
            auto ret = reinterpret_cast<const header *>(recv_str);
            if (ret->magicValue == 0x42dead42 and len == sizeof(header) + (ret->size)) {
                for (size_t i = 0; i < ret->size; i++)
                    data.push_back(((reinterpret_cast<const uint8_t *>(ret) + sizeof(header)))[i]);
                self.m_mutex.lock();
                self.m_buffer.push_back({ret->type, std::move(data)});
                self.m_mutex.unlock();
            }
            std::this_thread::yield();
        }
    }
};
}
```

5.15 Type.hpp

```
#pragma once

#include <cstdint>

namespace network {
enum type : uint8_t {
    Login,
    Logout,
    PingAlive,
    UsernameOK,
    UsernameKO,
    Lobby,
    Ready,
    NotReady,
    GameStart,
    GameUpdate,
    PlayerInput,
    PlayerDead,
};
}
```

5.16 Random.hpp

```
#pragma once

namespace tools {
    inline int random(int max) {
        srand(time(NULL));
        return rand() % max;
    }
}
```

5.17 ThreadPool.hpp

```
#pragma once

#include <thread>
#include <vector>
#include <utility>

class thread_pool
{
private:
    std::vector<std::thread> m_threads{};
public:
    thread_pool() = default;
    thread_pool(const thread_pool&) = delete;
    thread_pool(thread_pool&&) = delete;
    thread_pool &operator=(const thread_pool&) = delete;
    thread_pool &operator=(thread_pool&&) = delete;
    ~thread_pool() {
        release();
    };
    template<typename Function, typename ...Args>
    void add(Function &&fn, Args &&...args) {
        m_threads.emplace_back(std::forward<Function>(fn), std::forward<Args>(args)...);
    }
    void remove(size_t position) {
        if (position >= m_threads.size())
            throw std::out_of_range("thread_pool::remove");
        m_threads[position].join();
        m_threads.erase(m_threads.begin() + position);
    }
    void release() {
        for (auto &thread : m_threads)
            thread.join();
        m_threads.clear();
    }
};
```

5.18 Tools.hpp

```
#pragma once

#include <string>
#include <vector>
#include <SFML/Graphics.hpp>
#include "UI/Spritesheet.hpp"
#include "UI/Inputbox.hpp"

namespace tools {
    inline std::vector<std::string> string_to_vector(const std::string &str, char separator)
    {
        std::vector<std::string> array{};
        std::string temp{};
        size_t len = str.size();

        for (size_t i = 0; i < len; i++) {
            if (str[i] == separator) {
                array.push_back(temp);
                temp.clear();
            }
            else
                temp.push_back(str[i]);
        }
        if (temp.size() != 0) {
            array.push_back(temp);
        }
        return array;
    }
    inline bool mouse_is_in_shape(sf::Vector2i pmouse, sf::RectangleShape shape)
    {
        if (pmouse.x >= shape.getPosition().x && pmouse.x <= shape.getPosition().x + shape.getSize().x
        && pmouse.y >= shape.getPosition().y && pmouse.y <= shape.getPosition().y + shape.getSize().y) {
            return true;
        }
        return (false);
    }
    inline void state_button_login(const std::vector<Inputbox> &box, Spritesheet &sprite)
    {
        for (auto &c : box) {
            if (c.getText().getSize() == 0) {
                sprite.playImage(0);
                return;
            }
        }
        sprite.playImage(1);
    }
}
```

5.19 TypeChecker.hpp

```
#pragma once

#include <type_traits>

template<typename T, typename ...List>
concept TypeChecker = (std::is_same_v<T, List> || ...);
```

5.20 Core.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include "Network/Type.hpp"
#include "Network/Socket.hpp"
#include "UI/Status.hpp"
#include "UI/Menu.hpp"
#include "UI/Lobby.hpp"
#include "UI/Game.hpp"


class Core {
    public:
        Core();
        ~Core() = default;

        void loop();

    private:
        uint8_t m_status = MENU;
        sf::RenderWindow m_window{};
        network::socket m_socket;
        Menu m_menu;
        Lobby m_lobby;
        Game m_game;
};
```

5.21 Data.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <exception>
#include <map>
#include <functional>
#include "UI/Spritesheet.hpp"

namespace Data {
    class MissingAsset : public std::exception {
    private:
        const std::string m_error{};
    public:
        explicit MissingAsset(const std::string &path) : m_error("missing asset : " + path) {}
        ~MissingAsset() override = default;
        [[nodiscard]] inline const char *what() const noexcept override {return m_error.c_str();}
    };

    inline sf::Texture backgroundTexture{};
    inline sf::Font font{};

    inline sf::Music menuMusic{};
    inline sf::Music gameMusic{};

    inline sf::Texture tmenu{};
    inline sf::Texture tloginbutton{};
    inline sf::Texture tprofile{};
    inline sf::Texture tready{};
    inline sf::Texture treadybutton{};
    inline sf::Texture tdisconnectbutton{};
    inline sf::Texture tloading{};
    inline sf::Texture tmap{};
    inline sf::Texture tmapdead{};
    inline sf::Texture tplayer0{};
    inline sf::Texture tplayer1{};
    inline sf::Texture tplayer2{};
    inline sf::Texture tplayer3{};
    inline sf::Texture tenemy0{};
    inline sf::Texture tenemy1{};
    inline sf::Texture tenemy2{};
    inline sf::Texture tplayershot{};
    inline sf::Texture tenemyshot{};
    inline sf::Texture tlobbybutton{};
    inline sf::Texture tdisconnectbuttongame{};

    inline sf::Sound splayershot{};
    inline sf::SoundBuffer bplayershot{};


    inline std::map<std::string, std::function<Spritesheet()» factorytexturemap {
        {"player0", {[](){return Spritesheet(tplayer0, {0, 0}, 5, {160, 14}, {0, 0, 32, 14}, {2.5, 2.5},
    32);}}},
        {"player1", {[](){return Spritesheet(tplayer1, {0, 0}, 5, {160, 14}, {0, 0, 32, 14}, {2.5, 2.5},
    32);}}},
        {"player2", {[](){return Spritesheet(tplayer2, {0, 0}, 5, {160, 14}, {0, 0, 32, 14}, {2.5, 2.5},
    32);}}},
        {"player3", {[](){return Spritesheet(tplayer3, {0, 0}, 5, {160, 14}, {0, 0, 32, 14}, {2.5, 2.5},
    32);}}},
        {"enemy_plane", {[](){return Spritesheet(tenemy0, {0, 0}, 8, {168, 24}, {0, 0, 21, 24}, {2.5,
    2.5}, 21);}}},
        {"enemy_cyborg", {[](){return Spritesheet(tenemy1, {0, 0}, 4, {128, 31}, {0, 0, 32, 31}, {3, 3},
    32);}}},
        {"enemy2", {[](){return Spritesheet(tenemy1, {0, 0}, 4, {256, 22}, {0, 0, 32, 22}, {2, 2},
    32);}}},
        {"friend_shoot", {[](){return Spritesheet(tplayershot, {0, 0}, 1, {16, 4}, {0, 0, 16, 4}, {2,
    2});}}},
        {"enemy_shoot", {[](){return Spritesheet(tenemyshot, {0, 0}, 4, {28, 6}, {0, 0, 7, 6}, {2, 2},
    7);}}}
    };
```

```cpp

    void load();

    inline void loadAsset(sf::Music &asset, const std::string &path) {
        if (!asset.openFromFile(path))
            throw MissingAsset(path);
    }

    template<typename T>
    inline void loadAsset(T &asset, const std::string &path) {
        if (!asset.loadFromFile(path))
            throw MissingAsset(path);
    }
};
```

5.22 Game.hpp

```cpp
#pragma once

#include <SFML/Graphics.hpp>
#include "ECS/SparseArray.hpp"
#include "Network/Socket.hpp"
#include "UI/Spritesheet.hpp"
#include "UI/Textbox.hpp"
#include "UI/Status.hpp"
#include "UI/Data.hpp"

class Game {
    public:
        Game(sf::RenderWindow &window, network::socket &socket, uint8_t &status);
        ~Game() = default;

        void gameScreen();

        void getServState();

        void parseData(const std::string &str);


        void removeKilled(std::vector<size_t> id);

        void checkInput();

        void gameEvents();

        void eventsButton();

        void eventsPressedButton();

        void animSprites();

        void displayGame();

    private:
        sf::RenderWindow &m_window;
        network::socket &m_socket;
        uint8_t &m_status;
        Spritesheet m_map{Data::tmap, {0, 0}, 1023, {6016, 1080}, {0, 0, 1920, 1080}, {1, 1}, 4};
        sf::Event m_event{};
        sf::Clock m_clock{};
        sf::Clock m_clockinputs{};
        sf::Clock m_clockshots{};
        sf::Clock m_clockanim{};
        sf::Cursor m_cursor{};
        Spritesheet m_lobbybutton{Data::tlobbybutton, {571, 880}, 2, {540, 135}, {0, 0, 270, 135}, {0.7,
    0.5}, 270};
        Spritesheet m_disconnectbutton{Data::tdisconnectbuttongame, {1160, 880}, 2, {540, 135}, {0, 0,
    270, 135}, {0.7, 0.5}, 270};
        sf::Color m_grey{169, 169, 169, 255};
        sf::Color m_yellow{252, 245, 148, 255};
        sf::Color m_red{255, 80, 37, 255};
        Textbox m_lobbytext{30, "Lobby", m_yellow, {621, 893}};
        Textbox m_disconnecttext{24, "Disconnect", m_red, {1190, 898}};
        Textbox m_deadtext{40, "You are dead!", m_red, {825, 214}};
        sf::RectangleShape m_deadrectangle{{400, 80}};
        bool m_isdead{};
        ecs::sparse_array<Spritesheet> m_display{};
};
```

5.23 Inputbox.hpp

```
#pragma once

#include <iostream>
#include <SFML/Graphics.hpp>

constexpr auto DELETE_KEY = 8;
constexpr auto ESCAPE_KEY = 27;

class Inputbox : public sf::Drawable {
    public:
        Inputbox(int size = 15, sf::Color color = sf::Color::White, bool select = false, size_t lim = 0);
        ~Inputbox() = default;

        void inputLogic(char c);

        void setFont(const sf::Font &font);

        void setPosition(sf::Vector2f pos);

        void setLimit(size_t lim);

        void setSelected(bool sel);

        void setString(sf::String str);

        sf::String &getText();

        const sf::String &getText() const;

        bool &getSelected();

        sf::Text &getInputbox();

        void updateString();

        void draw(sf::RenderTarget &target, sf::RenderStates states) const override;

    private:
        sf::Text m_inputbox{};
        sf::String m_text{};
        sf::Vector2f m_pos{};
        bool m_selected = false;
        size_t m_limit = 0;
};
```

5.24 Lobby.hpp

```
#pragma once

#include <mutex>
#include <SFML/Graphics.hpp>
#include "UI/Spritesheet.hpp"
#include "UI/Profile.hpp"
#include "UI/Data.hpp"
#include "Network/Socket.hpp"

class Lobby {
    public:
        Lobby(sf::RenderWindow &window, network::socket &socket, uint8_t &status);
        ~Lobby();

        void initLobby();

        void endLobby();

        void lobbyScreen();

        void eventsLobby();

        void eventsReady();

        void eventsDisconnect();

        void eventsPressButton();

        void reqLobby();

        void parseData(const std::string &data);

        void displayLobby();

    private:
        sf::RenderWindow &m_window;
        network::socket &m_socket;
        uint8_t &m_status;
        Spritesheet m_menu{Data::tmenu, {0, 0}, 125, {3600, 10125}, {0, 0, 720, 405}, {2.667, 2.667}, 720};
        Spritesheet m_readybutton{Data::treadybutton, {870, 760}, 4, {540, 270}, {0, 0, 270, 135}, {0.7, 0.5}, 270};
        Spritesheet m_disconnectbutton{Data::tdisconnectbutton, {540, 760}, 2, {540, 135}, {0, 0, 270, 135}, {0.7, 0.5}, 270};
        sf::Color m_red{255, 80, 37, 255};
        sf::Color m_green{59, 184, 115, 255};
        Textbox m_lobbytext{60, "Lobby", sf::Color::Black, {880, 150}};
        Textbox m_readytext{30, "Ready", m_red, {920, 773}};
        Textbox m_disconnecttext{24, "Disconnect", sf::Color::Black, {570, 778}};
        sf::RectangleShape m_lobbyrectangle{{1000, 750}};
        sf::Color m_grey{180, 180, 180, 240};
        sf::Event m_event{};
        sf::Cursor m_cursor{};
        sf::Clock m_clock{};
        std::vector<Profile> m_profiles{};
        std::unique_ptr<std::thread> m_thread{};
        std::mutex m_lobbylock{};
        bool m_ready{};
        bool m_lobbyloop = true;
};
```

5.25 Menu.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include "Network/Socket.hpp"
#include "UI/Inputbox.hpp"
#include "UI/Textbox.hpp"
#include "UI/Spritesheet.hpp"
#include "UI/Data.hpp"

class Menu {
    public:
        Menu(sf::RenderWindow &window, network::socket &socket, uint8_t &status);
        ~Menu() = default;

        void initMenu();

        void loginScreen();

        void eventsInputbox();

        void checkInputboxSelected();

        void eventsMenu();

        void eventsLogin();

        void eventsUser();

        void tryLogin();

        void tryUsername();

        void displayMenu();

        void drawUser();

    private:
        sf::RenderWindow &m_window;
        network::socket &m_socket;
        uint8_t &m_status;
        Inputbox m_ip{20, sf::Color::Black, true, 15};
        Inputbox m_port{20, sf::Color::Black, false, 7};
        Inputbox m_user{20, sf::Color::Black, false, 25};
        sf::Color m_black{0, 0, 0, 200};
        sf::Color m_grey{180, 180, 180, 230};
        sf::RectangleShape m_loginrectangle{{400, 450}};
        sf::RectangleShape m_userrectangle{{400, 250}};
        sf::RectangleShape m_ipbox{{330, 25}};
        sf::RectangleShape m_portbox{{330, 25}};
        sf::RectangleShape m_userbox{{330, 25}};
        sf::RectangleShape m_errorbox{{400, 60}};
        Textbox m_texttitle{80, "R-Type", sf::Color::White, {110, 280}};
        Textbox m_textlogin{20, "Server login", sf::Color::White, {90, 410}};
        Textbox m_textip{17, "IP address", sf::Color::White, {80, 470}};
        Textbox m_textport{17, "Port", sf::Color::White, {80, 545}};
        Textbox m_textuser{30, "Enter a username",sf::Color::Black, {840, 450}};
        Textbox m_texterror{20, "Error: IP address or Port is incorrect", sf::Color::Yellow, {850, 900}};
        Spritesheet m_menu{Data::tmenu, {0, 0}, 125, {3600, 10125}, {0, 0, 720, 405}, {2.667, 2.667}, 720};
        Spritesheet m_loginbutton{Data::tloginbutton, {300, 640}, 3, {83, 75}, {0, 0, 83, 25}, {1.3, 1.3}, 83};
        Spritesheet m_userbutton{Data::tloginbutton, {910, 600}, 3, {83, 75}, {0, 0, 83, 25}, {1.3, 1.3}, 83};
        Spritesheet m_loading{Data::tloading, {0, 0}, 12, {612, 51}, {0, 0, 51, 51}, {0.4, 0.4}, 51};
        sf::Event m_event{};
        sf::Font m_font{};
        sf::Clock m_clock{};
        sf::Cursor m_cursor{};
        uint8_t m_logstatus = 0;
        bool m_isloading{};
};
```

5.26 Profile.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include "UI/Spritesheet.hpp"
#include "UI/Textbox.hpp"
#include "UI/Data.hpp"

class Profile : public sf::Drawable {
    public:
        Profile(std::string username, bool rdy, size_t nb);
        ~Profile() = default;

        void setPosition(sf::Vector2f pos);

        void draw(sf::RenderTarget &target, sf::RenderStates states) const override;

    private:
        Spritesheet m_profilepic{Data::tprofile, {0, 0}, 1, {300, 300}, {0, 0, 300, 300}, {0.2, 0.2}};
        Spritesheet m_ready{Data::tready, {0, 0}, 2, {248, 124}, {0, 0, 124, 124}, {0.5, 0.5}, 124};
        Textbox m_usernamebox{30, "", sf::Color::Black};
        sf::Color m_grey{230, 230, 230, 255};
        sf::RectangleShape m_profilerectangle{{800, 70}};
        sf::Vector2f m_pos{};
};
```

5.27 Spritesheet.hpp

```
#pragma once

#include <SFML/Graphics.hpp>

class Spritesheet : public sf::Drawable {
    public:
        Spritesheet(sf::Texture &text, sf::Vector2f pos = {0, 0}, size_t nb = 1, sf::Vector2f dim = {1920, 1080}, sf::IntRect rect = {0, 0, 1920, 1080}, sf::Vector2f scale = {1, 1}, int space = 0);
        ~Spritesheet() = default;

        void setNbSprite(size_t nb);

        void setSizeImage(sf::Vector2f size);

        void setTexture(const sf::Texture &text);

        void setPosition(sf::Vector2f pos);

        void setRectSize(sf::IntRect rect);

        void setRect(sf::IntRect rect);

        void setSpace(int space);

        void setScale(sf::Vector2f scale);

        size_t &getCurrentImage();

        size_t &getNbSprite();

        sf::Vector2f &getScale();

        sf::IntRect &getRect();

        sf::Vector2f &getSize();

         int &getSpace();

         sf::Vector2f &getPosition();

         const sf::Texture &getTexture();

         bool mouseIsInSprite(sf::Vector2i pmouse);

         void playImage(size_t nb);

         void animSprite();

         void draw(sf::RenderTarget &target, sf::RenderStates states) const override;

     private:
         sf::Sprite m_sprite{};
         sf::Vector2f m_pos{};
         sf::IntRect m_rect{};
         sf::Vector2f m_scale{};
         sf::Vector2f m_size{};
         int m_space = 0;
         size_t m_nb = 0;
         size_t m_current = 0;

};
```

5.28 SpriteSize.hpp

```
#pragma once

#include <map>
#include <string>
#include <utility>

struct vec2float
{
    float x, y;
};

namespace Data {
inline std::map<std::string, std::pair<vec2float, vec2float» sprite_data {
    {"player", {{32, 14}, {2.5, 2.5}}},
    {"player0", {{32, 14}, {2.5, 2.5}}},
    {"player1", {{32, 14}, {2.5, 2.5}}},
    {"player2", {{32, 14}, {2.5, 2.5}}},
    {"player3", {{32, 14}, {2.5, 2.5}}},
    {"enemy_plane", {{21, 24}, {2.5, 2.5}}},
    {"enemy_cyborg", {{32, 31}, {3, 3}}},
    {"enemy2", {{32, 22}, {2, 2}}},
    {"friend_shoot", {{16, 4}, {2, 2}}},
    {"enemy_shoot", {{7, 6}, {2, 2}}}
};
}
```

5.29 Status.hpp

```
#pragma once

#include <cstdint>

enum status : uint8_t {
    MENU,
    LOBBY,
    GAME,
    EXIT
};
```

5.30 Textbox.hpp

```
#pragma once
```

```

#include <SFML/Graphics.hpp>

class Textbox : public sf::Drawable {
    public:
        Textbox(size_t size = 20, sf::String str = "", sf::Color color = sf::Color::White, sf::Vector2f
      pos = {0, 0});
        ~Textbox() = default;

         void setSize(size_t size);

         void setString(sf::String str);

         void setColor(sf::Color color);

         void setPosition(sf::Vector2f pos);

         void setOutlineColor(sf::Color color, float size);

         void setFont(const sf::Font &font);

         sf::String &getString(void);

         sf::Vector2f &getPosition(void);

         void draw(sf::RenderTarget &target, sf::RenderStates states) const override;

     private:
         sf::Text m_text{};
         sf::String m_str{};
         sf::Vector2f m_pos{};
};
```

Index