The Dungeon Throne

1.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

The Dungeon Throne

Dzejrou, MFF UK

Thesis Name:

The Dungeon Throne: A 3D Dungeon Managment Game

Abstract:

The goal of this thesis is to design and implement a real-time strategy game in a 3D world with emphasis on high modifiability using a suitable scripting language. Inspired by the Dungeon Keeper series developed by Bullfrog Studios, the player's goal in this game is to protect his dungeon from endless armies of heroes raiding his domain with intentions to steal his treasures.

Once finished, the game's scripting engine should offer the ability to change data and logic of entities and systems to people with at least a basic understanding of programming. This will lead to easy future extensibility of the game and the possibility to create easily installable modifications.

Literature:

- Programming In Lua, 3rd Edition, Roberto Ierusalimschy, Lua.org 2013
- Game Engine Architecture, Jason Gregory, A K Peters/CRC Press 2014
- Programming Game AI By Example, Mat Buckland, Wordware Publishing Inc. 2005

Progress album:

`http://imgur.com/a/PDMd7`

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 action Namespace Reference

Functions representing actions that can be key bound.

Functions

- void CAST_SPELL_1 ()
  Casts the first spell in the spell tool window.
- void CAST_SPELL_2 ()
  Casts the second spell in the spell tool window.
- void CAST_SPELL_3 ()
  Casts the third spell in the spell tool window.
- void CAST_SPELL_4 ()
  Casts the fourth spell in the spell tool window.
- void NEXT ()
  Moves the spell selection to the right.
- void PREV ()
  Moves the spell selection to the left.
- void SPELL_TAB ()
  Switches the current tool to the spell tab.
- void BUILD_TAB ()
  Switches the current tool to the build tab.
- void MENU_TAB ()
  Switches the current tool to the menu tab.
- void RESET_CAMERA ()
  Resets the position and orientation of the camera.
- void QUICK_SAVE ()
  Saves the current game to the quick_save.lua file.
- void QUICK_LOAD ()
  Restores the game save in the quick_save.lua file.

5.1.1 Detailed Description

Functions representing actions that can be key bound.

5.1.2 Function Documentation

5.1.2.1 void action::BUILD_TAB ( )

Switches the current tool to the build tab.

Definition at line 493 of file OptionsWindow.cpp.

5.1.2.2 void action::CAST_SPELL_1 ( )

Casts the first spell in the spell tool window.

Definition at line 438 of file OptionsWindow.cpp.

5.1.2.3 void action::CAST_SPELL_2 ( )

Casts the second spell in the spell tool window.

Definition at line 446 of file OptionsWindow.cpp.

5.1.2.4 void action::CAST_SPELL_3 ( )

Casts the third spell in the spell tool window.

Definition at line 454 of file OptionsWindow.cpp.

5.1.2.5 void action::CAST_SPELL_4 ( )

Casts the fourth spell in the spell tool window.

Definition at line 462 of file OptionsWindow.cpp.

5.1.2.6 void action::MENU_TAB ( )

Switches the current tool to the menu tab.

Definition at line 500 of file OptionsWindow.cpp.

5.1.2.7 void action::NEXT ( )

Moves the spell selection to the right.

Definition at line 470 of file OptionsWindow.cpp.

5.1.2.8 void action::PREV ( )

Moves the spell selection to the left.

Definition at line 478 of file OptionsWindow.cpp.

5.1.2.9 void action::QUICK_LOAD ( )

Restores the game save in the quick_save.lua file.

Definition at line 521 of file OptionsWindow.cpp.

5.1.2.10 void action::QUICK_SAVE ( )

Saves the current game to the quick_save.lua file.

Definition at line 514 of file OptionsWindow.cpp.

5.1.2.11 void action::RESET_CAMERA ( )

Resets the position and orientation of the camera.

Definition at line 507 of file OptionsWindow.cpp.

5.1.2.12 void action::SPELL_TAB ( )

Switches the current tool to the spell tab.

Definition at line 486 of file OptionsWindow.cpp.

5.2 AIHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the ai component.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)
  Changes the blueprint table name of a given entity.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)
  Returns name of the blueprint table of a given entity (i.e.
- void set_state (EntitySystem &, tdt::uint, ENTITY_STATE::VAL)
  Changes the state of a given entity.
- ENTITY_STATE::VAL get_state (EntitySystem &, tdt::uint)
  Returns the state a given entity is in.

5.2.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the ai component.

5.2.2 Function Documentation

5.2.2.1 const std::string & AIHelper::get_blueprint (**EntitySystem** & *ents,* tdt::uint *id*)

Returns name of the blueprint table of a given entity (i.e.

the table containing it's init, update and finnish methods).

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 12 of file AIHelper.cpp.

5.2.2.2 ENTITY_STATE::VAL AIHelper::get_state (EntitySystem & *ents*, tdt::uint *id*)

Returns the state a given entity is in.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 30 of file AIHelper.cpp.

5.2.2.3 void AIHelper::set_blueprint (EntitySystem & *ents*, tdt::uint *id*, const std::string & *val*)

Changes the blueprint table name of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Name	of the new blueprint table.

Definition at line 5 of file AIHelper.cpp.

5.2.2.4 void AIHelper::set_state (EntitySystem & *ents*, tdt::uint *id*, ENTITY_STATE::VAL *val*)

Changes the state of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
New	state.

Definition at line 23 of file AIHelper.cpp.

5.3 CombatHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the combat component.

Functions

- void set_target (EntitySystem &, tdt::uint, tdt::uint)

  Changes the target of a given entity's attack.
- tdt::uint get_target (EntitySystem &, tdt::uint)

  Returns the target of a given entity's attack.
- void set_range (EntitySystem &, tdt::uint, tdt::real)

  Changes the attack range of a given entity.
- tdt::real get_range (EntitySystem &, tdt::uint)

  Returns the attack range of a given entity.
- void set_dmg_range (EntitySystem &, tdt::uint, tdt::uint, tdt::uint)

  Changes the damage range (min damage, max damage) that a given entity can deal when attacking.
- std::tuple< tdt::uint, tdt::uint > get_dmg_range (EntitySystem &, tdt::uint)

  Returns the damage range (in the form of a 2-member tuple) of a given entity.
- tdt::uint get_dmg (tdt::uint, tdt::uint)

  Returns a pseudo random damage value between given two numbers, used to calculate the damage of each individual attack.
- void set_cooldown (EntitySystem &, tdt::uint, tdt::real)

  Changes the cooldown (minimal time between attacks) of a given entity.
- tdt::real get_cooldown (EntitySystem &, tdt::uint)

  Returns the cooldown (minimal time between attacks) of a given entity.
- void set_atk_type (EntitySystem &, tdt::uint, ATTACK_TYPE)

  Changes the attack type of a given entity.
- ATTACK_TYPE get_atk_type (EntitySystem &, tdt::uint)

  Returns the attack type of a given entity.
- bool in_range (EntitySystem &, tdt::uint, tdt::uint)

  Returns true if a given entity is in attack range from another entity.
- void set_projectile_blueprint (EntitySystem &, tdt::uint, const std::string &)

  Sets the projectile table used when a given entity shoots.
- const std::string & get_projectile_blueprint (EntitySystem &, tdt::uint)

  Returns the projectile table used when a given entity shoots.

5.3.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the combat component.

5.3.2 Function Documentation

5.3.2.1 ATTACK_TYPE CombatHelper::get_atk_type (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the attack type of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 87 of file CombatHelper.cpp.

5.3.2.2 tdt::real CombatHelper::get_cooldown (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the cooldown (minimal time between attacks) of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 71 of file CombatHelper.cpp.

5.3.2.3 tdt::uint CombatHelper::get_dmg (tdt::uint *min,* tdt::uint *max*)

Returns a pseudo random damage value between given two numbers, used to calculate the damage of each individual attack.

Parameters

Minimal	damage value.
Maximal	damage value.

Definition at line 59 of file CombatHelper.cpp.

5.3.2.4 std::tuple< tdt::uint, tdt::uint > CombatHelper::get_dmg_range (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the damage range (in the form of a 2-member tuple) of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 50 of file CombatHelper.cpp.

5.3.2.5 const std::string & CombatHelper::get_projectile_blueprint (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the projectile table used when a given entity shoots.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 115 of file CombatHelper.cpp.

5.3.2.6 tdt::real CombatHelper::get_range (EntitySystem & *ents*, tdt::uint *id*)

Returns the attack range of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 31 of file CombatHelper.cpp.

5.3.2.7 tdt::uint CombatHelper::get_target (EntitySystem & *ents*, tdt::uint *id*)

Returns the target of a given entity's attack.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 15 of file CombatHelper.cpp.

5.3.2.8 bool CombatHelper::in_range (EntitySystem & *ents*, tdt::uint *id1*, tdt::uint *id2*)

Returns true if a given entity is in attack range from another entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity checking range.
ID	of the second entity.

Definition at line 96 of file CombatHelper.cpp.

5.3.2.9 void CombatHelper::set_atk_type (EntitySystem & *ents*, tdt::uint *id*, ATTACK_TYPE *type*)

Changes the attack type of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
The	new attack type.

Definition at line 80 of file CombatHelper.cpp.

5.3.2.10 void CombatHelper::set_cooldown (**EntitySystem** & *ents,* tdt::uint *id,* tdt::real *cd*)

Changes the cooldown (minimal time between attacks) of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
The	new cooldown value.

Definition at line 64 of file CombatHelper.cpp.

5.3.2.11 void CombatHelper::set_dmg_range (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *min,* tdt::uint *max*)

Changes the damage range (min damage, max damage) that a given entity can deal when attacking.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
Minimal	damage value.
Maximal	damage value.

Definition at line 40 of file CombatHelper.cpp.

5.3.2.12 void CombatHelper::set_projectile_blueprint (**EntitySystem** & *ents,* tdt::uint *id,* const std::string & *val*)

Sets the projectile table used when a given entity shoots.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
Name	of the projectile table.

Definition at line 108 of file CombatHelper.cpp.

5.3.2.13 void CombatHelper::set_range (**EntitySystem** & *ents,* tdt::uint *id,* tdt::real *range*)

Changes the attack range of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
The	new attack range.

Definition at line 24 of file CombatHelper.cpp.

5.3.2.14 void CombatHelper::set_target (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Changes the target of a given entity's attack.

Parameters

EntitySystem	containing the entity and it's target.
ID	of the entity.
ID	of the target.

Definition at line 8 of file CombatHelper.cpp.

5.4 CommandHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the command component.

Functions

- void set_command (EntitySystem &, tdt::uint, COMMAND_TYPE, bool=true)
  Sets the bit value of a given command for a given entity.
- bool test_command (EntitySystem &, tdt::uint, COMMAND_TYPE)
  Returns true if a given entity responds to a given type of command, false otherwise.
- void command_to_mine (EntitySystem &, SelectionBox &)
  Commands the miner with the smallest task queue (if any) to mine all selected mineable entities.
- void command_to_attack (EntitySystem &, SelectionBox &)
  Commands the combat unit with the smallest task queue (if any) to attack a selected enemy.
- void command_to_reposition (EntitySystem &, Ogre::Real, Ogre::Real)
  Commands the unit with the smallest task queue (if any) to move to a given position.
- void command_to_return_gold (EntitySystem &, CombatSystem &)
  Commands all miners that have gold on them to return it to the nearest gold vault.
- void command_to_fall_back (EntitySystem &)
  Commands all free units created at a spawner to return back.

5.4.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the command component.

5.4.2 Function Documentation

5.4.2.1 void CommandHelper::command_to_attack (EntitySystem & *ents*, SelectionBox & *selection*)

Commands the combat unit with the smallest task queue (if any) to attack a selected enemy.

Parameters

Reference	to the entity system containing the entity.
Selection	box used to select the enemy.

Definition at line 70 of file CommandHelper.cpp.

5.4.2.2 void CommandHelper::command_to_fall_back (EntitySystem & *ents*)

Commands all free units created at a spawner to return back.

Parameters

Reference	to the entity system containing the entity.

Definition at line 170 of file CommandHelper.cpp.

5.4.2.3 void CommandHelper::command_to_mine (EntitySystem & *ents*, SelectionBox & *selection*)

Commands the miner with the smallest task queue (if any) to mine all selected mineable entities.

Parameters

Reference	to the entity system containing the entity.
Selection	box used to select the mineable entities.

Definition at line 25 of file CommandHelper.cpp.

5.4.2.4 void CommandHelper::command_to_reposition (EntitySystem & *ents*, Ogre::Real *x*, Ogre::Real *y*)

Commands the unit with the smallest task queue (if any) to move to a given position.

Parameters

Reference	to the entity system containing the entity.
X	coordinate of the position.
Z	coordinate of the position.

Definition at line 109 of file CommandHelper.cpp.

5.4.2.5 void CommandHelper::command_to_return_gold (EntitySystem & *ents*, CombatSystem & *combat*)

Commands all miners that have gold on them to return it to the nearest gold vault.

Parameters

Reference	to the entity system containing the entity.
Combat	system used to search for deposits.

Definition at line 146 of file CommandHelper.cpp.

5.4.2.6 void CommandHelper::set_command (EntitySystem & *ents,* tdt::uint *id,* COMMAND_TYPE *command,* bool *val =* `true`)

Sets the bit value of a given command for a given entity.

Parameters

Reference	to the entity system containing the entity.
ID	of the entity.
Type	of the command.
The	new bit value, true = entity responds to the command type, false = entity ignores the command type.

Definition at line 9 of file CommandHelper.cpp.

5.4.2.7 bool CommandHelper::test_command (EntitySystem & *ents,* tdt::uint *id,* COMMAND_TYPE *command*)

Returns true if a given entity responds to a given type of command, false otherwise.

Parameters

Reference	to the entity system containing the entity.
ID	of the entity.
Type	of the command.

Definition at line 16 of file CommandHelper.cpp.

5.5 ConstructorHelper Namespace Reference

Namespace containing auxiliary functions that help with constructor component management.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)

  Sets the name of the blueprint table that handles the construction of a given entity.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)

  Returns the name of the blueprint table that handles the construction of a given entity.
- void call (EntitySystem &, tdt::uint)

  Calls the blueprint table that handles the construction of a given entity.

5.5.1 Detailed Description

Namespace containing auxiliary functions that help with constructor component management.

5.5.2 Function Documentation

5.5.2.1 void ConstructorHelper::call (EntitySystem & *ents,* tdt::uint *id*)

Calls the blueprint table that handles the construction of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 22 of file ConstructorHelper.cpp.

5.5.2.2 const std::string & ConstructorHelper::get_blueprint (EntitySystem & *ents,* tdt::uint *id*)

Returns the name of the blueprint table that handles the construction of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 13 of file ConstructorHelper.cpp.

5.5.2.3 void ConstructorHelper::set_blueprint (EntitySystem & *ents,* tdt::uint *id,* const std::string & *val*)

Sets the name of the blueprint table that handles the construction of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new blueprint name.

Definition at line 6 of file ConstructorHelper.cpp.

5.6 CounterHelper Namespace Reference

Namespace containing auxiliary functions that help with counter component management.

Functions

- bool increment (EntitySystem &, tdt::uint)
  Increments the counter of a given entity and returns true if the counter has reached the max value, false otherwise.
- bool decrement (EntitySystem &, tdt::uint)
  Decrements the counter of a given entity and returns true if the counter has reached the max value, false otherwise.
- void set_curr_value (EntitySystem &, tdt::uint, tdt::uint)
  Sets the current value of the counter of a given entity.
- tdt::uint get_curr_value (EntitySystem &, tdt::uint)
  Returns the current value of the counter of a given entity.
- void set_max_value (EntitySystem &, tdt::uint, tdt::uint)
  Sets the max value of the counter of a given entity.
- tdt::uint get_max_value (EntitySystem &, tdt::uint)
  Returns the max value of the counter of a given entity.

5.6.1 Detailed Description

Namespace containing auxiliary functions that help with counter component management.

5.6.2 Function Documentation

5.6.2.1 bool CounterHelper::decrement (EntitySystem & *ents*, tdt::uint *id*)

Decrements the counter of a given entity and returns true if the counter has reached the max value, false otherwise.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 17 of file CounterHelper.cpp.

5.6.2.2 tdt::uint CounterHelper::get_curr_value (EntitySystem & *ents*, tdt::uint *id*)

Returns the current value of the counter of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 36 of file CounterHelper.cpp.

5.6.2.3 tdt::uint CounterHelper::get_max_value (EntitySystem & *ents*, tdt::uint *id*)

Returns the max value of the counter of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 52 of file CounterHelper.cpp.

5.6.2.4 bool CounterHelper::increment (**EntitySystem** & *ents,* tdt::uint *id*)

Increments the counter of a given entity and returns true if the counter has reached the max value, false otherwise.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 5 of file CounterHelper.cpp.

5.6.2.5 void CounterHelper::set_curr_value (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the current value of the counter of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new counter value.

Definition at line 29 of file CounterHelper.cpp.

5.6.2.6 void CounterHelper::set_max_value (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the max value of the counter of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 45 of file CounterHelper.cpp.

5.7 DestructorHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the destructor component.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)

  Sets the name of the table that contains the "dtor" function which get's called when a given entity is destroyed.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)

  Returns the name of the table that contains the "dtor" function which get's called when a given entity is destroyed.
- void destroy (EntitySystem &, tdt::uint, bool=false, tdt::uint=Component::NO_ENTITY)

  Destroys a given entity and if possible calls it's destructor.

5.7.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the destructor component.

5.7.2 Function Documentation

5.7.2.1 void DestructorHelper::destroy (EntitySystem & *ents*, tdt::uint *id*, bool *supress_dtor* = `false`, tdt::uint *killer* = `Component::NO_ENTITY`)

Destroys a given entity and if possible calls it's destructor.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
If	true, the destructor won't be called.
ID	of the killer (if any).

Definition at line 25 of file DestructorHelper.cpp.

5.7.2.2 const std::string & DestructorHelper::get_blueprint (EntitySystem & *ents*, tdt::uint *id*)

Returns the name of the table that contains the "dtor" function which get's called when a given entity is destroyed.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 14 of file DestructorHelper.cpp.

5.7.2.3 void DestructorHelper::set_blueprint (EntitySystem & *ents*, tdt::uint *id*, const std::string & *val*)

Sets the name of the table that contains the "dtor" function which get's called when a given entity is destroyed.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
The	new blueprint table's name.

Definition at line 7 of file DestructorHelper.cpp.

5.8 EventHandlerHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the event handler component.

Functions

- void set_handler (EntitySystem &, tdt::uint, const std::string &)

  Sets the name of the table that contains event the handler function of a given entity.
- const std::string & get_handler (EntitySystem &, tdt::uint)

  Returns the name of the table that contains event the handler function of a given entity.
- bool can_handle (EntitySystem &, tdt::uint, EVENT_TYPE)

  Returns true if a given entity can handle a given event.
- void add_possible_event (EntitySystem &, tdt::uint, EVENT_TYPE)

  Adds a given event into the list of possible events of a given entity.
- void delete_possible_event (EntitySystem &, tdt::uint, EVENT_TYPE)

  Removes a given event from the list of possible events of a given entity.

5.8.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the event handler component.

5.8.2 Function Documentation

5.8.2.1 void EventHandlerHelper::add_possible_event (**EntitySystem** & *ents,* tdt::uint *id,* EVENT_TYPE *val*)

Adds a given event into the list of possible events of a given entity.

Parameters

Entity	system that contains the entity.
ID	of the entity.
The	type of the event.

Definition at line 32 of file EventHandlerHelper.cpp.

5.8.2.2 bool EventHandlerHelper::can_handle (EntitySystem & *ents,* tdt::uint *id,* EVENT_TYPE *val*)

Returns true if a given entity can handle a given event.

Parameters

Entity	system that contains the entity.
ID	of the entity.
The	type of the event.

Definition at line 23 of file EventHandlerHelper.cpp.

5.8.2.3 void EventHandlerHelper::delete_possible_event (EntitySystem & *ents,* tdt::uint *id,* EVENT_TYPE *val*)

Removes a given event from the list of possible events of a given entity.

Parameters

Entity	system that contains the entity.
ID	of the entity.
The	type of the event.

Definition at line 39 of file EventHandlerHelper.cpp.

5.8.2.4 const std::string & EventHandlerHelper::get_handler (EntitySystem & *ents,* tdt::uint *id*)

Returns the name of the table that contains event the handler function of a given entity.

Parameters

Entity	system that contains the entity.
ID	of the entity.

Definition at line 12 of file EventHandlerHelper.cpp.

5.8.2.5 void EventHandlerHelper::set_handler (EntitySystem & *ents,* tdt::uint *id,* const std::string & *val*)

Sets the name of the table that contains event the handler function of a given entity.

Parameters

Entity	system that contains the entity.
ID	of the entity.
The	new handler table name.

Definition at line 5 of file EventHandlerHelper.cpp.

5.9 EventHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the event component.

Functions

- void set_event_type (EntitySystem &, tdt::uint, EVENT_TYPE)

  Sets the type of a given event.
- EVENT_TYPE get_event_type (EntitySystem &, tdt::uint)

  Returns the type of a given event.
- void set_target (EntitySystem &, tdt::uint, tdt::uint)

  Sets the target entity (the subject of the event) of a given event.
- tdt::uint get_target (EntitySystem &, tdt::uint)

  Returns the target entity (the subject of the event) of a given event.
- void set_radius (EntitySystem &, tdt::uint, tdt::real)

  Sets the radius handlers have to be in in order to be able to handle a given event.
- tdt::real get_radius (EntitySystem &, tdt::uint)

  Returns the radius handlers have to be in in order to be able to handle a given event.
- void set_active (EntitySystem &, tdt::uint, bool=true)

  Sets the activity state of a given event.
- bool is_active (EntitySystem &, tdt::uint)

  Returns true if a given event is active, false otherwise.
- void set_event_handler (EntitySystem &, tdt::uint, tdt::uint)

  Sets the entity that handles a given event.
- tdt::uint get_event_handler (EntitySystem &, tdt::uint)

  Returns the entity that handles a given event.

5.9.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the event component.

5.9.2 Function Documentation

5.9.2.1 std::size_t EventHelper::get_event_handler (EntitySystem & *ents*, tdt::uint *id*)

Returns the entity that handles a given event.

Parameters

Entity	system containing both the entity and the event.
ID	of the event.

Definition at line 76 of file EventHelper.cpp.

5.9.2.2 EVENT_TYPE EventHelper::get_event_type (EntitySystem & *ents*, tdt::uint *id*)

Returns the type of a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.

Definition at line 12 of file EventHelper.cpp.

5.9.2.3 Ogre::Real EventHelper::get_radius (EntitySystem & *ents*, tdt::uint *id*)

Returns the radius handlers have to be in in order to be able to handle a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.

Definition at line 44 of file EventHelper.cpp.

5.9.2.4 std::size_t EventHelper::get_target (EntitySystem & *ents*, tdt::uint *id*)

Returns the target entity (the subject of the event) of a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.

Definition at line 28 of file EventHelper.cpp.

5.9.2.5 bool EventHelper::is_active (EntitySystem & *ents*, tdt::uint *id*)

Returns true if a given event is active, false otherwise.

Parameters

Entity	system that contains the entity.
ID	of the event.

Definition at line 60 of file EventHelper.cpp.

5.9.2.6 void EventHelper::set_active (EntitySystem & *ents*, tdt::uint *id*, bool *val* = `true`)

Sets the activity state of a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.
ID	for active, false for inactive.

Definition at line 53 of file EventHelper.cpp.

5.9.2.7 void EventHelper::set_event_handler (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the entity that handles a given event.

Parameters

Entity	system containing both the entity and the event.
ID	of the event.
ID	of the handling entity.

Definition at line 69 of file EventHelper.cpp.

5.9.2.8 void EventHelper::set_event_type (EntitySystem & *ents*, tdt::uint *id*, EVENT_TYPE *val*)

Sets the type of a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.
The	new type.

Definition at line 5 of file EventHelper.cpp.

5.9.2.9 void EventHelper::set_radius (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the radius handlers have to be in in order to be able to handle a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.
The	new radius.

Definition at line 37 of file EventHelper.cpp.

5.9.2.10 void EventHelper::set_target (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the target entity (the subject of the event) of a given event.

Parameters

Entity	system that contains the entity.
ID	of the event.
ID	of the target.

Definition at line 21 of file EventHelper.cpp.

5.10 ExperienceValueHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the experience value component.

Functions

- void set (EntitySystem &, tdt::uint, tdt::uint)

  Sets the experience value a given entity is worth.
- tdt::uint get (EntitySystem &, tdt::uint)

  Returns the experience value a given entity is worth.
- void increase (EntitySystem &, tdt::uint, tdt::uint)

  Increases the experience value a given entity is worth by a given value.
- void decrease (EntitySystem &, tdt::uint, tdt::uint)

  Decreases the experience value a given entity is worth by a given value.

5.10.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the experience value component.

5.10.2 Function Documentation

5.10.2.1 void ExperienceValueHelper::decrease (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Decreases the experience value a given entity is worth by a given value.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	value to decrease by.

Definition at line 28 of file ExperienceValueHelper.cpp.

5.10.2.2 tdt::uint ExperienceValueHelper::get (EntitySystem & *ents*, tdt::uint *id*)

Returns the experience value a given entity is worth.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 12 of file ExperienceValueHelper.cpp.

5.10.2.3 void ExperienceValueHelper::increase (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Increases the experience value a given entity is worth by a given value.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	value to increase by.

Definition at line 21 of file ExperienceValueHelper.cpp.

5.10.2.4 void ExperienceValueHelper::set (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the experience value a given entity is worth.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new experience value.

Definition at line 5 of file ExperienceValueHelper.cpp.

5.11 ExplosionHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the explosion component.

Functions

- void set_delta (EntitySystem &, tdt::uint, tdt::real)

Sets the value by which a given entity's explosion radius is updated on each of the graphics system updates.
- tdt::real get_delta (EntitySystem &, tdt::uint)

  Returns the value by which a given entity's explosion radius is updated on each of the graphics system updates.
- void set_max_radius (EntitySystem &, tdt::uint, tdt::real)

  Sets the radius after which a given entity with an explosion component gets deleted.
- tdt::real get_max_radius (EntitySystem &, tdt::uint)

  Returns the radius after which a given entity with an explosion component gets deleted.
- tdt::real get_curr_radius (EntitySystem &, tdt::uint)

  Returns the current radius of a given entity's explosion component.
- void increase_curr_radius (EntitySystem &, tdt::uint, tdt::real)

  Increases the radius of a given entity's explosion component by a given value.

5.11.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the explosion component.

5.11.2 Function Documentation

5.11.2.1 tdt::real ExplosionHelper::get_curr_radius (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the current radius of a given entity's explosion component.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 37 of file ExplosionHelper.cpp.

5.11.2.2 tdt::real ExplosionHelper::get_delta (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the value by which a given entity's explosion radius is updated on each of the graphics system updates.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 12 of file ExplosionHelper.cpp.

5.11.2.3 tdt::real ExplosionHelper::get_max_radius (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the radius after which a given entity with an explosion component gets deleted.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 28 of file ExplosionHelper.cpp.

5.11.2.4 void ExplosionHelper::increase_curr_radius (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Increases the radius of a given entity's explosion component by a given value.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	value to increase by.

Definition at line 46 of file ExplosionHelper.cpp.

5.11.2.5 void ExplosionHelper::set_delta (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the value by which a given entity's explosion radius is updated on each of the graphics system updates.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new delta value.

Definition at line 5 of file ExplosionHelper.cpp.

5.11.2.6 void ExplosionHelper::set_max_radius (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the radius after which a given entity with an explosion component gets deleted.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new max radius value.

Definition at line 21 of file ExplosionHelper.cpp.

5.12 FactionHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the faction component.

Functions

- void set_faction (EntitySystem &, tdt::uint, FACTION)
  Changes the FACTION of a given entity.
- FACTION get_faction (EntitySystem &, tdt::uint)
  Returns the FACTION of a given entity.
- const std::string & get_faction_name (EntitySystem &, tdt::uint)
  Returns the FACTION name (a string) of a given entity.

5.12.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the faction component.

5.12.2 Function Documentation

5.12.2.1 FACTION FactionHelper::get_faction (EntitySystem & *ents*, tdt::uint *id*)

Returns the FACTION of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 12 of file FactionHelper.cpp.

5.12.2.2 const std::string & FactionHelper::get_faction_name (EntitySystem & *ents*, tdt::uint *id*)

Returns the FACTION name (a string) of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 21 of file FactionHelper.cpp.

5.12.2.3 void FactionHelper::set_faction (EntitySystem & *ents*, tdt::uint *id*, FACTION *val*)

Changes the FACTION of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
The	new faction.

Definition at line 5 of file FactionHelper.cpp.

5.13 GoldHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the gold component.

Functions

- void set_curr_gold (EntitySystem &, tdt::uint, tdt::uint)

  Sets the gold value a given entity has, does not check if the new value is smalled than the limit, use add_gold if that's needed.
- tdt::uint get_curr_gold (EntitySystem &, tdt::uint)

  Returns the gold value a given entity has.
- void set_max_gold (EntitySystem &, tdt::uint, tdt::uint)

  Sets the limit of gold that a given entity can have.
- tdt::uint get_max_gold (EntitySystem &, tdt::uint)

  Returns the limit of gold that a given entity can have.
- tdt::uint add_gold (EntitySystem &, tdt::uint, tdt::uint)

  Adds a given gold value to a given entity up to it's gold limit, returns the amount of gold that superceeded the gold limit and thus wasn't added.
- tdt::uint sub_gold (EntitySystem &, tdt::uint, tdt::uint, bool=true)

  Removes a given amount of gold from a given entity, but does not subtract past zero.
- tdt::uint transfer_all_gold (EntitySystem &, tdt::uint, tdt::uint)

  Transfers all gold of an entity to another entity while keeping the (0, max) range in mind.
- bool gold_full (EntitySystem &, tdt::uint)

  Returns true if a given entity's gold storage is full, false otherwise.
- void **register_transaction_** (EntitySystem &, GoldComponent &, tdt::uint, tdt::uint, bool=true)

5.13.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the gold component.

5.13.2 Function Documentation

5.13.2.1 tdt::uint GoldHelper::add_gold (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *val*)

Adds a given gold value to a given entity up to it's gold limit, returns the amount of gold that superceeded the gold limit and thus wasn't added.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Amount	of gold to add.

Definition at line 50 of file GoldHelper.cpp.

5.13.2.2 tdt::uint GoldHelper::get_curr_gold (EntitySystem & *ents*, tdt::uint *id*)

Returns the gold value a given entity has.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 22 of file GoldHelper.cpp.

5.13.2.3 tdt::uint GoldHelper::get_max_gold (EntitySystem & *ents*, tdt::uint *id*)

Returns the limit of gold that a given entity can have.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 41 of file GoldHelper.cpp.

5.13.2.4 bool GoldHelper::gold_full (EntitySystem & *ents*, tdt::uint *id*)

Returns true if a given entity's gold storage is full, false otherwise.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 116 of file GoldHelper.cpp.

5.13.2.5 void GoldHelper::set_curr_gold (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the gold value a given entity has, does not check if the new value is smalled than the limit, use add_gold if that's needed.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new gold value.

Definition at line 8 of file GoldHelper.cpp.

5.13.2.6 void GoldHelper::set_max_gold (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the limit of gold that a given entity can have.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new limit.

Definition at line 31 of file GoldHelper.cpp.

5.13.2.7 tdt::uint GoldHelper::sub_gold (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*, bool *reg* = `true`)

Removes a given amount of gold from a given entity, but does not subtract past zero.

Returns amount of gold that could not be removed (remainder from the given amount after subtracting).

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Amount	of gold to subtract.
If	false, the transaction won't be registered (used when called from Player);

Definition at line 70 of file GoldHelper.cpp.

5.13.2.8 tdt::uint GoldHelper::transfer_all_gold (EntitySystem & *ents*, tdt::uint *id_from*, tdt::uint *id_to*)

Transfers all gold of an entity to another entity while keeping the (0, max) range in mind.

Returns amount of gold actually transfered.

Parameters

EntitySystem	that contains the entity.
ID	of the sender entity.
ID	of the receiver entity.

Definition at line 88 of file GoldHelper.cpp.

5.14 GraphicsHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the graphics component.

Enumerations

- enum PLANE { **X** = 0, **Y**, **Z** }

  Enum representing the planes in a 3D space.

Functions

- void set_mesh (EntitySystem &, tdt::uint, const std::string &)

  Sets the model of a given entity.
- const std::string & get_mesh (EntitySystem &, tdt::uint)

  Returns the name of the mesh of a given entity.
- void set_material (EntitySystem &, tdt::uint, const std::string &)

  Sets the name of the material a given entity is using.
- const std::string & get_material (EntitySystem &, tdt::uint)

  Returns the name of the material a given entity is using.
- void set_visible (EntitySystem &, tdt::uint, bool)

  Sets the visibility status of a given entity.
- bool is_visible (EntitySystem &, tdt::uint)

  Returns true if a given entity is visible, false otherwise.
- void set_manual_scaling (EntitySystem &, tdt::uint, bool)

  Sets the manual scaling status of a given entity.
- bool get_manual_scaling (EntitySystem &, tdt::uint)

  Returns true if a given entity's model has explicit dimensions or false if the dimensions of it's mesh are used.
- void set_scale (EntitySystem &, tdt::uint, const Ogre::Vector3 &)

  Changes the dimensions of a given entity (requires manual_scaling to be true).
- const Ogre::Vector3 & get_scale (EntitySystem &, tdt::uint)

  Returns the scale (the dimensions) of a given entity (requires manual_scaling to be true).
- void look_at (EntitySystem &, tdt::uint, tdt::uint)

  Rotates a given entity so that it faces another one.
- void rotate (EntitySystem &, tdt::uint, tdt::real, PLANE=PLANE::Y)

  Rotates a given entity by a given amount of radians.
- const Ogre::AxisAlignedBox & get_bounds (EntitySystem &, tdt::uint)

  Returns a given entity's bounding box.
- bool collide (EntitySystem &, tdt::uint, tdt::uint)

  Returns true if two given entities collide, false otherwise.
- void init_graphics_component (EntitySystem &, Ogre::SceneManager &, tdt::uint)

  Initializes the graphics component of a manually created entity by loading it's model into an Ogre::Entity and bounding it to a scene node.
- void set_query_flags (EntitySystem &, tdt::uint, tdt::uint)

  Sets the flags used for queries (like in CombatSystem::in_sight) of a given entity.
- tdt::uint get_query_flags (EntitySystem &, tdt::uint)

  Return the flags used for queries (like in CombatSystem::in_sight) of a given entity.
- void apply_scale (EntitySystem &, tdt::uint)

  Applies scale to a given entity's Ogre::Node if the entity has manual scaling enabled (the scale is a part of the component).

5.14.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the graphics component.

5.14.2 Enumeration Type Documentation

5.14.2.1 enum GraphicsHelper::PLANE [strong]

Enum representing the planes in a 3D space.

Definition at line 16 of file GraphicsHelper.hpp.

5.14.3 Function Documentation

5.14.3.1 void GraphicsHelper::apply_scale (EntitySystem & *ents*, tdt::uint *id*)

Applies scale to a given entity's Ogre::Node if the entity has manual scaling enabled (the scale is a part of the component).

Parameters

ID	of the entity.

Definition at line 211 of file GraphicsHelper.cpp.

5.14.3.2 bool GraphicsHelper::collide (EntitySystem & *ents*, tdt::uint *id1*, tdt::uint *id2*)

Returns true if two given entities collide, false otherwise.

Parameters

Reference	to the entity system that contains components.
ID	of the first entity.
ID	of the second entity.

Definition at line 145 of file GraphicsHelper.cpp.

5.14.3.3 const Ogre::AxisAlignedBox & GraphicsHelper::get_bounds (EntitySystem & *ents*, tdt::uint *id*)

Returns a given entity's bounding box.

Parameters

Reference	to the entity system that contains components.
ID	of th entity.

Note

The entity has to have a GraphicsComponent, because collision detection is done using Ogre's bounding boxes.

Definition at line 136 of file GraphicsHelper.cpp.

5.14.3.4 bool GraphicsHelper::get_manual_scaling (EntitySystem & *ents*, tdt::uint *id*)

Returns true if a given entity's model has explicit dimensions or false if the dimensions of it's mesh are used.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.

Definition at line 72 of file GraphicsHelper.cpp.

5.14.3.5 const std::string & GraphicsHelper::get_material (EntitySystem & *ents*, tdt::uint *id*)

Returns the name of the material a given entity is using.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.

Definition at line 34 of file GraphicsHelper.cpp.

5.14.3.6 const std::string & GraphicsHelper::get_mesh (EntitySystem & *ents*, tdt::uint *id*)

Returns the name of the mesh of a given entity.

Parameters

Reference	to the entity system that contains components.

Definition at line 12 of file GraphicsHelper.cpp.

5.14.3.7 tdt::uint GraphicsHelper::get_query_flags (EntitySystem & *ents*, tdt::uint *id*)

Return the flags used for queries (like in CombatSystem::in_sight) of a given entity.

Parameters

ID	of the entity.

Definition at line 202 of file GraphicsHelper.cpp.

5.14.3.8 const Ogre::Vector3 & GraphicsHelper::get_scale (EntitySystem & *ents*, tdt::uint *id*)

Returns the scale (the dimensions) of a given entity (requires manual_scaling to be true).

Parameters

Reference	to the entity system that contains components.
ID	of the entity.

Definition at line 88 of file GraphicsHelper.cpp.

5.14.3.9 void GraphicsHelper::init_graphics_component (EntitySystem & *ents*, Ogre::SceneManager & *scene*, tdt::uint *id*)

Initializes the graphics component of a manually created entity by loading it's model into an Ogre::Entity and bounding it to a scene node.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.

Definition at line 150 of file GraphicsHelper.cpp.

5.14.3.10 bool GraphicsHelper::is_visible (EntitySystem & *ents*, tdt::uint *id*)

Returns true if a given entity is visible, false otherwise.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.

Definition at line 56 of file GraphicsHelper.cpp.

5.14.3.11 void GraphicsHelper::look_at (EntitySystem & *ents*, tdt::uint *id1*, tdt::uint *id2*)

Rotates a given entity so that it faces another one.

Parameters

Reference	to the entity system that contains components.
ID	of the first entity.
ID	of the second entity.

Definition at line 99 of file GraphicsHelper.cpp.

5.14.3.12 void GraphicsHelper::rotate (EntitySystem & *ents*, tdt::uint *id*, tdt::real *delta*, PLANE *plane* = `PLANE::Y`)

Rotates a given entity by a given amount of radians.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.
Rotation	angle in radians.

Note

Ogre3D has conversion functions.

Definition at line 114 of file GraphicsHelper.cpp.

5.14.3.13 void GraphicsHelper::set_manual_scaling (EntitySystem & *ents*, tdt::uint *id*, bool *val*)

Sets the manual scaling status of a given entity.

(i.e. if the model should use dimensions stored in the mesh file or has them explicitly set.)

Parameters

Reference	to the entity system that contains components.
ID	of the entity.
True	for manual scaling, false for using the scale of the mesh.

Note

Requires a call to init_graphics_component to take effect.

Definition at line 65 of file GraphicsHelper.cpp.

5.14.3.14 void GraphicsHelper::set_material (EntitySystem & *ents*, tdt::uint *id*, const std::string & *material*)

Sets the name of the material a given entity is using.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.
Name	of the entity.

Note

Requires a call to init_graphcis_component to take effect.

Definition at line 23 of file GraphicsHelper.cpp.

5.14.3.15 void GraphicsHelper::set_mesh (EntitySystem & *ents*, tdt::uint *id*, const std::string & *mesh*)

Sets the model of a given entity.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.
Name	of the new mesh.

Note

Requires a call to init_graphics_component to take effect.

Definition at line 5 of file GraphicsHelper.cpp.

5.14.3.16 void GraphicsHelper::set_query_flags (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the flags used for queries (like in CombatSystem::in_sight) of a given entity.

Parameters

ID	of the entity.
The	new query flags.

Definition at line 195 of file GraphicsHelper.cpp.

5.14.3.17 void GraphicsHelper::set_scale (EntitySystem & *ents*, tdt::uint *id*, const Ogre::Vector3 & *val*)

Changes the dimensions of a given entity (requires manual_scaling to be true).

Parameters

Reference	to the entity system that contains components.
ID	of the entity.
The	new scale value.

Note

Requires a call to init_graphics_component to take effect.

Definition at line 81 of file GraphicsHelper.cpp.

5.14.3.18 void GraphicsHelper::set_visible (EntitySystem & *ents*, tdt::uint *id*, bool *val*)

Sets the visibility status of a given entity.

Parameters

Reference	to the entity system that contains components.
ID	of the entity.
True	for visible, false for invisible.

Definition at line 45 of file GraphicsHelper.cpp.

5.15 GridNodeHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the grid node component.

Functions

- const std::array< tdt::uint, GridNodeComponent::neighbour_count > & get_neighbours (EntitySystem &, tdt::uint)

    Returns an array containing IDs of all neighbours of a given node.
- bool is_free (EntitySystem &, tdt::uint)

    Returns true if the given node is free (duh...), false otherwise.
- bool area_free (EntitySystem &, tdt::uint, tdt::uint=1)

    Returns true if a given area (specified by a center node and a radius) is free.
- void set_free (EntitySystem &, tdt::uint, bool)

    Sets the free status of a given node.
- void set_free_selected (EntitySystem &, SelectionBox &, bool)

    Applies the GridSystem::set_free method to all currently selected nodes.
- std::tuple< tdt::uint, tdt::uint > get_board_coords (EntitySystem &, tdt::uint)

    Returns the board relative coordinates (row & column) of a given node.
- void set_resident (EntitySystem &, tdt::uint, tdt::uint)

    Sets the resident of a given node.
- tdt::uint get_resident (EntitySystem &, tdt::uint)

    Returns the resident of a given node.
- tdt::uint get_manhattan_distance (EntitySystem &, tdt::uint, tdt::uint)

    Returns the manhattan (definition available on Wikipedia...) distance between two nodes.
- tdt::uint get_node_in_dir (EntitySystem &, tdt::uint, int)

    Returns the closest node in a given direction.
- void set_portal_neighbour (EntitySystem &, tdt::uint, tdt::uint)

    Sets the portal node linked to this node.

5.15.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the grid node component.

5.15.2 Function Documentation

5.15.2.1 bool GridNodeHelper::area_free (EntitySystem & *ents*, tdt::uint *center*, tdt::uint *radius* = 1)

Returns true if a given area (specified by a center node and a radius) is free.

Parameters

EntitySystem	containing the nodes in the area.
ID	of the center node.
Radius	of the area.

Note

This counts also walkthrough buildings, as it's only used for building placing and not pathfinding.

Definition at line 27 of file GridNodeHelper.cpp.

5.15.2.2 std::tuple< tdt::uint, tdt::uint > GridNodeHelper::get_board_coords (EntitySystem & *ents,* tdt::uint *id*)

Returns the board relative coordinates (row & column) of a given node.

Parameters

EntitySystem	containing the node.
ID	of the node.

Definition at line 68 of file GridNodeHelper.cpp.

5.15.2.3 tdt::uint GridNodeHelper::get_manhattan_distance (EntitySystem & *ents,* tdt::uint *id1,* tdt::uint *id2*)

Returns the manhattan (definition available on Wikipedia...) distance between two nodes.

Parameters

EntitySystem	containing the nodes.
ID	of the source node.
ID	of the target node.

Definition at line 99 of file GridNodeHelper.cpp.

5.15.2.4 const std::array< tdt::uint, GridNodeComponent::neighbour_count > & GridNodeHelper::get_neighbours (EntitySystem & *ents,* tdt::uint *id*)

Returns an array containing IDs of all neighbours of a given node.

Parameters

EntitySystem	containing the node.
ID	of the node.

Definition at line 8 of file GridNodeHelper.cpp.

5.15.2.5 tdt::uint GridNodeHelper::get_node_in_dir (EntitySystem & *ents,* tdt::uint *id,* int *dir*)

Returns the closest node in a given direction.

Parameters

EntitySystem	containing the node.
ID	of the entity that is looking for the node.
The	direction represented by the DIRECTION::VAL enum.

Definition at line 110 of file GridNodeHelper.cpp.

5.15.2.6 tdt::uint GridNodeHelper::get_resident (EntitySystem & *ents,* tdt::uint *id*)

Returns the resident of a given node.

Parameters

ID	of the node.

Definition at line 90 of file GridNodeHelper.cpp.

5.15.2.7 bool GridNodeHelper::is_free (EntitySystem & *ents,* tdt::uint *id*)

Returns true if the given node is free (duh...), false otherwise.

Parameters

EntitySystem	containing the node.
ID	of the node.

Definition at line 18 of file GridNodeHelper.cpp.

5.15.2.8 void GridNodeHelper::set_free (EntitySystem & *ents,* tdt::uint *id,* bool *val*)

Sets the free status of a given node.

Parameters

EntitySystem	containing the node.
ID	of the node.
True	for free, false for not-so-free.

Definition at line 46 of file GridNodeHelper.cpp.

5.15.2.9 void GridNodeHelper::set_free_selected (**EntitySystem** & *ents*, **SelectionBox** & *box*, bool *val*)

Applies the GridSystem::set_free method to all currently selected nodes.

Parameters

EntitySystem	containing the nodes.
Reference	to the selection box that selected the nodes.
True	for free, false for not-so-free.

Definition at line 62 of file GridNodeHelper.cpp.

5.15.2.10 void GridNodeHelper::set_portal_neighbour (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *portal*)

Sets the portal node linked to this node.

Parameters

EntitySystem	containing the node.
ID	of this node.
ID	of the portal node.

Definition at line 125 of file GridNodeHelper.cpp.

5.15.2.11 void GridNodeHelper::set_resident (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the resident of a given node.

(Resident is an entity that is causing the node to be not free - like a wall, building etc.)

Parameters

EntitySystem	containing the node and the resident.
ID	of the node.
ID	of the resident.

Definition at line 77 of file GridNodeHelper.cpp.

5.16 HealthHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the health component.

Functions

- void set_health (EntitySystem &, tdt::uint, tdt::uint)

Sets the health of a given entity without any regard to it's maximal health.

- tdt::uint get_health (EntitySystem &, tdt::uint)

  Returns the current health amount of a given entity.
- void add_health (EntitySystem &, tdt::uint, tdt::uint)

  Increases the current health amount of an entity by a given amount up to the maximum value stored in it's Health↩ Component.
- void sub_health (EntitySystem &, tdt::uint, tdt::uint, bool=false)

  Subtracts a given amount from the current health of an entity, taking it's defense into account by default.
- void heal (EntitySystem &, tdt::uint)

  Sets the current health amount of an entity to it's maximum value.
- void buff (EntitySystem &, tdt::uint, tdt::uint)

  Increases the current and maximum health amount of an entity by a given value.
- void debuff (EntitySystem &, tdt::uint, tdt::uint)

  Reduces the current and maximum health amount of an entity by a given value.
- void set_regen (EntitySystem &, tdt::uint, tdt::uint)

  Sets the regeneration value of a given entity.
- tdt::uint get_regen (EntitySystem &, tdt::uint)

  Returns the regeneration value of a given entity.
- void set_alive (EntitySystem &, tdt::uint, bool)

  Allows to set the health status of an entity without adding/subing health.
- bool is_alive (EntitySystem &, tdt::uint)

  Returns true if a given entity is alive, false otherwise.
- void set_defense (EntitySystem &, tdt::uint, tdt::uint)

  Sets the amount of defense a given entity has to a given (absolute) amount.
- tdt::uint get_defense (EntitySystem &, tdt::uint)

  Returns the defense of a given entity.
- void add_defense (EntitySystem &, tdt::uint, tdt::uint)

  Increases the defense of an entity by a given amount.
- void sub_defense (EntitySystem &, tdt::uint, tdt::uint)

  Reduces the defense of an entity by a given amount.
- void ubercharge (EntitySystem &, tdt::uint)

  A cheat that sets the health, maximum health and defense to their highest possible values.

5.16.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the health component.

5.16.2 Function Documentation

5.16.2.1 void HealthHelper::add_defense (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Increases the defense of an entity by a given amount.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Amount	of defense to be added.

Definition at line 163 of file HealthHelper.cpp.

5.16.2.2 void HealthHelper::add_health (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Increases the current health amount of an entity by a given amount up to the maximum value stored in it's Health↩ Component.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Amount	of health to be added. Node: To increase health along with the maximum value, see HealthSystem::buff.

Definition at line 31 of file HealthHelper.cpp.

5.16.2.3 void HealthHelper::buff (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Increases the current and maximum health amount of an entity by a given value.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Amount	of health to be added.

Definition at line 79 of file HealthHelper.cpp.

5.16.2.4 void HealthHelper::debuff (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Reduces the current and maximum health amount of an entity by a given value.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Amount	of health to be subtracted.

Definition at line 93 of file HealthHelper.cpp.

5.16.2.5 tdt::uint HealthHelper::get_defense (EntitySystem & *ents*, tdt::uint *id*)

Returns the defense of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 154 of file HealthHelper.cpp.

5.16.2.6 tdt::uint HealthHelper::get_health (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the current health amount of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 22 of file HealthHelper.cpp.

5.16.2.7 tdt::uint HealthHelper::get_regen (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the regeneration value of a given entity.

Parameters

Reference	to the entity system containing components.
ID	o fthe entity.

Definition at line 122 of file HealthHelper.cpp.

5.16.2.8 void HealthHelper::heal (**EntitySystem** & *ents*, tdt::uint *id*)

Sets the current health amount of an entity to it's maximum value.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 66 of file HealthHelper.cpp.

5.16.2.9 bool HealthHelper::is_alive (**EntitySystem** & *ents*, tdt::uint *id*)

Returns true if a given entity is alive, false otherwise.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 138 of file HealthHelper.cpp.

5.16.2.10 void HealthHelper::set_alive (EntitySystem & *ents,* tdt::uint *id,* bool *val*)

Allows to set the health status of an entity without adding/subing health.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
True	for "alive" and false for "dead".

Definition at line 131 of file HealthHelper.cpp.

5.16.2.11 void HealthHelper::set_defense (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the amount of defense a given entity has to a given (absolute) amount.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
The	new defense value.

Definition at line 147 of file HealthHelper.cpp.

5.16.2.12 void HealthHelper::set_health (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the health of a given entity without any regard to it's maximal health.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
The	new health value.

Definition at line 7 of file HealthHelper.cpp.

5.16.2.13 void HealthHelper::set_regen (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the regeneration value of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
The	new regen value.

Definition at line 115 of file HealthHelper.cpp.

5.16.2.14 void HealthHelper::sub_defense (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Reduces the defense of an entity by a given amount.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Amounf	of defense to be removed.

Definition at line 170 of file HealthHelper.cpp.

5.16.2.15 void HealthHelper::sub_health (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*, bool *ignore_armor* = `false`)

Subtracts a given amount from the current health of an entity, taking it's defense into account by default.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Amount	of health to be subtracted.
Optional	boolean indicator, if true, the entity's defense will be ignored, otherwise it will be subtracted from the given amount.

Definition at line 44 of file HealthHelper.cpp.

5.16.2.16 void HealthHelper::ubercharge (EntitySystem & *ents*, tdt::uint *id*)

A cheat that sets the health, maximum health and defense to their highest possible values.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 182 of file HealthHelper.cpp.

5.17 HomingHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the homing component.

Functions

- void set_source (EntitySystem &, tdt::uint, tdt::uint)

  Sets the ID of the source (entity that shot it) of a given projectile.
- tdt::uint get_source (EntitySystem &, tdt::uint)

  Returns the ID of the entity that shot a given projectile.
- void set_target (EntitySystem &, tdt::uint, tdt::uint)

  Changes the target of a given homing projectile.
- tdt::uint get_target (EntitySystem &, tdt::uint)

  Returns the ID of the target of a given homing projectile.
- void set_dmg (EntitySystem &, tdt::uint, tdt::uint)

  Changes the damage of a given homing projectile.
- tdt::uint get_dmg (EntitySystem &, tdt::uint)

  Returns the damage value of a given projectile.

5.17.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the homing component.

5.17.2 Function Documentation

5.17.2.1 tdt::uint HomingHelper::get_dmg (EntitySystem & *ents*, tdt::uint *id*)

Returns the damage value of a given projectile.

Parameters

Reference	to the entity system containing components.
ID	of the projectile.

Definition at line 44 of file HomingHelper.cpp.

5.17.2.2 tdt::uint HomingHelper::get_source (EntitySystem & *ents*, tdt::uint *id*)

Returns the ID of the entity that shot a given projectile.

Parameters

Reference	to the entity system containing components.
ID	of the projectile.

Definition at line 12 of file HomingHelper.cpp.

5.17.2.3 tdt::uint HomingHelper::get_target (EntitySystem & *ents*, tdt::uint *id*)

Returns the ID of the target of a given homing projectile.

Parameters

Reference	to the entity system containing components.
ID	of the projectile.

Definition at line 28 of file HomingHelper.cpp.

5.17.2.4 void HomingHelper::set_dmg (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *dmg*)

Changes the damage of a given homing projectile.

Parameters

Reference	to the entity system containing components.
ID	of the projectile.
The	new damage value.

Definition at line 37 of file HomingHelper.cpp.

5.17.2.5 void HomingHelper::set_source (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *source*)

Sets the ID of the source (entity that shot it) of a given projectile.

Parameters

Reference	to the entity system containing components.
ID	of the projectile.
ID	of the source.

Definition at line 5 of file HomingHelper.cpp.

5.17.2.6 void HomingHelper::set_target (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *target*)

Changes the target of a given homing projectile.

Parameters

Reference	to the entity system containing components.
ID	of the projectile.
ID	of the target.

Definition at line 21 of file HomingHelper.cpp.

5.18 InputHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the input component.

Functions

- void set_input_handler (EntitySystem &, tdt::uint, const std::string &)

  Changes the name of the table that contains a given entity's input handler.
- const std::string & get_input_handler (EntitySystem &, tdt::uint)

  Returns name of the table that contains the input handling function of a given entity.

5.18.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the input component.

5.18.2 Function Documentation

5.18.2.1 const std::string & InputHelper::get_input_handler (EntitySystem & *ents,* tdt::uint *id*)

Returns name of the table that contains the input handling function of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 12 of file InputHelper.cpp.

5.18.2.2 void InputHelper::set_input_handler (EntitySystem & *ents,* tdt::uint *id,* const std::string & *handler*)

Changes the name of the table that contains a given entity's input handler.

Parameters

Reference	to the entity system containing components.
Name	of the new input handler (Lua function).

Note

The handler recieves the ID of the entity and the key number when it's called, for current keybindings use the game.enum.input Lua table.

Definition at line 5 of file InputHelper.cpp.

5.19 level_generators Namespace Reference

Namespace that contains different level generators that can be used in the game as well as the typedef for the DEFAULT_LEVEL_GENERATOR.

Classes

- class LevelGenerator

  Abstract parent class of all level generators, allows for different level generators used to create levels with minimal effort.
- class RandomLevelGenerator

  Level generator that uses simple RNG approach (counts the number of gold neighbours and increases the chance to spawn a gold deposit if needed).

Typedefs

- using DEFAULT_LEVEL_GENERATOR = RandomLevelGenerator

  Typedef used as the default level generator when the game starts.

5.19.1 Detailed Description

Namespace that contains different level generators that can be used in the game as well as the typedef for the DEFAULT_LEVEL_GENERATOR.

5.19.2 Typedef Documentation

5.19.2.1 using **level_generators::DEFAULT_LEVEL_GENERATOR** = typedef **RandomLevelGenerator**

Typedef used as the default level generator when the game starts.

Definition at line 83 of file LevelGenerators.hpp.

5.20 LightHelper Namespace Reference

Namespace containing auxiliary functions that help with the management of the light component.

Functions

- void set_visible (EntitySystem &, tdt::uint, bool)

  Sets the visibility status of a given light entity.
- void toggle_visible (EntitySystem &, tdt::uint)

  Toggles (visible -> invisible and vice versa) the visibility status of a given light entity.
- bool is_visible (EntitySystem &, tdt::uint)

  Returns the visibility status of a given light entity.
- void init (EntitySystem &, tdt::uint)

  Performs OGRE initialization of the light component of a given entity, used when loading a saved game.

5.20.1 Detailed Description

Namespace containing auxiliary functions that help with the management of the light component.

5.20.2 Function Documentation

5.20.2.1 void LightHelper::init (EntitySystem & *ents*, tdt::uint *id*)

Performs OGRE initialization of the light component of a given entity, used when loading a saved game.

Parameters

Entity	system that contains the light entity.
ID	of the entity.

Definition at line 28 of file LightHelper.cpp.

5.20.2.2 bool LightHelper::is_visible (EntitySystem & *ents*, tdt::uint *id*)

Returns the visibility status of a given light entity.

Parameters

Entity	system that contains the light entity.
ID	of the entity.

Definition at line 19 of file LightHelper.cpp.

5.20.2.3 void LightHelper::set_visible (EntitySystem & *ents*, tdt::uint *id*, bool *val*)

Sets the visibility status of a given light entity.

Parameters

Entity	system that contains the light entity.

Parameters

ID	of the entity.
The	new visibility status (true for visible).

Definition at line 5 of file LightHelper.cpp.

5.20.2.4 void LightHelper::toggle_visible (EntitySystem & *ents,* tdt::uint *id*)

Toggles (visible -> invisible and vice versa) the visibility status of a given light entity.

Parameters

Entity	system that contains the light entity.
ID	of the entity.

Definition at line 12 of file LightHelper.cpp.

5.21 LimitedLifeSpanHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the limited life span component.

Functions

- void set_max_time (EntitySystem &, tdt::uint, tdt::real)
  Sets the life span of a given entity.
- tdt::real get_max_time (EntitySystem &, tdt::uint)
  Returns the life span of a given entity.
- tdt::real get_curr_time (EntitySystem &, tdt::uint)
  Returns the life time of a given entity.
- void advance_curr_time (EntitySystem &, tdt::uint, tdt::real)
  Advances the life time of a given entity by a given value.

5.21.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the limited life span component.

5.21.2 Function Documentation

5.21.2.1 void LimitedLifeSpanHelper::advance_curr_time (EntitySystem & *ents,* tdt::uint *id,* tdt::real *val*)

Advances the life time of a given entity by a given value.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	value to advance by.

Definition at line 30 of file LimitedLifeSpanHelper.cpp.

5.21.2.2 tdt::real LimitedLifeSpanHelper::get_curr_time (EntitySystem & *ents,* tdt::uint *id*)

Returns the life time of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 21 of file LimitedLifeSpanHelper.cpp.

5.21.2.3 tdt::real LimitedLifeSpanHelper::get_max_time (EntitySystem & *ents,* tdt::uint *id*)

Returns the life span of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 12 of file LimitedLifeSpanHelper.cpp.

5.21.2.4 void LimitedLifeSpanHelper::set_max_time (EntitySystem & *ents,* tdt::uint *id,* tdt::real *val*)

Sets the life span of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new life span.

Definition at line 5 of file LimitedLifeSpanHelper.cpp.

5.22 ManaCrystalHelper Namespace Reference

Auxiliary namespace that contains functions that help with the management of the mana crystal component.

Functions

- void set_capacity (EntitySystem &, tdt::uint, tdt::uint)

  Sets the amount of max mana this entity adds to the player's mana capacity.
- tdt::uint get_capacity (EntitySystem &, tdt::uint)

  Returns the amount of max mana this entity adds to the player's mana capacity.
- void set_regen (EntitySystem &, tdt::uint, tdt::uint)

  Sets the amount of mana regen this entity adds to the player's total mana regen.
- tdt::uint get_regen (EntitySystem &, tdt::uint)

  Sets the amount of mana regen this entity adds to the player's total mana regen.

5.22.1 Detailed Description

Auxiliary namespace that contains functions that help with the management of the mana crystal component.

5.22.2 Function Documentation

5.22.2.1 tdt::uint ManaCrystalHelper::get_capacity (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the amount of max mana this entity adds to the player's mana capacity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 19 of file ManaCrystalHelper.cpp.

5.22.2.2 tdt::uint ManaCrystalHelper::get_regen (**EntitySystem** & *ents,* tdt::uint *id*)

Sets the amount of mana regen this entity adds to the player's total mana regen.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 41 of file ManaCrystalHelper.cpp.

5.22.2.3 void ManaCrystalHelper::set_capacity (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the amount of max mana this entity adds to the player's mana capacity.

Parameters

EntitySystem	that contains the entity.

Parameters

ID	of the entity.
The	new capacity value.

Definition at line 6 of file ManaCrystalHelper.cpp.

5.22.2.4 void ManaCrystalHelper::set_regen (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the amount of mana regen this entity adds to the player's total mana regen.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new capacity value.

Definition at line 28 of file ManaCrystalHelper.cpp.

5.23 ManaHelper Namespace Reference

Auxiliary namespace that contains functions that help with the management of the mana component.

Functions

- void add_mana (EntitySystem &, tdt::uint, tdt::uint)
  Adds a given amount of mana to a given entity's current mana pool.
- bool sub_mana (EntitySystem &, tdt::uint, tdt::uint)
  Removes a given amount of mana from a given entity's current mana pool.
- void set_mana (EntitySystem &, tdt::uint, tdt::uint)
  Sets the amount of mana a given entity has to a given amount.
- tdt::uint get_mana (EntitySystem &, tdt::uint)
  Returns the amount of mana a given entity has.
- void set_max_mana (EntitySystem &, tdt::uint, tdt::uint)
  Sets the max amount of mana a given entity can have.
- tdt::uint get_max_mana (EntitySystem &, tdt::uint)
  Returns the max amount of mana a given entity can have.
- void set_regen (EntitySystem &, tdt::uint, tdt::uint)
  Sets the mana regeneration value of a given entity.
- tdt::uint get_regen (EntitySystem &, tdt::uint)
  Returns the mana regeneration value of a given entity.

5.23.1 Detailed Description

Auxiliary namespace that contains functions that help with the management of the mana component.

5.23.2 Function Documentation

5.23.2.1 void ManaHelper::add_mana (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *val*)

Adds a given amount of mana to a given entity's current mana pool.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Amount	of mana to add.

Definition at line 6 of file ManaHelper.cpp.

5.23.2.2 tdt::uint ManaHelper::get_mana (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the amount of mana a given entity has.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 48 of file ManaHelper.cpp.

5.23.2.3 tdt::uint ManaHelper::get_max_mana (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the max amount of mana a given entity can have.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 72 of file ManaHelper.cpp.

5.23.2.4 tdt::uint ManaHelper::get_regen (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the mana regeneration value of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 94 of file ManaHelper.cpp.

5.23.2.5 void ManaHelper::set_mana (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the amount of mana a given entity has to a given amount.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new amount of mana.

Definition at line 35 of file ManaHelper.cpp.

5.23.2.6 void ManaHelper::set_max_mana (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the max amount of mana a given entity can have.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new mana limit.

Definition at line 57 of file ManaHelper.cpp.

5.23.2.7 void ManaHelper::set_regen (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the mana regeneration value of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new mana regeneration value.

Definition at line 81 of file ManaHelper.cpp.

5.23.2.8 bool ManaHelper::sub_mana (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *val*)

Removes a given amount of mana from a given entity's current mana pool.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Amount	of mana to subtract.

Definition at line 19 of file ManaHelper.cpp.

5.24 MovementHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the movement component.

Functions

- tdt::real get_speed_modifier (EntitySystem &, tdt::uint)
  Returns the speed modifier of a given entity.
- void set_speed_modifier (EntitySystem &, tdt::uint, tdt::real)
  Changes the speed modifier of a given entity to a given value.
- Ogre::Vector3 dir_to (EntitySystem &, tdt::uint, tdt::uint)
  Returns the direction from a given entity to another given entity.
- Ogre::Vector3 get_dir (EntitySystem &, tdt::uint)
  Returns the direction a given entity is facing.
- Ogre::Vector3 get_dir_back (EntitySystem &, tdt::uint)
  Returns the opposite direction to the direction a given entity is facing.
- Ogre::Vector3 get_dir_left (EntitySystem &, tdt::uint)
  Returns the direction perpendicular to the direction a given entity is facing.
- Ogre::Vector3 get_dir_right (EntitySystem &, tdt::uint)
  Returns the direction perpendicular to the direction a given entity is facing.
- void set_original_speed (EntitySystem &, tdt::uint, tdt::real)
  Sets the original value of a given entity's speed.
- tdt::real get_original_speed (EntitySystem &, tdt::uint)
  Returns the original speed of a given entity.
- void reset_speed (EntitySystem &, tdt::uint)
  Sets the speed of a given entity to it's original value.

5.24.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the movement component.

5.24.2 Function Documentation

5.24.2.1 Ogre::Vector3 MovementHelper::dir_to (EntitySystem & *ents,* tdt::uint *id1,* tdt::uint *id2*)

Returns the direction from a given entity to another given entity.

Parameters

Entity	system containing the entity.
ID	of the first entity.
ID	of the second entity.

Definition at line 21 of file MovementHelper.cpp.

5.24.2.2 Ogre::Vector3 MovementHelper::get_dir (EntitySystem & *ents*, tdt::uint *id*)

Returns the direction a given entity is facing.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 34 of file MovementHelper.cpp.

5.24.2.3 Ogre::Vector3 MovementHelper::get_dir_back (EntitySystem & *ents*, tdt::uint *id*)

Returns the opposite direction to the direction a given entity is facing.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 43 of file MovementHelper.cpp.

5.24.2.4 Ogre::Vector3 MovementHelper::get_dir_left (EntitySystem & *ents*, tdt::uint *id*)

Returns the direction perpendicular to the direction a given entity is facing.

(To the left.)

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 52 of file MovementHelper.cpp.

5.24.2.5 Ogre::Vector3 MovementHelper::get_dir_right (EntitySystem & *ents*, tdt::uint *id*)

Returns the direction perpendicular to the direction a given entity is facing.

(To the right.)

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 61 of file MovementHelper.cpp.

5.24.2.6 tdt::real MovementHelper::get_original_speed (EntitySystem & *ents*, tdt::uint *id*)

Returns the original speed of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 77 of file MovementHelper.cpp.

5.24.2.7 tdt::real MovementHelper::get_speed_modifier (EntitySystem & *ents*, tdt::uint *id*)

Returns the speed modifier of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 5 of file MovementHelper.cpp.

5.24.2.8 void MovementHelper::reset_speed (EntitySystem & *ents*, tdt::uint *id*)

Sets the speed of a given entity to it's original value.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 86 of file MovementHelper.cpp.

5.24.2.9 void MovementHelper::set_original_speed (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the original value of a given entity's speed.

(Used for serialization.)

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 70 of file MovementHelper.cpp.

5.24.2.10 void MovementHelper::set_speed_modifier (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Changes the speed modifier of a given entity to a given value.

Parameters

Entity	system containing the entity.
ID	of the entity.
New	speed value.

Definition at line 14 of file MovementHelper.cpp.

5.25 NameHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the name component.

Functions

- void set_name (EntitySystem &, tdt::uint, const std::string &)
  Sets the name of a given entity.
- const std::string & get_name (EntitySystem &, tdt::uint)
  Returns the name of a given entity.

5.25.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the name component.

5.25.2 Function Documentation

5.25.2.1 const std::string & NameHelper::get_name (EntitySystem & *ents*, tdt::uint *id*)

Returns the name of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 12 of file NameHelper.cpp.

5.25.2.2 void NameHelper::set_name (**EntitySystem** & *ents,* tdt::uint *id,* const std::string & *val*)

Sets the name of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new name.

Definition at line 5 of file NameHelper.cpp.

5.26 NotificationHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the notification component.

Functions

- void set_cooldown (EntitySystem &, tdt::uint, tdt::real)
  Sets the cooldown between notifications a given entity can send to the game's log.
- tdt::real get_cooldown (EntitySystem &, tdt::uint)
  Returns the cooldown between notifications a given entity can send to the game's log.
- void reset (EntitySystem &, tdt::uint)
  Resets the cooldown between notifications a given entity can send to the game's log.
- bool notify (EntitySystem &, tdt::uint, const std::string &)
- tdt::real get_curr_time (EntitySystem &, tdt::uint)
  Returns the time since the last notification a given entity has sent to the game's log.
- void advance_curr_time (EntitySystem &, tdt::uint, tdt::real)
  Advances the time since the last notification a given entity has sent to the game's log.

5.26.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the notification component.

5.26.2 Function Documentation

5.26.2.1 void NotificationHelper::advance_curr_time (**EntitySystem** & *ents,* tdt::uint *id,* tdt::real *val*)

Advances the time since the last notification a given entity has sent to the game's log.

Parameters

Entity	system containing the entity.
ID	of the entity.
Time	amount to advance by.

Definition at line 51 of file NotificationHelper.cpp.

5.26.2.2 tdt::real NotificationHelper::get_cooldown (EntitySystem & *ents,* tdt::uint *id*)

Returns the cooldown between notifications a given entity can send to the game's log.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 13 of file NotificationHelper.cpp.

5.26.2.3 tdt::real NotificationHelper::get_curr_time (EntitySystem & *ents,* tdt::uint *id*)

Returns the time since the last notification a given entity has sent to the game's log.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 42 of file NotificationHelper.cpp.

5.26.2.4 bool NotificationHelper::notify (EntitySystem & *ents,* tdt::uint *id,* const std::string & *msg*)

Parameters

Sends	a notification to the game's log while respecting the notification cooldown of the entity that sends the notification. (And sets the cooldown if needed.)
Entity	system containing the entity.
ID	of the entity.

Definition at line 29 of file NotificationHelper.cpp.

5.26.2.5 void NotificationHelper::reset (EntitySystem & *ents,* tdt::uint *id*)

Resets the cooldown between notifications a given entity can send to the game's log.

(That means that the entity can notify immediately.)

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 22 of file NotificationHelper.cpp.

5.26.2.6 void NotificationHelper::set_cooldown (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the cooldown between notifications a given entity can send to the game's log.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new cooldown.

Definition at line 6 of file NotificationHelper.cpp.

5.27 OnHitHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the on hit component.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)
  Sets the blueprint table handling incoming hits of a given entity.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)
  Returns the name of the on hit blueprint of a given entity.
- void call (EntitySystem &, tdt::uint, tdt::uint)
  Calls the on hit handler of a given entity.
- void set_cooldown (EntitySystem &, tdt::uint, tdt::real)
  Sets the cooldown between on hit blueprint calls of a given entity.
- tdt::real get_cooldown (EntitySystem &, tdt::uint)
  Returns the cooldown between on hit blueprint calls of a given entity.

5.27.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the on hit component.

5.27.2 Function Documentation

5.27.2.1 void OnHitHelper::call (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *hitter*)

Calls the on hit handler of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
ID	of the hitter. (Source of the attack.)

Definition at line 24 of file OnHitHelper.cpp.

5.27.2.2 const std::string & OnHitHelper::get_blueprint (EntitySystem & *ents*, tdt::uint *id*)

Returns the name of the on hit blueprint of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 13 of file OnHitHelper.cpp.

5.27.2.3 tdt::real OnHitHelper::get_cooldown (EntitySystem & *ents*, tdt::uint *id*)

Returns the cooldown between on hit blueprint calls of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 41 of file OnHitHelper.cpp.

5.27.2.4 void OnHitHelper::set_blueprint (EntitySystem & *ents*, tdt::uint *id*, const std::string & *val*)

Sets the blueprint table handling incoming hits of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new blueprint name.

Definition at line 6 of file OnHitHelper.cpp.

5.27.2.5 void OnHitHelper::set_cooldown (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the cooldown between on hit blueprint calls of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new cooldown.

Definition at line 34 of file OnHitHelper.cpp.

5.28 PathfindingHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the pathfinding component.

Functions

- const std::string & get_pathpfinding_blueprint (EntitySystem &, tdt::uint)

    Returns constant reference to the pathfinding blueprint of a given entity.
- void set_pathfinding_blueprint (EntitySystem &, tdt::uint, const std::string &)

    Changes the pathfinding blueprint of a given entity.
- std::deque< tdt::uint > & get_path (EntitySystem &, tdt::uint)

    Returns the node queue of a given entity's path.
- bool can_break (tdt::uint, const PathfindingComponent &, tdt::uint)

    Returns true if a given entity can break a structure residing on a given node (if any).
- tdt::real get_cost (tdt::uint, const PathfindingComponent &, tdt::uint, DIRECTION::VAL)

    Returns the cost a journey to a given node takes for a given entity.

5.28.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the pathfinding component.

5.28.2 Function Documentation

5.28.2.1 bool PathfindingHelper::can_break (tdt::uint *id1,* const **PathfindingComponent** & *comp,* tdt::uint *id2*)

Returns true if a given entity can break a structure residing on a given node (if any).

Parameters

ID	of the entity.
Pathfinding	component of the entity.
ID	of the node.

Definition at line 35 of file PathfindingHelper.cpp.

5.28.2.2 tdt::real PathfindingHelper::get_cost (tdt::uint *id1,* const **PathfindingComponent** & *comp,* tdt::uint *id2,* DIRECTION::VAL *dir*)

Returns the cost a journey to a given node takes for a given entity.

Parameters

ID	of the entity.
Pathfinding	component of the entity.
ID	of the node.

Definition at line 40 of file PathfindingHelper.cpp.

5.28.2.3 std::deque< tdt::uint > & PathfindingHelper::get_path (EntitySystem & *ents,* tdt::uint *id*)

Returns the node queue of a given entity's path.

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 24 of file PathfindingHelper.cpp.

5.28.2.4 const std::string & PathfindingHelper::get_pathpfinding_blueprint (EntitySystem & *ents,* tdt::uint *id*)

Returns constant reference to the pathfinding blueprint of a given entity.

(Which is used for the can_pass & can_break methods in lua.)

Parameters

EntitySystem	containing the entity.
ID	of the entity.

Definition at line 6 of file PathfindingHelper.cpp.

5.28.2.5 void PathfindingHelper::set_pathfinding_blueprint (EntitySystem & *ents,* tdt::uint *id,* const std::string & *blueprint*)

Changes the pathfinding blueprint of a given entity.

Parameters

EntitySystem	containing the entity.
ID	of the entity.
Name	of the new blueprint table.

Definition at line 17 of file PathfindingHelper.cpp.

5.29 PhysicsHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the physics component.

Functions

- void set_solid (EntitySystem &, tdt::uint, bool)

  Changes the solid state of a given entity.
- bool is_solid (EntitySystem &, tdt::uint)

  Returns true if a given entity is solid, false otherwise.
- void set_position (EntitySystem &, tdt::uint, const Ogre::Vector3 &)

  Sets the position of a given entity.
- const Ogre::Vector3 & get_position (EntitySystem &, tdt::uint)

  Returns the position of a given entity.
- void set_half_height (EntitySystem &, tdt::uint, tdt::real)

  Sets the half height of a given entity (which is used to balance the fact that some models do not have their center on the level of their feet).
- tdt::real get_half_height (EntitySystem &, tdt::uint)

  Returns the half height of a given entity (which is used to balance the fact that some models do not have their center on the level of their feet).
- void move_to (EntitySystem &, tdt::uint, Ogre::Vector3)

  Moves a given entity to a given point in space (absolute movement).
- tdt::real get_distance (EntitySystem &, tdt::uint, tdt::uint)

  Returns the distance between two given entities.
- tdt::real get_angle (Ogre::Vector3, Ogre::Vector3)

  Returns the size of the angle bewteen two given vectors as tdt::real which can then be used in Ogre::Radian constructor for conversion to Radians.
- void set_2d_position (EntitySystem &, tdt::uint, Ogre::Vector2)

  Sets the position of a given entity disregarding it's Y coordinate.
- Ogre::Vector2 get_2d_position (EntitySystem &, tdt::uint)

  Returns the X and Z coordinates of a given entity's position.

5.29.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the physics component.

5.29.2 Function Documentation

5.29.2.1 Ogre::Vector2 PhysicsHelper::get_2d_position (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the X and Z coordinates of a given entity's position.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Note

Cannot use const reference like in the usual get_position function because this method creates a proxy Vector2 which cannot be passed as an lvalue reference due to it being a temporary object.

Definition at line 108 of file PhysicsHelper.cpp.

5.29.2.2 tdt::real PhysicsHelper::get_angle (Ogre::Vector3 *v1*, Ogre::Vector3 *v2*)

Returns the size of the angle bewteen two given vectors as tdt::real which can then be used in Ogre::Radian constructor for conversion to Radians.

(The reason for this is that Lua does not have the notion of radians and as such using floating point numbers is easier.)

Parameters

Vector	#1.
Vector	#2.

Definition at line 88 of file PhysicsHelper.cpp.

5.29.2.3 tdt::real PhysicsHelper::get_distance (EntitySystem & *ents*, tdt::uint *id1*, tdt::uint *id2*)

Returns the distance between two given entities.

Parameters

Reference	to the entity system containing components.
ID	of the first entity.
ID	of the second entity.

Definition at line 73 of file PhysicsHelper.cpp.

5.29.2.4 tdt::real PhysicsHelper::get_half_height (EntitySystem & *ents*, tdt::uint *id*)

Returns the half height of a given entity (which is used to balance the fact that some models do not have their center on the level of their feet).

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 51 of file PhysicsHelper.cpp.

5.29.2.5 const Ogre::Vector3 & PhysicsHelper::get_position (EntitySystem & *ents*, tdt::uint *id*)

Returns the position of a given entity.

Parameters

Reference	to the entity system containing components. Parma: ID of the entity.

Definition at line 34 of file PhysicsHelper.cpp.

5.29.2.6 bool PhysicsHelper::is_solid (EntitySystem & *ents*, tdt::uint *id*)

Returns true if a given entity is solid, false otherwise.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 13 of file PhysicsHelper.cpp.

5.29.2.7 void PhysicsHelper::move_to (EntitySystem & *ents*, tdt::uint *id*, Ogre::Vector3 *pos*)

Moves a given entity to a given point in space (absolute movement).

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Target	coordinate.

Note

The difference with set_position is that this function also sets the position of the scene node this entity is attached to (if such node exists) and thus moves the entity's model as well.

Definition at line 60 of file PhysicsHelper.cpp.

5.29.2.8 void PhysicsHelper::set_2d_position (EntitySystem & *ents*, tdt::uint *id*, Ogre::Vector2 *val*)

Sets the position of a given entity disregarding it's Y coordinate.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new position (vector2 containing the x and z coordinates.)

Definition at line 93 of file PhysicsHelper.cpp.

5.29.2.9 void PhysicsHelper::set_half_height (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the half height of a given entity (which is used to balance the fact that some models do not have their center on the level of their feet).

Parameters

Reference	to the entity system containing components.
ID	of the entity.
The	new half height value.

Definition at line 44 of file PhysicsHelper.cpp.

5.29.2.10 void PhysicsHelper::set_position (EntitySystem & *ents*, tdt::uint *id*, const Ogre::Vector3 & *val*)

Sets the position of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
The	new position.

Note

The difference with move_to is that this function does not set the position of the scene node this entity is attached to and thus does not move the entity's model as well.

Definition at line 22 of file PhysicsHelper.cpp.

5.29.2.11 void PhysicsHelper::set_solid (EntitySystem & *ents*, tdt::uint *id*, bool *val*)

Changes the solid state of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.
True	for solid, false for non-solid.

Definition at line 6 of file PhysicsHelper.cpp.

5.30 PriceHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the price component.

Functions

- void set_price (EntitySystem &, tdt::uint, tdt::uint)

  Sets the price of a given entity.
- tdt::uint get_price (EntitySystem &, tdt::uint)

  Returns the price of a given entity.

5.30.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the price component.

5.30.2 Function Documentation

5.30.2.1 tdt::uint PriceHelper::get_price (EntitySystem & *ents*, tdt::uint *id*)

Returns the price of a given entity.

Parameters

Entity	system that contains the entity.
ID	of the entity.

Definition at line 12 of file PriceHelper.cpp.

5.30.2.2 void PriceHelper::set_price (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the price of a given entity.

Parameters

Entity	system that contains the entity.
ID	of the entity.
The	new price.

Definition at line 5 of file PriceHelper.cpp.

5.31 ProductHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the product component.

Functions

- void set_producer (EntitySystem &, tdt::uint, tdt::uint)

  Set's the producer of a given entity (the building that spawned it).
- tdt::uint get_producer (EntitySystem &, tdt::uint)

  Returns the producer of a given entity.

5.31.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the product component.

5.31.2 Function Documentation

5.31.2.1 tdt::uint ProductHelper::get_producer (EntitySystem & *ents,* tdt::uint *id*)

Returns the producer of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 12 of file ProductHelper.cpp.

5.31.2.2 void ProductHelper::set_producer (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *producer*)

Set's the producer of a given entity (the building that spawned it).

Parameters

Reference	to the entity system containing components.
ID	of the entity.
ID	of the producer.

Definition at line 5 of file ProductHelper.cpp.

5.32 ProductionHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the production component.

Functions

- void set_production_blueprint (EntitySystem &, tdt::uint, const std::string &)

Changes the name of the blueprint table used to spawn new entities.

- const std::string & get_production_blueprint (EntitySystem &, tdt::uint)

  Returns the name of the blueprint table used to spawn new entities.
- void set_production_limit (EntitySystem &, tdt::uint, tdt::uint)

  Sets the maximal number of entities a given building can spawn.
- tdt::uint get_production_limit (EntitySystem &, tdt::uint)

  Returns the maximal number of entities a given building can spawn.
- void set_production_cooldown (EntitySystem &, tdt::uint, tdt::real)

  Sets the time it takes for a given building to spawn a single entity.
- tdt::real get_production_cooldown (EntitySystem &, tdt::uint)

  Returns the time it takes for a given building to spawn a single entity.
- void set_production_progress (EntitySystem &, tdt::uint, tdt::real)

  Sets the current spawning progress (in seconds, not %).
- tdt::real get_production_progress (EntitySystem &, tdt::uint)

  Returns the current spawning progress (in seconds, not %).
- void set_production_count (EntitySystem &, tdt::uint, tdt::uint)

  Sets the amount of entities spawned by a given building, but does not spawn or delete entities to match this number(!).
- tdt::uint get_production_count (EntitySystem &, tdt::uint)

  Returns the amount of entities spawned by a given building that are still alive.

5.32.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the production component.

5.32.2 Function Documentation

5.32.2.1 const std::string & ProductionHelper::get_production_blueprint (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the name of the blueprint table used to spawn new entities.

Parameters

Reference	to the entity system containing components.
ID	of the building.

Definition at line 12 of file ProductionHelper.cpp.

5.32.2.2 tdt::real ProductionHelper::get_production_cooldown (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the time it takes for a given building to spawn a single entity.

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 46 of file ProductionHelper.cpp.

5.32.2.3 tdt::uint ProductionHelper::get_production_count (EntitySystem & *ents,* tdt::uint *id*)

Returns the amount of entities spawned by a given building that are still alive.

Parameters

Reference	to the entity system containing components.
ID	of the building.

Definition at line 83 of file ProductionHelper.cpp.

5.32.2.4 tdt::uint ProductionHelper::get_production_limit (EntitySystem & *ents,* tdt::uint *id*)

Returns the maximal number of entities a given building can spawn.

Parameters

Reference	to the entity system containing components.
ID	of the building.

Definition at line 30 of file ProductionHelper.cpp.

5.32.2.5 tdt::real ProductionHelper::get_production_progress (EntitySystem & *ents,* tdt::uint *id*)

Returns the current spawning progress (in seconds, not %).

Parameters

Reference	to the entity system containing components.
ID	of the building.

Definition at line 67 of file ProductionHelper.cpp.

5.32.2.6 void ProductionHelper::set_production_blueprint (EntitySystem & *ents,* tdt::uint *id,* const std::string & *blueprint*)

Changes the name of the blueprint table used to spawn new entities.

Parameters

Reference	to the entity system containing components.
ID	of the building.
Name	of the new building table.

Definition at line 5 of file ProductionHelper.cpp.

5.32.2.7 void ProductionHelper::set_production_cooldown (EntitySystem & *ents*, tdt::uint *id*, tdt::real *cd*)

Sets the time it takes for a given building to spawn a single entity.

Parameters

Reference	to the entity system containing components.
ID	of the building.
The	new entity spawning time.

Definition at line 39 of file ProductionHelper.cpp.

5.32.2.8 void ProductionHelper::set_production_count (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *count*)

Sets the amount of entities spawned by a given building, but does not spawn or delete entities to match this number(!).

Parameters

Reference	to the entity system containing components.
ID	of the building.
The	new entity spawned amount.

Definition at line 76 of file ProductionHelper.cpp.

5.32.2.9 void ProductionHelper::set_production_limit (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *limit*)

Sets the maximal number of entities a given building can spawn.

Parameters

Reference	to the entity system containing components.
ID	of the building.
The	new entity limit.

Definition at line 23 of file ProductionHelper.cpp.

5.32.2.10 void ProductionHelper::set_production_progress (EntitySystem & *ents*, tdt::uint *id*, tdt::real *prog*)

Sets the current spawning progress (in seconds, not %).

Parameters

Reference	to the entity system containing components.
ID	of the building.
The	new spawning progress time.

Note

Time amounts above the cooldown will be adjusted to match the cooldown which will result into instant spawn.

Definition at line 55 of file ProductionHelper.cpp.

5.33 SpellHelper Namespace Reference

Auxiliary namespace that contains functions that help with the management of the spell component.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)
  Sets the blueprint table of the spell a given entity can cast.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)
  Returns the blueprint table of the spell a given entity can cast.
- void set_cooldown (EntitySystem &, tdt::uint, tdt::real)
  Sets the time period between casts of a given entity.
- tdt::real get_cooldown (EntitySystem &, tdt::uint)
  Returns the time period between casts of a given entity.
- void advance_curr_time (EntitySystem &, tdt::uint, tdt::real)
  Advances the timer before the next spell can be cast by a given entity.
- void set_curr_time (EntitySystem &, tdt::uint, tdt::real)
  Sets the timer before the next spell can be cast by a given entity.
- tdt::real get_curr_time (EntitySystem &, tdt::uint)
  Returns the cooldown timer value of a given entity.
- void cast (EntitySystem &, tdt::uint)
  Makes a given entity to cast it's spell (if possible).

5.33.1 Detailed Description

Auxiliary namespace that contains functions that help with the management of the spell component.

5.33.2 Function Documentation

5.33.2.1 void SpellHelper::advance_curr_time (EntitySystem & *ents*, tdt::uint *id*, tdt::real *val*)

Advances the timer before the next spell can be cast by a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Time	to advance by.

Definition at line 38 of file SpellHelper.cpp.

5.33.2.2 void SpellHelper::cast (EntitySystem & *ents,* tdt::uint *id*)

Makes a given entity to cast it's spell (if possible).

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Note

Ignores cooldown.

Definition at line 61 of file SpellHelper.cpp.

5.33.2.3 const std::string & SpellHelper::get_blueprint (EntitySystem & *ents,* tdt::uint *id*)

Returns the blueprint table of the spell a given entity can cast.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 13 of file SpellHelper.cpp.

5.33.2.4 tdt::real SpellHelper::get_cooldown (EntitySystem & *ents,* tdt::uint *id*)

Returns the time period between casts of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 29 of file SpellHelper.cpp.

5.33.2.5 tdt::real SpellHelper::get_curr_time (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the cooldown timer value of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 52 of file SpellHelper.cpp.

5.33.2.6 void SpellHelper::set_blueprint (**EntitySystem** & *ents*, tdt::uint *id*, const std::string & *val*)

Sets the blueprint table of the spell a given entity can cast.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new spell blueprint.

Definition at line 6 of file SpellHelper.cpp.

5.33.2.7 void SpellHelper::set_cooldown (**EntitySystem** & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the time period between casts of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new cooldown value.

Definition at line 22 of file SpellHelper.cpp.

5.33.2.8 void SpellHelper::set_curr_time (**EntitySystem** & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the timer before the next spell can be cast by a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	new timer value (cooldown - timer == time remaining).

Definition at line 45 of file SpellHelper.cpp.

5.34 StructureHelper Namespace Reference

Auxiliary namespace containing the functions that help with the management of the structure component.

Functions

- void add_residences (EntitySystem &, tdt::uint, const std::vector< tdt::uint > &)

  Adds the members of a given vector (containing node IDs) as residences of a given entity (that has a structure component).
- void add_residence (EntitySystem &, tdt::uint, tdt::uint)

  Adds a single node as a residence to the residence list of a given entity.
- void set_radius (EntitySystem &, tdt::uint, tdt::uint)

  Sets the radius of the area a given structure occupies.
- tdt::uint get_radius (EntitySystem &, tdt::uint)

  Returns the radius of a structure (amount of grid nodes from the centre of the structure to one of the sides - not including the centre).
- void set_walk_through (EntitySystem &, tdt::uint, bool)

  Sets the walk through field of a structure, causing it to either block or allow pathfinding.
- bool is_walk_through (EntitySystem &, tdt::uint)

  Returns true if pathfinding is possible through the structure, false otherwise.

5.34.1 Detailed Description

Auxiliary namespace containing the functions that help with the management of the structure component.

5.34.2 Function Documentation

5.34.2.1 void StructureHelper::add_residence (EntitySystem & *ents,* tdt::uint *ent_id,* tdt::uint *node_id*)

Adds a single node as a residence to the residence list of a given entity.

Parameters

Reference	to the entity system containing components.
ID	of the structure.
ID	of the residence.

Definition at line 15 of file StructureHelper.cpp.

5.34.2.2 void StructureHelper::add_residences (EntitySystem & *ents,* tdt::uint *ent_id,* const std::vector< tdt::uint > & *residences*)

Adds the members of a given vector (containing node IDs) as residences of a given entity (that has a structure component).

Parameters

Reference	to the entity system containing components.
ID	of the entity.
Vector	containing the IDs of the new residences (nodes the entity is on).

Definition at line 5 of file StructureHelper.cpp.

5.34.2.3 tdt::uint StructureHelper::get_radius (EntitySystem & *ents,* tdt::uint *id*)

Returns the radius of a structure (amount of grid nodes from the centre of the structure to one of the sides - not including the centre).

Parameters

Reference	to the entity system containing components.
ID	of the entity.

Definition at line 29 of file StructureHelper.cpp.

5.34.2.4 bool StructureHelper::is_walk_through (EntitySystem & *ents,* tdt::uint *id*)

Returns true if pathfinding is possible through the structure, false otherwise.

Parameters

Reference	to the entity system containing components.
ID	of the structure.

Definition at line 45 of file StructureHelper.cpp.

5.34.2.5 void StructureHelper::set_radius (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *radius*)

Sets the radius of the area a given structure occupies.

Parameters

Reference	to the entity system containing components.
ID	of the entity (which has a structure component).
The	new radius.

Definition at line 22 of file StructureHelper.cpp.

5.34.2.6 void StructureHelper::set_walk_through (EntitySystem & *ents,* tdt::uint *id,* bool *on_off*)

Sets the walk through field of a structure, causing it to either block or allow pathfinding.

Parameters

Reference	to the entity system containing components.
ID	of the structure.
True	for walkable, false for not walkable.

Definition at line 38 of file StructureHelper.cpp.

5.35 TaskHandlerHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the task handler component.

Functions

- std::deque< tdt::uint > & get_task_queue (EntitySystem &, tdt::uint)

  Returns a reference to the task queue of a given entity.
- bool task_possible (EntitySystem &, tdt::uint, tdt::uint)

  Checks whether an entity can accept and complete a given task.
- bool task_possible (EntitySystem &, tdt::uint, TASK_TYPE)

  Checks whether an entity can accept and complete a task of agiven task type.
- void clear_task_queue (EntitySystem &, tdt::uint)

  Cancels all tasks in a given entity's task queue.
- void add_possible_task (EntitySystem &, tdt::uint, TASK_TYPE)

  Marks a given entity as available to accept tasks of a given task type.
- void delete_possible_task (EntitySystem &, tdt::uint, TASK_TYPE)

  Marks a given entity as unavailable to accept tasks of a given task type.
- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)

  Sets the handling blueprint of a given entity.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)

  Returns the handling blueprint of a given entity.

5.35.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the task handler component.

5.35.2 Function Documentation

5.35.2.1 void TaskHandlerHelper::add_possible_task (**EntitySystem** & *ents,* tdt::uint *id,* TASK_TYPE *type*)

Marks a given entity as available to accept tasks of a given task type.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Type	of the task to be added.

Definition at line 51 of file TaskHandlerHelper.cpp.

5.35.2.2 void TaskHandlerHelper::clear_task_queue (EntitySystem & *ents*, tdt::uint *id*)

Cancels all tasks in a given entity's task queue.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 36 of file TaskHandlerHelper.cpp.

5.35.2.3 void TaskHandlerHelper::delete_possible_task (EntitySystem & *ents*, tdt::uint *id*, TASK_TYPE *type*)

Marks a given entity as unavailable to accept tasks of a given task type.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Type	of the task to be deleted.

Definition at line 58 of file TaskHandlerHelper.cpp.

5.35.2.4 const std::string & TaskHandlerHelper::get_blueprint (EntitySystem & *ents*, tdt::uint *id*)

Returns the handling blueprint of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 72 of file TaskHandlerHelper.cpp.

5.35.2.5 std::deque< tdt::uint > & TaskHandlerHelper::get_task_queue (EntitySystem & *ents*, tdt::uint *id*)

Returns a reference to the task queue of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.

Definition at line 5 of file TaskHandlerHelper.cpp.

5.35.2.6 void TaskHandlerHelper::set_blueprint (EntitySystem & *ents*, tdt::uint *id*, const std::string & *val*)

Sets the handling blueprint of a given entity.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
The	name of the new blueprint.

Definition at line 65 of file TaskHandlerHelper.cpp.

5.35.2.7 bool TaskHandlerHelper::task_possible (EntitySystem & *ents*, tdt::uint *ent_id*, tdt::uint *task_id*)

Checks whether an entity can accept and complete a given task.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
ID	of the task.

Definition at line 15 of file TaskHandlerHelper.cpp.

5.35.2.8 bool TaskHandlerHelper::task_possible (EntitySystem & *ents*, tdt::uint *id*, TASK_TYPE *val*)

Checks whether an entity can accept and complete a task of agiven task type.

Parameters

EntitySystem	that contains the entity.
ID	of the entity.
Task	type to be tested.

Definition at line 27 of file TaskHandlerHelper.cpp.

5.36 TaskHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the task component.

Functions

- void set_task_source (EntitySystem &, tdt::uint, tdt::uint)

  Sets the source of a given task (that is, the entity that is completing the task).
- tdt::uint get_task_source (EntitySystem &, tdt::uint)

  Returns the source of a given task (that is, the entity that is completing the task).
- void set_task_target (EntitySystem &, tdt::uint, tdt::uint)

  Sets the target entity of a given task.
- tdt::uint get_task_target (EntitySystem &, tdt::uint)

  Returns the target entity of a given task.
- void set_task_type (EntitySystem &, tdt::uint, TASK_TYPE)

  Sets the task type of a given task.
- TASK_TYPE get_task_type (EntitySystem &, tdt::uint)

  Returns the task type of a given task.
- void add_task (EntitySystem &, tdt::uint, tdt::uint, bool=false)

  Assigns a new task to an entity (by adding it to the task queue).
- tdt::uint create_task (EntitySystem &, tdt::uint, TASK_TYPE)

  Creates a new task of a given tasks and returns it's ID.
- void cancel_task (EntitySystem &, tdt::uint)

  Destroys the TaskComponent of a given task, effectively stopping it's completion.
- void set_complete (EntitySystem &, tdt::uint)

  Sets a given task to a complete state.
- bool is_complete (EntitySystem &, tdt::uint)

  Returns true if a given task is complete, false otherwise.

5.36.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the task component.

5.36.2 Function Documentation

5.36.2.1 void TaskHelper::add_task (EntitySystem & *ents,* tdt::uint *ent_id,* tdt::uint *task_id,* bool *priority* = `false`)

Assigns a new task to an entity (by adding it to the task queue).

Parameters

Reference	to the entity system containing components.
ID	of the entity.
ID	of the task.
If	true, the task will be added to the fron of the task queue.

Definition at line 53 of file TaskHelper.cpp.

5.36.2.2 void TaskHelper::cancel_task (EntitySystem & *ents,* tdt::uint *task_id*)

Destroys the TaskComponent of a given task, effectively stopping it's completion.

Parameters

Reference	to the entity system containing components.
ID	of the task.

Definition at line 95 of file TaskHelper.cpp.

5.36.2.3 tdt::uint TaskHelper::create_task (EntitySystem & *ents*, tdt::uint *target*, TASK_TYPE *type*)

Creates a new task of a given tasks and returns it's ID.

Parameters

Reference	to the entity system containing components.
ID	of the task's target (goto location, kill target etc.).
Type	of the task.

Definition at line 80 of file TaskHelper.cpp.

5.36.2.4 tdt::uint TaskHelper::get_task_source (EntitySystem & *ents*, tdt::uint *id*)

Returns the source of a given task (that is, the entity that is completing the task).

Parameters

Reference	to the entity system containing components.
Task	ID.

Definition at line 12 of file TaskHelper.cpp.

5.36.2.5 tdt::uint TaskHelper::get_task_target (EntitySystem & *ents*, tdt::uint *id*)

Returns the target entity of a given task.

Parameters

Reference	to the entity system containing components.
Task	ID.

Definition at line 28 of file TaskHelper.cpp.

5.36.2.6 TASK_TYPE TaskHelper::get_task_type (EntitySystem & *ents*, tdt::uint *id*)

Returns the task type of a given task.

Parameters

Reference	to the entity system containing components.
Task	ID.

Definition at line 44 of file TaskHelper.cpp.

5.36.2.7 bool TaskHelper::is_complete (EntitySystem & *ents,* tdt::uint *id*)

Returns true if a given task is complete, false otherwise.

Parameters

Reference	to the entity system containing components.
ID	of the task.

Definition at line 122 of file TaskHelper.cpp.

5.36.2.8 void TaskHelper::set_complete (EntitySystem & *ents,* tdt::uint *id*)

Sets a given task to a complete state.

Parameters

Reference	to the entity system containing components.
ID	of the task.

Definition at line 115 of file TaskHelper.cpp.

5.36.2.9 void TaskHelper::set_task_source (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *source*)

Sets the source of a given task (that is, the entity that is completing the task).

Parameters

Reference	to the entity system containing components.
Task	ID.
Source	ID.

Definition at line 5 of file TaskHelper.cpp.

5.36.2.10 void TaskHelper::set_task_target (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *target*)

Sets the target entity of a given task.

Parameters

Reference	to the entity system containing components.
Task	ID.
Target	ID.

Definition at line 21 of file TaskHelper.cpp.

5.36.2.11 void TaskHelper::set_task_type (EntitySystem & *ents,* tdt::uint *id,* TASK_TYPE *type*)

Sets the task type of a given task.

Parameters

Reference	to the entity system containing components.
Task	ID.
The	new task type.

Definition at line 37 of file TaskHelper.cpp.

5.37 tdt Namespace Reference

Namespace containing numeric types used in the game.

Typedefs

- using **uint** = std::size_t
- using **real** = Ogre::Real

5.37.1 Detailed Description

Namespace containing numeric types used in the game.

5.38 TimeHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the time component.

Functions

- tdt::real get_curr_time (EntitySystem &, tdt::uint)

  Returns the time that has passed since the timer started.
- void advance_curr_time (EntitySystem &, tdt::uint, tdt::real)

  Adds a given time value to a given timer.
- void max_curr_time (EntitySystem &, tdt::uint)

  Completes a given timer (by maxing it's current time).
- void set_time_limit (EntitySystem &, tdt::uint, tdt::real)

  Sets the time a given timer requires for completion.
- tdt::real get_time_limit (EntitySystem &, tdt::uint)

  Returns the time a given timer requires for completion.
- void set_target (EntitySystem &, tdt::uint, tdt::uint)

  Sets the ID of the event a given timer starts/ends.
- tdt::uint get_target (EntitySystem &, tdt::uint)

  Returns the ID of the event a given timer starts/ends.
- void set_type (EntitySystem &, tdt::uint, TIME_EVENT)

  Sets the type of a given timer.
- TIME_EVENT get_type (EntitySystem &, tdt::uint)

  Returns the type of a given timer.

5.38.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the time component.

5.38.2 Function Documentation

5.38.2.1 void TimeHelper::advance_curr_time (**EntitySystem** & *ents,* tdt::uint *id,* tdt::real *val*)

Adds a given time value to a given timer.

Parameters

Entity	system which contains the timer.
ID	of the timer.
Time	to add.

Definition at line 14 of file TimeHelper.cpp.

5.38.2.2 tdt::real TimeHelper::get_curr_time (**EntitySystem** & *ents,* tdt::uint *id*)

Returns the time that has passed since the timer started.

Parameters

Entity	system which contains the timer.
ID	of th entity.

Definition at line 5 of file TimeHelper.cpp.

5.38.2.3 tdt::uint TimeHelper::get_target (EntitySystem & *ents*, tdt::uint *id*)

Returns the ID of the event a given timer starts/ends.

Parameters

Entity	system which contains the timer.
ID	of the timer.

Definition at line 51 of file TimeHelper.cpp.

5.38.2.4 tdt::real TimeHelper::get_time_limit (EntitySystem & *ents*, tdt::uint *id*)

Returns the time a given timer requires for completion.

Parameters

Entity	system which contains the timer.
ID	of the timer.

Definition at line 35 of file TimeHelper.cpp.

5.38.2.5 TIME_EVENT TimeHelper::get_type (EntitySystem & *ents*, tdt::uint *id*)

Returns the type of a given timer.

Parameters

Entity	system which contains the timer.
ID	of the timer.

Definition at line 67 of file TimeHelper.cpp.

5.38.2.6 void TimeHelper::max_curr_time (EntitySystem & *ents*, tdt::uint *id*)

Completes a given timer (by maxing it's current time).

Parameters

Entity	system which contains the timer.
ID	of the timer.

Definition at line 21 of file TimeHelper.cpp.

5.38.2.7 void TimeHelper::set_target (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *val*)

Sets the ID of the event a given timer starts/ends.

Parameters

Entity	system which contains the timer.
ID	of the timer.
ID	of the event.

Definition at line 44 of file TimeHelper.cpp.

5.38.2.8 void TimeHelper::set_time_limit (**EntitySystem** & *ents*, tdt::uint *id*, tdt::real *val*)

Sets the time a given timer requires for completion.

Parameters

Entity	system which contains the timer.
ID	of the timer.
The	new time limit.

Definition at line 28 of file TimeHelper.cpp.

5.38.2.9 void TimeHelper::set_type (**EntitySystem** & *ents*, tdt::uint *id*, TIME_EVENT *val*)

Sets the type of a given timer.

Parameters

Entity	system which contains the timer.
ID	of the timer.
The	new type.

Definition at line 60 of file TimeHelper.cpp.

5.39 TriggerHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the trigger component.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)
  Sets the blueprint table used used to handle triggering of a given entity.

- const std::string & get_blueprint (EntitySystem &, tdt::uint)

  Returns the blueprint table used used to handle triggering of a given entity.
- void set_linked_entity (EntitySystem &, tdt::uint, tdt::uint)
- tdt::uint get_linked_entity (EntitySystem &, tdt::uint)
- void set_cooldown (EntitySystem &, tdt::uint, tdt::real)

  Sets the cooldown before an entity can be triggered again.
- tdt::real get_cooldown (EntitySystem &, tdt::uint)

  Returns the cooldown before an entity can be triggered again.
- void trigger (EntitySystem &, tdt::uint, tdt::uint)

  Triggers an entity.
- bool can_be_triggered_by (EntitySystem &, tdt::uint, tdt::uint)

  Returns true if a given entity can be triggered by another given entity.
- void reset_timer (EntitySystem &, tdt::uint)

  Sets the trigger cooldown timer of a given entity to zero.
- void set_radius (EntitySystem &, tdt::uint, tdt::real)

  Sets the trigger radius of a given entity.
- tdt::real get_radius (EntitySystem &, tdt::uint)

  Returns the trigger radius of a given entity.

5.39.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the trigger component.

5.39.2 Function Documentation

5.39.2.1 bool TriggerHelper::can_be_triggered_by (**EntitySystem** & *ents*, tdt::uint *id*, tdt::uint *target*)

Returns true if a given entity can be triggered by another given entity.

Parameters

Entity	system containing the entity.
ID	of the entity that is supposed to get triggered.
ID	of the triggering entity.

Definition at line 61 of file TriggerHelper.cpp.

5.39.2.2 const std::string & TriggerHelper::get_blueprint (**EntitySystem** & *ents*, tdt::uint *id*)

Returns the blueprint table used used to handle triggering of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 13 of file TriggerHelper.cpp.

5.39.2.3 tdt::real TriggerHelper::get_cooldown (EntitySystem & *ents*, tdt::uint *id*)

Returns the cooldown before an entity can be triggered again.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 45 of file TriggerHelper.cpp.

5.39.2.4 tdt::uint TriggerHelper::get_linked_entity (EntitySystem & *ents*, tdt::uint *id*)

Parameters

Returns	the ID of the entity a given trigger entity is linked to.
Entity	system containing the entity.
ID	of the trigger entity.

Definition at line 29 of file TriggerHelper.cpp.

5.39.2.5 tdt::real TriggerHelper::get_radius (EntitySystem & *ents*, tdt::uint *id*)

Returns the trigger radius of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 91 of file TriggerHelper.cpp.

5.39.2.6 void TriggerHelper::reset_timer (EntitySystem & *ents*, tdt::uint *id*)

Sets the trigger cooldown timer of a given entity to zero.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 77 of file TriggerHelper.cpp.

5.39.2.7 void TriggerHelper::set_blueprint (**EntitySystem** & *ents,* tdt::uint *id,* const std::string & *val*)

Sets the blueprint table used used to handle triggering of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new blueprint name.

Definition at line 6 of file TriggerHelper.cpp.

5.39.2.8 void TriggerHelper::set_cooldown (**EntitySystem** & *ents,* tdt::uint *id,* tdt::real *val*)

Sets the cooldown before an entity can be triggered again.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 38 of file TriggerHelper.cpp.

5.39.2.9 void TriggerHelper::set_linked_entity (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *val*)

Parameters

Sets	the linked entity a given trigger entity is linked to.
Entity	system containing the entity.
ID	of the trigger entity.
ID	of the linked entity.

Definition at line 22 of file TriggerHelper.cpp.

5.39.2.10 void TriggerHelper::set_radius (**EntitySystem** & *ents,* tdt::uint *id,* tdt::real *val*)

Sets the trigger radius of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 84 of file TriggerHelper.cpp.

5.39.2.11 void TriggerHelper::trigger (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *target*)

Triggers an entity.

Parameters

Entity	system containing the entity.
ID	of the triggered entity.
ID	of the entity that triggered the triggered entity.

Definition at line 54 of file TriggerHelper.cpp.

5.40 UpgradeHelper Namespace Reference

Auxiliary namespace containing functions that help with the management of the upgrade component.

Functions

- void set_blueprint (EntitySystem &, tdt::uint, const std::string &)
  Sets the blueprint table that handles the upgrading of a given entity.
- const std::string & get_blueprint (EntitySystem &, tdt::uint)
  Returns the blueprint table that handles the upgrading of a given entity.
- tdt::uint set_experience (EntitySystem &, tdt::uint, tdt::uint)
  Sets the amount of experience a given entity has.
- tdt::uint get_experience (EntitySystem &, tdt::uint)
  Returns the amount of experience a given entity has.
- tdt::uint add_experience (EntitySystem &, tdt::uint, tdt::uint)
  Adds a given amount of experience to a given entity.
- void set_exp_needed (EntitySystem &, tdt::uint, tdt::uint)
  Sets the amount of experience needed for next level of a given entity.
- tdt::uint get_exp_needed (EntitySystem &, tdt::uint)
  Returns the amount of experience needed for next level of a given entity.
- void set_level (EntitySystem &, tdt::uint, tdt::uint)
  Sets the level of a given entity.
- tdt::uint get_level (EntitySystem &, tdt::uint)
  Returns the level of a given entity.
- void set_level_cap (EntitySystem &, tdt::uint, tdt::uint)
  Sets the maximum level a given entity can reach.
- tdt::uint get_level_cap (EntitySystem &, tdt::uint)
  Returns the maximum level a given entity can reach.
- bool can_level_up (EntitySystem &, tdt::uint)
  Returns true if a given entity can level up.
- void upgrade (EntitySystem &, tdt::uint)
  Upgrades a given entity that can level up.

5.40.1 Detailed Description

Auxiliary namespace containing functions that help with the management of the upgrade component.

5.40.2 Function Documentation

5.40.2.1 tdt::uint UpgradeHelper::add_experience (EntitySystem & *ents*, tdt::uint *id*, tdt::uint *val*)

Adds a given amount of experience to a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	amount of experience to be added.

Definition at line 59 of file UpgradeHelper.cpp.

5.40.2.2 bool UpgradeHelper::can_level_up (EntitySystem & *ents*, tdt::uint *id*)

Returns true if a given entity can level up.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 159 of file UpgradeHelper.cpp.

5.40.2.3 const std::string & UpgradeHelper::get_blueprint (EntitySystem & *ents*, tdt::uint *id*)

Returns the blueprint table that handles the upgrading of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 14 of file UpgradeHelper.cpp.

5.40.2.4 tdt::uint UpgradeHelper::get_exp_needed (EntitySystem & *ents*, tdt::uint *id*)

Returns the amount of experience needed for next level of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 106 of file UpgradeHelper.cpp.

5.40.2.5 tdt::uint UpgradeHelper::get_experience (EntitySystem & *ents*, tdt::uint *id*)

Returns the amount of experience a given entity has.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 50 of file UpgradeHelper.cpp.

5.40.2.6 tdt::uint UpgradeHelper::get_level (EntitySystem & *ents*, tdt::uint *id*)

Returns the level of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 128 of file UpgradeHelper.cpp.

5.40.2.7 tdt::uint UpgradeHelper::get_level_cap (EntitySystem & *ents*, tdt::uint *id*)

Returns the maximum level a given entity can reach.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 150 of file UpgradeHelper.cpp.

5.40.2.8 void UpgradeHelper::set_blueprint (EntitySystem & *ents*, tdt::uint *id*, const std::string & *val*)

Sets the blueprint table that handles the upgrading of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new blueprint name.

Definition at line 7 of file UpgradeHelper.cpp.

5.40.2.9 void UpgradeHelper::set_exp_needed (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the amount of experience needed for next level of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new experience amount needed.

Definition at line 86 of file UpgradeHelper.cpp.

5.40.2.10 tdt::uint UpgradeHelper::set_experience (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the amount of experience a given entity has.

Won't allow more experience than is needed for the next level and returns the remaining experience not added.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new experience amount.

Definition at line 23 of file UpgradeHelper.cpp.

5.40.2.11 void UpgradeHelper::set_level (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the level of a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new level.

Note

Does not change the attributes of the entity!

Definition at line 115 of file UpgradeHelper.cpp.

5.40.2.12 void UpgradeHelper::set_level_cap (EntitySystem & *ents,* tdt::uint *id,* tdt::uint *val*)

Sets the maximum level a given entity can reach.

Parameters

Entity	system containing the entity.
ID	of the entity.
The	new max level.

Definition at line 137 of file UpgradeHelper.cpp.

5.40.2.13 void UpgradeHelper::upgrade (EntitySystem & *ents*, tdt::uint *id*)

Upgrades a given entity that can level up.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 168 of file UpgradeHelper.cpp.

5.41 util Namespace Reference

The util namespace contains functors used as conditions in searches and other helper structures/functions used throughout the code.

Namespaces

- effect

  Contains effect functors that perform an action on the entity they are called on.
- heuristic

  Forward declarations.
- path_type

  Contains different path types, which are used to check if a path should be returned once found or when an augmenting edge is found to the path.
- pathfinding

  Contains the pathfinding algorithms used by the util::pathfind function.

Classes

- class EntityDestroyer

  A structure providing the private method EntitySystem::destroy_entity to the DestructorHelper::destroy function.
- struct HAS_GOLD

  Tests if a given entity has a gold component.
- struct IS_ENEMY

  Tests if if the entity it is called on is an enemy of the entity specified in it's constructor.
- struct IS_FRIENDLY

  Tests if if the entity it is called on is a friend of the entity specified in it's constructor.
- struct IS_FRIENDLY_OR_NEUTRAL

  Tests if if the entity it is called on is a friend of or neutral to the entity specified in it's constructor.
- struct IS_GOLD_VAULT

  Tests if a given entity is of friendly faction, has structure component and has gold component (that is, it's a gold vault).

Typedefs

- using DEFAULT_PATH_TYPE = path_type::FIRST_PATH

  Default types for the different pathfinding functors.
- using **DEFAULT_HEURISTIC** = heuristic::PORTAL_HEURISTIC
- using **DEFAULT_PATHFINDING_ALGORITHM** = pathfinding::A_STAR< DEFAULT_PATH_TYPE >

Functions

- template<typename ALGORITHM = util::DEFAULT_PATHFINDING_ALGORITHM>
  bool pathfind (EntitySystem &ents, tdt::uint id, tdt::uint target, util::heuristic::HEURISTIC &heuristic, bool add_path=true, bool allow_destruction=true)

  Finds a path using a given algorithm (specified as a template parameter) and heuristic and adds the path to the pathfinding entity if needed.
- int get_enum_direction (EntitySystem &, tdt::uint, tdt::uint)

  Returns the direction from a given entity to another given entity in the form of the direction enum (8 directional).
- tdt::uint get_random (tdt::uint, tdt::uint)

  Returns a random number within a given range.
- tdt::uint abs (int)

  Returns the absolute value of a given integer.

5.41.1 Detailed Description

The util namespace contains functors used as conditions in searches and other helper structures/functions used throughout the code.

The utim namespace contains various tools and utilities used by the game's engine.

Util namespace contains general tools and utilities used by the game's engine.

This part of the namespace contains functions used for pathfinding.

This part contains the pathfinding algorithms, heuristics and path types. Also contains the typedefs for DETAUL↩T_PATH_TYPE, DEFAULT_PATHFINDING_ALGORITHM and DEFAULT_HEURISTIC.

5.41.2 Typedef Documentation

5.41.2.1 using **util::DEFAULT_PATH_TYPE** = typedef **path_type::FIRST_PATH**

Default types for the different pathfinding functors.

Definition at line 339 of file PathfindingAlgorithms.hpp.

5.41.3 Function Documentation

5.41.3.1 tdt::uint util::abs (int *val*)

Returns the absolute value of a given integer.

Parameters

The	number we want absolute value of.

Definition at line 102 of file Util.cpp.

5.41.3.2 int util::get_enum_direction (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *target*)

Returns the direction from a given entity to another given entity in the form of the direction enum (8 directional).

Parameters

EntitySystem	containing both entities.
ID	of the first entity.
ID	of the second entity.

Note

The direction is #1 -> #2.

Definition at line 54 of file Util.cpp.

5.41.3.3 tdt::uint util::get_random (tdt::uint *min,* tdt::uint *max*)

Returns a random number within a given range.

Parameters

Lower	bound of the range.
Upper	bound of the range.

Definition at line 93 of file Util.cpp.

5.41.3.4 template<typename ALGORITHM = util::DEFAULT_PATHFINDING_ALGORITHM> bool util::pathfind (**EntitySystem** & *ents,* tdt::uint *id,* tdt::uint *target,* **util::heuristic::HEURISTIC** & *heuristic,* bool *add_path* = `true`, bool *allow_destruction* = `true`)

Finds a path using a given algorithm (specified as a template parameter) and heuristic and adds the path to the pathfinding entity if needed.

Parameters

Entity	system containing the entity and the pathfinding grid.
ID	of the pathfinding entity.
Target	of the pathfinding.
Heuristic	used by the pathfinding algorithm.
If	true, the path will be added to the pathdinding entity's pathfinding component.
If	true, the entity will be allowed to destroy blocks on it's way.

Definition at line 26 of file Pathfinding.hpp.

5.42 util::effect Namespace Reference

Contains effect functors that perform an action on the entity they are called on.

Classes

- struct DAMAGE_EFFECT

  Deals a random damage in a given range to the entity it's called on.
- struct FREEZE_EFFECT

  Freezes a given entity in place for a given time period.
- struct HEAL_EFFECT

  Fully heals the entity it's called on.
- struct LOWER_SPEED_EFFECT

  Halves the speed of the entity it's called on for a given time period.

5.42.1 Detailed Description

Contains effect functors that perform an action on the entity they are called on.

5.43 util::heuristic Namespace Reference

Forward declarations.

Classes

- struct HEURISTIC

  Abstract parent of all heuristics.
- struct MANHATTAN_DISTANCE

  Returns the manhattan distance between two nodes.
- struct NO_HEURISTIC

  Represents no heuristic by returning 0 all the time.
- struct PORTAL_HEURISTIC

  Variation of the Manhattan distance heuristic that takes portals into accounts.
- struct RUN_AWAY_HEURISTIC

  Used by entities that want to run away from an enemy.

5.43.1 Detailed Description

Forward declarations.

Contains the heuristics used by the pathfinding algorithms.

5.44 util::path_type Namespace Reference

Contains different path types, which are used to check if a path should be returned once found or when an augmenting edge is found to the path.

Classes

- struct BEST_PATH

    Finds the best path by refusing any paths found.
- struct FIRST_PATH

    Finds the first path by accepting the first path found.
- struct RANDOM_PATH

    Finds a random path by returning true only when a random number in the range (0, UPPER) is equal to 0.

5.44.1 Detailed Description

Contains different path types, which are used to check if a path should be returned once found or when an augmenting edge is found to the path.

5.45 util::pathfinding Namespace Reference

Contains the pathfinding algorithms used by the util::pathfind function.

Classes

- struct A_STAR

    Simple A$*$ pathfinding implementations with path type specified as a template parameter.

5.45.1 Detailed Description

Contains the pathfinding algorithms used by the util::pathfind function.

Chapter 6

Class Documentation

6.1 util::pathfinding::A_STAR< PATH_TYPE > Struct Template Reference

Simple A∗ pathfinding implementations with path type specified as a template parameter.

```
#include <PathfindingAlgorithms.hpp>
```

Static Public Member Functions

- static std::deque< tdt::uint > **get_path** (EntitySystem &ents, tdt::uint id, tdt::uint start, tdt::uint end, util↩ ::heuristic::HEURISTIC &heuristic, bool allow_destruction=true)

6.1.1 Detailed Description

template<typename PATH_TYPE = util::DEFAULT_PATH_TYPE>
struct util::pathfinding::A_STAR< PATH_TYPE >

Simple A∗ pathfinding implementations with path type specified as a template parameter.

Parameters

Entity	system containing the pathfinding entity and the grid.
ID	of the pathfinding entity.
ID	of the starting node.
ID	of the ending node.
Heuristic	to be used.
If	true, the entity will be allowed to destroy blocks along it's way.

Definition at line 42 of file PathfindingAlgorithms.hpp.

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

- tools/PathfindingAlgorithms.hpp

6.2 AIComponent Struct Reference

Holds info about the base Lua table to be called for initializing, updating and finnishing an entity as well as couple categorizing enums.

```
#include <Components.hpp>
```

Public Member Functions

- **AIComponent** (std::string &&s="ERROR", ENTITY_STATE::VAL st=ENTITY_STATE::NORMAL)
- **AIComponent** (const AIComponent &)=default
- **AIComponent** (AIComponent &&)=default
- AIComponent & **operator=** (const AIComponent &)=default
- AIComponent & **operator=** (AIComponent &&)=default

Public Attributes

- std::string **blueprint**
- ENTITY_STATE::VAL **state**

Static Public Attributes

- static constexpr int **type** = 2

6.2.1 Detailed Description

Holds info about the base Lua table to be called for initializing, updating and finnishing an entity as well as couple categorizing enums.

Definition at line 80 of file Components.hpp.

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

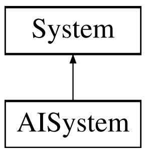
- Components.hpp

6.3 AISystem Class Reference

System handling the AI of entities by calling their update method every frame.

```
#include <AISystem.hpp>
```

Inheritance diagram for AISystem:



Public Member Functions

- AISystem (EntitySystem &)

  Constructor.
- ∼AISystem ()=default

  Destructor.
- void update (tdt::real) override

  Updates all valid entities by calling their update function stored in the AIComponent::blueprint table.
- void set_update_period (tdt::real)

  Sets the amount of seconds it takes before the next AI update will be performed.
- tdt::real get_update_period () const

  Returns the amount of seconds it takes before the next AI update will be performed.
- void force_update ()

  Sets the update timer equal to the period and thus forcing all entities' AI to be updated on next AISystem::update call.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- tdt::real update_timer_

  Used to track the time and check if the entities should be updated.
- tdt::real **update_period_**

6.3.1 Detailed Description

System handling the AI of entities by calling their update method every frame.

Definition at line 10 of file AISystem.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 AISystem::AISystem (EntitySystem & *ent*)

Constructor.

Parameters

Reference	to the game's entity system.

Definition at line 6 of file AISystem.cpp.

6.3.2.2 AISystem::∼AISystem ( ) [default]

Destructor.

6.3.3 Member Function Documentation

6.3.3.1 void AISystem::force_update ( )

Sets the update timer equal to the period and thus forcing all entities' AI to be updated on next AISystem::update call.

Definition at line 42 of file AISystem.cpp.

6.3.3.2 tdt::real AISystem::get_update_period ( ) const

Returns the amount of seconds it takes before the next AI update will be performed.

Definition at line 37 of file AISystem.cpp.

6.3.3.3 void AISystem::set_update_period (tdt::real *val*)

Sets the amount of seconds it takes before the next AI update will be performed.

Parameters

Update	period time (in seconds).

Definition at line 32 of file AISystem.cpp.

6.3.3.4 void AISystem::update (tdt::real *delta*) `[override]`, `[virtual]`

Updates all valid entities by calling their update function stored in the AIComponent::blueprint table.

Parameters

Time	since the last frame.

Implements System.

Definition at line 10 of file AISystem.cpp.

6.3.4 Member Data Documentation

6.3.4.1 EntitySystem& AISystem::entities_ `[private]`

Reference to the game's entity system.

Definition at line 54 of file AISystem.hpp.

6.3.4.2 tdt::real AISystem::update_timer_ `[private]`

Used to track the time and check if the entities should be updated.

Definition at line 59 of file AISystem.hpp.

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

- systems/AISystem.hpp
- systems/AISystem.cpp

6.4 AlignComponent Struct Reference

Holds information about an objects align states, i.e.

`#include <Components.hpp>`

Classes

- struct AlignState

Public Member Functions

- **AlignComponent** (const AlignComponent &)=default
- **AlignComponent** (AlignComponent &&)=default
- AlignComponent & **operator=** (const AlignComponent &)=default
- AlignComponent & **operator=** (AlignComponent &&)=default

Public Attributes

- std::array< AlignState, state_count > **states**

Static Public Attributes

- static constexpr int **type** = 24
- static constexpr int **state_count** = 5

6.4.1 Detailed Description

Holds information about an objects align states, i.e.

scale, model, etc for the different alignments of blocks (e.g. walls).

Definition at line 588 of file Components.hpp.

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

- Components.hpp

6.5 AlignComponent::AlignState Struct Reference

Public Attributes

- Ogre::Vector3 **scale**
- Ogre::Vector3 **position_offset**
- std::string **mesh**
- std::string **material**

6.5.1 Detailed Description

Definition at line 593 of file Components.hpp.

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

- Components.hpp

6.6 util::path_type::BEST_PATH Struct Reference

Finds the best path by refusing any paths found.

`#include <PathfindingAlgorithms.hpp>`

Static Public Member Functions

- static bool **return_path** ()

6.6.1 Detailed Description

Finds the best path by refusing any paths found.

Definition at line 143 of file PathfindingAlgorithms.hpp.

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

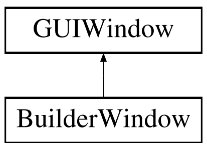
- tools/PathfindingAlgorithms.hpp

6.7 BuilderWindow Class Reference

Class representing the building selection window, allows the player to place registered (unlocked) buildings.

`#include <BuilderWindow.hpp>`

Inheritance diagram for BuilderWindow:



Public Member Functions

- BuilderWindow ()

  Constructor.
- ∼BuilderWindow ()=default

  Destructor.
- void register_building (const std::string &)

  Appends a table name to the vector of all building tables.
- void set_placer (EntityPlacer ∗)

  Sets the placer that is used to build.
- const std::vector< std::string > & get_buildings () const

  Returns a vector containing all the names of the registered buildings.
- void clear_buildings ()

  Removes all unlocked buildings.
- void build (int)

  Places the building on a given position.
- void dec_selection ()

  Decrements selection_number_ by one and updates the window.
- void inc_selection ()

  Increments selection_number_ by one and updates the window.

Protected Member Functions

- void init_ () override

  Initializes the window and subscribes events.

Private Member Functions

- const std::string & get_building_ (std::size_t)

  Range checked buildings_ index access, returns the name of the building at a given index or "UNKNOWN" if the index is out of bounds.
- void update_selection_ ()

  Updates building names on the buttons.

Private Attributes

- std::vector< std::string > buildings_

  Names of all registered buildings.
- std::size_t selection_number_

  Number of the current rightmost selection.
- EntityPlacer ∗ placer_

  Placer used to build.

Additional Inherited Members

6.7.1 Detailed Description

Class representing the building selection window, allows the player to place registered (unlocked) buildings.

Definition at line 12 of file BuilderWindow.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 BuilderWindow::BuilderWindow ( )

Constructor.

Definition at line 5 of file BuilderWindow.cpp.

6.7.2.2 BuilderWindow::∼BuilderWindow ( ) [default]

Destructor.

6.7.3 Member Function Documentation

6.7.3.1 void BuilderWindow::build (int *build_num*)

Places the building on a given position.

Parameters

The	button position. (1-4)

Definition at line 37 of file BuilderWindow.cpp.

6.7.3.2 void BuilderWindow::clear_buildings ( )

Removes all unlocked buildings.

Definition at line 30 of file BuilderWindow.cpp.

6.7.3.3 void BuilderWindow::dec_selection ( )

Decrements selection_number_ by one and updates the window.

Definition at line 101 of file BuilderWindow.cpp.

6.7.3.4 const std::string & BuilderWindow::get_building_ (std::size_t *index*) [private]

Range checked buildings_ index access, returns the name of the building at a given index or "UNKNOWN" if the index is out of bounds.

Parameters

Index	of the building in the buildings_ vector.

Definition at line 119 of file BuilderWindow.cpp.

6.7.3.5 const std::vector< std::string > & BuilderWindow::get_buildings ( ) const

Returns a vector containing all the names of the registered buildings.

(Used for serialization.)

Definition at line 25 of file BuilderWindow.cpp.

6.7.3.6 void BuilderWindow::inc_selection ( )

Increments selection_number_ by one and updates the window.

Definition at line 110 of file BuilderWindow.cpp.

6.7.3.7 void BuilderWindow::init_ ( ) `[override]`, `[protected]`, `[virtual]`

Initializes the window and subscribes events.

Implements GUIWindow.

Definition at line 48 of file BuilderWindow.cpp.

6.7.3.8 void BuilderWindow::register_building (const std::string & *tname*)

Appends a table name to the vector of all building tables.

Parameters

Name	of the table to register.

Definition at line 9 of file BuilderWindow.cpp.

6.7.3.9 void BuilderWindow::set_placer (EntityPlacer ∗ *p*)

Sets the placer that is used to build.

Parameters

The	new entity placer.

Definition at line 20 of file BuilderWindow.cpp.

6.7.3.10 void BuilderWindow::update_selection_ ( ) [private]

Updates building names on the buttons.

Definition at line 129 of file BuilderWindow.cpp.

6.7.4 Member Data Documentation

6.7.4.1 std::vector<std::string> BuilderWindow::buildings_ [private]

Names of all registered buildings.

Definition at line 89 of file BuilderWindow.hpp.

6.7.4.2 EntityPlacer* BuilderWindow::placer_ [private]

Placer used to build.

Definition at line 100 of file BuilderWindow.hpp.

6.7.4.3 std::size_t BuilderWindow::selection_number_ [private]

Number of the current rightmost selection.

The window shows buildings with indices <selection_number_ - 3, selection_number_>.

Definition at line 95 of file BuilderWindow.hpp.

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

- gui/BuilderWindow.hpp
- gui/BuilderWindow.cpp

6.8 Camera Class Reference

Class wrapping the Ogre camera object, allowing RTS-like movement and switching to free mode.

```
#include <Camera.hpp>
```

Public Member Functions

- Camera ()=default

    Constructor.
- ∼Camera ()=default

    Destructor.
- void init (Ogre::Camera ∗)

    Initializes the wrapper with a camera object.
- void set_position (const Ogre::Vector2 &)

    Sets the 2D position of the camera (X and Z axes).
- const Ogre::Vector3 & get_position () const

    Returns the 3D position of the camera (including height).
- void set_direction (const Ogre::Vector3 &)

    Changes the direction the camera is facing.
- const Ogre::Vector3 & get_direction () const

    Returns the direction the camera is facing.
- void look_at (const Ogre::Vector2 &)

    Makes the camera to look at a point on the ground.
- void reset ()

    Resets the camera's position and orientation.
- void set_start (const Ogre::Vector2 &, const Ogre::Vector2 &, tdt::real)

    Sets the starting stats of the camera.
- void set_free_mode (bool)

    Changes the movement mode of the camera.
- bool get_free_mode () const

    Returns true if the camera is in free mode, false otherwise.
- void update (tdt::real)

    Updates the movement of the camera.
- void key_pressed (CEGUI::Key::Scan)

    Moves the camera if a movement key was pressed.
- void key_released (CEGUI::Key::Scan)

    Moves the camera if a movement key was released.
- void move (DIRECTION::VAL, tdt::real)

    Moves the camera in a given direction, used for mouse movement.
- void set_height (tdt::real)

    Changes the height of the camera.
- tdt::real get_height () const

    Returns the height of the camera.
- void pitch (const Ogre::Degree &)

    Rotates the camera around the side-to-side axis.
- void yaw (const Ogre::Degree &)

    Rotates the camera around the vertical axis.

Private Attributes

- Ogre::Camera ∗ camera_

    Ogre camera that is wrapped.
- std::tuple< Ogre::Vector3, Ogre::Vector3 > start_

    Starting stats of the camera, used when resetting.
- bool free_mode_

Determines the mode of the camera, if true, it can fly around the level, if false, it can move in an RTS-like fashion.

- Ogre::Vector3 movement_direction_

  Direction vector of the free mode movement.
- tdt::real speed_

  Speed modifier of the camera.
- tdt::real height_

  Y axis the camera is locked at when free mode is disabled.

Friends

- class **GameSerializer**
- class **Game**

6.8.1 Detailed Description

Class wrapping the Ogre camera object, allowing RTS-like movement and switching to free mode.

Definition at line 13 of file Camera.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Camera::Camera ( ) [default]

Constructor.

6.8.2.2 Camera::∼Camera ( ) [default]

Destructor.

6.8.3 Member Function Documentation

6.8.3.1 const Ogre::Vector3 & Camera::get_direction ( ) const

Returns the direction the camera is facing.

Definition at line 27 of file Camera.cpp.

6.8.3.2 bool Camera::get_free_mode ( ) const

Returns true if the camera is in free mode, false otherwise.

Definition at line 60 of file Camera.cpp.

6.8.3.3 tdt::real Camera::get_height ( ) const

Returns the height of the camera.

Definition at line 163 of file Camera.cpp.

6.8.3.4 const Ogre::Vector3 & Camera::get_position ( ) const

Returns the 3D position of the camera (including height).

Definition at line 16 of file Camera.cpp.

6.8.3.5 void Camera::init (Ogre::Camera * *cam*)

Initializes the wrapper with a camera object.

Parameters

Camera	to be wrapped.

Definition at line 4 of file Camera.cpp.

6.8.3.6 void Camera::key_pressed (CEGUI::Key::Scan *key*)

Moves the camera if a movement key was pressed.

Parameters

Pressed	key.

Definition at line 82 of file Camera.cpp.

6.8.3.7 void Camera::key_released (CEGUI::Key::Scan *key*)

Moves the camera if a movement key was released.

Parameters

Released	key.

Definition at line 107 of file Camera.cpp.

6.8.3.8 void Camera::look_at (const Ogre::Vector2 & *val*)

Makes the camera to look at a point on the ground.

Parameters

2D	location of the point (X and Z axes).

Definition at line 32 of file Camera.cpp.

6.8.3.9 void Camera::move (DIRECTION::VAL *dir,* tdt::real *delta*)

Moves the camera in a given direction, used for mouse movement.

Parameters

Direction	of the movement.
Time	since the last frame.

Definition at line 132 of file Camera.cpp.

6.8.3.10 void Camera::pitch (const Ogre::Degree & *val*)

Rotates the camera around the side-to-side axis.

Parameters

Amount	of degrees to rotate by.

Definition at line 168 of file Camera.cpp.

6.8.3.11 void Camera::reset ( )

Resets the camera's position and orientation.

Definition at line 38 of file Camera.cpp.

6.8.3.12 void Camera::set_direction (const Ogre::Vector3 & *val*)

Changes the direction the camera is facing.

Parameters

The	new direction vector.

Definition at line 21 of file Camera.cpp.

6.8.3.13 void Camera::set_free_mode (bool *val*)

Changes the movement mode of the camera.

Parameters

If	true, the camera will be in free mode, otherwise it will return to RTS mode.

Definition at line 53 of file Camera.cpp.

6.8.3.14 void Camera::set_height (tdt::real *val*)

Changes the height of the camera.

Parameters

The	new height.

Definition at line 154 of file Camera.cpp.

6.8.3.15 void Camera::set_position (const Ogre::Vector2 & *val*)

Sets the 2D position of the camera (X and Z axes).

Parameters

The	new position.

Definition at line 10 of file Camera.cpp.

6.8.3.16 void Camera::set_start (const Ogre::Vector2 & *position,* const Ogre::Vector2 & *center,* tdt::real *height*)

Sets the starting stats of the camera.

(Used upon reset.)

Parameters

Starting	position.
Starting	point that camera is looking at.
Starting	height of the camera.

Definition at line 46 of file Camera.cpp.

6.8.3.17 void Camera::update (tdt::real *delta*)

Updates the movement of the camera.

Parameters

Time	since the last frame.

Definition at line 65 of file Camera.cpp.

6.8.3.18 void Camera::yaw (const Ogre::Degree & *val*)

Rotates the camera around the vertical axis.

Parameters

Amount	of degrees to rotate by.

Definition at line 173 of file Camera.cpp.

6.8.4 Member Data Documentation

6.8.4.1 Ogre::Camera∗ Camera::camera_ `[private]`

Ogre camera that is wrapped.

Definition at line 142 of file Camera.hpp.

6.8.4.2 bool Camera::free_mode_ `[private]`

Determines the mode of the camera, if true, it can fly around the level, if false, it can move in an RTS-like fashion.

Definition at line 154 of file Camera.hpp.

6.8.4.3 tdt::real Camera::height_ `[private]`

Y axis the camera is locked at when free mode is disabled.

Definition at line 170 of file Camera.hpp.

6.8.4.4 Ogre::Vector3 Camera::movement_direction_ `[private]`

Direction vector of the free mode movement.

Definition at line 159 of file Camera.hpp.

6.8.4.5 tdt::real Camera::speed_ [private]

Speed modifier of the camera.

Definition at line 164 of file Camera.hpp.

6.8.4.6 std::tuple<Ogre::Vector3, Ogre::Vector3> Camera::start_ [private]

Starting stats of the camera, used when resetting.

Definition at line 147 of file Camera.hpp.

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

- tools/Camera.hpp
- tools/Camera.cpp

6.9 CombatComponent Struct Reference

Holds info about an entity's attack types and damage.

`#include <Components.hpp>`

Public Member Functions

- **CombatComponent** (tdt::uint target=Component::NO_ENTITY, tdt::uint mi=0, tdt::uint ma=0, tdt::real cd=0, tdt::real r=0.f, int type=0, bool p=false, std::string &&proj="ERROR")
- **CombatComponent** (const CombatComponent &)=default
- **CombatComponent** (CombatComponent &&)=default
- CombatComponent & **operator=** (const CombatComponent &)=default
- CombatComponent & **operator=** (CombatComponent &&)=default

Public Attributes

- tdt::uint **curr_target**
- tdt::uint **min_dmg**
- tdt::uint **max_dmg**
- tdt::real **cd_time**
- tdt::real **cooldown**
- tdt::real **range**
- ATTACK_TYPE **atk_type**
- bool **pursue**
- std::string **projectile_blueprint**

Static Public Attributes

- static constexpr int **type** = 5

6.9.1 Detailed Description

Holds info about an entity's attack types and damage.

Definition at line 152 of file Components.hpp.

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

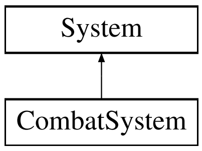
- Components.hpp

6.10 CombatSystem Class Reference

Manages auto attack melee and ranged combat, special melee and ranged attacks will be both handled by the spellcasting system.

```
#include <CombatSystem.hpp>
```

Inheritance diagram for CombatSystem:



Public Member Functions

- CombatSystem (EntitySystem &, Ogre::SceneManager &, GridSystem &)

  Constructor.
- ∼CombatSystem ()=default

  Destructor.
- void update (tdt::real) override

  Updates all auto attack combat in the game currently in progress.
- bool in_sight (tdt::uint, tdt::uint) const

  Returns true if two given entities can see each other, false otherwise.
- bool in_sight_wrt_BB (tdt::uint, tdt::uint) const

  Returns true if two given entities can see each other, false otherwise.
- tdt::uint get_closest_entity (tdt::uint, bool=true, bool=false) const

  Returns the ID of the closest entity (from a given entity's position), Component::NO_ENTITY otherwise.
- tdt::uint get_closest_structure (tdt::uint, bool=true, bool=false) const

  Returns the ID of the closest structure (from a given entity's position), Component::NO_ENTITY otherwise.
- tdt::uint get_closest_entity_thats_not (tdt::uint, tdt::uint, bool=true, bool=false) const

  Returns the ID of the closest entity (from a given entity's position) ignoring a given entity, Component::NO_ENTITY otherwise.
- tdt::uint get_closest_gold_deposit (tdt::uint, bool=false) const

  Returns the ID of the closest gold deposit (entity with both structure and gold components).
- tdt::uint get_closest_gold_vault (tdt::uint, bool=false, bool=false) const

  Returns the ID of the closest gold vault that can store player's gold.
- template<typename CONT , typename COND >
  tdt::uint get_closest_entity (tdt::uint id, COND &condition, bool only_sight=true) const

Returns the ID of the closest entity that has a given component, meets a given condition and is accessible.

- template<typename CONT , typename COND , typename EFFECT >
  void apply_effect_to_entities_in_range (tdt::uint id, COND &condition, EFFECT &effect, tdt::real range)

  Applies a given effect (functor given by the template argument EFFECT) to all entities conforming the condition (functor given by the template argument COND) in a given range from a given entity.
- void apply_heal_to_entities_in_range (tdt::uint, tdt::real)

  Heals all friendly entities within a given range from a given entity.
- void apply_damage_to_entities_in_range (tdt::uint, tdt::real, tdt::uint, tdt::uint)

  Damages all friendly entities within a given range from a given entity.
- void apply_slow_to_entities_in_range (tdt::uint, tdt::real, tdt::real)

  Slows all enemy entities within a given range from a given entity for a given time period.
- void apply_freeze_to_entities_in_range (tdt::uint, tdt::real, tdt::real)

  Freezes all enemy entities within a given range from a given entity for a given time period.
- void apply_slow_to (tdt::uint, tdt::real)

  Slows a given entity for a given time period.
- void apply_freeze_to (tdt::uint, tdt::real)

  Freezes a given entity for a given time period.
- void run_away_from (tdt::uint, tdt::uint, tdt::uint)

  Tries to find a path used by an entity to run away from another entity.
- void set_max_run_away_attempts (tdt::uint)

  Sets the maximum amount of pathfinding attempts for running away.
- tdt::uint get_max_run_away_attempts ()

  Returns the maximum amount of pathfinding attempts for running away.
- bool enemy_in_range (tdt::uint)

  Returns true if an enemy is in range from a given entity, false otherwise.
- template<>
  const std::map< tdt::uint, ALL_COMPONENTS > & get_container () const

  Specific case of the get_container method, which returns the map containing <ID, component bitset> map containing all entities.

Private Member Functions

- template<typename COMP >
  const std::map< tdt::uint, COMP > & get_container () const

  Retuns a map containing pairs of IDs and components of a given type, use the type ALL_COMPONENTS to get the <ID, component bitset> container.
- void create_homing_projectile (tdt::uint, CombatComponent &)

  Creates a new homing projectile at the position of a given entity homing at the entity's current target.
- void run_away_from_ (tdt::uint, tdt::uint, tdt::uint)

  Tries to find a path used by an entity to run away from another entity.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system (component retrieval).
- Ogre::RaySceneQuery & ray_query_

  Reference to the ray cast used to check if two entities can see each other.
- GridSystem & grid_

  Used to check if an entity is accessible.
- RayCaster ray_caster_

Used for polygon precise line of sight checking.

- tdt::uint max_run_away_attempts_ {10}

  Maximum amount of pathfindings performed when running away from an enemy.

- std::queue< std::tuple< tdt::uint, tdt::uint, tdt::uint > > run_away_queue_

  Used to distribute run away pathfindings over frames (otherwise something like a meteor falling on a big group of entities would freeze the game until all run away pathfindings are done.)

6.10.1 Detailed Description

Manages auto attack melee and ranged combat, special melee and ranged attacks will be both handled by the spellcasting system.

Definition at line 28 of file CombatSystem.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 CombatSystem::CombatSystem (EntitySystem & *ents*, Ogre::SceneManager & *scene*, GridSystem & *grid*)

Constructor.

Parameters

Reference	to the game's entity system (component retrieval).
Reference	to the main scene manager (ray casting).
Reference	to the game's grid system (accessibility).

Definition at line 11 of file CombatSystem.cpp.

6.10.2.2 CombatSystem::∼CombatSystem ( ) [default]

Destructor.

6.10.3 Member Function Documentation

6.10.3.1 void CombatSystem::apply_damage_to_entities_in_range (tdt::uint *id*, tdt::real *range*, tdt::uint *min*, tdt::uint *max*)

Damages all friendly entities within a given range from a given entity.

Parameters

ID	of the entity.
The	range.
Minimal	damage value.
Maximal	damage value.

Definition at line 268 of file CombatSystem.cpp.

6.10.3.2 template<typename CONT , typename COND , typename EFFECT > void CombatSystem::apply↩_effect_to_entities_in_range (tdt::uint *id*, COND & *condition*, EFFECT & *effect*, tdt::real *range*) [inline]

Applies a given effect (functor given by the template argument EFFECT) to all entities conforming the condition (functor given by the template argument COND) in a given range from a given entity.

Parameters

ID	of the entity.
Instance	of the condition functor.
Instance	of the effect functor.
The	radius.

Note

The explicit template specialization determines over which component container this method will iterate, use a component name for a specific components only or ALL_COMPONENTS for the component bitset map (which allows to iterate over all entitites regardless of their components).

Definition at line 170 of file CombatSystem.hpp.

6.10.3.3 void CombatSystem::apply_freeze_to (tdt::uint *id*, tdt::real *time*)

Freezes a given entity for a given time period.

Parameters

ID	of the entity.
The	time period for which the freeze is active.

Definition at line 295 of file CombatSystem.cpp.

6.10.3.4 void CombatSystem::apply_freeze_to_entities_in_range (tdt::uint *id*, tdt::real *range*, tdt::real *time*)

Freezes all enemy entities within a given range from a given entity for a given time period.

Parameters

ID	of the entity.
The	range.
The	time period for which the freeze is active.

Definition at line 282 of file CombatSystem.cpp.

6.10.3.5 void CombatSystem::apply_heal_to_entities_in_range (tdt::uint *id,* tdt::real *range*)

Heals all friendly entities within a given range from a given entity.

Parameters

ID	of the entity.
The	range.

Definition at line 261 of file CombatSystem.cpp.

6.10.3.6 void CombatSystem::apply_slow_to (tdt::uint *id,* tdt::real *time*)

Slows a given entity for a given time period.

Parameters

ID	of the entity.
The	time period for which the freeze is active.

Definition at line 289 of file CombatSystem.cpp.

6.10.3.7 void CombatSystem::apply_slow_to_entities_in_range (tdt::uint *id,* tdt::real *range,* tdt::real *time*)

Slows all enemy entities within a given range from a given entity for a given time period.

Parameters

ID	of the entity.
The	range.
The	time period for which the slow is active.

Definition at line 275 of file CombatSystem.cpp.

6.10.3.8 void CombatSystem::create_homing_projectile (tdt::uint *caster,* CombatComponent & *combat*) `[private]`

Creates a new homing projectile at the position of a given entity homing at the entity's current target.

Parameters

ID	of the caster entity.
Reference	to the caster entity's combat component.

Definition at line 387 of file CombatSystem.cpp.

6.10.3.9 bool CombatSystem::enemy_in_range (tdt::uint *id*)

Returns true if an enemy is in range from a given entity, false otherwise.

Parameters

ID	of the entity.

Definition at line 356 of file CombatSystem.cpp.

6.10.3.10 std::size_t CombatSystem::get_closest_entity (tdt::uint *id,* bool *only_sight* = `true`*,* bool *friendly* = `false`) const

Returns the ID of the closest entity (from a given entity's position), Component::NO_ENTITY otherwise.

Parameters

ID	of the entity from whose position the search is performed.
If	true, will return only entities in sight.
If	true, will return only friendly entities (enemies otherwise).

Definition at line 208 of file CombatSystem.cpp.

6.10.3.11 template<typename CONT , typename COND > tdt::uint CombatSystem::get_closest_entity (tdt::uint *id,* COND & *condition,* bool *only_sight* = `true`) const [inline]

Returns the ID of the closest entity that has a given component, meets a given condition and is accessible.

Parameters

ID	of the entity that is searching.
Functor	representing the condition.
If	true, only entities in sight get checked.

Note

The explicit template specialization determines over which component container this method will iterate, use a component name for a specific components only or ALL_COMPONENTS for the component bitset map (which allows to iterate over all entitites regardless of their components).

Definition at line 128 of file CombatSystem.hpp.

6.10.3.12 std::size_t CombatSystem::get_closest_entity_thats_not (tdt::uint *id,* tdt::uint *ignored,* bool *only_sight* = `true`*,* bool *friendly* = `false`) const

Returns the ID of the closest entity (from a given entity's position) ignoring a given entity, Component::NO_ENTITY otherwise.

Parameters

ID	of the entity from whose position the search is performed.
ID	of the entity that's ignored in the search.
If	true, will return only entities in sight.
If	true, will return only friendly entities (enemies otherwise).

Definition at line 224 of file CombatSystem.cpp.

6.10.3.13 std::size_t CombatSystem::get_closest_gold_deposit (tdt::uint *id*, bool *only_sight* = `false`) const

Returns the ID of the closest gold deposit (entity with both structure and gold components).

Parameters

ID	of the entity that looks for the gold deposit.
If	true, only deposits in sight will be checked.

Definition at line 238 of file CombatSystem.cpp.

6.10.3.14 std::size_t CombatSystem::get_closest_gold_vault (tdt::uint *id*, bool *only_sight* = `false`, bool *only_free* = `false`) const

Returns the ID of the closest gold vault that can store player's gold.

Parameters

ID	of the entity that looks for the gold vault.
If	true, only vaults in sight will be checked.
If	true, only vaults that have free space for more gold will be checked.

Definition at line 247 of file CombatSystem.cpp.

6.10.3.15 std::size_t CombatSystem::get_closest_structure (tdt::uint *id*, bool *only_sight* = `true`, bool *friendly* = `false`) const

Returns the ID of the closest structure (from a given entity's position), Component::NO_ENTITY otherwise.

Parameters

ID	of the entity from whose position the search is performed.
If	true, will return only structures in sight.
If	true, will return friendly structures (enemies otherwise).

Definition at line 216 of file CombatSystem.cpp.

6.10.3.16 template<typename COMP > const std::map<tdt::uint, COMP>& CombatSystem::get_container ( ) const [inline], [private]

Retuns a map containing pairs of IDs and components of a given type, use the type ALL_COMPONENTS to get the <ID, component bitset> container.

Definition at line 269 of file CombatSystem.hpp.

6.10.3.17 template<> const std::map<tdt::uint, ALL_COMPONENTS>& CombatSystem::get_container ( ) const [inline]

Specific case of the get_container method, which returns the map containing <ID, component bitset> map containing all entities.

Definition at line 331 of file CombatSystem.hpp.

6.10.3.18 std::size_t CombatSystem::get_max_run_away_attempts ( )

Returns the maximum amount of pathfinding attempts for running away.

Definition at line 351 of file CombatSystem.cpp.

6.10.3.19 bool CombatSystem::in_sight (tdt::uint *ent_id,* tdt::uint *target*) const

Returns true if two given entities can see each other, false otherwise.

Tests polygons.

Parameters

ID	of the first entity.
ID	of the second entity. NOTE: Tests only if entities that have query flags of WALL or BUILDING are in the way, allows to see through other friendly/enemy/neutral entities.

Definition at line 157 of file CombatSystem.cpp.

6.10.3.20 bool CombatSystem::in_sight_wrt_BB (tdt::uint *ent_id,* tdt::uint *target*) const

Returns true if two given entities can see each other, false otherwise.

Tests only bounding boxes.

Parameters

ID	of the first entity.
ID	of the second entity. NOTE: Tests only if entities that have query flags of WALL or BUILDING are in the way, allows to see through other friendly/enemy/neutral entities.

Definition at line 181 of file CombatSystem.cpp.

6.10.3.21 void CombatSystem::run_away_from (tdt::uint *id,* tdt::uint *from_id,* tdt::uint *min_node_count*)

Tries to find a path used by an entity to run away from another entity.

Parameters

ID	of the entity running away.
ID	of the entity that is ran away from.
Minimal	amount of nodes the path has to have (will be ignored if the amount of attempts surpasses the maximum amount).

Note

This is just a dummy that enqueues the entity for pathfinding.

Definition at line 346 of file CombatSystem.cpp.

6.10.3.22 void CombatSystem::run_away_from_ (tdt::uint *id,* tdt::uint *from_id,* tdt::uint *min_node_count*) [private]

Tries to find a path used by an entity to run away from another entity.

Parameters

ID	of the entity running away.
ID	of the entity that is ran away from.
Minimal	amount of nodes the path has to have (will be ignored if the amount of attempts surpasses the maximum amount).

Note

This is the actual implementation.

Definition at line 301 of file CombatSystem.cpp.

6.10.3.23 void CombatSystem::set_max_run_away_attempts (tdt::uint *val*)

Sets the maximum amount of pathfinding attempts for running away.

Parameters

The	new maximum amount.

Definition at line 341 of file CombatSystem.cpp.

6.10.3.24 void CombatSystem::update (tdt::real *delta*) [override], [virtual]

Updates all auto attack combat in the game currently in progress.

Parameters

Time	since the last frame.

Check if the target is in range and in sight and if so, attack, otherwise pursue the target if necessary.

Implements System.

Definition at line 19 of file CombatSystem.cpp.

6.10.4 Member Data Documentation

6.10.4.1 EntitySystem& CombatSystem::entities_ [private]

Reference to the game's entity system (component retrieval).

Definition at line 296 of file CombatSystem.hpp.

6.10.4.2 GridSystem& CombatSystem::grid_ [private]

Used to check if an entity is accessible.

Definition at line 306 of file CombatSystem.hpp.

6.10.4.3 tdt::uint CombatSystem::max_run_away_attempts_ {10} [private]

Maximum amount of pathfindings performed when running away from an enemy.

Definition at line 316 of file CombatSystem.hpp.

6.10.4.4 RayCaster CombatSystem::ray_caster_ [private]

Used for polygon precise line of sight checking.

Definition at line 311 of file CombatSystem.hpp.

6.10.4.5 Ogre::RaySceneQuery& CombatSystem::ray_query_ [private]

Reference to the ray cast used to check if two entities can see each other.

Definition at line 301 of file CombatSystem.hpp.

6.10.4.6 std::queue<std::tuple<tdt::uint, tdt::uint, tdt::uint> > CombatSystem::run_away_queue_ [private]

Used to distribute run away pathfindings over frames (otherwise something like a meteor falling on a big group of entities would freeze the game until all run away pathfindings are done.)

Definition at line 323 of file CombatSystem.hpp.

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

- systems/CombatSystem.hpp
- systems/CombatSystem.cpp

6.11 CommandComponent Struct Reference

Contains a list of commands an entity can respond to.

`#include <Components.hpp>`

Public Member Functions

- **CommandComponent** (const CommandComponent &)=default
- **CommandComponent** (CommandComponent &&)=default
- CommandComponent & **operator=** (const CommandComponent &)=default
- CommandComponent & **operator=** (CommandComponent &&)=default

Public Attributes

- std::bitset<(int) COMMAND_TYPE::COUNT > **possible_commands**

Static Public Attributes

- static constexpr int **type** = 37

6.11.1 Detailed Description

Contains a list of commands an entity can respond to.

Definition at line 859 of file Components.hpp.

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

- Components.hpp

6.12 Component Struct Reference

Static Public Attributes

- static constexpr int **count** = 40
- static constexpr tdt::uint **NO_ENTITY** = std::numeric_limits<tdt::uint>::max()

6.12.1 Detailed Description

Definition at line 14 of file Components.hpp.

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

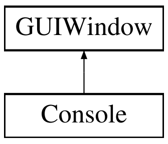
- Components.hpp

6.13 Console Class Reference

Class representing the ingame developers console that allows for runtime execution of Lua code.

`#include <Console.hpp>`

Inheritance diagram for Console:



Public Member Functions

- Console ()

    Constructor.
- ~Console ()=default

    Destructor.
- void set_visible (bool) override

    Changes the visibility and text capturing of the console window.
- bool handle_text (const CEGUI::EventArgs &)

    Event handler that is called by CEGUI whenever a text is entered.
- bool execute (const CEGUI::EventArgs &)

    Event handler that is called by CEGUI whenever the EXECUTE button is pressed.
- void print_text (const std::string &, CEGUI::Colour=CEGUI::Colour{0xFFFFFFFF})

    Prints a given message into the console window.
- void scroll_down (tdt::uint=1)

    Scrolls the console output down one line, used so that messages from the outside can scroll all the way down to the last line of the output.
- void update_fps (tdt::real, tdt::real)

    Updates the FPS value next to the console name.
- void set_history (tdt::uint)

    Sets the number of entries that will be shown in the console's history.
- tdt::uint get_history () const

    Returns the number of entries that will be shown in the console's history.
- void clear ()

    Clears the console log.

Static Public Attributes

- static const CEGUI::Colour **RED_TEXT** = CEGUI::Colour{1.f, 0.f, 0.f}
- static const CEGUI::Colour **GREEN_TEXT** = CEGUI::Colour{0.f, 1.f, 0.f}
- static const CEGUI::Colour **ORANGE_TEXT** = CEGUI::Colour{1.f, .5f, 0.1f}
- static const CEGUI::Colour **BLUE_TEXT** = CEGUI::Colour{0.f, 0.f, 1.f}

Protected Member Functions

- void init_ () override

  Initializes the console and subscribes it to events.

Private Attributes

- CEGUI::Listbox ∗ list_box_

  Pointer to the CEGUI ListBox widget that serves as console output.
- std::string curr_command_

  String containing the current command, allows to append more statements to it until the EXECUTE button is pressed, which allows for multi-line Lua code.
- tdt::real time_since_last_fps_update_

  Monitors the time passed since the fps label was last updated, used to make sure the fps update won't slow down the game as it involves float to string conversion.
- tdt::uint console_history_

  Limits the number of console entries that will be shown in it's history.

Additional Inherited Members

6.13.1 Detailed Description

Class representing the ingame developers console that allows for runtime execution of Lua code.

Definition at line 12 of file Console.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Console::Console ( )

Constructor.

Definition at line 10 of file Console.cpp.

6.13.2.2 Console::∼Console ( ) [default]

Destructor.

6.13.3 Member Function Documentation

6.13.3.1 void Console::clear ( )

Clears the console log.

Definition at line 130 of file Console.cpp.

6.13.3.2 bool Console::execute (const CEGUI::EventArgs & *args*)

Event handler that is called by CEGUI whenever the EXECUTE button is pressed.

Parameters

Reference	to the CEGUI argument.

Definition at line 44 of file Console.cpp.

6.13.3.3 tdt::uint Console::get_history ( ) const

Returns the number of entries that will be shown in the console's history.

Definition at line 125 of file Console.cpp.

6.13.3.4 bool Console::handle_text (const CEGUI::EventArgs & *args*)

Event handler that is called by CEGUI whenever a text is entered.

Parameters

Reference	to the CEGUI argument.

Definition at line 35 of file Console.cpp.

6.13.3.5 void Console::init_ ( ) `[override], [protected], [virtual]`

Initializes the console and subscribes it to events.

Implements GUIWindow.

Definition at line 15 of file Console.cpp.

6.13.3.6 void Console::print_text (const std::string & *msg,* CEGUI::Colour *col* = `CEGUI::Colour{0xFFFFFFFF}`)

Prints a given message into the console window.

Parameters

Message	to be printed.
Colour	of the message text, defaults to white.

Definition at line 82 of file Console.cpp.

6.13.3.7 void Console::scroll_down (tdt::uint *num_of_scrolls* = `1`)

Scrolls the console output down one line, used so that messages from the outside can scroll all the way down to the last line of the output.

Parameters

Amount	of lines to scroll.

Definition at line 104 of file Console.cpp.

6.13.3.8 void Console::set_history (tdt::uint *val*)

Sets the number of entries that will be shown in the console's history.

Parameters

The	new entry count.

Definition at line 120 of file Console.cpp.

6.13.3.9 void Console::set_visible (bool *visible*) `[override], [virtual]`

Changes the visibility and text capturing of the console window.

Parameters

The	new visibility state.

Reimplemented from GUIWindow.

Definition at line 28 of file Console.cpp.

6.13.3.10 void Console::update_fps (tdt::real *delta,* tdt::real *fps*)

Updates the FPS value next to the console name.

Parameters

Time	since the last frame.
The	new framerate.

Definition at line 110 of file Console.cpp.

6.13.4 Member Data Documentation

6.13.4.1 tdt::uint Console::console_history_ `[private]`

Limits the number of console entries that will be shown in it's history.

Definition at line 110 of file Console.hpp.

6.13.4.2 std::string Console::curr_command_ [private]

String containing the current command, allows to append more statements to it until the EXECUTE button is pressed, which allows for multi-line Lua code.

Definition at line 98 of file Console.hpp.

6.13.4.3 CEGUI::Listbox* Console::list_box_ [private]

Pointer to the CEGUI ListBox widget that serves as console output.

Definition at line 91 of file Console.hpp.

6.13.4.4 tdt::real Console::time_since_last_fps_update_ [private]

Monitors the time passed since the fps label was last updated, used to make sure the fps update won't slow down the game as it involves float to string conversion.

Definition at line 105 of file Console.hpp.

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

- gui/Console.hpp
- gui/Console.cpp

6.14 ConstructorComponent Struct Reference

Contains the blueprint that gets called when an entity that has this component is created.

```
#include <Components.hpp>
```

Public Member Functions

- **ConstructorComponent** (std::string &&b="ERROR")
- **ConstructorComponent** (const ConstructorComponent &)=default
- **ConstructorComponent** (ConstructorComponent &&)=default
- ConstructorComponent & **operator=** (const ConstructorComponent &)=default
- ConstructorComponent & **operator=** (ConstructorComponent &&)=default

Public Attributes

- std::string **blueprint**

Static Public Attributes

- static constexpr int **type** = 28

6.14.1 Detailed Description

Contains the blueprint that gets called when an entity that has this component is created.

Definition at line 669 of file Components.hpp.

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

- Components.hpp

6.15 CounterComponent Struct Reference

A simple incrementing counter.

```
#include <Components.hpp>
```

Public Member Functions

- **CounterComponent** (tdt::uint max=0)
- **CounterComponent** (const CounterComponent &)=default
- **CounterComponent** (CounterComponent &&)=default
- CounterComponent & **operator=** (const CounterComponent &)=default
- CounterComponent & **operator=** (CounterComponent &&)=default

Public Attributes

- tdt::uint **curr_value**
- tdt::uint **max_value**

Static Public Attributes

- static constexpr int **type** = 38

6.15.1 Detailed Description

A simple incrementing counter.

Note

The max value is not enforced and serves only for manual checking.

Definition at line 878 of file Components.hpp.

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

- Components.hpp

6.16 util::effect::DAMAGE_EFFECT Struct Reference

Deals a random damage in a given range to the entity it's called on.

```
#include <Effects.hpp>
```

Public Member Functions

- DAMAGE_EFFECT (EntitySystem &, tdt::uint, tdt::uint)

  Constructor.
- ∼DAMAGE_EFFECT ()=default

  Destructor.
- void operator() (tdt::uint)

  Applies the damage effect to a given entity.

Private Attributes

- tdt::uint min_

  The damage range.
- tdt::uint **max_**
- EntitySystem & entities_

  Entity system containing the entities this effect will be used on.

6.16.1 Detailed Description

Deals a random damage in a given range to the entity it's called on.

Definition at line 23 of file Effects.hpp.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 util::effect::DAMAGE_EFFECT::DAMAGE_EFFECT (**EntitySystem** & *ents,* tdt::uint *min,* tdt::uint *max*)

Constructor.

Parameters

Entity	system containing the entities this effect will be used on.
Lower	bound of the damage range.
Upper	bound of the damage range.

Definition at line 7 of file Effects.cpp.

6.16.2.2 util::effect::DAMAGE_EFFECT::∼DAMAGE_EFFECT ( ) `[default]`

Destructor.

6.16.3 Member Function Documentation

6.16.3.1 void util::effect::DAMAGE_EFFECT::operator() (tdt::uint *id*)

Applies the damage effect to a given entity.

Parameters

ID	of the entity.

Definition at line 11 of file Effects.cpp.

6.16.4 Member Data Documentation

6.16.4.1 EntitySystem& util::effect::DAMAGE_EFFECT::entities_ `[private]`

Entity system containing the entities this effect will be used on.

Definition at line 55 of file Effects.hpp.

6.16.4.2 tdt::uint util::effect::DAMAGE_EFFECT::min_ `[private]`

The damage range.

Definition at line 49 of file Effects.hpp.

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

- tools/Effects.hpp
- tools/Effects.cpp

6.17 DestructorComponent Struct Reference

Contains name of the table that contains the function (called "dtor") which is called when an entity is destroyed.

`#include <Components.hpp>`

Public Member Functions

- **DestructorComponent** (std::string b="ERROR")
- **DestructorComponent** (const DestructorComponent &)=default
- **DestructorComponent** (DestructorComponent &&)=default
- DestructorComponent & **operator=** (const DestructorComponent &)=default
- DestructorComponent & **operator=** (DestructorComponent &&)=default

Public Attributes

- std::string **blueprint**

Static Public Attributes

- static constexpr int **type** = 20

6.17.1 Detailed Description

Contains name of the table that contains the function (called "dtor") which is called when an entity is destroyed.

Definition at line 508 of file Components.hpp.

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

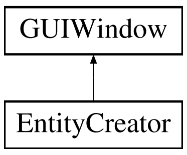
- Components.hpp

6.18 EntityCreator Class Reference

Class representing the debugging GUI window used to place and create entities during runtime.

```
#include <EntityCreator.hpp>
```

Inheritance diagram for EntityCreator:



Public Member Functions

- EntityCreator (EntityPlacer &, EntitySystem &)

  Constructor, loads the gui layout for this window and registers callbacks.
- ∼EntityCreator ()

  Destructor.
- bool place (const CEGUI::EventArgs &)

  Function that is called when the player/developer presses the "place now" button, sets the currently selected entity blueprint for placing.
- bool change_to_place (const CEGUI::EventArgs &)

  Function that is called when the player/developer presses the "place" button, changing the creator to the placing mode.
- bool change_to_create (const CEGUI::EventArgs &)

  Function that is called when the player/developer presses the "create" button, changing the creator to the creation mode.
- bool actualize_list (const CEGUI::EventArgs &)

  Function that is called when the player/developer presses the "actualize list" button and displays the set of all selected entities to the list box.

Protected Member Functions

- void init_ () override

  Initializes the EntityCreator.

Private Attributes

- EntityPlacer & placer_

  Reference to the game's entity placer, used to set the blueprint table and visibility mode when the player/developer presses the "place now" button.
- std::set< std::string > & registered_entities_

  Reference to the list of all registered entity blueprint names used for updates.
- CEGUI::Listbox ∗ list_box_

  Auxiliary pointer to the list box sub window for easy access when updating the entity blueprint list.

Additional Inherited Members

6.18.1 Detailed Description

Class representing the debugging GUI window used to place and create entities during runtime.

Definition at line 14 of file EntityCreator.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 EntityCreator::EntityCreator (EntityPlacer & *placer*, EntitySystem & *ents*)

Constructor, loads the gui layout for this window and registers callbacks.

Parameters

Reference	to the game's entity placer.
Reference	to the game's entity system.

Definition at line 6 of file EntityCreator.cpp.

6.18.2.2 EntityCreator::∼EntityCreator ( ) `[inline]`

Destructor.

Definition at line 27 of file EntityCreator.hpp.

6.18.3 Member Function Documentation

6.18.3.1 bool EntityCreator::actualize_list (const CEGUI::EventArgs & *args*)

Function that is called when the player/developer presses the "actualize list" button and displays the set of all selected entities to the list box.

Parameters

Reference	to the CEGUI event arguments.

Definition at line 34 of file EntityCreator.cpp.

6.18.3.2 bool EntityCreator::change_to_create (const CEGUI::EventArgs & *args*)

Function that is called when the player/developer presses the "create" button, changing the creator to the creation mode.

Parameters

Reference	to the CEGUI event arguments.

Definition at line 28 of file EntityCreator.cpp.

6.18.3.3 bool EntityCreator::change_to_place (const CEGUI::EventArgs & *args*)

Function that is called when the player/developer presses the "place" button, changing the creator to the placing mode.

Parameters

Reference	to the CEGUI event arguments.

Definition at line 22 of file EntityCreator.cpp.

6.18.3.4 void EntityCreator::init_ ( ) [override], [protected], [virtual]

Initializes the EntityCreator.

Implements GUIWindow.

Definition at line 51 of file EntityCreator.cpp.

6.18.3.5 bool EntityCreator::place (const CEGUI::EventArgs & *args*)

Function that is called when the player/developer presses the "place now" button, sets the currently selected entity blueprint for placing.

Parameters

Reference	to the CEGUI event arguments.

Definition at line 10 of file EntityCreator.cpp.

6.18.4 Member Data Documentation

6.18.4.1 CEGUI::Listbox∗ EntityCreator::list_box_ `[private]`

Auxiliary pointer to the list box sub window for easy access when updating the entity blueprint list.

Definition at line 80 of file EntityCreator.hpp.

6.18.4.2 EntityPlacer& EntityCreator::placer_ `[private]`

Reference to the game's entity placer, used to set the blueprint table and visibility mode when the player/developer presses the "place now" button.

Definition at line 69 of file EntityCreator.hpp.

6.18.4.3 std::set<std::string>& EntityCreator::registered_entities_ `[private]`

Reference to the list of all registered entity blueprint names used for updates.

Definition at line 74 of file EntityCreator.hpp.

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

- gui/EntityCreator.hpp
- gui/EntityCreator.cpp

6.19 util::EntityDestroyer Class Reference

A structure providing the private method EntitySystem::destroy_entity to the DestructorHelper::destroy function.

`#include <Util.hpp>`

Static Private Member Functions

- static void destroy (EntitySystem &, tdt::uint)
  Destroy a given entity.

Friends

- void **DestructorHelper::destroy** (EntitySystem &, tdt::uint, bool, tdt::uint)

6.19.1 Detailed Description

A structure providing the private method EntitySystem::destroy_entity to the DestructorHelper::destroy function.

The reason for the existence of this struct is that it provides only this one method and keeps others private.

Definition at line 194 of file Util.hpp.

6.19.2 Member Function Documentation

6.19.2.1 void util::EntityDestroyer::destroy (**EntitySystem** & *ents,* tdt::uint *id*) `[static]`, `[private]`

Destroy a given entity.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 40 of file Util.cpp.

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

- tools/Util.hpp
- tools/Util.cpp

6.20 EntityPlacer Class Reference

Class allowing the player/developer to place entities on the ground.

```
#include <EntityPlacer.hpp>
```

Public Member Functions

- EntityPlacer (EntitySystem &, GridSystem &, Ogre::SceneManager &)

  Constructor.
- ∼EntityPlacer ()

  Destructor.
- void set_current_entity_table (const std::string &, bool=false)

  Checks if a given entity blueprint contains a graphics component and if so, sets it as the currently placed entity by creating a dummy entity following the cursor.
- void update_position (const Ogre::Vector3 &)

  Called every frame when the entity placer is active, adjusts the dummy entity's position to the cursor's.
- tdt::uint place ()

  Creates a new entity from the blueprint table at the mouse cursor's current position and informs the developer in the developer console.
- void set_visible (bool)

  Sets the visibility status of the placer (and it's dummy entity).
- bool is_visible () const

  Returns true if the placer is currently active (and thus the dummy entity visible), false otherwise.
- void toggle_placing_when_game_paused ()

  Toggles (=negates) the flag for entity placing while the game is paused.
- bool can_place_when_game_paused () const

  Returns true if the placer can place entities when the game is paused, false otherwise.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- GridSystem & grid_

  Reference to the game's grid system.
- Ogre::Vector3 curr_position_

  Current position of the dummy entity (used to avoid the need to pass the position again on the actual placement cal).
- Ogre::SceneNode ∗ placing_node_

  Pointer to the scene node the dummy entity is attached to.
- bool visible_

  The placer's visibility (and active) status.
- std::string table_name_

  Name of the blueprint of the currently placed entity.
- tdt::real half_height_

  Used to correctly place the newly created entity on the ground.
- bool placing_structure_

  Determines if a structure (wall, building etc.) is being placed, this will make the dummy entity to snap to the grid nodes.
- tdt::uint structure_radius_

  Radius of the placed structure (if it's covering more than one node).
- Ogre::SceneManager & mgr_

  Scene manager used to hold the dummy node and manipulate models.
- Ogre::Entity ∗ ent_

  Entity representing the mesh of the placed object.
- tdt::uint price_

  Price of the currently placed entity.
- bool can_place_when_game_paused_ {false}

  Used for debugging, allows to place spawners when the game is paused and this their production is halted.

6.20.1 Detailed Description

Class allowing the player/developer to place entities on the ground.

(Mainly used by the EntityCreator class, but can be invoked by "game.place_entity(TABLE)" where TABLE is the name of the desired entity's blueprint)

Definition at line 15 of file EntityPlacer.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 EntityPlacer::EntityPlacer (EntitySystem & *ents,* GridSystem & *grid,* Ogre::SceneManager & *mgr*)

Constructor.

Parameters

Reference	to the game's entity system.
Reference	to the game's grid system, used for node snapping when placing structures (i.e. walls, buildings...).
Scene	manager that will hold the dummy node.

Definition at line 8 of file EntityPlacer.cpp.

6.20.2.2 EntityPlacer::∼EntityPlacer ( )

Destructor.

Definition at line 15 of file EntityPlacer.cpp.

6.20.3 Member Function Documentation

6.20.3.1 bool EntityPlacer::can_place_when_game_paused ( ) const

Returns true if the placer can place entities when the game is paused, false otherwise.

Definition at line 154 of file EntityPlacer.cpp.

6.20.3.2 bool EntityPlacer::is_visible ( ) const

Returns true if the placer is currently active (and thus the dummy entity visible), false otherwise.

Definition at line 144 of file EntityPlacer.cpp.

6.20.3.3 tdt::uint EntityPlacer::place ( )

Creates a new entity from the blueprint table at the mouse cursor's current position and informs the developer in the developer console.

Definition at line 96 of file EntityPlacer.cpp.

6.20.3.4 void EntityPlacer::set_current_entity_table (const std::string & *table_name,* bool *use_price* = `false`)

Checks if a given entity blueprint contains a graphics component and if so, sets it as the currently placed entity by creating a dummy entity following the cursor.

Parameters

Name	of the blueprint table (describing the placed entity).
If	true, the cost of the entity will be subtracted from the player's gold and the entity won't be placed if the player does not have sufficient funds.

Definition at line 22 of file EntityPlacer.cpp.

6.20.3.5 void EntityPlacer::set_visible (bool *on_off*)

Sets the visibility status of the placer (and it's dummy entity).

Parameters

The	new visibility status.

Definition at line 135 of file EntityPlacer.cpp.

6.20.3.6 void EntityPlacer::toggle_placing_when_game_paused ( )

Toggles (=negates) the flag for entity placing while the game is paused.

Definition at line 149 of file EntityPlacer.cpp.

6.20.3.7 void EntityPlacer::update_position (const Ogre::Vector3 & *pos*)

Called every frame when the entity placer is active, adjusts the dummy entity's position to the cursor's.

Parameters

Position	of the mouse cursor (recieved from Game::get_mouse_click_position).

Definition at line 73 of file EntityPlacer.cpp.

6.20.4 Member Data Documentation

6.20.4.1 bool EntityPlacer::can_place_when_game_paused_ {false} [private]

Used for debugging, allows to place spawners when the game is paused and this their production is halted.

Definition at line 150 of file EntityPlacer.hpp.

6.20.4.2 Ogre::Vector3 EntityPlacer::curr_position_ [private]

Current position of the dummy entity (used to avoid the need to pass the position again on the actual placement cal).

Definition at line 95 of file EntityPlacer.hpp.

6.20.4.3 Ogre::Entity* EntityPlacer::ent_ [private]

Entity representing the mesh of the placed object.

Definition at line 139 of file EntityPlacer.hpp.

6.20.4.4 EntitySystem& EntityPlacer::entities_ [private]

Reference to the game's entity system.

(Used to create entities as well as checking the loaded tables.)

Definition at line 83 of file EntityPlacer.hpp.

6.20.4.5 GridSystem& EntityPlacer::grid_ [private]

Reference to the game's grid system.

(Used for node snapping when placing structures).

Definition at line 89 of file EntityPlacer.hpp.

6.20.4.6 tdt::real EntityPlacer::half_height_ [private]

Used to correctly place the newly created entity on the ground.

(The mouse cursor position is at 0 Y coordinate, but the entity might need to be higher due to it's central point not being at the bottom of the model.)

Definition at line 118 of file EntityPlacer.hpp.

6.20.4.7 Ogre::SceneManager& EntityPlacer::mgr_ [private]

Scene manager used to hold the dummy node and manipulate models.

Definition at line 134 of file EntityPlacer.hpp.

6.20.4.8 Ogre::SceneNode* EntityPlacer::placing_node_ [private]

Pointer to the scene node the dummy entity is attached to.

Definition at line 100 of file EntityPlacer.hpp.

6.20.4.9 bool EntityPlacer::placing_structure_ [private]

Determines if a structure (wall, building etc.) is being placed, this will make the dummy entity to snap to the grid nodes.

Definition at line 124 of file EntityPlacer.hpp.

6.20.4.10 tdt::uint EntityPlacer::price_ [private]

Price of the currently placed entity.

Definition at line 144 of file EntityPlacer.hpp.

6.20.4.11 tdt::uint EntityPlacer::structure_radius_ [private]

Radius of the placed structure (if it's covering more than one node).

Definition at line 129 of file EntityPlacer.hpp.

6.20.4.12 std::string EntityPlacer::table_name_ [private]

Name of the blueprint of the currently placed entity.

Definition at line 110 of file EntityPlacer.hpp.

6.20.4.13 bool EntityPlacer::visible_ [private]

The placer's visibility (and active) status.

Definition at line 105 of file EntityPlacer.hpp.

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

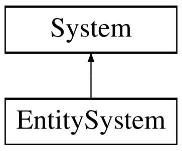
- tools/EntityPlacer.hpp
- tools/EntityPlacer.cpp

6.21 EntitySystem Class Reference

The EntitySystem class handles everything related to entities, like addition and removal of components, testing if an entity has a component or retrieval of components belonging to particular entities.

```
#include <EntitySystem.hpp>
```

Inheritance diagram for EntitySystem:



Public Member Functions

- EntitySystem (Ogre::SceneManager &)

  Constructor.
- ∼EntitySystem ()=default

  Destructor.
- void update (tdt::real) override

  Checks for entities with no components and if any are found, deletes them.
- tdt::uint get_new_id ()

  Returns first available entity id.
- void cleanup ()

  Removes all entities that have no components and individual components marked for deletion from their entities (this is used so that the Lua code does not delete an entity/a component from a container while C++ iterates over it).
- tdt::uint create_entity (const std::string &="", const Ogre::Vector3 &=Ogre::Vector3{0.f, 0.f, 0.f})

  Creates a new entity from a blueprint.
- const std::map< tdt::uint, std::bitset< Component::count > > & get_component_list () const

  Breif: Returns const reference to the component list, so that it can be used to iterate over all entities.
- template<typename COMP >
  bool has_component (tdt::uint id)

  Tests whether a given entity has a component specialized by the template argument.
- bool has_component (tdt::uint, tdt::uint) const

  Tests whether a given entity has a component of a given type (used from Lua as it cannot use templates).
- template<typename COMP >
  COMP ∗ get_component (tdt::uint id)

  Returns a bool-component pointer pair, in which the first bool member determines if the component was found and the second is a pointer to the component.
- template<typename COMP >
  void set_component (tdt::uint id, COMP comp)

  Changes a component (type specified by template argument) of and entity or assigns a new component it that entity didn't have it.
- template<typename COMP >
  std::map< tdt::uint, COMP > & get_component_container ()

  Returns the map associated with the component specified by the template argument.
- template<typename COMP >
  void add_component (tdt::uint id)

  Adds a components to the given enetity using it's default constructor (all values have to be set afterwards).
- void add_component (tdt::uint, int)

  Allows to add a component based on it's ID.
- template<typename COMP >
  void delete_component (tdt::uint id)

  Marks a component (specified by template argument) for given entity for deletion.
- void delete_component (tdt::uint, int)

  Allows to enqueue a component for deletion based on it's ID.
- void register_entity (const std::string &)

  Registers an entity that has been loaded from a Lua script.
- std::set< std::string > & get_registered_entities ()

  Returns a reference to the set containing all entity tables registered during the game's runtime.
- bool exists (tdt::uint) const

  Checks if a given entity exists and returns true if it does, false otherwise.
- Ogre::SceneManager & get_scene_manager ()

  Returns a reference to the scene manager all entities of this system are attached to (if they have a graphics component).
- void delete_entities ()

Deletes all entities int the game, used before loading a new game.

- template<>
  std::map< tdt::uint, PhysicsComponent > & get_component_container ()

  Specializations of the EntitySystem::get_component_container method.

- template<>
  std::map< tdt::uint, HealthComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, AIComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, GraphicsComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, MovementComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, CombatComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, EventComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, InputComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, TimeComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ManaComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, SpellComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ProductionComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, GridNodeComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ProductComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, PathfindingComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, TaskComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, TaskHandlerComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, StructureComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, HomingComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, EventHandlerComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, DestructorComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, GoldComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, FactionComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, PriceComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, AlignComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, MineComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ManaCrystalComponent > & **get_component_container** ()

- template<>
  std::map< tdt::uint, OnHitComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ConstructorComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, TriggerComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, UpgradeComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, NotificationComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ExplosionComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, LimitedLifeSpanComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, NameComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, ExperienceValueComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, LightComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, CommandComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, CounterComponent > & **get_component_container** ()
- template<>
  std::map< tdt::uint, PortalComponent > & **get_component_container** ()

Public Attributes

- std::string NO_BLUEPRINT {"ERROR"}

  Used in helpers when no component exists and we still need to return the blueprint name (in this case the ERROR blueprint) by reference.
- std::array< std::string, 3 > FACTION_NAME {"FRIENDLY", "ENEMY", "NEUTRAL"}

  Used in when translating the faction enum to a string in the FactionHelper.

Private Types

- typedef void(EntitySystem::∗ **LoaderFuncPtr**) (tdt::uint, const std::string &)
- typedef void(EntitySystem::∗ **AdderFuncPtr**) (tdt::uint)
- typedef void(EntitySystem::∗ **DeleterFuncPtr**) (tdt::uint)
- typedef void(EntitySystem::∗ **ImmediateDeleterFuncPtr**) (tdt::uint)

Private Member Functions

- template<typename COMP >
  void load_component (tdt::uint id, const std::string &table_name)

  Loads a component from a Lua script.
- void destroy_entity (tdt::uint)

  Removes an entity from the system, thus killing/destroying it.
- template<typename COMP >
  void delete_component_now (tdt::uint id)

  Deletes a component.

- void delete_component_now (tdt::uint, int)

  Deletes a component.
- void init_function_arrays ()

  Initializes all arrays holding pointers to the component manipulating methods.
- template<typename COMP >
  void clean_up_component (tdt::uint)

  Deletes all necessary data when destroying a component (like Ogre related objects, other entities, tasks etc.).
- template<>
  void load_component (tdt::uint id, const std::string &table_name)

  Specializations of the EntitySystem::load_component method.
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)

- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void **load_component** (tdt::uint id, const std::string &table_name)
- template<>
  void clean_up_component (tdt::uint id)

  Specializations of the EntitySystem::clean_up_component method.
- template<>
  void clean_up_component (tdt::uint id)
- template<>
  void clean_up_component (tdt::uint id)
- template<>
  void **clean_up_component** (tdt::uint id)
- template<>
  void **clean_up_component** (tdt::uint id)
- template<>
  void **clean_up_component** (tdt::uint id)
- template<>
  void **clean_up_component** (tdt::uint id)
- template<>
  void **clean_up_component** (tdt::uint id)

Private Attributes

- std::map< tdt::uint, std::bitset< Component::count > > entities_

  Contains bitsets describing component availability.
- std::vector< tdt::uint > to_be_destroyed_

  Used to mark components or entire entities for removal.
- std::vector< std::pair< tdt::uint, int > > **components_to_be_removed_**
- std::map< tdt::uint, PhysicsComponent > physics_ {}

  Contain components specified by the entity ID.
- std::map< tdt::uint, HealthComponent > **health_** {}

- std::map< tdt::uint, AIComponent > **ai_** {}
- std::map< tdt::uint, GraphicsComponent > **graphics_** {}
- std::map< tdt::uint, MovementComponent > **movement_** {}
- std::map< tdt::uint, CombatComponent > **combat_** {}
- std::map< tdt::uint, EventComponent > **event_** {}
- std::map< tdt::uint, InputComponent > **input_** {}
- std::map< tdt::uint, TimeComponent > **time_** {}
- std::map< tdt::uint, ManaComponent > **mana_** {}
- std::map< tdt::uint, SpellComponent > **spell_** {}
- std::map< tdt::uint, ProductionComponent > **production_** {}
- std::map< tdt::uint, GridNodeComponent > **grid_node_** {}
- std::map< tdt::uint, ProductComponent > **product_** {}
- std::map< tdt::uint, PathfindingComponent > **pathfinding_** {}
- std::map< tdt::uint, TaskComponent > **task_** {}
- std::map< tdt::uint, TaskHandlerComponent > **task_handler_** {}
- std::map< tdt::uint, StructureComponent > **structure_** {}
- std::map< tdt::uint, HomingComponent > **homing_** {}
- std::map< tdt::uint, EventHandlerComponent > **event_handler_** {}
- std::map< tdt::uint, DestructorComponent > **destructor_** {}
- std::map< tdt::uint, GoldComponent > **gold_** {}
- std::map< tdt::uint, FactionComponent > **faction_** {}
- std::map< tdt::uint, PriceComponent > **price_** {}
- std::map< tdt::uint, AlignComponent > **align_** {}
- std::map< tdt::uint, MineComponent > **mine_** {}
- std::map< tdt::uint, ManaCrystalComponent > **mana_crystal_** {}
- std::map< tdt::uint, OnHitComponent > **on_hit_** {}
- std::map< tdt::uint, ConstructorComponent > **constructor_** {}
- std::map< tdt::uint, TriggerComponent > **trigger_** {}
- std::map< tdt::uint, UpgradeComponent > **upgrade_** {}
- std::map< tdt::uint, NotificationComponent > **notification_** {}
- std::map< tdt::uint, ExplosionComponent > **explosion_** {}
- std::map< tdt::uint, LimitedLifeSpanComponent > **limited_life_span_** {}
- std::map< tdt::uint, NameComponent > **name_** {}
- std::map< tdt::uint, ExperienceValueComponent > **exp_value_** {}
- std::map< tdt::uint, LightComponent > **light_** {}
- std::map< tdt::uint, CommandComponent > **command_** {}
- std::map< tdt::uint, CounterComponent > **counter_** {}
- std::map< tdt::uint, PortalComponent > **portal_** {}
- Ogre::SceneManager & scene_

  Reference to the game's scene manager used to create nodes and entities.
- std::set< std::string > entity_register_

  Contains the names of all loaded entity tables.
- std::array< LoaderFuncPtr, Component::count > loaders_ {}

  These arrays contain pointers to the component managment methods for easier use when Lua interacts with C++, since Lua doesn't know anything about C++ types and templates.
- std::array< AdderFuncPtr, Component::count > **adders_** {}
- std::array< DeleterFuncPtr, Component::count > **deleters_** {}
- std::array< ImmediateDeleterFuncPtr, Component::count > **immediate_deleters_** {}
- tdt::uint curr_id_ {}

  Keeps track of the highest ID given to an entity.
- std::vector< tdt::uint > constructors_to_be_called_ {}

  Entities that should have their constructors called on the next update (freshly created), this will allow the creator of the entity to set it's position if he could not pass that position to the create_entity function.

Friends

- class **util::EntityDestroyer**

6.21.1 Detailed Description

The EntitySystem class handles everything related to entities, like addition and removal of components, testing if an entity has a component or retrieval of components belonging to particular entities.

Definition at line 21 of file EntitySystem.hpp.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 EntitySystem::EntitySystem (Ogre::SceneManager & *mgr*)

Constructor.

Parameters

Reference	to the game's scene manager used to create nodes and entities.

Definition at line 32 of file EntitySystem.cpp.

6.21.2.2 EntitySystem::∼EntitySystem ( ) `[default]`

Destructor.

6.21.3 Member Function Documentation

6.21.3.1 template<typename COMP > void EntitySystem::add_component (tdt::uint *id*) `[inline]`

Adds a components to the given enetity using it's default constructor (all values have to be set afterwards).

Parameters

ID	of the entity.

Definition at line 140 of file EntitySystem.hpp.

6.21.3.2 void EntitySystem::add_component (tdt::uint *ent_id,* int *comp_id*)

Allows to add a component based on it's ID.

Parameters

ID	of the entity.
ID	of the component.

Definition at line 154 of file EntitySystem.cpp.

6.21.3.3 template<typename COMP > void EntitySystem::clean_up_component (tdt::uint) [inline], [private]

Deletes all necessary data when destroying a component (like Ogre related objects, other entities, tasks etc.).

Parameters

ID	of the entity.

Definition at line 263 of file EntitySystem.hpp.

6.21.3.4 template<> void EntitySystem::clean_up_component (tdt::uint *id*) [inline], [private]

Specializations of the EntitySystem::clean_up_component method.

Definition at line 1044 of file EntitySystem.hpp.

6.21.3.5 template<> void EntitySystem::clean_up_component (tdt::uint *id*) [inline], [private]

Since this is probably called while iterating over the list of all entities, creating new entity would probably invalidate the iterator, so the gold pile that is supposed to be picked up is given the event component (which is then removed when handled).

Better to not make it maximal, as we would still prefer a close miner to pick it up (it will increase in time if the event is not handled).

Definition at line 1057 of file EntitySystem.hpp.

6.21.3.6 template<> void EntitySystem::clean_up_component (tdt::uint *id*) [inline], [private]

There is a slight chance that an entity would get killed right after destroying a gold deposit but before it could handle the event, so in that case just mark that event as global, so other miners can pick it up.

Definition at line 1102 of file EntitySystem.hpp.

6.21.3.7 void EntitySystem::cleanup ( )

Removes all entities that have no components and individual components marked for deletion from their entities (this is used so that the Lua code does not delete an entity/a component from a container while C++ iterates over it).

Remove entire entities. NOTE: Creating a new vector and swaping it with the to_be_destroyed_ vector, because when a TaskHandlerComponent is deleted, new entities (the tasks) are added which might result in iterator invalidation.

Definition at line 62 of file EntitySystem.cpp.

6.21.3.8 tdt::uint EntitySystem::create_entity (const std::string & *table_name* = `""`, const Ogre::Vector3 & *position* = `Ogre::Vector3{0.f, 0.f, 0.f}`)

Creates a new entity from a blueprint.

Parameters

Name	of the Lua table containing the entity blueprint.
Optional	position of the entity.

Definition at line 109 of file EntitySystem.cpp.

6.21.3.9 template<typename COMP > void EntitySystem::delete_component (tdt::uint *id*) `[inline]`

Marks a component (specified by template argument) for given entity for deletion.

Parameters

ID	of the entity.

Definition at line 162 of file EntitySystem.hpp.

6.21.3.10 void EntitySystem::delete_component (tdt::uint *ent_id*, int *comp_id*)

Allows to enqueue a component for deletion based on it's ID.

Parameters

ID	of the entity.
ID	of the component.

Definition at line 160 of file EntitySystem.cpp.

6.21.3.11 template<typename COMP > void EntitySystem::delete_component_now (tdt::uint *id*) `[inline]`, `[private]`

Deletes a component.

Parameters

ID	of the entity.

Definition at line 235 of file EntitySystem.hpp.

6.21.3.12 void EntitySystem::delete_component_now (tdt::uint *ent_id,* int *comp_id*) [private]

Deletes a component.

Parameters

ID	of the entity.
ID	of the component.

Definition at line 166 of file EntitySystem.cpp.

6.21.3.13 void EntitySystem::delete_entities ( )

Deletes all entities int the game, used before loading a new game.

Definition at line 188 of file EntitySystem.cpp.

6.21.3.14 void EntitySystem::destroy_entity (tdt::uint *id*) [private]

Removes an entity from the system, thus killing/destroying it.

Parameters

ID	of the entity.

Definition at line 144 of file EntitySystem.cpp.

6.21.3.15 bool EntitySystem::exists (tdt::uint *id*) const

Checks if a given entity exists and returns true if it does, false otherwise.

Parameters

ID	of the entity.

Definition at line 182 of file EntitySystem.cpp.

6.21.3.16 template<typename COMP > COMP* EntitySystem::get_component (tdt::uint *id*) [inline]

Returns a bool-component pointer pair, in which the first bool member determines if the component was found and the second is a pointer to the component.

Parameters

ID	of the entity whose component we ask for.

Definition at line 100 of file EntitySystem.hpp.

6.21.3.17 template< typename COMP > std::map< tdt::uint, COMP >& EntitySystem::get_component_container ( )

Returns the map associated with the component specified by the template argument.

6.21.3.18 template<> std::map< tdt::uint, PhysicsComponent >& EntitySystem::get_component_container ( ) [inline]

Specializations of the EntitySystem::get_component_container method.

Definition at line 360 of file EntitySystem.hpp.

6.21.3.19 const std::map< tdt::uint, std::bitset< Component::count > > & EntitySystem::get_component_list ( ) const

Breif: Returns const reference to the component list, so that it can be used to iterate over all entities.

Definition at line 149 of file EntitySystem.cpp.

6.21.3.20 tdt::uint EntitySystem::get_new_id ( )

Returns first available entity id.

Definition at line 44 of file EntitySystem.cpp.

6.21.3.21 std::set< std::string > & EntitySystem::get_registered_entities ( )

Returns a reference to the set containing all entity tables registered during the game's runtime.

Definition at line 177 of file EntitySystem.cpp.

6.21.3.22 Ogre::SceneManager& EntitySystem::get_scene_manager ( ) [inline]

Returns a reference to the scene manager all entities of this system are attached to (if they have a graphics component).

Definition at line 197 of file EntitySystem.hpp.

6.21.3.23 template< typename COMP > bool EntitySystem::has_component (tdt::uint *id*) [inline]

Tests whether a given entity has a component specialized by the template argument.

Parameters

ID	of the entity being checked.

Note

VS2015RC does not let me use variadic templates with recursion to check multiple components for some reason, investigate!

Definition at line 80 of file EntitySystem.hpp.

6.21.3.24 bool EntitySystem::has_component (tdt::uint *id*, tdt::uint *comp*) const

Tests whether a given entity has a component of a given type (used from Lua as it cannot use templates).

Parameters

ID	of the entity.
Type	of the component.

Definition at line 362 of file EntitySystem.cpp.

6.21.3.25 void EntitySystem::init_function_arrays ( ) `[private]`

Initializes all arrays holding pointers to the component manipulating methods.

Definition at line 195 of file EntitySystem.cpp.

6.21.3.26 template<typename COMP > void EntitySystem::load_component (tdt::uint *id*, const std::string & *table_name*) `[private]`

Loads a component from a Lua script.

Parameters

ID	of the entity.
Name	of the table containing the component.

6.21.3.27 template<> void EntitySystem::load_component (tdt::uint *id*, const std::string & *table_name*) `[inline]`, `[private]`

Specializations of the EntitySystem::load_component method.

Note

Following components can only be created manually and thus don't have load_component specialization. GridNodeComponent (created by GridSystem::add_node) ProductComponent (production id is assigned during runtime) TaskComponent (tasks are specified by their types and are added through the TaskHelper)

Definition at line 607 of file EntitySystem.hpp.

6.21.3.28 void EntitySystem::register_entity (const std::string & *table_name*)

Registers an entity that has been loaded from a Lua script.

(If it has been registered previously, the register ignores it.)

Parameters

Name	of the table containing the info about the entity.

Definition at line 172 of file EntitySystem.cpp.

6.21.3.29 template<typename COMP > void EntitySystem::set_component (tdt::uint *id,* COMP *comp*) [inline]

Changes a component (type specified by template argument) of and entity or assigns a new component it that entity didn't have it.

Parameters

ID	of the entity.
Component	to be assigned.

Definition at line 116 of file EntitySystem.hpp.

6.21.3.30 void EntitySystem::update (tdt::real) [override], [virtual]

Checks for entities with no components and if any are found, deletes them.

Parameters

Time	since the last frame.

Implements System.

Definition at line 39 of file EntitySystem.cpp.

6.21.4 Member Data Documentation

6.21.4.1 std::vector<tdt::uint> EntitySystem::constructors_to_be_called_ {} [private]

Entities that should have their constructors called on the next update (freshly created), this will allow the creator of the entity to set it's position if he could not pass that position to the create_entity function.

Definition at line 353 of file EntitySystem.hpp.

6.21.4.2 tdt::uint EntitySystem::curr_id_ {} [private]

Keeps track of the highest ID given to an entity.

Definition at line 345 of file EntitySystem.hpp.

6.21.4.3 std::map<tdt::uint, std::bitset<Component::count> > EntitySystem::entities_ [private]

Contains bitsets describing component availability.

Definition at line 269 of file EntitySystem.hpp.

6.21.4.4 std::set<std::string> EntitySystem::entity_register_ [private]

Contains the names of all loaded entity tables.

Definition at line 330 of file EntitySystem.hpp.

6.21.4.5 std::array<std::string, 3> EntitySystem::FACTION_NAME {"FRIENDLY", "ENEMY", "NEUTRAL"}

Used in when translating the faction enum to a string in the FactionHelper.

Definition at line 213 of file EntitySystem.hpp.

6.21.4.6 std::array<LoaderFuncPtr, Component::count> EntitySystem::loaders_ {} [private]

These arrays contain pointers to the component managment methods for easier use when Lua interacts with C++, since Lua doesn't know anything about C++ types and templates.

Definition at line 337 of file EntitySystem.hpp.

6.21.4.7 std::string EntitySystem::NO_BLUEPRINT {"ERROR"}

Used in helpers when no component exists and we still need to return the blueprint name (in this case the ERROR blueprint) by reference.

Definition at line 208 of file EntitySystem.hpp.

6.21.4.8 std::map<tdt::uint, PhysicsComponent> EntitySystem::physics_ {} [private]

Contain components specified by the entity ID.

Initialized here to avoid a long initializing list in the constructor.

Definition at line 281 of file EntitySystem.hpp.

6.21.4.9 Ogre::SceneManager& EntitySystem::scene_ [private]

Reference to the game's scene manager used to create nodes and entities.

Definition at line 325 of file EntitySystem.hpp.

6.21.4.10 std::vector<tdt::uint> EntitySystem::to_be_destroyed_ [private]

Used to mark components or entire entities for removal.

Definition at line 274 of file EntitySystem.hpp.

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

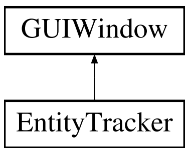
- systems/EntitySystem.hpp
- systems/EntitySystem.cpp

6.22 EntityTracker Class Reference

A window that monitors the stats of the currently selected entity and allows for it's upgrading once it has enough experience.

`#include <EntityTracker.hpp>`

Inheritance diagram for EntityTracker:



Public Member Functions

- EntityTracker ()
  Constructor.
- ~EntityTracker ()=default
  Destructor.
- void set_tracked_entity (tdt::uint, EntitySystem &)
  Sets the new tracked entity and loads it's data.
- tdt::uint get_tracked_entity () const
  Returns the ID of the currently tracked entity.
- void update_tracking (const std::string &, const std::string &)
  Updates a single stat of the entity tracker.
- void clear ()
  Clears the entity tracker's window, that is sets all values to 0/0 and the id to UNKNOWN.
- void init_upgrade_butt (EntitySystem ∗)
  Adds a callback to the UPGRADE button that upgrades an entity.
- void show_upgrade_butt (bool)
  Sets the visibility status of the UPGRADE button.

Protected Member Functions

- void init_ () override

  Initializes the window and sets all event subscribers.

Private Attributes

- tdt::uint curr_tracked_entity_

  ID of the currently tracked entity.
- EntitySystem ∗ entities_

  Used for upgrading.

Additional Inherited Members

6.22.1 Detailed Description

A window that monitors the stats of the currently selected entity and allows for it's upgrading once it has enough experience.

Definition at line 11 of file EntityTracker.hpp.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 EntityTracker::EntityTracker ( )

Constructor.

Definition at line 7 of file EntityTracker.cpp.

6.22.2.2 EntityTracker::∼EntityTracker ( ) [default]

Destructor.

6.22.3 Member Function Documentation

6.22.3.1 void EntityTracker::clear ( )

Clears the entity tracker's window, that is sets all values to 0/0 and the id to UNKNOWN.

Definition at line 70 of file EntityTracker.cpp.

6.22.3.2 tdt::uint EntityTracker::get_tracked_entity ( ) const

Returns the ID of the currently tracked entity.

Definition at line 59 of file EntityTracker.cpp.

6.22.3.3 void EntityTracker::init_( ) [override], [protected], [virtual]

Initializes the window and sets all event subscribers.

Implements GUIWindow.

Definition at line 138 of file EntityTracker.cpp.

6.22.3.4 void EntityTracker::init_upgrade_butt (EntitySystem * *ents*)

Adds a callback to the UPGRADE button that upgrades an entity.

Parameters

Entity	system containing entities that will be upgraded by this button.

Definition at line 87 of file EntityTracker.cpp.

6.22.3.5 void EntityTracker::set_tracked_entity (tdt::uint *id*, EntitySystem & *ents*)

Sets the new tracked entity and loads it's data.

Parameters

ID	of the entity.
EntitySystem	that contains the entity.

Definition at line 11 of file EntityTracker.cpp.

6.22.3.6 void EntityTracker::show_upgrade_butt (bool *val*)

Sets the visibility status of the UPGRADE button.

Parameters

True	for visible, false for invisible.

Definition at line 133 of file EntityTracker.cpp.

6.22.3.7 void EntityTracker::update_tracking (const std::string & *label*, const std::string & *value*)

Updates a single stat of the entity tracker.

Parameters

Name	of the stat label (e.g. "HP_LABEL", "GOLD_LABEL" etc).
String	with the new value.

Note

The value should have the form "[CURRENT_VALUE]/[MAX_VALUE]".

Definition at line 64 of file EntityTracker.cpp.

6.22.4 Member Data Documentation

6.22.4.1 tdt::uint EntityTracker::curr_tracked_entity_ [private]

ID of the currently tracked entity.

Definition at line 72 of file EntityTracker.hpp.

6.22.4.2 EntitySystem∗ EntityTracker::entities_ [private]

Used for upgrading.

Definition at line 77 of file EntityTracker.hpp.

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

- gui/EntityTracker.hpp
- gui/EntityTracker.cpp

6.23 EventComponent Struct Reference

Represents events that happen in the game, like gold seams dropping gold, curing an entity of poisoning or triggers from traps etc.

`#include <Components.hpp>`

Public Member Functions

- **EventComponent** (EVENT_TYPE ev=EVENT_TYPE::NONE, tdt::uint t=Component::NO_ENTITY, tdt::real r=0.f, bool a=true)
- **EventComponent** (const EventComponent &)=default
- **EventComponent** (EventComponent &&)=default
- EventComponent & **operator=** (const EventComponent &)=default
- EventComponent & **operator=** (EventComponent &&)=default

Public Attributes

- EVENT_TYPE **event_type**
- tdt::uint **target**
- tdt::uint **handler**
- tdt::real **radius**
- bool **active**

Static Public Attributes

- static constexpr int **type** = 6

6.23.1 Detailed Description

Represents events that happen in the game, like gold seams dropping gold, curing an entity of poisoning or triggers from traps etc.

Definition at line 183 of file Components.hpp.

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

- Components.hpp

6.24 EventHandlerComponent Struct Reference

Allows to cherry pink when it comes to event handling and handle only certain events.

```
#include <Components.hpp>
```

Public Member Functions

- **EventHandlerComponent** (std::string &&h="ERROR")
- **EventHandlerComponent** (const EventHandlerComponent &)=default
- **EventHandlerComponent** (EventHandlerComponent &&)=default
- EventHandlerComponent & **operator=** (const EventHandlerComponent &)=default
- EventHandlerComponent & **operator=** (EventHandlerComponent &&)=default

Public Attributes

- std::string **handler**
- std::bitset<(int) EVENT_TYPE::COUNT > **possible_events**

Static Public Attributes

- static constexpr int **type** = 19

6.24.1 Detailed Description

Allows to cherry pink when it comes to event handling and handle only certain events.

(Also, only entities with this component will react to events.)

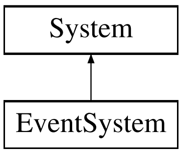
Definition at line 487 of file Components.hpp.

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

- Components.hpp

6.25 EventSystem Class Reference

Inheritance diagram for EventSystem:



Public Member Functions

- EventSystem (EntitySystem &)

  Constructor.
- ∼EventSystem ()=default

  Destructor.
- void update (tdt::real) override

  Checks for handlers of all active events and if needed, handles them.
- void set_update_period (tdt::real)

  Sets the time it takes before an update is performed (in seconds).
- tdt::real get_update_period () const

  Returns the time it takes before an update is performed (in seconds).
- void set_update_time_multiplier (tdt::real)

  Sets the value by which the frame time is multiplied before it's added to the update timer.
- tdt::real get_update_time_multiplier () const

  Returns the value by which the frame time is multiplied before it's added to the update timer.

Private Member Functions

- bool handle_event_ (tdt::uint, tdt::uint)

  Handles a given event by a given entity that can handle it, returns true if the event will be destroyed after this call (single handler event), false if the event persist (multi handler event).

Private Attributes

- EntitySystem & entities_

  Entity system that has entities this system will manage.
- tdt::real update_period_

  Allow for less frequent updates, since per frame updates are quite unnecessary and resource wasting.
- tdt::real **curr_update_time_**
- tdt::real update_time_multiplier_

  Allows to speed up/slow down the update timer.

6.25.1 Detailed Description

Definition at line 7 of file EventSystem.hpp.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 EventSystem::EventSystem (**EntitySystem** & *ents*)

Constructor.

Parameters

Entity	system that has entities this system will manage.

Definition at line 6 of file EventSystem.cpp.

6.25.2.2 EventSystem::∼EventSystem ( ) [default]

Destructor.

6.25.3 Member Function Documentation

6.25.3.1 tdt::real EventSystem::get_update_period ( ) const

Returns the time it takes before an update is performed (in seconds).

Definition at line 63 of file EventSystem.cpp.

6.25.3.2 tdt::real EventSystem::get_update_time_multiplier ( ) const

Returns the value by which the frame time is multiplied before it's added to the update timer.

Definition at line 73 of file EventSystem.cpp.

6.25.3.3 bool EventSystem::handle_event_ (tdt::uint *handler*, tdt::uint *evt*) [private]

Handles a given event by a given entity that can handle it, returns true if the event will be destroyed after this call (single handler event), false if the event persist (multi handler event).

Parameters

ID	of the handler.
ID	of the event.

Definition at line 78 of file EventSystem.cpp.

6.25.3.4 void EventSystem::set_update_period (tdt::real *val*)

Sets the time it takes before an update is performed (in seconds).

Parameters

The	new update time period.

Definition at line 58 of file EventSystem.cpp.

6.25.3.5 void EventSystem::set_update_time_multiplier (tdt::real *val*)

Sets the value by which the frame time is multiplied before it's added to the update timer.

Parameters

The	new multiplier value.

Definition at line 68 of file EventSystem.cpp.

6.25.3.6 void EventSystem::update (tdt::real *delta*) [override], [virtual]

Checks for handlers of all active events and if needed, handles them.

Parameters

Time	since last frame.

Implements System.

Definition at line 11 of file EventSystem.cpp.

6.25.4 Member Data Documentation

6.25.4.1 EntitySystem& EventSystem::entities_ [private]

Entity system that has entities this system will manage.

Definition at line 67 of file EventSystem.hpp.

6.25.4.2 tdt::real EventSystem::update_period_ [private]

Allow for less frequent updates, since per frame updates are quite unnecessary and resource wasting.

Definition at line 73 of file EventSystem.hpp.

6.25.4.3 tdt::real EventSystem::update_time_multiplier_ [private]

Allows to speed up/slow down the update timer.

Definition at line 78 of file EventSystem.hpp.

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

- systems/EventSystem.hpp
- systems/EventSystem.cpp

6.26 lpp::Exception Class Reference

Exception class used to throw exception from the Script class.

```
#include <LppScript.hpp>
```

Public Member Functions

- Exception (const std::string &msg="NO MSG", Script::state L=nullptr)

  Constructor.
- const char ∗ what () const

  Returns the message of this exception.
- const char ∗ what_lua () const

  Returns the Lua error message if possible.
- bool has_lua_state () const

  Returns true if a Lua state is captured by this exception.

Private Attributes

- std::string msg_

  Message the exception was called with.
- Script::state L_

  Pointer to the Lua state, used for stack manipulation (like lua error retrieval).

6.26.1 Detailed Description

Exception class used to throw exception from the Script class.

Definition at line 266 of file LppScript.hpp.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 lpp::Exception::Exception (const std::string & *msg* = `"NO MSG"`, Script::state *L* = `nullptr`) `[inline]`

Constructor.

Parameters

Message	of the exception.

Definition at line 273 of file LppScript.hpp.

6.26.3 Member Function Documentation

6.26.3.1 bool lpp::Exception::has_lua_state ( ) const

Returns true if a Lua state is captured by this exception.

Definition at line 128 of file LppScript.cpp.

6.26.3.2 const char ∗ lpp::Exception::what ( ) const

Returns the message of this exception.

lpp::Exception definitions:

Definition at line 114 of file LppScript.cpp.

6.26.3.3 const char ∗ lpp::Exception::what_lua ( ) const

Returns the Lua error message if possible.

Definition at line 119 of file LppScript.cpp.

6.26.4 Member Data Documentation

6.26.4.1 Script::state lpp::Exception::L_ `[private]`

Pointer to the Lua state, used for stack manipulation (like lua error retrieval).

Definition at line 301 of file LppScript.hpp.

6.26.4.2 std::string lpp::Exception::msg_ `[private]`

Message the exception was called with.

Definition at line 296 of file LppScript.hpp.

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

- lppscript/LppScript.hpp
- lppscript/LppScript.cpp

6.27 ExperienceValueComponent Struct Reference

The amount of experience the entity yields when killed.

`#include <Components.hpp>`

Public Member Functions

- **ExperienceValueComponent** (tdt::uint v=0)
- **ExperienceValueComponent** (const ExperienceValueComponent &)=default
- **ExperienceValueComponent** (ExperienceValueComponent &&)=default
- ExperienceValueComponent & **operator=** (const ExperienceValueComponent &)=default
- ExperienceValueComponent & **operator=** (ExperienceValueComponent &&)=default

Public Attributes

- tdt::uint **value**

Static Public Attributes

- static constexpr int **type** = 35

6.27.1 Detailed Description

The amount of experience the entity yields when killed.

Definition at line 822 of file Components.hpp.

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

- Components.hpp

6.28 ExplosionComponent Struct Reference

Component used to create the visual effect of an explosion, the damage should be done in the explosion's constructor so that it's not applied on each frame.

```
#include <Components.hpp>
```

Public Member Functions

- **ExplosionComponent** (tdt::real d=0.f, tdt::real rad=0.f)
- **ExplosionComponent** (const ExplosionComponent &)=default
- **ExplosionComponent** (ExplosionComponent &&)=default
- ExplosionComponent & **operator=** (const ExplosionComponent &)=default
- ExplosionComponent & **operator=** (ExplosionComponent &&)=default

Public Attributes

- tdt::real **delta**
- tdt::real **max_radius**
- tdt::real **curr_radius**

Static Public Attributes

- static constexpr int **type** = 32

6.28.1 Detailed Description

Component used to create the visual effect of an explosion, the damage should be done in the explosion's constructor so that it's not applied on each frame.

Definition at line 761 of file Components.hpp.

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

- Components.hpp

6.29 FactionComponent Struct Reference

Represents the faction an entity that has this component if a member of.

```
#include <Components.hpp>
```

Public Member Functions

- **FactionComponent** (FACTION f=FACTION::NEUTRAL)
- **FactionComponent** (const FactionComponent &)=default
- **FactionComponent** (FactionComponent &&)=default
- FactionComponent & **operator=** (const FactionComponent &)=default
- FactionComponent & **operator=** (FactionComponent &&)=default

Public Attributes

- FACTION **faction**

Static Public Attributes

- static constexpr int **type** = 22

6.29.1 Detailed Description

Represents the faction an entity that has this component if a member of.

Definition at line 549 of file Components.hpp.

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

- Components.hpp

6.30 util::path_type::FIRST_PATH Struct Reference

Finds the first path by accepting the first path found.

```
#include <PathfindingAlgorithms.hpp>
```

Static Public Member Functions

- static bool **return_path** ()

6.30.1 Detailed Description

Finds the first path by accepting the first path found.

Definition at line 154 of file PathfindingAlgorithms.hpp.

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

- tools/PathfindingAlgorithms.hpp

6.31 util::effect::FREEZE_EFFECT Struct Reference

Freezes a given entity in place for a given time period.

```
#include <Effects.hpp>
```

Public Member Functions

- FREEZE_EFFECT (EntitySystem &, tdt::real)

  Constructor.
- ∼FREEZE_EFFECT ()=default

  Destructor.
- void operator() (tdt::uint)

  Freezes a given entity in place.

Private Attributes

- EntitySystem & entities_

  Entity system containing the entities this effect will be called on.
- tdt::real time_

  The duration of the freeze.

6.31.1 Detailed Description

Freezes a given entity in place for a given time period.

(This effect stops movement but not any other action.)

Definition at line 133 of file Effects.hpp.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 util::effect::FREEZE_EFFECT::FREEZE_EFFECT (EntitySystem & *ents,* tdt::real *time*)

Constructor.

Parameters

Entity	system containing the entities this effect will be called on.
Duration	of the freeze.

Definition at line 58 of file Effects.cpp.

6.31.2.2 util::effect::FREEZE_EFFECT::∼FREEZE_EFFECT ( ) `[default]`

Destructor.

6.31.3 Member Function Documentation

6.31.3.1 void util::effect::FREEZE_EFFECT::operator() (tdt::uint *id*)

Freezes a given entity in place.

Parameters

ID	of the entity.

Definition at line 62 of file Effects.cpp.

6.31.4 Member Data Documentation

6.31.4.1 EntitySystem& util::effect::FREEZE_EFFECT::entities_ `[private]`

Entity system containing the entities this effect will be called on.

Definition at line 159 of file Effects.hpp.

6.31.4.2 tdt::real util::effect::FREEZE_EFFECT::time_ `[private]`

The duration of the freeze.

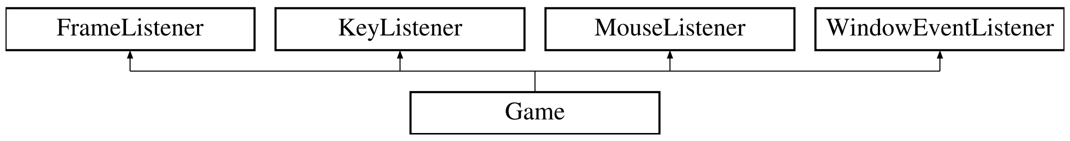
Definition at line 164 of file Effects.hpp.

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

- tools/Effects.hpp
- tools/Effects.cpp

6.32 Game Class Reference

Inheritance diagram for Game:



Public Member Functions

- Game ()

    Constructor.
- ∼Game ()

    Destructor.
- void run ()

    Starts the game.
- void update (tdt::real)

    Updates the game in one frame.
- void set_state (GAME_STATE)

    Changes the game's state.
- void new_game (tdt::uint, tdt::uint)

    Creates a new game with the given dimensions.
- void create_empty_level (tdt::uint, tdt::uint)

    Creates an empty level with the given dimensions.
- void reset_camera ()

    Resets the main camera's position and orientation to it's original state.
- void reset_unlocks ()

    Restores the unlocks to their initial state.
- void set_throne_id (tdt::uint)

    Sets the ID of the entity that represents the Dungeon Throne.
- tdt::uint get_throne_id () const

    Returns the ID of the entity that represents the Dungeon Throne.

Protected Member Functions

- bool frameRenderingQueued (const Ogre::FrameEvent &) override

    Called when the previous frame is queued for rendering, used for game updates.
- bool keyPressed (const OIS::KeyEvent &) override

    Called when a key is pressed.
- bool keyReleased (const OIS::KeyEvent &) override

    Called when a key is released.
- bool mouseMoved (const OIS::MouseEvent &) override

    Called when the mouse is moved.
- bool mousePressed (const OIS::MouseEvent &, OIS::MouseButtonID) override

    Called when a mouse button is pressed.
- bool mouseReleased (const OIS::MouseEvent &, OIS::MouseButtonID) override

    Called when a mouse button is released.
- void windowResized (Ogre::RenderWindow ∗rw) override

    Called when the window is resized.
- void windowClosed (Ogre::RenderWindow ∗rw) override

    Called when the window is closed.

Private Member Functions

- void ogre_init ()

  Init methods.
- void **ois_init** ()
- void **cegui_init** ()
- CEGUI::MouseButton ois_to_cegui (OIS::MouseButtonID)

  Converts an OIS key code to CEGUI key code.
- void toggle_camera_free_mode ()

  Toggles the free camera movement mode.
- std::pair< bool, Ogre::Vector3 > get_mouse_click_position (const OIS::MouseEvent &) const

  Returns a pair consisting of the location of the point where the player clicked on the ground plane and a boolean indicating if the click was indeed on the ground plane.

Private Attributes

- GAME_STATE state_

  Current game state.
- std::unique_ptr< Ogre::Root > root_

  Pointers to Ogre3D objects.
- Ogre::SceneManager ∗ **scene_mgr_**
- Ogre::RenderWindow ∗ **window_**
- Ogre::Viewport ∗ **main_view_**
- Ogre::Light ∗ **main_light_**
- OIS::InputManager ∗ **input_**
- OIS::Keyboard ∗ **keyboard_**
- OIS::Mouse ∗ **mouse_**
- std::unique_ptr< Camera > main_cam_

  Camera rendering to the window.
- std::unique_ptr< EntitySystem > entity_system_ {nullptr}

  Unique pointers to systems (sadly all systems require the EntitySystem, which requires Ogre::SceneManager, so the all have to be instantiated after the ogre_init method call and thus cannot be references).
- std::unique_ptr< HealthSystem > **health_system_** {nullptr}
- std::unique_ptr< MovementSystem > **movement_system_** {nullptr}
- std::unique_ptr< AISystem > **ai_system_** {nullptr}
- std::unique_ptr< InputSystem > **input_system_** {nullptr}
- std::unique_ptr< GridSystem > **grid_system_** {nullptr}
- std::unique_ptr< TaskSystem > **task_system_** {nullptr}
- std::unique_ptr< CombatSystem > **combat_system_** {nullptr}
- std::unique_ptr< ProductionSystem > **production_system_** {nullptr}
- std::unique_ptr< TimeSystem > **time_system_** {nullptr}
- std::unique_ptr< EventSystem > **event_system_** {nullptr}
- std::unique_ptr< GraphicsSystem > **graphics_system_** {nullptr}
- std::unique_ptr< TriggerSystem > **trigger_system_** {nullptr}
- std::unique_ptr< ManaSpellSystem > **mana_spell_system_** {nullptr}
- std::unique_ptr< WaveSystem > **wave_system_** {nullptr}
- std::unique_ptr< GameSerializer > game_serializer_ {nullptr}

  Used to save the game.
- std::vector< System ∗ > systems_ {}

  Vector of all systems used for updating the game's logic.
- CEGUI::OgreRenderer ∗ renderer_

  CEGUI renderer.

- std::unique_ptr< EntityPlacer > placer_

  Allows to spawn entities with the mouse ingame.
- std::unique_ptr< Ogre::Plane > ground_

  The game's ground represented by a plane, used for ray intersections.
- Ogre::Entity ∗ ground_entity_

  Entity holding the ground plane.
- std::unique_ptr< SelectionBox > selection_box_

  Selection box used to select multiple entities at once.
- std::unique_ptr< EntityCreator > entity_creator_

  A gui window that allows to place entities selected from a menu.
- Ogre::Vector2 mouse_position_

  Saved position of the mouse cursor (2D).
- std::unique_ptr< level_generators::LevelGenerator > level_generator_

  Used to create new levels (that is, to distribute walls and gold deposits).
- std::unique_ptr< Spellcaster > spell_caster_

  Allows the player to cast spells of different types (positional, targeted etc.).
- tdt::uint throne_id_

  ID of the entity representing the Dungeon Throne, losing which ends the game.

Friends

- class **GameSerializer**
- class **LuaInterface**
- class **GUI**
- void **action::QUICK_LOAD** ()
- void **action::QUICK_SAVE** ()
- void **action::RESET_CAMERA** ()

6.32.1 Detailed Description

Definition at line 40 of file Game.hpp.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 Game::Game ( )

Constructor.

Definition at line 30 of file Game.cpp.

6.32.2.2 Game::∼Game ( )

Destructor.

Definition at line 104 of file Game.cpp.

6.32.3 Member Function Documentation

6.32.3.1 void Game::create_empty_level (tdt::uint *width,* tdt::uint *height*)

Creates an empty level with the given dimensions.

Parameters

Width	of the level.
Height	of the level.

Definition at line 186 of file Game.cpp.

6.32.3.2 bool Game::frameRenderingQueued (const Ogre::FrameEvent & *event*) [override], [protected]

Called when the previous frame is queued for rendering, used for game updates.

Definition at line 264 of file Game.cpp.

6.32.3.3 std::pair< bool, Ogre::Vector3 > Game::get_mouse_click_position (const OIS::MouseEvent & *event*) const [private]

Returns a pair consisting of the location of the point where the player clicked on the ground plane and a boolean indicating if the click was indeed on the ground plane.

Definition at line 631 of file Game.cpp.

6.32.3.4 tdt::uint Game::get_throne_id ( ) const

Returns the ID of the entity that represents the Dungeon Throne.

Definition at line 259 of file Game.cpp.

6.32.3.5 bool Game::keyPressed (const OIS::KeyEvent & *event*) [override], [protected]

Called when a key is pressed.

Definition at line 278 of file Game.cpp.

6.32.3.6 bool Game::keyReleased (const OIS::KeyEvent & *event*) [override], [protected]

Called when a key is released.

This will make sure global and targeted spells go off even when a hotkey is used.

Definition at line 318 of file Game.cpp.

6.32.3.7 bool Game::mouseMoved (const OIS::MouseEvent & *event*) [override], [protected]

Called when the mouse is moved.

Definition at line 348 of file Game.cpp.

6.32.3.8 bool Game::mousePressed (const OIS::MouseEvent & *event*, OIS::MouseButtonID *id*) [override], [protected]

Called when a mouse button is pressed.

Note: If the spell is targeted and not one entity is selected, the spell shall be casted on mouse button release that makes sufficient selection.

Definition at line 384 of file Game.cpp.

6.32.3.9 bool Game::mouseReleased (const OIS::MouseEvent & *event*, OIS::MouseButtonID *id*) [override], [protected]

Called when a mouse button is released.

Note: Selected the spell without a target, so when we select one target, apply the effect on him. This will also cause global spells to be casted immediately after clicking the button.

Definition at line 435 of file Game.cpp.

6.32.3.10 void Game::new_game (tdt::uint *width*, tdt::uint *height*)

Creates a new game with the given dimensions.

Parameters

Width	of the level.
Height	of the level.

Definition at line 172 of file Game.cpp.

6.32.3.11 void Game::ogre_init ( ) [private]

Init methods.

Definition at line 503 of file Game.cpp.

6.32.3.12 CEGUI::MouseButton Game::ois_to_cegui (OIS::MouseButtonID *id*) [private]

Converts an OIS key code to CEGUI key code.

Parameters

O↩ IS	key code.

Definition at line 611 of file Game.cpp.

6.32.3.13 void Game::reset_camera ( )

Resets the main camera's position and orientation to it's original state.

Definition at line 231 of file Game.cpp.

6.32.3.14 void Game::reset_unlocks ( )

Restores the unlocks to their initial state.

Definition at line 236 of file Game.cpp.

6.32.3.15 void Game::run ( )

Starts the game.

Definition at line 110 of file Game.cpp.

6.32.3.16 void Game::set_state (GAME_STATE *state*)

Changes the game's state.

Parameters

The	new state.

Definition at line 154 of file Game.cpp.

6.32.3.17 void Game::set_throne_id (tdt::uint *id*)

Sets the ID of the entity that represents the Dungeon Throne.

Parameters

The	new ID.

Definition at line 254 of file Game.cpp.

6.32.3.18 void Game::toggle_camera_free_mode ( ) `[private]`

Toggles the free camera movement mode.

Definition at line 626 of file Game.cpp.

6.32.3.19 void Game::update (tdt::real *delta*)

Updates the game in one frame.

Parameters

Time	since the last frame.

Definition at line 118 of file Game.cpp.

6.32.3.20 void Game::windowClosed (Ogre::RenderWindow ∗ *rw*) `[override]`, `[protected]`

Called when the window is closed.

Definition at line 485 of file Game.cpp.

6.32.3.21 void Game::windowResized (Ogre::RenderWindow ∗ *rw*) `[override]`, `[protected]`

Called when the window is resized.

Definition at line 471 of file Game.cpp.

6.32.4 Member Data Documentation

6.32.4.1 std::unique_ptr<EntityCreator> Game::entity_creator_ `[private]`

A gui window that allows to place entities selected from a menu.

(TODO: Allow to create/modify entities.)

Definition at line 272 of file Game.hpp.

6.32.4.2 std::unique_ptr<EntitySystem> Game::entity_system_ {nullptr} `[private]`

Unique pointers to systems (sadly all systems require the EntitySystem, which requires Ogre::SceneManager, so the all have to be instantiated after the ogre_init method call and thus cannot be references).

Definition at line 199 of file Game.hpp.

6.32.4.3 std::unique_ptr<GameSerializer> Game::game_serializer_ {nullptr} `[private]`

Used to save the game.

Definition at line 218 of file Game.hpp.

6.32.4.4 std::unique_ptr<Ogre::Plane> Game::ground_ `[private]`

The game's ground represented by a plane, used for ray intersections.

Definition at line 244 of file Game.hpp.

6.32.4.5 Ogre::Entity* Game::ground_entity_ [private]

Entity holding the ground plane.

Used for easier plane switching.

Definition at line 249 of file Game.hpp.

6.32.4.6 std::unique_ptr<level_generators::LevelGenerator> Game::level_generator_ [private]

Used to create new levels (that is, to distribute walls and gold deposits).

Definition at line 283 of file Game.hpp.

6.32.4.7 std::unique_ptr<Camera> Game::main_cam_ [private]

Camera rendering to the window.

Definition at line 192 of file Game.hpp.

6.32.4.8 Ogre::Vector2 Game::mouse_position_ [private]

Saved position of the mouse cursor (2D).

Definition at line 277 of file Game.hpp.

6.32.4.9 std::unique_ptr<EntityPlacer> Game::placer_ [private]

Allows to spawn entities with the mouse ingame.

Definition at line 239 of file Game.hpp.

6.32.4.10 CEGUI::OgreRenderer* Game::renderer_ [private]

CEGUI renderer.

Definition at line 228 of file Game.hpp.

6.32.4.11 std::unique_ptr<Ogre::Root> Game::root_ [private]

Pointers to Ogre3D objects.

Definition at line 179 of file Game.hpp.

6.32.4.12 std::unique_ptr<SelectionBox> Game::selection_box_ [private]

Selection box used to select multiple entities at once.

Definition at line 266 of file Game.hpp.

6.32.4.13 std::unique_ptr<Spellcaster> Game::spell_caster_ [private]

Allows the player to cast spells of different types (positional, targeted etc.).

Definition at line 288 of file Game.hpp.

6.32.4.14 GAME_STATE Game::state_ [private]

Current game state.

Definition at line 174 of file Game.hpp.

6.32.4.15 std::vector<System*> Game::systems_ {} [private]

Vector of all systems used for updating the game's logic.

Definition at line 223 of file Game.hpp.

6.32.4.16 tdt::uint Game::throne_id_ [private]

ID of the entity representing the Dungeon Throne, losing which ends the game.

Definition at line 294 of file Game.hpp.

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

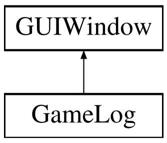
- Game.hpp
- Game.cpp

6.33 GameLog Class Reference

Class representing the log window used to show messages to the player.

`#include <GameLog.hpp>`

Inheritance diagram for GameLog:



Public Member Functions

- GameLog ()

  Constructor.
- ∼GameLog ()=default

  Destructor.
- void clear ()

  Clears the game's log by deleting all it's entries.
- void print (const std::string &)

  Prints a string to the game's log.
- void set_history (tdt::uint)

  Sets the amount of entries kept in the game's log.
- tdt::uint get_history () const

  Returns the amoung of entries kept in the game's log.

Protected Member Functions

- void init_ () override

  Initializes the game log (called by parent's init).

Private Attributes

- tdt::uint log_history_

  Number of entires kept in the game log.
- CEGUI::Listbox ∗ log_

  Pointer to the log window for easy access (as it might get called quite often, this will avoid frequent lookups).

Additional Inherited Members

6.33.1 Detailed Description

Class representing the log window used to show messages to the player.

(No space for gold, enemies attacking etc.)

Note

For debug/technical etc messages, see the Console class.

Definition at line 15 of file GameLog.hpp.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 GameLog::GameLog ( )

Constructor.

Definition at line 4 of file GameLog.cpp.

6.33.2.2 GameLog::∼GameLog ( ) [default]

Destructor.

6.33.3 Member Function Documentation

6.33.3.1 void GameLog::clear ( )

Clears the game's log by deleting all it's entries.

Definition at line 8 of file GameLog.cpp.

6.33.3.2 tdt::uint GameLog::get_history ( ) const

Returns the amoung of entries kept in the game's log.

Definition at line 36 of file GameLog.cpp.

6.33.3.3 void GameLog::init_ ( ) [override], [protected], [virtual]

Initializes the game log (called by parent's init).

Implements GUIWindow.

Definition at line 41 of file GameLog.cpp.

6.33.3.4 void GameLog::print (const std::string & *msg*)

Prints a string to the game's log.

Parameters

String	to be printed.

Definition at line 13 of file GameLog.cpp.

6.33.3.5 void GameLog::set_history (tdt::uint *val*)

Sets the amount of entries kept in the game's log.

Parameters

The	new log history.

Definition at line 31 of file GameLog.cpp.

6.33.4 Member Data Documentation

6.33.4.1 CEGUI::Listbox∗ GameLog::log_ [private]

Pointer to the log window for easy access (as it might get called quite often, this will avoid frequent lookups).

Definition at line 66 of file GameLog.hpp.

6.33.4.2 tdt::uint GameLog::log_history_ [private]

Number of entires kept in the game log.

Definition at line 60 of file GameLog.hpp.

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

- gui/GameLog.hpp
- gui/GameLog.cpp

6.34 GameSerializer Class Reference

Class that is used to save (by using Lua code generation) and loading the game (by executing said code).

`#include <GameSerializer.hpp>`

Public Member Functions

- GameSerializer (EntitySystem &)

    Constructor.
- ∼GameSerializer ()

    Destructor.
- void save_game (Game &, const std::string &="quick_save")

    Creates a Lua script that is to be used as a save file by serializing every entity into a sequence of commands that create this entity from scratch when executed.
- void load_game (Game &, const std::string &="quick_save")

    Executes a given Lua script containing a serialized game, effectively restoring the state of that game.

Private Types

- typedef void(GameSerializer::∗ **SerializerFuncPtr**) (tdt::uint, const std::string &)

Private Member Functions

- void save_tasks ()

    Adds commands to the save file that assign all tasks (has to be done last).
- std::string save_wave_system (Game &)

    Returns a string containing commands that will restore the wave system to it's current state.
- std::string save_unlocks ()

    Returns a string containing commands that will restore the unlock system to it's current state.
- template<typename COMP >
  void save_component (tdt::uint, const std::string &)

    Generates code that constructs a single component.
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)

- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)
- template<>
  void **save_component** (tdt::uint id, const std::string &tbl_name)

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system, used for component access.
- lpp::Script & script_

  Reference to the lpp::Script singleton for easy use.
- std::vector< std::pair< tdt::uint, tdt::uint > > task_pairs_

  Contains entity - task pairs that should be added in the save_tasks method.
- std::ofstream file_

  Main file stream (no need for ifstream, since loading is done through Lua).
- std::vector< std::string > save_entities_

  Auxiliary vectors that allows to place entity creation at the top (so no entity variables are nil when loading a game) and component definitions at the bottom.
- std::vector< std::string > **save_components_**
- std::array< SerializerFuncPtr, Component::count > serializers_

  Pointers to the different save_component instances allowing for easy runtime differencing between components.

6.34.1 Detailed Description

Class that is used to save (by using Lua code generation) and loading the game (by executing said code).

Definition at line 22 of file GameSerializer.hpp.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 GameSerializer::GameSerializer (EntitySystem & *ents*)

Constructor.

Parameters

Reference	to the game's entity system.

Definition at line 17 of file GameSerializer.cpp.

6.34.2.2 GameSerializer::∼GameSerializer ( ) `[inline]`

Destructor.

Definition at line 35 of file GameSerializer.hpp.

6.34.3 Member Function Documentation

6.34.3.1 void GameSerializer::load_game (Game & *game,* const std::string & *fname* = `"quick_save"`)

Executes a given Lua script containing a serialized game, effectively restoring the state of that game.

Parameters

Reference	to the game object (currently used for console entries, but might be used more in the future).
Name	of the save file to load.

Definition at line 142 of file GameSerializer.cpp.

6.34.3.2 template<typename COMP > void GameSerializer::save_component (tdt::uint , const std::string &) `[private]`

Generates code that constructs a single component.

Parameters

ID	of the component to serialize (type specialized as template argument).
Name	of the variable already in the save file that holds the new ID.

6.34.3.3 void GameSerializer::save_game (Game & *game*, const std::string & *fname* = "quick_save")

Creates a Lua script that is to be used as a save file by serializing every entity into a sequence of commands that create this entity from scratch when executed.

Parameters

Reference	to the Game object (to be able to save all necessary data).
Name	of the save file.

Definition at line 64 of file GameSerializer.cpp.

6.34.3.4 void GameSerializer::save_tasks ( ) [private]

Adds commands to the save file that assign all tasks (has to be done last).

Definition at line 172 of file GameSerializer.cpp.

6.34.3.5 std::string GameSerializer::save_unlocks ( ) [private]

Returns a string containing commands that will restore the unlock system to it's current state.

Definition at line 213 of file GameSerializer.cpp.

6.34.3.6 std::string GameSerializer::save_wave_system (Game & *game*) [private]

Returns a string containing commands that will restore the wave system to it's current state.

Parameters

Reference	to the game object that contains the wave system.

Definition at line 183 of file GameSerializer.cpp.

6.34.4 Member Data Documentation

6.34.4.1 EntitySystem& GameSerializer::entities_ [private]

Reference to the game's entity system, used for component access.

Definition at line 85 of file GameSerializer.hpp.

6.34.4.2 std::ofstream GameSerializer::file_ [private]

Main file stream (no need for ifstream, since loading is done through Lua).

Definition at line 100 of file GameSerializer.hpp.

6.34.4.3 std::vector<std::string> GameSerializer::save_entities_ [private]

Auxiliary vectors that allows to place entity creation at the top (so no entity variables are nil when loading a game) and component definitions at the bottom.

Definition at line 106 of file GameSerializer.hpp.

6.34.4.4 lpp::Script& GameSerializer::script_ [private]

Reference to the lpp::Script singleton for easy use.

Definition at line 90 of file GameSerializer.hpp.

6.34.4.5 std::array<SerializerFuncPtr, Component::count> GameSerializer::serializers_ [private]

Pointers to the different save_component instances allowing for easy runtime differencing between components.

Definition at line 112 of file GameSerializer.hpp.

6.34.4.6 std::vector<std::pair<tdt::uint, tdt::uint> > GameSerializer::task_pairs_ [private]

Contains entity - task pairs that should be added in the save_tasks method.

Definition at line 95 of file GameSerializer.hpp.

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

- tools/GameSerializer.hpp
- tools/GameSerializer.cpp

6.35 GoldComponent Struct Reference

Represents a gold amount an entity is holding, be it a gold seam, worker minion or gold depository.

```
#include <Components.hpp>
```

Public Member Functions

- **GoldComponent** (tdt::uint max=0, tdt::uint curr=0)
- **GoldComponent** (const GoldComponent &)=default
- **GoldComponent** (GoldComponent &&)=default
- GoldComponent & **operator=** (const GoldComponent &)=default
- GoldComponent & **operator=** (GoldComponent &&)=default

Public Attributes

- tdt::uint **max_amount**
- tdt::uint **curr_amount**

Static Public Attributes

- static constexpr int **type** = 21

6.35.1 Detailed Description

Represents a gold amount an entity is holding, be it a gold seam, worker minion or gold depository.

Definition at line 528 of file Components.hpp.

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

- Components.hpp

6.36 GraphicsComponent Struct Reference

Holds info related to the Ogre3D rendering library.

```
#include <Components.hpp>
```

Public Member Functions

- **GraphicsComponent** (std::string &&me="ogrehead.mesh", std::string &&ma="Ogre", bool v=true, bool manual=false, Ogre::Vector3 sc=Ogre::Vector3{0, 0, 0})
- **GraphicsComponent** (const GraphicsComponent &)=default
- **GraphicsComponent** (GraphicsComponent &&)=default
- GraphicsComponent & **operator=** (const GraphicsComponent &)=default
- GraphicsComponent & **operator=** (GraphicsComponent &&)=default

Public Attributes

- std::string **mesh**
- std::string **material**
- bool **visible**
- Ogre::SceneNode ∗ **node**
- Ogre::Entity ∗ **entity**
- bool **manual_scaling**
- Ogre::Vector3 **scale**

Static Public Attributes

- static constexpr int **type** = 3

6.36.1 Detailed Description

Holds info related to the Ogre3D rendering library.

Definition at line 101 of file Components.hpp.

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

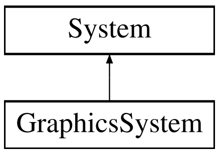
- Components.hpp

6.37 GraphicsSystem Class Reference

System that performs all graphics related updates.

`#include <GraphicsSystem.hpp>`

Inheritance diagram for GraphicsSystem:



Public Member Functions

- GraphicsSystem (EntitySystem &)

    Constructor.
- ∼GraphicsSystem ()=default

    Destructor.
- void update (tdt::real) override

    Performs all graphics updates.
- void set_update_period (tdt::real)

    Sets the time period before the next update.
- tdt::real get_update_period () const

    Returns the time period between updates.

Private Attributes

- EntitySystem & entities_

    Entity system that contains entities this system is working with.
- tdt::real update_timer_

    Used to avoid per frame updates and allows dynamic update periods.
- tdt::real **update_period_**

6.37.1 Detailed Description

System that performs all graphics related updates.

Definition at line 10 of file GraphicsSystem.hpp.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 GraphicsSystem::GraphicsSystem (**EntitySystem** & *ents*)

Constructor.

Parameters

The	game's entity system.

Definition at line 4 of file GraphicsSystem.cpp.

6.37.2.2 GraphicsSystem::∼GraphicsSystem ( ) [default]

Destructor.

6.37.3 Member Function Documentation

6.37.3.1 tdt::real GraphicsSystem::get_update_period ( ) const

Returns the time period between updates.

Definition at line 45 of file GraphicsSystem.cpp.

6.37.3.2 void GraphicsSystem::set_update_period (tdt::real *val*)

Sets the time period before the next update.

Parameters

The	new period.

Definition at line 40 of file GraphicsSystem.cpp.

6.37.3.3 void GraphicsSystem::update (tdt::real *delta*) [override], [virtual]

Performs all graphics updates.

Parameters

Time	since the last frame.

Implements System.

Definition at line 8 of file GraphicsSystem.cpp.

6.37.4 Member Data Documentation

6.37.4.1 EntitySystem& GraphicsSystem::entities_ [private]

Entity system that contains entities this system is working with.

Definition at line 45 of file GraphicsSystem.hpp.

6.37.4.2 tdt::real GraphicsSystem::update_timer_ `[private]`

Used to avoid per frame updates and allows dynamic update periods.

Definition at line 51 of file GraphicsSystem.hpp.

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

- systems/GraphicsSystem.hpp
- systems/GraphicsSystem.cpp

6.38 Grid Class Reference

Class representing the pathfinding grid.

`#include <Grid.hpp>`

Public Member Functions

- bool in_board (tdt::uint) const

  Returns true if a given node is in the grid.
- const std::set< tdt::uint > & get_freed () const

  Returns a constant reference to the list of freed nodes.
- const std::set< tdt::uint > & get_unfreed () const

  Returns a constant reference to the list of unfreed nodes.
- void clear_freed ()

  Removes all nodes from the list of freed nodes.
- void clear_unfreed ()

  Removes all nodes from the list of unfreed nodes.
- tdt::uint add_node (EntitySystem &, Ogre::Vector2)

  Created a new node at the given position.
- void add_freed (tdt::uint)

  Adds a given node to the list of the freed nodes.
- void add_unfreed (tdt::uint)

  Adds a given node to the list of the unfreed nodes.
- void remove_node (tdt::uint)

  Removes a given node from the node list.
- tdt::uint get_node (tdt::uint, tdt::uint) const

  Returns the ID of a node at a given position in the grid.
- tdt::uint get_node_from_position (tdt::real, tdt::real) const

  Returns the ID of a node that is closed to a given world coorinate.
- void create_graph (EntitySystem &, Ogre::Vector2, tdt::uint, tdt::uint, tdt::real)

  Generates a grid graph with the given parameters to be used for pathfinding.
- tdt::real get_distance () const

  Returns the distance between two nodes in the four non-diagonal directions.
- tdt::uint get_random_free_node () const

  Breif: Returns a random node within the graph.
- Ogre::Vector2 get_center_position (EntitySystem &) const

  Returns the 2D position of the central node of the grid (or one of them if the grid has even dimensions).

- Grid (const Grid &)=delete

  Since there should be only one grid at all times accesible from the Grid::instance method, all copy/move operations are disabled for this class.
- Grid & **operator=** (const Grid &)=delete
- **Grid** (Grid &&)=delete
- Grid & **operator=** (Grid &&)=delete
- bool place_at_random_free_node (EntitySystem &, tdt::uint)

  Places a given entity at a random node that is not obstructed by a building.
- bool distribute_to_adjacent_free_nodes (EntitySystem &, tdt::uint, const std::vector< tdt::uint > &)

  Distributes a given set of entities on free nodes adjacent to a given central node.

Static Public Member Functions

- static Grid & instance ()

  Returns a reference to the static instance of this class.

Private Member Functions

- Grid ()=default

  Constructor.
- ∼Grid ()

  Destructor.
- void link_ (tdt::uint, std::vector< GridNodeComponent ∗ > &)

  Generates a neighbour list for a given node (thus linking it to the graph).

Private Attributes

- std::vector< tdt::uint > nodes_

  Vector containing the IDs of the nodes in the grid, basically representing a 2D matrix stored in a 1D container.
- std::set< tdt::uint > freed_

  Auxiliary vectors containing IDs of the nodes that have been freed/unfreed on last frame.
- std::set< tdt::uint > **unfreed_**
- tdt::uint width_

  Dimensions of the grid in node count.
- tdt::uint **height_**
- tdt::real distance_

  Distance between two adjacent nodes.
- tdt::uint starting_index_

  ID of the first node, this allows for the node IDs to be outside the (0, width_ ∗ height_) range,.
- Ogre::Vector2 start_

  Coordinates of the starting node of the grid.
- std::vector< tdt::uint > free_nodes_

  Used for easier returning of a random free node.

Friends

- class GameSerializer

  GameSerializer is a friend class so that it can easily access the grid realted data (like dimensions and node distance) when saving the game.

6.38.1 Detailed Description

Class representing the pathfinding grid.

Definition at line 13 of file Grid.hpp.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 Grid::Grid (const Grid &) [delete]

Since there should be only one grid at all times accesible from the Grid::instance method, all copy/move operations are disabled for this class.

6.38.2.2 Grid::Grid ( ) [private], [default]

Constructor.

Kept private since there should be only one grid at all times.

6.38.2.3 Grid::∼Grid ( ) [inline], [private]

Destructor.

Definition at line 162 of file Grid.hpp.

6.38.3 Member Function Documentation

6.38.3.1 void Grid::add_freed (tdt::uint *id*)

Adds a given node to the list of the freed nodes.

Parameters

ID	of the node.

Definition at line 50 of file Grid.cpp.

6.38.3.2 tdt::uint Grid::add_node (EntitySystem & *ents,* Ogre::Vector2 *pos*)

Created a new node at the given position.

Parameters

EntitySystem	that contains the node.
2D	position of the node.

Definition at line 34 of file Grid.cpp.

6.38.3.3 void Grid::add_unfreed (tdt::uint *id*)

Adds a given node to the list of the unfreed nodes.

Parameters

ID	of the node.

Definition at line 70 of file Grid.cpp.

6.38.3.4 void Grid::clear_freed ( )

Removes all nodes from the list of freed nodes.

Definition at line 24 of file Grid.cpp.

6.38.3.5 void Grid::clear_unfreed ( )

Removes all nodes from the list of unfreed nodes.

Definition at line 29 of file Grid.cpp.

6.38.3.6 void Grid::create_graph (**EntitySystem** & *ents,* Ogre::Vector2 *start,* tdt::uint *w,* tdt::uint *h,* tdt::real *d*)

Generates a grid graph with the given parameters to be used for pathfinding.

Parameters

EntitySystem	that will contain the nodes.
Starting	position (x,z axes) of the grid.
Width	of the graph (in node count).
Height	of the graph (in node count).
Distance	between adjascent nodes.

Definition at line 112 of file Grid.cpp.

6.38.3.7 bool Grid::distribute_to_adjacent_free_nodes (**EntitySystem** & *ents,* tdt::uint *node,* const std::vector< tdt::uint > & *ids*)

Distributes a given set of entities on free nodes adjacent to a given central node.

Returns true if the placement was possible, false otherwise.

Parameters

Entity	system containing the entitites.
ID	of the central node.
Vector	of IDs of the entities.

Definition at line 186 of file Grid.cpp.

6.38.3.8 Ogre::Vector2 Grid::get_center_position (EntitySystem & *ents*) const

Returns the 2D position of the central node of the grid (or one of them if the grid has even dimensions).

Parameters

EntitySystem	that contains components of the nodes.

Definition at line 171 of file Grid.cpp.

6.38.3.9 tdt::real Grid::get_distance ( ) const

Returns the distance between two nodes in the four non-diagonal directions.

Definition at line 151 of file Grid.cpp.

6.38.3.10 const std::set< tdt::uint > & Grid::get_freed ( ) const

Returns a constant reference to the list of freed nodes.

Definition at line 14 of file Grid.cpp.

6.38.3.11 tdt::uint Grid::get_node (tdt::uint *x*, tdt::uint *y*) const

Returns the ID of a node at a given position in the grid.

Parameters

Column	number.
Row	number.

Definition at line 86 of file Grid.cpp.

6.38.3.12 tdt::uint Grid::get_node_from_position (tdt::real *x*, tdt::real *y*) const

Returns the ID of a node that is closed to a given world coorinate.

Parameters

X	axis coordinate.
Z	axis coordinate.

Note

Adding the ability to specify in what direction the node must be might be beneficial for pathfinding.

Definition at line 94 of file Grid.cpp.

6.38.3.13 tdt::uint Grid::get_random_free_node ( ) const

Breif: Returns a random node within the graph.

Definition at line 163 of file Grid.cpp.

6.38.3.14 const std::set< tdt::uint > & Grid::get_unfreed ( ) const

Returns a constant reference to the list of unfreed nodes.

Definition at line 19 of file Grid.cpp.

6.38.3.15 bool Grid::in_board (tdt::uint *id*) const

Returns true if a given node is in the grid.

Parameters

ID	of the node.

Definition at line 9 of file Grid.cpp.

6.38.3.16 Grid & Grid::instance ( ) `[static]`

Returns a reference to the static instance of this class.

Note

Handles initialization and safe destruction by itself.

Definition at line 156 of file Grid.cpp.

6.38.3.17 void Grid::link_ (tdt::uint *index,* std::vector< GridNodeComponent ∗ > & *comps*) `[private]`

Generates a neighbour list for a given node (thus linking it to the graph).

Parameters

ID	of the node.
Auxuliary	vector containing component pointers for fast access. (This method will ever be called only in the GridSystem::create_graph method, which already has such a vector and so it's used here too.)

Definition at line 218 of file Grid.cpp.

6.38.3.18 bool Grid::place_at_random_free_node (EntitySystem & *ents*, tdt::uint *id*)

Places a given entity at a random node that is not obstructed by a building.

Returns true if the placement was possible, false otherwise.

Parameters

Entity	system containing the entity.
ID	of the entity.

Definition at line 177 of file Grid.cpp.

6.38.3.19 void Grid::remove_node (tdt::uint *id*)

Removes a given node from the node list.

Parameters

ID	of the node. TODO: Possibly implement "unlink"?

Definition at line 80 of file Grid.cpp.

6.38.4 Friends And Related Function Documentation

6.38.4.1 friend class GameSerializer `[friend]`

GameSerializer is a friend class so that it can easily access the grid realted data (like dimensions and node distance) when saving the game.

Definition at line 19 of file Grid.hpp.

6.38.5 Member Data Documentation

6.38.5.1 tdt::real Grid::distance_ `[private]`

Distance between two adjascent nodes.

Definition at line 195 of file Grid.hpp.

6.38.5.2 std::vector< tdt::uint > Grid::free_nodes_ [private]

Used for easier returning of a random free node.

Definition at line 213 of file Grid.hpp.

6.38.5.3 std::set< tdt::uint > Grid::freed_ [private]

Auxiliary vectors containing IDs of the nodes that have been freed/unfreed on last frame.

Used for pathfinding correction and structure model changes.

Definition at line 184 of file Grid.hpp.

6.38.5.4 std::vector< tdt::uint > Grid::nodes_ [private]

Vector containing the IDs of the nodes in the grid, basically representing a 2D matrix stored in a 1D container.

Definition at line 177 of file Grid.hpp.

6.38.5.5 Ogre::Vector2 Grid::start_ [private]

Coordinates of the starting node of the grid.

(This 2D vector contains X and Z coordinates despite it's second member being names Y).

Definition at line 208 of file Grid.hpp.

6.38.5.6 tdt::uint Grid::starting_index_ [private]

ID of the first node, this allows for the node IDs to be outside the (0, width_ ∗ height_) range,.

Definition at line 201 of file Grid.hpp.

6.38.5.7 tdt::uint Grid::width_ [private]

Dimensions of the grid in node count.

(Actual dimensions = dimensions ∗ distance.)

Definition at line 190 of file Grid.hpp.

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

- tools/Grid.hpp
- tools/Grid.cpp

6.39 GridNodeComponent Struct Reference

Holds GridNode's neighbour nodes.

```
#include <Components.hpp>
```

Public Member Functions

- **GridNodeComponent** (std::array< tdt::uint, neighbour_count > neigh=std::array< tdt::uint, neighbour_↩ count >{}, bool f=true, tdt::uint pos_x=0, tdt::uint pos_y=0, tdt::uint res=Component::NO_ENTITY)
- **GridNodeComponent** (const GridNodeComponent &)=default
- **GridNodeComponent** (GridNodeComponent &&)=default
- GridNodeComponent & **operator=** (const GridNodeComponent &)=default
- GridNodeComponent & **operator=** (GridNodeComponent &&)=default

Public Attributes

- std::array< tdt::uint, neighbour_count > **neighbours**
- bool **free**
- tdt::uint **x**
- tdt::uint **y**
- tdt::uint **resident**

Static Public Attributes

- static constexpr int **type** = 12
- static constexpr tdt::uint **neighbour_count** = 9

6.39.1 Detailed Description

Holds GridNode's neighbour nodes.

Note

The neighbours are set to the maximum value of tdt::uint to fix a state when one or more neighbours weren't set (won't have that many nodes so the A$*$ algorithm will ignore them).

Definition at line 321 of file Components.hpp.

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

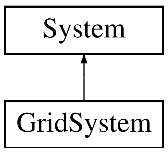
- Components.hpp

6.40 GridSystem Class Reference

Represents the pathfinding graph used by the game and provides several methods related to pathfinding that can be used in Lua.

`#include <GridSystem.hpp>`

Inheritance diagram for GridSystem:



Public Member Functions

- GridSystem (EntitySystem &, Ogre::SceneManager &)

  Constructor.
- ∼GridSystem ()=default

  Destructor.
- void update (tdt::real) override

  Checks if any nodes were freed or unfreed and if so, corrects any path that had those nodes in it.
- void create_graphics ()

  Creates and initializes Ogre models for nodes, which allows the developer to display them for testing purposes.
- void delete_graphics ()

  Deletes Ogre models of all nodes.
- void set_visible (bool)

  Changes the visibility status of the nodes if the graphics have been created already, does nothing otherwise.
- bool is_visible () const

  Returns true if the the grid models are visible, false otherwise.
- void place_structure (tdt::uint, tdt::uint, tdt::uint)

  Places a structure (building, wall...) by changing the nodes it is placed on to not free, managing residences etc.

Private Member Functions

- void update_neighbours_ (tdt::uint)

  Updates the nodes resident's alignment (if possible) when it's neighbour was freed/unfreed.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- Ogre::SceneManager & scene_mgr_

  Reference to the game's main scene manager (used for graphics creation).
- bool graphics_loaded_

  Determine if the graphics have been loaded and if the graph is visible (which is only relevant if the former is true).
- bool **graph_visible_**

6.40.1 Detailed Description

Represents the pathfinding graph used by the game and provides several methods related to pathfinding that can be used in Lua.

Definition at line 11 of file GridSystem.hpp.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 GridSystem::GridSystem (EntitySystem & *ents,* Ogre::SceneManager & *scene*)

Constructor.

Parameters

Reference	to the game's entity system.
Reference	to the game's main scene manager.

Definition at line 12 of file GridSystem.cpp.

6.40.2.2 GridSystem::∼GridSystem ( ) `[default]`

Destructor.

6.40.3 Member Function Documentation

6.40.3.1 void GridSystem::create_graphics ( )

Creates and initializes Ogre models for nodes, which allows the developer to display them for testing purposes.

Definition at line 80 of file GridSystem.cpp.

6.40.3.2 void GridSystem::delete_graphics ( )

Deletes Ogre models of all nodes.

Definition at line 106 of file GridSystem.cpp.

6.40.3.3 bool GridSystem::is_visible ( ) const

Returns true if the the grid models are visible, false otherwise.

Definition at line 130 of file GridSystem.cpp.

6.40.3.4 void GridSystem::place_structure (tdt::uint *ent_id,* tdt::uint *node_id,* tdt::uint *radius*)

Places a structure (building, wall...) by changing the nodes it is placed on to not free, managing residences etc.

Parameters

ID	of the structure.
ID	of the central node.
Radius	of the structure.

Definition at line 138 of file GridSystem.cpp.

6.40.3.5 void GridSystem::set_visible (bool *on_off*)

Changes the visibility status of the nodes if the graphics have been created already, does nothing otherwise.

Parameters

The	new visibility status.

Definition at line 114 of file GridSystem.cpp.

6.40.3.6 void GridSystem::update (tdt::real) `[override]`, `[virtual]`

Checks if any nodes were freed or unfreed and if so, corrects any path that had those nodes in it.

Parameters

Time	since the last frame.

Implements System.

Definition at line 17 of file GridSystem.cpp.

6.40.3.7 void GridSystem::update_neighbours_ (tdt::uint *id*) `[private]`

Updates the nodes resident's alignment (if possible) when it's neighbour was freed/unfreed.

Parameters

ID	of the node.

Definition at line 168 of file GridSystem.cpp.

6.40.4 Member Data Documentation

6.40.4.1 EntitySystem& GridSystem::entities_ `[private]`

Reference to the game's entity system.

Definition at line 76 of file GridSystem.hpp.

6.40.4.2 bool GridSystem::graphics_loaded_ [private]

Determine if the graphics have been loaded and if the graph is visible (which is only relevant if the former is true).

Definition at line 88 of file GridSystem.hpp.

6.40.4.3 Ogre::SceneManager& GridSystem::scene_mgr_ [private]

Reference to the game's main scene manager (used for graphics creation).

Definition at line 81 of file GridSystem.hpp.

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

- systems/GridSystem.hpp
- systems/GridSystem.cpp

6.41 GUI Class Reference

Represents the game's main graphical user interface (i.e.

`#include <GUI.hpp>`

Public Member Functions

- ∼GUI ()
  Destructor.
- void init (Game ∗)
  Initializes the selection by loading the layout and registering all event handlers.
- void set_visible (bool)
  Set's the GUI's visibility status.
- bool is_visible () const
  Returns true if the entire GUI is visible, false otherwise.
- void set_visible (const std::string &, bool)
  Set's the visibility status of a particular window.
- bool is_visible (const std::string &) const
  Returns the visibility status of a particular window.
- void show_load_save_dialog (const std::string &)
  Shows the load/save dialog window.
- CEGUI::Window ∗ get_window ()
  Returns a pointer to the root window.
- CEGUI::Window ∗ get_window (const std::string &)
  Returns a pointer to a given subwindow of the root window.
- Console & get_console ()
  Returns a reference to the game's dev console.
- EntityTracker & get_tracker ()
  Returns a reference to the entity tracker.
- GameLog & get_log ()

Returns a reference to the game's log.
- BuilderWindow & get_builder ()

  Returns a reference to the builder window.
- TopBar & get_top_bar ()

  Returns a reference to the top bar.
- ResearchWindow & get_research ()

  Returns a reference to the research window.
- SpellCastingWindow & get_spell_casting ()

  Returns a reference to the spell casting window.
- MessageToPlayerWindow & get_message ()

  Returns a reference to the message to player window.
- OptionsWindow & get_options ()

  Returns a reference to the options window.
- bool escape_pressed ()

  Notifies the GUI that the escape key was pressed so that it can close windows if needed.
- void set_curr_tool_visible (bool)

  Sets the visibility status of the current tool window.
- void set_curr_tool (const std::string &)

  Sets the current tool window.
- const std::string & get_curr_tool ()

  Returns the name of the current tool window.
- GUI (const GUI &)=delete
- **GUI** (GUI &&)=delete
- GUI & **operator=** (const GUI &)=delete
- GUI & **operator=** (GUI &&)=delete

Static Public Member Functions

- static GUI & instance ()

  Returns the singleton instance.

Private Member Functions

- GUI ()

  Constructor, private because of the singleton pattern.
- void list_directory (const std::string &, CEGUI::Listbox &, bool=false)

  Fills a given list box with the names of all files in a given directory.

Private Attributes

- CEGUI::Window ∗ window_

  Pointer to the root window of the layout.
- std::string curr_tool_

  Name of the current tool in the tools subwindow.
- Game ∗ game_

  Pointer to the game instance used by button event handlers.
- EntityTracker tracker_

  ID of the entity that is currently tracked by the entity viewer.
- Console console_

Game's developer console.
- GameLog log_

  Game's log, used to show messages to the player.
- BuilderWindow builder_

  Allows the player to place buildings.
- TopBar top_bar_

  Shows game info at the top of the screen.
- ResearchWindow research_

  Game's research window that allows the player to buy buildings, units and spells.
- SpellCastingWindow spell_casting_

  Allows the player to cast spells.
- CEGUI::Window ∗ menu_

  Allows easier access to the menu subwindow.
- MessageToPlayerWindow message_

  Allows the game to show a message to the player with any of the following buttons: OK, YES, NO.
- OptionsWindow options_

  Allows to change the resolution, window mode and key bindings.

Friends

- class **OptionsWindow**
- void **action::QUICK_LOAD** ()
- void **action::QUICK_SAVE** ()
- void **action::RESET_CAMERA** ()

6.41.1 Detailed Description

Represents the game's main graphical user interface (i.e.

any windows except for the development ones like the console or entity creator).

Definition at line 26 of file GUI.hpp.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 GUI::∼GUI ( ) [inline]

Destructor.

Definition at line 36 of file GUI.hpp.

6.41.2.2 GUI::GUI (const GUI &) [delete]

Note

Since VS2015 seems to have some problems with C++ standard (generates default copy/move constructors and operators even if default constructor is created), these constructors/operators are explicitly deleted.

6.41.2.3 GUI::GUI ( ) [private]

Constructor, private because of the singleton pattern.

Definition at line 7 of file GUI.cpp.

6.41.3 Member Function Documentation

6.41.3.1 bool GUI::escape_pressed ( )

Notifies the GUI that the escape key was pressed so that it can close windows if needed.

Returns true if anything has been closed, false otherwise.

Note

CEGUI event system does not seem to work properly :/

This will allow the return from the main menu (which does not happend when a submenu window is visible).

Definition at line 414 of file GUI.cpp.

6.41.3.2 BuilderWindow & GUI::get_builder ( )

Returns a reference to the builder window.

Definition at line 384 of file GUI.cpp.

6.41.3.3 Console & GUI::get_console ( )

Returns a reference to the game's dev console.

Definition at line 369 of file GUI.cpp.

6.41.3.4 const std::string & GUI::get_curr_tool ( )

Returns the name of the current tool window.

Definition at line 465 of file GUI.cpp.

6.41.3.5 GameLog & GUI::get_log ( )

Returns a reference to the game's log.

Definition at line 379 of file GUI.cpp.

6.41.3.6 MessageToPlayerWindow & GUI::get_message ( )

Returns a reference to the message to player window.

Definition at line 404 of file GUI.cpp.

6.41.3.7 OptionsWindow & GUI::get_options ( )

Returns a reference to the options window.

Definition at line 409 of file GUI.cpp.

6.41.3.8 ResearchWindow & GUI::get_research ( )

Returns a reference to the research window.

Definition at line 394 of file GUI.cpp.

6.41.3.9 SpellCastingWindow & GUI::get_spell_casting ( )

Returns a reference to the spell casting window.

Definition at line 399 of file GUI.cpp.

6.41.3.10 TopBar & GUI::get_top_bar ( )

Returns a reference to the top bar.

Definition at line 389 of file GUI.cpp.

6.41.3.11 EntityTracker & GUI::get_tracker ( )

Returns a reference to the entity tracker.

Definition at line 374 of file GUI.cpp.

6.41.3.12 CEGUI::Window ∗ GUI::get_window ( )

Returns a pointer to the root window.

Definition at line 359 of file GUI.cpp.

6.41.3.13 CEGUI::Window ∗ GUI::get_window (const std::string & *name*)

Returns a pointer to a given subwindow of the root window.

Parameters

Name	of the window.

Definition at line 364 of file GUI.cpp.

6.41.3.14 void GUI::init (Game ∗ *game*)

Initializes the selection by loading the layout and registering all event handlers.

Parameters

Pointer	to the game object, which is used by the event handlers (like the quit button etc).

The options menu was stored in it's own separate file for easier design.

MAIN MENU

TOOL SELECTION

MENU

Definition at line 14 of file GUI.cpp.

6.41.3.15 GUI & GUI::instance ( ) [static]

Returns the singleton instance.

Definition at line 335 of file GUI.cpp.

6.41.3.16 bool GUI::is_visible ( ) const

Returns true if the entire GUI is visible, false otherwise.

Definition at line 320 of file GUI.cpp.

6.41.3.17 bool GUI::is_visible (const std::string & *wname*) const

Returns the visibility status of a particular window.

Parameters

Name	(path) of the window, without the root window prefix.

Definition at line 330 of file GUI.cpp.

6.41.3.18 void GUI::list_directory (const std::string & *dir,* CEGUI::Listbox & *box,* bool *strip_ext* = `false`) [private]

Fills a given list box with the names of all files in a given directory.

Parameters

Name	of the directory.
List	box to be filled.
If	true, the .lua extension will be cut from the file name if present.

Definition at line 470 of file GUI.cpp.

6.41.3.19 void GUI::set_curr_tool (const std::string & *val*)

Sets the current tool window.

Parameters

The	name of the tool. (MENU, SPELL, BUILD)

Definition at line 459 of file GUI.cpp.

6.41.3.20 void GUI::set_curr_tool_visible (bool *val*)

Sets the visibility status of the current tool window.

Parameters

The	new visibility status.

Definition at line 454 of file GUI.cpp.

6.41.3.21 void GUI::set_visible (bool *val*)

Set's the GUI's visibility status.

Parameters

The	new visibility status. TODO: Toggle GUI with a hotkey for screen capturing?

Definition at line 315 of file GUI.cpp.

6.41.3.22 void GUI::set_visible (const std::string & *wname,* bool *val*)

Set's the visibility status of a particular window.

Parameters

Name	(path) of the window, without the root window prefix. (e.g. "TOOLS/TOOL_SELECTION/SPELL_SELECTION")
The	new visibility status.

Definition at line 325 of file GUI.cpp.

6.41.3.23 void GUI::show_load_save_dialog (const std::string & *type*)

Shows the load/save dialog window.

Parameters

Either	"SAVE" or "LOAD", will determine the functionality of the window.

Definition at line 341 of file GUI.cpp.

6.41.4 Member Data Documentation

6.41.4.1 BuilderWindow GUI::builder_ `[private]`

Allows the player to place buildings.

Definition at line 226 of file GUI.hpp.

6.41.4.2 Console GUI::console_ `[private]`

Game's developer console.

Definition at line 216 of file GUI.hpp.

6.41.4.3 std::string GUI::curr_tool_ `[private]`

Name of the current tool in the tools subwindow.

Definition at line 201 of file GUI.hpp.

6.41.4.4 Game∗ GUI::game_ `[private]`

Pointer to the game instance used by button event handlers.

Definition at line 206 of file GUI.hpp.

6.41.4.5 GameLog GUI::log_ [private]

Game's log, used to show messages to the player.

Definition at line 221 of file GUI.hpp.

6.41.4.6 CEGUI::Window* GUI::menu_ [private]

Allows easier access to the menu subwindow.

Definition at line 247 of file GUI.hpp.

6.41.4.7 MessageToPlayerWindow GUI::message_ [private]

Allows the game to show a message to the player with any of the following buttons: OK, YES, NO.

Definition at line 253 of file GUI.hpp.

6.41.4.8 OptionsWindow GUI::options_ [private]

Allows to change the resolution, window mode and key bindings.

Definition at line 259 of file GUI.hpp.

6.41.4.9 ResearchWindow GUI::research_ [private]

Game's research window that allows the player to buy buildings, units and spells.

Definition at line 237 of file GUI.hpp.

6.41.4.10 SpellCastingWindow GUI::spell_casting_ [private]

Allows the player to cast spells.

Definition at line 242 of file GUI.hpp.

6.41.4.11 TopBar GUI::top_bar_ [private]

Shows game info at the top of the screen.

Definition at line 231 of file GUI.hpp.

6.41.4.12 EntityTracker GUI::tracker_ [private]

ID of the entity that is currently tracked by the entity viewer.

Definition at line 211 of file GUI.hpp.

6.41.4.13 CEGUI::Window∗ GUI::window_ [private]

Pointer to the root window of the layout.

Definition at line 196 of file GUI.hpp.

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

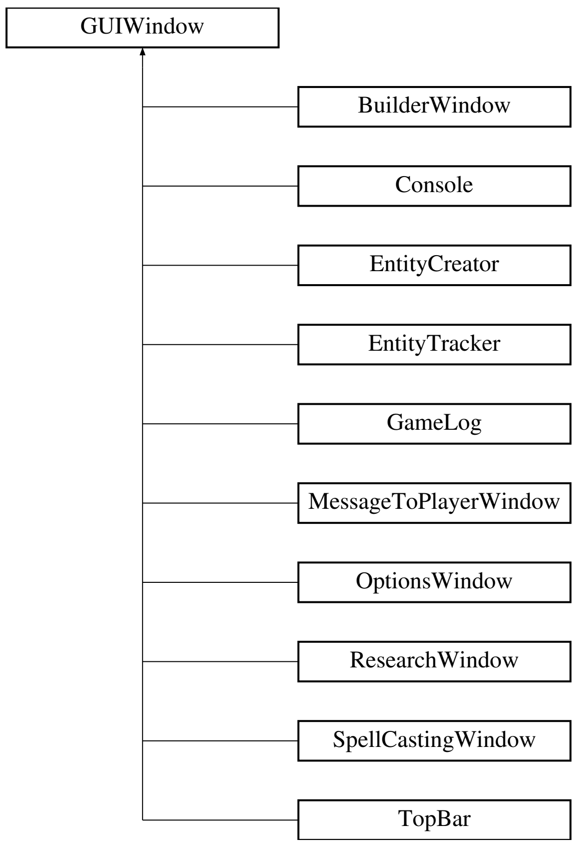
- gui/GUI.hpp
- gui/GUI.cpp

6.42 GUIWindow Class Reference

Abstract class that custom GUI windows inherit from, prevents unnecessary rewriting of common functions (like visibility setting and window_ assignment on init).

```
#include <GUIWindow.hpp>
```

Inheritance diagram for GUIWindow:



Public Member Functions

- GUIWindow ()

  Constructor.
- virtual ∼GUIWindow ()=default

  Destructor.
- void init (CEGUI::Window ∗)

  Initializes the window_ variable and calls the protected init_ function.
- virtual void set_visible (bool)

  Sets the visibolity status of this window.
- bool is_visible () const

  Returns true if the window is visible, false otherwise.

Protected Member Functions

- virtual void init_ ()=0

  Specific init function for each inheriting class.

Protected Attributes

- CEGUI::Window ∗ window_

  Root window.

6.42.1 Detailed Description

Abstract class that custom GUI windows inherit from, prevents unnecessary rewriting of common functions (like visibility setting and window_ assignment on init).

Definition at line 12 of file GUIWindow.hpp.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 GUIWindow::GUIWindow ( )

Constructor.

Definition at line 4 of file GUIWindow.cpp.

6.42.2.2 virtual GUIWindow::∼GUIWindow ( ) `[virtual]`, `[default]`

Destructor.

6.42.3 Member Function Documentation

6.42.3.1 void GUIWindow::init (CEGUI::Window ∗ *w*)

Initializes the window_ variable and calls the protected init_ function.

Definition at line 8 of file GUIWindow.cpp.

6.42.3.2 virtual void GUIWindow::init_ ( ) [protected], [pure virtual]

Specific init function for each inheriting class.

Implemented in SpellCastingWindow, Console, ResearchWindow, BuilderWindow, EntityTracker, EntityCreator, MessageToPlayerWindow, OptionsWindow, GameLog, and TopBar.

6.42.3.3 bool GUIWindow::is_visible ( ) const

Returns true if the window is visible, false otherwise.

Definition at line 19 of file GUIWindow.cpp.

6.42.3.4 void GUIWindow::set_visible (bool *val*) [virtual]

Sets the visibolity status of this window.

Parameters

The	new visibility status.

Reimplemented in Console.

Definition at line 14 of file GUIWindow.cpp.

6.42.4 Member Data Documentation

6.42.4.1 CEGUI::Window∗ GUIWindow::window_ [protected]

Root window.

Definition at line 47 of file GUIWindow.hpp.

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

- gui/GUIWindow.hpp
- gui/GUIWindow.cpp

6.43 util::HAS_GOLD Struct Reference

Tests if a given entity has a gold component.

```
#include <Util.hpp>
```

Public Member Functions

- HAS_GOLD (EntitySystem &)

  Constructor.
- ∼HAS_GOLD ()=default

  Destructor.
- bool operator() (tdt::uint)

  Tests if a given entity has a gold component.

Private Attributes

- EntitySystem & entities_

  Entity system containing all tested entities.

6.43.1 Detailed Description

Tests if a given entity has a gold component.

Definition at line 132 of file Util.hpp.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 util::HAS_GOLD::HAS_GOLD (EntitySystem & *ents*)

Constructor.

Parameters

Entity	system containing all tested entities.

Definition at line 45 of file Util.cpp.

6.43.2.2 util::HAS_GOLD::∼HAS_GOLD ( ) `[default]`

Destructor.

6.43.3 Member Function Documentation

6.43.3.1 bool util::HAS_GOLD::operator() (tdt::uint *id*)

Tests if a given entity has a gold component.

Parameters

ID	of the entity.

Definition at line 49 of file Util.cpp.

6.43.4 Member Data Documentation

6.43.4.1 EntitySystem& util::HAS_GOLD::entities_ [private]

Entity system containing all tested entities.

Definition at line 155 of file Util.hpp.

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

- tools/Util.hpp
- tools/Util.cpp

6.44 util::effect::HEAL_EFFECT Struct Reference

Fully heals the entity it's called on.

```
#include <Effects.hpp>
```

Public Member Functions

- HEAL_EFFECT (EntitySystem &)

  Constructor.
- ∼HEAL_EFFECT ()=default

  Destructor.
- void operator() (tdt::uint)

  Fully heals a given entity.

Private Attributes

- EntitySystem & entities_

  Entity system containing the entities this effect will be called on.

6.44.1 Detailed Description

Fully heals the entity it's called on.

Definition at line 61 of file Effects.hpp.

6.44.2 Constructor & Destructor Documentation

6.44.2.1 util::effect::HEAL_EFFECT::HEAL_EFFECT (**EntitySystem** & *ents*)

Constructor.

Parameters

Entity	system containing the entities this effect will be called on.

Definition at line 16 of file Effects.cpp.

6.44.2.2 util::effect::HEAL_EFFECT::∼HEAL_EFFECT ( ) `[default]`

Destructor.

6.44.3 Member Function Documentation

6.44.3.1 void util::effect::HEAL_EFFECT::operator() (tdt::uint *id*)

Fully heals a given entity.

Parameters

ID	of the entity.

Definition at line 20 of file Effects.cpp.

6.44.4 Member Data Documentation

6.44.4.1 **EntitySystem**& util::effect::HEAL_EFFECT::entities_ `[private]`

Entity system containing the entities this effect will be called on.

Definition at line 86 of file Effects.hpp.

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

- tools/Effects.hpp
- tools/Effects.cpp

6.45 HealthComponent Struct Reference

Holds info about an entity's health and regeneration.

```
#include <Components.hpp>
```

Public Member Functions

- **HealthComponent** (tdt::uint max=0, tdt::uint reg=0, tdt::uint def=0, bool a=true)
- **HealthComponent** (const HealthComponent &)=default
- **HealthComponent** (HealthComponent &&)=default
- HealthComponent & **operator=** (const HealthComponent &)=default
- HealthComponent & **operator=** (HealthComponent &&)=default

Public Attributes

- tdt::uint **curr_hp**
- tdt::uint **max_hp**
- tdt::uint **regen**
- tdt::uint **defense**
- bool **alive**

Static Public Attributes

- static constexpr int **type** = 1

6.45.1 Detailed Description

Holds info about an entity's health and regeneration.

Definition at line 55 of file Components.hpp.

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

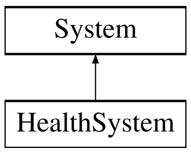
- Components.hpp

6.46 HealthSystem Class Reference

System that manages the regeneration and health of entities on each frame.

```
#include <HealthSystem.hpp>
```

Inheritance diagram for HealthSystem:



Public Member Functions

- HealthSystem (EntitySystem &)

  Constructor.
- ∼HealthSystem ()=default

  Destructor.
- void update (tdt::real) override

  Updates a the system by checking every valid entity's health (or death) status and applying health regeneration if necessary.
- void update_regen (tdt::real)

  Adds one to the regeneration timer, simulating continuous regeration, should be called once per frame before the update method.
- void set_regen_period (tdt::real)

  Sets the amount of time it takes for one regeneration tick to happen (in seconds).
- tdt::real get_regen_period () const

  Returns the amount of time it takes for one regeneration tick to happen (in seconds).

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- tdt::real regen_timer_

  Amount of frames since the last regeneration tick.
- tdt::real regen_period_

  Amount of frames per regeneration period.
- bool regen_

  True if this frame's update should renerate health.

6.46.1 Detailed Description

System that manages the regeneration and health of entities on each frame.

Definition at line 11 of file HealthSystem.hpp.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 HealthSystem::HealthSystem (EntitySystem & *ent*)

Constructor.

Parameters

Reference	to the game's entity system.

Definition at line 6 of file HealthSystem.cpp.

6.46.2.2 HealthSystem::∼HealthSystem ( ) [default]

Destructor.

6.46.3 Member Function Documentation

6.46.3.1 tdt::real HealthSystem::get_regen_period ( ) const

Returns the amount of time it takes for one regeneration tick to happen (in seconds).

Definition at line 43 of file HealthSystem.cpp.

6.46.3.2 void HealthSystem::set_regen_period (tdt::real *val*)

Sets the amount of time it takes for one regeneration tick to happen (in seconds).

Parameters

The	new regen period.

Definition at line 38 of file HealthSystem.cpp.

6.46.3.3 void HealthSystem::update (tdt::real *delta*) [override], [virtual]

Updates a the system by checking every valid entity's health (or death) status and applying health regeneration if necessary.

Parameters

Time	since the last frame.

Implements System.

Definition at line 11 of file HealthSystem.cpp.

6.46.3.4 void HealthSystem::update_regen (tdt::real *delta*)

Adds one to the regeneration timer, simulating continuous regeration, should be called once per frame before the update method.

Parameters

Time	since the last frame.

Definition at line 23 of file HealthSystem.cpp.

6.46.4 Member Data Documentation

6.46.4.1 EntitySystem& HealthSystem::entities_ [private]

Reference to the game's entity system.

Definition at line 56 of file HealthSystem.hpp.

6.46.4.2 bool HealthSystem::regen_ [private]

True if this frame's update should renerate health.

Definition at line 71 of file HealthSystem.hpp.

6.46.4.3 tdt::real HealthSystem::regen_period_ [private]

Amount of frames per regeneration period.

Definition at line 66 of file HealthSystem.hpp.

6.46.4.4 tdt::real HealthSystem::regen_timer_ [private]

Amount of frames since the last regeneration tick.

Definition at line 61 of file HealthSystem.hpp.

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

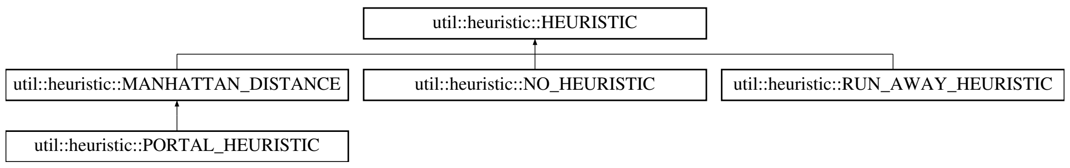
- systems/HealthSystem.hpp
- systems/HealthSystem.cpp

6.47 util::heuristic::HEURISTIC Struct Reference

Abstract parent of all heuristics.

`#include <PathfindingAlgorithms.hpp>`

Inheritance diagram for util::heuristic::HEURISTIC:


util::heuristic::HEURISTIC
util::heuristic::MANHATTAN_DISTANCE
util::heuristic::NO_HEURISTIC
util::heuristic::RUN_AWAY_HEURISTIC
util::heuristic::PORTAL_HEURISTIC

Public Member Functions

- **HEURISTIC** (EntitySystem &ents)
- virtual tdt::real **get_cost** (tdt::uint id1, tdt::uint id2)=0

Protected Attributes

- EntitySystem & **entities_**

6.47.1 Detailed Description

Abstract parent of all heuristics.

Inheritance hierarchy used instead of static functions (like in the case of PATH_TYPEs) to allow for a heuristic to have a state.

Definition at line 188 of file PathfindingAlgorithms.hpp.

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

- tools/PathfindingAlgorithms.hpp

6.48 HomingComponent Struct Reference

Used for projectiles that are supposed to follow a target and deal damage when they hit it.

```
#include <Components.hpp>
```

Public Member Functions

- **HomingComponent** (tdt::uint s=Component::NO_ENTITY, tdt::uint t=Component::NO_ENTITY, tdt::uint d=0)
- **HomingComponent** (const HomingComponent &)=default
- **HomingComponent** (HomingComponent &&)=default
- HomingComponent & **operator=** (const HomingComponent &)=default
- HomingComponent & **operator=** (HomingComponent &&)=default

Public Attributes

- tdt::uint **source**
- tdt::uint **target**
- tdt::uint **dmg**

Static Public Attributes

- static constexpr int **type** = 18

6.48.1 Detailed Description

Used for projectiles that are supposed to follow a target and deal damage when they hit it.

Definition at line 462 of file Components.hpp.

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

- Components.hpp

6.49 InputComponent Struct Reference

Holds info related to direct player input applied to an entity.

```
#include <Components.hpp>
```

Public Member Functions

- **InputComponent** (std::string &&handler="ERROR.input_handler")
- **InputComponent** (const InputComponent &)=default
- **InputComponent** (InputComponent &&)=default
- InputComponent & **operator=** (const InputComponent &)=default
- InputComponent & **operator=** (InputComponent &&)=default

Public Attributes

- std::string **input_handler**

Static Public Attributes

- static constexpr int **type** = 7

6.49.1 Detailed Description

Holds info related to direct player input applied to an entity.

Definition at line 207 of file Components.hpp.

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

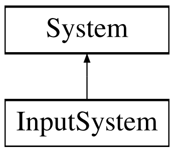
- Components.hpp

6.50 InputSystem Class Reference

System handling entities controlled by the player's keyboard input and changing the game's view mode (between 1st and 3rd person).

```
#include <InputSystem.hpp>
```

Inheritance diagram for InputSystem:



Public Member Functions

- InputSystem (EntitySystem &, OIS::Keyboard &, Ogre::Camera &)

  Constructor.
- ∼InputSystem ()=default

  Destructor.
- void update (tdt::real) override

  Handles the input for the entity that is currently in the first person mode.
- bool is_first_person () const

  Returns true if the game is in the first person mode, returns false otherwise.
- void set_first_person (bool, tdt::uint=0)

  Changes the first person mode status for an entity and also loads an InputComponent for it if it does not have it but has an AIComponent holding information about it's input_handler method.
- void rebind (int, int)

  Rebinds a given key with an OIS key number.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- bool first_person_

  Determines if the first person view is turned on.
- tdt::uint first_person_id_

  If the first person view is turned on, this holds ID of the entity being controlled.
- OIS::Keyboard & keyboard_

  Reference to the keyboard being used.
- int KEY_UP

  Current keybindings, allow rebinding.
- int **KEY_DOWN**
- int **KEY_LEFT**
- int **KEY_RIGHT**
- Ogre::Camera & cam_

  Reference to the game's main camera.
- Ogre::Vector3 cam_position_

  Backup of the camera info for easy restoring once the game leaves the first person view.

- Ogre::Quaternion **cam_orientation_**
- std::unique_ptr< AIComponent > ai_backup_

  Backup of the AI component when entering first person view.
- std::unique_ptr< TaskHandlerComponent > task_backup_

  Backup of the task component when entering first person view.
- bool delete_input_

  Determines if the InputComponent of the entity being controlled in the first person view should be deleted once the game changes back to third person view.

6.50.1 Detailed Description

System handling entities controlled by the player's keyboard input and changing the game's view mode (between 1st and 3rd person).

Definition at line 16 of file InputSystem.hpp.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 InputSystem::InputSystem (EntitySystem & *ents*, OIS::Keyboard & *key*, Ogre::Camera & *cam*)

Constructor.

Parameters

Reference	to the game's EntitySystem instance.
Reference	to the keyboard being used.
Reference	to the camera for it's manipulation during the 1st person mode.

Definition at line 5 of file InputSystem.cpp.

6.50.2.2 InputSystem::∼InputSystem ( ) [default]

Destructor.

6.50.3 Member Function Documentation

6.50.3.1 bool InputSystem::is_first_person ( ) const

Returns true if the game is in the first person mode, returns false otherwise.

Definition at line 56 of file InputSystem.cpp.

6.50.3.2 void InputSystem::rebind (int *key*, int *new_key*)

Rebinds a given key with an OIS key number.

Use OIS::KC_W, OIS::KC_S, OIS::KC_A and OIS::KC_D to determine which key should be rebinded.

Parameters

Key	to be rebinded.
The	new key.

Definition at line 160 of file InputSystem.cpp.

6.50.3.3 void InputSystem::set_first_person (bool *on_off*, tdt::uint *id* = 0)

Changes the first person mode status for an entity and also loads an InputComponent for it if it does not have it but has an AIComponent holding information about it's input_handler method.

(Backups the AIComponent in such cases and restores it when the entity leaves first person mode.)

Definition at line 61 of file InputSystem.cpp.

6.50.3.4 void InputSystem::update (tdt::real *delta*) [override], [virtual]

Handles the input for the entity that is currently in the first person mode.

Parameters

Time	since the last frame.

Implements System.

Definition at line 12 of file InputSystem.cpp.

6.50.4 Member Data Documentation

6.50.4.1 std::unique_ptr<AIComponent> InputSystem::ai_backup_ [private]

Backup of the AI component when entering first person view.

Definition at line 101 of file InputSystem.hpp.

6.50.4.2 Ogre::Camera& InputSystem::cam_ [private]

Reference to the game's main camera.

Definition at line 89 of file InputSystem.hpp.

6.50.4.3 Ogre::Vector3 InputSystem::cam_position_ [private]

Backup of the camera info for easy restoring once the game leaves the first person view.

Definition at line 95 of file InputSystem.hpp.

6.50.4.4 bool InputSystem::delete_input_ [private]

Determines if the InputComponent of the entity being controlled in the first person view should be deleted once the game changes back to third person view.

Definition at line 112 of file InputSystem.hpp.

6.50.4.5 EntitySystem& InputSystem::entities_ [private]

Reference to the game's entity system.

Definition at line 64 of file InputSystem.hpp.

6.50.4.6 bool InputSystem::first_person_ [private]

Determines if the first person view is turned on.

Definition at line 69 of file InputSystem.hpp.

6.50.4.7 tdt::uint InputSystem::first_person_id_ [private]

If the first person view is turned on, this holds ID of the entity being controlled.

Definition at line 74 of file InputSystem.hpp.

6.50.4.8 int InputSystem::KEY_UP [private]

Current keybindings, allow rebinding.

Definition at line 84 of file InputSystem.hpp.

6.50.4.9 OIS::Keyboard& InputSystem::keyboard_ [private]

Reference to the keyboard being used.

Definition at line 79 of file InputSystem.hpp.

6.50.4.10 std::unique_ptr<TaskHandlerComponent> InputSystem::task_backup_ [private]

Backup of the task component when entering first person view.

Definition at line 106 of file InputSystem.hpp.

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

- systems/InputSystem.hpp
- systems/InputSystem.cpp

6.51 util::IS_ENEMY Struct Reference

Tests if if the entity it is called on is an enemy of the entity specified in it's constructor.

```
#include <Util.hpp>
```

Public Member Functions

- IS_ENEMY (EntitySystem &, tdt::uint)

  Constructor.
- ∼IS_ENEMY ()=default

  Destructor.
- bool operator() (tdt::uint)

  Tests if a given entity is an enemy of the entity specified in the constructor.

Private Attributes

- FACTION enemy_faction_

  Faction that is hostile towards the entity performing the search.
- EntitySystem & entities_

  Entity system containing all tested entities.

6.51.1 Detailed Description

Tests if if the entity it is called on is an enemy of the entity specified in it's constructor.

Definition at line 19 of file Util.hpp.

6.51.2 Constructor & Destructor Documentation

6.51.2.1 util::IS_ENEMY::IS_ENEMY (EntitySystem & *ents*, tdt::uint *id*)

Constructor.

Parameters

Entity	system containing all tested entities.
ID	of the entity towards which others are tested for the enemy status.

Definition at line 5 of file Util.cpp.

6.51.2.2 util::IS_ENEMY::∼IS_ENEMY ( ) `[default]`

Destructor.

6.51.3 Member Function Documentation

6.51.3.1 bool util::IS_ENEMY::operator() (tdt::uint *id*)

Tests if a given entity is an enemy of the entity specified in the constructor.

Parameters

ID	of the entity.

Definition at line 15 of file Util.cpp.

6.51.4 Member Data Documentation

6.51.4.1 FACTION util::IS_ENEMY::enemy_faction_ `[private]`

Faction that is hostile towards the entity performing the search.

Definition at line 45 of file Util.hpp.

6.51.4.2 EntitySystem& util::IS_ENEMY::entities_ `[private]`

Entity system containing all tested entities.

Definition at line 50 of file Util.hpp.

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

- tools/Util.hpp
- tools/Util.cpp

6.52 util::IS_FRIENDLY Struct Reference

Tests if if the entity it is called on is a friend of the entity specified in it's constructor.

```
#include <Util.hpp>
```

Public Member Functions

- IS_FRIENDLY (EntitySystem &, tdt::uint)

  Constructor.
- ∼IS_FRIENDLY ()=default

  Destructor.
- bool operator() (tdt::uint)

  Tests if a given entity is a friend of the entity specified in the constructor.

Private Attributes

- FACTION faction_

  Faction that is friendly towards the entity performing the search.
- EntitySystem & entities_

  Entity system containing all tested entities.

6.52.1 Detailed Description

Tests if if the entity it is called on is a friend of the entity specified in it's constructor.

Definition at line 57 of file Util.hpp.

6.52.2 Constructor & Destructor Documentation

6.52.2.1 util::IS_FRIENDLY::IS_FRIENDLY (**EntitySystem** & *ents*, tdt::uint *id*)

Constructor.

Parameters

Entity	system containing all tested entities.
ID	of the entity towards which others are tested for the friendly status.

Definition at line 21 of file Util.cpp.

6.52.2.2 util::IS_FRIENDLY::∼IS_FRIENDLY ( ) `[default]`

Destructor.

6.52.3 Member Function Documentation

6.52.3.1 bool util::IS_FRIENDLY::operator() (tdt::uint *id*)

Tests if a given entity is a friend of the entity specified in the constructor.

Parameters

ID	of the entity.

Definition at line 25 of file Util.cpp.

6.52.4 Member Data Documentation

6.52.4.1 EntitySystem& util::IS_FRIENDLY::entities_ `[private]`

Entity system containing all tested entities.

Definition at line 88 of file Util.hpp.

6.52.4.2 FACTION util::IS_FRIENDLY::faction_ `[private]`

Faction that is friendly towards the entity performing the search.

Definition at line 83 of file Util.hpp.

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

- tools/Util.hpp
- tools/Util.cpp

6.53 util::IS_FRIENDLY_OR_NEUTRAL Struct Reference

Tests if if the entity it is called on is a friend of or neutral to the entity specified in it's constructor.

`#include <Util.hpp>`

Public Member Functions

- IS_FRIENDLY_OR_NEUTRAL (EntitySystem &, tdt::uint)
  Constructor.
- ∼IS_FRIENDLY_OR_NEUTRAL ()=default
  Destructor.
- bool operator() (tdt::uint)
  Tests if a given entity is a friend of or neutral to the entity specified in the constructor.

Private Attributes

- FACTION faction_
  Faction that is friendly towards the entity performing the search.
- EntitySystem & entities_
  Entity system containing all tested entities.

6.53.1 Detailed Description

Tests if if the entity it is called on is a friend of or neutral to the entity specified in it's constructor.

Definition at line 95 of file Util.hpp.

6.53.2 Constructor & Destructor Documentation

6.53.2.1 util::IS_FRIENDLY_OR_NEUTRAL::IS_FRIENDLY_OR_NEUTRAL (EntitySystem & *ents*, tdt::uint *id*)

Constructor.

Parameters

Entity	system containing all tested entities.
ID	of the entity towards which others are tested for the friendly/neutrality status.

Definition at line 30 of file Util.cpp.

6.53.2.2 util::IS_FRIENDLY_OR_NEUTRAL::∼IS_FRIENDLY_OR_NEUTRAL ( ) [default]

Destructor.

6.53.3 Member Function Documentation

6.53.3.1 bool util::IS_FRIENDLY_OR_NEUTRAL::operator() (tdt::uint *id*)

Tests if a given entity is a friend of or neutral to the entity specified in the constructor.

Parameters

ID	of the entity.

Definition at line 34 of file Util.cpp.

6.53.4 Member Data Documentation

6.53.4.1 EntitySystem& util::IS_FRIENDLY_OR_NEUTRAL::entities_ [private]

Entity system containing all tested entities.

Definition at line 126 of file Util.hpp.

6.53.4.2 FACTION util::IS_FRIENDLY_OR_NEUTRAL::faction_ [private]

Faction that is friendly towards the entity performing the search.

Definition at line 121 of file Util.hpp.

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

- tools/Util.hpp
- tools/Util.cpp

6.54 util::IS_GOLD_VAULT Struct Reference

Tests if a given entity is of friendly faction, has structure component and has gold component (that is, it's a gold vault).

```
#include <Util.hpp>
```

Public Member Functions

- IS_GOLD_VAULT (EntitySystem &)

  Constructor.
- ∼IS_GOLD_VAULT ()=default

  Destructor.
- bool operator() (tdt::uint)

  Tests a given entity.

Private Attributes

- EntitySystem & entities_

  Entity system that contains all tested entities.

6.54.1 Detailed Description

Tests if a given entity is of friendly faction, has structure component and has gold component (that is, it's a gold vault).

Definition at line 163 of file Util.hpp.

6.54.2 Constructor & Destructor Documentation

6.54.2.1 util::IS_GOLD_VAULT::IS_GOLD_VAULT (**EntitySystem &** *ents*)

Constructor.

Parameters

Entity	system that contains all tested entities.

Definition at line 110 of file Util.cpp.

6.54.2.2 util::IS_GOLD_VAULT::∼IS_GOLD_VAULT ( ) `[default]`

Destructor.

6.54.3 Member Function Documentation

6.54.3.1 bool util::IS_GOLD_VAULT::operator() (tdt::uint *id*)

Tests a given entity.

Parameters

ID	of the entity.

Definition at line 114 of file Util.cpp.

6.54.4 Member Data Documentation

6.54.4.1 EntitySystem& util::IS_GOLD_VAULT::entities_ `[private]`

Entity system that contains all tested entities.

Definition at line 186 of file Util.hpp.

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

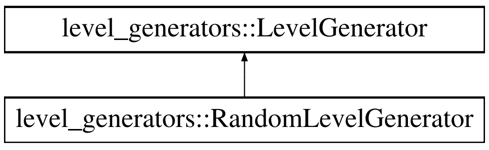
- tools/Util.hpp
- tools/Util.cpp

6.55 level_generators::LevelGenerator Class Reference

Abstract parent class of all level generators, allows for different level generators used to create levels with minimal effort.

`#include <LevelGenerators.hpp>`

Inheritance diagram for level_generators::LevelGenerator:


level_generators::LevelGenerator
level_generators::RandomLevelGenerator

Public Member Functions

- LevelGenerator (EntitySystem &, tdt::uint)

  Constructor.
- virtual ~LevelGenerator ()=default

  Destructor.
- virtual void generate (tdt::uint, tdt::uint, WaveSystem &)=0

  Generates a level with the given dimensions.

Protected Attributes

- EntitySystem & entities_

  Entity system that contains the level's entities.
- tdt::uint cycles_

  Number of cycles done while generating the world.

6.55.1 Detailed Description

Abstract parent class of all level generators, allows for different level generators used to create levels with minimal effort.

Definition at line 20 of file LevelGenerators.hpp.

6.55.2 Constructor & Destructor Documentation

6.55.2.1 level_generators::LevelGenerator::LevelGenerator (EntitySystem & *ents*, tdt::uint *c*)

Constructor.

Parameters

Entity	system that contains the level's entities.
Number	of iterations done while generating.

Definition at line 10 of file LevelGenerators.cpp.

6.55.2.2 virtual level_generators::LevelGenerator::∼LevelGenerator ( ) `[virtual]`, `[default]`

Destructor.

6.55.3 Member Function Documentation

6.55.3.1 virtual void level_generators::LevelGenerator::generate (tdt::uint , tdt::uint , WaveSystem &) `[pure virtual]`

Generates a level with the given dimensions.

Parameters

Width	of the level.
Height	of the level.
Wave	system that will have it's spawn nodes set.

Implemented in level_generators::RandomLevelGenerator.

6.55.4 Member Data Documentation

6.55.4.1 tdt::uint level_generators::LevelGenerator::cycles_ `[protected]`

Number of cycles done while generating the world.

Definition at line 52 of file LevelGenerators.hpp.

6.55.4.2 EntitySystem& level_generators::LevelGenerator::entities_ `[protected]`

Entity system that contains the level's entities.

Definition at line 47 of file LevelGenerators.hpp.

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

- tools/LevelGenerators.hpp
- tools/LevelGenerators.cpp

6.56 LightComponent Struct Reference

Allows an entity to emit light to it's surrounding area.

`#include <Components.hpp>`

Public Member Functions

- **LightComponent** (const LightComponent &)=default
- **LightComponent** (LightComponent &&)=default
- LightComponent & **operator=** (const LightComponent &)=default
- LightComponent & **operator=** (LightComponent &&)=default

Public Attributes

- Ogre::SceneNode ∗ **node**
- Ogre::Light ∗ **light**

Static Public Attributes

- static constexpr int **type** = 36

6.56.1 Detailed Description

Allows an entity to emit light to it's surrounding area.

Definition at line 841 of file Components.hpp.

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

- Components.hpp

6.57 LimitedLifeSpanComponent Struct Reference

Allows to create entities that are automatically killed (summons) after a certain amount of time has passed (lifespan).

`#include <Components.hpp>`

Public Member Functions

- **LimitedLifeSpanComponent** (tdt::real max=0.f)
- **LimitedLifeSpanComponent** (const LimitedLifeSpanComponent &)=default
- **LimitedLifeSpanComponent** (LimitedLifeSpanComponent &&)=default
- LimitedLifeSpanComponent & **operator=** (const LimitedLifeSpanComponent &)=default
- LimitedLifeSpanComponent & **operator=** (LimitedLifeSpanComponent &&)=default

Public Attributes

- tdt::real **curr_time**
- tdt::real **max_time**

Static Public Attributes

- static constexpr int **type** = 33

6.57.1 Detailed Description

Allows to create entities that are automatically killed (summons) after a certain amount of time has passed (lifespan).

Definition at line 783 of file Components.hpp.

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

- Components.hpp

6.58 util::effect::LOWER_SPEED_EFFECT Struct Reference

Halves the speed of the entity it's called on for a given time period.

`#include <Effects.hpp>`

Public Member Functions

- LOWER_SPEED_EFFECT (EntitySystem &, tdt::real)

  Constructor.
- ∼LOWER_SPEED_EFFECT ()=default

  Destructor.
- void operator() (tdt::uint)

  Halves the speed of a given entity.

Private Attributes

- EntitySystem & entities_

    Entity system containing the entities this effect will be called on.
- tdt::real time_

    The time period that has to pass before the speed of the affected entities gets restored.

6.58.1 Detailed Description

Halves the speed of the entity it's called on for a given time period.

Definition at line 93 of file Effects.hpp.

6.58.2 Constructor & Destructor Documentation

6.58.2.1 util::effect::LOWER_SPEED_EFFECT::LOWER_SPEED_EFFECT (EntitySystem & *ents,* tdt::real *time*)

Constructor.

Parameters

Entity	system containing the entities this effect will be called on.
The	time period before the speed is restored.

Definition at line 25 of file Effects.cpp.

6.58.2.2 util::effect::LOWER_SPEED_EFFECT::∼LOWER_SPEED_EFFECT ( ) [default]

Destructor.

6.58.3 Member Function Documentation

6.58.3.1 void util::effect::LOWER_SPEED_EFFECT::operator() (tdt::uint *id*)

Halves the speed of a given entity.

Parameters

ID	of the entity.

Definition at line 29 of file Effects.cpp.

6.58.4 Member Data Documentation

6.58.4.1 EntitySystem& util::effect::LOWER_SPEED_EFFECT::entities_ `[private]`

Entity system containing the entities this effect will be called on.

Definition at line 119 of file Effects.hpp.

6.58.4.2 tdt::real util::effect::LOWER_SPEED_EFFECT::time_ `[private]`

The time period that has to pass before the speed of the affected entities gets restored.

Definition at line 125 of file Effects.hpp.

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

- tools/Effects.hpp
- tools/Effects.cpp

6.59 LuaInterface Class Reference

Class that creates an interface between engine (C++) and logic (Lua) code.

```
#include <LuaInterface.hpp>
```

Public Member Functions

- LuaInterface ()=delete

  This is a static class, so all constructors are deleted.
- **LuaInterface** (const LuaInterface &)=delete
- **LuaInterface** (LuaInterface &&)=delete

Static Public Member Functions

- static void init (Game ∗)

  Sets the lua_this pointer and registers all C++ API functions to Lua.

Static Private Member Functions

- static int lua_get_avg_fps (lpp::Script::state)
- static int **lua_get_fps** (lpp::Script::state)
- static int **lua_print** (lpp::Script::state)
- static int **lua_set_game_state** (lpp::Script::state)
- static int **lua_toggle_bounding_boxes** (lpp::Script::state)
- static int **lua_toggle_camera_free_mode** (lpp::Script::state)
- static int **lua_toggle_entity_creator** (lpp::Script::state)
- static int **lua_list_selected** (lpp::Script::state)
- static int **lua_destroy_selected** (lpp::Script::state)
- static int **lua_kill_selected** (lpp::Script::state)
- static int **lua_list_components_of** (lpp::Script::state)
- static int **lua_load** (lpp::Script::state)
- static int **lua_reload_all** (lpp::Script::state)
- static int **lua_save_game** (lpp::Script::state)
- static int **lua_load_game** (lpp::Script::state)
- static int **lua_get_cursor_position** (lpp::Script::state)
- static int **lua_can_place_when_game_paused** (lpp::Script::state)
- static int **lua_toggle_placing_when_game_paused** (lpp::Script::state)
- static int **lua_new_game** (lpp::Script::state)
- static int **lua_create_empty_level** (lpp::Script::state)
- static int **lua_reset_unlocks** (lpp::Script::state)
- static int **lua_get_random** (lpp::Script::state)
- static int **lua_set_key_bind** (lpp::Script::state)
- static int **lua_get_first_selected** (lpp::Script::state)
- static int **lua_get_enemies** (lpp::Script::state)
- static int **lua_get_friends** (lpp::Script::state)
- static int **lua_set_throne_id** (lpp::Script::state)
- static int **lua_get_throne_id** (lpp::Script::state)
- static int **lua_command_to_mine** (lpp::Script::state)
- static int **lua_command_to_attack** (lpp::Script::state)
- static int **lua_command_to_reposition** (lpp::Script::state)
- static int **lua_command_to_return_gold** (lpp::Script::state)
- static int **lua_command_to_fall_back** (lpp::Script::state)
- static int **lua_get_enum_direction** (lpp::Script::state)
- static int **lua_get_node_in_dir** (lpp::Script::state)
- static int **lua_set_mesh** (lpp::Script::state)
- static int **lua_set_material** (lpp::Script::state)
- static int **lua_set_visible** (lpp::Script::state)
- static int **lua_set_manual_scaling** (lpp::Script::state)
- static int **lua_set_scale** (lpp::Script::state)
- static int **lua_get_mesh** (lpp::Script::state)
- static int **lua_get_material** (lpp::Script::state)
- static int **lua_is_visible** (lpp::Script::state)
- static int **lua_get_manual_scaling** (lpp::Script::state)
- static int **lua_get_scale** (lpp::Script::state)
- static int **lua_look_at** (lpp::Script::state)
- static int **lua_rotate_x** (lpp::Script::state)
- static int **lua_rotate_y** (lpp::Script::state)
- static int **lua_rotate_z** (lpp::Script::state)
- static int **lua_collide** (lpp::Script::state)
- static int **lua_set_query_flags** (lpp::Script::state)
- static int **lua_get_query_flags** (lpp::Script::state)
- static int **lua_apply_scale** (lpp::Script::state)

- static int **lua_set_graphics_update_period** (lpp::Script::state)
- static int **lua_get_graphics_update_period** (lpp::Script::state)
- static int **lua_create_entity** (lpp::Script::state)
- static int **lua_destroy_entity** (lpp::Script::state)
- static int **lua_add_component** (lpp::Script::state)
- static int **lua_delete_component** (lpp::Script::state)
- static int **lua_init_graphics_component** (lpp::Script::state)
- static int **lua_list_entity_tables** (lpp::Script::state)
- static int **lua_place_entity** (lpp::Script::state)
- static int **lua_register_entity** (lpp::Script::state)
- static int **lua_exists** (lpp::Script::state)
- static int **lua_kill_entity** (lpp::Script::state)
- static int **lua_has_component** (lpp::Script::state)
- static int **lua_entity_reset_state** (lpp::Script::state)
- static int **lua_set_position** (lpp::Script::state)
- static int **lua_get_position** (lpp::Script::state)
- static int **lua_is_solid** (lpp::Script::state)
- static int **lua_set_solid** (lpp::Script::state)
- static int **lua_set_half_height** (lpp::Script::state)
- static int **lua_get_half_height** (lpp::Script::state)
- static int **lua_get_distance** (lpp::Script::state)
- static int **lua_get_angle** (lpp::Script::state)
- static int **lua_get_angle_between** (lpp::Script::state)
- static int **lua_set_2d_position** (lpp::Script::state)
- static int **lua_get_2d_position** (lpp::Script::state)
- static int **lua_move_to** (lpp::Script::state)
- static int **lua_move** (lpp::Script::state)
- static int **lua_can_move_to** (lpp::Script::state)
- static int **lua_get_speed_modifier** (lpp::Script::state)
- static int **lua_set_speed_modifier** (lpp::Script::state)
- static int **lua_dir_to** (lpp::Script::state)
- static int **lua_get_dir** (lpp::Script::state)
- static int **lua_get_dir_back** (lpp::Script::state)
- static int **lua_get_dir_left** (lpp::Script::state)
- static int **lua_get_dir_right** (lpp::Script::state)
- static int **lua_set_original_speed** (lpp::Script::state)
- static int **lua_get_original_speed** (lpp::Script::state)
- static int **lua_reset_speed** (lpp::Script::state)
- static int **lua_set_health** (lpp::Script::state)
- static int **lua_get_health** (lpp::Script::state)
- static int **lua_add_health** (lpp::Script::state)
- static int **lua_sub_health** (lpp::Script::state)
- static int **lua_heal** (lpp::Script::state)
- static int **lua_buff** (lpp::Script::state)
- static int **lua_debuff** (lpp::Script::state)
- static int **lua_set_defense** (lpp::Script::state)
- static int **lua_get_defense** (lpp::Script::state)
- static int **lua_add_defense** (lpp::Script::state)
- static int **lua_sub_defense** (lpp::Script::state)
- static int **lua_set_regen** (lpp::Script::state)
- static int **lua_get_regen** (lpp::Script::state)
- static int **lua_set_alive** (lpp::Script::state)
- static int **lua_is_alive** (lpp::Script::state)
- static int **lua_ubercharge** (lpp::Script::state)
- static int **lua_set_regen_period** (lpp::Script::state)

- static int **lua_get_regen_period** (lpp::Script::state)
- static int **lua_get_blueprint** (lpp::Script::state)
- static int **lua_get_state** (lpp::Script::state)
- static int **lua_get_faction** (lpp::Script::state)
- static int **lua_set_blueprint** (lpp::Script::state)
- static int **lua_set_state** (lpp::Script::state)
- static int **lua_set_faction** (lpp::Script::state)
- static int **lua_set_update_period** (lpp::Script::state)
- static int **lua_get_update_period** (lpp::Script::state)
- static int **lua_force_update** (lpp::Script::state)
- static int **lua_get_faction_name** (lpp::Script::state)
- static int **lua_set_input_handler** (lpp::Script::state)
- static int **lua_get_input_handler** (lpp::Script::state)
- static int **lua_toggle_first_person** (lpp::Script::state)
- static int **lua_add_node** (lpp::Script::state)
- static int **lua_get_node** (lpp::Script::state)
- static int **lua_get_node_from_position** (lpp::Script::state)
- static int **lua_create_grid_graphics** (lpp::Script::state)
- static int **lua_delete_grid_graphics** (lpp::Script::state)
- static int **lua_toggle_grid_visible** (lpp::Script::state)
- static int **lua_is_free** (lpp::Script::state)
- static int **lua_set_free** (lpp::Script::state)
- static int **lua_set_free_selected** (lpp::Script::state)
- static int **lua_pathfind** (lpp::Script::state)
- static int **lua_pop_first_path_node** (lpp::Script::state)
- static int **lua_pop_last_path_node** (lpp::Script::state)
- static int **lua_path_queue_empty** (lpp::Script::state)
- static int **lua_clear_path** (lpp::Script::state)
- static int **lua_set_pathfinding_blueprint** (lpp::Script::state)
- static int **lua_get_pathfinding_blueprint** (lpp::Script::state)
- static int **lua_create_graph** (lpp::Script::state)
- static int **lua_set_resident** (lpp::Script::state)
- static int **lua_get_resident** (lpp::Script::state)
- static int **lua_add_residences** (lpp::Script::state)
- static int **lua_add_residence** (lpp::Script::state)
- static int **lua_set_radius** (lpp::Script::state)
- static int **lua_set_walk_through** (lpp::Script::state)
- static int **lua_is_walk_throuth** (lpp::Script::state)
- static int **lua_place_at_random_free_node** (lpp::Script::state)
- static int **lua_distribute_to_adjacent_free_nodes** (lpp::Script::state)
- static int **lua_get_random_free_node** (lpp::Script::state)
- static int **lua_set_portal_neighbour** (lpp::Script::state)
- static int **lua_get_next_pathfinding_node** (lpp::Script::state)
- static int **lua_get_target_pathfinding_node** (lpp::Script::state)
- static int **lua_pathfinding_skip_next_node** (lpp::Script::state)
- static int **lua_pathfinding_after_next_node** (lpp::Script::state)
- static int **lua_add_task** (lpp::Script::state)
- static int **lua_add_priority_task** (lpp::Script::state)
- static int **lua_cancel_task** (lpp::Script::state)
- static int **lua_create_task** (lpp::Script::state)
- static int **lua_list_tasks_of** (lpp::Script::state)
- static int **lua_task_possible** (lpp::Script::state)
- static int **lua_task_type_possibe** (lpp::Script::state)
- static int **lua_clear_task_queue** (lpp::Script::state)
- static int **lua_set_task_source** (lpp::Script::state)

- static int **lua_get_task_source** (lpp::Script::state)
- static int **lua_set_task_target** (lpp::Script::state)
- static int **lua_get_task_target** (lpp::Script::state)
- static int **lua_set_task_type** (lpp::Script::state)
- static int **lua_get_task_type** (lpp::Script::state)
- static int **lua_add_possible_task** (lpp::Script::state)
- static int **lua_delete_possible_task** (lpp::Script::state)
- static int **lua_set_task_handling_blueprint** (lpp::Script::state)
- static int **lua_get_task_handling_blueprint** (lpp::Script::state)
- static int **lua_set_task_complete** (lpp::Script::state)
- static int **lua_is_task_complete** (lpp::Script::state)
- static int **lua_task_clear** (lpp::Script::state)
- static int **lua_set_combat_target** (lpp::Script::state)
- static int **lua_get_combat_target** (lpp::Script::state)
- static int **lua_set_range** (lpp::Script::state)
- static int **lua_get_range** (lpp::Script::state)
- static int **lua_set_dmg_range** (lpp::Script::state)
- static int **lua_get_dmg_range** (lpp::Script::state)
- static int **lua_get_dmg** (lpp::Script::state)
- static int **lua_set_cooldown** (lpp::Script::state)
- static int **lua_get_cooldown** (lpp::Script::state)
- static int **lua_set_atk_type** (lpp::Script::state)
- static int **lua_get_atk_type** (lpp::Script::state)
- static int **lua_set_homing_source** (lpp::Script::state)
- static int **lua_get_homing_source** (lpp::Script::state)
- static int **lua_set_homing_target** (lpp::Script::state)
- static int **lua_get_homing_target** (lpp::Script::state)
- static int **lua_set_homing_dmg** (lpp::Script::state)
- static int **lua_get_homing_dmg** (lpp::Script::state)
- static int **lua_closest_enemy_in_sight** (lpp::Script::state)
- static int **lua_closest_friendly_in_sight** (lpp::Script::state)
- static int **lua_closest_enemy** (lpp::Script::state)
- static int **lua_closest_friendly** (lpp::Script::state)
- static int **lua_closest_enemy_in_sight_thats_not** (lpp::Script::state)
- static int **lua_closest_friendly_in_sight_thats_not** (lpp::Script::state)
- static int **lua_closest_enemy_thats_not** (lpp::Script::state)
- static int **lua_closest_friendly_thats_not** (lpp::Script::state)
- static int **lua_in_sight** (lpp::Script::state)
- static int **lua_run_away_from** (lpp::Script::state)
- static int **lua_set_max_run_away_attempts** (lpp::Script::state)
- static int **lua_get_max_run_away_attempts** (lpp::Script::state)
- static int **lua_apply_heal_to_entities_in_range** (lpp::Script::state)
- static int **lua_apply_damage_to_entities_in_range** (lpp::Script::state)
- static int **lua_apply_slow_to_entities_in_range** (lpp::Script::state)
- static int **lua_apply_freeze_to_entities_in_range** (lpp::Script::state)
- static int **lua_in_range** (lpp::Script::state)
- static int **lua_set_projectile_blueprint** (lpp::Script::state)
- static int **lua_get_projectile_blueprint** (lpp::Script::state)
- static int **lua_apply_slow_to** (lpp::Script::state)
- static int **lua_apply_freeze_to** (lpp::Script::state)
- static int **lua_enemy_in_range** (lpp::Script::state)
- static int **lua_closest_friendly_structure** (lpp::Script::state)
- static int **lua_closest_enemy_structure** (lpp::Script::state)
- static int **lua_closest_friendly_structure_in_sight** (lpp::Script::state)
- static int **lua_closest_enemy_structure_in_sight** (lpp::Script::state)

- static int **lua_set_production_blueprint** (lpp::Script::state)
- static int **lua_get_production_blueprint** (lpp::Script::state)
- static int **lua_set_production_limit** (lpp::Script::state)
- static int **lua_get_production_limit** (lpp::Script::state)
- static int **lua_set_production_cooldown** (lpp::Script::state)
- static int **lua_get_production_cooldown** (lpp::Script::state)
- static int **lua_set_production_progress** (lpp::Script::state)
- static int **lua_get_production_progress** (lpp::Script::state)
- static int **lua_set_production_count** (lpp::Script::state)
- static int **lua_get_production_count** (lpp::Script::state)
- static int **lua_set_producer** (lpp::Script::state)
- static int **lua_get_producer** (lpp::Script::state)
- static int **lua_instant_production** (lpp::Script::state)
- static int **lua_set_production_multiplier** (lpp::Script::state)
- static int **lua_get_production_multiplier** (lpp::Script::state)
- static int **lua_double_production** (lpp::Script::state)
- static int **lua_increase_production** (lpp::Script::state)
- static int **lua_get_curr_time** (lpp::Script::state)
- static int **lua_advance_curr_time** (lpp::Script::state)
- static int **lua_max_curr_time** (lpp::Script::state)
- static int **lua_set_time_limit** (lpp::Script::state)
- static int **lua_get_time_limit** (lpp::Script::state)
- static int **lua_set_timer_target** (lpp::Script::state)
- static int **lua_get_timer_target** (lpp::Script::state)
- static int **lua_set_timer_type** (lpp::Script::state)
- static int **lua_get_timer_type** (lpp::Script::state)
- static int **lua_advance_all_timers** (lpp::Script::state)
- static int **lua_advance_all_timers_of_type** (lpp::Script::state)
- static int **lua_set_timer_multiplier** (lpp::Script::state)
- static int **lua_get_timer_multiplier** (lpp::Script::state)
- static int **lua_set_event_type** (lpp::Script::state)
- static int **lua_get_event_type** (lpp::Script::state)
- static int **lua_set_event_target** (lpp::Script::state)
- static int **lua_get_event_target** (lpp::Script::state)
- static int **lua_set_event_radius** (lpp::Script::state)
- static int **lua_get_event_radius** (lpp::Script::state)
- static int **lua_set_event_active** (lpp::Script::state)
- static int **lua_is_event_active** (lpp::Script::state)
- static int **lua_set_handler_of_event** (lpp::Script::state)
- static int **lua_get_handler_of_event** (lpp::Script::state)
- static int **lua_set_event_handler** (lpp::Script::state)
- static int **lua_get_event_handler** (lpp::Script::state)
- static int **lua_can_handle_event** (lpp::Script::state)
- static int **lua_add_possible_event** (lpp::Script::state)
- static int **lua_delete_possible_event** (lpp::Script::state)
- static int **lua_set_event_update_period** (lpp::Script::state)
- static int **lua_get_event_update_period** (lpp::Script::state)
- static int **lua_set_event_update_multiplier** (lpp::Script::state)
- static int **lua_get_event_update_multiplier** (lpp::Script::state)
- static int **lua_set_destructor_blueprint** (lpp::Script::state)
- static int **lua_get_destructor_blueprint** (lpp::Script::state)
- static int **lua_set_curr_gold** (lpp::Script::state)
- static int **lua_get_curr_gold** (lpp::Script::state)
- static int **lua_set_max_gold** (lpp::Script::state)
- static int **lua_get_max_gold** (lpp::Script::state)

- static int **lua_add_gold** (lpp::Script::state)
- static int **lua_sub_gold** (lpp::Script::state)
- static int **lua_transfer_all_gold** (lpp::Script::state)
- static int **lua_get_closest_gold_deposit** (lpp::Script::state)
- static int **lua_get_closest_gold_deposit_in_sight** (lpp::Script::state)
- static int **lua_gold_full** (lpp::Script::state)
- static int **lua_is_gold_vault** (lpp::Script::state)
- static int **lua_get_closest_gold_vault** (lpp::Script::state)
- static int **lua_get_closest_gold_vault_in_sight** (lpp::Script::state)
- static int **lua_get_closest_free_gold_vault** (lpp::Script::state)
- static int **lua_get_closest_free_gold_vault_in_sight** (lpp::Script::state)
- static int **lua_exists_free_gold_vault** (lpp::Script::state)
- static int **lua_set_gui_visible** (lpp::Script::state)
- static int **lua_is_gui_visible** (lpp::Script::state)
- static int **lua_set_window_visible** (lpp::Script::state)
- static int **lua_is_window_visible** (lpp::Script::state)
- static int **lua_show_save_dialog** (lpp::Script::state)
- static int **lua_show_load_dialog** (lpp::Script::state)
- static int **lua_clear_log** (lpp::Script::state)
- static int **lua_print_to_log** (lpp::Script::state)
- static int **lua_set_log_history** (lpp::Script::state)
- static int **lua_get_log_history** (lpp::Script::state)
- static int **lua_set_log_visible** (lpp::Script::state)
- static int **lua_is_log_visible** (lpp::Script::state)
- static int **lua_set_tracked_entity** (lpp::Script::state)
- static int **lua_get_tracked_entity** (lpp::Script::state)
- static int **lua_update_tracking** (lpp::Script::state)
- static int **lua_clear_entity_tracker** (lpp::Script::state)
- static int **lua_set_tracker_visible** (lpp::Script::state)
- static int **lua_is_tracker_visible** (lpp::Script::state)
- static int **lua_console_scroll_down** (lpp::Script::state)
- static int **lua_set_console_history** (lpp::Script::state)
- static int **lua_get_console_history** (lpp::Script::state)
- static int **lua_set_console_visible** (lpp::Script::state)
- static int **lua_is_console_visible** (lpp::Script::state)
- static int **lua_clear_console** (lpp::Script::state)
- static int **lua_set_builder_visible** (lpp::Script::state)
- static int **lua_is_builder_visible** (lpp::Script::state)
- static int **lua_register_building** (lpp::Script::state)
- static int **lua_research_show** (lpp::Script::state)
- static int **lua_free_research** (lpp::Script::state)
- static int **lua_research_all** (lpp::Script::state)
- static int **lua_dummy_unlock** (lpp::Script::state)
- static int **lua_research_reset** (lpp::Script::state)
- static int **lua_add_player_gold** (lpp::Script::state)
- static int **lua_sub_player_gold** (lpp::Script::state)
- static int **lua_add_player_mana** (lpp::Script::state)
- static int **lua_sub_player_mana** (lpp::Script::state)
- static int **lua_add_player_max_units** (lpp::Script::state)
- static int **lua_sub_player_max_units** (lpp::Script::state)
- static int **lua_add_player_curr_units** (lpp::Script::state)
- static int **lua_sub_player_curr_units** (lpp::Script::state)
- static int **lua_get_player_gold** (lpp::Script::state)
- static int **lua_get_player_mana** (lpp::Script::state)
- static int **lua_player_reset** (lpp::Script::state)

- static int **lua_nulify_player_stats** (lpp::Script::state)
- static int **lua_add_player_max_mana** (lpp::Script::state)
- static int **lua_sub_player_max_mana** (lpp::Script::state)
- static int **lua_get_player_max_mana** (lpp::Script::state)
- static int **lua_add_player_mana_regen** (lpp::Script::state)
- static int **lua_sub_player_mana_regen** (lpp::Script::state)
- static int **lua_get_player_mana_regen** (lpp::Script::state)
- static int **lua_set_price** (lpp::Script::state)
- static int **lua_get_price** (lpp::Script::state)
- static int **lua_register_spell** (lpp::Script::state)
- static int **lua_spellcaster_set_type** (lpp::Script::state)
- static int **lua_spellcaster_get_type** (lpp::Script::state)
- static int **lua_spellcaster_set_spell** (lpp::Script::state)
- static int **lua_spellcaster_get_spell** (lpp::Script::state)
- static int **lua_spellcaster_get_last_type** (lpp::Script::state)
- static int **lua_spellcaster_get_last_spell** (lpp::Script::state)
- static int **lua_spellcaster_set_last_spell_id** (lpp::Script::state)
- static int **lua_spellcaster_get_last_spell_id** (lpp::Script::state)
- static int **lua_spellcaster_is_casting** (lpp::Script::state)
- static int **lua_spellcaster_stop_casting** (lpp::Script::state)
- static int **lua_align_set_material** (lpp::Script::state)
- static int **lua_align_get_material** (lpp::Script::state)
- static int **lua_align_set_mesh** (lpp::Script::state)
- static int **lua_align_get_mesh** (lpp::Script::state)
- static int **lua_align_set_position_offset** (lpp::Script::state)
- static int **lua_align_get_position_offset** (lpp::Script::state)
- static int **lua_align_set_scale** (lpp::Script::state)
- static int **lua_align_get_scale** (lpp::Script::state)
- static int **lua_mana_crystal_set_cap** (lpp::Script::state)
- static int **lua_mana_crystal_get_cap** (lpp::Script::state)
- static int **lua_mana_crystal_set_regen** (lpp::Script::state)
- static int **lua_mana_crystal_get_regen** (lpp::Script::state)
- static int **lua_on_hit_set_blueprint** (lpp::Script::state)
- static int **lua_on_hit_get_blueprint** (lpp::Script::state)
- static int **lua_on_hit_call** (lpp::Script::state)
- static int **lua_on_hit_set_cooldown** (lpp::Script::state)
- static int **lua_on_hit_get_cooldown** (lpp::Script::state)
- static int **lua_constructor_set_blueprint** (lpp::Script::state)
- static int **lua_constructor_get_blueprint** (lpp::Script::state)
- static int **lua_constructor_call** (lpp::Script::state)
- static int **lua_trigger_set_blueprint** (lpp::Script::state)
- static int **lua_trigger_get_blueprint** (lpp::Script::state)
- static int **lua_trigger_set_linked_entity** (lpp::Script::state)
- static int **lua_trigger_get_linked_entity** (lpp::Script::state)
- static int **lua_trigger_set_cooldown** (lpp::Script::state)
- static int **lua_trigger_get_cooldown** (lpp::Script::state)
- static int **lua_trigger_trigger** (lpp::Script::state)
- static int **lua_trigger_set_check_period** (lpp::Script::state)
- static int **lua_trigger_get_check_period** (lpp::Script::state)
- static int **lua_trigger_can_be_triggered_by** (lpp::Script::state)
- static int **lua_trigger_reset_timer** (lpp::Script::state)
- static int **lua_trigger_set_radius** (lpp::Script::state)
- static int **lua_trigger_get_radius** (lpp::Script::state)
- static int **lua_upgrade_set_blueprint** (lpp::Script::state)
- static int **lua_upgrade_get_blueprint** (lpp::Script::state)

- static int **lua_upgrade_set_experience** (lpp::Script::state)
- static int **lua_upgrade_get_experience** (lpp::Script::state)
- static int **lua_upgrade_add_experience** (lpp::Script::state)
- static int **lua_upgrade_set_exp_needed** (lpp::Script::state)
- static int **lua_upgrade_get_exp_needed** (lpp::Script::state)
- static int **lua_upgrade_set_level** (lpp::Script::state)
- static int **lua_upgrade_get_level** (lpp::Script::state)
- static int **lua_upgrade_set_level_cap** (lpp::Script::state)
- static int **lua_upgrade_get_level_cap** (lpp::Script::state)
- static int **lua_upgrade_can_level_up** (lpp::Script::state)
- static int **lua_upgrade_upgrade** (lpp::Script::state)
- static int **lua_upgrade_all_level_up** (lpp::Script::state)
- static int **lua_notification_set_cooldown** (lpp::Script::state)
- static int **lua_notification_get_cooldown** (lpp::Script::state)
- static int **lua_notification_reset** (lpp::Script::state)
- static int **lua_notification_notify** (lpp::Script::state)
- static int **lua_notification_get_curr_time** (lpp::Script::state)
- static int **lua_notification_advance_curr_time** (lpp::Script::state)
- static int **lua_explosion_set_delta** (lpp::Script::state)
- static int **lua_explosion_get_delta** (lpp::Script::state)
- static int **lua_explosion_set_max_radius** (lpp::Script::state)
- static int **lua_explosion_get_max_radius** (lpp::Script::state)
- static int **lua_explosion_get_curr_radius** (lpp::Script::state)
- static int **lua_explosion_increase_curr_radius** (lpp::Script::state)
- static int **lua_lls_set_max_time** (lpp::Script::state)
- static int **lua_lls_get_max_time** (lpp::Script::state)
- static int **lua_lls_get_curr_time** (lpp::Script::state)
- static int **lua_lls_advance_curr_time** (lpp::Script::state)
- static int **lua_name_set** (lpp::Script::state)
- static int **lua_name_get** (lpp::Script::state)
- static int **lua_exp_val_set** (lpp::Script::state)
- static int **lua_exp_val_get** (lpp::Script::state)
- static int **lua_exp_val_inc** (lpp::Script::state)
- static int **lua_exp_val_dec** (lpp::Script::state)
- static int **lua_mana_set_regen_period** (lpp::Script::state)
- static int **lua_mana_get_regen_period** (lpp::Script::state)
- static int **lua_wave_next_wave** (lpp::Script::state)
- static int **lua_wave_advance_countdown** (lpp::Script::state)
- static int **lua_wave_entity_died** (lpp::Script::state)
- static int **lua_wave_start** (lpp::Script::state)
- static int **lua_wave_pause** (lpp::Script::state)
- static int **lua_wave_set_entity_total** (lpp::Script::state)
- static int **lua_wave_get_entity_total** (lpp::Script::state)
- static int **lua_wave_set_wave_count** (lpp::Script::state)
- static int **lua_wave_get_wave_count** (lpp::Script::state)
- static int **lua_wave_add_spawn_node** (lpp::Script::state)
- static int **lua_wave_clear_spawn_nodes** (lpp::Script::state)
- static int **lua_wave_set_spawn_cooldown** (lpp::Script::state)
- static int **lua_wave_get_spawn_cooldown** (lpp::Script::state)
- static int **lua_wave_add_entity_blueprint** (lpp::Script::state)
- static int **lua_wave_set_table** (lpp::Script::state)
- static int **lua_wave_get_table** (lpp::Script::state)
- static int **lua_wave_set_curr_wave_number** (lpp::Script::state)
- static int **lua_wave_get_curr_wave_number** (lpp::Script::state)
- static int **lua_wave_set_countdown** (lpp::Script::state)

- static int **lua_wave_get_countdown** (lpp::Script::state)
- static int **lua_wave_set_state** (lpp::Script::state)
- static int **lua_wave_get_state** (lpp::Script::state)
- static int **lua_wave_update_label_text** (lpp::Script::state)
- static int **lua_wave_set_spawn_timer** (lpp::Script::state)
- static int **lua_wave_get_spawn_timer** (lpp::Script::state)
- static int **lua_wave_set_wave_entities** (lpp::Script::state)
- static int **lua_wave_get_wave_entities** (lpp::Script::state)
- static int **lua_wave_set_entities_spawned** (lpp::Script::state)
- static int **lua_wave_get_entities_spawned** (lpp::Script::state)
- static int **lua_wave_clear_entity_blueprints** (lpp::Script::state)
- static int **lua_wave_list** (lpp::Script::state)
- static int **lua_wave_set_endless_mode** (lpp::Script::state)
- static int **lua_wave_get_endless_mode** (lpp::Script::state)
- static int **lua_wave_turn_endless_on** (lpp::Script::state)
- static int **lua_msg_to_plr_show** (lpp::Script::state)
- static int **lua_msg_to_plr_show_ok** (lpp::Script::state)
- static int **lua_msg_to_plr_show_yes_no** (lpp::Script::state)
- static int **lua_msg_set_butt_label** (lpp::Script::state)
- static int **lua_msg_reset_butt_labels** (lpp::Script::state)
- static int **lua_mana_add** (lpp::Script::state)
- static int **lua_mana_sub** (lpp::Script::state)
- static int **lua_mana_set** (lpp::Script::state)
- static int **lua_mana_get** (lpp::Script::state)
- static int **lua_mana_set_max** (lpp::Script::state)
- static int **lua_mana_get_max** (lpp::Script::state)
- static int **lua_mana_set_regen** (lpp::Script::state)
- static int **lua_mana_get_regen** (lpp::Script::state)
- static int **lua_ent_spell_set_blueprint** (lpp::Script::state)
- static int **lua_ent_spell_get_blueprint** (lpp::Script::state)
- static int **lua_ent_spell_set_cooldown** (lpp::Script::state)
- static int **lua_ent_spell_get_cooldown** (lpp::Script::state)
- static int **lua_ent_spell_advance_curr_time** (lpp::Script::state)
- static int **lua_ent_spell_set_curr_time** (lpp::Script::state)
- static int **lua_ent_spell_get_curr_time** (lpp::Script::state)
- static int **lua_ent_spell_cast** (lpp::Script::state)
- static int **lua_light_set_visible** (lpp::Script::state)
- static int **lua_light_toggle_visible** (lpp::Script::state)
- static int **lua_light_is_visible** (lpp::Script::state)
- static int **lua_light_init** (lpp::Script::state)
- static int **lua_command_set** (lpp::Script::state)
- static int **lua_command_test** (lpp::Script::state)
- static int **lua_counter_increment** (lpp::Script::state)
- static int **lua_counter_decrement** (lpp::Script::state)
- static int **lua_counter_set_curr_value** (lpp::Script::state)
- static int **lua_counter_get_curr_value** (lpp::Script::state)
- static int **lua_counter_set_max_value** (lpp::Script::state)
- static int **lua_counter_get_max_value** (lpp::Script::state)

Static Private Attributes

- static Game ∗ lua_this {}

  Lua requires all functions registered in it to be static (because it doesn't know anything about C++).
- static EntitySystem ∗ **ents** {}

6.59.1 Detailed Description

Class that creates an interface between engine (C++) and logic (Lua) code.

Definition at line 12 of file LuaInterface.hpp.

6.59.2 Constructor & Destructor Documentation

6.59.2.1 LuaInterface::LuaInterface ( ) [delete]

This is a static class, so all constructors are deleted.

6.59.3 Member Function Documentation

6.59.3.1 void LuaInterface::init (Game ∗ *game*) [static]

Sets the lua_this pointer and registers all C++ API functions to Lua.

Parameters

Pointer	to the game object that provides the game data to Lua.

Definition at line 44 of file LuaInterface.cpp.

6.59.3.2 int LuaInterface::lua_get_avg_fps (lpp::Script::state *L*) [static], [private]

Note

Function definitions below act as an interface between C++ and Lua, they all have to have the signature int fname(lpp::Script::state) and return the number of results pushed onto the Lua stack (Lua allows to return multiple results if needed). Important: These functions will have their arguments on the stack in REVERSED ORDER! (Because, you know, it's a stack...)

Definition at line 892 of file LuaInterface.cpp.

6.59.4 Member Data Documentation

6.59.4.1 Game ∗ LuaInterface::lua_this {} [static], [private]

Lua requires all functions registered in it to be static (because it doesn't know anything about C++).

Static member initialization, will be set in the init method.

Definition at line 34 of file LuaInterface.hpp.

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

- LuaInterface.hpp
- LuaInterface.cpp

6.60 ManaComponent Struct Reference

Allows an entity to cast spell by providing the mana resource.

```
#include <Components.hpp>
```

Public Member Functions

- **ManaComponent** (tdt::uint max=0, tdt::uint regen=0)
- **ManaComponent** (const ManaComponent &)=default
- **ManaComponent** (ManaComponent &&)=default
- ManaComponent & **operator=** (const ManaComponent &)=default
- ManaComponent & **operator=** (ManaComponent &&)=default

Public Attributes

- tdt::uint **curr_mana**
- tdt::uint **max_mana**
- tdt::uint **mana_regen**

Static Public Attributes

- static constexpr int **type** = 9

6.60.1 Detailed Description

Allows an entity to cast spell by providing the mana resource.

Definition at line 251 of file Components.hpp.

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

- Components.hpp

6.61 ManaCrystalComponent Struct Reference

Allows an entity to increase the player's mana capacity and regeneration rate while it's alive.

```
#include <Components.hpp>
```

Public Member Functions

- **ManaCrystalComponent** (tdt::uint cap=0, tdt::uint regen=0)
- **ManaCrystalComponent** (const ManaCrystalComponent &)=default
- **ManaCrystalComponent** (ManaCrystalComponent &&)=default
- ManaCrystalComponent & **operator=** (const ManaCrystalComponent &)=default
- ManaCrystalComponent & **operator=** (ManaCrystalComponent &&)=default

Public Attributes

- tdt::uint **cap_increase**
- tdt::uint **regen_increase**

Static Public Attributes

- static constexpr int **type** = 26

6.61.1 Detailed Description

Allows an entity to increase the player's mana capacity and regeneration rate while it's alive.

Definition at line 626 of file Components.hpp.

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

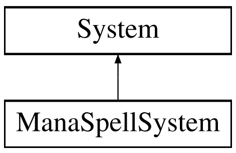
- Components.hpp

6.62 ManaSpellSystem Class Reference

Regenerates mana to the player and all entities that have mana.

`#include <ManaSpellSystem.hpp>`

Inheritance diagram for ManaSpellSystem:



Public Member Functions

- ManaSpellSystem (EntitySystem &)

  Constructor.
- ∼ManaSpellSystem ()=default

  Destructor.
- void update (tdt::real) override

  Regenerates mana if necessary and performs entity spell casting if off cooldown.
- void set_regen_period (tdt::real)

  Sets the time period between mana regens.
- tdt::real get_regen_period () const

  Returns the time period between mana regens.

Private Attributes

- EntitySystem & entities_

  Entity system containing entities that this system works with.
- tdt::real regen_timer_

  Allow for dynamic periods between mana regeneration updates.
- tdt::real **regen_period_**

6.62.1 Detailed Description

Regenerates mana to the player and all entities that have mana.

Also takes care of entity spell casting.

Definition at line 11 of file ManaSpellSystem.hpp.

6.62.2 Constructor & Destructor Documentation

6.62.2.1 ManaSpellSystem::ManaSpellSystem (EntitySystem & *ents*)

Constructor.

Parameters

Entity	system containing entities this system works with.

Definition at line 6 of file ManaSpellSystem.cpp.

6.62.2.2 ManaSpellSystem::∼ManaSpellSystem ( ) `[default]`

Destructor.

6.62.3 Member Function Documentation

6.62.3.1 tdt::real ManaSpellSystem::get_regen_period ( ) const

Returns the time period between mana regens.

Definition at line 45 of file ManaSpellSystem.cpp.

6.62.3.2 void ManaSpellSystem::set_regen_period (tdt::real *val*)

Sets the time period between mana regens.

Parameters

The	new time period.

Definition at line 40 of file ManaSpellSystem.cpp.

6.62.3.3 void ManaSpellSystem::update (tdt::real *delta*) [override], [virtual]

Regenerates mana if necessary and performs entity spell casting if off cooldown.

Parameters

Time	since last frame.

Implements System.

Definition at line 10 of file ManaSpellSystem.cpp.

6.62.4 Member Data Documentation

6.62.4.1 EntitySystem& ManaSpellSystem::entities_ [private]

Entity system containing entities that this system works with.

Definition at line 48 of file ManaSpellSystem.hpp.

6.62.4.2 tdt::real ManaSpellSystem::regen_timer_ [private]

Allow for dynamic periods between mana regeneration updates.

Definition at line 54 of file ManaSpellSystem.hpp.

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

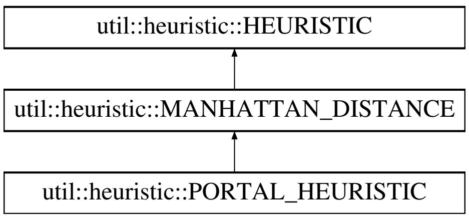
- systems/ManaSpellSystem.hpp
- systems/ManaSpellSystem.cpp

6.63 util::heuristic::MANHATTAN_DISTANCE Struct Reference

Returns the manhattan distance between two nodes.

`#include <PathfindingAlgorithms.hpp>`

Inheritance diagram for util::heuristic::MANHATTAN_DISTANCE:


util::heuristic::HEURISTIC
util::heuristic::MANHATTAN_DISTANCE
util::heuristic::PORTAL_HEURISTIC

Public Member Functions

- **MANHATTAN_DISTANCE** (EntitySystem &ents)
- tdt::real **get_cost** (tdt::uint id1, tdt::uint id2) override

Additional Inherited Members

6.63.1 Detailed Description

Returns the manhattan distance between two nodes.

(Well, actually, it's an octal distance :)

Definition at line 206 of file PathfindingAlgorithms.hpp.

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

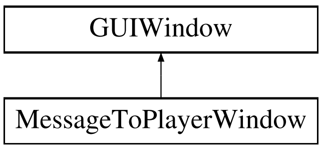
- tools/PathfindingAlgorithms.hpp

6.64 MessageToPlayerWindow Class Reference

A window that can show the player a text message with 1, 2 or 3 buttons (custom labels) that can call assigned functions.

```
#include <MessageToPlayerWindow.hpp>
```

Inheritance diagram for MessageToPlayerWindow:



Public Member Functions

- MessageToPlayerWindow ()

  Constructor.
- ~MessageToPlayerWindow ()=default

  Destructor.
- void show (const std::string &="NONE", const std::string &="NONE", const std::string &="NONE", const std↵ ::string &="NONE")

  Shows a given text message to the player and assigns callbacks to the buttons.
- void set_butt_label (const std::string &, const std::string &)

  Sets the label of a given button.
- void reset_butt_labels ()

  Resets the button labels to their default values.

Protected Member Functions

- void init_ ()

  Initializes this window.

Private Attributes

- lpp::Script & script_

  Reference to the scripting engine for easier use when calling Lua callbacks.
- std::string ok_func_

  Labels of the buttons.
- std::string **yes_func_**
- std::string **no_func_**

Additional Inherited Members

6.64.1 Detailed Description

A window that can show the player a text message with 1, 2 or 3 buttons (custom labels) that can call assigned functions.

Button names: (used for setting labels) 1st on the left: NO 2nd on the left: YES 1st on the right: OK

Definition at line 18 of file MessageToPlayerWindow.hpp.

6.64.2 Constructor & Destructor Documentation

6.64.2.1 MessageToPlayerWindow::MessageToPlayerWindow ( )

Constructor.

Definition at line 5 of file MessageToPlayerWindow.cpp.

6.64.2.2 MessageToPlayerWindow::∼MessageToPlayerWindow ( ) [default]

Destructor.

6.64.3 Member Function Documentation

6.64.3.1 void MessageToPlayerWindow::init_ ( ) [protected], [virtual]

Initializes this window.

Implements GUIWindow.

Definition at line 53 of file MessageToPlayerWindow.cpp.

6.64.3.2 void MessageToPlayerWindow::reset_butt_labels ( )

Resets the button labels to their default values.

("OK", "YES" and "NO")

Definition at line 46 of file MessageToPlayerWindow.cpp.

6.64.3.3 void MessageToPlayerWindow::set_butt_label (const std::string & *butt,* const std::string & *val*)

Sets the label of a given button.

Parameters

The	name of the button (OK, YES, NO).
The	new label.

Definition at line 41 of file MessageToPlayerWindow.cpp.

6.64.3.4 void MessageToPlayerWindow::show (const std::string & *msg* = `"NONE"`, const std::string & *ok* = `"NONE"`, const std::string & *yes* = `"NONE"`, const std::string & *no* = `"NONE"`)

Shows a given text message to the player and assigns callbacks to the buttons.

Parameters

The	message.
Callback	for the OK button.
Callback	for the YES button.
Callback	for the NO button.

Note

If a callback passes is "NONE", the assigned button will not be shown.
The callbacks are strings of names of the Lua functions that ought to be called when the button is pressed.

Definition at line 10 of file MessageToPlayerWindow.cpp.

6.64.4 Member Data Documentation

6.64.4.1 std::string MessageToPlayerWindow::ok_func_ `[private]`

Labels of the buttons.

Definition at line 74 of file MessageToPlayerWindow.hpp.

6.64.4.2 lpp::Script& MessageToPlayerWindow::script_ `[private]`

Reference to the scripting engine for easier use when calling Lua callbacks.

Definition at line 69 of file MessageToPlayerWindow.hpp.

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

- gui/MessageToPlayerWindow.hpp
- gui/MessageToPlayerWindow.cpp

6.65 MineComponent Struct Reference

Dummy component that signals that an entity having it can be mined.

```
#include <Components.hpp>
```

Static Public Attributes

- static constexpr int **type** = 25

6.65.1 Detailed Description

Dummy component that signals that an entity having it can be mined.

Definition at line 617 of file Components.hpp.

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

- Components.hpp

6.66 MovementComponent Struct Reference

Holds info related to movement, if an entity has this component it should also have a Physics component (containing the entity's position), otherwise the MovementSystem might not work correctly.

```
#include <Components.hpp>
```

Public Member Functions

- **MovementComponent** (tdt::real speed=0.f)
- **MovementComponent** (const MovementComponent &)=default
- **MovementComponent** (MovementComponent &&)=default
- MovementComponent & **operator=** (const MovementComponent &)=default
- MovementComponent & **operator=** (MovementComponent &&)=default

Public Attributes

- tdt::real **speed_modifier**
- tdt::real **original_speed**

Static Public Attributes

- static constexpr int **type** = 4

6.66.1 Detailed Description

Holds info related to movement, if an entity has this component it should also have a Physics component (containing the entity's position), otherwise the MovementSystem might not work correctly.

Definition at line 132 of file Components.hpp.

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

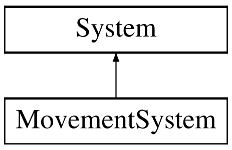
- Components.hpp

6.67 MovementSystem Class Reference

System handling movement related updates and containing movement & physics related methods.

```
#include <MovementSystem.hpp>
```

Inheritance diagram for MovementSystem:



Public Member Functions

- MovementSystem (EntitySystem &)

    Constructor.
- ∼MovementSystem ()

    Destructor.
- void update (Ogre::Real)

    Updates the movement system.
- bool can_move_to (std::size_t, Ogre::Vector3)

    Returns true if a given entity can move to a given point in space, false otherwise.
- bool checked_move (std::size_t, Ogre::Vector3)

    Sets a given entity to move in a given direction, returns true if such movement is possible and false otherwise.
- bool move (std::size_t, Ogre::Vector3)

    Sets a given entity to move in a given direction, returns true if such movement is possible and false otherwise.

Private Attributes

- EntitySystem & entities_

    Reference to the game's entity system.
- Ogre::Real last_delta_

    Time between the last frame and this frame, used so that lua calls to the move functions still use the time of this frame.

6.67.1 Detailed Description

System handling movement related updates and containing movement & physics related methods.

Definition at line 10 of file MovementSystem.hpp.

6.67.2 Constructor & Destructor Documentation

6.67.2.1 MovementSystem::MovementSystem (EntitySystem & *ents*)

Constructor.

Parameters

Reference	to the game's entity system.

Definition at line 7 of file MovementSystem.cpp.

6.67.2.2 MovementSystem::∼MovementSystem ( ) [inline]

Destructor.

Definition at line 22 of file MovementSystem.hpp.

6.67.3 Member Function Documentation

6.67.3.1 bool MovementSystem::can_move_to (std::size_t *id,* Ogre::Vector3 *pos*)

Returns true if a given entity can move to a given point in space, false otherwise.

Parameters

ID	of the entity.
Target	coordinate.

Definition at line 49 of file MovementSystem.cpp.

6.67.3.2 bool MovementSystem::checked_move (std::size_t *id,* Ogre::Vector3 *dir_vector*)

Sets a given entity to move in a given direction, returns true if such movement is possible and false otherwise.

The move will then be applied in the update method.

Parameters

ID	of the entity.
Directional	vector.

Note

Every vector passed to this method should be normalised and the length of the move will be increased by the entity's speed modifier. This is not enforced though.
Checks for collisions.

Definition at line 85 of file MovementSystem.cpp.

6.67.3.3 bool MovementSystem::move (std::size_t *id,* Ogre::Vector3 *dir_vector*)

Sets a given entity to move in a given direction, returns true if such movement is possible and false otherwise.

The move will then be applied in the update method.

Parameters

ID	of the entity.
Directional	vector.

Note

Every vector passed to this method should be normalised and the length of the move will be increased by the entity's speed modifier. This is not enforced though.
Does not check for collisions.

Definition at line 110 of file MovementSystem.cpp.

6.67.3.4 void MovementSystem::update (Ogre::Real *delta*) `[virtual]`

Updates the movement system.

Parameters

Time	since the last frame.

Implements System.

Definition at line 11 of file MovementSystem.cpp.

6.67.4 Member Data Documentation

6.67.4.1 EntitySystem& MovementSystem::entities_ `[private]`

Reference to the game's entity system.

Definition at line 63 of file MovementSystem.hpp.

6.67.4.2 Ogre::Real MovementSystem::last_delta_ `[private]`

Time between the last frame and this frame, used so that lua calls to the move functions still use the time of this frame.

Definition at line 69 of file MovementSystem.hpp.

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

- systems/MovementSystem.hpp
- systems/MovementSystem.cpp

6.68 NameComponent Struct Reference

Name of the entity shown in the entity viewer.

`#include <Components.hpp>`

Public Member Functions

- **NameComponent** (std::string &&n="ERROR")
- **NameComponent** (const NameComponent &)=default
- **NameComponent** (NameComponent &&)=default
- NameComponent & **operator=** (const NameComponent &)=default
- NameComponent & **operator=** (NameComponent &&)=default

Public Attributes

- std::string **name**

Static Public Attributes

- static constexpr int **type** = 34

6.68.1 Detailed Description

Name of the entity shown in the entity viewer.

Definition at line 803 of file Components.hpp.

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

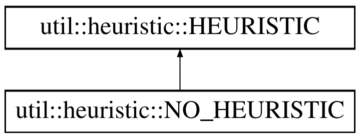
- Components.hpp

6.69 util::heuristic::NO_HEURISTIC Struct Reference

Represents no heuristic by returning 0 all the time.

```
#include <PathfindingAlgorithms.hpp>
```

Inheritance diagram for util::heuristic::NO_HEURISTIC:



Public Member Functions

- **NO_HEURISTIC** (EntitySystem &ents)
- tdt::real **get_cost** (tdt::uint id1, tdt::uint id2) override

Additional Inherited Members

6.69.1 Detailed Description

Represents no heuristic by returning 0 all the time.

Definition at line 223 of file PathfindingAlgorithms.hpp.

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

- tools/PathfindingAlgorithms.hpp

6.70 NotificationComponent Struct Reference

Allows to keep track about notification cooldown, so that an entity doesn't spam the player with messages on reoccuring events in a short time period.

```
#include <Components.hpp>
```

Public Member Functions

- **NotificationComponent** (tdt::real cd=0.f)
- **NotificationComponent** (const NotificationComponent &)=default
- **NotificationComponent** (NotificationComponent &&)=default
- NotificationComponent & **operator=** (const NotificationComponent &)=default
- NotificationComponent & **operator=** (NotificationComponent &&)=default

Public Attributes

- tdt::real **curr_time**
- tdt::real **cooldown**

Static Public Attributes

- static constexpr int **type** = 31

6.70.1 Detailed Description

Allows to keep track about notification cooldown, so that an entity doesn't spam the player with messages on reoccuring events in a short time period.

Definition at line 739 of file Components.hpp.

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

- Components.hpp

6.71 OnHitComponent Struct Reference

Contains the blueprint table which gets called when an entity that has this component gets hit.

```
#include <Components.hpp>
```

Public Member Functions

- **OnHitComponent** (std::string &&b="ERROR", tdt::real cd=0.f)
- **OnHitComponent** (const OnHitComponent &)=default
- **OnHitComponent** (OnHitComponent &&)=default
- OnHitComponent & **operator=** (const OnHitComponent &)=default
- OnHitComponent & **operator=** (OnHitComponent &&)=default

Public Attributes

- std::string **blueprint**
- tdt::real **curr_time**
- tdt::real **cooldown**

Static Public Attributes

- static constexpr int **type** = 27

6.71.1 Detailed Description

Contains the blueprint table which gets called when an entity that has this component gets hit.

Definition at line 647 of file Components.hpp.

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

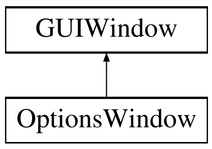
- Components.hpp

6.72 OptionsWindow Class Reference

Options menu window that lets the player to change the resolution, window mode and keybinging.

`#include <OptionsWindow.hpp>`

Inheritance diagram for OptionsWindow:



Public Member Functions

- OptionsWindow ()

  Constructor.
- ∼OptionsWindow ()=default

  Destructor.
- void add_start_parameters (Ogre::RenderWindow ∗, Ogre::Viewport ∗, CEGUI::OgreRenderer ∗)

  Saves pointers to the window, view and renderer and sets starting values for the resolution, window mode etc.
- bool key_pressed (CEGUI::Key::Scan)

  Registers a key press for keybinding.
- void set_key_bind (KEY_BIND_ACTION::VAL, CEGUI::Key::Scan)

  Binds a given action to a given key.

Protected Member Functions

- void init_ () override

  Initializes the CEGUI window.

Private Types

- typedef void(∗ **ActionFuncPtr**) ()

Private Member Functions

- void apply_ ()

  Applies the graphical changes and saves key bindings into a script that allows persistent key binds.
- void update_labels_ ()

  Updates the button texts and labels to reflect the new settings.
- void **update_fonts_** ()
- void **update_font_of_window_** (CEGUI::Window ∗, const std::string &)
- const std::string & get_key_bind_name_ (KEY_BIND_ACTION::VAL)

  Returns the name of the key a given action is bound to.

Private Attributes

- Ogre::RenderWindow ∗ render_window_

  Window the game is rendered to.
- Ogre::Viewport ∗ view_

  Main viewport of the render window.
- CEGUI::OgreRenderer ∗ renderer_

  Renderer used by CEGUI (used to sync resolution between game and UI).
- tdt::uint width_

  Dimensions of the display resolution.
- tdt::uint **height_**
- bool fullscreen_

  True for fullscreen, false for windowed mode.
- std::map< CEGUI::Key::Scan, std::string > key_names_

  This map connects key codes with their names (used for key bind buttons).
- std::array< ActionFuncPtr, KEY_BIND_ACTION::COUNT > actions_

  Action functions assigned to actions.
- std::array< CEGUI::Key::Scan, KEY_BIND_ACTION::COUNT > key_binds_

  Keys bound to actions.
- std::map< std::string, std::tuple< tdt::uint, tdt::uint > > resolutions_

  Resolution strings and their assigned dimensions.
- KEY_BIND_ACTION::VAL currently_binded_action_

  Saves the currently binded action so that the next key press can be used.

Additional Inherited Members

6.72.1 Detailed Description

Options menu window that lets the player to change the resolution, window mode and keybinging.

Definition at line 18 of file OptionsWindow.hpp.

6.72.2 Constructor & Destructor Documentation

6.72.2.1 OptionsWindow::OptionsWindow ( )

Constructor.

Definition at line 8 of file OptionsWindow.cpp.

6.72.2.2 OptionsWindow::∼OptionsWindow ( ) [default]

Destructor.

6.72.3 Member Function Documentation

6.72.3.1 void OptionsWindow::add_start_parameters (Ogre::RenderWindow ∗ *window,* Ogre::Viewport ∗ *view,* CEGUI::OgreRenderer ∗ *renderer*)

Saves pointers to the window, view and renderer and sets starting values for the resolution, window mode etc.

Parameters

Window	that the game is rendered to.
Viewport	of the window.
Renderer	used by CEGUI.

Definition at line 13 of file OptionsWindow.cpp.

6.72.3.2 void OptionsWindow::apply_ ( ) [private]

Applies the graphical changes and saves key bindings into a script that allows persistent key binds.

This will make sure that key bindings are persistent.

Definition at line 307 of file OptionsWindow.cpp.

6.72.3.3 const std::string & OptionsWindow::get_key_bind_name_ (KEY_BIND_ACTION::VAL *action*) [private]

Returns the name of the key a given action is bound to.

Parameters

The	action.

Definition at line 425 of file OptionsWindow.cpp.

6.72.3.4 void OptionsWindow::init_ ( ) [override], [protected], [virtual]

Initializes the CEGUI window.

Implements GUIWindow.

Definition at line 64 of file OptionsWindow.cpp.

6.72.3.5 bool OptionsWindow::key_pressed (CEGUI::Key::Scan *key*)

Registers a key press for keybinding.

Returns true if the click was consumed, false otherwise.

Parameters

Key	pressed.

Definition at line 26 of file OptionsWindow.cpp.

6.72.3.6 void OptionsWindow::set_key_bind (KEY_BIND_ACTION::VAL *action,* CEGUI::Key::Scan *key*)

Binds a given action to a given key.

Parameters

Action	to be bound.
Key	to be bound.

Definition at line 58 of file OptionsWindow.cpp.

6.72.3.7 void OptionsWindow::update_labels_ ( ) `[private]`

Updates the button texts and labels to reflect the new settings.

Definition at line 348 of file OptionsWindow.cpp.

6.72.4 Member Data Documentation

6.72.4.1 std::array<ActionFuncPtr, KEY_BIND_ACTION::COUNT> OptionsWindow::actions_ `[private]`

Action functions assigned to actions.

Definition at line 124 of file OptionsWindow.hpp.

6.72.4.2 KEY_BIND_ACTION::VAL OptionsWindow::currently_binded_action_ `[private]`

Saves the currently binded action so that the next key press can be used.

Definition at line 140 of file OptionsWindow.hpp.

6.72.4.3 bool OptionsWindow::fullscreen_ `[private]`

True for fullscreen, false for windowed mode.

Definition at line 114 of file OptionsWindow.hpp.

6.72.4.4 std::array< CEGUI::Key::Scan, KEY_BIND_ACTION::COUNT > OptionsWindow::key_binds_ [private]

Keys bound to actions.

Definition at line 129 of file OptionsWindow.hpp.

6.72.4.5 std::map< CEGUI::Key::Scan, std::string > OptionsWindow::key_names_ [private]

This map connects key codes with their names (used for key bind buttons).

Definition at line 119 of file OptionsWindow.hpp.

6.72.4.6 Ogre::RenderWindow* OptionsWindow::render_window_ [private]

Window the game is rendered to.

Definition at line 94 of file OptionsWindow.hpp.

6.72.4.7 CEGUI::OgreRenderer* OptionsWindow::renderer_ [private]

Renderer used by CEGUI (used to sync resolution between game and UI).

Definition at line 104 of file OptionsWindow.hpp.

6.72.4.8 std::map< std::string, std::tuple< tdt::uint, tdt::uint > > OptionsWindow::resolutions_ [private]

Resolution strings and their assigned dimensions.

Definition at line 134 of file OptionsWindow.hpp.

6.72.4.9 Ogre::Viewport* OptionsWindow::view_ [private]

Main viewport of the render window.

Definition at line 99 of file OptionsWindow.hpp.

6.72.4.10 tdt::uint OptionsWindow::width_ [private]

Dimensions of the display resolution.

Definition at line 109 of file OptionsWindow.hpp.

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

- gui/OptionsWindow.hpp
- gui/OptionsWindow.cpp

6.73 PathfindingComponent Struct Reference

Holds data related to the entity's current path.

```
#include <Components.hpp>
```

Public Member Functions

- **PathfindingComponent** (std::string &&b="ERROR", tdt::uint tar=0, tdt::uint last=0)
- **PathfindingComponent** (const PathfindingComponent &)=default
- **PathfindingComponent** (PathfindingComponent &&)=default
- PathfindingComponent & **operator=** (const PathfindingComponent &)=default
- PathfindingComponent & **operator=** (PathfindingComponent &&)=default

Public Attributes

- tdt::uint **target_id**
- tdt::uint **last_id**
- std::deque< tdt::uint > **path_queue**
- std::string **blueprint**

Static Public Attributes

- static constexpr int **type** = 14

6.73.1 Detailed Description

Holds data related to the entity's current path.

Definition at line 367 of file Components.hpp.

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

- Components.hpp

6.74 PhysicsComponent Struct Reference

Components.

```
#include <Components.hpp>
```

Public Member Functions

- **PhysicsComponent** (bool s=false, Ogre::Vector3 pos=Ogre::Vector3{0, 0, 0}, tdt::real hh=0.f)
- **PhysicsComponent** (const PhysicsComponent &)=default
- **PhysicsComponent** (PhysicsComponent &&)=default
- PhysicsComponent & **operator=** (const PhysicsComponent &)=default
- PhysicsComponent & **operator=** (PhysicsComponent &&)=default

Public Attributes

- bool **solid**
- Ogre::Vector3 **position**
- tdt::real **half_height**

Static Public Attributes

- static constexpr int **type** = 0

6.74.1 Detailed Description

Components.

Note

To be able to manually create components without blueprints, all components must have either default constructors or constructors with default values for all parameters, to be able to back components up, they need to provide a copy constructor.
As constructors are called in the EntitySystem::load_component function, strings will be read from Lua and immediately discarded. Hence it was decided to make the string parameters accepted as rvalues for faster construction. Holds info related to physical interaction of an entity with the rest of the game world.

Definition at line 34 of file Components.hpp.

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

- Components.hpp

6.75 Player Class Reference

Auxiliary class representing the player's resources, since the nature of the game does not allow creating player as an entity (only one player can exist at a time) and it allows for easy GUI modifications when the amount of these resources changes (had player been an entity, ever gold/mana addition would require a check which would be true only in minimal number of cases).

```
#include <Player.hpp>
```

Public Member Functions

- ∼Player ()

  Destructor.
- void add_gold (tdt::uint)

  Adds gold to the player's gold stash.
- bool sub_gold (tdt::uint)

  Removes gold from the player's gold stash if possible, returns true if the player has enough, false otherwise.
- void add_mana (tdt::uint)

  Adds mana to the player's mana pool.
- bool sub_mana (tdt::uint)

  Removes mana from the player's mana pool if possible, returns true if the player has enough, false otherwise.
- void add_max_mana (tdt::uint)

  Breif: Increases the mana capacity of the player by a given amount.
- bool sub_max_mana (tdt::uint)

  Decreases the mana capacity of the player by a given amount if possible.
- void add_mana_regen (tdt::uint)

  Increases the player's mana regeneration by a given amount.
- bool sub_mana_regen (tdt::uint)

  Decreases the player's mana regeneration by a given amount if possible.
- void add_max_unit (tdt::uint)

  Adds max units to the player's unit amount.
- bool sub_max_unit (tdt::uint)

  Removes max units from the player's unit amount if possible, returns true if the player has enough, false otherwise.
- void add_curr_unit (tdt::uint)

  Adds current units to the player's unit amount.
- bool sub_curr_unit (tdt::uint)

  Removes current units from the player's unit amount if possible, returns true if the player has enough, false otherwise.
- tdt::uint get_gold () const

  Returns the amount of gold the player currently has.
- tdt::uint get_mana () const

  Returns the amount of mana the player currently has.
- tdt::uint get_max_mana () const

  Returns the mana capacity of the player.
- tdt::uint get_mana_regen () const

  Returns the value of the player's mana regeneration.
- void reset ()

  Sets all of the player's stats to their default values.
- void nulify_all_stats ()

  Sets all of the player's stats to zero (used for loading).
- void init (EntitySystem ∗)

  Initializes the player class.
- void set_initial_unlocks (const std::vector< std::string > &, const std::vector< std::string > &)

  Sets the spells and buildings that are unlocked from the start.
- const std::vector< std::string > & get_initial_spells () const

  Returns a vector of all spells that are unlocked at the start of a new game.
- const std::vector< std::string > & get_initial_buildings () const

  Returns a vector of all buildings that are unlocked at the start of a new game.

Static Public Member Functions

- static Player & instance ()

  Returns a reference to the singleton instance.

Private Member Functions

- Player ()

  Constructor.

Private Attributes

- tdt::uint gold_

  Amount of total gold the player currently has to spend.
- tdt::uint mana_

  Amount of mana the player currently has to spend.
- tdt::uint max_mana_

  Max amount of mana the player can have.
- tdt::uint mana_regen_

  Amount of mana that is added to the player's mana pool on every regen tick.
- tdt::uint units_curr_

  Amount of currently alive units.
- tdt::uint units_max_

  Amount of all units (even those that are respawning).
- const tdt::uint uint_max_

  Helper value for overflow checking, contains tdt::uint max value.
- EntitySystem ∗ entities_

  Used to subtract gold from gold vault so that player gold and vault gold is synchronized.
- std::vector< std::string > initial_spell_unlocks_

  Holds the names of the spells that are available to the player from the beggining.
- std::vector< std::string > initial_building_unlocks_

  Holds the names of the buildings that are available to the player from the beggining.

6.75.1 Detailed Description

Auxiliary class representing the player's resources, since the nature of the game does not allow creating player as an entity (only one player can exist at a time) and it allows for easy GUI modifications when the amount of these resources changes (had player been an entity, ever gold/mana addition would require a check which would be true only in minimal number of cases).

Definition at line 15 of file Player.hpp.

6.75.2 Constructor & Destructor Documentation

6.75.2.1 Player::∼Player ( ) [inline]

Destructor.

Definition at line 21 of file Player.hpp.

6.75.2.2 Player::Player ( ) [private]

Constructor.

Definition at line 30 of file Player.cpp.

6.75.3 Member Function Documentation

6.75.3.1 void Player::add_curr_unit (tdt::uint *val*)

Adds current units to the player's unit amount.

Parameters

Amount	to add.

Definition at line 174 of file Player.cpp.

6.75.3.2 void Player::add_gold (tdt::uint *val*)

Adds gold to the player's gold stash.

Parameters

Amount	to add.

Definition at line 35 of file Player.cpp.

6.75.3.3 void Player::add_mana (tdt::uint *val*)

Adds mana to the player's mana pool.

Parameters

Amount	to add.

Definition at line 71 of file Player.cpp.

6.75.3.4 void Player::add_mana_regen (tdt::uint *val*)

Increases the player's mana regeneration by a given amount.

Parameters

Amount	to increase by.

Definition at line 126 of file Player.cpp.

6.75.3.5 void Player::add_max_mana (tdt::uint *val*)

Breif: Increases the mana capacity of the player by a given amount.

Parameters

Amount	to add.

Definition at line 101 of file Player.cpp.

6.75.3.6 void Player::add_max_unit (tdt::uint *val*)

Adds max units to the player's unit amount.

Parameters

Amount	to add.

Definition at line 151 of file Player.cpp.

6.75.3.7 tdt::uint Player::get_gold ( ) const

Returns the amount of gold the player currently has.

Definition at line 197 of file Player.cpp.

6.75.3.8 const std::vector< std::string > & Player::get_initial_buildings ( ) const

Returns a vector of all buildings that are unlocked at the start of a new game.

Definition at line 25 of file Player.cpp.

6.75.3.9 const std::vector< std::string > & Player::get_initial_spells ( ) const

Returns a vector of all spells that are unlocked at the start of a new game.

Definition at line 20 of file Player.cpp.

6.75.3.10 tdt::uint Player::get_mana ( ) const

Returns the amount of mana the player currently has.

Definition at line 202 of file Player.cpp.

6.75.3.11 tdt::uint Player::get_mana_regen ( ) const

Returns the value of the player's mana regeneration.

Definition at line 212 of file Player.cpp.

6.75.3.12 tdt::uint Player::get_max_mana ( ) const

Returns the mana capacity of the player.

Definition at line 207 of file Player.cpp.

6.75.3.13 void Player::init (EntitySystem $*$ *ents*)

Initializes the player class.

Parameters

EntitySystem	used to get gold vaults to subtract gold from.

Definition at line 8 of file Player.cpp.

6.75.3.14 static Player& Player::instance ( ) [inline], [static]

Returns a reference to the singleton instance.

Definition at line 140 of file Player.hpp.

6.75.3.15 void Player::nulify_all_stats ( )

Sets all of the player's stats to zero (used for loading).

Definition at line 229 of file Player.cpp.

6.75.3.16 void Player::reset ( )

Sets all of the player's stats to their default values.

Definition at line 217 of file Player.cpp.

6.75.3.17 void Player::set_initial_unlocks (const std::vector< std::string > & *spells,* const std::vector< std::string > & *buildings*)

Sets the spells and buildings that are unlocked from the start.

Parameters

Spell	unlocks.
Building	unlocks.

Definition at line 14 of file Player.cpp.

6.75.3.18 bool Player::sub_curr_unit (tdt::uint *val*)

Removes current units from the player's unit amount if possible, returns true if the player has enough, false otherwise.

Parameters

Amount	to remove.

Definition at line 184 of file Player.cpp.

6.75.3.19 bool Player::sub_gold (tdt::uint *val*)

Removes gold from the player's gold stash if possible, returns true if the player has enough, false otherwise.

Parameters

Amount	to remove.

Definition at line 44 of file Player.cpp.

6.75.3.20 bool Player::sub_mana (tdt::uint *val*)

Removes mana from the player's mana pool if possible, returns true if the player has enough, false otherwise.

Parameters

Amount	to remove.

Definition at line 87 of file Player.cpp.

6.75.3.21 bool Player::sub_mana_regen (tdt::uint *val*)

Decreases the player's mana regeneration by a given amount if possible.

Returns true if the player has enough, false otherwise.

Parameters

Amount	to decrease by.

Definition at line 137 of file Player.cpp.

6.75.3.22 bool Player::sub_max_mana (tdt::uint *val*)

Decreases the mana capacity of the player by a given amount if possible.

Returns true if the player has enough, false otherwise.

Parameters

Amount	to decrease by.

Definition at line 112 of file Player.cpp.

6.75.3.23 bool Player::sub_max_unit (tdt::uint *val*)

Removes max units from the player's unit amount if possible, returns true if the player has enough, false otherwise.

Parameters

Amount	to remove.

Definition at line 161 of file Player.cpp.

6.75.4 Member Data Documentation

6.75.4.1 EntitySystem∗ Player::entities_ `[private]`

Used to subtract gold from gold vault so that player gold and vault gold is synchronized.

Definition at line 219 of file Player.hpp.

6.75.4.2 tdt::uint Player::gold_ `[private]`

Amount of total gold the player currently has to spend.

Note

Only gold stored in vaults is counted, not that on units.

Definition at line 182 of file Player.hpp.

6.75.4.3 std::vector<std::string> Player::initial_building_unlocks_ [private]

Holds the names of the buildings that are available to the player from the beggining.

(That is, they are unlocked in the Lua script they are defined.)

Definition at line 233 of file Player.hpp.

6.75.4.4 std::vector<std::string> Player::initial_spell_unlocks_ [private]

Holds the names of the spells that are available to the player from the beggining.

(That is, they are unlocked in the Lua script they are defined.)

Definition at line 226 of file Player.hpp.

6.75.4.5 tdt::uint Player::mana_ [private]

Amount of mana the player currently has to spend.

Definition at line 187 of file Player.hpp.

6.75.4.6 tdt::uint Player::mana_regen_ [private]

Amount of mana that is added to the player's mana pool on every regen tick.

Definition at line 198 of file Player.hpp.

6.75.4.7 tdt::uint Player::max_mana_ [private]

Max amount of mana the player can have.

Definition at line 192 of file Player.hpp.

6.75.4.8 const tdt::uint Player::uint_max_ [private]

Helper value for overflow checking, contains tdt::uint max value.

Definition at line 213 of file Player.hpp.

6.75.4.9 tdt::uint Player::units_curr_ [private]

Amount of currently alive units.

Definition at line 203 of file Player.hpp.

6.75.4.10 tdt::uint Player::units_max_ `[private]`

Amount of all units (even those that are respawning).

Definition at line 208 of file Player.hpp.

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

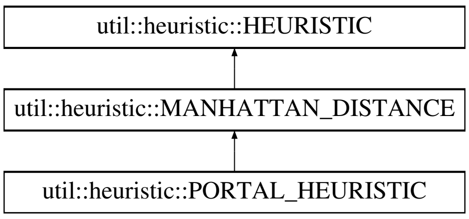
- tools/Player.hpp
- tools/Player.cpp

6.76 util::heuristic::PORTAL_HEURISTIC Struct Reference

Variation of the Manhattan distance heuristic that takes portals into accounts.

`#include <PathfindingAlgorithms.hpp>`

Inheritance diagram for util::heuristic::PORTAL_HEURISTIC:


util::heuristic::HEURISTIC
util::heuristic::MANHATTAN_DISTANCE
util::heuristic::PORTAL_HEURISTIC

Public Member Functions

- **PORTAL_HEURISTIC** (EntitySystem &ents)
- tdt::real **get_cost** (tdt::uint id1, tdt::uint id2) override

Private Member Functions

- std::tuple< tdt::uint, tdt::uint > get_closest_portal (tdt::uint id)
  Returns the nodes that have the closest portal pair from a given entity on them.

Additional Inherited Members

6.76.1 Detailed Description

Variation of the Manhattan distance heuristic that takes portals into accounts.

Note

This heuristic won't help with complex chains of portals. For that, the BEST_PATH path type would be needed to check every single portal combination. (But for basic portal usage, this heuristic works fine.)

Definition at line 267 of file PathfindingAlgorithms.hpp.

6.76.2 Member Function Documentation

6.76.2.1 std::tuple<tdt::uint, tdt::uint> util::heuristic::PORTAL_HEURISTIC::get_closest_portal (tdt::uint *id*) `[inline]`, `[private]`

Returns the nodes that have the closest portal pair from a given entity on them.

Parameters

ID	of the entity.

Definition at line 298 of file PathfindingAlgorithms.hpp.

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

- tools/PathfindingAlgorithms.hpp

6.77 PortalComponent Struct Reference

Dummy component that signals that an entity having it is a portal - which is used in pathfinding.

```
#include <Components.hpp>
```

Static Public Attributes

- static constexpr int **type** = 39

6.77.1 Detailed Description

Dummy component that signals that an entity having it is a portal - which is used in pathfinding.

Definition at line 899 of file Components.hpp.

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

- Components.hpp

6.78 PriceComponent Struct Reference

Represents either gold or mana cost of an entity.

```
#include <Components.hpp>
```

Public Member Functions

- **PriceComponent** (tdt::uint p=0)
- **PriceComponent** (const PriceComponent &)=default
- **PriceComponent** (PriceComponent &&)=default
- PriceComponent & **operator=** (const PriceComponent &)=default
- PriceComponent & **operator=** (PriceComponent &&)=default

Public Attributes

- tdt::uint **price**

Static Public Attributes

- static constexpr int **type** = 23

6.78.1 Detailed Description

Represents either gold or mana cost of an entity.

Definition at line 568 of file Components.hpp.

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

- Components.hpp

6.79 ProductComponent Struct Reference

References the producer of the entity that has this component.

```
#include <Components.hpp>
```

Public Member Functions

- **ProductComponent** (tdt::uint prod_id=Component::NO_ENTITY)
- **ProductComponent** (const ProductComponent &)=default
- **ProductComponent** (ProductComponent &&)=default
- ProductComponent & **operator=** (const ProductComponent &)=default
- ProductComponent & **operator=** (ProductComponent &&)=default

Public Attributes

- tdt::uint **producer**

Static Public Attributes

- static constexpr int **type** = 13

6.79.1 Detailed Description

References the producer of the entity that has this component.

(Producer == building/tile that spawned it.)

Definition at line 348 of file Components.hpp.

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

- Components.hpp

6.80 ProductionComponent Struct Reference

Allows scheduled production of new entities (spawners) of a given type up to a maximum amount.

```
#include <Components.hpp>
```

Public Member Functions

- **ProductionComponent** (std::string &&b="ERROR", tdt::uint l=1, tdt::real cd=0.f)
- **ProductionComponent** (const ProductionComponent &)=default
- **ProductionComponent** (ProductionComponent &&)=default
- ProductionComponent & **operator=** (const ProductionComponent &)=default
- ProductionComponent & **operator=** (ProductionComponent &&)=default

Public Attributes

- std::string **product_blueprint**
- tdt::uint **curr_produced**
- tdt::uint **max_produced**
- tdt::real **cooldown**
- tdt::real **curr_cd**

Static Public Attributes

- static constexpr int **type** = 11

6.80.1 Detailed Description

Allows scheduled production of new entities (spawners) of a given type up to a maximum amount.

Definition at line 294 of file Components.hpp.

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

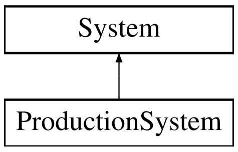
- Components.hpp

6.81 ProductionSystem Class Reference

System taking care of entities spawned by buildings and the spawn counts allowing for a constant amount of entities (related to the number of buildings spawning entities of that blueprint table).

```
#include <ProductionSystem.hpp>
```

Inheritance diagram for ProductionSystem:



Public Member Functions

- ProductionSystem (EntitySystem &)
  Constructor.
- ∼ProductionSystem ()
  Destructor.
- void update (tdt::real) override
  Iterates over all buildings and when possible, spawns new entities.
- void spawn_entity (tdt::uint, const std::string &)
  Spawns a single entity created by a building.
- void set_time_multiplier (tdt::real)
  Sets the time value by which the frame times are multiplied when added to the production timers.
- tdt::real get_time_multiplier ()
  Returns the time value by which the frame times are multiplied when added to the production timers.

Private Attributes

- EntitySystem & entities_
  Reference to the game's entity system (component retrieval).
- Grid & grid_
  Reference to the game's pathfinding grid (spawn positioning).
- tdt::real time_multiplier_
  Allows to speed up/slow down the production of all buildings.

6.81.1 Detailed Description

System taking care of entities spawned by buildings and the spawn counts allowing for a constant amount of entities (related to the number of buildings spawning entities of that blueprint table).

Definition at line 14 of file ProductionSystem.hpp.

6.81.2 Constructor & Destructor Documentation

6.81.2.1 ProductionSystem::ProductionSystem (EntitySystem & *ents*)

Constructor.

Parameters

Reference	to the game's entity system.

Definition at line 7 of file ProductionSystem.cpp.

6.81.2.2 ProductionSystem::∼ProductionSystem ( ) [inline]

Destructor.

Definition at line 26 of file ProductionSystem.hpp.

6.81.3 Member Function Documentation

6.81.3.1 tdt::real ProductionSystem::get_time_multiplier ( )

Returns the time value by which the frame times are multiplied when added to the production timers.

Definition at line 125 of file ProductionSystem.cpp.

6.81.3.2 void ProductionSystem::set_time_multiplier (tdt::real *val*)

Sets the time value by which the frame times are multiplied when added to the production timers.

Parameters

The	new time multiplier.

Definition at line 120 of file ProductionSystem.cpp.

6.81.3.3 void ProductionSystem::spawn_entity (tdt::uint *producer,* const std::string & *blueprint*)

Spawns a single entity created by a building.

Parameters

ID	of the building.
Name	of the blueprint table of the spawned entity.

This checks all edges of the building to find a free spot for the entity to spawn on.

Definition at line 29 of file ProductionSystem.cpp.

6.81.3.4 void ProductionSystem::update (tdt::real *delta*) `[override]`, `[virtual]`

Iterates over all buildings and when possible, spawns new entities.

Parameters

Time	since last frame.

Implements System.

Definition at line 11 of file ProductionSystem.cpp.

6.81.4 Member Data Documentation

6.81.4.1 EntitySystem& ProductionSystem::entities_ [private]

Reference to the game's entity system (component retrieval).

Definition at line 58 of file ProductionSystem.hpp.

6.81.4.2 Grid& ProductionSystem::grid_ [private]

Reference to the game's pathfinding grid (spawn positioning).

Definition at line 63 of file ProductionSystem.hpp.

6.81.4.3 tdt::real ProductionSystem::time_multiplier_ [private]

Allows to speed up/slow down the production of all buildings.

Definition at line 68 of file ProductionSystem.hpp.

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

- systems/ProductionSystem.hpp
- systems/ProductionSystem.cpp

6.82 util::path_type::RANDOM_PATH< UPPER > Struct Template Reference

Finds a random path by returning true only when a random number in the range (0, UPPER) is equal to 0.

```
#include <PathfindingAlgorithms.hpp>
```

Static Public Member Functions

- static bool **return_path** ()

6.82.1 Detailed Description

template<int UPPER>
struct util::path_type::RANDOM_PATH< UPPER >

Finds a random path by returning true only when a random number in the range (0, UPPER) is equal to 0.

(UPPER is specialized as a template parameter.)

Definition at line 169 of file PathfindingAlgorithms.hpp.

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

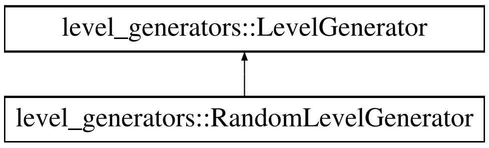
- tools/PathfindingAlgorithms.hpp

6.83 level_generators::RandomLevelGenerator Class Reference

Level generator that uses simple RNG approach (counts the number of gold neighbours and increases the chance to spawn a gold deposit if needed).

```
#include <LevelGenerators.hpp>
```

Inheritance diagram for level_generators::RandomLevelGenerator:


level_generators::LevelGenerator
level_generators::RandomLevelGenerator

Public Member Functions

- RandomLevelGenerator (EntitySystem &, tdt::uint)
  Constructor.
- void generate (tdt::uint, tdt::uint, WaveSystem &) override
  Generates a level with the given dimensions using an RNG approach.

Additional Inherited Members

6.83.1 Detailed Description

Level generator that uses simple RNG approach (counts the number of gold neighbours and increases the chance to spawn a gold deposit if needed).

Definition at line 60 of file LevelGenerators.hpp.

6.83.2 Constructor & Destructor Documentation

6.83.2.1 level_generators::RandomLevelGenerator::RandomLevelGenerator (EntitySystem & *ents*, tdt::uint *c*)

Constructor.

Parameters

Entity	system that contains the level's entities.
Number	of iterations done while generating.

Definition at line 14 of file LevelGenerators.cpp.

6.83.3 Member Function Documentation

6.83.3.1 void level_generators::RandomLevelGenerator::generate (tdt::uint *width*, tdt::uint *height*, WaveSystem & *wsystem*) [override], [virtual]

Generates a level with the given dimensions using an RNG approach.

Parameters

Width	of the level.
Height	of the level.
Wave	system that will have it's spawn nodes set.

0 == free space 1 == wall 2 == gold deposit 3 == border 4 == walkway 5 == light source 6 == throne 7 == vault 8 == mine

Implements level_generators::LevelGenerator.

Definition at line 18 of file LevelGenerators.cpp.

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

- tools/LevelGenerators.hpp
- tools/LevelGenerators.cpp

6.84 RayCaster Class Reference

Manages polygon precise raycasting used with half walls that have empty spaces in their bounding boxes.

```
#include <RayCaster.hpp>
```

Public Member Functions

- RayCaster (Ogre::SceneManager &)
    Constructor.
- std::pair< bool, tdt::real > cast (const Ogre::Vector3 &, const Ogre::Vector3 &, const std::string &="") const
    Casts a ray that checks for polygon level collisions on it's way.

Private Member Functions

- void get_info (const Ogre::Entity &, tdt::uint &, tdt::uint &, std::vector< Ogre::Vector3 > &, std::vector< tdt↵ ::uint > &, const Ogre::Vector3 &, const Ogre::Quaternion &, const Ogre::Vector3 &) const

    Returns information about vertices of an entity by modifying it's size_t and vector parameters.

Private Attributes

- Ogre::RaySceneQuery ∗ query_

    Query used for the collision ray cast.

6.84.1 Detailed Description

Manages polygon precise raycasting used with half walls that have empty spaces in their bounding boxes.

Note

Strongly inspired by `http://www.ogre3d.org/tikiwiki/Raycasting+to+the+polygon+level` from the official Ogre3D wiki, big thanks to all contributors.

Definition at line 14 of file RayCaster.hpp.

6.84.2 Constructor & Destructor Documentation

6.84.2.1 RayCaster::RayCaster (Ogre::SceneManager & *mgr*)

Constructor.

Parameters

Scene	manager that is used to create the ray query.

Definition at line 6 of file RayCaster.cpp.

6.84.3 Member Function Documentation

6.84.3.1 std::pair< bool, tdt::real > RayCaster::cast (const Ogre::Vector3 & *start*, const Ogre::Vector3 & *dir*, const std::string & *target* = "") const

Casts a ray that checks for polygon level collisions on it's way.

Returns a pair of a bool, signaling whether a hit was made and a distance to the place of the collision, if any occured.

Parameters

Starting	position of the ray.
Direction	of the ray.
Name	of the entity to be ignored (the target) if it's a wall.

Definition at line 13 of file RayCaster.cpp.

6.84.3.2 void RayCaster::get_info (const Ogre::Entity & *ent,* tdt::uint & *v_count,* tdt::uint & *i_count,* std::vector< Ogre::Vector3 > & *verts,* std::vector< tdt::uint > & *inds,* const Ogre::Vector3 & *position,* const Ogre::Quaternion & *orientation,* const Ogre::Vector3 & *scale*) const `[private]`

Returns information about vertices of an entity by modifying it's size_t and vector parameters.

Parameters

Entity	to be checked.
Number	of vertices, will be set inside the function.
Number	of indices, will be set inside the function.
Vector	of vertex point positions, will be filled inside the function.
Vector	of indices of the vertex points inside the vertex vector above.
Position	of the entity.
Orientation	of the entity.
Scale	of the entity.

Definition at line 73 of file RayCaster.cpp.

6.84.4 Member Data Documentation

6.84.4.1 Ogre::RaySceneQuery* RayCaster::query_ `[private]`

Query used for the collision ray cast.

Definition at line 38 of file RayCaster.hpp.

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

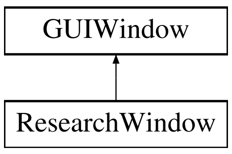
- tools/RayCaster.hpp
- tools/RayCaster.cpp

6.85 ResearchWindow Class Reference

Class that represents the research window in the game, which allows the player to unlock new buildings and spells.

`#include <ResearchWindow.hpp>`

Inheritance diagram for ResearchWindow:



Public Member Functions

- ResearchWindow ()

  Constructor.
- ∼ResearchWindow ()=default

  Destructor.
- void unlock (tdt::uint, tdt::uint)

  Unlocks a single research point at a given position in the research table.
- void dummy_unlock (tdt::uint, tdt::uint)

  Unlocks a single research point without activating it.
- const std::array< bool, 42 > & get_unlocked () const

  Returns a reference to the unlock table, used for serialization.
- void show (tdt::uint, tdt::uint, bool=true)

  Shows a single research point at a given position in the research table.
- void free_research ()

  Cheat that changes the price of any research to 0.
- void research_all ()

  Cheat that unlocks all research points.
- void reset_research ()

  Resets the research state so that all items can be unlocked again.

Protected Member Functions

- void init_ () override

  Initializes this window.

Private Member Functions

- tdt::uint get_price_ (tdt::uint, tdt::uint)

  Returns the price in gold of a research point at the given position in the research table.
- bool is_unlocked_ (tdt::uint, tdt::uint)

  Returns true if the research point at the given position is unlocked, false otherwise.

Private Attributes

- lpp::Script ∗ script_

  Pointer to the Lua Script used for easier access.
- const tdt::uint rows_ {6}

  Number of rows that the research table has.
- const tdt::uint cols_ {7}

  Number of columns that the research table has.
- std::array< tdt::uint, 42 > prices_

  Contains prices of the individual research points.
- std::array< bool, 42 > unlocked_

  Contains information about the lock status of the individual research points.

Friends

- class **GameSerializer**

Additional Inherited Members

6.85.1 Detailed Description

Class that represents the research window in the game, which allows the player to unlock new buildings and spells.

Note

Because this class is so tightly bound to Lua, the indices have to be adjusted when accessing the prices_ and unlocked_ arrays, since Lua uses indices starting at one when handling arrays.

Definition at line 18 of file ResearchWindow.hpp.

6.85.2 Constructor & Destructor Documentation

6.85.2.1 ResearchWindow::ResearchWindow ( )

Constructor.

Definition at line 6 of file ResearchWindow.cpp.

6.85.2.2 ResearchWindow::∼ResearchWindow ( ) `[default]`

Destructor.

6.85.3 Member Function Documentation

6.85.3.1 void ResearchWindow::dummy_unlock (tdt::uint *i*, tdt::uint *j*)

Unlocks a single research point without activating it.

Used for serialization.

Parameters

Row	number.
Column	number.

Definition at line 28 of file ResearchWindow.cpp.

6.85.3.2 void ResearchWindow::free_research ( )

Cheat that changes the price of any research to 0.

Definition at line 51 of file ResearchWindow.cpp.

6.85.3.3 tdt::uint ResearchWindow::get_price_ (tdt::uint *i*, tdt::uint *j*) [private]

Returns the price in gold of a research point at the given position in the research table.

Parameters

Row	number.
Column	number.

Definition at line 154 of file ResearchWindow.cpp.

6.85.3.4 const std::array< bool, 42 > & ResearchWindow::get_unlocked ( ) const

Returns a reference to the unlock table, used for serialization.

Definition at line 41 of file ResearchWindow.cpp.

6.85.3.5 void ResearchWindow::init_ ( ) [override], [protected], [virtual]

Initializes this window.

Makes sure all buttons can be used in the beggining.

Research button initialization.

Note

AlfiskoSkin does not have tooltip support, maybe create own skin? butt->setTooltipText(script_->call<std↩ ::string, tdt::uint, tdt::uint>("game.gui.research.get_tooltip", i, j));

Implements GUIWindow.

Definition at line 95 of file ResearchWindow.cpp.

6.85.3.6 bool ResearchWindow::is_unlocked_ (tdt::uint *i*, tdt::uint *j*) [private]

Returns true if the research point at the given position is unlocked, false otherwise.

Parameters

Row	number.
Column	number.

Definition at line 163 of file ResearchWindow.cpp.

6.85.3.7 void ResearchWindow::research_all ( )

Cheat that unlocks all research points.

Definition at line 57 of file ResearchWindow.cpp.

6.85.3.8 void ResearchWindow::reset_research ( )

Resets the research state so that all items can be unlocked again.

Definition at line 68 of file ResearchWindow.cpp.

6.85.3.9 void ResearchWindow::show (tdt::uint *i*, tdt::uint *j*, bool *val* = `true`)

Shows a single research point at a given position in the research table.

Parameters

Row	number.
Column	number.
If	true, shows the button, otherwise it hides it.

Definition at line 46 of file ResearchWindow.cpp.

6.85.3.10 void ResearchWindow::unlock (tdt::uint *i*, tdt::uint *j*)

Unlocks a single research point at a given position in the research table.

Parameters

Row	number.
Column	number.

Definition at line 10 of file ResearchWindow.cpp.

6.85.4 Member Data Documentation

6.85.4.1 const tdt::uint ResearchWindow::cols_ {7} `[private]`

Number of columns that the research table has.

Definition at line 115 of file ResearchWindow.hpp.

6.85.4.2 std::array<tdt::uint, 42> ResearchWindow::prices_ `[private]`

Contains prices of the individual research points.

Used to avoid unnecessary Lua lookups.

Definition at line 121 of file ResearchWindow.hpp.

6.85.4.3 const tdt::uint ResearchWindow::rows_ {6} [private]

Number of rows that the research table has.

Definition at line 110 of file ResearchWindow.hpp.

6.85.4.4 lpp::Script* ResearchWindow::script_ [private]

Pointer to the Lua Script used for easier access.

Definition at line 105 of file ResearchWindow.hpp.

6.85.4.5 std::array<bool, 42> ResearchWindow::unlocked_ [private]

Contains information about the lock status of the individual research points.

Used to avoid unnecessary Lua lookups.

Definition at line 128 of file ResearchWindow.hpp.

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

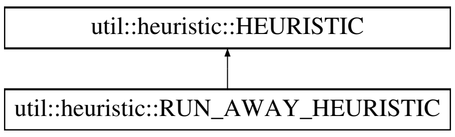
- gui/ResearchWindow.hpp
- gui/ResearchWindow.cpp

6.86 util::heuristic::RUN_AWAY_HEURISTIC Struct Reference

Used by entities that want to run away from an enemy.

`#include <PathfindingAlgorithms.hpp>`

Inheritance diagram for util::heuristic::RUN_AWAY_HEURISTIC:


util::heuristic::HEURISTIC
util::heuristic::RUN_AWAY_HEURISTIC

Public Member Functions

- **RUN_AWAY_HEURISTIC** (EntitySystem &ents, tdt::uint from)
- tdt::real **get_cost** (tdt::uint id1, tdt::uint id2) override

Private Attributes

- tdt::uint **from_**

Additional Inherited Members

6.86.1 Detailed Description

Used by entities that want to run away from an enemy.

Definition at line 240 of file PathfindingAlgorithms.hpp.

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

- tools/PathfindingAlgorithms.hpp

6.87 lpp::Script Class Reference

Class representing a Lua script, allows to register C++ functions, load variables, call functions, execute strings containing Lua code and other functionalities.

```
#include <LppScript.hpp>
```

Public Types

- using **state** = lua_State ∗
- using **regs** = luaL_Reg

Public Member Functions

- Script (const Script &)=delete

  Copying this script might cause the Lua state get closed when one of the copies gets destroyed and would cause the game to be unable to use it's scripting engine (and thus crashing probably).
- ∼Script ()

  Destructor, closes the Lua virtual machine.
- state get_state ()

  Returns the lua state representing the Lua virtual machine.
- void execute (const std::string &)

  Executes a given string from within Lua.
- void register_function (const std::string &, lua_CFunction)

  Registers a C++ function which can then be used from within Lua.
- void load (const std::string &)

  Loads, compiles and executes a Lua script.
- bool is_nil (const std::string &)

  Returns true if a given value is nil, false otherwise.
- template<typename T >
  T get (const std::string &name)

  Retrieves and returns a value from Lua.
- template<typename Result , typename... Args>
  Result call (const std::string &fname, Args...as)

  Calls a given Lua function.

- template<typename Result >
  Result call (const std::string &fname)

  Calls a given Lua function.
- template<typename T >
  void set (const std::string &name, T val)

  Sets a given variable to a given value.
- template<typename T >
  std::vector< T > get_vector (const std::string &name)

  Retrieves a Lua array table (integer indexing) in the form of a C++ vector.
- std::string get_stack_contents ()

  Returns string representation of the Lua stack.
- void reload_all_scripts ()

  Reloads all script files that have been previously loaded.
- template<>
  void **set** (const std::string &name, bool val)

Static Public Member Functions

- static Script & instance ()

  Returns a reference to the lpp::Script singleton.

Private Member Functions

- Script ()

  Constructor, kept private because of the use of the singleton pattern.
- std::string get_field_to_stack (const std::string &)

  Gets a nested value (inside a table hierarchy) on top of the stack and returns the name of the final variable (without table prefixes).
- void clear_stack ()

  Pops everything off the stack.
- template<typename T >
  T get_ (const std::string &name="unknown")

  Returns the value stored on top of the stack.
- template<typename Arg , typename... Args>
  int push_args (Arg a, Args...as)

  Pushed a variadic list of arguments onto the stack to be passed as arguments to a Lua function call, returns the amount of arguments pushed onto the stack.
- template<typename Arg >
  int push_args (Arg a)

  Bottom case of the push_args recursive call.
- template<typename Arg >
  void push_arg (Arg a)

  Pushes a single value onto the Lua stack.
- template<>
  int **get_** (const std::string &name)
- template<>
  bool **get_** (const std::string &name)
- template<>
  void **get_** (const std::string &name)
- template<>
  void push_arg (int arg)

  Specializations of the method lpp::Script::push_arg, which pushes a single function argument onto the Lua stack.
- template<>
  void **push_arg** (float arg)
- template<>
  void **push_arg** (bool arg)

Private Attributes

- state L

  Lua state representing the Lua virtual machine.
- std::set< std::string > loaded_scripts_

  Containes the names of all scripts loaded during the current runtime.

6.87.1 Detailed Description

Class representing a Lua script, allows to register C++ functions, load variables, call functions, execute strings containing Lua code and other functionalities.

Definition at line 18 of file LppScript.hpp.

6.87.2 Constructor & Destructor Documentation

6.87.2.1 lpp::Script::Script (const Script &) `[delete]`

Copying this script might cause the Lua state get closed when one of the copies gets destroyed and would cause the game to be unable to use it's scripting engine (and thus crashing probably).

6.87.2.2 lpp::Script::∼Script ( ) `[inline]`

Destructor, closes the Lua virtual machine.

Definition at line 34 of file LppScript.hpp.

6.87.2.3 lpp::Script::Script ( ) `[private]`

Constructor, kept private because of the use of the singleton pattern.

lpp::Script definitions:

Definition at line 7 of file LppScript.cpp.

6.87.3 Member Function Documentation

6.87.3.1 template<typename Result , typename... Args> Result lpp::Script::call (const std::string & *fname*, Args... *as*) `[inline]`

Calls a given Lua function.

Parameters

Name	of the function.
Variadic	list of arguments that are passed to the function.

Definition at line 99 of file LppScript.hpp.

6.87.3.2 template<typename Result > Result lpp::Script::call (const std::string & *fname*) `[inline]`

Calls a given Lua function.

Parameters

Name	of the function.

Definition at line 121 of file LppScript.hpp.

6.87.3.3 void lpp::Script::clear_stack ( ) `[private]`

Pops everything off the stack.

Definition at line 66 of file LppScript.cpp.

6.87.3.4 void lpp::Script::execute (const std::string & *command*)

Executes a given string from within Lua.

Parameters

String	containing commands to be executed.

Definition at line 14 of file LppScript.cpp.

6.87.3.5 template<typename T > T lpp::Script::get (const std::string & *name*) `[inline]`

Retrieves and returns a value from Lua.

Parameters

Name	of the variable containing the desired value.

Definition at line 74 of file LppScript.hpp.

6.87.3.6 template<typename T > T lpp::Script::get_ (const std::string & *name* = `"unknown"`) `[private]`

Returns the value stored on top of the stack.

6.87.3.7 std::string lpp::Script::get_field_to_stack (const std::string & *name*) [private]

Gets a nested value (inside a table hierarchy) on top of the stack and returns the name of the final variable (without table prefixes).

Parameters

Full	name of the variable.

Definition at line 47 of file LppScript.cpp.

6.87.3.8 std::string lpp::Script::get_stack_contents ( )

Returns string representation of the Lua stack.

Definition at line 72 of file LppScript.cpp.

6.87.3.9 state lpp::Script::get_state ( ) [inline]

Returns the lua state representing the Lua virtual machine.

Definition at line 39 of file LppScript.hpp.

6.87.3.10 template<typename T > std::vector<T> lpp::Script::get_vector (const std::string & *name*) [inline]

Retrieves a Lua array table (integer indexing) in the form of a C++ vector.

Parameters

Name	of the array.

Definition at line 153 of file LppScript.hpp.

6.87.3.11 lpp::Script & lpp::Script::instance ( ) [static]

Returns a reference to the lpp::Script singleton.

Definition at line 105 of file LppScript.cpp.

6.87.3.12 bool lpp::Script::is_nil (const std::string & *name*)

Returns true if a given value is nil, false otherwise.

Parameters

Name	of the variable containing the desired value.

Definition at line 34 of file LppScript.cpp.

6.87.3.13 void lpp::Script::load (const std::string & *fname*)

Loads, compiles and executes a Lua script.

Parameters

Name	of the script file.

Definition at line 25 of file LppScript.cpp.

6.87.3.14 template<typename Arg > void lpp::Script::push_arg (Arg *a*) [inline], [private]

Pushes a single value onto the Lua stack.

Parameters

Value	to be pushed.

Note

By default does nothing, only specialised versions push anything.

Definition at line 250 of file LppScript.hpp.

6.87.3.15 template<> void lpp::Script::push_arg (int *arg*) [inline], [private]

Specializations of the method lpp::Script::push_arg, which pushes a single function argument onto the Lua stack.

Definition at line 370 of file LppScript.hpp.

6.87.3.16 template<typename Arg , typename... Args> int lpp::Script::push_args (Arg *a,* Args... *as*) [inline], [private]

Pushed a variadic list of arguments onto the stack to be passed as arguments to a Lua function call, returns the amount of arguments pushed onto the stack.

Parameters

First	argument in the list.
Tail	argument list used in recursive call.

Definition at line 226 of file LppScript.hpp.

6.87.3.17 template<typename Arg > int lpp::Script::push_args (Arg *a*) [inline], [private]

Bottom case of the push_args recursive call.

Parameters

Argument	to be pushed onto the stack.

Definition at line 237 of file LppScript.hpp.

6.87.3.18 void lpp::Script::register_function (const std::string & *name,* lua_CFunction *fn*)

Registers a C++ function which can then be used from within Lua.

Parameters

Name	of the function.
Function	to be registered, it has to have the signature int fname(lua_State∗) and return the number of results pushed onto the stack, arguments are on the stack.

Definition at line 20 of file LppScript.cpp.

6.87.3.19 void lpp::Script::reload_all_scripts ( )

Reloads all script files that have been previously loaded.

Definition at line 99 of file LppScript.cpp.

6.87.3.20 template<typename T > void lpp::Script::set (const std::string & *name,* T *val*) [inline]

Sets a given variable to a given value.

Parameters

Variable	to be changed.
Value	that the variable should be changed to.

Definition at line 142 of file LppScript.hpp.

6.87.4 Member Data Documentation

6.87.4.1 state lpp::Script::L [private]

Lua state representing the Lua virtual machine.

Definition at line 250 of file LppScript.hpp.

6.87.4.2 std::set< std::string > lpp::Script::loaded_scripts_ [private]

Containes the names of all scripts loaded during the current runtime.

Definition at line 260 of file LppScript.hpp.

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

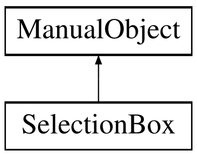
- lppscript/LppScript.hpp
- lppscript/LppScript.cpp

6.88 SelectionBox Class Reference

Class representing the ingame selection box (created by moving the left mouse while pressing the left mouse button) to select multiple entities on screen or a single entity (by simply clicking).

`#include <SelectionBox.hpp>`

Inheritance diagram for SelectionBox:



Public Member Functions

- SelectionBox (const Ogre::String &, EntitySystem &, Ogre::PlaneBoundedVolumeListSceneQuery &, Ogre↩ ::RaySceneQuery &, Ogre::SceneManager &)

  Constructor.
- ∼SelectionBox ()

  Destructor.
- void set_corners (tdt::real, tdt::real, tdt::real, tdt::real)

  Recreates the selection box with the given coordinates.
- void set_corners (const Ogre::Vector2 &, const Ogre::Vector2 &)

  Overload of the SelectionBox::set_corners(float, float, float, float) method used as a simple interface when using vectors.
- std::vector< tdt::uint > & get_selected_entities ()

  Returns (by reference) a vector containing ID's of all currently selected entities.
- void select_object (Ogre::MovableObject &)

  Adds a given object to the vector of selected entities.
- void clear_selected_entities ()

  Deselects all currently selected entities.
- void execute_selection (const Ogre::Vector2 &, Ogre::Camera &, bool=false)

  Performs the selection, to be called when the mouse is released.
- void set_starting_point (const Ogre::Vector2 &)

  Sets the starting point of the current selection, should be called upon the initial mouse button click.
- void set_selecting (bool)

  Sets the selecting mode (and the visibility flag of the selection box).
- bool is_selecting () const

  Returns true if the selection box is currently selecting.
- void extend_to (const Ogre::Vector2 &)

  Extends the selection box to a given coordinate, using the start_ coordinate provided by SelectionBox::set_starting↩ _point as the initial point.

Private Member Functions

- void execute_single_selection (Ogre::Camera &)

  Used within the SelectionBox::execute_selection method when the selection box is two small (i.e.

Private Attributes

- std::vector< tdt::uint > selected_entities_

  Currently selected entities.
- EntitySystem & entities_

  Reference to the game's entity system, used to identify an entity ID from an entity SceneNode.
- Ogre::Vector2 start_

  Vector containing the coordinates of the starting point of the selection.
- Ogre::PlaneBoundedVolumeListSceneQuery & volume_query_

  Scene query used to find all the entities that are within the selection box.
- Ogre::RaySceneQuery & ray_query_

  Scene query used to find an entity when the selection box is too small.
- bool selection_in_progress_

  Determines if the player is currently selecting entities.
- Ogre::SceneManager & scene_mgr_

  Reference to the game's main scene manager.

6.88.1 Detailed Description

Class representing the ingame selection box (created by moving the left mouse while pressing the left mouse button) to select multiple entities on screen or a single entity (by simply clicking).

Definition at line 13 of file SelectionBox.hpp.

6.88.2 Constructor & Destructor Documentation

6.88.2.1 SelectionBox::SelectionBox (const Ogre::String & *name*, EntitySystem & *ents*, Ogre::PlaneBounded↩ VolumeListSceneQuery & *vol_query*, Ogre::RaySceneQuery & *ray_query*, Ogre::SceneManager & *mgr*)

Constructor.

Parameters

Name	of this object (passed to ManualObject constructor - hence the type).
Reference	to a volume query used for multi selection.
Reference	to a ray query used for single selection.
Reference	to the game's main scene manager.

Definition at line 5 of file SelectionBox.cpp.

6.88.2.2 SelectionBox::∼SelectionBox ( )

Destructor.

Note

Even thous the queries are captured by reference, they need to be destroyed manually using the scene manager.

Definition at line 19 of file SelectionBox.cpp.

6.88.3 Member Function Documentation

6.88.3.1 void SelectionBox::clear_selected_entities ( )

Deselects all currently selected entities.

Definition at line 76 of file SelectionBox.cpp.

6.88.3.2 void SelectionBox::execute_selection (const Ogre::Vector2 & *end,* Ogre::Camera & *cam,* bool *append* = `false`)

Performs the selection, to be called when the mouse is released.

Parameters

Coordinate	of the end point of the box (mouse position).
Reference	to the camera the query should be performed from.
True	if the the new selection should be appended to the current selection.

Note

The starting point has to be saved prior to this call (i.e. when the mouse button gets pressed) in order for the selection to work properly.

Definition at line 88 of file SelectionBox.cpp.

6.88.3.3 void SelectionBox::execute_single_selection (Ogre::Camera & *cam*) `[private]`

Used within the SelectionBox::execute_selection method when the selection box is two small (i.e.

a single click) and performs a ray query selecting a single closest entity under the player's cursor.

Parameters

Reference	to the game's main camera.

Definition at line 144 of file SelectionBox.cpp.

6.88.3.4 void SelectionBox::extend_to (const Ogre::Vector2 & *end*)

Extends the selection box to a given coordinate, using the start_ coordinate provided by SelectionBox::set_←
starting_point as the initial point.

Parameters

Coordinate	of the ending point.

Definition at line 139 of file SelectionBox.cpp.

6.88.3.5 std::vector< tdt::uint > & SelectionBox::get_selected_entities ( )

Returns (by reference) a vector containing ID's of all currently selected entities.

Definition at line 57 of file SelectionBox.cpp.

6.88.3.6 bool SelectionBox::is_selecting ( ) const

Returns true if the selection box is currently selecting.

Definition at line 134 of file SelectionBox.cpp.

6.88.3.7 void SelectionBox::select_object (Ogre::MovableObject & *obj*)

Adds a given object to the vector of selected entities.

Parameters

Reference	to the object from which the entity ID will be deduced.

Definition at line 62 of file SelectionBox.cpp.

6.88.3.8 void SelectionBox::set_corners (tdt::real *left,* tdt::real *top,* tdt::real *right,* tdt::real *bott*)

Recreates the selection box with the given coordinates.

Parameters

Left	side coordinate.
Top	side coordinate.
Right	side coordinate.
Bottom	axis coordinate.

Neccessary translation, mouse positions are normalized to belong to the (0,1) range, but the Ogre::ManualObject creation requires them to belong to the (-1, 1).

Definition at line 25 of file SelectionBox.cpp.

6.88.3.9 void SelectionBox::set_corners (const Ogre::Vector2 & *t_l,* const Ogre::Vector2 & *b_r*)

Overload of the SelectionBox::set_corners(float, float, float, float) method used as a simple interface when using vectors.

Parameters

Coordinate	of the top left corner.
Coordinate	of the bottom right corner.

Definition at line 50 of file SelectionBox.cpp.

6.88.3.10 void SelectionBox::set_selecting (bool *sel*)

Sets the selecting mode (and the visibility flag of the selection box).

Parameters

Bool	value representing the new visibility mode.

Definition at line 128 of file SelectionBox.cpp.

6.88.3.11 void SelectionBox::set_starting_point (const Ogre::Vector2 & *start*)

Sets the starting point of the current selection, should be called upon the initial mouse button click.

Parameters

Coordinate	of the starting point.

Definition at line 123 of file SelectionBox.cpp.

6.88.4 Member Data Documentation

6.88.4.1 EntitySystem& SelectionBox::entities_ [private]

Reference to the game's entity system, used to identify an entity ID from an entity SceneNode.

Definition at line 120 of file SelectionBox.hpp.

6.88.4.2 Ogre::RaySceneQuery& SelectionBox::ray_query_ [private]

Scene query used to find an entity when the selection box is too small.

Definition at line 136 of file SelectionBox.hpp.

6.88.4.3 Ogre::SceneManager& SelectionBox::scene_mgr_ [private]

Reference to the game's main scene manager.

Definition at line 146 of file SelectionBox.hpp.

6.88.4.4 std::vector<tdt::uint> SelectionBox::selected_entities_ [private]

Currently selected entities.

Definition at line 114 of file SelectionBox.hpp.

6.88.4.5 bool SelectionBox::selection_in_progress_ [private]

Determines if the player is currently selecting entities.

Definition at line 141 of file SelectionBox.hpp.

6.88.4.6 Ogre::Vector2 SelectionBox::start_ [private]

Vector containing the coordinates of the starting point of the selection.

(Ending point will be specified in the SelectionBox::execute_selection method.)

Definition at line 126 of file SelectionBox.hpp.

6.88.4.7 Ogre::PlaneBoundedVolumeListSceneQuery& SelectionBox::volume_query_ [private]

Scene query used to find all the entities that are within the selection box.

Definition at line 131 of file SelectionBox.hpp.

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

- tools/SelectionBox.hpp
- tools/SelectionBox.cpp

6.89 Spellcaster::SPELL Struct Reference

A structure representing a spell by containing it's type and name.

Public Attributes

- SPELL_TYPE **type_**
- std::string **spell_**

6.89.1 Detailed Description

A structure representing a spell by containing it's type and name.

Definition at line 102 of file Spellcaster.hpp.

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

- tools/Spellcaster.hpp

6.90 Spellcaster Class Reference

A utility class that manages the player's spell casting and is usually called from input handlers and the spell casting window.

```
#include <Spellcaster.hpp>
```

Classes

- struct SPELL

  A structure representing a spell by containing it's type and name.

Public Member Functions

- Spellcaster (EntityPlacer &, SelectionBox &)

  Constructor.
- ∼Spellcaster ()=default

  Destructor.
- void set_spell_type (SPELL_TYPE)

  Sets the type of the currently casted spell.
- SPELL_TYPE get_spell_type () const

  Returns the type of the currently casted spell.
- void set_spell (const std::string &)

  Sets the name of the currently casted spell.
- const std::string & get_spell () const

  Returns the name of the currently casted spell.
- void cast (Ogre::Vector2=Ogre::Vector2{})

  Applies the effect of the currently casted spell.
- SPELL_TYPE get_last_spell_type () const

  Returns the type of the previously casted spell.
- const std::string & get_last_spell () const

  Returns the name of the previously casted spell.
- void set_last_spell_id (tdt::uint)

  Sets the ID of the entity created by the last spell.
- tdt::uint get_last_spell_id () const

  Returns the ID of the entity created by the last spell.
- bool is_casting () const

  Returns true if this spellcaster is currently casting, false otherwise.
- void stop_casting ()

  Immediately stops any cast being performed.

Private Attributes

- EntityPlacer & placer_

  Used to place entities created by placing spell.
- SelectionBox & selector_

  Used to get selected targets for targeted spells.
- lpp::Script & script_

  Reference to the scripting engine used for easier calls to the spell functions.
- SPELL curr_spell_

  The spell that is currently being casted.
- SPELL last_spell_

  The spell that has been casted previously.
- tdt::uint last_spell_id_

  ID of the entity created by a placing spell (if any).

6.90.1 Detailed Description

A utility class that manages the player's spell casting and is usually called from input handlers and the spell casting window.

Definition at line 18 of file Spellcaster.hpp.

6.90.2 Constructor & Destructor Documentation

6.90.2.1 Spellcaster::Spellcaster (EntityPlacer & *placer*, SelectionBox & *selector*)

Constructor.

Parameters

The	placer that is used for placing spells.
Selector	that is used for targeted spells.

Definition at line 8 of file Spellcaster.cpp.

6.90.2.2 Spellcaster::∼Spellcaster ( ) `[default]`

Destructor.

6.90.3 Member Function Documentation

6.90.3.1 void Spellcaster::cast (Ogre::Vector2 *mouse_position* = `Ogre::Vector2{}`)

Applies the effect of the currently casted spell.

Parameters

Optional	mouse position parameter for positional spells.

Definition at line 36 of file Spellcaster.cpp.

6.90.3.2 const std::string & Spellcaster::get_last_spell ( ) const

Returns the name of the previously casted spell.

(Used if spells have sequential effect - like portals.)

Definition at line 83 of file Spellcaster.cpp.

6.90.3.3 tdt::uint Spellcaster::get_last_spell_id ( ) const

Returns the ID of the entity created by the last spell.

(Used if spells have sequential effect - like portals.)

Definition at line 93 of file Spellcaster.cpp.

6.90.3.4 SPELL_TYPE Spellcaster::get_last_spell_type ( ) const

Returns the type of the previously casted spell.

(Used if spells have sequential effect - like portals.)

Definition at line 78 of file Spellcaster.cpp.

6.90.3.5 const std::string & Spellcaster::get_spell ( ) const

Returns the name of the currently casted spell.

Definition at line 31 of file Spellcaster.cpp.

6.90.3.6 SPELL_TYPE Spellcaster::get_spell_type ( ) const

Returns the type of the currently casted spell.

Definition at line 20 of file Spellcaster.cpp.

6.90.3.7 bool Spellcaster::is_casting ( ) const

Returns true if this spellcaster is currently casting, false otherwise.

Definition at line 98 of file Spellcaster.cpp.

6.90.3.8 void Spellcaster::set_last_spell_id (tdt::uint *val*)

Sets the ID of the entity created by the last spell.

(Used if spells have sequential effect - like portals.)

Parameters

The	ID of the entity created.

Definition at line 88 of file Spellcaster.cpp.

6.90.3.9 void Spellcaster::set_spell (const std::string & *val*)

Sets the name of the currently casted spell.

Parameters

The	new name.

Definition at line 25 of file Spellcaster.cpp.

6.90.3.10 void Spellcaster::set_spell_type (SPELL_TYPE *val*)

Sets the type of the currently casted spell.

Parameters

The	new spell type.

Definition at line 14 of file Spellcaster.cpp.

6.90.3.11 void Spellcaster::stop_casting ( )

Immediately stops any cast being performed.

Definition at line 103 of file Spellcaster.cpp.

6.90.4 Member Data Documentation

6.90.4.1 SPELL Spellcaster::curr_spell_ [private]

The spell that is currently being casted.

Definition at line 127 of file Spellcaster.hpp.

6.90.4.2 SPELL Spellcaster::last_spell_ [private]

The spell that has been casted previously.

Definition at line 132 of file Spellcaster.hpp.

6.90.4.3 tdt::uint Spellcaster::last_spell_id_ [private]

ID of the entity created by a placing spell (if any).

Definition at line 137 of file Spellcaster.hpp.

6.90.4.4 EntityPlacer& Spellcaster::placer_ [private]

Used to place entities created by placing spell.

Definition at line 111 of file Spellcaster.hpp.

6.90.4.5 lpp::Script& Spellcaster::script_ [private]

Reference to the scripting engine used for easier calls to the spell functions.

Definition at line 122 of file Spellcaster.hpp.

6.90.4.6 SelectionBox& Spellcaster::selector_ [private]

Used to get selected targets for targeted spells.

Definition at line 116 of file Spellcaster.hpp.

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

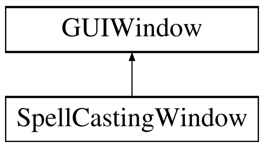
- tools/Spellcaster.hpp
- tools/Spellcaster.cpp

6.91 SpellCastingWindow Class Reference

Class representing the spell selection window, allows the player to cast registered (unlocked) spells.

`#include <SpellCastingWindow.hpp>`

Inheritance diagram for SpellCastingWindow:



Public Member Functions

- SpellCastingWindow ()

  Constructor.
- ∼SpellCastingWindow ()=default

  Destructor.
- void register_spell (const std::string &)

  Appends a table name to the vector of all spell tables.
- void set_caster (Spellcaster ∗)

  Sets the caster instance used to cast the spells.
- void deactivate_current_spell ()

  Hides the "active" label.
- const std::vector< std::string > & get_spells () const

  Returns a vector containing the names of all unlocked spells.
- void clear_spells ()

  Removes all unlocked spell.
- bool dec_selection ()

  Decrements selection_number_ by one and updates the window.
- bool inc_selection ()

  Increments selection_number_ by one and updates the window.
- void set_spell_active (int)

  Marks a given spell as active.
- void cast (int)

  Casts a spell at a given position in the roster.

Protected Member Functions

- void init_ () override

  Initializes the window and subscribes events.

Private Member Functions

- const std::string & get_spell_ (std::size_t)

  Range checked buildings_ index access, returns the name of the building at a given index or "UNKNOWN" if the index is out of bounds.
- void update_selection_ ()

  Updates building names on the buttons.

Private Attributes

- std::vector< std::string > spells_

  Names of all registered buildings.
- std::size_t selection_number_

  Number of the current rightmost selection.
- lpp::Script ∗ script_

  Pointer to the scripting engine used for easier access.
- Spellcaster ∗ caster_

  Does the actual spell casting once a spell is selected in this window.
- int curr_active_spell_

  Keeps track of the spell that is being currently casted.

Additional Inherited Members

6.91.1 Detailed Description

Class representing the spell selection window, allows the player to cast registered (unlocked) spells.

Definition at line 16 of file SpellCastingWindow.hpp.

6.91.2 Constructor & Destructor Documentation

6.91.2.1 SpellCastingWindow::SpellCastingWindow ( )

Constructor.

Definition at line 7 of file SpellCastingWindow.cpp.

6.91.2.2 SpellCastingWindow::∼SpellCastingWindow ( ) `[default]`

Destructor.

6.91.3 Member Function Documentation

6.91.3.1 void SpellCastingWindow::cast (int *spell_num*)

Casts a spell at a given position in the roster.

Parameters

The	position of the spell (1-4).

Definition at line 49 of file SpellCastingWindow.cpp.

6.91.3.2 void SpellCastingWindow::clear_spells ( )

Removes all unlocked spell.

Definition at line 38 of file SpellCastingWindow.cpp.

6.91.3.3 void SpellCastingWindow::deactivate_current_spell ( )

Hides the "active" label.

Definition at line 27 of file SpellCastingWindow.cpp.

6.91.3.4 bool SpellCastingWindow::dec_selection ( )

Decrements selection_number_ by one and updates the window.

Definition at line 115 of file SpellCastingWindow.cpp.

6.91.3.5 const std::string & SpellCastingWindow::get_spell_ (std::size_t *index*) [private]

Range checked buildings_ index access, returns the name of the building at a given index or "UNKNOWN" if the index is out of bounds.

Parameters

Index	of the building in the buildings_ vector.

Definition at line 142 of file SpellCastingWindow.cpp.

6.91.3.6 const std::vector< std::string > & SpellCastingWindow::get_spells ( ) const

Returns a vector containing the names of all unlocked spells.

(Used for serialization.)

Definition at line 33 of file SpellCastingWindow.cpp.

6.91.3.7 bool SpellCastingWindow::inc_selection ( )

Increments selection_number_ by one and updates the window.

Definition at line 129 of file SpellCastingWindow.cpp.

6.91.3.8 void SpellCastingWindow::init_ ( ) [override], [protected], [virtual]

Initializes the window and subscribes events.

Implements GUIWindow.

Definition at line 61 of file SpellCastingWindow.cpp.

6.91.3.9 void SpellCastingWindow::register_spell (const std::string & *name*)

Appends a table name to the vector of all spell tables.

Parameters

Name	of the table to register.

Definition at line 11 of file SpellCastingWindow.cpp.

6.91.3.10 void SpellCastingWindow::set_caster (Spellcaster ∗ *caster*)

Sets the caster instance used to cast the spells.

Parameters

The	new spell caster.

Definition at line 22 of file SpellCastingWindow.cpp.

6.91.3.11 void SpellCastingWindow::set_spell_active (int *val*)

Marks a given spell as active.

Parameters

Position	of the spell in the current view (0-3).

Definition at line 182 of file SpellCastingWindow.cpp.

6.91.3.12 void SpellCastingWindow::update_selection_ ( ) [private]

Updates building names on the buttons.

Definition at line 152 of file SpellCastingWindow.cpp.

6.91.4 Member Data Documentation

6.91.4.1 Spellcaster∗ SpellCastingWindow::caster_ [private]

Does the actual spell casting once a spell is selected in this window.

Definition at line 124 of file SpellCastingWindow.hpp.

6.91.4.2 int SpellCastingWindow::curr_active_spell_ [private]

Keeps track of the spell that is being currently casted.

Definition at line 129 of file SpellCastingWindow.hpp.

6.91.4.3 lpp::Script∗ SpellCastingWindow::script_ [private]

Pointer to the scripting engine used for easier access.

Definition at line 118 of file SpellCastingWindow.hpp.

6.91.4.4 std::size_t SpellCastingWindow::selection_number_ [private]

Number of the current rightmost selection.

The window shows buildings with indices <selection_number_ - 3, selection_number_>.

Definition at line 112 of file SpellCastingWindow.hpp.

6.91.4.5 std::vector<std::string> SpellCastingWindow::spells_ [private]

Names of all registered buildings.

Definition at line 106 of file SpellCastingWindow.hpp.

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

- gui/SpellCastingWindow.hpp
- gui/SpellCastingWindow.cpp

6.92 SpellComponent Struct Reference

Allows an entity to periodically cast a spell.

```
#include <Components.hpp>
```

Public Member Functions

- **SpellComponent** (std::string &&b="ERROR", tdt::real cd=0.f)
- **SpellComponent** (const SpellComponent &)=default
- **SpellComponent** (SpellComponent &&)=default
- SpellComponent & **operator=** (const SpellComponent &)=default
- SpellComponent & **operator=** (SpellComponent &&)=default

Public Attributes

- std::string **blueprint**
- tdt::real **cd_time**
- tdt::real **cooldown**

Static Public Attributes

- static constexpr int **type** = 10

6.92.1 Detailed Description

Allows an entity to periodically cast a spell.

Definition at line 272 of file Components.hpp.

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

- Components.hpp

6.93 StructureComponent Struct Reference

Defines a building (or a wall), by holding it's radius (of the area it takes in the grid) and vector of nodes that it sits on.

```
#include <Components.hpp>
```

Public Member Functions

- **StructureComponent** (tdt::uint r=1, bool wt=false)
- **StructureComponent** (const StructureComponent &)=default
- **StructureComponent** (StructureComponent &&)=default
- StructureComponent & **operator=** (const StructureComponent &)=default
- StructureComponent & **operator=** (StructureComponent &&)=default

Public Attributes

- tdt::uint **radius**
- bool **walk_through**
- std::vector< tdt::uint > **residences**

Static Public Attributes

- static constexpr int **type** = 17

6.93.1 Detailed Description

Defines a building (or a wall), by holding it's radius (of the area it takes in the grid) and vector of nodes that it sits on.

Definition at line 440 of file Components.hpp.

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

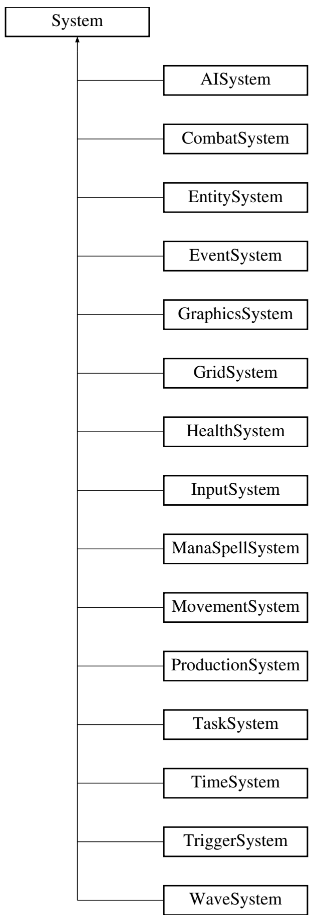
- Components.hpp

6.94 System Class Reference

Parent class of all systems.

```
#include <System.hpp>
```

Inheritance diagram for System:



Public Member Functions

- virtual void update (tdt::real)=0

  Updates the system.
- virtual ∼System ()=default

  Destructor.

6.94.1 Detailed Description

Parent class of all systems.

Definition at line 8 of file System.hpp.

6.94.2 Constructor & Destructor Documentation

6.94.2.1 virtual System::∼System ( ) [virtual], [default]

Destructor.

6.94.3 Member Function Documentation

6.94.3.1 virtual void System::update (tdt::real) [pure virtual]

Updates the system.

Parameters

Time	since the last frame.

Implemented in CombatSystem, EntitySystem, WaveSystem, InputSystem, TaskSystem, ProductionSystem, Grid↩ System, TriggerSystem, HealthSystem, ManaSpellSystem, AISystem, GraphicsSystem, MovementSystem, Time↩ System, and EventSystem.

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

- systems/System.hpp

6.95 TaskComponent Struct Reference

Defines a task by giving it a type, source (the task handler) and a target (subject of the task).

```
#include <Components.hpp>
```

Public Member Functions

- **TaskComponent** (tdt::uint target_id=Component::NO_ENTITY, tdt::uint source_id=Component::NO_ENTI↩ TY, TASK_TYPE t_type=TASK_TYPE::NONE)
- **TaskComponent** (const TaskComponent &)=default
- **TaskComponent** (TaskComponent &&)=default
- TaskComponent & **operator=** (const TaskComponent &)=default
- TaskComponent & **operator=** (TaskComponent &&)=default

Public Attributes

- TASK_TYPE **task_type**
- tdt::uint **source**
- tdt::uint **target**
- bool **complete**

Static Public Attributes

- static constexpr int **type** = 15

6.95.1 Detailed Description

Defines a task by giving it a type, source (the task handler) and a target (subject of the task).

Handling of these tasks is done via the TaskHandlerComponent below.

Definition at line 390 of file Components.hpp.

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

- Components.hpp

6.96 TaskHandlerComponent Struct Reference

Task queue and register of possible tasks, every entity that is able to actually do something on it's own should have it.

```
#include <Components.hpp>
```

Public Member Functions

- **TaskHandlerComponent** (std::string &&b="ERROR")
- **TaskHandlerComponent** (const TaskHandlerComponent &)=default
- **TaskHandlerComponent** (TaskHandlerComponent &&)=default
- TaskHandlerComponent & **operator=** (const TaskHandlerComponent &)=default
- TaskHandlerComponent & **operator=** (TaskHandlerComponent &&)=default

Public Attributes

- tdt::uint **curr_task**
- std::bitset< (int) TASK_TYPE::COUNT > **possible_tasks**
- std::deque< tdt::uint > **task_queue**
- bool **busy**
- std::string **blueprint**

Static Public Attributes

- static constexpr int **type** = 16

6.96.1 Detailed Description

Task queue and register of possible tasks, every entity that is able to actually do something on it's own should have it.

Definition at line 415 of file Components.hpp.

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

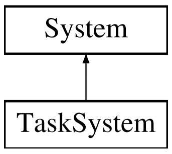
- Components.hpp

6.97 TaskSystem Class Reference

System managing all entities with the TaskComponent, their creation, assignment, lifetime checks and canceling.

```
#include <TaskSystem.hpp>
```

Inheritance diagram for TaskSystem:



Public Member Functions

- TaskSystem (EntitySystem &, GridSystem &, CombatSystem &)

  Constructor.
- ∼TaskSystem ()=default

  Destructor.
- void update (tdt::real) override

  Manages the lifetime of tasks on each frame.
- const std::string & get_task_name (TASK_TYPE) const

  Translates a task type enum value into a string that can be displayed in the developer's console.

Private Member Functions

- void next_task_ (TaskHandlerComponent &)

  Set's the current task to the task in the front of the respective task queue and removes it from the queue.
- bool handle_task_ (tdt::uint, TaskHandlerComponent &)

  Executes a single task.
- bool current_task_completed_ (tdt::uint, TaskHandlerComponent &)

  Checks whether the current task of a given entity has been completed.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- GridSystem & grid_

  Reference to the game's grid system.
- CombatSystem & combat_

  Reference to the game's combat system used for line of sight checking.
- std::map< TASK_TYPE, std::string > task_names_

  Map used for task type translation.

6.97.1 Detailed Description

System managing all entities with the TaskComponent, their creation, assignment, lifetime checks and canceling.

Definition at line 18 of file TaskSystem.hpp.

6.97.2 Constructor & Destructor Documentation

6.97.2.1 TaskSystem::TaskSystem (**EntitySystem** & *ents*, **GridSystem** & *grid*, **CombatSystem** & *comb*)

Constructor.

Parameters

Reference	to the game's entity system.
Reference	to the game's grid system.

Definition at line 7 of file TaskSystem.cpp.

6.97.2.2 TaskSystem::~TaskSystem ( ) [default]

Destructor.

6.97.3 Member Function Documentation

6.97.3.1 bool TaskSystem::current_task_completed_ (tdt::uint *id*, **TaskHandlerComponent** & *handler*) [private]

Checks whether the current task of a given entity has been completed.

Parameters

ID	of the handling entity.
Reference	to the entity's TaskHandlerComponent.

Definition at line 70 of file TaskSystem.cpp.

6.97.3.2 const std::string & TaskSystem::get_task_name (TASK_TYPE *type*) const

Translates a task type enum value into a string that can be displayed in the developer's console.

Parameters

Task	type to be translated.

Definition at line 45 of file TaskSystem.cpp.

6.97.3.3 bool TaskSystem::handle_task_ (tdt::uint *id,* TaskHandlerComponent & *handler*) [private]

Executes a single task.

Parameters

ID	of the entity that the task is assigned to.
Reference	to the task handling component of the assigned entity. (For easier look up of the blueprint.)

Definition at line 63 of file TaskSystem.cpp.

6.97.3.4 void TaskSystem::next_task_ (TaskHandlerComponent & *comp*) [private]

Set's the current task to the task in the front of the respective task queue and removes it from the queue.

Parameters

Reference	to the TaskHandlerComponent containing the aforementioned task queue.

Definition at line 54 of file TaskSystem.cpp.

6.97.3.5 void TaskSystem::update (tdt::real *delta*) [override], [virtual]

Manages the lifetime of tasks on each frame.

Parameters

Time	since the last frame.

Implements System.

Definition at line 16 of file TaskSystem.cpp.

6.97.4 Member Data Documentation

6.97.4.1 CombatSystem& TaskSystem::combat_ `[private]`

Reference to the game's combat system used for line of sight checking.

Definition at line 82 of file TaskSystem.hpp.

6.97.4.2 EntitySystem& TaskSystem::entities_ `[private]`

Reference to the game's entity system.

Definition at line 72 of file TaskSystem.hpp.

6.97.4.3 GridSystem& TaskSystem::grid_ `[private]`

Reference to the game's grid system.

Definition at line 77 of file TaskSystem.hpp.

6.97.4.4 std::map<TASK_TYPE, std::string> TaskSystem::task_names_ `[private]`

Map used for task type translation.

Definition at line 87 of file TaskSystem.hpp.

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

- systems/TaskSystem.hpp
- systems/TaskSystem.cpp

6.98 TimeComponent Struct Reference

Represents a timer that after a certain amount of time can start end an event (it's target).

```
#include <Components.hpp>
```

Public Member Functions

- **TimeComponent** (TIME_EVENT ev=TIME_EVENT::NONE, tdt::real limit=0.f, tdt::uint t=Component::NO_↩ ENTITY)
- **TimeComponent** (const TimeComponent &)=default
- **TimeComponent** (TimeComponent &&)=default
- TimeComponent & **operator=** (const TimeComponent &)=default
- TimeComponent & **operator=** (TimeComponent &&)=default

Public Attributes

- tdt::real **curr_time**
- tdt::real **time_limit**
- tdt::uint **target**
- TIME_EVENT **event_type**

Static Public Attributes

- static constexpr int **type** = 8

6.98.1 Detailed Description

Represents a timer that after a certain amount of time can start end an event (it's target).

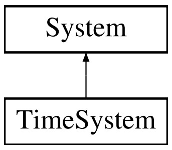
Definition at line 228 of file Components.hpp.

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

- Components.hpp

6.99 TimeSystem Class Reference

Inheritance diagram for TimeSystem:



Public Member Functions

- TimeSystem (EntitySystem &)

  Constructor.
- ∼TimeSystem ()=default

  Destructor.
- void update (tdt::real) override

  Updates the time passed for all TimeComponents and handles those that surpassed their target time.
- void advance_all_timers (tdt::real)

  Adds a given time value to all TimeComponents.
- void advance_all_timers_of_type (tdt::real, TIME_EVENT)

  Adds a given time value to all TimeComponents that match the given time event type.
- void set_time_multiplier (tdt::real=1.f)

  Sets the time value by which are all frame times multiplied when added to timers (this allows to slow/speed up all timers).
- tdt::real get_time_multiplier ()

  Returns the time value by which are all frame times multiplied when added to timers.

Private Member Functions

- void handle_event_ (tdt::uint, TimeComponent &)

  Handles a time event when it's timer finnishes.

Private Attributes

- EntitySystem & entities_

  Reference to the game's entity system.
- tdt::real time_multiplier_

  Allows to speed up all timers.

6.99.1 Detailed Description

Definition at line 9 of file TimeSystem.hpp.

6.99.2 Constructor & Destructor Documentation

6.99.2.1 TimeSystem::TimeSystem (EntitySystem & *ents*)

Constructor.

Parameters

Reference	to the game's entity system.

Definition at line 7 of file TimeSystem.cpp.

6.99.2.2 TimeSystem::∼TimeSystem ( ) [default]

Destructor.

6.99.3 Member Function Documentation

6.99.3.1 void TimeSystem::advance_all_timers (tdt::real *delta*)

Adds a given time value to all TimeComponents.

Parameters

Time	to add.

Definition at line 48 of file TimeSystem.cpp.

6.99.3.2 void TimeSystem::advance_all_timers_of_type (tdt::real *delta,* TIME_EVENT *type*)

Adds a given time value to all TimeComponents that match the given time event type.

Parameters

Time	to add.
Time	even type to match.

Definition at line 54 of file TimeSystem.cpp.

6.99.3.3 tdt::real TimeSystem::get_time_multiplier ( )

Returns the time value by which are all frame times multiplied when added to timers.

Definition at line 68 of file TimeSystem.cpp.

6.99.3.4 void TimeSystem::handle_event_ (tdt::uint *id,* TimeComponent & *comp*) [private]

Handles a time event when it's timer finnishes.

Parameters

ID	of the time event.
Reference	to the TimeComponent of this time event.

Definition at line 73 of file TimeSystem.cpp.

6.99.3.5 void TimeSystem::set_time_multiplier (tdt::real *val* = 1.f)

Sets the time value by which are all frame times multiplied when added to timers (this allows to slow/speed up all timers).

Parameters

The	new time multiplier.

Definition at line 63 of file TimeSystem.cpp.

6.99.3.6 void TimeSystem::update (tdt::real *delta*) [override], [virtual]

Updates the time passed for all TimeComponents and handles those that surpassed their target time.

Parameters

Time	since last frame.

Implements System.

Definition at line 11 of file TimeSystem.cpp.

6.99.4 Member Data Documentation

6.99.4.1 EntitySystem& TimeSystem::entities_ [private]

Reference to the game's entity system.

Definition at line 68 of file TimeSystem.hpp.

6.99.4.2 tdt::real TimeSystem::time_multiplier_ [private]

Allows to speed up all timers.

Definition at line 73 of file TimeSystem.hpp.

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

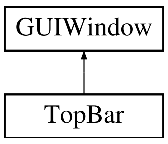
- systems/TimeSystem.hpp
- systems/TimeSystem.cpp

6.100 TopBar Class Reference

Class representing an info bar on the top of the screen displaying the name of the game, player's gold, mana, units and the current time.

`#include <TopBar.hpp>`

Inheritance diagram for TopBar:



Public Member Functions

- TopBar ()
  Constructor.
- ∼TopBar ()=default
  Destructor.
- void update_time (tdt::real)
  Updates the current time on the top bar if a second passed since the last time update.
- void update_label (const std::string &, const std::string &)
  Sets the given label's text to the given string.

Protected Member Functions

- void init_ () override
  Initializes the top bar.

Private Attributes

- tdt::real tdelta_
  Time since the last "Current Time" update.

Additional Inherited Members

6.100.1 Detailed Description

Class representing an info bar on the top of the screen displaying the name of the game, player's gold, mana, units and the current time.

Definition at line 12 of file TopBar.hpp.

6.100.2 Constructor & Destructor Documentation

6.100.2.1 TopBar::TopBar ( )

Constructor.

Definition at line 5 of file TopBar.cpp.

6.100.2.2 TopBar::∼TopBar ( ) [default]

Destructor.

6.100.3 Member Function Documentation

6.100.3.1 void TopBar::init_ ( ) [override], [protected], [virtual]

Initializes the top bar.

Implements GUIWindow.

Definition at line 28 of file TopBar.cpp.

6.100.3.2 void TopBar::update_label (const std::string & *label*, const std::string & *val*)

Sets the given label's text to the given string.

Parameters

Label	to change.
New	text.

Definition at line 23 of file TopBar.cpp.

6.100.3.3 void TopBar::update_time (tdt::real *delta*)

Updates the current time on the top bar if a second passed since the last time update.

Parameters

Time	since the last frame.

Definition at line 9 of file TopBar.cpp.

6.100.4 Member Data Documentation

6.100.4.1 tdt::real TopBar::tdelta_ `[private]`

Time since the last "Current Time" update.

Definition at line 49 of file TopBar.hpp.

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

- gui/TopBar.hpp
- gui/TopBar.cpp

6.101 TriggerComponent Struct Reference

Allows an entity to cause an effect (by calling it's blueprint) when its triggered (stepped on) or can notify a linked entity which causes the effect.

`#include <Components.hpp>`

Public Member Functions

- **TriggerComponent** (std::string &&b="ERROR", tdt::real cd=0.f, tdt::real rad=0.f)
- **TriggerComponent** (const TriggerComponent &)=default
- **TriggerComponent** (TriggerComponent &&)=default
- TriggerComponent & **operator=** (const TriggerComponent &)=default
- TriggerComponent & **operator=** (TriggerComponent &&)=default

Public Attributes

- std::string **blueprint**
- tdt::uint **linked_entity**
- tdt::real **curr_time**
- tdt::real **cooldown**
- tdt::real **radius**

Static Public Attributes

- static constexpr int **type** = 29

6.101.1 Detailed Description

Allows an entity to cause an effect (by calling it's blueprint) when its triggered (stepped on) or can notify a linked entity which causes the effect.

Definition at line 689 of file Components.hpp.

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

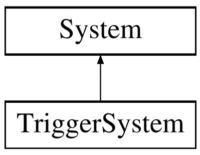
- Components.hpp

6.102 TriggerSystem Class Reference

Handles triggers by checking if an entity is standing in their radius when they are off cooldowns.

```
#include <TriggerSystem.hpp>
```

Inheritance diagram for TriggerSystem:



Public Member Functions

- TriggerSystem (EntitySystem &)

  Constructor.
- ∼TriggerSystem ()=default

  Destructor.
- void update (tdt::real) override

  Checks if any entities have been triggered and performs their associated actions if they were.
- void set_check_period (tdt::real)

  Sets the time period between trigger checks.
- tdt::real get_check_period () const

  Returns the time period between trigger checks.

Private Attributes

- EntitySystem & entities_

  Entity system containing the entities this system works with.
- tdt::real check_timer_

  Allow for dynamic time periods between trigger checks.
- tdt::real **check_period_**

6.102.1 Detailed Description

Handles triggers by checking if an entity is standing in their radius when they are off cooldowns.

Definition at line 11 of file TriggerSystem.hpp.

6.102.2 Constructor & Destructor Documentation

6.102.2.1 TriggerSystem::TriggerSystem (EntitySystem & *ents*)

Constructor.

Parameters

Entity	system containing entities this system works with.

Definition at line 6 of file TriggerSystem.cpp.

6.102.2.2 TriggerSystem::∼TriggerSystem ( ) `[default]`

Destructor.

6.102.3 Member Function Documentation

6.102.3.1 tdt::real TriggerSystem::get_check_period ( ) const

Returns the time period between trigger checks.

Definition at line 65 of file TriggerSystem.cpp.

6.102.3.2 void TriggerSystem::set_check_period (tdt::real *val*)

Sets the time period between trigger checks.

Parameters

The	new time period.

Definition at line 60 of file TriggerSystem.cpp.

6.102.3.3 void TriggerSystem::update (tdt::real *delta*) [override], [virtual]

Checks if any entities have been triggered and performs their associated actions if they were.

Parameters

Time	since the last frame.

Implements System.

Definition at line 10 of file TriggerSystem.cpp.

6.102.4 Member Data Documentation

6.102.4.1 tdt::real TriggerSystem::check_timer_ [private]

Allow for dynamic time periods between trigger checks.

Definition at line 53 of file TriggerSystem.hpp.

6.102.4.2 EntitySystem& TriggerSystem::entities_ [private]

Entity system containing the entities this system works with.

Definition at line 48 of file TriggerSystem.hpp.

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

- systems/TriggerSystem.hpp
- systems/TriggerSystem.cpp

6.103 UpgradeComponent Struct Reference

Represents the game's leveling system component, contains info about experience and leveling progression as well as the blueprint that gets called on level up.

```
#include <Components.hpp>
```

Public Member Functions

- **UpgradeComponent** (std::string &&b="ERROR", tdt::uint exp=100, tdt::uint cap=5)
- **UpgradeComponent** (const UpgradeComponent &)=default
- **UpgradeComponent** (UpgradeComponent &&)=default
- UpgradeComponent & **operator=** (const UpgradeComponent &)=default
- UpgradeComponent & **operator=** (UpgradeComponent &&)=default

Public Attributes

- std::string **blueprint**
- tdt::uint **experience**
- tdt::uint **exp_needed**
- tdt::uint **level**
- tdt::uint **level_cap**

Static Public Attributes

- static constexpr int **type** = 30

6.103.1 Detailed Description

Represents the game's leveling system component, contains info about experience and leveling progression as well as the blueprint that gets called on level up.

Definition at line 714 of file Components.hpp.

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

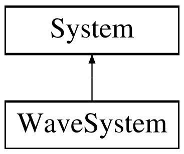
- Components.hpp

6.104 WaveSystem Class Reference

This system creates the entities attacking the player's dungeon in a similar fashion to tower defense games.

```
#include <WaveSystem.hpp>
```

Inheritance diagram for WaveSystem:



Public Member Functions

- WaveSystem (EntitySystem &)

    Constructor.
- ∼WaveSystem ()=default

    Destructor.
- void update (tdt::real) override

    Updates all necessary timers and checks if a wave has arrived and if so, spawns it.
- void set_countdown_window (CEGUI::Window ∗)

    Saves pointer to the window that is used to show the countdown to the next wave.
- void next_wave ()

    Finnishes the wave countdown, causing a wave to be spawned on the next update call.
- void advance_wave_countdown (tdt::uint)

    Advances the countdown by a given amount of time.
- void wave_entity_died ()

    Registers a death of an entity that belongs to the current wave (called by it's destructor).
- void start ()

    Starts (and initializes) the wave system.
- void pause (bool)

    Changes the state of the system.
- void set_entity_total (tdt::uint)

    Sets the total number of entities the wave is gonna have.
- tdt::uint get_entity_total () const

    Returns the total number of entities the wave is gonna have.
- void set_wave_count (tdt::uint)

    Sets the number of waves before the system stops.
- tdt::uint get_wave_count () const

    Returns the total number of waves the system has.
- void add_spawn_node (tdt::uint)

    Adds a grid node to the spawn list.
- void clear_spawn_nodes ()

    Removes any registered spawn nodes.
- void set_spawn_cooldown (tdt::real)

    Sets the delay between spawns (if # of spawn nodes is smaller than # of wave entities).
- tdt::real get_spawn_cooldown () const

    Returns the delay between spawns.
- void update_label_text ()

    Updates the countdown window's text.
- void add_entity_blueprint (const std::string &)

    Adds a blueprint table for a new wave entity.
- const std::vector< std::string > & get_entity_blueprints () const

    Returns a vector of all entities in the wave.
- void set_wave_table (const std::string &)

    Sets the table this system is using to create waves.
- const std::string & get_wave_table () const

    Returns the table this system is using to create waves.
- const std::vector< tdt::uint > & get_spawning_nodes () const

    Returns a vector of nodes that are marked as spawners for the wave.
- void set_curr_wave_number (tdt::uint)

    Sets the number of the current wave.
- tdt::uint get_curr_wave_number () const

Returns the number of the current wave.

- void set_countdown_value (tdt::uint)

  Sets the time remaining before the next wave.
- tdt::uint get_countdown_value () const

  Returns the time remaining before the next wave.
- void set_state (WAVE_STATE)

  Changes the state of the wave system.
- WAVE_STATE get_state () const

  Returns the state of the wave system.
- void set_spawn_timer (tdt::real)

  Sets the time on the spawn timer, which causes the next spawn batch to appear when it reaches the spawn cooldown value.
- tdt::real get_spawn_timer () const

  Returns the time on the spawn timer, which causes the next spawn batch to appear when it reaches the spawn cooldown value.
- void set_wave_entities (tdt::uint)

  Sets the number of entities the current wave has alive.
- tdt::uint get_wave_entities () const

  Returns the number of entities the current wave has alive.
- void set_entities_spawned (tdt::uint)

  Sets the number of entities already spawned in this wave.
- tdt::uint get_entities_spawned () const

  Returns the number of entities already spawned in this wave.
- void clear_entity_blueprints ()

  Clears any entity blueprints that were gonna be used in the next wave.
- void set_endless_mode (bool)

  Sets the endless flag (true -> last wave repeats).
- bool get_endless_mode () const

  Returns the endless flag (true -> last wave repeats).

Private Member Functions

- void start_wave ()

  Starts the current wave.
- void end_wave ()

  Ends the current wave.
- void spawn ()

  Spawns the next batch of entities.

Private Attributes

- WAVE_STATE state_

  The current state of the system.
- EntitySystem & entities_

  Entity system in which the wave entities will be created.
- tdt::uint curr_wave_number_

  Number of the current wave.
- tdt::uint wave_count_

  Total number of waves.
- tdt::uint wave_entities_

Number of entities in the current wave that are spawned and still alive.
- tdt::uint entities_spawned_

  Number of entities that were already spawned during this wave.
- tdt::uint entities_total_

  Total number of entities in the current wave.
- tdt::uint next_wave_countdown_

  Time in seconds till the next wave starts.
- tdt::real second_timer_

  Auxiliary timer that is used to measure seconds.
- std::string label_text_

  Text that is displayed in the countdown window.
- CEGUI::Window ∗ window_

  Pointer to the countdown window.
- lpp::Script & script_

  Reference to the script engine for easier use.
- std::vector< tdt::uint > spawning_nodes_

  Nodes that are used for entity spawning.
- tdt::real spawn_timer_

  Timers used for entity spawning.
- tdt::real **spawn_cooldown_**
- std::string wave_table_

  Name of the table containing init, wstart and wend functions that define a wave system.
- std::vector< std::string > entity_blueprints_

  Entities that are gonna be spawned.
- bool endless_mode_

  If true, the current wave keeps repeating.

6.104.1 Detailed Description

This system creates the entities attacking the player's dungeon in a similar fashion to tower defense games.

Note

This systems contains a large number of setters/getters (while most of other systems do not). This is because this system has to be fully serialized when saving the game to save the game's progress.

Definition at line 25 of file WaveSystem.hpp.

6.104.2 Constructor & Destructor Documentation

6.104.2.1 WaveSystem::WaveSystem (EntitySystem & *ents*)

Constructor.

Parameters

EntitySystem	in which wave entities will be created.

Definition at line 7 of file WaveSystem.cpp.

6.104.2.2 WaveSystem::∼WaveSystem ( ) [default]

Destructor.

6.104.3 Member Function Documentation

6.104.3.1 void WaveSystem::add_entity_blueprint (const std::string & *val*)

Adds a blueprint table for a new wave entity.

Parameters

Name	of the blueprint table.

Definition at line 169 of file WaveSystem.cpp.

6.104.3.2 void WaveSystem::add_spawn_node (tdt::uint *id*)

Adds a grid node to the spawn list.

Parameters

ID	of the node.

Definition at line 117 of file WaveSystem.cpp.

6.104.3.3 void WaveSystem::advance_wave_countdown (tdt::uint *val*)

Advances the countdown by a given amount of time.

Parameters

Number	of seconds to subtract from the countdown.

Definition at line 64 of file WaveSystem.cpp.

6.104.3.4 void WaveSystem::clear_entity_blueprints ( )

Clears any entity blueprints that were gonna be used in the next wave.

Definition at line 254 of file WaveSystem.cpp.

6.104.3.5 void WaveSystem::clear_spawn_nodes ( )

Removes any registered spawn nodes.

Definition at line 122 of file WaveSystem.cpp.

6.104.3.6 void WaveSystem::end_wave ( ) [private]

Ends the current wave.

Definition at line 280 of file WaveSystem.cpp.

6.104.3.7 tdt::uint WaveSystem::get_countdown_value ( ) const

Returns the time remaining before the next wave.

Definition at line 209 of file WaveSystem.cpp.

6.104.3.8 tdt::uint WaveSystem::get_curr_wave_number ( ) const

Returns the number of the current wave.

Definition at line 199 of file WaveSystem.cpp.

6.104.3.9 bool WaveSystem::get_endless_mode ( ) const

Returns the endless flag (true -> last wave repeats).

Definition at line 264 of file WaveSystem.cpp.

6.104.3.10 tdt::uint WaveSystem::get_entities_spawned ( ) const

Returns the number of entities already spawned in this wave.

Definition at line 249 of file WaveSystem.cpp.

6.104.3.11 const std::vector< std::string > & WaveSystem::get_entity_blueprints ( ) const

Returns a vector of all entities in the wave.

Definition at line 174 of file WaveSystem.cpp.

6.104.3.12 tdt::uint WaveSystem::get_entity_total ( ) const

Returns the total number of entities the wave is gonna have.

Definition at line 102 of file WaveSystem.cpp.

6.104.3.13 tdt::real WaveSystem::get_spawn_cooldown ( ) const

Returns the delay between spawns.

Definition at line 132 of file WaveSystem.cpp.

6.104.3.14 tdt::real WaveSystem::get_spawn_timer ( ) const

Returns the time on the spawn timer, which causes the next spawn batch to appear when it reaches the spawn cooldown value.

Definition at line 229 of file WaveSystem.cpp.

6.104.3.15 const std::vector< tdt::uint > & WaveSystem::get_spawning_nodes ( ) const

Returns a vector of nodes that are marked as spawners for the wave.

Definition at line 189 of file WaveSystem.cpp.

6.104.3.16 WAVE_STATE WaveSystem::get_state ( ) const

Returns the state of the wave system.

Definition at line 219 of file WaveSystem.cpp.

6.104.3.17 tdt::uint WaveSystem::get_wave_count ( ) const

Returns the total number of waves the system has.

Definition at line 112 of file WaveSystem.cpp.

6.104.3.18 tdt::uint WaveSystem::get_wave_entities ( ) const

Returns the number of entities the current wave has alive.

Definition at line 239 of file WaveSystem.cpp.

6.104.3.19 const std::string & WaveSystem::get_wave_table ( ) const

Returns the table this system is using to create waves.

Definition at line 184 of file WaveSystem.cpp.

6.104.3.20 void WaveSystem::next_wave ( )

Finnishes the wave countdown, causing a wave to be spawned on the next update call.

Definition at line 59 of file WaveSystem.cpp.

6.104.3.21 void WaveSystem::pause (bool *val*)

Changes the state of the system.

Parameters

If	true, the wave countdown gets paused, if false, the wave countdown gets resumed.

Definition at line 89 of file WaveSystem.cpp.

6.104.3.22 void WaveSystem::set_countdown_value (tdt::uint *val*)

Sets the time remaining before the next wave.

Parameters

The	new countdown time value.

Definition at line 204 of file WaveSystem.cpp.

6.104.3.23 void WaveSystem::set_countdown_window (CEGUI::Window ∗ *win*)

Saves pointer to the window that is used to show the countdown to the next wave.

Parameters

The	new window.

Definition at line 52 of file WaveSystem.cpp.

6.104.3.24 void WaveSystem::set_curr_wave_number (tdt::uint *val*)

Sets the number of the current wave.

Parameters

The	new wave number.

Definition at line 194 of file WaveSystem.cpp.

6.104.3.25 void WaveSystem::set_endless_mode (bool *val*)

Sets the endless flag (true -> last wave repeats).

Parameters

If	true, endless mode is turned on, otherwise it's turned off.

Definition at line 259 of file WaveSystem.cpp.

6.104.3.26 void WaveSystem::set_entities_spawned (tdt::uint *val*)

Sets the number of entities already spawned in this wave.

Parameters

The	new entity count.

Definition at line 244 of file WaveSystem.cpp.

6.104.3.27 void WaveSystem::set_entity_total (tdt::uint *val*)

Sets the total number of entities the wave is gonna have.

Parameters

The	new number of entities.

Definition at line 97 of file WaveSystem.cpp.

6.104.3.28 void WaveSystem::set_spawn_cooldown (tdt::real *val*)

Sets the delay between spawns (if # of spawn nodes is smaller than # of wave entities).

Parameters

The	new spawn cooldown.

Definition at line 127 of file WaveSystem.cpp.

6.104.3.29 void WaveSystem::set_spawn_timer (tdt::real *val*)

Sets the time on the spawn timer, which causes the next spawn batch to appear when it reaches the spawn cooldown value.

Parameters

The	new spawn timer value.

Definition at line 224 of file WaveSystem.cpp.

6.104.3.30 void WaveSystem::set_state (WAVE_STATE *val*)

Changes the state of the wave system.

Parameters

The	new state.

Definition at line 214 of file WaveSystem.cpp.

6.104.3.31 void WaveSystem::set_wave_count (tdt::uint *val*)

Sets the number of waves before the system stops.

Parameters

The	new number of waves.

Definition at line 107 of file WaveSystem.cpp.

6.104.3.32 void WaveSystem::set_wave_entities (tdt::uint *val*)

Sets the number of entities the current wave has alive.

Parameters

The	enw number of entities.

Definition at line 234 of file WaveSystem.cpp.

6.104.3.33 void WaveSystem::set_wave_table (const std::string & *val*)

Sets the table this system is using to create waves.

Parameters

Name	of the new wave table.

Definition at line 179 of file WaveSystem.cpp.

6.104.3.34 void WaveSystem::spawn ( ) `[private]`

Spawns the next batch of entities.

Definition at line 293 of file WaveSystem.cpp.

6.104.3.35 void WaveSystem::start ( )

Starts (and initializes) the wave system.

Definition at line 81 of file WaveSystem.cpp.

6.104.3.36 void WaveSystem::start_wave ( ) `[private]`

Starts the current wave.

Definition at line 269 of file WaveSystem.cpp.

6.104.3.37 void WaveSystem::update (tdt::real *delta*) `[override]`, `[virtual]`

Updates all necessary timers and checks if a wave has arrived and if so, spawns it.

Parameters

Time	since last frame.

Implements System.

Definition at line 15 of file WaveSystem.cpp.

6.104.3.38 void WaveSystem::update_label_text ( )

Updates the countdown window's text.

Definition at line 137 of file WaveSystem.cpp.

6.104.3.39 void WaveSystem::wave_entity_died ( )

Registers a death of an entity that belongs to the current wave (called by it's destructor).

Definition at line 72 of file WaveSystem.cpp.

6.104.4 Member Data Documentation

6.104.4.1 tdt::uint WaveSystem::curr_wave_number_ `[private]`

Number of the current wave.

Definition at line 278 of file WaveSystem.hpp.

6.104.4.2 bool WaveSystem::endless_mode_ [private]

If true, the current wave keeps repeating.

Definition at line 351 of file WaveSystem.hpp.

6.104.4.3 EntitySystem& WaveSystem::entities_ [private]

Entity system in which the wave entities will be created.

Definition at line 273 of file WaveSystem.hpp.

6.104.4.4 tdt::uint WaveSystem::entities_spawned_ [private]

Number of entities that were already spawned during this wave.

Definition at line 295 of file WaveSystem.hpp.

6.104.4.5 tdt::uint WaveSystem::entities_total_ [private]

Total number of entities in the current wave.

Definition at line 300 of file WaveSystem.hpp.

6.104.4.6 std::vector<std::string> WaveSystem::entity_blueprints_ [private]

Entities that are gonna be spawned.

Definition at line 346 of file WaveSystem.hpp.

6.104.4.7 std::string WaveSystem::label_text_ [private]

Text that is displayed in the countdown window.

Definition at line 315 of file WaveSystem.hpp.

6.104.4.8 tdt::uint WaveSystem::next_wave_countdown_ [private]

Time in seconds till the next wave starts.

Definition at line 305 of file WaveSystem.hpp.

6.104.4.9 lpp::Script& WaveSystem::script_ [private]

Reference to the script engine for easier use.

Definition at line 325 of file WaveSystem.hpp.

6.104.4.10 tdt::real WaveSystem::second_timer_ [private]

Auxiliary timer that is used to measure seconds.

Definition at line 310 of file WaveSystem.hpp.

6.104.4.11 tdt::real WaveSystem::spawn_timer_ [private]

Timers used for entity spawning.

Definition at line 335 of file WaveSystem.hpp.

6.104.4.12 std::vector<tdt::uint> WaveSystem::spawning_nodes_ [private]

Nodes that are used for entity spawning.

Definition at line 330 of file WaveSystem.hpp.

6.104.4.13 WAVE_STATE WaveSystem::state_ [private]

The current state of the system.

Definition at line 268 of file WaveSystem.hpp.

6.104.4.14 tdt::uint WaveSystem::wave_count_ [private]

Total number of waves.

Definition at line 283 of file WaveSystem.hpp.

6.104.4.15 tdt::uint WaveSystem::wave_entities_ [private]

Number of entities in the current wave that are spawned and still alive.

Definition at line 289 of file WaveSystem.hpp.

6.104.4.16 std::string WaveSystem::wave_table_ [private]

Name of the table containing init, wstart and wend functions that define a wave system.

Definition at line 341 of file WaveSystem.hpp.

6.104.4.17 CEGUI::Window* WaveSystem::window_ [private]

Pointer to the countdown window.

Definition at line 320 of file WaveSystem.hpp.

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

- systems/WaveSystem.hpp
- systems/WaveSystem.cpp

Index