

3dchess

Generated by Doxygen 1.8.6

Mon Jan 6 2014 02:31:52

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	Namespace Documentation	7
4.1	DebugTools Namespace Reference	7
4.1.1	Detailed Description	7
4.2	freetype Namespace Reference	7
4.2.1	Detailed Description	8
4.2.2	Function Documentation	8
4.2.2.1	next_p2	8
4.2.2.2	pop_projection_matrix	8
4.2.2.3	print	8
4.2.2.4	pushScreenCoordinateMatrix	8
5	Class Documentation	9
5.1	AbstractEvaluator Class Reference	9
5.2	AbstractGameLogic Class Reference	9
5.2.1	Detailed Description	10
5.2.2	Member Function Documentation	10
5.2.2.1	addObserver	10
5.2.2.2	getConfiguration	10
5.2.2.3	getWinner	10
5.2.2.4	isGameOver	11
5.2.2.5	start	11
5.3	AbstractGameObserver Class Reference	11
5.3.1	Detailed Description	11
5.3.2	Member Function Documentation	12

5.3.2.1	onGameOver	12
5.3.2.2	onGameStart	12
5.3.2.3	onTurnEnd	12
5.3.2.4	onTurnStart	12
5.3.2.5	onTurnTimeout	13
5.4	AbstractPlayer Class Reference	13
5.4.1	Detailed Description	13
5.4.2	Member Function Documentation	14
5.4.2.1	doMakeTurn	14
5.4.2.2	onSetColor	14
5.5	AbstractState Class Reference	14
5.5.1	Detailed Description	15
5.5.2	Member Function Documentation	15
5.5.2.1	enter	15
5.5.2.2	exit	15
5.5.2.3	run	15
5.6	AIPlayer Class Reference	16
5.6.1	Detailed Description	16
5.6.2	Constructor & Destructor Documentation	16
5.6.2.1	AIPlayer	16
5.6.3	Member Function Documentation	17
5.6.3.1	doMakeTurn	17
5.6.3.2	getState	17
5.6.3.3	onGameOver	17
5.6.3.4	onGameStart	17
5.6.3.5	onSetColor	17
5.7	AnimationHelper Class Reference	18
5.7.1	Detailed Description	18
5.7.2	Member Enumeration Documentation	18
5.7.2.1	FunctionType	18
5.7.3	Constructor & Destructor Documentation	19
5.7.3.1	AnimationHelper	19
5.7.4	Member Function Documentation	20
5.7.4.1	ease	20
5.7.4.2	hasStopped	20
5.8	AssimpHelper Class Reference	20
5.8.1	Detailed Description	20
5.8.2	Member Function Documentation	21
5.8.2.1	importScene	21
5.9	ChessBoard Class Reference	22

5.10 ChessSet Class Reference	23
5.10.1 Detailed Description	23
5.10.2 Member Function Documentation	23
5.10.2.1 getResourcesCount	23
5.10.2.2 loadResources	23
5.10.2.3 registerLoadCallback	24
5.10.2.4 setState	25
5.11 ConsolePlayer Class Reference	25
5.11.1 Detailed Description	26
5.11.2 Member Function Documentation	26
5.11.2.1 doMakeTurn	26
5.11.2.2 onGameOver	26
5.11.2.3 onGameStart	26
5.11.2.4 onSetColor	27
5.11.2.5 onTurnEnd	27
5.12 DummyPlayer Class Reference	27
5.12.1 Detailed Description	28
5.12.2 Member Function Documentation	28
5.12.2.1 doMakeTurn	28
5.12.2.2 onSetColor	28
5.13 StateMachine::EventMap Struct Reference	28
5.13.1 Detailed Description	29
5.14 freetype::font_data Struct Reference	29
5.14.1 Detailed Description	29
5.15 GameConfiguration Struct Reference	30
5.15.1 Detailed Description	30
5.15.2 Member Function Documentation	30
5.15.2.1 load	30
5.15.2.2 save	31
5.15.2.3 save	32
5.16 GameLogic Class Reference	32
5.16.1 Detailed Description	33
5.16.2 Constructor & Destructor Documentation	33
5.16.2.1 GameLogic	33
5.16.3 Member Function Documentation	33
5.16.3.1 addObserver	33
5.16.3.2 getConfiguration	33
5.16.3.3 getWinner	33
5.16.3.4 isGameOver	34
5.17 GamePlay Class Reference	34

5.17.1 Detailed Description	35
5.17.2 Constructor & Destructor Documentation	35
5.17.2.1 GamePlay	35
5.17.3 Member Function Documentation	35
5.17.3.1 enter	35
5.17.3.2 exit	35
5.17.3.3 run	35
5.17.3.4 setCapturedPiecesList	36
5.18 GameState Class Reference	36
5.19 GuiObserver Class Reference	36
5.19.1 Detailed Description	37
5.19.2 Constructor & Destructor Documentation	37
5.19.2.1 GuiObserver	37
5.19.3 Member Function Documentation	37
5.19.3.1 onGameOver	37
5.19.3.2 onGameStart	37
5.19.3.3 onTurnEnd	38
5.19.3.4 onTurnStart	38
5.19.3.5 onTurnTimeout	38
5.20 GuiWindow Class Reference	38
5.20.1 Detailed Description	39
5.20.2 Constructor & Destructor Documentation	40
5.20.2.1 GuiWindow	40
5.20.3 Member Function Documentation	40
5.20.3.1 getCameraDistanceToOrigin	40
5.20.3.2 getHeight	40
5.20.3.3 getWidth	40
5.20.3.4 isFullscreen	40
5.20.3.5 printHeadline	40
5.20.3.6 printSubHeadline	41
5.20.3.7 printText	41
5.20.3.8 printTextCenter	41
5.20.3.9 printTextSmall	41
5.20.3.10 switchWindowMode	42
5.21 has_toString< T > Struct Template Reference	42
5.22 IncrementalBoardEvaluator Class Reference	42
5.22.1 Detailed Description	43
5.23 IncrementalZobristHasher< THASH, SEED > Class Template Reference	43
5.23.1 Detailed Description	43
5.24 LoggingGameObserver Class Reference	43

5.24.1 Detailed Description	44
5.24.2 Member Function Documentation	44
5.24.2.1 onGameOver	44
5.24.2.2 onGameStart	44
5.24.2.3 onTurnEnd	44
5.24.2.4 onTurnStart	44
5.24.2.5 onTurnTimeout	44
5.25 MaterialEvaluator Class Reference	45
5.26 Menu2D Class Reference	45
5.27 Menu2DItem Class Reference	46
5.27.1 Detailed Description	46
5.27.2 Constructor & Destructor Documentation	46
5.27.2.1 Menu2DItem	46
5.27.3 Member Function Documentation	47
5.27.3.1 draw	47
5.27.3.2 mouseMoved	47
5.27.3.3 mousePressed	47
5.27.3.4 mouseReleased	47
5.27.3.5 onClick	47
5.27.3.6 setPosition	48
5.28 MenuGameMode Class Reference	48
5.28.1 Detailed Description	49
5.28.2 Member Function Documentation	49
5.28.2.1 enter	49
5.28.2.2 exit	49
5.28.2.3 run	49
5.29 MenuLoadGame Class Reference	49
5.29.1 Detailed Description	50
5.29.2 Member Function Documentation	50
5.29.2.1 enter	50
5.29.2.2 exit	50
5.29.2.3 run	51
5.30 MenuMain Class Reference	51
5.30.1 Detailed Description	52
5.30.2 Member Function Documentation	52
5.30.2.1 enter	52
5.30.2.2 exit	52
5.30.2.3 run	52
5.31 MenuOptions Class Reference	52
5.31.1 Detailed Description	53

5.31.2	Member Function Documentation	53
5.31.2.1	enter	53
5.31.2.2	exit	53
5.31.2.3	run	54
5.32	MenuPlayerColor Class Reference	54
5.32.1	Detailed Description	54
5.32.2	Member Function Documentation	55
5.32.2.1	enter	55
5.32.2.2	exit	55
5.32.2.3	run	55
5.33	Mesh Class Reference	55
5.33.1	Detailed Description	56
5.33.2	Constructor & Destructor Documentation	56
5.33.2.1	Mesh	56
5.34	Model Class Reference	56
5.34.1	Detailed Description	57
5.34.2	Constructor & Destructor Documentation	57
5.34.2.1	Model	57
5.34.3	Member Function Documentation	57
5.34.3.1	setColor	57
5.34.3.2	setCorrectionValues	57
5.34.3.3	setPosition	58
5.35	Negamax Class Reference	58
5.35.1	Detailed Description	58
5.35.2	Member Function Documentation	59
5.35.2.1	search	59
5.36	ObjectHelper Class Reference	59
5.36.1	Detailed Description	59
5.36.2	Member Function Documentation	60
5.36.2.1	create2DGradientRectList	60
5.36.2.2	create2DRectList	61
5.36.2.3	createCubeList	61
5.37	ObserverDispatcherProxy Class Reference	62
5.37.1	Detailed Description	62
5.37.2	Member Function Documentation	62
5.37.2.1	onGameOver	62
5.37.2.2	onGameStart	63
5.37.2.3	onTurnEnd	63
5.37.2.4	onTurnStart	63
5.37.2.5	onTurnTimeout	63

5.38	Negamax::PerfCounters Struct Reference	63
5.38.1	Detailed Description	64
5.39	Piece Struct Reference	64
5.40	PlayerDispatcherProxy Class Reference	64
5.40.1	Detailed Description	65
5.40.2	Member Function Documentation	65
5.40.2.1	doMakeTurn	65
5.40.2.2	onGameOver	66
5.40.2.3	onGameStart	66
5.40.2.4	onSetColor	66
5.40.2.5	onTurnEnd	66
5.40.2.6	onTurnStart	67
5.40.2.7	onTurnTimeout	67
5.41	PoF Struct Reference	67
5.42	Negamax::Result Struct Reference	67
5.42.1	Detailed Description	68
5.43	ServiceDispatcher Class Reference	68
5.43.1	Detailed Description	69
5.43.2	Member Function Documentation	69
5.43.2.1	poll	69
5.43.2.2	post	69
5.43.2.3	postPromise	69
5.43.2.4	run	70
5.44	ServiceDispatcherThread Class Reference	70
5.44.1	Detailed Description	70
5.44.2	Constructor & Destructor Documentation	71
5.44.2.1	ServiceDispatcherThread	71
5.44.3	Member Function Documentation	71
5.44.3.1	stop	71
5.45	StateMachine Class Reference	71
5.45.1	Detailed Description	72
5.45.2	Member Function Documentation	72
5.45.2.1	getInstance	72
5.45.2.2	run	72
5.45.2.3	setNextState	72
5.45.2.4	setStartState	72
5.46	Turn Class Reference	73
5.47	TurnGenerator Class Reference	73

Chapter 1

Namespace Index

1.1 Namespace List

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

DebugTools	Contains functions for helping with debugging tasks	7
freetype	FreeType Headers	7

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

AbstractEvaluator	9
MaterialEvaluator	45
AbstractGameLogic	9
GameLogic	32
AbstractGameObserver	11
AbstractPlayer	13
AIPlayer	16
ConsolePlayer	25
DummyPlayer	27
PlayerDispatcherProxy	64
GuiObserver	36
LoggingGameObserver	43
ObserverDispatcherProxy	62
AbstractState	14
GamePlay	34
MenuGameMode	48
MenuLoadGame	49
MenuMain	51
MenuOptions	52
MenuPlayerColor	54
AnimationHelper	18
AssimpHelper	20
ChessBoard	22
ChessSet	23
StateMachine::EventMap	28
freetype::font_data	29
GameConfiguration	30
GameState	36
GuiWindow	38
has_toString< T >	42
IncrementalBoardEvaluator	42
IncrementalZobristHasher< THASH, SEED >	43
Menu2D	45
Menu2DItem	46
Mesh	55
Model	56
Negamax	58

ObjectHelper	59
Negamax::PerfCounters	63
Piece	64
PoF	67
Negamax::Result	67
ServiceDispatcher	68
ObserverDispatcherProxy	62
PlayerDispatcherProxy	64
ServiceDispatcherThread	70
ConsolePlayer	25
DummyPlayer	27
StateMachine	71
Turn	73
TurnGenerator	73

Chapter 3

Class Index

3.1 Class List

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

AbstractEvaluator	9
AbstractGameLogic	
Interface for chess game logic implementations	9
AbstractGameObserver	
Allows to observe relevant GameEvents inside the GameLogic . Classes of this type can be registered with the GameLogic to be notified of relevant game events	11
AbstractPlayer	
Class a player has to implement to interact with the GameLogic . Every player is also a AbstractGameObserver which is notified of relevant game events. You do not need to register the player as an observer for this to happen	13
AbstractState	
Interface for modelling a game state	14
AIPlayer	
Artificial intelligence player implementation	16
AnimationHelper	18
AssimpHelper	20
ChessBoard	22
ChessSet	
The ChessSet holds all the figures together with the board needed for the chess game	23
ConsolePlayer	
Class which takes human player interaction from a console	25
DummyPlayer	
Player implementation which takes random turns after random amounts of time	27
StateMachine::EventMap	
Structure for holding user events	28
freetype::font_data	29
GameConfiguration	
Class for holding game configuration parameters	30
GameLogic	
GameLogic implementation for a game of chess with observers	32
GamePlay	
Class which holds the state GamePlay . This state is the essential part of all states. The whole game play is hold in this state	34
GameState	36
GuiObserver	
Allows to observe relevant GameEvents inside the GameLogic . Classes of this type can be registered with the GameLogic to be notified of relevant game events	36
GuiWindow	38

has_toString< T >	42
IncrementalBoardEvaluator	
Class for incrementally estimating game state	42
IncrementalZobristHasher< THASH, SEED >	
Zobrist-Hash implementation. Uses mt19937 to initialize from a fixed seed 452134	43
LoggingGameObserver	
AbstractGameObserver which simply logs occurring events	43
MaterialEvaluator	45
Menu2D	45
Menu2DItem	
This class describes a button of a menu. The button is a rectangle with textures depending on one of three states (normal, hover, active). The buttons can be used with the mouse cursor	46
MenuGameMode	
Class which holds the state GameMode . This state let the user choose one of two modes. The <i>AI</i> vs. <i>AI</i> mode which shows a chess match between two artificial computer players where the user can only watch the game. In the <i>Player</i> vs. <i>AI</i> mode, the user can play against an artificial computer player	48
MenuLoadGame	
Class which holds the state LoadGame . The user can load a previously saved game from one of three game slots	49
MenuMain	
Class which holds the state MenuMain . in this menu, the user can start a new game, load a previously saved game, go to the options menu or quit the game	51
MenuOptions	
Class which holds the state MenuOption . This state let the user toggle between the fullscreen view or the windowed mode of the game	52
MenuPlayerColor	
Class which holds the state PlayerColor . This state let the user choose between black or white for the chess model figures	54
Mesh	
Wrapper class for the Assimp library	55
Model	
Representing a chess figure (e.g. King, Queen, ...)	56
Negamax	
Implementation of a Negamax algorithm	58
ObjectHelper	
Helper class for creating static OpenGL display lists to boost the drawing of OpenGL objects	59
ObserverDispatcherProxy	
Proxy for transporting AbstractGameObserver events between threads	62
Negamax::PerfCounters	
Structure with performance counters used for debugging and evaluation	63
Piece	64
PlayerDispatcherProxy	
Proxy for transporting AbstractGamePlayer events between threads	64
PoF	67
Negamax::Result	
Structure for holding search results	67
ServiceDispatcher	
Provides functionality for safely running operations in a thread	68
ServiceDispatcherThread	
Provides functionality for safely running operations in a thread	70
StateMachine	
Class which manages the states	71
Turn	73
TurnGenerator	73

Chapter 4

Namespace Documentation

4.1 DebugTools Namespace Reference

Contains functions for helping with debugging tasks.

Functions

- string [tolInitializerList](#) (const std::array< [Piece](#), 64 > &board)
Returns the code needed to initialize a board to the given state.
- template<typename Rng >
[GameState](#) **generateRandomState** (size_t maxTurns, Rng &rng)
- template<typename Rng >
[ChessBoard](#) **generateRandomBoard** (size_t maxTurns, Rng &rng)

4.1.1 Detailed Description

Contains functions for helping with debugging tasks.

4.2 freetype Namespace Reference

FreeType Headers.

Classes

- struct [font_data](#)

Functions

- int [next_p2](#) (int a)
- void [make_dlist](#) (FT_Face face, char ch, GLuint list_base, GLuint *tex_base)
Create a display list corresponding to the give character.
- void [pushScreenCoordinateMatrix](#) ()
- void [pop_projection_matrix](#) ()
- void [print](#) (const [font_data](#) &ft_font, float x, float y, const char *fmt,...)

4.2.1 Detailed Description

FreeType Headers. OpenGL Headers Some STL headers Using the STL exception library increases the chances that someone else using our code will correctly catch any exceptions that we throw. MSVC will spit out all sorts of useless warnings if you create vectors of strings, this pragma gets rid of them. Wrap everything in a namespace, that we can use common function names like "print" without worrying about overlapping with anyone else's code.

4.2.2 Function Documentation

4.2.2.1 `int freetype::next_p2(int a) [inline]`

This function gets the first power of 2 \geq the int that we pass it.

4.2.2.2 `void freetype::pop_projection_matrix() [inline]`

Pops the projection matrix without changing the current MatrixMode.

4.2.2.3 `void freetype::print(const font_data & ft_font, float x, float y, const char * fmt, ...)`

Much like Nehe's glPrint function, but modified to work with freetype fonts.

The flagship function of the library - this thing will print out text at window coordinates x,y, using the font ft_font. The current modelview matrix will also be applied to the text.

4.2.2.4 `void freetype::pushScreenCoordinateMatrix() [inline]`

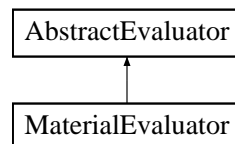
A fairly straight forward function that pushes a projection matrix that will make object world coordinates identical to window coordinates.

Chapter 5

Class Documentation

5.1 AbstractEvaluator Class Reference

Inheritance diagram for AbstractEvaluator:



Public Member Functions

- virtual Score **getScore** (const [GameState](#) &gameState) const =0

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

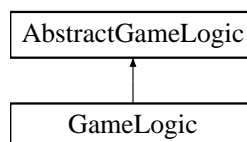
- S:/dev/3dchess/src/logic/interface/AbstractEvaluator.h

5.2 AbstractGameLogic Class Reference

Interface for chess game logic implementations.

```
#include <AbstractGameLogic.h>
```

Inheritance diagram for AbstractGameLogic:



Public Member Functions

- virtual AbstractPlayerPtr **getWhitePlayer** () const =0
- virtual AbstractPlayerPtr **getBlackPlayer** () const =0
- virtual void [addObserver](#) (AbstractGameObserverPtr observer)=0

Registers an observer for game events.

- virtual bool [isGameOver](#) () const =0
- virtual PlayerColor [getWinner](#) () const =0
- virtual GameConfigurationPtr [getConfiguration](#) () const =0
- virtual void [start](#) ()

Starts the game logic thread.

- virtual void [join](#) ()

Will block until the logic thread terminated. Be sure to call stop first to initiate logic thread shutdown.

- virtual void [stop](#) ()=0

Initiates a shutdown of the game logic.

Protected Member Functions

- virtual void [run](#) ()=0

Actual game logic function. Called by start function on the game logic thread to run the actual logic.

Protected Attributes

- std::thread [m_thread](#)

Game logic thread.

5.2.1 Detailed Description

Interface for chess game logic implementations.

5.2.2 Member Function Documentation

5.2.2.1 virtual void [AbstractGameLogic::addObserver](#) ([AbstractGameObserverPtr observer](#)) [pure virtual]

Registers an observer for game events.

See Also

[AbstractGameObserver](#) for the available events.

Parameters

<i>observer</i>	Observer to register.
-----------------	-----------------------

Implemented in [GameLogic](#).

5.2.2.2 virtual GameConfigurationPtr [AbstractGameLogic::getConfiguration](#) () const [pure virtual]

Returns

[GameConfiguration](#) currently used.

Implemented in [GameLogic](#).

5.2.2.3 virtual PlayerColor [AbstractGameLogic::getWinner](#) () const [pure virtual]

Returns

If isGameOver returns the winner of the game.

Implemented in [GameLogic](#).

5.2.2.4 `virtual bool AbstractGameLogic::isGameOver () const [pure virtual]`

Returns

true if game has ended.

Implemented in [GameLogic](#).

5.2.2.5 `virtual void AbstractGameLogic::start () [inline],[virtual]`

Starts the game logic thread.

See Also

[run](#)

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

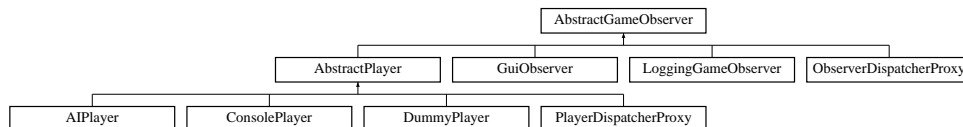
- `S:/dev/3dchess/src/logic/interface/AbstractGameLogic.h`

5.3 AbstractGameObserver Class Reference

Allows to observe relevant GameEvents inside the [GameLogic](#). Classes of this type can be registered with the [GameLogic](#) to be notified of relevant game events.

```
#include <AbstractGameObserver.h>
```

Inheritance diagram for AbstractGameObserver:



Public Member Functions

- virtual void [onGameStart](#) ([GameState](#) state, [GameConfiguration](#) config)
Called when the game starts.
- virtual void [onTurnStart](#) (PlayerColor who)
Called if a player is asked to perform a turn.
- virtual void [onTurnEnd](#) (PlayerColor who, [Turn](#) turn, [GameState](#) newState)
Called if a player ended its turn.
- virtual void [onTurnTimeout](#) (PlayerColor who, std::chrono::seconds timeout)
Called if a players turn is aborted due to timeout.
- virtual void [onGameOver](#) ([GameState](#) state, PlayerColor winner)
Called when a game started with onGameStart is over.

5.3.1 Detailed Description

Allows to observe relevant GameEvents inside the [GameLogic](#). Classes of this type can be registered with the [GameLogic](#) to be notified of relevant game events.

Note

A Observer is only required to stay in a valid state for one game. It is free to halt its operations after the end of the game.

Warning

None of the functions in the class must block.

5.3.2 Member Function Documentation

5.3.2.1 `virtual void AbstractGameObserver::onGameOver (GameState state, PlayerColor winner)` `[inline]`, `[virtual]`

Called when a game started with `onGameStart` is over.

Parameters

<i>state</i>	State on game over.
<i>winner</i>	Winner of the game.

Reimplemented in [PlayerDispatcherProxy](#), [GuiObserver](#), [ObserverDispatcherProxy](#), [AIPlayer](#), [ConsolePlayer](#), and [LoggingGameObserver](#).

5.3.2.2 `virtual void AbstractGameObserver::onGameStart (GameState state, GameConfiguration config)` `[inline]`, `[virtual]`

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
<i>config</i>	Valid GameConfiguration for this game.

Reimplemented in [PlayerDispatcherProxy](#), [GuiObserver](#), [AIPlayer](#), [ObserverDispatcherProxy](#), [ConsolePlayer](#), and [LoggingGameObserver](#).

5.3.2.3 `virtual void AbstractGameObserver::onTurnEnd (PlayerColor who, Turn turn, GameState newState)` `[inline]`, `[virtual]`

Called if a player ended its turn.

Parameters

<i>who</i>	Color of the player doing the turn.
<i>turn</i>	Turn the player decided on.
<i>newState</i>	State after the player performed the turn.

Reimplemented in [PlayerDispatcherProxy](#), [GuiObserver](#), [ObserverDispatcherProxy](#), [ConsolePlayer](#), and [LoggingGameObserver](#).

5.3.2.4 `virtual void AbstractGameObserver::onTurnStart (PlayerColor who)` `[inline]`, `[virtual]`

Called if a player is asked to perform a turn.

Parameters

<i>who</i>	Color of the player doing the turn.
------------	-------------------------------------

Reimplemented in [PlayerDispatcherProxy](#), [GuiObserver](#), [ObserverDispatcherProxy](#), and [LoggingGameObserver](#).

5.3.2.5 `virtual void AbstractGameObserver::onTurnTimeout (PlayerColor who, std::chrono::seconds timeout)` `[inline]`, `[virtual]`

Called if a players turn is aborted due to timeout.

Parameters

<i>who</i>	Color of the player who got interrupted.
<i>timeout</i>	Length of the time limit that got violated.

Reimplemented in [PlayerDispatcherProxy](#), [GuiObserver](#), [ObserverDispatcherProxy](#), and [LoggingGameObserver](#).

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

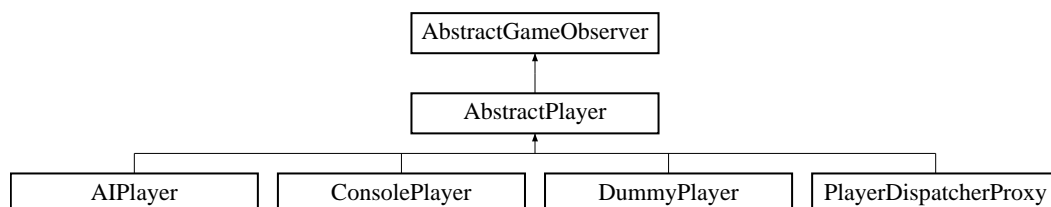
- `S:/dev/3dchess/src/logic/interface/AbstractGameObserver.h`

5.4 AbstractPlayer Class Reference

Class a player has to implement to interact with the [GameLogic](#). Every player is also a [AbstractGameObserver](#) which is notified of relevant game events. You do not need to register the player as an observer for this to happen.

```
#include <AbstractPlayer.h>
```

Inheritance diagram for AbstractPlayer:



Public Member Functions

- `virtual void onSetColor (PlayerColor color)=0`
Notifies that player what color he will be playing. Called before onGameStart.
- `virtual std::future< Turn > doMakeTurn (GameState state)=0`
Asks the player to make his turn.
- `virtual void doAbortTurn ()=0`
Asks the player to abort a turn asked for with doMakeTurn. When this is called the [GameLogic](#) will no longer react to the completion of the future for that turn. A use of this function is the abortion of a turn due to timeout.

5.4.1 Detailed Description

Class a player has to implement to interact with the [GameLogic](#). Every player is also a [AbstractGameObserver](#) which is notified of relevant game events. You do not need to register the player as an observer for this to happen.

Note

A Observer is only required to stay in a valid state for one game. It is free to halt its operations after the end of the game.

Warning

None of the functions in the class must block.

5.4.2 Member Function Documentation**5.4.2.1** `virtual std::future<Turn> AbstractPlayer::doMakeTurn (GameState state) [pure virtual]`

Asks the player to make his turn.

Warning

This function must not block. It is to return immediatly. The players turn is to be set on the returned future.

Note

The game logic can abort its request for a player to make his turn using the doAbortTurn function at any time.

Parameters

<i>state</i>	Current state of the game.
--------------	----------------------------

Returns

A future to the turn to make.

Implemented in [DummyPlayer](#), [PlayerDispatcherProxy](#), [AIPlayer](#), and [ConsolePlayer](#).

5.4.2.2 `virtual void AbstractPlayer::onSetColor (PlayerColor color) [pure virtual]`

Notifies that player what color he will be playing. Called before onGameStart.

Parameters

<i>color</i>	Color the player has.
--------------	-----------------------

Implemented in [DummyPlayer](#), [AIPlayer](#), [PlayerDispatcherProxy](#), and [ConsolePlayer](#).

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

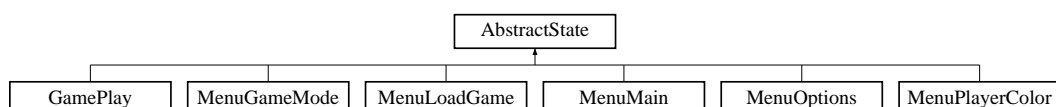
- `S:/dev/3dchess/src/logic/interface/AbstractPlayer.h`

5.5 AbstractState Class Reference

Interface for modelling a game state.

```
#include <AbstractState.h>
```

Inheritance diagram for AbstractState:



Public Member Functions

- virtual void [enter](#) ()=0
Enters the state for the first time. This will setup all the state related stuff.
- virtual [AbstractState](#) * [run](#) ()=0
Runs the current state and does all the work.
- virtual void [exit](#) ()=0
Exits the current state and cleans up all allocated resources.
- virtual void [draw](#) ()=0
Draws something state related stuff on the screen.

5.5.1 Detailed Description

Interface for modelling a game state.

Note

To [run\(\)](#) a state, first [enter\(\)](#) the state.

5.5.2 Member Function Documentation

5.5.2.1 virtual void AbstractState::enter () [pure virtual]

Enters the state for the first time. This will setup all the state related stuff.

Note

To [run\(\)](#) the current state, first [enter\(\)](#) it.

Implemented in [GamePlay](#), [MenuGameMode](#), [MenuLoadGame](#), [MenuOptions](#), [MenuMain](#), and [MenuPlayerColor](#).

5.5.2.2 virtual void AbstractState::exit () [pure virtual]

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implemented in [GamePlay](#), [MenuGameMode](#), [MenuLoadGame](#), [MenuOptions](#), [MenuMain](#), and [MenuPlayerColor](#).

5.5.2.3 virtual AbstractState* AbstractState::run () [pure virtual]

Runs the current state and does all the work.

Returns

[AbstractState](#)* the state which should be run after this state. A nullptr if the game should be exited.

Implemented in [GamePlay](#), [MenuGameMode](#), [MenuLoadGame](#), [MenuOptions](#), [MenuMain](#), and [MenuPlayerColor](#).

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

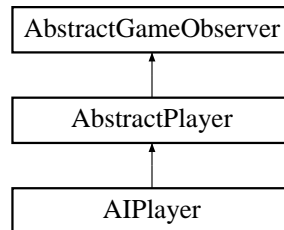
- S:/dev/3dchess/src/gui/interface/AbstractState.h

5.6 AIPlayer Class Reference

Artificial intelligence player implementation.

```
#include <AIPlayer.h>
```

Inheritance diagram for AIPlayer:



Public Types

- enum [States](#) { **PREPARATION**, **PONDERING**, **PLAYING**, **STOPPED** }
States for [AIPlayer](#).

Public Member Functions

- [AIPlayer](#) ()
Creates a new [AIPlayer](#).
- void [start](#) ()
Starts the [AIPlayer](#) thread.
- virtual void [onSetColor](#) (PlayerColor color) override
Notifies that player what color he will be playing. Called before onGameStart.
- virtual void [onGameStart](#) (GameState state, GameConfiguration config) override
Called when the game starts.
- virtual std::future< [Turn](#) > [doMakeTurn](#) (GameState state) override
Asks the player to make his turn.
- virtual void [doAbortTurn](#) () override
Asks the player to abort a turn asked for with doMakeTurn. When this is called the [GameLogic](#) will no longer react to the completion of the future for that turn. A use of this function is the abortion of a turn due to timeout.
- virtual void [onGameOver](#) (GameState, PlayerColor) override
Called when a game started with onGameStart is over.
- [States](#) [getState](#) () const

5.6.1 Detailed Description

Artificial intelligence player implementation.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 AIPlayer::AIPlayer ()

Creates a new [AIPlayer](#).

Note

Don't forget to [start\(\)](#) it.

5.6.3 Member Function Documentation

5.6.3.1 `future< Turn > AIPlayer::doMakeTurn (GameState state) [override],[virtual]`

Asks the player to make his turn.

Warning

This function must not block. It is to return immediatly. The players turn is to be set on the returned future.

Note

The game logic can abort its request for a player to make his turn using the `doAbortTurn` function at any time.

Parameters

<i>state</i>	Current state of the game.
--------------	----------------------------

Returns

A future to the turn to make.

Implements [AbstractPlayer](#).

5.6.3.2 `AIPlayer::States AIPlayer::getState () const`

Returns

Return current state.

5.6.3.3 `void AIPlayer::onGameOver (GameState state, PlayerColor winner) [override],[virtual]`

Called when a game started with `onGameStart` is over.

Parameters

<i>state</i>	State on game over.
<i>winner</i>	Winner of the game.

Reimplemented from [AbstractGameObserver](#).

5.6.3.4 `void AIPlayer::onGameStart (GameState state, GameConfiguration config) [override],[virtual]`

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
<i>config</i>	Valid GameConfiguration for this game.

Reimplemented from [AbstractGameObserver](#).

5.6.3.5 `void AIPlayer::onSetColor (PlayerColor color) [override],[virtual]`

Notifies that player what color he will be playing. Called before `onGameStart`.

Parameters

<i>color</i>	Color the player has.
--------------	-----------------------

Implements [AbstractPlayer](#).

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

- S:/dev/3dchess/src/ai/AIPlayer.h
- S:/dev/3dchess/src/ai/AIPlayer.cpp

5.7 AnimationHelper Class Reference

```
#include <AnimationHelper.h>
```

Public Types

- enum [FunctionType](#) { [EASE_LINEAR](#), [EASE_OUTSINE](#) }

The possible time function types.

Public Member Functions

- [AnimationHelper](#) (const int duration)
Creates a new [AnimationHelper](#) object.
- void [setStartNowOrKeepIt](#) ()
Sets the current time as start point for the animation. If this method is called multiple times, only the first call will take effect.
- void [reset](#) ()
Resets the start time stamp to the current time.
- float [ease](#) ([FunctionType](#) type, const float lowerBound, const float upperBound)
The percentage of the range between the lowerBound and upperBound.
- bool [hasStopped](#) ()
Gets the status of the animation.

5.7.1 Detailed Description

The class helps to create animations by providing time dependent methods.

5.7.2 Member Enumeration Documentation

5.7.2.1 enum AnimationHelper::FunctionType

The possible time function types.

Enumerator

EASE_LINEAR Linear.

EASE_OUTSINE Sinus like curve.

5.7.3 Constructor & Destructor Documentation

5.7.3.1 AnimationHelper::AnimationHelper (const int *duration*)

Creates a new [AnimationHelper](#) object.

Parameters

<i>duration</i>	The period how long the animation should took.
-----------------	--

5.7.4 Member Function Documentation

5.7.4.1 float AnimationHelper::ease (**FunctionType** *type*, const float *lowerBound*, const float *upperBound*)

The percentage of the range between the lowerBound and upperBound.

Note

The lowerBound must be less than upperBound.

Parameters

<i>type</i>	One of the FunctionType as defined above.
<i>lowerBound</i>	The lower bound of the range.
<i>upperBound</i>	The upper bound of the range.

Returns

The numeric value in percent. This will show the completeness of the animation in percent between 0.0 and 1.0;

5.7.4.2 bool AnimationHelper::hasStopped ()

Gets the status of the animation.

Returns

True if the animation was started and has already finished. False if not.

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

- S:/dev/3dchess/src/gui/AnimationHelper.h
- S:/dev/3dchess/src/gui/AnimationHelper.cpp

5.8 AssimpHelper Class Reference

```
#include <AssimpHelper.h>
```

Public Member Functions

- void [importScene](#) (std::string filename)
Imports the scene by filename.
- void [drawScene](#) ()
Draws the scene.

5.8.1 Detailed Description

Assimp wrapper class to handle scene modeling in an more comfortable way.

5.8.2 Member Function Documentation

5.8.2.1 void AssimpHelper::importScene (std::string *filename*)

Imports the scene by filename.

Parameters

<i>filename</i>	The filename of the scene to import.
-----------------	--------------------------------------

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

- S:/dev/3dchess/src/gui/AssimpHelper.h
- S:/dev/3dchess/src/gui/AssimpHelper.cpp

5.9 ChessBoard Class Reference

Public Member Functions

- **ChessBoard** (std::array< [Piece](#), 64 > board, PlayerColor [nextPlayer](#)=White)
- void **applyTurn** (const [Turn](#) &t)
- std::array< [Piece](#), 64 > **getBoard** () const
- std::vector< [Piece](#) > **getCapturedPieces** () const
- bool [hasBlackPieces](#) () const
Returns true if black pieces are on the board.
- bool [hasWhitePieces](#) () const
Returns true if white pieces are on the board.
- PlayerColor [getNextPlayer](#) () const
Return next player to make a turn.
- Score [getScore](#) (PlayerColor color) const
Returns the current estimated score according to the internal estimator.
- bool **operator==** (const [ChessBoard](#) &other) const
- bool **operator!=** (const [ChessBoard](#) &other) const
- std::string **toString** () const
- File [getEnPassantFile](#) () const
Returns the file where en-passant rights exist. NoFile if none.
- std::array< bool, NUM_PLAYERS > [getShortCastleRights](#) () const
Returns short castle rights for players.
- std::array< bool, NUM_PLAYERS > [getLongCastleRights](#) () const
Returns long castle rights for players.

Protected Attributes

- std::array< std::array
 < BitBoard, NUM_PIECETYPES+1 >
 , NUM_PLAYERS > **bb**
- std::array< bool, NUM_PLAYERS > [shortCastleRight](#)
Short castle rights for players.
- std::array< bool, NUM_PLAYERS > [longCastleRight](#)
Long castle rights for players.
- uint8_t [enPassantRightForFiles](#)
Bitmask for enemy en passant rights for a file next turn.
- PlayerColor [nextPlayer](#)
Player doing the next turn.

Friends

- class **TurnGenerator**

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

- S:/dev/3dchess/src/logic/ChessBoard.h
- S:/dev/3dchess/src/logic/ChessBoard.cpp

5.10 ChessSet Class Reference

The [ChessSet](#) holds all the figures together with the board needed for the chess game.

```
#include <ChessSet.h>
```

Public Member Functions

- void [setState](#) (std::array< [Piece](#), 64 > state, PlayerColor lastPlayer, [Turn](#) lastTurn)
Sets the new chess state.
- int [getResourcesCount](#) ()
Returns the number of big resources which must be loaded for initializing the [ChessSet](#).
- void [registerLoadCallback](#) (const boost::function< void(std::string)> &callback)
Registers a function as callback.
- void [loadResources](#) ()
Loads all resources, builds the models and the chess board.
- void [draw](#) ()
Draws the whole [ChessSet](#). This includes all models and the chess board. Depending in the current state.

5.10.1 Detailed Description

The [ChessSet](#) holds all the figures together with the board needed for the chess game.

5.10.2 Member Function Documentation

5.10.2.1 int ChessSet::getResourcesCount ()

Returns the number of big resources which must be loaded for initializing the [ChessSet](#).

Note

This can be used for a progress bar.

Returns

The number of big resources.

5.10.2.2 void ChessSet::loadResources ()

Loads all resources, builds the models and the chess board.

Note

If you've registered a function as callback, you will be informed on each resource which is loaded.

5.10.2.3 `void ChessSet::registerLoadCallback (const boost::function< void(std::string)> & callback)`

Registers a function as callback.

Parameters

<i>callback</i>	The function which will be called when a resource was successfully loaded.
-----------------	--

5.10.2.4 void ChessSet::setState (std::array< Piece, 64 > state, PlayerColor lastPlayer, Turn lastTurn)

Sets the new chess state.

Note

You need to call this only, when there's a visible change like moving a figure from field A to field B.

Parameters

<i>state</i>	The state of the current board. Only the given pieces on the fields will be drawn.
--------------	--

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

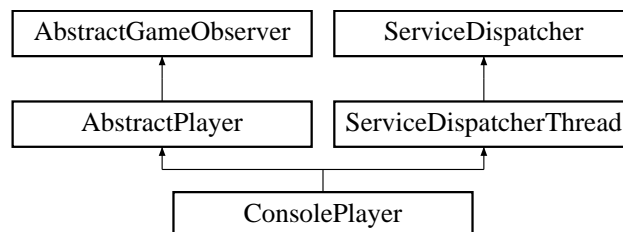
- S:/dev/3dchess/src/gui/ChessSet.h
- S:/dev/3dchess/src/gui/ChessSet.cpp

5.11 ConsolePlayer Class Reference

Class which takes human player interaction from a console.

```
#include <ConsolePlayer.h>
```

Inheritance diagram for ConsolePlayer:



Public Member Functions

- virtual void [onSetColor](#) (PlayerColor color) override
Notifies that player what color he will be playing. Called before onGameStart.
- virtual void [onGameStart](#) (GameState state, GameConfiguration config) override
Called when the game starts.
- virtual std::future< Turn > [doMakeTurn](#) (GameState state) override
Asks the player to make his turn.
- virtual void [onTurnEnd](#) (PlayerColor color, Turn turn, GameState newState) override
Called if a player ended its turn.
- virtual void [doAbortTurn](#) () override
Asks the player to abort a turn asked for with doMakeTurn. When this is called the GameLogic will no longer react to the completion of the future for that turn. A use of this function is the abortion of a turn due to timeout.
- virtual void [onGameOver](#) (GameState state, PlayerColor winner) override
Called when a game started with onGameStart is over.

Additional Inherited Members

5.11.1 Detailed Description

Class which takes human player interaction from a console.

Warning

Has serious issues on turn timeout due to blocking console reads.

5.11.2 Member Function Documentation

5.11.2.1 `future< Turn > ConsolePlayer::doMakeTurn (GameState state) [override],[virtual]`

Asks the player to make his turn.

Warning

This function must not block. It is to return immediatly. The players turn is to be set on the returned future.

Note

The game logic can abort its request for a player to make his turn using the `doAbortTurn` function at any time.

Parameters

<i>state</i>	Current state of the game.
--------------	----------------------------

Returns

A future to the turn to make.

Implements [AbstractPlayer](#).

5.11.2.2 `void ConsolePlayer::onGameOver (GameState state, PlayerColor winner) [override],[virtual]`

Called when a game started with `onGameStart` is over.

Parameters

<i>state</i>	State on game over.
<i>winner</i>	Winner of the game.

Reimplemented from [AbstractGameObserver](#).

5.11.2.3 `void ConsolePlayer::onGameStart (GameState state, GameConfiguration config) [override],[virtual]`

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
--------------	--

<i>config</i>	Valid GameConfiguration for this game.
---------------	--

Reimplemented from [AbstractGameObserver](#).

5.11.2.4 void ConsolePlayer::onSetColor (PlayerColor *color*) [override],[virtual]

Notifies that player what color he will be playing. Called before onGameStart.

Parameters

<i>color</i>	Color the player has.
--------------	-----------------------

Implements [AbstractPlayer](#).

5.11.2.5 void ConsolePlayer::onTurnEnd (PlayerColor *who*, Turn *turn*, GameState *newState*) [override],[virtual]

Called if a player ended its turn.

Parameters

<i>who</i>	Color of the player doing the turn.
<i>turn</i>	Turn the player decided on.
<i>newState</i>	State after the player performed the turn.

Reimplemented from [AbstractGameObserver](#).

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

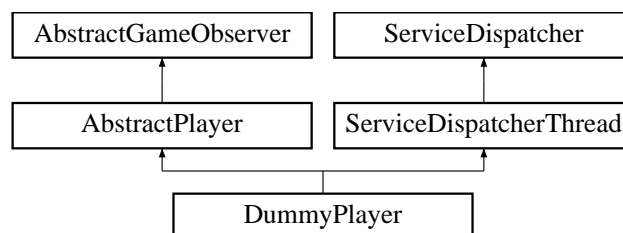
- S:/dev/3dchess/src/misc/ConsolePlayer.h
- S:/dev/3dchess/src/misc/ConsolePlayer.cpp

5.12 DummyPlayer Class Reference

Player implementation which takes random turns after random amounts of time.

```
#include <DummyPlayer.h>
```

Inheritance diagram for DummyPlayer:



Public Member Functions

- virtual void [onSetColor](#) (PlayerColor *color*) override
Notifies that player what color he will be playing. Called before onGameStart.
- virtual void [doAbortTurn](#) () override
Asks the player to abort a turn asked for with doMakeTurn. When this is called the [GameLogic](#) will no longer react to the completion of the future for that turn. A use of this function is the abortion of a turn due to timeout.
- virtual std::future< [Turn](#) > [doMakeTurn](#) (GameState *state*) override
Asks the player to make his turn.

Additional Inherited Members

5.12.1 Detailed Description

Player implementation which takes random turns after random amounts of time.

Warning

Does not react to doAbortTurn events.

5.12.2 Member Function Documentation

5.12.2.1 `virtual std::future<Turn> DummyPlayer::doMakeTurn (GameState state) [inline],[override],[virtual]`

Asks the player to make his turn.

Warning

This function must not block. It is to return immediatly. The players turn is to be set on the returned future.

Note

The game logic can abort its request for a player to make his turn using the doAbortTurn function at any time.

Parameters

<i>state</i>	Current state of the game.
--------------	----------------------------

Returns

A future to the turn to make.

Implements [AbstractPlayer](#).

5.12.2.2 `virtual void DummyPlayer::onSetColor (PlayerColor color) [inline],[override],[virtual]`

Notifies that player what color he will be playing. Called before onGameStart.

Parameters

<i>color</i>	Color the player has.
--------------	-----------------------

Implements [AbstractPlayer](#).

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

- S:/dev/3dchess/src/misc/DummyPlayer.h

5.13 StateMachine::EventMap Struct Reference

Structure for holding user events.

```
#include <StateMachine.h>
```

Public Attributes

- bool **mouseMoved** = false
- int **mouseX** = 0
- int **mouseY** = 0
- bool **mouseDown** = false
- bool **mouseUp** = false
- bool **keyLeft** = false
- bool **keyRight** = false
- bool **keyDown** = false
- bool **keyUp** = false
- bool **keyEscape** = false
- bool **key0** = false
- bool **key1** = false
- bool **keyA** = false
- bool **keyY** = false

5.13.1 Detailed Description

Structure for holding user events.

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

- S:/dev/3dchess/src/gui/StateMachine.h

5.14 freetype::font_data Struct Reference

```
#include <FreeType.h>
```

Public Member Functions

- void **init** (const char *fname, unsigned int h)
- void **clean** ()

Public Attributes

- float **h**
Holds the height of the font.
- GLuint * **textures**
Holds the texture id's.
- GLuint **list_base**
Holds the first display list id.

5.14.1 Detailed Description

This holds all of the information related to any freetype font that we want to create.

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

- S:/dev/3dchess/src/gui/FreeType.h
- S:/dev/3dchess/src/gui/FreeType.cpp

5.15 GameConfiguration Struct Reference

Class for holding game configuration parameters.

```
#include <GameConfiguration.h>
```

Public Member Functions

- bool [save](#) (const std::string &path) const
Saves this configuration to the given path.
- bool **operator==** (const [GameConfiguration](#) &other) const
- std::string **toString** () const

Static Public Member Functions

- static boost::optional
 < [GameConfiguration](#) > [load](#) (const std::string &path)
 Loads a game configuration from disk.
- static bool [save](#) (const [GameConfiguration](#) &config, const std::string &path)
 Saves a given game configuration to a file.

Public Attributes

- int [timeBetweenTurnsInSeconds](#)
 Minimum time between turns for display purposes.
- int [maximumTurnTimeInSeconds](#)
 Maximum time between turns after which to time out a move.

Friends

- class **boost::serialization::access**

5.15.1 Detailed Description

Class for holding game configuration parameters.

Note

Can be stored and read from disc using save/load.

5.15.2 Member Function Documentation

5.15.2.1 boost::optional< [GameConfiguration](#) > [GameConfiguration::load](#) (const std::string & *path*) [static]

Loads a game configuration from disk.

Parameters

<i>path</i>	Path to file.
-------------	---------------

Returns

[GameConfiguration](#) on success. boost::none on failure.

5.15.2.2 `bool GameConfiguration::save (const GameConfiguration & config, const std::string & path)` `[static]`

Saves a given game configuration to a file.

Parameters

<i>config</i>	Configuration to save.
<i>path</i>	Path to save configuration to.

Returns

True on success.

5.15.2.3 bool GameConfiguration::save (const std::string & *path*) const

Saves this configuration to the given path.

Parameters

<i>path</i>	Path to file to save to.
-------------	--------------------------

Returns

True on success.

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

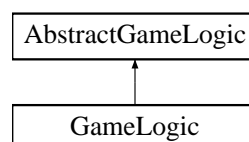
- S:/dev/3dchess/src/core/GameConfiguration.h
- S:/dev/3dchess/src/core/GameConfiguration.cpp

5.16 GameLogic Class Reference

[GameLogic](#) implementation for a game of chess with observers.

```
#include <GameLogic.h>
```

Inheritance diagram for GameLogic:



Public Member Functions

- [GameLogic](#) (AbstractPlayerPtr white, AbstractPlayerPtr black, GameConfigurationPtr config)
Sets up a [GameLogic](#) object for one chess game.
- virtual AbstractPlayerPtr **getWhitePlayer** () const override
- virtual AbstractPlayerPtr **getBlackPlayer** () const override
- virtual void [addObserver](#) (AbstractGameObserverPtr observer) override
Registers an observer for game events.
- virtual bool [isGameOver](#) () const override
- virtual PlayerColor [getWinner](#) () const override
- virtual GameConfigurationPtr [getConfiguration](#) () const override
- virtual void [stop](#) () override
Initiates a shutdown of the game logic.

Additional Inherited Members

5.16.1 Detailed Description

[GameLogic](#) implementation for a game of chess with observers.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 `GameLogic::GameLogic (AbstractPlayerPtr white, AbstractPlayerPtr black, GameConfigurationPtr config)`

Sets up a [GameLogic](#) object for one chess game.

Note

Don't forget to start operation by calling `start`.

Parameters

<i>white</i>	White player reference
<i>black</i>	Black player reference
<i>config</i>	Configuration for this game

5.16.3 Member Function Documentation

5.16.3.1 `void GameLogic::addObserver (AbstractGameObserverPtr observer) [override], [virtual]`

Registers an observer for game events.

See Also

[AbstractGameObserver](#) for the available events.

Parameters

<i>observer</i>	Observer to register.
-----------------	-----------------------

Implements [AbstractGameLogic](#).

5.16.3.2 `GameConfigurationPtr GameLogic::getConfiguration () const [override], [virtual]`

Returns

[GameConfiguration](#) currently used.

Implements [AbstractGameLogic](#).

5.16.3.3 `PlayerColor GameLogic::getWinner () const [override], [virtual]`

Returns

`isGameOver` returns the winner of the game.

Implements [AbstractGameLogic](#).

5.16.3.4 `bool GameLogic::isGameOver () const [override],[virtual]`

Returns

true if game has ended.

Implements [AbstractGameLogic](#).

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

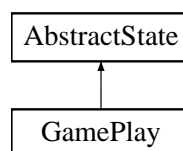
- S:/dev/3dchess/src/logic/GameLogic.h
- S:/dev/3dchess/src/logic/GameLogic.cpp

5.17 GamePlay Class Reference

Class which holds the state [GamePlay](#). This state is the essential part of all states. The whole game play is hold in this state.

```
#include <GamePlay.h>
```

Inheritance diagram for GamePlay:



Public Types

- enum [GameMode](#) { **AI_VS_AI**, **PLAYER_VS_AI** }

The possible game modes.

Public Member Functions

- [GamePlay](#) ([GameMode](#) mode, PlayerColor firstPlayerColor)
Creates a new game.
- void [enter](#) () override
Enters the state for the first time. This will setup all the state related stuff.
- [AbstractState](#) * [run](#) () override
Runs the current state and does all the work.
- void [exit](#) () override
Exits the current state and cleans up all allocated resources.
- void [draw](#) ()
Draws something state related stuff on the screen.
- void **startShowText** (std::string text)
- void **switchToPlayerColor** (PlayerColor color)
- void [setState](#) (std::array< [Piece](#), 64 > state, PlayerColor lastPlayer, [Turn](#) lastTurn)
Method for setting the new chess state. This method is non-blocking.
- void **setState** (std::array< [Piece](#), 64 > state)
- void [setCapturedPiecesList](#) (std::vector< [Piece](#) > piecesList)
Method for setting the new turn, which changed the chess state.
- void **onBeforeLoadNextResource** (std::string resourceName)

- void **onResumeGame** ()
- void **onSaveGame** ()
- void **onLeaveGame** ()
- void **onBackToMenu** ()

5.17.1 Detailed Description

Class which holds the state [GamePlay](#). This state is the essential part of all states. The whole game play is hold in this state.

Note

To [run\(\)](#) a state, first [enter\(\)](#) the state.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 `GamePlay::GamePlay (GameMode mode, PlayerColor firstPlayerColor)`

Creates a new game.

Parameters

<i>mode</i>	The GameMode (<i>AI vs. AI</i> or <i>Player vs. AI</i>).
<i>firstPlayerColor</i>	The color of the player which takes the first turn.

5.17.3 Member Function Documentation

5.17.3.1 `void GamePlay::enter () [override], [virtual]`

Enters the state for the first time. This will setup all the state related stuff.

Note

To [run\(\)](#) the current state, first [enter\(\)](#) it.

Implements [AbstractState](#).

5.17.3.2 `void GamePlay::exit () [override], [virtual]`

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implements [AbstractState](#).

5.17.3.3 `AbstractState* GamePlay::run () [override], [virtual]`

Runs the current state and does all the work.

Returns

`AbstractState*` the state which should be run after this state. A nullptr if the game should be exited.

Implements [AbstractState](#).

5.17.3.4 void Gameplay::setCapturedPiecesList (std::vector< Piece > piecesList)

Method for setting the new turn, which changed the chess state.

Note

Be sure to first call this and *after* call setState. This method is non-blocking.

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

- S:/dev/3dchess/src/gui/states/GamePlay.h
- S:/dev/3dchess/src/gui/states/GamePlay.cpp

5.18 GameState Class Reference

Public Member Functions

- **GameState** (const [ChessBoard](#) &chessBoard)
- virtual void **init** ()
- virtual std::vector< [Turn](#) > **getTurnList** () const
- virtual void **applyTurn** (const [Turn](#) &turn)
- virtual PlayerColor **getNextPlayer** () const
- virtual const [ChessBoard](#) & **getChessBoard** () const
- bool **isGameOver** () const
- PlayerColor **getWinner** () const
- Score **getScore** () const
- *Returns current score estimate from next players POV.*
- bool **operator==** (const [GameState](#) &other) const
- bool **operator!=** (const [GameState](#) &other) const
- std::string **toString** () const

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

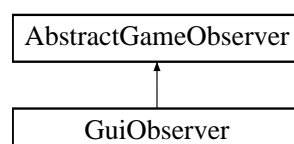
- S:/dev/3dchess/src/logic/GameState.h
- S:/dev/3dchess/src/logic/GameState.cpp

5.19 GuiObserver Class Reference

Allows to observe relevant GameEvents inside the [GameLogic](#). Classes of this type can be registered with the [GameLogic](#) to be notified of relevant game events.

```
#include <GuiObserver.h>
```

Inheritance diagram for GuiObserver:



Public Member Functions

- [GuiObserver](#) ([ChessSetPtr](#) chessSetPtr, [GamePlay](#) &gamePlayState)
Creates a new observer object.
- void [onGameStart](#) ([GameState](#) state, [GameConfiguration](#) config) override
Called when the game starts.
- void [onTurnStart](#) ([PlayerColor](#) who) override
Called if a player is asked to perform a turn.
- void [onTurnEnd](#) ([PlayerColor](#) who, [Turn](#) turn, [GameState](#) newState) override
Called if a player ended its turn.
- void [onTurnTimeout](#) ([PlayerColor](#) who, [std::chrono::seconds](#) timeout) override
Called if a players turn is aborted due to timeout.
- void [onGameOver](#) ([GameState](#) state, [PlayerColor](#) winner) override
Called when a game started with onGameStart is over.

5.19.1 Detailed Description

Allows to observe relevant GameEvents inside the [GameLogic](#). Classes of this type can be registered with the [GameLogic](#) to be notified of relevant game events.

Note

A Observer is only required to stay in a valid state for one game. It is free to halt its operations after the end of the game.

Warning

None of the functions in the class must block.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 [GuiObserver::GuiObserver](#) ([ChessSetPtr](#) chessSetPtr, [GamePlay](#) & gamePlayState)

Creates a new observer object.

Parameters

<i>chessSetPtr</i>	A shared pointer to the ChessSet object.
<i>gamePlayState</i>	A reference to the GamePlay state.

5.19.3 Member Function Documentation

5.19.3.1 void [GuiObserver::onGameOver](#) ([GameState](#) state, [PlayerColor](#) winner) [override],[virtual]

Called when a game started with onGameStart is over.

Parameters

<i>state</i>	State on game over.
<i>winner</i>	Winner of the game.

Reimplemented from [AbstractGameObserver](#).

5.19.3.2 void [GuiObserver::onGameStart](#) ([GameState](#) state, [GameConfiguration](#) config) [override],[virtual]

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
<i>config</i>	Valid GameConfiguration for this game.

Reimplemented from [AbstractGameObserver](#).

5.19.3.3 `void GuiObserver::onTurnEnd (PlayerColor who, Turn turn, GameState newState) [override], [virtual]`

Called if a player ended its turn.

Parameters

<i>who</i>	Color of the player doing the turn.
<i>turn</i>	Turn the player decided on.
<i>newState</i>	State after the player performed the turn.

Reimplemented from [AbstractGameObserver](#).

5.19.3.4 `void GuiObserver::onTurnStart (PlayerColor who) [override], [virtual]`

Called if a player is asked to perform a turn.

Parameters

<i>who</i>	Color of the player doing the turn.
------------	-------------------------------------

Reimplemented from [AbstractGameObserver](#).

5.19.3.5 `void GuiObserver::onTurnTimeout (PlayerColor who, std::chrono::seconds timeout) [override], [virtual]`

Called if a players turn is aborted due to timeout.

Parameters

<i>who</i>	Color of the player who got interrupted.
<i>timeout</i>	Length of the time limit that got violated.

Reimplemented from [AbstractGameObserver](#).

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

- S:/dev/3dchess/src/gui/GuiObserver.h
- S:/dev/3dchess/src/gui/GuiObserver.cpp

5.20 GuiWindow Class Reference

```
#include <GuiWindow.h>
```

Public Types

- enum [WindowMode](#) { **FULLSCREEN**, **WINDOW** }
Available window modes.
- enum [fontSize](#) { **HEADLINE** = 42, **SUB_HEADLINE** = 28, **TEXT** = 20, **TEXT_SMALL** = 15 }
The available font sizes.

Public Member Functions

- [GuiWindow](#) (std::string title, bool fullscreen, int width, int height)
Creates a new GUI window.
- void [exec](#) ()
Initiates the window and starts the execution loop.
- int [getWidth](#) ()
Gets the width of the window.
- int [getHeight](#) ()
Gets the height of the window.
- int [getCameraDistanceToOrigin](#) ()
Gets the distance between the camera and the world coordinate origin.
- bool [isFullscreen](#) ()
Checks if the window is currently in fullscreen mode.
- void [set2DMode](#) ()
Set the model view matrix to draw 2D.
- void [set3DMode](#) ()
Set the model view matrix to draw 3D.
- void [swapFrameBufferNow](#) ()
The frame buffer will normally be swapped at the end of the execution loop. If you want to swap it earlier, use this method to force a frame buffer swap immediately.
- void [switchWindowMode](#) ([WindowMode](#) mode)
Switches the window mode to one of the available modes.
- void [printHeadline](#) (std::string text)
Prints the headline text at the top left location.
- void [printSubHeadline](#) (std::string text)
Prints the subheadline text at the top left location directly under the headline.
- void [printTextCenter](#) (float red, float green, float blue, std::string text)
Prints text at the center of the window's viewport.
- void [printText](#) (int x, int y, float red, float green, float blue, std::string text)
Prints text at the given position of the window's viewport with normal text size.
- void [printTextSmall](#) (int x, int y, float red, float green, float blue, std::string text)
Prints text at the given position of the window's viewport with small text size.

Public Attributes

- float [m_cX](#)
Camera position in world coordinates.
- float [m_cY](#)
- float [m_cZ](#)
- float [m_cameraAngleX](#)
Camera angle in degree.
- float [m_cameraAngleY](#)
- float [m_cameraAngleZ](#)
- float [m_fov](#)
field of view (is the extent of the observable world that is seen at any given moment)

5.20.1 Detailed Description

This class is a wrapper which holds the window with the OpenGL context. The [GuiWindow](#) will handle keyboard and mouse events and provides methods to switch OpenGL matrix modes and camera position. The window can also toggle between fullscreen and window mode.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 `GuiWindow::GuiWindow (std::string title, bool fullscreen, int width, int height)`

Creates a new GUI window.

Parameters

<i>title</i>	The title/name of the window which is shown in the top window location.
<i>fullscreen</i>	True to start in fullscreen, false to start in window mode.
<i>width</i>	The width of the window.
<i>height</i>	The height of the window.

5.20.3 Member Function Documentation

5.20.3.1 `int GuiWindow::getCameraDistanceToOrigin ()`

Gets the distance between the camera and the world coordinate origin.

Returns

The distance between camera and world origin.

5.20.3.2 `int GuiWindow::getHeight ()`

Gets the height of the window.

Returns

The height of the window.

5.20.3.3 `int GuiWindow::getWidth ()`

Gets the width of the window.

Returns

The width of the window.

5.20.3.4 `bool GuiWindow::isFullscreen ()`

Checks if the window is currently in fullscreen mode.

Returns

True if the window is in fullscreen mode, false if not.

5.20.3.5 `void GuiWindow::printHeadline (std::string text)`

Prints the headline text at the top left location.

Parameters

<i>text</i>	The text to draw.
-------------	-------------------

5.20.3.6 void GuiWindow::printSubHeadline (std::string *text*)

Prints the subheadline text at the top left location directly under the headline.

Parameters

<i>text</i>	The text to draw.
-------------	-------------------

5.20.3.7 void GuiWindow::printText (int *x*, int *y*, float *red*, float *green*, float *blue*, std::string *text*)

Prints text at the given position of the window's viewport with normal text size.

Parameters

<i>x</i>	The x location in the viewport.
<i>y</i>	The y location in the viewport.
<i>red</i>	The red amount of color between 0.0 and 1.0
<i>green</i>	The green amount of color between 0.0 and 1.0
<i>blue</i>	The blue amount of color between 0.0 and 1.0
<i>text</i>	The text to draw.

Note

The origin of the viewport is the upper left corner.

5.20.3.8 void GuiWindow::printTextCenter (float *red*, float *green*, float *blue*, std::string *text*)

Prints text at the center of the window's viewport.

Parameters

<i>red</i>	The red amount of color between 0.0 and 1.0
<i>green</i>	The green amount of color between 0.0 and 1.0
<i>blue</i>	The blue amount of color between 0.0 and 1.0
<i>text</i>	The text to draw.

5.20.3.9 void GuiWindow::printTextSmall (int *x*, int *y*, float *red*, float *green*, float *blue*, std::string *text*)

Prints text at the given position of the window's viewport with small text size.

Parameters

<i>x</i>	The x location in the viewport.
<i>y</i>	The y location in the viewport.
<i>red</i>	The red amount of color between 0.0 and 1.0
<i>green</i>	The green amount of color between 0.0 and 1.0
<i>blue</i>	The blue amount of color between 0.0 and 1.0

<i>text</i>	The text to draw.
-------------	-------------------

Note

The origin of the viewport is the upper left corner.

5.20.3.10 void GuiWindow::switchWindowMode (WindowMode mode)

Switches the window mode to one of the available modes.

Parameters

<i>mode</i>	A window mode, see above.
-------------	---------------------------

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

- S:/dev/3dchess/src/gui/GuiWindow.h
- S:/dev/3dchess/src/gui/GuiWindow.cpp

5.21 has_toString< T > Struct Template Reference**Public Types**

- enum { **value** = std::is_same<decltype(test<T>(0)), yes>::value }

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

- S:/dev/3dchess/src/misc/helper.h

5.22 IncrementalBoardEvaluator Class Reference

Class for incrementally estimating game state.

```
#include <Evaluators.h>
```

Public Member Functions

- [IncrementalBoardEvaluator](#) ()
Initializes the evaluator for a prestine board.
- [IncrementalBoardEvaluator](#) (const std::array< [Piece](#), 64 > &board)
Initializes the evaluator for an already played board.
- void [moveIncrement](#) (const [Turn](#) &turn)
Updates estimate for the moving of the piece in give turn.
- void [captureIncrement](#) (Field field, const [Piece](#) &piece)
Updates estimate for a capture of the given piece on the given field.
- Score [getScore](#) (PlayerColor color) const
Returns the score from the perspective of the given player color.
- bool **operator==** (const [IncrementalBoardEvaluator](#) &other) const

Static Public Member Functions

- static Score [estimateFullBoard](#) (const std::array< [Piece](#), 64 > &board)
Gives a full estimate for the given board.

5.22.1 Detailed Description

Class for incrementally estimating game state.

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

- S:/dev/3dchess/src/logic/Evaluators.h
- S:/dev/3dchess/src/logic/Evaluators.cpp

5.23 IncrementalZobristHasher< THASH, SEED > Class Template Reference

Zobrist-Hash implementation. Uses mt19937 to initialize from a fixed seed 452134.

```
#include <IncrementalZobristHasher.h>
```

5.23.1 Detailed Description

```
template<typename THASH, uint64_t SEED = 46170>class IncrementalZobristHasher< THASH, SEED >
```

Zobrist-Hash implementation. Uses mt19937 to initialize from a fixed seed 452134.

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

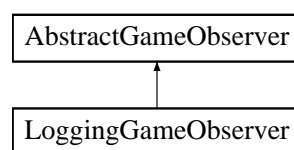
- S:/dev/3dchess/src/logic/IncrementalZobristHasher.h
- S:/dev/3dchess/src/logic/IncrementalZobristHasher.cpp

5.24 LoggingGameObserver Class Reference

[AbstractGameObserver](#) which simply logs occurring events.

```
#include <LoggingGameObserver.h>
```

Inheritance diagram for LoggingGameObserver:



Public Member Functions

- void [onGameStart](#) ([GameState](#) state, [GameConfiguration](#) config) override
Called when the game starts.
- void [onTurnStart](#) (PlayerColor who) override
Called if a player is asked to perform a turn.
- void [onTurnEnd](#) (PlayerColor who, [Turn](#) turn, [GameState](#) newState) override
Called if a player ended its turn.
- void [onTurnTimeout](#) (PlayerColor who, std::chrono::seconds timeout) override
Called if a players turn is aborted due to timeout.
- void [onGameOver](#) ([GameState](#) state, PlayerColor winner) override
Called when a game started with onGameStart is over.

5.24.1 Detailed Description

[AbstractGameObserver](#) which simply logs occuring events.

5.24.2 Member Function Documentation

5.24.2.1 `void LoggingGameObserver::onGameOver (GameState state, PlayerColor winner)` `[override]`,
`[virtual]`

Called when a game started with `onGameStart` is over.

Parameters

<i>state</i>	State on game over.
<i>winner</i>	Winner of the game.

Reimplemented from [AbstractGameObserver](#).

5.24.2.2 `void LoggingGameObserver::onGameStart (GameState state, GameConfiguration config)` `[override]`,
`[virtual]`

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
<i>config</i>	Valid GameConfiguration for this game.

Reimplemented from [AbstractGameObserver](#).

5.24.2.3 `void LoggingGameObserver::onTurnEnd (PlayerColor who, Turn turn, GameState newState)` `[override]`,
`[virtual]`

Called if a player ended its turn.

Parameters

<i>who</i>	Color of the player doing the turn.
<i>turn</i>	Turn the player decided on.
<i>newState</i>	State after the player performed the turn.

Reimplemented from [AbstractGameObserver](#).

5.24.2.4 `void LoggingGameObserver::onTurnStart (PlayerColor who)` `[override]`, `[virtual]`

Called if a player is asked to perform a turn.

Parameters

<i>who</i>	Color of the player doing the turn.
------------	-------------------------------------

Reimplemented from [AbstractGameObserver](#).

5.24.2.5 `void LoggingGameObserver::onTurnTimeout (PlayerColor who, std::chrono::seconds timeout)` `[override]`,
`[virtual]`

Called if a players turn is aborted due to timeout.

Parameters

<i>who</i>	Color of the player who got interrupted.
<i>timeout</i>	Length of the time limit that got violated.

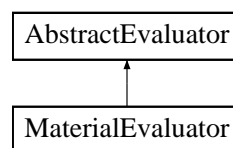
Reimplemented from [AbstractGameObserver](#).

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

- S:/dev/3dchess/src/misc/LoggingGameObserver.h
- S:/dev/3dchess/src/misc/LoggingGameObserver.cpp

5.25 MaterialEvaluator Class Reference

Inheritance diagram for MaterialEvaluator:



Public Member Functions

- virtual Score **getScore** (const [GameState](#) &gameState) const override
- virtual Score **getMaterialWorth** (PlayerColor player, const [ChessBoard](#) &board) const

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

- S:/dev/3dchess/src/logic/Evaluators.h
- S:/dev/3dchess/src/logic/Evaluators.cpp

5.26 Menu2D Class Reference

Public Member Functions

- **Menu2D** (int windowHeight, int windowHeight)
- Menu2DItemPtr **addButton** (std::string filename)
- void **draw** ()
- void **mouseMoved** (const int x, const int y)
- void **mousePressed** ()
- void **mouseReleased** ()
- void **windowResized** (int newWidth, int newHeight)
- void **resetAnimation** ()

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

- S:/dev/3dchess/src/gui/Menu2D.h
- S:/dev/3dchess/src/gui/Menu2D.cpp

5.27 Menu2DItem Class Reference

This class describes a button of a menu. The button is a rectangle with textures depending on one of three states (normal, hover, active). The buttons can be used with the mouse cursor.

```
#include <Menu2DItem.h>
```

Public Member Functions

- [Menu2DItem](#) (std::string filename, int width, int height)
Creates a new menu button item.
- void [setPosition](#) (int x, int y)
Sets the buttons position in viewport coordinates.
- void [draw](#) ()
Draws the button on the previously set position. This method will also consider the button state.
- void [mouseMoved](#) (int x, int y)
Updates the mouse's pointer/cursor position.
- void [mousePressed](#) (int x, int y)
Sets the coordinates, where the mouse clicked in the viewport.
- void [mouseReleased](#) (int x, int y)
Sets the coordinates, where the mouse click was released in the viewport.
- void [onClick](#) (const boost::function< void()> &slot)
Add a function or method which should be called via boost signals when the button is clicked. So the given method can do something.
- void [unClick](#) ()
Remove all click signals.

5.27.1 Detailed Description

This class describes a button of a menu. The button is a rectangle with textures depending on one of three states (normal, hover, active). The buttons can be used with the mouse cursor.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 Menu2DItem::Menu2DItem (std::string filename, int width, int height)

Creates a new menu button item.

Parameters

<i>filename</i>	The filename relative to the executable located in resources/bt_n<filename>.
<i>width</i>	The width of the button.
<i>height</i>	The height of the button.

Note

You must provide a texture for each state in the /resources folder relative to the executable.

For example:

- bt_nBack.png for the normal (no action, simple button) state.
- bt_aBack.png for the active (pressed) state.
- bt_hBack.png for the hover (mouse above the button) state.

Provide the filename without the prefix *bt_n*, *bt_a* and *bt_h*, this is automatically added.

5.27.3 Member Function Documentation

5.27.3.1 void Menu2DItem::draw ()

Draws the button on the previously set position. This method will also consider the button state.

Note

To provide the correct button state, you must update the mouse position. See the methods below.

5.27.3.2 void Menu2DItem::mouseMoved (int x, int y)

Updates the mouse's pointer/cursor position.

Parameters

x	The mouse's x position.
x	The mouse's y position.

Note

You must use this method only if the mouse is moved but the mouse button is neither pressed nor released.

5.27.3.3 void Menu2DItem::mousePressed (int x, int y)

Sets the coordinates, where the mouse clicked in the viewport.

Parameters

x	The mouse's x position.
x	The mouse's y position.

Note

You must use this method only if the mouse was clicked but the mouse button is neither moved nor released.

5.27.3.4 void Menu2DItem::mouseReleased (int x, int y)

Sets the coordinates, where the mouse click was released in the viewport.

Parameters

x	The mouse's x position.
x	The mouse's y position.

Note

You must use this method only if the mouse was released but the mouse button is neither pressed nor the mouse is moved.

5.27.3.5 void Menu2DItem::onClick (const boost::function< void()> & slot)

Add a function or method which should be called via boost signals when the button is clicked. So the given method can do something.

Parameters

<i>slot</i>	The function/method to call when the button is clicked.
-------------	---

5.27.3.6 void Menu2DItem::setPosition (int x, int y)

Sets the buttons position in viewport coordinates.

Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

Note

The origin of the viewport is the upper left corner.

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

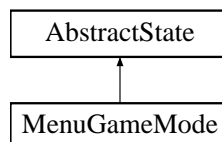
- S:/dev/3dchess/src/gui/Menu2DItem.h
- S:/dev/3dchess/src/gui/Menu2DItem.cpp

5.28 MenuGameMode Class Reference

Class which holds the state GameMode. This state let the user choose one of two modes. The *AI vs. AI* mode which shows a chess match between two artificial computer players where the user can only watch the game. In the *Player vs. AI* mode, the user can play against an artificial computer player.

```
#include <MenuGameMode.h>
```

Inheritance diagram for MenuGameMode:



Public Member Functions

- [MenuGameMode \(\)](#)
Creates a new menu GameMode State object.
- void [enter \(\)](#) override
Enters the state for the first time. This will setup all the state related stuff.
- [AbstractState * run \(\)](#) override
Runs the current state and does all the work.
- void [exit \(\)](#) override
Exits the current state and cleans up all allocated resources.
- void [draw \(\)](#)
Draws something state related stuff on the screen.
- void [onModeAIVsAI \(\)](#)
This method is called if the user chose the AI vs. AI mode.
- void [onModePlayerVsAI \(\)](#)
This method is called if the user chose the Player vs. AI mode.
- void [onMenuBack \(\)](#)
This method is called if the user chose the back button.

5.28.1 Detailed Description

Class which holds the state `GameMode`. This state let the user choose one of two modes. The *AI vs. AI* mode which shows a chess match between two artificial computer players where the user can only watch the game. In the *Player vs. AI* mode, the user can play against an artificial computer player.

Note

To `run()` a state, first `enter()` the state.

5.28.2 Member Function Documentation

5.28.2.1 `void MenuGameMode::enter () [override],[virtual]`

Enters the state for the first time. This will setup all the state related stuff.

Note

To `run()` the current state, first `enter()` it.

Implements [AbstractState](#).

5.28.2.2 `void MenuGameMode::exit () [override],[virtual]`

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implements [AbstractState](#).

5.28.2.3 `AbstractState* MenuGameMode::run () [override],[virtual]`

Runs the current state and does all the work.

Returns

`AbstractState*` the state which should be run after this state. A `nullptr` if the game should be exited.

Implements [AbstractState](#).

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

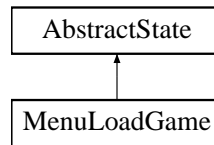
- `S:/dev/3dchess/src/gui/states/MenuGameMode.h`
- `S:/dev/3dchess/src/gui/states/MenuGameMode.cpp`

5.29 MenuLoadGame Class Reference

Class which holds the state `LoadGame`. The user can load a previously saved game from one of three game slots.

```
#include <MenuLoadGame.h>
```

Inheritance diagram for `MenuLoadGame`:



Public Member Functions

- [MenuLoadGame](#) ()
Creates a new menu LoadGame State object.
- void [enter](#) () override
Enters the state for the first time. This will setup all the state related stuff.
- [AbstractState](#) * [run](#) () override
Runs the current state and does all the work.
- void [exit](#) () override
Exits the current state and cleans up all allocated resources.
- void [draw](#) ()
Draws something state related stuff on the screen.
- void [onMenuBack](#) ()
This method is called if the user chose the back button.

5.29.1 Detailed Description

Class which holds the state LoadGame. The user can load a previously saved game from one of three game slots.

Note

To [run\(\)](#) a state, first [enter\(\)](#) the state.

5.29.2 Member Function Documentation

5.29.2.1 void MenuLoadGame::enter () [override],[virtual]

Enters the state for the first time. This will setup all the state related stuff.

Note

To [run\(\)](#) the current state, first [enter\(\)](#) it.

Implements [AbstractState](#).

5.29.2.2 void MenuLoadGame::exit () [override],[virtual]

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implements [AbstractState](#).

5.29.2.3 AbstractState * MenuLoadGame::run () [override],[virtual]

Runs the current state and does all the work.

Returns

AbstractState* the state which should be run after this state. A nullptr if the game should be exited.

Implements [AbstractState](#).

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

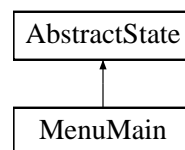
- S:/dev/3dchess/src/gui/states/MenuLoadGame.h
- S:/dev/3dchess/src/gui/states/MenuLoadGame.cpp

5.30 MenuMain Class Reference

Class which holds the state [MenuMain](#). in this menu, the user can start a new game, load a previously saved game, go to the options menu or quit the game.

```
#include <MenuMain.h>
```

Inheritance diagram for MenuMain:



Public Member Functions

- [MenuMain](#) ()
Creates a new menu MainMenu State object.
- void [enter](#) () override
Enters the state for the first time. This will setup all the state related stuff.
- [AbstractState](#) * [run](#) () override
Runs the current state and does all the work.
- void [exit](#) () override
Exits the current state and cleans up all allocated resources.
- void [draw](#) ()
Draws something state related stuff on the screen.
- void [onNewGame](#) ()
This method is called if the user chose to play a new game.
- void [onLoadGame](#) ()
This method is called if the user chose to load a game.
- void [onOptions](#) ()
This method is called if the user chose to go to the options menu.
- void [onExitGame](#) ()
This method is called if the user chose to exit the game.

5.30.1 Detailed Description

Class which holds the state [MenuMain](#). In this menu, the user can start a new game, load a previously saved game, go to the options menu or quit the game.

Note

To [run\(\)](#) a state, first [enter\(\)](#) the state.

5.30.2 Member Function Documentation

5.30.2.1 `void MenuMain::enter () [override], [virtual]`

Enters the state for the first time. This will setup all the state related stuff.

Note

To [run\(\)](#) the current state, first [enter\(\)](#) it.

Implements [AbstractState](#).

5.30.2.2 `void MenuMain::exit () [override], [virtual]`

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implements [AbstractState](#).

5.30.2.3 `AbstractState * MenuMain::run () [override], [virtual]`

Runs the current state and does all the work.

Returns

`AbstractState*` the state which should be run after this state. A `nullptr` if the game should be exited.

Implements [AbstractState](#).

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

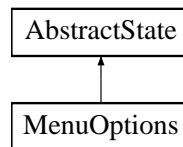
- `S:/dev/3dchess/src/gui/states/MenuMain.h`
- `S:/dev/3dchess/src/gui/states/MenuMain.cpp`

5.31 MenuOptions Class Reference

Class which holds the state `MenuOption`. This state let the user toggle between the fullscreen view or the windowed mode of the game.

```
#include <MenuOptions.h>
```

Inheritance diagram for `MenuOptions`:



Public Member Functions

- [MenuOptions](#) ()
Creates a new menu Options State object.
- void [enter](#) () override
Enters the state for the first time. This will setup all the state related stuff.
- [AbstractState](#) * [run](#) () override
Runs the current state and does all the work.
- void [exit](#) () override
Exits the current state and cleans up all allocated resources.
- void [draw](#) ()
Draws something state related stuff on the screen.
- void [onResolutionChange](#) ()
This method is called if the user chose to change the resolution.
- void [onMenuBack](#) ()
This method is called if the user chose to go back to the menu he was, before he get here.

5.31.1 Detailed Description

Class which holds the state MenuOption. This state let the user toggle between the fullscreen view or the windowed mode of the game.

Note

To [run\(\)](#) a state, first [enter\(\)](#) the state.

5.31.2 Member Function Documentation

5.31.2.1 void MenuOptions::enter () [override],[virtual]

Enters the state for the first time. This will setup all the state related stuff.

Note

To [run\(\)](#) the current state, first [enter\(\)](#) it.

Implements [AbstractState](#).

5.31.2.2 void MenuOptions::exit () [override],[virtual]

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implements [AbstractState](#).

5.31.2.3 AbstractState * MenuOptions::run () [override],[virtual]

Runs the current state and does all the work.

Returns

AbstractState* the state which should be run after this state. A nullptr if the game should be exited.

Implements [AbstractState](#).

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

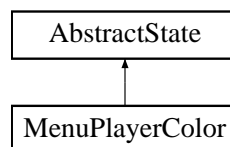
- S:/dev/3dchess/src/gui/states/MenuOptions.h
- S:/dev/3dchess/src/gui/states/MenuOptions.cpp

5.32 MenuPlayerColor Class Reference

Class which holds the state PlayerColor. This state let the user choose between black or white for the chess model figures.

```
#include <MenuPlayerColor.h>
```

Inheritance diagram for MenuPlayerColor:



Public Member Functions

- [MenuPlayerColor](#) ()
Creates a new menu PlayerColor State object.
- void [enter](#) () override
Enters the state for the first time. This will setup all the state related stuff.
- [AbstractState](#) * [run](#) () override
Runs the current state and does all the work.
- void [exit](#) () override
Exits the current state and cleans up all allocated resources.
- void [draw](#) ()
Draws something state related stuff on the screen.
- void [onColorWhite](#) ()
This method is called if the user chose the white color as player color.
- void [onColorBlack](#) ()
This method is called if the user chose the black color as player color.
- void [onMenuBack](#) ()
This method is called if the user chose to go back to the menu he was, before he get here.

5.32.1 Detailed Description

Class which holds the state PlayerColor. This state let the user choose between black or white for the chess model figures.

Note

To `run()` a state, first `enter()` the state.

5.32.2 Member Function Documentation

5.32.2.1 `void MenuPlayerColor::enter () [override],[virtual]`

Enters the state for the first time. This will setup all the state related stuff.

Note

To `run()` the current state, first `enter()` it.

Implements [AbstractState](#).

5.32.2.2 `void MenuPlayerColor::exit () [override],[virtual]`

Exits the current state and cleans up all allocated resources.

Note

This is the last method to call, before the object is deleted.

Implements [AbstractState](#).

5.32.2.3 `AbstractState * MenuPlayerColor::run () [override],[virtual]`

Runs the current state and does all the work.

Returns

`AbstractState*` the state which should be run after this state. A `nullptr` if the game should be exited.

Implements [AbstractState](#).

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

- `S:/dev/3dchess/src/gui/states/MenuPlayerColor.h`
- `S:/dev/3dchess/src/gui/states/MenuPlayerColor.cpp`

5.33 Mesh Class Reference

Wrapper class for the Assimp library.

```
#include <Mesh.h>
```

Public Member Functions

- [Mesh](#) (unsigned int `numVertices`, `aiVector3D *vertices`, `aiVector3D *normals`, unsigned int `numFaces`, `aiFace *faces`)

Creates a new [Mesh](#) object.

Public Attributes

- `aiVector3D * vertices`
The model's vertices.
- `aiVector3D * normals`
The model's normals.
- `aiVector3D * textureCoords`
The model's texture coordinates.
- `GLuint * indices`
The model's indices.
- `GLuint numVertices`
The number of vertices.
- `GLuint numIndices`
The number of indices.

5.33.1 Detailed Description

Wrapper class for the Assimp library.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 `Mesh::Mesh (unsigned int numVertices, aiVector3D * vertices, aiVector3D * normals, unsigned int numFaces, aiFace * faces)`

Creates a new `Mesh` object.

Parameters

<i>numVertices</i>	The number of model vertices.
<i>vertices</i>	The model's vertices itself.
<i>normals</i>	The model's normals itself.
<i>numFaces</i>	The number of model faces.
<i>faces</i>	The model's faces itself.

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

- `S:/dev/3dchess/src/gui/Mesh.h`
- `S:/dev/3dchess/src/gui/Mesh.cpp`

5.34 Model Class Reference

Representing a chess figure (e.g. King, Queen, ...).

```
#include <Model.h>
```

Public Types

- enum `Color` { `BLACK`, `WHITE` }
Possible model colors.

Public Member Functions

- [Model](#) (std::string file)
Loads the model from the filesystem.
- void [loadScene](#) ()
Imports the model from filesystem.
- void [setCorrectionValues](#) (int localX, int localY, int localZ, float scaleFactor, int rotateX, int rotateY, int rotateZ)
Corrects the model positioning if the model (in the given file) is not proper located at 0/0/0 local space coordinates, the rotation or the scaling factor is wrong.
- void [setColor](#) ([Color](#) color)
Sets the models color.
- void [setPosition](#) (int globalX, int globalY, int globalZ)
Sets a new global position (world coordinates) for the model.
- void [draw](#) ()
Draws the model at configured world coordinates.

5.34.1 Detailed Description

Representing a chess figure (e.g. King, Queen, ...).

5.34.2 Constructor & Destructor Documentation

5.34.2.1 Model::Model (std::string file)

Loads the model from the filesystem.

Parameters

<i>file</i>	The filename with directory relative to the game's executable file.
-------------	---

5.34.3 Member Function Documentation

5.34.3.1 void Model::setColor ([Color](#) color)

Sets the models color.

Parameters

<i>color</i>	The color of the model.
--------------	-------------------------

5.34.3.2 void Model::setCorrectionValues (int *localX*, int *localY*, int *localZ*, float *scaleFactor*, int *rotateX*, int *rotateY*, int *rotateZ*)

Corrects the model positioning if the model (in the given file) is not proper located at 0/0/0 local space coordinates, the rotation or the scaling factor is wrong.

Note

This should only be used if there's no proper model file available.

Parameters

<i>localX</i>	Sets the local x coordinate.
<i>localY</i>	Sets the local y coordinate.
<i>localZ</i>	Sets the local z coordinate.
<i>scaleFactor</i>	The scaling factor to shrink or enlarge.
<i>rotateX</i>	The rotation in degree along the x axis.
<i>rotateY</i>	The rotation in degree along the y axis.
<i>rotateZ</i>	The rotation in degree along the z axis.

5.34.3.3 void Model::setPosition (int *globalX*, int *globalY*, int *globalZ*)

Sets a new global position (world coordinates) for the model.

Parameters

<i>globalX</i>	The global x coordinate.
<i>globalY</i>	The global y coordinate.
<i>globalZ</i>	The global z coordinate.

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

- S:/dev/3dchess/src/gui/Model.h
- S:/dev/3dchess/src/gui/Model.cpp

5.35 Negamax Class Reference

Implementation of a [Negamax](#) algorithm.

```
#include <Negamax.h>
```

Classes

- struct [PerfCounters](#)
Structure with performance counters used for debugging and evaluation.
- struct [Result](#)
Structure for holding search results.

Public Member Functions

- [Negamax](#) ()
Creates a new algorithm instance.
- template<typename TGameState = GameState, bool AB_CUTOFF_ENABLED = true, bool MOVE_ORDERING_ENABLED = true>
[Result search](#) (const TGameState &state, size_t maxDepth)
Search given state up to maxDepth full turns.

Public Attributes

- struct [Negamax::PerfCounters](#) **m_counters**

5.35.1 Detailed Description

Implementation of a [Negamax](#) algorithm.

5.35.2 Member Function Documentation

5.35.2.1 `template<typename TGameState = GameState, bool AB_CUTOFF_ENABLED = true, bool MOVE_ORDERING_ENABLED = true> Result Negamax::search (const TGameState & state, size_t maxDepth) [inline]`

Search given state up to maxDepth full turns.

Template Parameters

<i>TGameState</i>	Type of game state so GameState is mockable.
<i>AB_CUTOFF_ENABLE</i>	If false Alpha-Beta cutoff feature is disabled.
<i>MOVE_ORDERING_ENABLED</i>	If false move ordering is disabled.

Parameters

<i>state</i>	Game state to search.
<i>maxDepthInTurns</i>	Number of full turns (ply and return ply) to search.

Returns

[Result](#) of the search.

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

- S:/dev/3dchess/src/ai/Negamax.h

5.36 ObjectHelper Class Reference

Helper class for creating static OpenGL display lists to boost the drawing of OpenGL objects.

```
#include <ObjectHelper.h>
```

Static Public Member Functions

- static GLuint [createCubeList](#) (float size, float x, float y, float z)
Creates a new OpenGL display list for a cube.
- static GLuint [create2DRectList](#) (float width, float height, float viewportX, float viewportY, float colorR, float colorG, float colorB)
Creates a new OpenGL display list for a 2D rectangle box.
- static GLuint [create2DGradientRectList](#) (float width, float height, float viewportX, float viewportY, float fromColorR, float fromColorG, float fromColorB, float toColorR, float toColorG, float toColorB)
Creates a new OpenGL display list for a 2D rectangle box with gradient color from top to bottom.

5.36.1 Detailed Description

Helper class for creating static OpenGL display lists to boost the drawing of OpenGL objects.

Note

See <http://www.opengl.org/documentation/specs/version1.1/glspec1.1/node123.-html> for more details about display lists.

5.36.2 Member Function Documentation

5.36.2.1 GLuint ObjectHelper::create2DGradientRectList (float *width*, float *height*, float *viewportX*, float *viewportY*, float *fromColorR*, float *fromColorG*, float *fromColorB*, float *toColorR*, float *toColorG*, float *toColorB*) [static]

Creates a new OpenGL display list for a 2D rectangle box with gradient color from top to bottom.

Parameters

<i>width</i>	The width of the rectangle.
<i>height</i>	The height of the rectangle.
<i>viewportX</i>	The viewport x coordinate from the left top corner.
<i>viewportY</i>	The viewport y coordinate from the left top corner.
<i>fromColorR</i>	The red color value between 0.0 and 1.0 at the top edge of the rectangle.
<i>fromColorG</i>	The green color value between 0.0 and 1.0 at the top edge of the rectangle.
<i>fromColorB</i>	The blue color value between 0.0 and 1.0 at the top edge of the rectangle.
<i>toColorR</i>	The red color value between 0.0 and 1.0 at the bottom edge of the rectangle.
<i>toColorG</i>	The green color value between 0.0 and 1.0 at the bottom edge of the rectangle.
<i>toColorB</i>	The blue color value between 0.0 and 1.0 at the bottom edge of the rectangle.

Returns

GLuint A display list index which holds the compiled rectangle.

5.36.2.2 GLuint ObjectHelper::create2DRectList (float *width*, float *height*, float *viewportX*, float *viewportY*, float *colorR*, float *colorG*, float *colorB*) [static]

Creates a new OpenGL display list for a 2D rectangle box.

Parameters

<i>width</i>	The width of the rectangle.
<i>height</i>	The height of the rectangle.
<i>viewportX</i>	The viewport x coordinate from the left top corner.
<i>viewportY</i>	The viewport y coordinate from the left top corner.
<i>colorR</i>	The red color value between 0.0 and 1.0.
<i>colorG</i>	The green color value between 0.0 and 1.0.
<i>colorB</i>	The blue color value between 0.0 and 1.0.

Returns

GLuint A display list index which holds the compiled rectangle.

5.36.2.3 GLuint ObjectHelper::createCubeList (float *size*, float *x*, float *y*, float *z*) [static]

Creates a new OpenGL display list for a cube.

Parameters

<i>size</i>	The size of an edge of the cube.
<i>x</i>	The position of the cube in x world coordinate.
<i>y</i>	The position of the cube in y world coordinate.
<i>z</i>	The position of the cube in z world coordinate.

Returns

GLuint A display list index which holds the compiled cube.

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

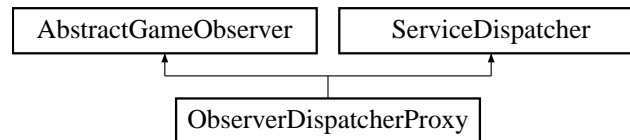
- S:/dev/3dchess/src/gui/ObjectHelper.h
- S:/dev/3dchess/src/gui/ObjectHelper.cpp

5.37 ObserverDispatcherProxy Class Reference

Proxy for transporting [AbstractGameObserver](#) events between threads.

```
#include <ObserverDispatcherProxy.h>
```

Inheritance diagram for ObserverDispatcherProxy:



Public Member Functions

- **ObserverDispatcherProxy** (AbstractGameObserverPtr observer)
- virtual void **onGameStart** (GameState state, GameConfiguration config) override
Called when the game starts.
- virtual void **onTurnStart** (PlayerColor who) override
Called if a player is asked to perform a turn.
- virtual void **onTurnEnd** (PlayerColor who, Turn turn, GameState newState) override
Called if a player ended its turn.
- virtual void **onTurnTimeout** (PlayerColor who, std::chrono::seconds timeout) override
Called if a player's turn is aborted due to timeout.
- virtual void **onGameOver** (GameState state, PlayerColor winner) override
Called when a game started with onGameStart is over.

Additional Inherited Members

5.37.1 Detailed Description

Proxy for transporting [AbstractGameObserver](#) events between threads.

As the [GameLogic](#) and other game components run on different threads it is essential to safely transport game events between them. Without any additional precautions [AbstractGameObserver](#) implementations will have their handlers called on the [GameLogic](#) thread they are registered on with all implied thread safety concerns.

This proxy will serialize calls coming in from the game logic in a thread-safe way and replay them once its poll method is called in the customers thread.

5.37.2 Member Function Documentation

5.37.2.1 virtual void ObserverDispatcherProxy::onGameOver (GameState state, PlayerColor winner) [inline],
[override], [virtual]

Called when a game started with onGameStart is over.

Parameters

<i>state</i>	State on game over.
--------------	---------------------

<i>winner</i>	Winner of the game.
---------------	---------------------

Reimplemented from [AbstractGameObserver](#).

5.37.2.2 `virtual void ObserverDispatcherProxy::onGameStart (GameState state, GameConfiguration config)`
`[inline], [override], [virtual]`

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
<i>config</i>	Valid GameConfiguration for this game.

Reimplemented from [AbstractGameObserver](#).

5.37.2.3 `virtual void ObserverDispatcherProxy::onTurnEnd (PlayerColor who, Turn turn, GameState newState)`
`[inline], [override], [virtual]`

Called if a player ended its turn.

Parameters

<i>who</i>	Color of the player doing the turn.
<i>turn</i>	Turn the player decided on.
<i>newState</i>	State after the player performed the turn.

Reimplemented from [AbstractGameObserver](#).

5.37.2.4 `virtual void ObserverDispatcherProxy::onTurnStart (PlayerColor who)` `[inline], [override], [virtual]`

Called if a player is asked to perform a turn.

Parameters

<i>who</i>	Color of the player doing the turn.
------------	-------------------------------------

Reimplemented from [AbstractGameObserver](#).

5.37.2.5 `virtual void ObserverDispatcherProxy::onTurnTimeout (PlayerColor who, std::chrono::seconds timeout)`
`[inline], [override], [virtual]`

Called if a players turn is aborted due to timeout.

Parameters

<i>who</i>	Color of the player who got interrupted.
<i>timeout</i>	Length of the time limit that got violated.

Reimplemented from [AbstractGameObserver](#).

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

- `S:/dev/3dchess/src/logic/threading/ObserverDispatcherProxy.h`

5.38 Negamax::PerfCounters Struct Reference

Structure with performance counters used for debugging and evaluation.

```
#include <Negamax.h>
```

Public Member Functions

- `std::string toString ()` const

Public Attributes

- `uint64_t nodes`
Number of nodes searched.
- `uint64_t cutoffs`
Number of branches cut-off using Alpha-Beta.
- `uint64_t updates`
Number of best result updates during search.
- `std::chrono::milliseconds duration`
Time taken for last search.

5.38.1 Detailed Description

Structure with performance counters used for debugging and evaluation.

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

- `S:/dev/3dchess/src/ai/Negamax.h`

5.39 Piece Struct Reference

Public Member Functions

- **Piece** (PlayerColor player, PieceType pieceType)
- `bool operator==` (const [Piece](#) &other) const

Public Attributes

- PlayerColor **player**
- PieceType **type**

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

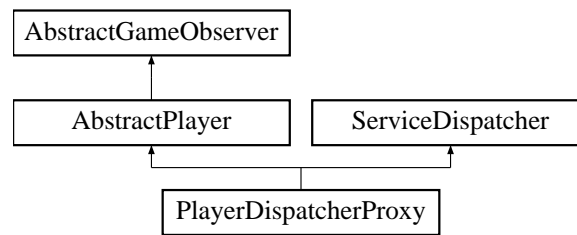
- `S:/dev/3dchess/src/logic/ChessTypes.h`
- `S:/dev/3dchess/src/logic/ChessTypes.cpp`

5.40 PlayerDispatcherProxy Class Reference

Proxy for transporting AbstractGamePlayer events between threads.

```
#include <PlayerDispatcherProxy.h>
```

Inheritance diagram for PlayerDispatcherProxy:



Public Member Functions

- **PlayerDispatcherProxy** (AbstractPlayerPtr player)
- virtual void **onSetColor** (PlayerColor color) override
Notifies that player what color he will be playing. Called before onGameStart.
- virtual std::future< Turn > **doMakeTurn** (GameState state) override
Asks the player to make his turn.
- virtual void **doAbortTurn** () override
Asks the player to abort a turn asked for with doMakeTurn. When this is called the GameLogic will no longer react to the completion of the future for that turn. A use of this function is the abortion of a turn due to timeout.
- virtual void **onGameStart** (GameState state, GameConfiguration config) override
Called when the game starts.
- virtual void **onTurnStart** (PlayerColor who) override
Called if a player is asked to perform a turn.
- virtual void **onTurnEnd** (PlayerColor who, Turn turn, GameState newState) override
Called if a player ended its turn.
- virtual void **onTurnTimeout** (PlayerColor who, std::chrono::seconds timeout) override
Called if a players turn is aborted due to timeout.
- virtual void **onGameOver** (GameState state, PlayerColor winner) override
Called when a game started with onGameStart is over.

Additional Inherited Members

5.40.1 Detailed Description

Proxy for transporting AbstractGamePlayer events between threads.

As the GameLogic and other game components run on different threads it is essential to safely transport game events between them. Without any additional precautions AbstractGamePlayer implementations will have their handlers called on the GameLogic thread they are registered on with all implied thread safety concerns.

This proxy will serialize calls coming in from the game logic in a thread-safe way and replay them once its poll method is called in the customers thread.

5.40.2 Member Function Documentation

5.40.2.1 virtual std::future<Turn> PlayerDispatcherProxy::doMakeTurn (GameState state) [inline],
[override], [virtual]

Asks the player to make his turn.

Warning

This function must not block. It is to return immediatly. The players turn is to be set on the returned future.

Note

The game logic can abort its request for a player to make his turn using the `doAbortTurn` function at any time.

Parameters

<i>state</i>	Current state of the game.
--------------	----------------------------

Returns

A future to the turn to make.

Implements [AbstractPlayer](#).

5.40.2.2 `virtual void PlayerDispatcherProxy::onGameOver (GameState state, PlayerColor winner) [inline], [override], [virtual]`

Called when a game started with `onGameStart` is over.

Parameters

<i>state</i>	State on game over.
<i>winner</i>	Winner of the game.

Reimplemented from [AbstractGameObserver](#).

5.40.2.3 `virtual void PlayerDispatcherProxy::onGameStart (GameState state, GameConfiguration config) [inline], [override], [virtual]`

Called when the game starts.

Parameters

<i>state</i>	GameState on game start.
<i>config</i>	Valid GameConfiguration for this game.

Reimplemented from [AbstractGameObserver](#).

5.40.2.4 `virtual void PlayerDispatcherProxy::onSetColor (PlayerColor color) [inline], [override], [virtual]`

Notifies that player what color he will be playing. Called before `onGameStart`.

Parameters

<i>color</i>	Color the player has.
--------------	-----------------------

Implements [AbstractPlayer](#).

5.40.2.5 `virtual void PlayerDispatcherProxy::onTurnEnd (PlayerColor who, Turn turn, GameState newState) [inline], [override], [virtual]`

Called if a player ended its turn.

Parameters

<i>who</i>	Color of the player doing the turn.
------------	-------------------------------------

<i>turn</i>	Turn the player decided on.
<i>newState</i>	State after the player performed the turn.

Reimplemented from [AbstractGameObserver](#).

5.40.2.6 `virtual void PlayerDispatcherProxy::onTurnStart (PlayerColor who)` `[inline],[override],[virtual]`

Called if a player is asked to perform a turn.

Parameters

<i>who</i>	Color of the player doing the turn.
------------	-------------------------------------

Reimplemented from [AbstractGameObserver](#).

5.40.2.7 `virtual void PlayerDispatcherProxy::onTurnTimeout (PlayerColor who, std::chrono::seconds timeout)`
`[inline],[override],[virtual]`

Called if a players turn is aborted due to timeout.

Parameters

<i>who</i>	Color of the player who got interrupted.
<i>timeout</i>	Length of the time limit that got violated.

Reimplemented from [AbstractGameObserver](#).

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

- `S:/dev/3dchess/src/logic/threading/PlayerDispatcherProxy.h`

5.41 PoF Struct Reference

Public Member Functions

- **PoF** ([Piece](#) piece, Field field)

Public Attributes

- [Piece](#) **piece**
- Field **field**

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

- `S:/dev/3dchess/src/logic/ChessBoard.h`

5.42 Negamax::Result Struct Reference

Structure for holding search results.

```
#include <Negamax.h>
```

Public Member Functions

- `Result operator-` () const
Negates score. Syntax sugar to get closer to algorithm notation.
- bool `operator<` (const `Result` &other)
- bool `operator<=` (const `Result` &other)
- bool `operator>=` (const `Result` &other)
- bool `operator>` (const `Result` &other)
- bool `operator==` (const `Result` &other) const
- `std::string toString` () const

Public Attributes

- Score `score`
Evaluator score estimation for this turn.
- `boost::optional< Turn > turn`
Turn to make to advance towards score.

5.42.1 Detailed Description

Structure for holding search results.

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

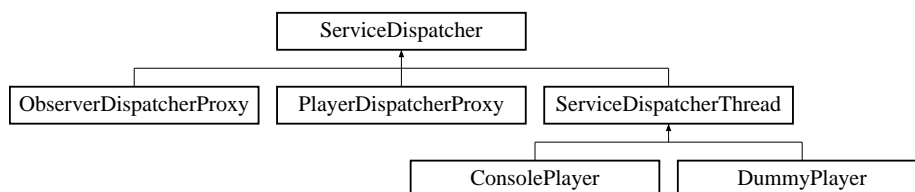
- `S:/dev/3dchess/src/ai/Negamax.h`

5.43 ServiceDispatcher Class Reference

Provides functionality for safely running operations in a thread.

```
#include <ServiceDispatcher.h>
```

Inheritance diagram for ServiceDispatcher:



Public Member Functions

- void `poll` ()
Replays all posted functions in the calling thread.

Protected Member Functions

- `template<typename Function >`
void `post` (Function &&function)
Store a given function.

- `template<typename Function >`
`auto postPromise (Function &&function) -> decltype(std::promise< typename std::result_of< Function()>::type >().get_future())`
Stores a given function and returns a future on its return value.
- `void run ()`
Runs underlying boost asio io_service.
- `void resetWork ()`
Drops queued functions.
- `void stopService ()`
Stops underlying service.

5.43.1 Detailed Description

Provides functionality for safely running operations in a thread.

For components running on different threads it is essential to safely transport events between them. Without any additional precautions functions will execute on the thread they are called.

This dispatcher can store functions in a thread-safe way and replay them once its poll method is called in the customers thread.

5.43.2 Member Function Documentation

5.43.2.1 `void ServiceDispatcher::poll ()` `[inline]`

Replays all posted functions in the calling thread.

See Also

[post](#)
[postPromise](#)

5.43.2.2 `template<typename Function > void ServiceDispatcher::post (Function && function)` `[inline]`, `[protected]`

Store a given function.

Parameters

<i>function</i>	Function to store and later replay.
-----------------	-------------------------------------

5.43.2.3 `template<typename Function > auto ServiceDispatcher::postPromise (Function && function) ->` `decltype(std::promise<typename std::result_of<Function()>::type>().get_future())` `[inline]`, `[protected]`

Stores a given function and returns a future on its return value.

Parameters

<i>function</i>	Function to store and later replay.
-----------------	-------------------------------------

Returns

Future on result of given function.

5.43.2.4 void ServiceDispatcher::run () [inline],[protected]

Runs underlying boost asio io_service.

Note

Will block until work is completed.

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

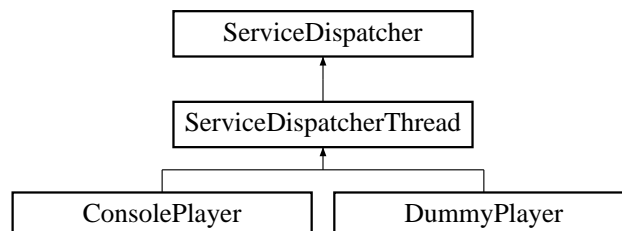
- S:/dev/3dchess/src/logic/threading/ServiceDispatcher.h

5.44 ServiceDispatcherThread Class Reference

Provides functionality for safely running operations in a thread.

```
#include <ServiceDispatcherThread.h>
```

Inheritance diagram for ServiceDispatcherThread:



Public Member Functions

- [ServiceDispatcherThread](#) ()
Creates a [ServiceDispatcherThread](#).
- virtual [~ServiceDispatcherThread](#) ()
Destroy dispatcher. Stops internal thread and discards all remaining calls.
- virtual void [start](#) ()
Start the dispatcher thread.
- virtual void [stop](#) (bool force=false)
Stops the execution of this tread.

Public Attributes

- std::thread [m_thread](#)
Thread this object is running its event loop on after start.

Additional Inherited Members

5.44.1 Detailed Description

Provides functionality for safely running operations in a thread.

Uses [ServiceDispatcher](#) to move function calls into its own thread and execute them.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 ServiceDispatcherThread::ServiceDispatcherThread () [inline]

Creates a [ServiceDispatcherThread](#).

Note

Don't forget to [start\(\)](#) it.

5.44.3 Member Function Documentation

5.44.3.1 virtual void ServiceDispatcherThread::stop (bool *force* = false) [inline], [virtual]

Stops the execution of this tread.

Parameters

<i>force</i>	If true remaining calls are dropped. Otherwise shutdown is deferred until all calls currently in the queue are processed.
--------------	---

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

- S:/dev/3dchess/src/logic/threading/ServiceDispatcherThread.h

5.45 StateMachine Class Reference

Class which manages the states.

```
#include <StateMachine.h>
```

Classes

- struct [EventMap](#)
Structure for holding user events.

Public Member Functions

- void [setStartState](#) ([AbstractState](#) *startState)
Sets the start state and setup the state.
- [AbstractState](#) * [run](#) ()
Runs the current state.
- void [setNextState](#) ([AbstractState](#) *state)
Sets the next state which should be run.

Static Public Member Functions

- static [StateMachine](#) & [getInstance](#) ()
Gets an instance of the [StateMachine](#).

Public Attributes

- struct [StateMachine::EventMap](#) **eventmap**
- [GuiWindow](#) * **window**

Holds the pointer to the [GuiWindow](#) object, to access gui related methods.

5.45.1 Detailed Description

Class which manages the states.

Note

This is a singleton, you can get only one instance of the [StateMachine](#). Don't forget to update the events if they occur.

5.45.2 Member Function Documentation

5.45.2.1 static StateMachine& StateMachine::getInstance () [inline],[static]

Gets an instance of the [StateMachine](#).

Note

This is a singleton. So you can only get one instance.

Returns

[StateMachine](#)& A reference to the [StateMachine](#).

5.45.2.2 AbstractState * StateMachine::run ()

Runs the current state.

Returns

The [AbstractState](#) pointer to the state which must be [run\(\)](#) the next time.

5.45.2.3 void StateMachine::setNextState (AbstractState * state)

Sets the next state which should be run.

Parameters

<i>state</i>	The next state.
--------------	-----------------

5.45.2.4 void StateMachine::setStartState (AbstractState * startState)

Sets the start state and setup the state.

Parameters

<i>startState</i>	The start state.
-------------------	------------------

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

- S:/dev/3dchess/src/gui/StateMachine.h
- S:/dev/3dchess/src/gui/StateMachine.cpp

5.46 Turn Class Reference

Public Types

- enum **Action** { **Move**, **Castle**, **Forfeit**, **Pass** }

Public Member Functions

- **Turn** ([Piece](#) piece, Field from, Field to, Action action)
- bool **operator==** (const [Turn](#) &other) const
- bool **operator!=** (const [Turn](#) &other) const
- std::string **toString** () const

Static Public Member Functions

- static [Turn](#) **move** ([Piece](#) piece, Field from, Field to)

Public Attributes

- [Piece](#) **piece**
- Field **from**
- Field **to**
- enum [Turn::Action](#) **action**

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

- S:/dev/3dchess/src/logic/Turn.h
- S:/dev/3dchess/src/logic/Turn.cpp

5.47 TurnGenerator Class Reference

Public Member Functions

- virtual std::vector< [Turn](#) > **generateTurns** (PlayerColor player, const [ChessBoard](#) &cb) const
- virtual BitBoard **calcTurns** ([Piece](#) piece, BitBoard bbPiece, const [ChessBoard](#) &cb) const
- virtual std::vector< [Turn](#) > **bitBoardToTurns** ([Piece](#) piece, BitBoard bbPiece, BitBoard bbTurns) const
- virtual BitBoard **calcKingTurns** (BitBoard king, BitBoard allOwnPieces) const
- virtual BitBoard **calcKnightTurns** (BitBoard knights, BitBoard allOwnPieces) const
- virtual BitBoard **calcPawnTurns** (BitBoard pawns, BitBoard allOppPieces, BitBoard allPieces, PlayerColor player) const
- virtual BitBoard **calcQueenTurns** (BitBoard queens, BitBoard allOppPieces, BitBoard allPieces) const
- virtual BitBoard **calcBishopTurns** (BitBoard bishops, BitBoard allOppPieces, BitBoard allPieces) const
- virtual BitBoard **calcRookTurns** (BitBoard rooks, BitBoard allOppPieces, BitBoard allPieces) const
- virtual BitBoard **maskRank** (Rank rank) const

- virtual BitBoard **clearRank** (Rank rank) const
- virtual BitBoard **maskFile** (File file) const
- virtual BitBoard **clearFile** (File file) const
- virtual BitBoard **getBitsE** (BitBoard bbPiece) const
- virtual BitBoard **getBitsW** (BitBoard bbPiece) const
- virtual BitBoard **getBitsN** (BitBoard bbPiece) const
- virtual BitBoard **getBitsS** (BitBoard bbPiece) const
- virtual BitBoard **getBitsNE** (BitBoard bbPiece) const
- virtual BitBoard **getBitsNW** (BitBoard bbPiece) const
- virtual BitBoard **getBitsSE** (BitBoard bbPiece) const
- virtual BitBoard **getBitsSW** (BitBoard bbPiece) const

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

- S:/dev/3dchess/src/logic/TurnGenerator.h
- S:/dev/3dchess/src/logic/TurnGenerator.cpp

Index

- AIPlayer, [16](#)
 - AIPlayer, [16](#)
 - AIPlayer, [16](#)
 - doMakeTurn, [17](#)
 - getState, [17](#)
 - onGameOver, [17](#)
 - onGameStart, [17](#)
 - onSetColor, [17](#)
- AbstractEvaluator, [9](#)
- AbstractGameLogic, [9](#)
 - addObserver, [10](#)
 - getConfiguration, [10](#)
 - getWinner, [10](#)
 - isGameOver, [10](#)
 - start, [11](#)
- AbstractGameObserver, [11](#)
 - onGameOver, [12](#)
 - onGameStart, [12](#)
 - onTurnEnd, [12](#)
 - onTurnStart, [12](#)
 - onTurnTimeout, [13](#)
- AbstractPlayer, [13](#)
 - doMakeTurn, [14](#)
 - onSetColor, [14](#)
- AbstractState, [14](#)
 - enter, [15](#)
 - exit, [15](#)
 - run, [15](#)
- addObserver
 - AbstractGameLogic, [10](#)
 - GameLogic, [33](#)
- AnimationHelper
 - EASE_LINEAR, [18](#)
 - EASE_OUTSINE, [18](#)
- AnimationHelper, [18](#)
 - AnimationHelper, [19](#)
 - AnimationHelper, [19](#)
 - ease, [20](#)
 - FunctionType, [18](#)
 - hasStopped, [20](#)
- AssimpHelper, [20](#)
 - importScene, [21](#)
- ChessBoard, [22](#)
- ChessSet, [23](#)
 - getResourcesCount, [23](#)
 - loadResources, [23](#)
 - registerLoadCallback, [23](#)
 - setState, [25](#)
- ConsolePlayer, [25](#)
 - doMakeTurn, [26](#)
 - onGameOver, [26](#)
 - onGameStart, [26](#)
 - onSetColor, [27](#)
 - onTurnEnd, [27](#)
- create2DGradientRectList
 - ObjectHelper, [60](#)
- create2DRectList
 - ObjectHelper, [61](#)
- createCubeList
 - ObjectHelper, [61](#)
- DebugTools, [7](#)
- doMakeTurn
 - AbstractPlayer, [14](#)
 - AIPlayer, [17](#)
 - ConsolePlayer, [26](#)
 - DummyPlayer, [28](#)
 - PlayerDispatcherProxy, [65](#)
- draw
 - Menu2DItem, [47](#)
- DummyPlayer, [27](#)
 - doMakeTurn, [28](#)
 - onSetColor, [28](#)
- EASE_LINEAR
 - AnimationHelper, [18](#)
- EASE_OUTSINE
 - AnimationHelper, [18](#)
- ease
 - AnimationHelper, [20](#)
- enter
 - AbstractState, [15](#)
 - GamePlay, [35](#)
 - MenuGameMode, [49](#)
 - MenuLoadGame, [50](#)
 - MenuMain, [52](#)
 - MenuOptions, [53](#)
 - MenuPlayerColor, [55](#)
- exit
 - AbstractState, [15](#)
 - GamePlay, [35](#)
 - MenuGameMode, [49](#)
 - MenuLoadGame, [50](#)
 - MenuMain, [52](#)
 - MenuOptions, [53](#)
 - MenuPlayerColor, [55](#)
- freetype, [7](#)
 - next_p2, [8](#)

- pop_projection_matrix, 8
 - print, 8
 - pushScreenCoordinateMatrix, 8
- freetype::font_data, 29
- FunctionType
 - AnimationHelper, 18
- GameConfiguration, 30
 - load, 30
 - save, 30, 32
- GameLogic, 32
 - addObserver, 33
 - GameLogic, 33
 - GameLogic, 33
 - getConfiguration, 33
 - getWinner, 33
 - isGameOver, 33
- GamePlay, 34
 - enter, 35
 - exit, 35
 - GamePlay, 35
 - GamePlay, 35
 - run, 35
 - setCapturedPiecesList, 35
- GameState, 36
- getCameraDistanceToOrigin
 - GuiWindow, 40
- getConfiguration
 - AbstractGameLogic, 10
 - GameLogic, 33
- getHeight
 - GuiWindow, 40
- getInstance
 - StateMachine, 72
- getResourcesCount
 - ChessSet, 23
- getState
 - AIPlayer, 17
- getWidth
 - GuiWindow, 40
- getWinner
 - AbstractGameLogic, 10
 - GameLogic, 33
- GuiObserver, 36
 - GuiObserver, 37
 - GuiObserver, 37
 - onGameOver, 37
 - onGameStart, 37
 - onTurnEnd, 38
 - onTurnStart, 38
 - onTurnTimeout, 38
- GuiWindow, 38
 - getCameraDistanceToOrigin, 40
 - getHeight, 40
 - getWidth, 40
 - GuiWindow, 40
 - GuiWindow, 40
 - isFullscreen, 40
 - printHeadline, 40
 - printSubHeadline, 41
 - printText, 41
 - printTextCenter, 41
 - printTextSmall, 41
 - switchWindowMode, 42
- has_toString< T >, 42
- hasStopped
 - AnimationHelper, 20
- importScene
 - AssimpHelper, 21
- IncrementalBoardEvaluator, 42
- IncrementalZobristHasher< THASH, SEED >, 43
- isFullscreen
 - GuiWindow, 40
- isGameOver
 - AbstractGameLogic, 10
 - GameLogic, 33
- load
 - GameConfiguration, 30
- loadResources
 - ChessSet, 23
- LoggingGameObserver, 43
 - onGameOver, 44
 - onGameStart, 44
 - onTurnEnd, 44
 - onTurnStart, 44
 - onTurnTimeout, 44
- MaterialEvaluator, 45
- Menu2D, 45
- Menu2DItem, 46
 - draw, 47
 - Menu2DItem, 46
 - Menu2DItem, 46
 - mouseMoved, 47
 - mousePressed, 47
 - mouseReleased, 47
 - onClick, 47
 - setPosition, 48
- MenuGameMode, 48
 - enter, 49
 - exit, 49
 - run, 49
- MenuLoadGame, 49
 - enter, 50
 - exit, 50
 - run, 50
- MenuMain, 51
 - enter, 52
 - exit, 52
 - run, 52
- MenuOptions, 52
 - enter, 53
 - exit, 53
 - run, 53
- MenuPlayerColor, 54

- enter, 55
- exit, 55
- run, 55
- Mesh, 55
 - Mesh, 56
- Model, 56
 - Model, 57
 - setColor, 57
 - setCorrectionValues, 57
 - setPosition, 58
- mouseMoved
 - Menu2DItem, 47
- mousePressed
 - Menu2DItem, 47
- mouseReleased
 - Menu2DItem, 47
- Negamax, 58
 - search, 59
- Negamax::PerfCounters, 63
- Negamax::Result, 67
- next_p2
 - freetype, 8
- ObjectHelper, 59
 - create2DGradientRectList, 60
 - create2DRectList, 61
 - createCubeList, 61
- ObserverDispatcherProxy, 62
 - onGameOver, 62
 - onGameStart, 63
 - onTurnEnd, 63
 - onTurnStart, 63
 - onTurnTimeout, 63
- onClick
 - Menu2DItem, 47
- onGameOver
 - AbstractGameObserver, 12
 - AIPlayer, 17
 - ConsolePlayer, 26
 - GuiObserver, 37
 - LoggingGameObserver, 44
 - ObserverDispatcherProxy, 62
 - PlayerDispatcherProxy, 66
- onGameStart
 - AbstractGameObserver, 12
 - AIPlayer, 17
 - ConsolePlayer, 26
 - GuiObserver, 37
 - LoggingGameObserver, 44
 - ObserverDispatcherProxy, 63
 - PlayerDispatcherProxy, 66
- onSetColor
 - AbstractPlayer, 14
 - AIPlayer, 17
 - ConsolePlayer, 27
 - DummyPlayer, 28
 - PlayerDispatcherProxy, 66
- onTurnEnd
 - AbstractGameObserver, 12
 - ConsolePlayer, 27
 - GuiObserver, 38
 - LoggingGameObserver, 44
 - ObserverDispatcherProxy, 63
 - PlayerDispatcherProxy, 66
- onTurnStart
 - AbstractGameObserver, 12
 - GuiObserver, 38
 - LoggingGameObserver, 44
 - ObserverDispatcherProxy, 63
 - PlayerDispatcherProxy, 67
- onTurnTimeout
 - AbstractGameObserver, 13
 - GuiObserver, 38
 - LoggingGameObserver, 44
 - ObserverDispatcherProxy, 63
 - PlayerDispatcherProxy, 67
- Piece, 64
- PlayerDispatcherProxy, 64
 - doMakeTurn, 65
 - onGameOver, 66
 - onGameStart, 66
 - onSetColor, 66
 - onTurnEnd, 66
 - onTurnStart, 67
 - onTurnTimeout, 67
- PoF, 67
- poll
 - ServiceDispatcher, 69
- pop_projection_matrix
 - freetype, 8
- post
 - ServiceDispatcher, 69
- postPromise
 - ServiceDispatcher, 69
- print
 - freetype, 8
- printHeadline
 - GuiWindow, 40
- printSubHeadline
 - GuiWindow, 41
- printText
 - GuiWindow, 41
- printTextCenter
 - GuiWindow, 41
- printTextSmall
 - GuiWindow, 41
- pushScreenCoordinateMatrix
 - freetype, 8
- registerLoadCallback
 - ChessSet, 23
- run
 - AbstractState, 15
 - GamePlay, 35
 - MenuGameMode, 49
 - MenuLoadGame, 50

- MenuMain, [52](#)
- MenuOptions, [53](#)
- MenuPlayerColor, [55](#)
- ServiceDispatcher, [69](#)
- StateMachine, [72](#)
- save
 - GameConfiguration, [30](#), [32](#)
- search
 - Negamax, [59](#)
- ServiceDispatcher, [68](#)
 - poll, [69](#)
 - post, [69](#)
 - postPromise, [69](#)
 - run, [69](#)
- ServiceDispatcherThread, [70](#)
 - ServiceDispatcherThread, [71](#)
 - ServiceDispatcherThread, [71](#)
 - stop, [71](#)
- setCapturedPiecesList
 - GamePlay, [35](#)
- setColor
 - Model, [57](#)
- setCorrectionValues
 - Model, [57](#)
- setNextState
 - StateMachine, [72](#)
- setPosition
 - Menu2DItem, [48](#)
 - Model, [58](#)
- setStartState
 - StateMachine, [72](#)
- setState
 - ChessSet, [25](#)
- start
 - AbstractGameLogic, [11](#)
- StateMachine, [71](#)
 - getInstance, [72](#)
 - run, [72](#)
 - setNextState, [72](#)
 - setStartState, [72](#)
- StateMachine::EventMap, [28](#)
- stop
 - ServiceDispatcherThread, [71](#)
- switchWindowMode
 - GuiWindow, [42](#)
- Turn, [73](#)
- TurnGenerator, [73](#)