3dchess

Generated by Doxygen 1.8.6

Thu Mar 6 2014 23:38:47

Contents

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 DebugTools Namespace Reference

Contains functions for helping with debugging tasks.

Functions

- string toInitializerList (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)

  Generates a random GameState. Emulating a games a game with up to maxTurns random moves.
- template<typename Rng >
  ChessBoard generateRandomBoard (size_t maxTurns, Rng &rng)

  Generates a random Board.

4.1.1 Detailed Description

Contains functions for helping with debugging tasks.

4.1.2 Function Documentation

4.1.2.1 template<typename Rng > ChessBoard DebugTools::generateRandomBoard (size_t *maxTurns,* Rng & *rng*)

Generates a random Board.

See Also

generateRandomState

4.1.2.2 template<typename Rng > GameState DebugTools::generateRandomState (size_t *maxTurns,* Rng & *rng*)

Generates a random GameState. Emulating a games a game with up to maxTurns random moves.

Parameters

maxTurns	Limit for number of moves.
rng	C++ Random number generator to use.

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 coresponding 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 corretly 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 >= 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

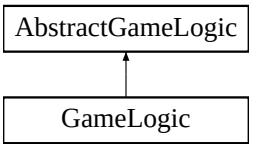
Class Documentation

5.1 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.

5.1.1 Detailed Description

Interface for chess game logic implementations.

5.1.2 Member Function Documentation

5.1.2.1 virtual void AbstractGameLogic::addObserver (AbstractGameObserverPtr *observer*) [pure virtual]

Registers an observer for game events.

See Also

AbstractGameObserver for the available events.

Parameters

observer	Observer to register.

Implemented in GameLogic.

5.1.2.2 virtual GameConfigurationPtr AbstractGameLogic::getConfiguration ( ) const [pure virtual]

Returns

GameConfiguration currently used.

Implemented in GameLogic.

5.1.2.3 virtual PlayerColor AbstractGameLogic::getWinner ( ) const [pure virtual]

Returns

If isGameOver returns the winner of the game.

Implemented in GameLogic.

5.1.2.4 virtual bool AbstractGameLogic::isGameOver ( ) const [pure virtual]

Returns

true if game has ended.

Implemented in GameLogic.

5.1.2.5 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:

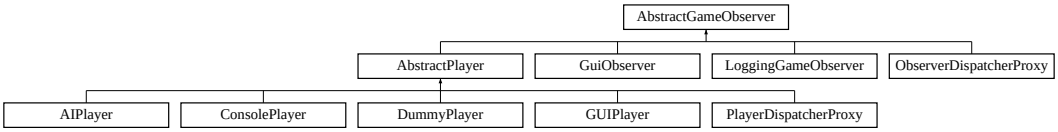
- S:/dev/3dchess/src/logic/interface/AbstractGameLogic.h

5.2 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.

5.2.1 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.

5.2.2 Member Function Documentation

5.2.2.1 virtual void AbstractGameObserver::onGameOver (GameState *state*, PlayerColor *winner*) [inline], [virtual]

Called when a game started with onGameStart is over.

Parameters

state	State on game over.
winner	Winner of the game.

Reimplemented in PlayerDispatcherProxy, GuiObserver, ObserverDispatcherProxy, AIPlayer, ConsolePlayer, and LoggingGameObserver.

5.2.2.2 virtual void AbstractGameObserver::onGameStart (GameState *state*, GameConfiguration *config*) [inline], [virtual]

Called when the game starts.

Parameters

state	GameState on game start.
config	Valid GameConfiguration for this game.

Reimplemented in PlayerDispatcherProxy, AIPlayer, GuiObserver, ObserverDispatcherProxy, ConsolePlayer, and LoggingGameObserver.

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

Called if a player ended its turn.

Parameters

who	Color of the player doing the turn.
turn	Turn the player decided on.
newState	State after the player performed the turn.

Reimplemented in PlayerDispatcherProxy, GuiObserver, ObserverDispatcherProxy, ConsolePlayer, and LoggingGameObserver.

5.2.2.4 virtual void AbstractGameObserver::onTurnStart (PlayerColor *who*) [inline], [virtual]

Called if a player is asked to perform a turn.

Parameters

who	Color of the player doing the turn.

Reimplemented in PlayerDispatcherProxy, GuiObserver, ObserverDispatcherProxy, and LoggingGameObserver.

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

Called if a players turn is aborted due to timeout.

Parameters

who	Color of the player who got interrupted.
timeout	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:

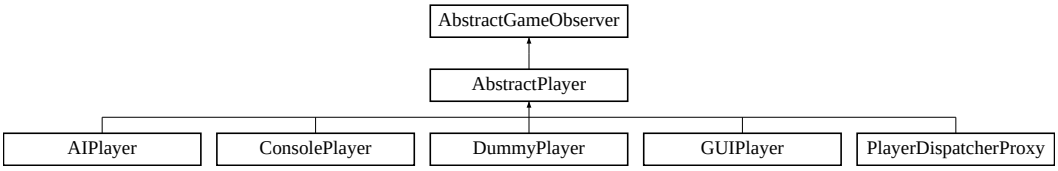
- S:/dev/3dchess/src/logic/interface/AbstractGameObserver.h

5.3 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.

5.3.1 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.

5.3.2 Member Function Documentation

5.3.2.1 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

state	Current state of the game.

Returns

A future to the turn to make.

Implemented in GUIPlayer, DummyPlayer, AIPlayer, PlayerDispatcherProxy, and ConsolePlayer.

5.3.2.2 virtual void AbstractPlayer::onSetColor (PlayerColor *color*) [pure virtual]

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

Parameters

color	Color the player has.

Implemented in GUIPlayer, AIPlayer, DummyPlayer, PlayerDispatcherProxy, and ConsolePlayer.

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

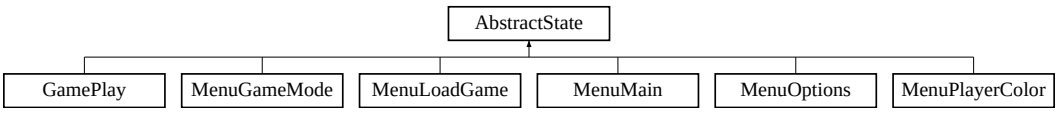
- S:/dev/3dchess/src/logic/interface/AbstractPlayer.h

5.4 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.

5.4.1 Detailed Description

Interface for modelling a game state.

Note

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

5.4.2 Member Function Documentation

5.4.2.1 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.

5.4.2.2 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.

5.4.2.3 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

5.5 AIConfiguration Class Reference

AI configuration class.

`#include <GameConfiguration.h>`

Public Member Functions

- std::string **toString** () const

Static Public Member Functions

- static AIConfiguration **defaults** ()

Public Attributes

- std::string name

  Name of this configuration.
- std::string openingBook

  Relative path to opening book. Empty for none.
- int maximumTimeForTurnInSeconds

  Time allowed to take for turn.
- bool ponderDuringOpposingPly

  Ponder when not playing.
- size_t maximumDepth

  Hard depth limit.

Friends

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

5.5.1 Detailed Description

AI configuration class.

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

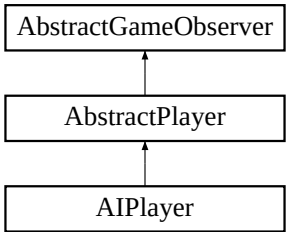
- S:/dev/3dchess/src/core/GameConfiguration.h
- S:/dev/3dchess/src/core/GameConfiguration.cpp

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 (const AIConfiguration &config, const std::string &name="AIPlayer", int seed=5253)

  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 (const AIConfiguration & *config*, const std::string & *name* = `"AIPlayer"`, int *seed* = `5253`)

Creates a new AIPlayer.

Parameters

AIConfiguration	configuration to use for AI
name	Logger channel name to use
seed	Seed to use for random operations for the player.

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

state	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

state	State on game over.
winner	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

state	GameState on game start.
config	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

color	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

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

`#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

duration	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

type	One of the FunctionType as defined above.
lowerBound	The lower bound of the range.
upperBound	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 ArrowNavigationHandler Class Reference

The class helps to handle the keyboard navigation with the arrow keys.

`#include <ArrowNavigationHandler.h>`

Public Types

- enum ArrowKey { **UP**, **RIGHT**, **DOWN**, **LEFT** }
  Enumeration for the arrow keys directions.

Public Member Functions

- ArrowNavigationHandler (bool inverseNavigation)
  Creates a new ArrowNavigationHandler object.
- void onKey (ArrowKey direction)
  This method must be called if an arrow key is pressed.
- Field getCursorPosition ()
  Returns the current cursor position.

5.8.1 Detailed Description

The class helps to handle the keyboard navigation with the arrow keys.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 ArrowNavigationHandler::ArrowNavigationHandler (bool *inverseNavigation*)

Creates a new ArrowNavigationHandler object.

Parameters

inverse-Navigation	If true left/right and up/down are changed to the opposite.

5.8.3 Member Function Documentation

5.8.3.1 Field ArrowNavigationHandler::getCursorPosition ( )

Returns the current cursor position.

Returns

The cursor position as Field.

5.8.3.2 void ArrowNavigationHandler::onKey (ArrowKey *direction*)

This method must be called if an arrow key is pressed.

Parameters

direction	the ArrowKey direction (see above).

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

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

5.9 AssimpHelper Class Reference

`#include <AssimpHelper.h>`

Public Member Functions

- void importScene (std::string filename)

  Imports the scene by filename.
- void drawScene ()

  Draws the scene.

5.9.1 Detailed Description

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

5.9.2 Member Function Documentation

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

Imports the scene by filename.

Parameters

filename	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

5.10 ChessBoard Class Reference

Chessboard representation and logic implementation.

`#include <ChessBoard.h>`

Public Member Functions

- **ChessBoard** (std::array< Piece, 64 > board, PlayerColor nextPlayer, std::array< bool, NUM_PLAYERS > shortCastleRight, std::array< bool, NUM_PLAYERS > longCastleRight, Field enPassantSquare, int halfMoveClock, int fullMoveClock)
- void applyTurn (const Turn &t)

  Applies the given turn on current chessboard.
- std::array< Piece, 64 > getBoard () const

  Returns the chessboard in array representation.
- 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, size_t depth=0) const

  Returns the current estimated score.
- Hash getHash () const

  Returns hash for current position.
- int getHalfMoveClock () const

  Returns half move clock.
- int getFullMoveClock () const

  Returns full move clock.
- std::string toFEN () const

  Converts the current board state into FEN notation.
- Field getEnPassantSquare () const

  Returns the field where en-passant rights exist. ERR 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.
- std::array< bool, NUM_PLAYERS > getKingInCheck () const

  Returns whether the king of the player is in check or not.
- bool isStalemate () const

  Gameover-Flag for stalemate position (gameover, no winner).
- std::array< bool, NUM_PLAYERS > getCheckmate () const

  Gameover-Flag for checkmate.

- bool isGameOver () const

  Returns true if the game is over.
- bool isDrawDueTo50MovesRule () const

  Returns true if the game is draw due to the 50 moves rule.
- PlayerColor getWinner () const

  Returns the winner of the game. Returns Player color or NoPlayer on draw.
- Piece getLastCapturedPiece () const

  Returns the captured piece from the last turn or Piece(NoPlayer, NoType) if no piece was captured.
- bool **operator==** (const ChessBoard &other) const
- bool **operator!=** (const ChessBoard &other) const
- std::string **toString** () const

Static Public Member Functions

- static ChessBoard fromFEN (const std::string &fen)

  Create a chessboard from a Forsyth–Edwards Notation string. http://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation.

Protected Member Functions

- void updateBitBoards ()

  Updates the helper bit boards.
- void setKingInCheck (PlayerColor player, bool kingInCheck)

  Set or unset the kingInCheck-Flag.
- void setStalemate ()

  Set the stalemate-Flag.
- void setCheckmate (PlayerColor player)

  Set the checkmate-Flag.

Protected Attributes

- std::array< std::array
  < BitBoard, NUM_PIECETYPES+1 >
  , NUM_PLAYERS > m_bb

  We use bit boards for internal turn generation. At least twelve bit boards are needed for complete board representation + some additional helper boards.

Friends

- class **TurnGenerator**
- class **IncrementalZobristHasher**

5.10.1 Detailed Description

Chessboard representation and logic implementation.

5.10.2 Member Function Documentation

5.10.2.1 ChessBoard ChessBoard::fromFEN (const std::string & *fen*) [static]

Create a chessboard from a Forsyth–Edwards Notation string. http://en.wikipedia.org/wiki/-Forsyth%E2%80%93Edwards_Notation.

Warning

This function does no validation. Do not pass invalid FEN.

Parameters

fen	FEN String.

5.10.2.2 string ChessBoard::toFEN ( ) const

Converts the current board state into FEN notation.

Returns

State in FEN notation.

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

5.11 ChessSet Class Reference

The ChessSet holds all the figures together with the board needed for the chess game.

`#include <ChessSet.h>`

Public Types

- enum TileStyle { **NORMAL**, **CURSOR**, **MOVE**, **CASTLE** }
  The style type for a tile.

Public Member Functions

- ChessSet ()
  Creates a new chess set.
- void setState (std::array< Piece, 64 > state, PlayerColor lastPlayer, Turn lastTurn)
  Sets the new chess state.
- void drawActionTileAt (Field which, TileStyle style)
  Draws an action tile at a given field with a given style.
- 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.

5.11.1 Detailed Description

The ChessSet holds all the figures together with the board needed for the chess game.

5.11.2 Member Function Documentation

5.11.2.1 void ChessSet::drawActionTileAt (Field *which,* TileStyle *style*)

Draws an action tile at a given field with a given style.

Parameters

which	The field where the tile should be drawn.
style	The action tile's style.

5.11.2.2 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.

5.11.2.3 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.

5.11.2.4 void ChessSet::registerLoadCallback (const boost::function< void(std::string)> & *callback*)

Registers a function as callback.

Parameters

callback	The function which will be called when a resource was successfully loaded.

5.11.2.5 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

state	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:

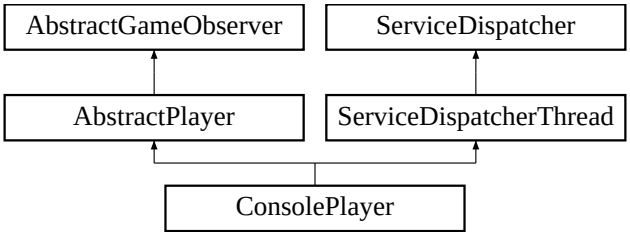
- S:/dev/3dchess/src/gui/ChessSet.h
- S:/dev/3dchess/src/gui/ChessSet.cpp

5.12 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

5.12.1 Detailed Description

Class which takes human player interaction from a console.

Warning

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

5.12.2 Member Function Documentation

5.12.2.1 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

state	Current state of the game.

Returns

A future to the turn to make.

Implements AbstractPlayer.

5.12.2.2 void ConsolePlayer::onGameOver (GameState *state*, PlayerColor *winner*) [override], [virtual]

Called when a game started with onGameStart is over.

Parameters

state	State on game over.
winner	Winner of the game.

Reimplemented from AbstractGameObserver.

5.12.2.3 void ConsolePlayer::onGameStart (GameState *state*, GameConfiguration *config*) [override], [virtual]

Called when the game starts.

Parameters

state	GameState on game start.
config	Valid GameConfiguration for this game.

Reimplemented from AbstractGameObserver.

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

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

Parameters

color	Color the player has.

Implements AbstractPlayer.

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

Called if a player ended its turn.

Parameters

who	Color of the player doing the turn.
turn	Turn the player decided on.
newState	State after the player performed the turn.

Reimplemented from AbstractGameObserver.

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

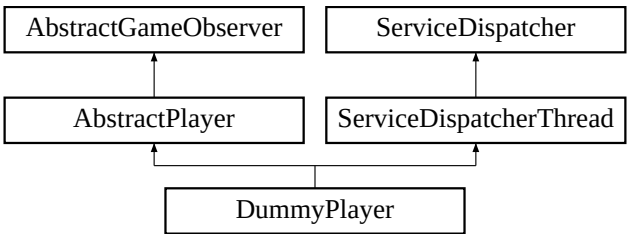
- S:/dev/3dchess/src/misc/ConsolePlayer.h
- S:/dev/3dchess/src/misc/ConsolePlayer.cpp

5.13 DummyPlayer Class Reference

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

`#include <DummyPlayer.h>`

Inheritance diagram for DummyPlayer:



Public Member Functions

- **DummyPlayer** (int seed=1234)
- 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

5.13.1 Detailed Description

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

Warning

Does not react to doAbortTurn events.

5.13.2 Member Function Documentation

5.13.2.1 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

state	Current state of the game.

Returns

A future to the turn to make.

Implements AbstractPlayer.

5.13.2.2 virtual void DummyPlayer::onSetColor (PlayerColor *color*) [inline], [override], [virtual]

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

Parameters

color	Color the player has.

Implements AbstractPlayer.

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

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

5.14 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 **keyReturn** = false

- bool **key0** = false
- bool **key1** = false
- bool **key2** = false
- bool **key3** = false
- bool **key4** = false
- bool **keyA** = false
- bool **keyY** = false
- bool **keyR** = false

5.14.1 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

5.15 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.

5.15.1 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

5.16 GameConfiguration Class 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.
- std::string initialGameStateFEN
  Initial game state as FEN string.
- int aiSelected
  Selected ai configuration (must match entry in ai)
- std::vector< AIConfiguration > ai
  List of ai configurations ordered by difficulty (simplest first)

Friends

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

5.16.1 Detailed Description

Class for holding game configuration parameters.

Note

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

5.16.2 Member Function Documentation

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

Loads a game configuration from disk.

Parameters

path	Path to file.

Returns

GameConfiguration on success. boost::none on failure.

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

Saves a given game configuration to a file.

Parameters

config	Configuration to save.
path	Path to save configuration to.

Returns

True on success.

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

Saves this configuration to the given path.

Parameters

path	Path to file to save to.

Returns

True on success.

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

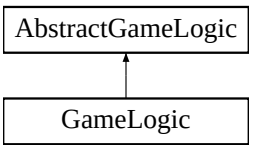
- S:/dev/3dchess/src/core/GameConfiguration.h
- S:/dev/3dchess/src/core/GameConfiguration.cpp

5.17 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, GameState initialGameState=GameState())

  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

5.17.1 Detailed Description

GameLogic implementation for a game of chess with observers.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 GameLogic::GameLogic (AbstractPlayerPtr *white*, AbstractPlayerPtr *black*, GameConfigurationPtr *config*, GameState *initialGameState* = GameState ())

Sets up a GameLogic object for one chess game.

Note

Don't forget to start operation by calling start.

Parameters

white	White player reference
black	Black player reference
config	Configuration for this game
initialGameState	Initial state when starting the game

5.17.3 Member Function Documentation

5.17.3.1 void GameLogic::addObserver (AbstractGameObserverPtr *observer*) [override], [virtual]

Registers an observer for game events.

See Also

AbstractGameObserver for the available events.

Parameters

observer	Observer to register.

Implements AbstractGameLogic.

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

Returns

GameConfiguration currently used.

Implements AbstractGameLogic.

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

Returns

If isGameOver returns the winner of the game.

Implements AbstractGameLogic.

5.17.3.4 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:

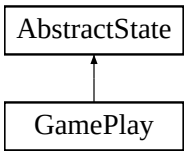
- S:/dev/3dchess/src/logic/GameLogic.h
- S:/dev/3dchess/src/logic/GameLogic.cpp

5.18 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 to chose.

Public Member Functions

- GamePlay (GameMode mode, PlayerColor humanPlayerColor, std::string initialFen="")

  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 all relevant and state related stuff on the screen.
- void startShowText (std::string text)

  Call this function to draw text on the right bottom side of the viewport.
- void switchToPlayerColor (PlayerColor color)

  When the other player is on turn, this method switched the camera position and shows a small text message which player is on turn.
- 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)

Method for setting the new chess state. This method is non-blocking.

- void setGameState (const GameState &gameState)

  Sets the game state to the given one.

- void onPlayerIsOnTurn (PlayerColor who)

  This method changed the internal state to interact with a human player if the game mode is PLAYER_VS_AI.

- void onPlayerAbortTurn ()

  If the human player aborts his current turn, this method switches to the AI player.

- std::future< Turn > doMakePlayerTurn ()

  Tells the human to make a turn.

5.18.1 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.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 GamePlay::GamePlay (GameMode *mode*, PlayerColor *humanPlayerColor*, std::string *initialFen* = " ")

Creates a new game.

Parameters

mode	The GameMode (*AI vs. AI* or *Player vs. AI*).
firstPlayerColor	The color of the player which takes the first turn.
initialFen	If set overrides the configured initial FEN

5.18.3 Member Function Documentation

5.18.3.1 std::future< Turn > GamePlay::doMakePlayerTurn ( )

Tells the human to make a turn.

Returns

The Turn as a future.

5.18.3.2 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.

5.18.3.3 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.

5.18.3.4 void GamePlay::onPlayerAbortTurn ( )

If the human player aborts his current turn, this method switches to the AI player.

Note

This method currently only works for PLAYER_VS_AI. If there are in the future other game modes, this must be corrected.

5.18.3.5 void GamePlay::onPlayerIsOnTurn (PlayerColor *who*)

This method changed the internal state to interact with a human player if the game mode is PLAYER_VS_AI.

Parameters

who	The color of the player who's on turn.

Note

This method should be called on each turn start. Otherwise the GUI can't react to keyboard input for a human player.

5.18.3.6 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.

5.18.3.7 void GamePlay::setGameState (const GameState & *gameState*)

Sets the game state to the given one.

Parameters

gameState	The new gameState to set.

5.18.3.8 void GamePlay::setState (std::array< Piece, 64 > *state*, PlayerColor *lastPlayer*, Turn *lastTurn*)

Method for setting the new chess state. This method is non-blocking.

Parameters

state	The current chess board state to set.
lastPlayer	The player which was last on turn.
lastTurn	The last turn which was made.

5.18.3.9 void GamePlay::setState (std::array< Piece, 64 > *state*)

Method for setting the new chess state. This method is non-blocking.

Parameters

state	The current chess board state to set.

Note

This should only be called one time, when the game starts.

5.18.3.10 void GamePlay::startShowText (std::string *text*)

Call this function to draw text on the right bottom side of the viewport.

Parameters

text	The string to show.

5.18.3.11 void GamePlay::switchToPlayerColor (PlayerColor *color*)

When the other player is on turn, this method switched the camera position and shows a small text message which player is on turn.

Parameters

color	The PlayerColor of the current player, which is on turn.

Note

This should always be called to show a message to the user.

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

5.19 GameState Class Reference

Public Member Functions

- **GameState** (const ChessBoard &chessBoard)
- std::vector< Turn > getTurnList () const

  Returns a list with all possible and legal turns.
- void applyTurn (const Turn &turn)

  Applies the given turn on current chessboard.
- PlayerColor getNextPlayer () const

Return next player to make a turn.

- const ChessBoard & getChessBoard () const

  Provides access to the chessboard.

- Piece getLastCapturedPiece () const

  Returns the captured piece from the last turn or Piece(NoPlayer, NoType) if no piece was captured.

- bool isGameOver () const

  Returns true if the game is over.

- PlayerColor getWinner () const

  Returns the winner of the game. Returns Player color or NoPlayer on draw.

- bool isDrawDueTo50MovesRule () const

  Returns true if the game is draw due to the 50 moves rule.

- Score getScore (size_t depth=0) const

  Returns current score estimate from next players POV.

- Hash getHash () const

  Returns hash for current position.

- std::string toFEN () const

  Converts the current game state into FEN notation.

- bool **operator==** (const GameState &other) const
- bool **operator!=** (const GameState &other) const
- std::string **toString** () const

Static Public Member Functions

- static GameState fromFEN (const std::string &fen)

  Create a GameState from a Forsyth-Edwards Notation string. http://en.wikipedia.org/wiki/-Forsyth%E2%80%93Edwards_Notation.

5.19.1 Member Function Documentation

5.19.1.1 GameState GameState::fromFEN (const std::string & *fen*) [static]

Create a GameState from a Forsyth-Edwards Notation string. http://en.wikipedia.org/wiki/-Forsyth%E2%80%93Edwards_Notation.

Warning

This function does no validation. Do not pass invalid FEN.

Parameters

fen	FEN String.

5.19.1.2 std::string GameState::toFEN ( ) const

Converts the current game state into FEN notation.

Returns

State in FEN notation.

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

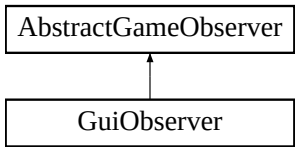
- S:/dev/3dchess/src/logic/GameState.h
- S:/dev/3dchess/src/logic/GameState.cpp

5.20 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.

5.20.1 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.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 GuiObserver::GuiObserver (ChessSetPtr *chessSetPtr,* GamePlay & *gamePlayState*)

Creates a new observer object.

Parameters

chessSetPtr	A shared pointer to the ChessSet object.
gamePlayState	A reference to the GamePlay state.

5.20.3 Member Function Documentation

5.20.3.1 void GuiObserver::onGameOver (GameState *state*, PlayerColor *winner*) [override], [virtual]

Called when a game started with onGameStart is over.

Parameters

state	State on game over.
winner	Winner of the game.

Reimplemented from AbstractGameObserver.

5.20.3.2 void GuiObserver::onGameStart (GameState *state*, GameConfiguration *config*) [override], [virtual]

Called when the game starts.

Parameters

state	GameState on game start.
config	Valid GameConfiguration for this game.

Reimplemented from AbstractGameObserver.

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

Called if a player ended its turn.

Parameters

who	Color of the player doing the turn.
turn	Turn the player decided on.
newState	State after the player performed the turn.

Reimplemented from AbstractGameObserver.

5.20.3.4 void GuiObserver::onTurnStart (PlayerColor *who*) [override], [virtual]

Called if a player is asked to perform a turn.

Parameters

who	Color of the player doing the turn.

Reimplemented from AbstractGameObserver.

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

Called if a players turn is aborted due to timeout.

Parameters

who	Color of the player who got interrupted.
timeout	Length of the time limit that got violated.

Reimplemented from AbstractGameObserver.

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

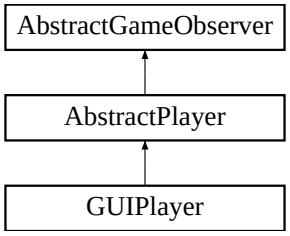
- S:/dev/3dchess/src/gui/GuiObserver.h
- S:/dev/3dchess/src/gui/GuiObserver.cpp

5.21 GUIPlayer Class Reference

Player implementation for a real/human user.

`#include <GUIPlayer.h>`

Inheritance diagram for GUIPlayer:



Public Member Functions

- GUIPlayer (GamePlay &gp)

  Creates a new human 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.

5.21.1 Detailed Description

Player implementation for a real/human user.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 GUIPlayer::GUIPlayer (GamePlay & *gp*) [inline]

Creates a new human player.

Parameters

gp	The reference to the GamePlay state.

5.21.3 Member Function Documentation

5.21.3.1 virtual std::future<Turn> GUIPlayer::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

state	Current state of the game.

Returns

A future to the turn to make.

Implements AbstractPlayer.

5.21.3.2 virtual void GUIPlayer::onSetColor (PlayerColor *color*) [inline], [override], [virtual]

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

Parameters

color	Color the player has.

Implements AbstractPlayer.

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

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

5.22 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 ()

  Inits 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 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)

5.22.1 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.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 GuiWindow::GuiWindow (std::string *title,* bool *fullscreen,* int *width,* int *height*)

Creates a new GUI window.

Parameters

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

5.22.3 Member Function Documentation

5.22.3.1 int GuiWindow::getCameraDistanceToOrigin ( )

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

Returns

The distance between camera and world origin.

5.22.3.2 int GuiWindow::getHeight ( )

Gets the height of the window.

Returns

The height of the window.

5.22.3.3 int GuiWindow::getWidth ( )

Gets the width of the window.

Returns

The width of the window.

5.22.3.4 bool GuiWindow::isFullscreen ( )

Checks if the window is currently in fullscreen mode.

Returns

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

5.22.3.5 void GuiWindow::printHeadline (std::string *text*)

Prints the headline text at the top left location.

Parameters

text	The text to draw.

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

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

Parameters

text	The text to draw.

5.22.3.7 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

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

Note

The origin of the viewport is the upper left corner.

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

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

Parameters

red	The red amount of color between 0.0 and 1.0
green	The green amount of color between 0.0 and 1.0
blue	The blue amount of color between 0.0 and 1.0
text	The text to draw.

5.22.3.9 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

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

text	The text to draw.

Note

The origin of the viewport is the upper left corner.

5.22.3.10 void GuiWindow::switchWindowMode (WindowMode *mode*)

Switches the window mode to one of the available modes.

Parameters

mode	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

5.23 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

5.24 IncrementalMaterialAndPSTEvaluator Class Reference

Class for incrementally estimating game state using PST and Material. Uses fixed piece square tables and a fixed material evaluation to incrementally calculate a score for the current board position during the game.

`#include <IncrementalMaterialAndPSTEvaluator.h>`

Public Member Functions

- IncrementalMaterialAndPSTEvaluator ()

  Initializes the evaluator for a prestine board.
- IncrementalMaterialAndPSTEvaluator (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.
- void promotionIncrement (const Turn &turn, PieceType targetType)

  Updates estimate for the promotion of a piece.
- Score getScore (PlayerColor color) const

  Returns the score from the perspective of the given player color.
- bool **operator==** (const IncrementalMaterialAndPSTEvaluator &other) const

Static Public Member Functions

- static Score estimateFullBoard (const std::array< Piece, 64 > &board)

  Gives a full estimate for the given board.

5.24.1 Detailed Description

Class for incrementally estimating game state using PST and Material. Uses fixed piece square tables and a fixed material evaluation to incrementally calculate a score for the current board position during the game.

Warning

Does not handle game over conditions

See Also

ChessBoard

Note

Not valid once game is over.

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

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

5.25 IncrementalZobristHasher Class Reference

Incremental polyglot constants based Zobrist-Hash implementation.

`#include <IncrementalZobristHasher.h>`

Public Types

- using **Hash** = uint64_t

Public Member Functions

- **IncrementalZobristHasher** (const ChessBoard &board)
- Hash getHash () const

  Returns the current zobrist hash.
- void clearedEnPassantSquare (Field enPassantSquare)

  Called when the en passant field is cleared.
- void moveIncrement (const Turn &turn)

  Called for a move update.
- void promotionIncrement (const Turn &turn, PieceType targetType)

  Called for the promotion of a pawn.
- void captureIncrement (Field field, const Piece &capturedPiece)

  Called when a piece is captured.
- void turnAppliedIncrement ()

  Updates hash for now active player.

- void newEnPassantPossibility (const Turn &turn, BitBoard opposingPawns)

  Called when turn might give the enemy an en passant possibility.
- void updateCastlingRights (const std::array< bool, NUM_PLAYERS > &prevShortCastleLeft, const std::array< bool, NUM_PLAYERS > &prevLongCastleRight, const std::array< bool, NUM_PLAYERS > &shortCastleRight, const std::array< bool, NUM_PLAYERS > &longCastleRight)

  Called to potentially update castling rights.
- bool **operator==** (const IncrementalZobristHasher &other) const

Static Public Member Functions

- static Hash hashFullBoard (const ChessBoard &board)

  Gives a full estimate for the given board.

5.25.1 Detailed Description

Incremental polyglot constants based Zobrist-Hash implementation.

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

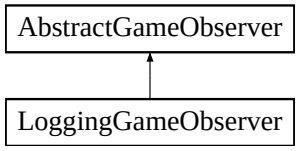
- S:/dev/3dchess/src/logic/IncrementalZobristHasher.h
- S:/dev/3dchess/src/logic/IncrementalZobristHasher.cpp

5.26 LoggingGameObserver Class Reference

AbstractGameObserver which simply logs occuring 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.

5.26.1 Detailed Description

AbstractGameObserver which simply logs occuring events.

5.26.2 Member Function Documentation

5.26.2.1 void LoggingGameObserver::onGameOver (GameState *state*, PlayerColor *winner*) [override], [virtual]

Called when a game started with onGameStart is over.

Parameters

state	State on game over.
winner	Winner of the game.

Reimplemented from AbstractGameObserver.

5.26.2.2 void LoggingGameObserver::onGameStart (GameState *state*, GameConfiguration *config*) [override], [virtual]

Called when the game starts.

Parameters

state	GameState on game start.
config	Valid GameConfiguration for this game.

Reimplemented from AbstractGameObserver.

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

Called if a player ended its turn.

Parameters

who	Color of the player doing the turn.
turn	Turn the player decided on.
newState	State after the player performed the turn.

Reimplemented from AbstractGameObserver.

5.26.2.4 void LoggingGameObserver::onTurnStart (PlayerColor *who*) [override], [virtual]

Called if a player is asked to perform a turn.

Parameters

who	Color of the player doing the turn.

Reimplemented from AbstractGameObserver.

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

Called if a players turn is aborted due to timeout.

Parameters

who	Color of the player who got interrupted.
timeout	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

5.27 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

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

slot	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

x	The x coordinate.
y	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:

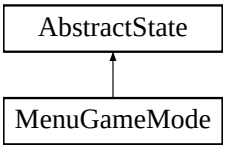
- S:/dev/3dchess/src/gui/Menu2DItem.h
- S:/dev/3dchess/src/gui/Menu2DItem.cpp

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:

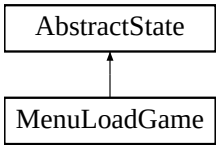
- S:/dev/3dchess/src/gui/states/MenuGameMode.h
- S:/dev/3dchess/src/gui/states/MenuGameMode.cpp

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:

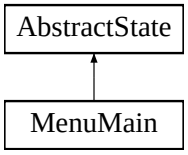
- S:/dev/3dchess/src/gui/states/MenuLoadGame.h
- S:/dev/3dchess/src/gui/states/MenuLoadGame.cpp

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:

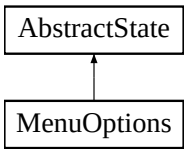
- S:/dev/3dchess/src/gui/states/MenuMain.h
- S:/dev/3dchess/src/gui/states/MenuMain.cpp

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:

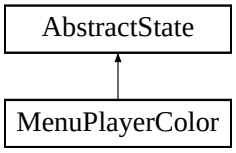
- S:/dev/3dchess/src/gui/states/MenuOptions.h
- S:/dev/3dchess/src/gui/states/MenuOptions.cpp

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

- GLuint m_numVertices

  The number of vertices.
- aiVector3D * m_vertices

  The model's vertices.
- aiVector3D * m_normals

  The model's normals.
- aiVector3D * m_textureCoords

  The model's texture coordinates.
- GLuint * m_indices

  The model's indices.
- GLuint m_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

numVertices	The number of model vertices.
vertices	The model's vertices itself.
normals	The model's normals itself.
numFaces	The number of model faces.
faces	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

file	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

color	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

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

globalX	The global x coordinate.
globalY	The global y coordinate.
globalZ	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 PolyglotBookEntry::Move Struct Reference

Chess move.

`#include <PolyglotBook.h>`

Public Member Functions

- bool **operator==** (const Move &other) const
- std::string **toString** () const

Public Attributes

- Field from

  Source field. For castling king source.
- Field to

  Target field. For castling king target.
- PieceType promotion_piece

  If promotion piece type to promote to.

5.35.1 Detailed Description

Chess move.

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

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

5.36 Negamax< TGameState, AB_CUTOFF_ENABLED, MOVE_ORDERING_ENABLED, TRANSPOSITION_TABLES_ENABLED > Class Template Reference

Implementation of a Negamax algorithm.

`#include <Negamax.h>`

Classes

- struct PerfCounters

  Structure with performance counters used for debugging and evaluation.

Public Member Functions

- Negamax ()

  Creates a new algorithm instance.
- NegamaxResult search (const TGameState &state, size_t maxDepth)

  Search given state up to maxDepth full turns.
- void abort ()

  Aborts the currently running calculation. Call from another thread to abort currently running search.

Public Attributes

- struct Negamax::PerfCounters **m_counters**

5.36.1 Detailed Description

template<typename TGameState = GameState, bool AB_CUTOFF_ENABLED = true, bool MOVE_ORDERING_ENABLED = true, bool TRANSPOSITION_TABLES_ENABLED = true>class Negamax< TGameState, AB_CUTOFF_ENABLED, MOVE_ORDERING_ENABLED, TRANSPOSITION_TABLES_ENABLED >

Implementation of a Negamax algorithm.

Template Parameters

TGameState	Type of game state so GameState is mockable.
AB_CUTOFF_ENABLE	If false Alpha-Beta cutoff feature is disabled.
MOVE_ORDERING_ENABLED	If false move ordering is disabled.
TRANSPOSITION_TABLES_ENABLED	If false transposition tables are disabled.

5.36.2 Member Function Documentation

5.36.2.1 template<typename TGameState = GameState, bool AB_CUTOFF_ENABLED = true, bool MOVE_ORDERING_ENABLED = true, bool TRANSPOSITION_TABLES_ENABLED = true> NegamaxResult Negamax< TGameState, AB_CUTOFF_ENABLED, MOVE_ORDERING_ENABLED, TRANSPOSITION_TABLES_ENABLED >::search (const TGameState & *state*, size_t *maxDepth*) [inline]

Search given state up to maxDepth full turns.

Parameters

state	Game state to search.
maxDepthInTurns	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

5.37 NegamaxResult Struct Reference

Structure for holding search results.

`#include <Negamax.h>`

Public Member Functions

- NegamaxResult operator- () const

  Negates score. Syntax sugar to get closer to algorithm notation.
- bool isVictoryCertain () const

  Returns true if the search found a certain victory.
- bool **operator<** (const NegamaxResult &other)
- bool **operator<=** (const NegamaxResult &other)
- bool **operator>=** (const NegamaxResult &other)
- bool **operator>** (const NegamaxResult &other)
- bool **operator==** (const NegamaxResult &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.

5.37.1 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

5.38 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.

5.38.1 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.

5.38.2 Member Function Documentation

5.38.2.1 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

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

5.38.2.2 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

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

Returns

GLuint A display list index which holds the compiled rectangle.

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

Creates a new OpenGL display list for a cube.

Parameters

size	The size of an edge of the cube.
x	The position of the cube in x world coordinate.
y	The position of the cube in y world coordinate.
z	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:

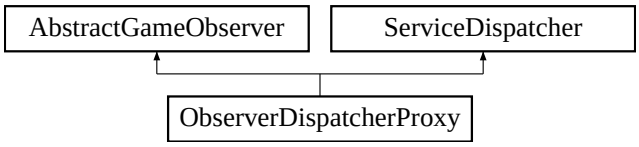
- S:/dev/3dchess/src/gui/ObjectHelper.h
- S:/dev/3dchess/src/gui/ObjectHelper.cpp

5.39 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 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

5.39.1 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.

5.39.2 Member Function Documentation

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

Called when a game started with onGameStart is over.

Parameters

state	State on game over.

winner	Winner of the game.

Reimplemented from AbstractGameObserver.

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

Called when the game starts.

Parameters

state	GameState on game start.
config	Valid GameConfiguration for this game.

Reimplemented from AbstractGameObserver.

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

Called if a player ended its turn.

Parameters

who	Color of the player doing the turn.
turn	Turn the player decided on.
newState	State after the player performed the turn.

Reimplemented from AbstractGameObserver.

5.39.2.4 virtual void ObserverDispatcherProxy::onTurnStart (PlayerColor *who*) [inline], [override], [virtual]

Called if a player is asked to perform a turn.

Parameters

who	Color of the player doing the turn.

Reimplemented from AbstractGameObserver.

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

Called if a players turn is aborted due to timeout.

Parameters

who	Color of the player who got interrupted.
timeout	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

5.40 Negamax< TGameState, AB_CUTOFF_ENABLED, MOVE_ORDERING_ENABLED, TRANSPOSITION_TABLES_ENABLED >::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.
- uint64_t transpositionTableHits

  Number of transposition table hits during search.
- std::chrono::milliseconds duration

  Time taken for last search.

5.40.1 Detailed Description

template<typename TGameState = GameState, bool AB_CUTOFF_ENABLED = true, bool MOVE_ORDERING_ENABLED = true, bool TRANSPOSITION_TABLES_ENABLED = true>struct Negamax< TGameState, AB_CUTOFF_ENABLED, MOVE_ORDERING_ENABLED, TRANSPOSITION_TABLES_ENABLED >::PerfCounters

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

5.41 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:

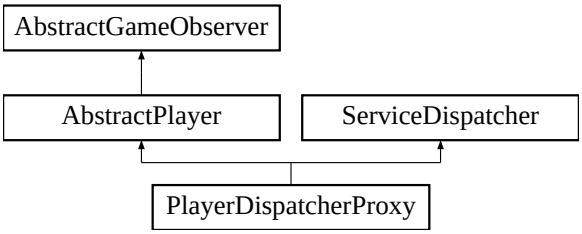
- S:/dev/3dchess/src/logic/ChessTypes.h
- S:/dev/3dchess/src/logic/ChessTypes.cpp

5.42 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

5.42.1 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.

5.42.2 Member Function Documentation

5.42.2.1 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

state	Current state of the game.

Returns

A future to the turn to make.

Implements AbstractPlayer.

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

Called when a game started with onGameStart is over.

Parameters

state	State on game over.
winner	Winner of the game.

Reimplemented from AbstractGameObserver.

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

Called when the game starts.

Parameters

state	GameState on game start.
config	Valid GameConfiguration for this game.

Reimplemented from AbstractGameObserver.

5.42.2.4 virtual void PlayerDispatcherProxy::onSetColor (PlayerColor *color*) [inline], [override], [virtual]

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

Parameters

color	Color the player has.

Implements AbstractPlayer.

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

Called if a player ended its turn.

Parameters

who	Color of the player doing the turn.
turn	Turn the player decided on.
newState	State after the player performed the turn.

Reimplemented from AbstractGameObserver.

5.42.2.6 virtual void PlayerDispatcherProxy::onTurnStart (PlayerColor *who*) [inline], [override], [virtual]

Called if a player is asked to perform a turn.

Parameters

who	Color of the player doing the turn.

Reimplemented from AbstractGameObserver.

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

Called if a players turn is aborted due to timeout.

Parameters

who	Color of the player who got interrupted.
timeout	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

5.43 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

5.44 PolyglotBook Class Reference

Class able to lookup entries from a polyglot opening file. Format as described on http://hgm.nubati.net/book_format.html.

`#include <PolyglotBook.h>`

Public Member Functions

- PolyglotBook (int seed=5235)

  Creates a new book instance.
- bool open (const std::string &book)

  Opens a given book file and reads it into memory.
- std::vector< PolyglotBookEntry > lookup (uint64_t key) const

  Performs a lookup for the given key and returns all entries with matching key.
- boost::optional < PolyglotBookEntry > getWeightedEntry (uint64_t key)

  Selects a entry based on the relative weights of the results from a lookup of key. A turns probability of being chosen is weight/sum(weights)
- boost::optional < PolyglotBookEntry > getBestEntry (uint64_t key) const

  Selects the entry with maximum weight from a lookup of key.
- size_t getNumberOfEntries () const

  Returns the number of entries in the book.

5.44.1 Detailed Description

Class able to lookup entries from a polyglot opening file. Format as described on http://hgm.nubati.net/book_format.html.

Note

Book is held in memory for the lifetime of the object.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 PolyglotBook::PolyglotBook (int *seed* = 5235) [explicit]

Creates a new book instance.

See Also

open

Parameters

seed	Seed which to use for selections with chance.

5.44.3 Member Function Documentation

5.44.3.1 boost::optional< PolyglotBookEntry > PolyglotBook::getBestEntry (uint64_t *key*) const

Selects the entry with maximum weight from a lookup of key.

Parameters

key	Zobrist hash for position

Returns

boost::none if no turn found.

5.44.3.2 boost::optional< PolyglotBookEntry > PolyglotBook::getWeightedEntry (uint64_t *key*)

Selects a entry based on the relative weights of the results from a lookup of key. A turns probability of being chosen is weight/sum(weights)

Parameters

key	Zobrist hash for position

Returns

boost::none if no turn found.

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

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

5.45 PolyglotBookEntry Struct Reference

Single entry in in polyglot book adjusted for this engine.

`#include <PolyglotBook.h>`

Classes

- struct Move

  Chess move.

Public Member Functions

- bool **isPromotion** () const
- bool mightBeCastlingMove () const

  Returns true if this might be a castling move.
- Field getKingCastlingTarget () const

  Returns the target of the king piece for this move if it were a castling one.
- bool **operator==** (const PolyglotBookEntry &other) const
- std::string **toString** () const

Public Attributes

- uint64_t key

  Entry key (Zobrist hash)
- struct PolyglotBookEntry::Move **move**
- uint16_t weight

  Score for move from book.

5.45.1 Detailed Description

Single entry in in polyglot book adjusted for this engine.

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

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

5.46 ResourceInitializer Class Reference

`#include <ResourceInitializer.h>`

Public Member Functions

- ResourceInitializer ()

  Creates a new ResourceInitializer object.
- ChessSetPtr load ()

  Loads the whole chess set, shows the progress bar and the file which is loaded.

5.46.1 Detailed Description

This class will initialize the chess figures/models and the chess board. It will also show the status with a progress bar and the model which is loading.

5.46.2 Member Function Documentation

5.46.2.1 ChessSetPtr ResourceInitializer::load ( )

Loads the whole chess set, shows the progress bar and the file which is loaded.

Returns

The smart pointer to the chess set.

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

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

5.47 SaveGame Class Reference

SaveGame class for a single savegame.

`#include <SaveGame.h>`

Public Member Functions

- **SaveGame** (std::string fen_, GamePlay::GameMode gameMode_, PlayerColor humanPlayerColor_)
- bool save (const std::string &path) const

  Saves this game to the given path.
- bool saveToSlot (int slot)

  Convinience function which saves to a path determined by the slot.

Static Public Member Functions

- static boost::optional< SaveGame > load (const std::string &path)

  Loads a savegame from disk.
- static boost::optional< SaveGame > loadFromSlot (int slot)

  Loads a savegame from a slot path.
- static bool save (const SaveGame &saveGame, const std::string &path)

  Saves a given save game to a file.

Public Attributes

- std::string fen

  Saved state in fen notation.
- GamePlay::GameMode gameMode

  Game mode (ai vs human etc.)
- PlayerColor humanPlayerColor

  Color of the human player if human vs ai.

Friends

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

5.47.1 Detailed Description

SaveGame class for a single savegame.

5.47.2 Member Function Documentation

5.47.2.1 boost::optional< SaveGame > SaveGame::load (const std::string & *path*) [static]

Loads a savegame from disk.

Parameters

path	Path to file.

Returns

SaveGame on success. boost::none on failure.

5.47.2.2 boost::optional< SaveGame > SaveGame::loadFromSlot (int *slot*) [static]

Loads a savegame from a slot path.

Parameters

slot	number

Returns

SaveGame on success. boost::none on failure.

5.47.2.3 bool SaveGame::save (const SaveGame & *saveGame,* const std::string & *path*) [static]

Saves a given save game to a file.

Parameters

saveGame	Save game to save.
path	Path to save file to.

Returns

True on success.

5.47.2.4 bool SaveGame::save (const std::string & *path*) const

Saves this game to the given path.

Parameters

path	Path to file to save to.

Returns

True on success.

5.47.2.5 bool SaveGame::saveToSlot (int *slot*)

Convinience function which saves to a path determined by the slot.

Returns

true on success

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

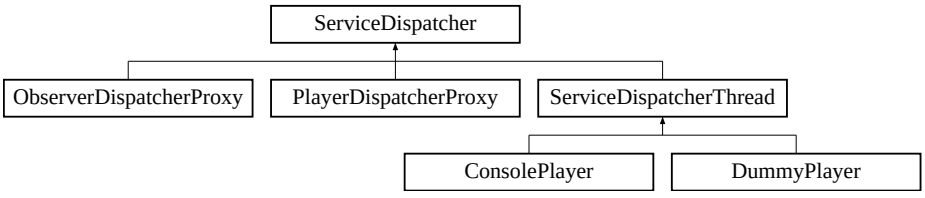
- S:/dev/3dchess/src/gui/SaveGame.h
- S:/dev/3dchess/src/gui/SaveGame.cpp

5.48 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.

5.48.1 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.

5.48.2 Member Function Documentation

5.48.2.1 void ServiceDispatcher::poll ( ) [inline]

Replays all posted functions in the calling thread.

See Also

post
postPromise

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

Store a given function.

Parameters

function	Function to store and later replay.

5.48.2.3 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

function	Function to store and later replay.

Returns

Future on result of given function.

5.48.2.4 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:

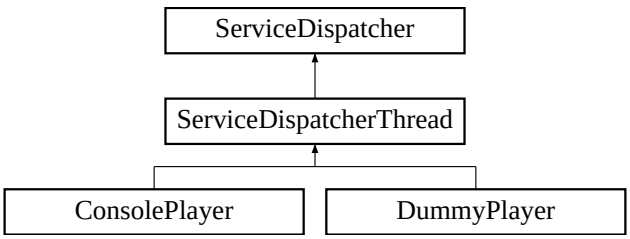
- S:/dev/3dchess/src/logic/threading/ServiceDispatcher.h

5.49 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

5.49.1 Detailed Description

Provides functionality for safely running operations in a thread.

Uses ServiceDispatcher to move function calls into its own thread and execute them.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 ServiceDispatcherThread::ServiceDispatcherThread ( ) [inline]

Creates a ServiceDispatcherThread.

Note

Don't forget to start() it.

5.49.3 Member Function Documentation

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

Stops the execution of this tread.

Parameters

force	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

5.50 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.

5.50.1 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.

5.50.2 Member Function Documentation

5.50.2.1 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.

5.50.2.2 AbstractState * StateMachine::run ( )

Runs the current state.

Returns

The AbstractState pointer to the state which must be run() the next time.

5.50.2.3 void StateMachine::setNextState (AbstractState * *state*)

Sets the next state which should be run.

Parameters

state	The next state.

5.50.2.4 void StateMachine::setStartState (AbstractState * *startState*)

Sets the start state and setup the state.

Parameters

startState	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

5.51 TranspositionTable Class Reference

Transposition table with fixed size. Hashed on hash of transposition table entry. Offers limited internal collision detection against class 2 errors by checking hash in entry before returning. Class 1 errors should handled externally if problematic.

`#include <TranspositionTable.h>`

Public Member Functions

- TranspositionTable (size_t tablesize=4000037)

  Creates an empty transposition table of given size.
- void maybeUpdate (TranspositionTableEntry entry)

  Stores the given entry if it meets table replacement criteria. Stores the given entry either if it belongs to a different position than the current one or if its depth is greater than the previous entry for this position. This relies on the assumption that deeper entries most likely took more positions into account thus representing a greater investment in compute time.
- boost::optional
  < TranspositionTableEntry > lookup (Hash hash) const

  Lookup hash in table.
- size_t getTableSize () const

  Returns the number of possible independent table entries.

5.51.1 Detailed Description

Transposition table with fixed size. Hashed on hash of transposition table entry. Offers limited internal collision detection against class 2 errors by checking hash in entry before returning. Class 1 errors should handled externally if problematic.

5.51.2 Constructor & Destructor Documentation

5.51.2.1 TranspositionTable::TranspositionTable (size_t *tablesize* = `4000037`) [inline]

Creates an empty transposition table of given size.

Parameters

tablesize	Number of independent spaces in hashtable.

Note

To ensure even distribution tablesize should be prime.

5.51.3 Member Function Documentation

5.51.3.1 boost::optional<TranspositionTableEntry> TranspositionTable::lookup (Hash *hash*) const [inline]

Lookup hash in table.

Note

Not secure against zobrist hash collisions.

Returns

Option to entry if in table. boost::none otherwise.

5.51.3.2 void TranspositionTable::maybeUpdate (TranspositionTableEntry *entry*) [inline]

Stores the given entry if it meets table replacement criteria. Stores the given entry either if it belongs to a different position than the current one or if its depth is greater than the previous entry for this position. This relies on the assumption that deeper entries most likely took more positions into account thus representing a greater investment in compute time.

Parameters

entry	Entry to store.

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

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

5.52 TranspositionTableEntry Struct Reference

Single entry in transposition table.

`#include <TranspositionTable.h>`

Public Types

- enum BoundType { LOWER, UPPER, EXACT }

  Describes the guarantees for the entry.

Public Member Functions

- bool isLowerBound () const

  Returns true if entry score is lower bound to score attainable by turn.
- bool isUpperBound () const

  Returns true if entry score is upper bound to score attainable by turn.
- bool isExactBound () const

  Returns true if score is exactly what is attainable by turn.
- std::string **toString** () const

Public Attributes

- Hash hash

  Hash identifying position (might collide)
- Turn turn

  Best turn from this position.
- Score score

  Estimated score (.
- enum TranspositionTableEntry::BoundType **boundType**
- size_t **depth**

5.52.1 Detailed Description

Single entry in transposition table.

See Also

TranspositionTable

5.52.2 Member Enumeration Documentation

5.52.2.1 enum TranspositionTableEntry::BoundType

Describes the guarantees for the entry.

Enumerator

LOWER Score is lower bound to score attainable by turn.

UPPER Score is upper bound to score attainable by turn.

EXACT Score is exactly what is attainable by turn.

5.52.3 Member Data Documentation

5.52.3.1 Score TranspositionTableEntry::score

Estimated score (.

See Also

boundType,
depth)

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

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

5.53 Turn Class Reference

Public Types

- enum **Action** {
  Move, **Castle**, **Forfeit**, **Pass**,
  PromotionQueen, **PromotionBishop**, **PromotionKnight**, **PromotionRook** }

Public Member Functions

- **Turn** (Piece piece, Field from, Field to, Action action)
- bool **isMove** () const
- bool **isCastling** () const
- bool **isPromotion** () const
- bool **isForfeit** () const
- bool **isPass** () const
- PieceType **getPromotionPieceType** () const
- 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)
- static Turn **castle** (Piece piece, Field from, Field to)
- static Turn **promotionQueen** (Piece piece, Field from, Field to)
- static Turn **promotionBishop** (Piece piece, Field from, Field to)
- static Turn **promotionRook** (Piece piece, Field from, Field to)
- static Turn **promotionKnight** (Piece piece, Field from, Field to)
- static Turn **pass** (PlayerColor player)

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

5.54 TurnGenerator Class Reference

Turn generation (based on bitboards) and gameover detection.

`#include <TurnGenerator.h>`

Public Member Functions

- void initFlags (ChessBoard &cb)

  Sets the kingInCheck-Flag, based on the given chessboard.
- std::vector< Turn > getTurnList () const

  Returns the generated turns.
- void generateTurns (PlayerColor player, ChessBoard &cb)

  Generates turns for the given player color, based on the given chessboard.
- void bitBoardToTurns (Piece piece, Field from, BitBoard bbTurns, BitBoard bbAllOppTurns, ChessBoard &cb, Turns &turnsOut)

  Creates turn objects from bitboards adds it to turnsOut list.

- BitBoard **calcMoveTurns** (Piece piece, BitBoard bbPiece, BitBoard bbAllOppTurns, const ChessBoard &cb)
- BitBoard **calcUnCheckFields** (PlayerColor opp, const ChessBoard &cb)
- BitBoard **calcAllOppTurns** (PlayerColor opp, const ChessBoard &cb)
- BitBoard **calcShortCastleTurns** (PlayerColor player, BitBoard bbAllPieces, BitBoard bbAllOppTurns)
- BitBoard **calcLongCastleTurns** (PlayerColor player, BitBoard bbAllPieces, BitBoard bbAllOppTurns)
- BitBoard **calcKingTurns** (BitBoard king, BitBoard allOwnPieces, BitBoard allOppTurns) const
- BitBoard **calcKnightTurns** (BitBoard knights, BitBoard allOwnPieces) const
- BitBoard **calcPawnTurns** (BitBoard pawns, BitBoard allPieces, BitBoard allOppPieces, PlayerColor player, Field enPassantSquare) const
- BitBoard **calcPawnMoveTurns** (BitBoard pawns, BitBoard allPieces, PlayerColor player) const
- BitBoard **calcPawnAttackTurns** (BitBoard pawns, BitBoard allOppPieces, PlayerColor player, Field enPassantSquare) const
- BitBoard **calcQueenTurns** (BitBoard queens, BitBoard allOppPieces, BitBoard allPieces) const
- BitBoard **calcBishopTurns** (BitBoard bishops, BitBoard allOppPieces, BitBoard allPieces) const
- BitBoard **calcRookTurns** (BitBoard rooks, BitBoard allOppPieces, BitBoard allPieces) const
- BitBoard **maskRank** (Rank rank) const
- BitBoard **clearRank** (Rank rank) const
- BitBoard **maskFile** (File file) const
- BitBoard **clearFile** (File file) const
- BitBoard **getBitsE** (BitBoard bbPiece) const
- BitBoard **getBitsW** (BitBoard bbPiece) const
- BitBoard **getBitsN** (BitBoard bbPiece) const
- BitBoard **getBitsS** (BitBoard bbPiece) const
- BitBoard **getBitsNE** (BitBoard bbPiece) const
- BitBoard **getBitsNW** (BitBoard bbPiece) const
- BitBoard **getBitsSE** (BitBoard bbPiece) const
- BitBoard **getBitsSW** (BitBoard bbPiece) const

Public Attributes

- std::vector< Turn > turnList

  Contains the generated turns.

5.54.1 Detailed Description

Turn generation (based on bitboards) and gameover detection.

5.54.2 Member Function Documentation

5.54.2.1 std::vector< Turn > TurnGenerator::getTurnList ( ) const

Returns the generated turns.

Warning

The generateTurns-function needs to be called previously.

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