BrickGame

1

Generated by Doxygen 1.9.8

Chapter 1

Brick Game

The project consists of two distinct games, Tetris and Snake, developed for both CLI (Command-Line Interface) and Desktop environments. The Tetris game was implemented in C, while the Snake game was implemented in C++. The CLI interface utilized the ncurses library, while the Desktop interface leveraged the Qt GUI library.

Main targets in the Makefile:

all: The default target that builds the project by running `install` and `gcov_report`.

install: Builds the project for the CLI and Desktop interfaces.

install_ui_cli: Compiles and creates the main game executable for the CLI interface.

run_cli: Runs the compiled game executable for the CLI interface.

install_ui_qt: Builds the UI with Qt using CMake.

run_desktop: Runs the compiled game executable for the Desktop interface.

uninstall: Removes all build artifacts and clears the project directory.

rebuild: Rebuilds the project, removing and reinstalling the project.

test: Runs tests using CTest and Google Test.

ctest: Compiles and runs tests using CTest.

gtest: Compiles and runs tests using Google Test.

gcov_report: Generates a coverage report using gcov.

gcov_show: Opens the generated coverage report.

peer: Clears build artifacts and generates a coverage report.

dvi: Opens the generated DVI documentation.

dist: Creates a distribution tarball from the project directory.

clean: Removes all build artifacts, intermediate files, and temporary directories.

1.1 Snake

BrickGame Snake is a popular mobile game that is a variation of the classic Snake game. The game is developed by BrickGame Studio and is available for both Android and iOS devices.

The game is played on a grid, where you control a snake that moves horizontally and vertically. The snake's body is made up of blocks, and it grows longer as you collect food pellets. The goal is to guide the snake to eat as many food pellets as possible while avoiding collisions with walls and its own body.

1.2 Controller cli

- **Left Arrow**: Moves the left.
- **Right Arrow**: Moves the right.
- **Down Arrow**: Moves the down.
- **Up Arrow**: Moves the up.
- **Space**: Speeding up.
- **q**: Exit the game.
- **p or P**: Pause.
- **enter**: Button for confirmation.

1.3 Controller desktop

- **Left Arrow**: Moves the left.
- **Right Arrow**: Moves the right.
- **Down Arrow**: Moves the down.
- **Up Arrow**: Moves the up.
- **Space**: Speeding up.
- **esc**: Exit the game.
- **pause or p or P**: Pause.
- **enter**: Button for confirmation.

1.4 Point and Levels

Each time a player gains 5 points, the level increases by 1. Increasing the level increases the speed of the snake. The maximum number of levels is 10. If a player can reach 200 points, he wins.

1.5 Diagram

This diagram describes the machine used:

1.5.1 Game states

The game consists of several states, each with different behaviors and transitions.

- **START**: the game is in the idle state, waiting for a user command to start or exit.
- **SPAWN**: after entering this state, the game creates a new actor object and transitions to another state. This state occurs when the game restarts, collides with an object or ends the game (In case the player wants to play again).
- **MOVE**: the game enters this state after spawning a new actor object. A timer is started and the direction of the snake's movement is selected. The game can also go into PAUSE state if the user pauses the game.
- **SHIFT**: the game enters this state after the timer expires. The snake's position is updated and the game checks for collisions with walls or itself or fruit. If a collision occurs, the game switches to the COLLISION state. Otherwise, it switches back to the MOVE state.
- **COLLISION**: the game enters this state when the snake collides with an object. If the snake collides with its own body or with a wall, the game ends in defeat. If the snake collides with a fruit object and its length is maximized, the game ends in victory. Otherwise, the game goes back to the SPAWN state.
- **GAMEOVER**: the game is in the idle state, waiting for a user signal to restart or exit.
- **PAUSE**: The game enters this state when the user pauses the game. The game freezes and waits for the user to resume play by clicking on the pause button.

Enjoy playing Brick Game Snake!

1.6 Tetris

Brick Game Tetris is a classic arcade game in which players control falling tetrominoes (geometric shapes made of four square blocks) to create complete rows of blocks without gaps. When a row is full, it is cleared and the player earns points. The game ends when the tetromino stack reaches the top of the playing field.

1.7 Controller cli

- **Left Arrow**: Moves the falling tetromino to the left.
- **Right Arrow**: Moves the falling tetromino to the right.
- **Down Arrow**: Instantly lowers the tetrominoes to the very bottom of the playing field.
- **Up Arrow**: Not used.
- **Space**: Rotate the falling tetromino clockwise.
- **q**: Exit the game.
- **p or P**: Pause.
- **enter**: Button for confirmation.

1.8 Controller desktop

- **Left Arrow**: Moves the falling tetromino to the left.
- **Right Arrow**: Moves the falling tetromino to the right.
- **Down Arrow**: Instantly lowers the tetrominoes to the very bottom of the playing field.
- **Up Arrow**: Not used.
- **Space**: Rotate the falling tetromino clockwise.
- **esc**: Exit the game.
- **pause or p or P**: Pause.
- **enter**: Button for confirmation.

1.9 Point

Players earn points by creating complete rows of blocks. Points increase depending on the number of simultaneously cleared rows:

- **For one level**: 100 points
- **For two levels**: 300 points
- **For three levels**: 700 points
- **For four levels**: 1500 points

1.10 Levels

As the player gains points and clears the ranks, he advances through the levels.

- **First level**: 600 points
- **Second level**: 1200 points
- **Third level**: 1800 points
- **Level Four**: 2400 points
- **Level 5**: 3000 points
- **Level Six**: 3600 points
- **Seventh level**: 4200 points
- **Level Eight**: 4800 points
- **Level Nine**: 5400 points
- **Tenth level**: 6000 points

1.11 Speed

With each level, the speed at which the tetrominoes fall increases:

- **Level 1**: Speed factor: 300000
- **Level 2**: Speed Factor: 290000
- **Level 3**: Speed Factor: 280000
- **Level 4**: Speed Factor: 270000
- **Level 5**: Speed Factor: 260000
- **Level 6**: Speed Factor: 250000
- **Level 7**: Speed Factor: 240000
- **Level 8**: Speed Factor: 230000
- **Level 9**: Speed Factor: 220000
- **Level 10**: Speed Factor: 210000

1.12 Diagram

This diagram describes the machine used:

1.12.1 Game states

The game consists of several states, each with different behaviors and transitions.

- **START**: the game is in an idle state, waiting for a user command to start or exit.
- **SPAWN**: after entering this state, the game creates a new model and the next model is defined and transitions to another state. This state occurs when the game restarts, collides or terminates the game (in case the player wants to play again).
- **MOVE**: the game enters this state after a new object is spawned. A timer is started, x-axis movement can be performed and the shape can be rotated. If a collision occurs, the game enters the COLLISION state. The game can also go to the PAUSE state if the user pauses the game.
- **SHIFT**: the game enters this state when the timer expires. The position of the figure changes along the y-axis. If a collision occurs, the game enters the COLLISION state. Otherwise it returns to the MOVE state. COLLISION: the game enters this state when the figure collides. If the piece collides on the top line, the game is over. Otherwise the game returns to the SPAWN state.
- **GAMEOVER**: the game is in the idle state, waiting for a user signal to restart or exit.
- **PAUSE**: the game enters this state when the user pauses the game. The game freezes and waits for the user to resume the game by pressing the pause button.

Enjoy playing Brick Game Tetris!

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Topic Documentation

6.1 User Interface Macros

Collection of macros for ui parameters and accessors.

Macros

- #define MAX_CELLS 200
- #define WINDOW_FIELD_HEIGHT HFIELD + 2
- #define HEIGHT_BOARD_NEXT 6
- #define HEIGHT_WIN_PAUSE 3
- #define HEIGHT_WIN_START 9
- #define HEIGHT_WIN_EXIT 9
- #define WINDOW_FIELD_WIDTH WFIELD * 2 + 2
- #define WIDTH_BOARD_NEXT 15
- #define WIDTH_WIN_PAUSE 40
- #define WIDTH_WIN_START 45
- #define WIDTH_WIN_EXIT 40
- #define WIN_PAUSE 0
- #define WIN_START 1
- #define WIN_EXIT 2
- #define WIN_FIELD 3
- #define SIZE_BOX 1
- #define DRAW_POS_Y1 1
- #define DRAW_POS_Y2 3
- #define DRAW_POS_Y3 5
- #define DRAW_POS_Y4 7
- #define START_X_BOAR 3
- #define START_Y_BOAR 3
- #define GET_POS_X1(x) x * 2 + 1

  Macro to return the first position of the square on the window.
- #define GET_POS_X2(x) GET_POS_X1(x) + 1

  Macro to return the second position of the square in the window.
- #define GETMAXWH(height, width) getmaxyx(stdscr, height, width);

  Macro to get terminal dimensions.
- #define SET_COLOR_PAIR(win, index) wbkgdset(win, COLOR_PAIR(index));

Macro for setting window color pair.

- #define CHECK_WIN(win) (win != NULL)
- #define CHECK_FIELD(field) (field != NULL)
- #define DRAW_BOX(win) box(win, 0, 0);

  Macro for drawing a box.
- #define COLOR_BOX(win, index) wbkgd(win, COLOR_PAIR(index));

  Macro for setting box color pair.
- #define COLOR_TEXT(win, index) wattron(win, COLOR_PAIR(index));

  Macro for setting text color pair.
- #define DRAW_CELL(x, y, win)

  Macro for inserting a square.
- #define GET_INFO_PRINT(index)

  The macro returns the text according to the index, indices from the `INDEX_TEXT_INFO` enumeration.

6.1.1 Detailed Description

Collection of macros for ui parameters and accessors.

6.1.2 Macro Definition Documentation

6.1.2.1 CHECK_FIELD

```
#define CHECK_FIELD(
              field ) (field != NULL)
```

Checking the field for NULL

6.1.2.2 CHECK_WIN

```
#define CHECK_WIN(
              win ) (win != NULL)
```

Checking the window for NULL

6.1.2.3 COLOR_BOX

```
#define COLOR_BOX(
              win,
              index ) wbkgd(win, COLOR_PAIR(index));
```

Macro for setting box color pair.

Parameters

win	The box in which the color will be set
index	Color pair index

6.1.2.4 COLOR_TEXT

```
#define COLOR_TEXT(
              win,
              index ) wattron(win, COLOR_PAIR(index));
```

Macro for setting text color pair.

Parameters

win	The box in which the color will be set
index	Color pair index

6.1.2.5 DRAW_BOX

```
#define DRAW_BOX(
              win ) box(win, 0, 0);
```

Macro for drawing a box.

Parameters

win	drawing window

6.1.2.6 DRAW_CELL

```
#define DRAW_CELL(
              x,
              y,
              win )
```

Value:

```
  mvwaddch(win, y, GET_POS_X1(x), ' '); \
  mvwaddch(win, y, GET_POS_X2(x), ' ');
```

Macro for inserting a square.

Parameters

win	The window in which the square will be installed
x	x position from the field
y	y position from the field

6.1.2.7 DRAW_POS_Y1

```
#define DRAW_POS_Y1 1
```

First y-axis cordinate for drawing

6.1.2.8 DRAW_POS_Y2

```
#define DRAW_POS_Y2 3
```

Second y-axis cordinate for drawing

6.1.2.9 DRAW_POS_Y3

```
#define DRAW_POS_Y3 5
```

Third y-axis cordinate for drawing

6.1.2.10 DRAW_POS_Y4

```
#define DRAW_POS_Y4 7
```

Fourth cordinate on the y-axis for drawing

6.1.2.11 GET_INFO_PRINT

```
#define GET_INFO_PRINT(
              index )
```

Value:

```
  (((index) == ITNAME)         ? "TETRIS"                                  \
   : ((index) == ISNAME)       ? "SNAKE"                                   \
   : ((index) == IHIGH_SCORE)  ? "HIGH SCORE: "                            \
   : ((index) == ISCORE)       ? "SCORE: "                                 \
   : ((index) == ILEVEL)       ? "LEVEL: "                                 \
   : ((index) == INEXT)        ? "NEXT"                                    \
   : ((index) == IPAUSE)       ? "PAUSE"                                   \
   : ((index) == INAMEBG)      ? "Brick Games"                             \
   : ((index) == IPRESSENTER)  ? "START Game press <Enter>"                \
   : ((index) == IPRESSESC)    ? "EXIT Game press <q>"                     \
   : ((index) == ILOSE)        ? "You lose"                                \
   : ((index) == IWIN)         ? "You Win"                                 \
   : ((index) == IEXITORREST)  ? "Press <q> to exit or <ENTER> to restart" \
   : ((index) == ISTART)       ? "START"                                   \
   : ((index) == IEXIT)        ? "EXIT"                                    \
   : ((index) == ISELECT)      ? "SELECT THE GAME AND THEN CLICK START"    \
   : ((index) == ISSTART)      ? "start"                                   \
   : ((index) == ISEND)        ? "end"                                     \
   : ((index) == ISNOT)        ? "not"                                     \
                               : NULL)
```

The macro returns the text according to the index, indices from the INDEX_TEXT_INFO enumeration.

6.1.2.12 GET_POS_X1

```
#define GET_POS_X1(
              x ) x * 2 + 1
```

Macro to return the first position of the square on the window.

Parameters

x	current position in the field array

Returns

The first position of the part square

6.1.2.13 GET_POS_X2

```
#define GET_POS_X2(
              x ) GET_POS_X1(x) + 1
```

Macro to return the second position of the square in the window.

Parameters

x	current position in the field array

Returns

second position of a part of the square

6.1.2.14 GETMAXWH

```
#define GETMAXWH(
              height,
              width ) getmaxyx(stdscr, height, width);
```

Macro to get terminal dimensions.

Parameters

width	variable to record the width
height	variable to record the height

6.1.2.15 HEIGHT_BOARD_NEXT

```
#define HEIGHT_BOARD_NEXT 6
```

Height of the `Board Next`

6.1.2.16 HEIGHT_WIN_EXIT

```
#define HEIGHT_WIN_EXIT 9
```

Height of the `exit window`

6.1.2.17 HEIGHT_WIN_PAUSE

```
#define HEIGHT_WIN_PAUSE 3
```

Height of the `pause window`

6.1.2.18 HEIGHT_WIN_START

```
#define HEIGHT_WIN_START 9
```

Height of the `start window`

6.1.2.19 MAX_CELLS

```
#define MAX_CELLS 200
```

Max count cells

6.1.2.20 SET_COLOR_PAIR

```
#define SET_COLOR_PAIR(
              win,
              index ) wbkgdset(win, COLOR_PAIR(index));
```

Macro for setting window color pair.

Parameters

win	The window in which the color will be set
index	Color pair index

6.1.2.21 SIZE_BOX

```
#define SIZE_BOX 1
```

Thickness box

6.1.2.22 START_X_BOAR

```
#define START_X_BOAR  3
```

Initial x-axis position for drawing the frame for the next shape

6.1.2.23 START_Y_BOAR

```
#define START_Y_BOAR  3
```

Initial y-axis position for drawing the frame for the next shape

6.1.2.24 WIDTH_BOARD_NEXT

```
#define WIDTH_BOARD_NEXT 15
```

Width of the `Board Next`

6.1.2.25 WIDTH_WIN_EXIT

```
#define WIDTH_WIN_EXIT 40
```

Width of the `exit window`

6.1.2.26 WIDTH_WIN_PAUSE

```
#define WIDTH_WIN_PAUSE 40
```

Width of the `pause window`

6.1.2.27 WIDTH_WIN_START

```
#define WIDTH_WIN_START 45
```

Width of the `start window`

6.1.2.28 WIN_EXIT

```
#define WIN_EXIT 2
```

Exit window code

6.1.2.29 WIN_FIELD

```
#define WIN_FIELD 3
```

Field window code

6.1.2.30 WIN_PAUSE

```
#define WIN_PAUSE 0
```

Pause window code

6.1.2.31 WIN_START

```
#define WIN_START 1
```

Start window code

6.1.2.32 WINDOW_FIELD_HEIGHT

```
#define WINDOW_FIELD_HEIGHT HFIELD + 2
```

Window field height

6.1.2.33 WINDOW_FIELD_WIDTH

```
#define WINDOW_FIELD_WIDTH WFIELD * 2 + 2
```

Window field width

Chapter 7

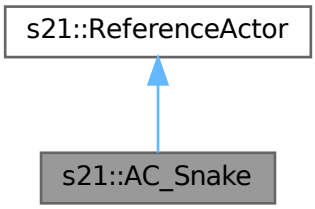
Class Documentation

7.1 s21::AC_Snake Class Reference

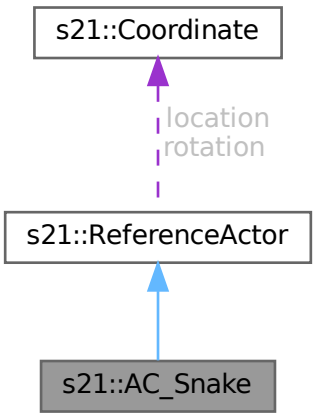
A class representing a snake.

`#include <AC_Snake.hpp>`

Inheritance diagram for s21::AC_Snake:



Collaboration diagram for s21::AC_Snake:



Public Member Functions

- AC_Snake (void)

  Construct a new ac snake::ac snake object.
- AC_Snake (int length, Coordinate location)

  Construct a new ac snake::ac snake object.
- ~**AC_Snake** ()

  Destroy the ac snake::ac snake object.
- void move () override

  Moves the snake.
- void setRotation (const int x, const int y) override

  Sets the rotation of the snake.
- void setRotation (const Coordinate coord) override

  Sets the rotation of the snake.
- void **IncreaseLength** (void)

  Increases the length of the snake by adding its current position to its body.
- const std::vector< Coordinate > & getSnake (void) const

  Returns the current state of the snake.
- int getLength (void) const

  Returns the current length of the snake.

Public Member Functions inherited from s21::ReferenceActor

- **ReferenceActor** (void)

  Construct a new Reference Actor:: Reference Actor object.
- ReferenceActor (const std::string &name_, const bool alive, const bool block, const Coordinate loc, const Coordinate rot)

  Constructor for the actor with initial values.
- virtual Coordinate getRotation (void)

Method to get the actor's rotation.

- virtual void setLocation (const int x, const int y)

  Method to set the actor's location.
- virtual void setLocation (const Coordinate coord)

  Method to set the actor's location.
- virtual Coordinate getLocation (void)

  Method to get the actor's location.
- virtual void setIsAlive (bool alive)

  Method to set whether the actor is alive or not.
- virtual bool getIsAlive (void)

  Method to get whether the actor is alive or not.
- virtual void setMovementBlocked (bool block)

  Method to set whether the actor's movement is blocked or not.
- virtual bool getMovementBlocked (void)

  Method to get whether the actor's movement is blocked or not.

Additional Inherited Members

Protected Attributes inherited from s21::ReferenceActor

- std::string name
- bool isAlive
- bool movementBlocked
- Coordinate location
- Coordinate rotation

7.1.1 Detailed Description

A class representing a snake.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 AC_Snake() [1/2]

```
s21::AC_Snake::AC_Snake (
            void  )
```

Construct a new ac snake::ac snake object.

The snake's body, which is a vector of Coordinate objects representing the snake's segments

7.1.2.2 AC_Snake() [2/2]

```
s21::AC_Snake::AC_Snake (
            int length,
            Coordinate location )
```

Construct a new ac snake::ac snake object.

Parameters

length	length The initial length of the snake
location	location The initial location of the snake

7.1.3 Member Function Documentation

7.1.3.1 getLength()

```
int s21::AC_Snake::getLength (
            void  ) const
```

Returns the current length of the snake.

Returns

The length of the snake, which is an integer representing the number of body segments

7.1.3.2 getSnake()

```
const std::vector< Coordinate > & s21::AC_Snake::getSnake (
            void  ) const
```

Returns the current state of the snake.

Returns

A constant reference to the snake's body, which is a vector of Coordinate objects

7.1.3.3 move()

```
void s21::AC_Snake::move (
            void  )  [override], [virtual]
```

Moves the snake.

Note

This method is only called if there is no blockage

Reimplemented from s21::ReferenceActor.

7.1.3.4 setRotation() [1/2]

```
void s21::AC_Snake::setRotation (
            const Coordinate coord )  [override], [virtual]
```

Sets the rotation of the snake.

Parameters

coord	A Coordinate object representing the new rotation

Reimplemented from s21::ReferenceActor.

7.1.3.5 setRotation() [2/2]

```
void s21::AC_Snake::setRotation (
            const int x,
            const int y )  [override], [virtual]
```

Sets the rotation of the snake.

Parameters

x	The x-coordinate of the rotation
y	The y-coordinate of the rotation

Reimplemented from s21::ReferenceActor.

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

- src/brick_game/snake/AC_Snake.hpp
- src/brick_game/snake/AC_Snake.cpp

7.2 s21::Coordinate Struct Reference

A structure representing a coordinate in a 2D space.

```
#include <ReferenceActor.hpp>
```

Public Member Functions

- bool operator== (const Coordinate &other) const

  Equality operator for comparing two coordinates.
- bool operator!= (const Coordinate &other) const

  Inequality operator for comparing two coordinates.
- Coordinate & operator+= (const Coordinate &other)

  Assignment operator for adding another coordinate to this one.

Public Attributes

- int x
- int y

7.2.1 Detailed Description

A structure representing a coordinate in a 2D space.

7.2.2 Member Function Documentation

7.2.2.1 operator"!=()

```
bool s21::Coordinate::operator!= (
            const Coordinate & other ) const  [inline]
```

Inequality operator for comparing two coordinates.

This operator checks if two coordinates have different x and/or y values.

Parameters

other	The other coordinate to compare with.

Returns

true if the coordinates are not equal, false otherwise.

7.2.2.2 operator+=()

```
Coordinate & s21::Coordinate::operator+= (
            const Coordinate & other )  [inline]
```

Assignment operator for adding another coordinate to this one.

This operator adds the x and y values of the other coordinate to this one.

Parameters

other	The other coordinate to add to this one.

Returns

A reference to this coordinate after the operation.

7.2.2.3 operator==()

```
bool s21::Coordinate::operator== (
            const Coordinate & other ) const  [inline]
```

Equality operator for comparing two coordinates.

This operator checks if two coordinates have the same x and y values.

Parameters

other	The other coordinate to compare with.

Returns

true if the coordinates are equal, false otherwise.

7.2.3 Member Data Documentation

7.2.3.1 x

```
int s21::Coordinate::x
```

The x-coordinate of the point.

7.2.3.2 y

```
int s21::Coordinate::y
```

The y-coordinate of the point.

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

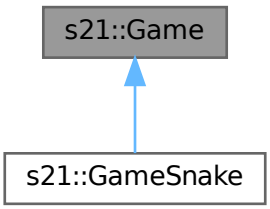
- src/brick_game/common/ReferenceActor/ReferenceActor.hpp

7.3 s21::Game Class Reference

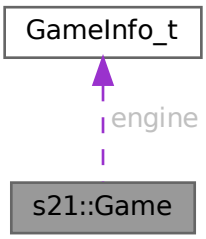
The base class for the game.

```
#include <ReferenceClassGame.hpp>
```

Inheritance diagram for s21::Game:



Collaboration diagram for s21::Game:



Public Member Functions

- **Game** (void)

  Construct a new Game object.
- Game (const std::string &filename)

  Construct a new Game object.
- virtual void userInput (UserAction_t action, bool hold)=0

  Processes user input based on the current game state.
- virtual GameInfo_t updateCurrentState ()

  Updates and retrieves the current game state information.
- virtual void cleanField ()

  Method to clean the game field.
- virtual void save ()

  Method to save the game high_score.
- virtual void readSave ()

  Method to read saved game high_score.

Protected Attributes

- GameInfo_t engine
- std::string filenameSaving

7.3.1 Detailed Description

The base class for the game.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Game()

```
s21::Game::Game (
            const std::string & filename )  [inline], [explicit]
```

Construct a new Game object.

Parameters

filename	fillename BD

7.3.3 Member Function Documentation

7.3.3.1 cleanField()

```
virtual void s21::Game::cleanField ( )  [inline], [virtual]
```

Method to clean the game field.

This method cleans the game field by setting all values to zero.

7.3.3.2 readSave()

```
virtual void s21::Game::readSave ( )  [inline], [virtual]
```

Method to read saved game high_score.

This method reads saved game high_score from a file and updates the high score accordingly.

7.3.3.3 save()

```
virtual void s21::Game::save ( )  [inline], [virtual]
```

Method to save the game high_score.

This method saves the game high_score to a file.

7.3.3.4 updateCurrentState()

```
virtual GameInfo_t s21::Game::updateCurrentState ( )  [inline], [virtual]
```

Updates and retrieves the current game state information.

This function updates and retrieves the current game state information, including the field, next piece, score, level, speed, and pause state.

Returns

GameInfo_t The updated game state information.

Reimplemented in s21::GameSnake.

7.3.3.5 userInput()

```
virtual void s21::Game::userInput (
            UserAction_t action,
            bool hold )  [pure virtual]
```

Processes user input based on the current game state.

Parameters

action	The user action to process.
hold	Indicates whether the action is being held.

Implemented in s21::GameSnake.

7.3.4 Member Data Documentation

7.3.4.1 engine

```
GameInfo_t s21::Game::engine  [protected]
```

Struct Gameinfo

7.3.4.2 filenameSaving

```
std::string s21::Game::filenameSaving  [protected]
```

filename BD

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

- src/brick_game/common/ReferenceGame/ReferenceClassGame.hpp

7.4 GameInfo_t Struct Reference

A structure holding information about a game.

```
#include <Info.h>
```

Public Attributes

- int ** field
- int ** next
- int score
- int high_score
- int level
- int speed
- int pause

7.4.1 Detailed Description

A structure holding information about a game.

7.4.2 Member Data Documentation

7.4.2.1 field

```
int** GameInfo_t::field
```

A 2D array of pointers to game field.

7.4.2.2 high_score

```
int GameInfo_t::high_score
```

The high score of the game.

7.4.2.3 level

```
int GameInfo_t::level
```

The current level of the game.

7.4.2.4 next

```
int** GameInfo_t::next
```

A 2D array of pointers to next.

7.4.2.5 pause

```
int GameInfo_t::pause
```

A flag indicating whether the game is paused or not.

7.4.2.6 score

```
int GameInfo_t::score
```

The current score of the game.

7.4.2.7 speed

```
int GameInfo_t::speed
```

The current speed of the game.

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

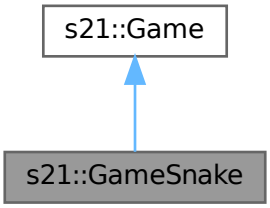
- src/brick_game/common/Info/Info.h

7.5 s21::GameSnake Class Reference

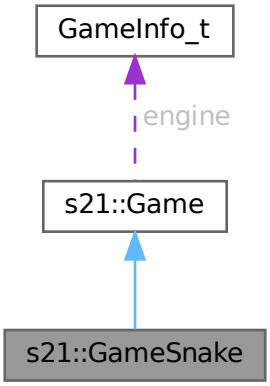
The GameSnake class represents the snake game.

`#include <GameSnake.hpp>`

Inheritance diagram for s21::GameSnake:



Collaboration diagram for s21::GameSnake:



Public Member Functions

- GameSnake (void)

  Construct a new Game Snake:: Game Snake object.
- ~**GameSnake** ()

  Destroy the Game Snake:: Game Snake object.
- GameState_t getState () const

  Gets the current state of the game.
- void setState (const GameState_t sstate)

Sets the current state of the game to a new value.

- void reset (const GameState_t reset_state)

  Resets the game state to a specific state.
- void userInput (UserAction_t action, bool hold) override

  Processes user input based on the current game state.
- GameInfo_t updateCurrentState () override

  Updates and retrieves the current game state information.

Public Member Functions inherited from s21::Game

- **Game** (void)

  Construct a new Game object.
- Game (const std::string &filename)

  Construct a new Game object.
- virtual void cleanField ()

  Method to clean the game field.
- virtual void save ()

  Method to save the game high_score.
- virtual void readSave ()

  Method to read saved game high_score.

Additional Inherited Members

Protected Attributes inherited from s21::Game

- GameInfo_t engine
- std::string filenameSaving

7.5.1 Detailed Description

The GameSnake class represents the snake game.

This class extends the Game class and provides functionality for the snake game.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 GameSnake()

```
s21::GameSnake::GameSnake (
            void  )
```

Construct a new Game Snake:: Game Snake object.

The current state of the game.

7.5.3 Member Function Documentation

7.5.3.1 getState()

```
GameState_t s21::GameSnake::getState ( ) const
```

Gets the current state of the game.

Returns

GameState_t

7.5.3.2 reset()

```
void s21::GameSnake::reset (
            const GameState_t reset_state )
```

Resets the game state to a specific state.

Parameters

reset_state	The state to which the game should be moved

7.5.3.3 setState()

```
void s21::GameSnake::setState (
            const GameState_t sstate )
```

Sets the current state of the game to a new value.

Parameters

sstate	

7.5.3.4 updateCurrentState()

```
GameInfo_t s21::GameSnake::updateCurrentState ( )  [override], [virtual]
```

Updates and retrieves the current game state information.

This function updates and retrieves the current game state information, including the field, next piece, score, level, speed, and pause state.

Returns

GameInfo_t The updated game state information.

Reimplemented from s21::Game.

7.5.3.5 userInput()

```
void s21::GameSnake::userInput (
            UserAction_t action,
            bool hold )  [override], [virtual]
```

Processes user input based on the current game state.

Parameters

action	The user action to process.
hold	Indicates whether the action is being held.

Implements s21::Game.

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

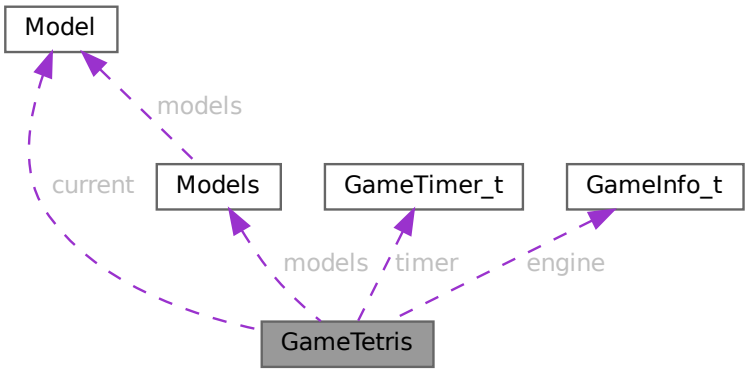
- src/brick_game/snake/GameSnake.hpp
- src/brick_game/snake/GameSnake.cpp

7.6 GameTetris Struct Reference

Structure representing game parameters.

`#include <GameTetris.h>`

Collaboration diagram for GameTetris:



Public Attributes

- Models models
- Model current
- int current_color
- size_t index_next
- GameState_t state
- GameTimer_t timer
- GameInfo_t engine

7.6.1 Detailed Description

Structure representing game parameters.

7.6.2 Member Data Documentation

7.6.2.1 current

```
Model GameTetris::current
```

Current model in the game.

7.6.2.2 current_color

```
int GameTetris::current_color
```

Current model color

7.6.2.3 engine

```
GameInfo_t GameTetris::engine
```

Information about the game.

7.6.2.4 index_next

```
size_t GameTetris::index_next
```

Next model index

7.6.2.5 models

```
Models GameTetris::models
```

Collection of available models.

7.6.2.6 state

```
GameState_t GameTetris::state
```

Current state of the game.

7.6.2.7 timer

```
GameTimer_t GameTetris::timer
```

Timer for shifting

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

- src/brick_game/tetris/GameTetris.h

7.7 s21::GameTimer Class Reference

Game timer class.

```
#include <GameTimer.hpp>
```

Public Types

- using **type_time** = std::chrono::time_point< std::chrono::steady_clock >

  Type time - std::chrono::time_point<std::chrono::steady_clock>

Public Member Functions

- **GameTimer** (void)

  Construct a new Game Timer:: Game Timer object.
- void **updateStartTime** (void)

  Updates the start time of the timer.
- void **updateEndTime** (void)

  Updates the end time of the timer.
- void setDuration (const std::chrono::milliseconds time)

  Sets the duration of the timer.
- const std::chrono::milliseconds getDuration (void) const

  Gets the duration of the timer.
- void setActive (bool active_)

  Sets the active of the timer.
- bool getActive (void) const

  Gets the duration of the timer.

7.7.1 Detailed Description

Game timer class.

7.7.2 Member Function Documentation

7.7.2.1 getActive()

```
bool s21::GameTimer::getActive (
            void  ) const
```

Gets the duration of the timer.

Returns

true

false

7.7.2.2 getDuration()

```
const std::chrono::milliseconds s21::GameTimer::getDuration (
            void  ) const
```

Gets the duration of the timer.

Returns

const std::chrono::milliseconds

7.7.2.3 setActive()

```
void s21::GameTimer::setActive (
            bool active_ )
```

Sets the active of the timer.

Parameters

*active_*	Status active

7.7.2.4 setDuration()

```
void s21::GameTimer::setDuration (
            const std::chrono::milliseconds time )
```

Sets the duration of the timer.

Parameters

time	duration

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

- src/brick_game/common/Timer/GameTimer.hpp
- src/brick_game/common/Timer/GameTimer.cpp

7.8 GameTimer_t Struct Reference

Structure representing a game timer.

`#include <Timer.h>`

Public Attributes

- bool indicator
- pthread_t * thread

7.8.1 Detailed Description

Structure representing a game timer.

7.8.2 Member Data Documentation

7.8.2.1 indicator

```
bool GameTimer_t::indicator
```

State indicator of the timer.

7.8.2.2 thread

```
pthread_t* GameTimer_t::thread
```

Pointer to the timer thread.

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

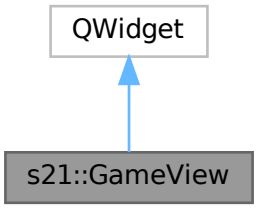
- src/brick_game/tetris/Timer/Timer.h

7.9 s21::GameView Class Reference

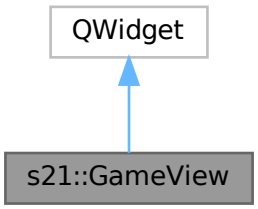
A class for representing a game view.

`#include <GameView.hpp>`

Inheritance diagram for s21::GameView:



Collaboration diagram for s21::GameView:



Signals

- void userActionReceived (const QKeyEvent *event)

  Signals a user action received.

Public Member Functions

- GameView (QWidget *parent=nullptr)

  The default constructor for GameView.
- void updateGameInfo (const GameInfo_t &info, const QString addition="not")

  Updates the game information.
- void setGameSelected (const QString &gameName)

  Sets the selected game.
- QString getGameSelected (void)

  Gets the selected game.
- void keyPressEvent (QKeyEvent *event) override

  Handles a key press event.

Protected Member Functions

- void paintEvent (QPaintEvent *event) override

  Paints the game view.
- void drawInfo (QPainter &painter, QBrush &brush)

  Draws the game information.
- void drawField (QPainter &painter, QBrush &brush)

  Draws the game field.
- void drawNext (QPainter &painter, QBrush &brush)

  Draws the next section.
- void **initSetting** ()

  Initializes the game setting.
- QColor getColor (const int code)

  Gets the color of a specific code.
- void drawInfoAddition (QPainter &painter, const int infoOffsetY)

  Draws the game information addition.
- void drawInfoImage (QPainter &painter, QRect &infoRect, const int infoOffsetY)

  Draws the game information image.

7.9.1 Detailed Description

A class for representing a game view.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 GameView()

```
s21::GameView::GameView (
            QWidget * parent = nullptr )  [explicit]
```

The default constructor for GameView.

Parameters

parent	The parent widget.

7.9.3 Member Function Documentation

7.9.3.1 drawField()

```
void s21::GameView::drawField (
            QPainter & painter,
            QBrush & brush )  [protected]
```

Draws the game field.

Parameters

painter	The painter.
brush	The brush.

7.9.3.2 drawInfo()

```
void s21::GameView::drawInfo (
            QPainter & painter,
            QBrush & brush )  [protected]
```

Draws the game information.

Parameters

painter	The painter.
brush	The brush.

7.9.3.3 drawInfoAddition()

```
void s21::GameView::drawInfoAddition (
            QPainter & painter,
            const int infoOffsetY )  [protected]
```

Draws the game information addition.

Parameters

painter	The painter.
infoOffsetY	The offset y-coordinate for the information addition.

7.9.3.4 drawInfoImage()

```
void s21::GameView::drawInfoImage (
            QPainter & painter,
            QRect & infoRect,
            const int infoOffsetY )  [protected]
```

Draws the game information image.

Parameters

painter	The painter.
infoRect	The rectangle for the information image.
infoOffsetY	The offset y-coordinate for the information image.

7.9.3.5 drawNext()

```
void s21::GameView::drawNext (
            QPainter & painter,
            QBrush & brush )  [protected]
```

Draws the next section.

Parameters

painter	The painter.
brush	The brush.

7.9.3.6 getColor()

```
QColor s21::GameView::getColor (
            const int code )  [protected]
```

Gets the color of a specific code.

Parameters

code	The code for the color.

Returns

The color corresponding to the code.

7.9.3.7 getGameSelected()

```
QString s21::GameView::getGameSelected (
            void  )
```

Gets the selected game.

Returns

The name of the selected game.

7.9.3.8 keyPressEvent()

```
void s21::GameView::keyPressEvent (
            QKeyEvent * event )  [override]
```

Handles a key press event.

Parameters

event	The key press event.

7.9.3.9 paintEvent()

```
void s21::GameView::paintEvent (
            QPaintEvent * event )  [override], [protected]
```

Paints the game view.

Parameters

event	The paint event.

7.9.3.10 setGameSelected()

```
void s21::GameView::setGameSelected (
            const QString & gameName )
```

Sets the selected game.

Parameters

gameName	The name of the selected game.

7.9.3.11 updateGameInfo()

```
void s21::GameView::updateGameInfo (
            const GameInfo_t & info,
            const QString addition = "not" )
```

Updates the game information.

Parameters

info	The game information.
addition	The addition to the game information.

7.9.3.12 userActionReceived

```
void s21::GameView::userActionReceived (
            const QKeyEvent * event )  [signal]
```

Signals a user action received.

Parameters

event	The user action event.

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

- src/gui/desktop/View/GameView.hpp
- src/gui/desktop/View/GameView.cpp
- src/gui/desktop/View/GameViewDraw.cpp

7.10 GameWindows Struct Reference

Structure representing game windows.

`#include <GameUI.h>`

Public Attributes

- bool infoDraw
- WINDOW * fieldw
- WINDOW * infow

7.10.1 Detailed Description

Structure representing game windows.

This structure contains pointers to the game field window and the game info window, as well as a boolean flag indicating whether the info window has been drawn.

7.10.2 Member Data Documentation

7.10.2.1 fieldw

`WINDOW* GameWindows::fieldw`

Pointer to the game field window.

7.10.2.2 infoDraw

`bool GameWindows::infoDraw`

Flag indicating whether the info window has been drawn.

7.10.2.3 infow

```
WINDOW* GameWindows::infow
```

Pointer to the game info window.

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

- src/gui/cli/UI/GameUI.h

7.11 Mainwindow Struct Reference

A structure holding information about a main window.

```
#include <MainWindow.h>
```

Public Attributes

- char title [BUFF_STR]
- char description [BUFF_STR]
- size_t sgame
- size_t cgame
- char games [BASE_COUNT][BUFF_STR]
- size_t sinfo
- size_t cinfo
- char info [BASE_COUNT][BUFF_STR]
- bool vertical
- bool centered

7.11.1 Detailed Description

A structure holding information about a main window.

7.11.2 Member Data Documentation

7.11.2.1 centered

```
bool Mainwindow::centered
```

A flag indicating whether the window is centered or not.

7.11.2.2 cgame

```
size_t Mainwindow::cgame
```

The count of games in the array.

7.11.2.3 cinfo

```
size_t Mainwindow::cinfo
```

The count of information items in the array.

7.11.2.4 description

```
char Mainwindow::description[BUFF_STR]
```

A character array holding a description of the main window.

7.11.2.5 games

```
char Mainwindow::games[BASE_COUNT][BUFF_STR]
```

A character array holding the names of games.

7.11.2.6 info

```
char Mainwindow::info[BASE_COUNT][BUFF_STR]
```

A character array holding the information items.

7.11.2.7 sgame

```
size_t Mainwindow::sgame
```

The select of a game in the array.

7.11.2.8 sinfo

```
size_t Mainwindow::sinfo
```

The select of an information item in the array.

7.11.2.9 title

```
char Mainwindow::title[BUFF_STR]
```

A character array holding the title of the main window.

7.11.2.10 vertical

```
bool Mainwindow::vertical
```

A flag indicating whether the window is displayed vertically or not.

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

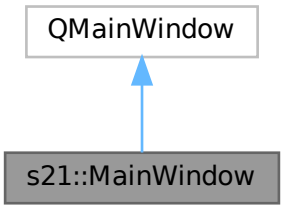
- src/gui/cli/MainWindow.h

7.12 s21::MainWindow Class Reference

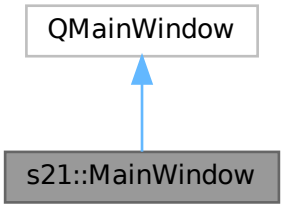
The main window class for the game.

```
#include <mainwindow.hpp>
```

Inheritance diagram for s21::MainWindow:



Collaboration diagram for s21::MainWindow:



Public Member Functions

- MainWindow (QWidget *parent=nullptr)

  The constructor for the main window.
- ~**MainWindow** ()

  The destructor for the main window.

7.12.1 Detailed Description

The main window class for the game.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 MainWindow()

```
s21::MainWindow::MainWindow (
            QWidget * parent = nullptr )  [explicit]
```

The constructor for the main window.

Parameters

parent	The parent widget.

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

- src/gui/desktop/mainwindow.hpp
- src/gui/desktop/mainwindow.cpp

7.13 Model Struct Reference

Structure representing a game model.

```
#include <GameModel.h>
```

Public Attributes

- size_t rows
- size_t cols
- int ** model_
- int position [SIZE_COORD]
- int center [SIZE_COORD]

7.13.1 Detailed Description

Structure representing a game model.

This structure represents a game model, which consists of rows and columns defining its size, a 2D array storing the model's shape, an array storing the model's position, and another array storing the coordinates of the model's center.

7.13.2 Member Data Documentation

7.13.2.1 center

```
int Model::center[SIZE_COORD]
```

Array storing the coordinates of the model's center.

7.13.2.2 cols

```
size_t Model::cols
```

Number of columns in the model.

7.13.2.3 model_

```
int** Model::model_
```

2D array representing the model's shape.

7.13.2.4 position

```
int Model::position[SIZE_COORD]
```

Array storing the model's position.

7.13.2.5 rows

```
size_t Model::rows
```

Number of rows in the model.

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

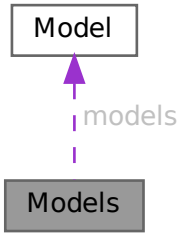
- src/brick_game/tetris/Model/GameModel.h

7.14 Models Struct Reference

Structure representing a collection of game models.

```
#include <GameModel.h>
```

Collaboration diagram for Models:



Public Attributes

- Model * models
- size_t count

7.14.1 Detailed Description

Structure representing a collection of game models.

This structure represents a collection of game models, containing an array of `Model` structures and a count indicating the number of models in the collection.

7.14.2 Member Data Documentation

7.14.2.1 count

```
size_t Models::count
```

Number of models in the collection.

7.14.2.2 models

```
Model* Models::models
```

Pointer to an array of `Model` structures representing the game models.

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

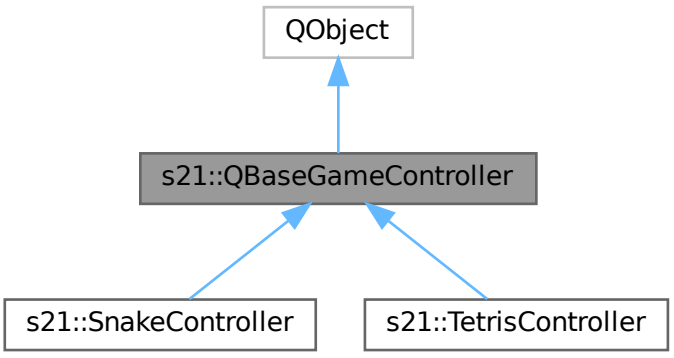
- src/brick_game/tetris/Model/GameModel.h

7.15 s21::QBaseGameController Class Reference

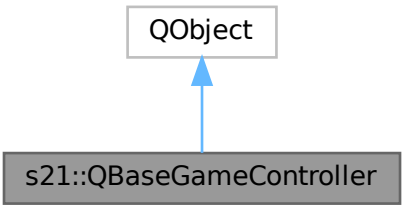
A base class for game controllers.

```
#include <QBaseController.hpp>
```

Inheritance diagram for s21::QBaseGameController:



Collaboration diagram for s21::QBaseGameController:



Public Slots

- virtual void **updateView** ()=0

  Updates the game view.
- virtual void **sendInputSignal** ()=0

  Sends an input signal.
- virtual void onUserActionReceived (const QKeyEvent ∗event)=0

  Handles a user action received event.

Signals

- void **finished** ()

  Signals that the game controller has finished.

Public Member Functions

- QBaseGameController (QObject ∗parent=nullptr)

  The default constructor for QBaseGameController.
- virtual ∼**QBaseGameController** ()=default

  The destructor for QBaseGameController.
- virtual void run ()=0

  Runs the game controller.
- virtual void stop ()=0

  Stops the game controller.

Protected Attributes

- QTimer ∗ timer_input
- QTimer ∗ timer_output

7.15.1 Detailed Description

A base class for game controllers.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 QBaseGameController()

```
s21::QBaseGameController::QBaseGameController (
            QObject * parent = nullptr )  [inline], [explicit]
```

The default constructor for QBaseGameController.

Parameters

parent	The parent object.

7.15.3 Member Function Documentation

7.15.3.1 onUserActionReceived

```
virtual void s21::QBaseGameController::onUserActionReceived (
            const QKeyEvent * event )  [pure virtual], [slot]
```

Handles a user action received event.

Parameters

event	The user action received event.

7.15.3.2 run()

```
virtual void s21::QBaseGameController::run ( )  [pure virtual]
```

Runs the game controller.

Implemented in s21::SnakeController, and s21::TetrisController.

7.15.3.3 stop()

```
virtual void s21::QBaseGameController::stop ( )  [pure virtual]
```

Stops the game controller.

Implemented in s21::SnakeController, and s21::TetrisController.

7.15.4 Member Data Documentation

7.15.4.1 timer_input

```
QTimer* s21::QBaseGameController::timer_input  [protected]
```

Timer user input update

7.15.4.2 timer_output

```
QTimer* s21::QBaseGameController::timer_output  [protected]
```

Timer interface update

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

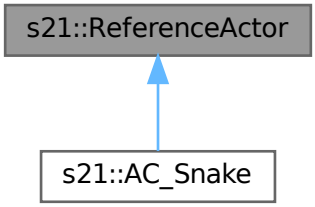
- src/gui/desktop/Controller/QBaseController.hpp

7.16 s21::ReferenceActor Class Reference

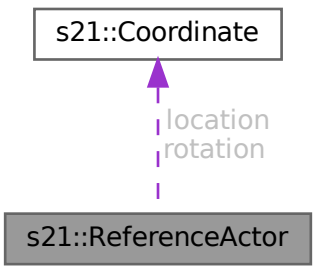
A base class for actors in a game.

```
#include <ReferenceActor.hpp>
```

Inheritance diagram for s21::ReferenceActor:



Collaboration diagram for s21::ReferenceActor:



Public Member Functions

- **ReferenceActor** (void)

  Construct a new Reference Actor:: Reference Actor object.
- ReferenceActor (const std::string &name_, const bool alive, const bool block, const Coordinate loc, const Coordinate rot)

  Constructor for the actor with initial values.
- virtual void move (void)

  Method to move the actor.
- virtual void setRotation (const int x, const int y)

  Method to set the actor's rotation.
- virtual void setRotation (const Coordinate coord)

  Method to set the actor's rotation.
- virtual Coordinate getRotation (void)

  Method to get the actor's rotation.
- virtual void setLocation (const int x, const int y)

  Method to set the actor's location.
- virtual void setLocation (const Coordinate coord)

  Method to set the actor's location.
- virtual Coordinate getLocation (void)

  Method to get the actor's location.
- virtual void setIsAlive (bool alive)

  Method to set whether the actor is alive or not.
- virtual bool getIsAlive (void)

  Method to get whether the actor is alive or not.
- virtual void setMovementBlocked (bool block)

  Method to set whether the actor's movement is blocked or not.
- virtual bool getMovementBlocked (void)

  Method to get whether the actor's movement is blocked or not.

Protected Attributes

- std::string name
- bool isAlive
- bool movementBlocked
- Coordinate location
- Coordinate rotation

7.16.1 Detailed Description

A base class for actors in a game.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 ReferenceActor()

```
s21::ReferenceActor::ReferenceActor (
            const std::string & name_,
            const bool alive,
            const bool block,
            const Coordinate loc,
            const Coordinate rot )
```

Constructor for the actor with initial values.

This constructor initializes the actor with a given name, alive status, movement blockage, location, and rotation.

Parameters

name↩ _	The name of the actor.
alive	Whether the actor is alive.
block	Whether the actor's movement is blocked.
loc	The actor's initial location.
rot	The actor's initial rotation.

7.16.3 Member Function Documentation

7.16.3.1 getIsAlive()

```
bool s21::ReferenceActor::getIsAlive (
            void  )  [virtual]
```

Method to get whether the actor is alive or not.

This method returns whether the actor is currently alive or not as a boolean value.

Returns

Whether the actor is alive or not.

7.16.3.2 getLocation()

```
Coordinate s21::ReferenceActor::getLocation (
            void  )  [virtual]
```

Method to get the actor's location.

This method returns the actor's current location as a Coordinate object.

Returns

The actor's current location.

7.16.3.3 getMovementBlocked()

```
bool s21::ReferenceActor::getMovementBlocked (
            void  )  [virtual]
```

Method to get whether the actor's movement is blocked or not.

This method returns whether the actor's movement is currently blocked or not as a boolean value.

Returns

Whether the actor's movement is blocked or not

7.16.3.4 getRotation()

```
Coordinate s21::ReferenceActor::getRotation (
            void  )  [virtual]
```

Method to get the actor's rotation.

This method returns the actor's current rotation as a Coordinate object.

Returns

The actor's current rotation.

7.16.3.5 move()

```
void s21::ReferenceActor::move (
            void  )  [virtual]
```

Method to move the actor.

This method is pure virtual and must be implemented by derived classes.

Reimplemented in s21::AC_Snake.

7.16.3.6 setIsAlive()

```
void s21::ReferenceActor::setIsAlive (
            bool alive )  [virtual]
```

Method to set whether the actor is alive or not.

This method sets whether the actor is alive or not based on a boolean value.

Parameters

alive	Whether the actor should be considered alive or not.

7.16.3.7 setLocation() [1/2]

```
void s21::ReferenceActor::setLocation (
            const Coordinate coord )  [virtual]
```

Method to set the actor's location.

This method sets the actor's location using a Coordinate object.

Parameters

coord	The new location coordinate.

7.16.3.8 setLocation() [2/2]

```
void s21::ReferenceActor::setLocation (
            const int x,
            const int y )  [virtual]
```

Method to set the actor's location.

This method sets the actor's location using integer coordinates (x, y).

Parameters

x	The x-coordinate of the new location.
y	The y-coordinate of the new location.

7.16.3.9 setMovementBlocked()

```
void s21::ReferenceActor::setMovementBlocked (
            bool block )  [virtual]
```

Method to set whether the actor's movement is blocked or not.

This method sets whether the actor's movement is blocked or not based on a boolean value.

Parameters

block	Whether the actor's movement should be blocked or not.

7.16.3.10 setRotation() [1/2]

```
void s21::ReferenceActor::setRotation (
            const Coordinate coord )  [virtual]
```

Method to set the actor's rotation.

This method sets the actor's rotation using a Coordinate object.

Parameters

coord	The new rotation coordinate.

Reimplemented in s21::AC_Snake.

7.16.3.11 setRotation() [2/2]

```
void s21::ReferenceActor::setRotation (
            const int x,
            const int y )  [virtual]
```

Method to set the actor's rotation.

This method sets the actor's rotation using integer coordinates (x, y).

Parameters

x	The x-coordinate of the new rotation.
y	The y-coordinate of the new rotation.

Reimplemented in s21::AC_Snake.

7.16.4 Member Data Documentation

7.16.4.1 isAlive

```
bool s21::ReferenceActor::isAlive  [protected]
```

Actor isAlive

7.16.4.2 location

```
Coordinate s21::ReferenceActor::location  [protected]
```

Actor location

7.16.4.3 movementBlocked

```
bool s21::ReferenceActor::movementBlocked  [protected]
```

Variable movement blocked

7.16.4.4 name

```
std::string s21::ReferenceActor::name  [protected]
```

Actor name

7.16.4.5 rotation

```
Coordinate s21::ReferenceActor::rotation  [protected]
```

Actor rotation

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

- src/brick_game/common/ReferenceActor/ReferenceActor.hpp
- src/brick_game/common/ReferenceActor/ReferenceActor.cpp

7.17 SizeText Struct Reference

Structure for displaying terminal information.

`#include <GameUI.h>`

Public Attributes

- int height
- int width
- int y
- int x

7.17.1 Detailed Description

Structure for displaying terminal information.

7.17.2 Member Data Documentation

7.17.2.1 height

`int SizeText::height`

Terminal height

7.17.2.2 width

`int SizeText::width`

Terminal width

7.17.2.3 x

`int SizeText::x`

Center on `x`

7.17.2.4 y

`int SizeText::y`

Center on `y`

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

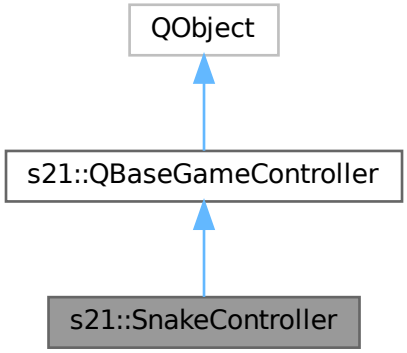
- src/gui/cli/UI/GameUI.h

7.18 s21::SnakeController Class Reference

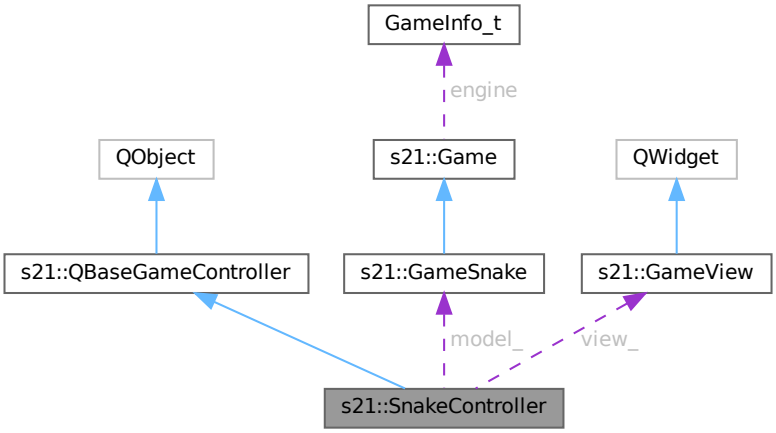
A controller for the Snake game.

`#include <SnakeController.hpp>`

Inheritance diagram for s21::SnakeController:



Collaboration diagram for s21::SnakeController:



Public Slots

- void **updateView** () override

  Updates the game view.
- void **sendInputSignal** () override

  Sends an input signal to the game controller.
- void **onUserActionReceived** (const QKeyEvent ∗event) override

  Handles a user action received event for the Snake game controller.

Public Slots inherited from s21::QBaseGameController

- virtual void **updateView** ()=0

  Updates the game view.
- virtual void **sendInputSignal** ()=0

  Sends an input signal.
- virtual void onUserActionReceived (const QKeyEvent ∗event)=0

  Handles a user action received event.

Public Member Functions

- SnakeController (GameSnake ∗model, GameView ∗view, QObject ∗parent=nullptr)

  The constructor for SnakeController.
- void run () override

  Runs the Snake game controller.
- void stop () override

  Stops the Snake game controller.

Public Member Functions inherited from s21::QBaseGameController

- QBaseGameController (QObject ∗parent=nullptr)

  The default constructor for QBaseGameController.
- virtual ∼**QBaseGameController** ()=default

  The destructor for QBaseGameController.

Protected Attributes

- GameSnake ∗ model_
- GameView ∗ view_
- UserAction_t action_

Protected Attributes inherited from s21::QBaseGameController

- QTimer ∗ timer_input
- QTimer ∗ timer_output

Additional Inherited Members

Signals inherited from s21::QBaseGameController

- void **finished** ()

  Signals that the game controller has finished.

7.18.1 Detailed Description

A controller for the Snake game.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 SnakeController()

```
s21::SnakeController::SnakeController (
            GameSnake * model,
            GameView * view,
            QObject * parent = nullptr )  [explicit]
```

The constructor for SnakeController.

Parameters

model	The game model.
view	The game view.
parent	The parent object.

7.18.3 Member Function Documentation

7.18.3.1 run()

```
void s21::SnakeController::run ( )  [override], [virtual]
```

Runs the Snake game controller.

Implements s21::QBaseGameController.

7.18.3.2 stop()

```
void s21::SnakeController::stop ( )  [override], [virtual]
```

Stops the Snake game controller.

Implements s21::QBaseGameController.

7.18.4 Member Data Documentation

7.18.4.1 action_

```
UserAction_t s21::SnakeController::action_  [protected]
```

User input action

7.18.4.2 model_

```
GameSnake* s21::SnakeController::model_  [protected]
```

Game model

7.18.4.3 view_

```
GameView* s21::SnakeController::view_  [protected]
```

Game view

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

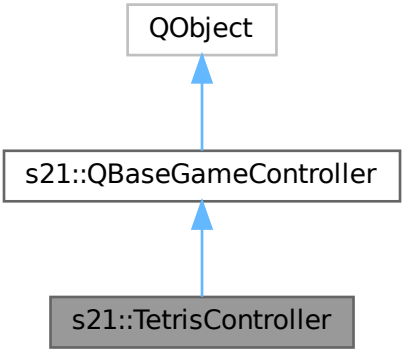
- src/gui/desktop/Controller/SnakeController.hpp
- src/gui/desktop/Controller/SnakeController.cpp

7.19 s21::TetrisController Class Reference

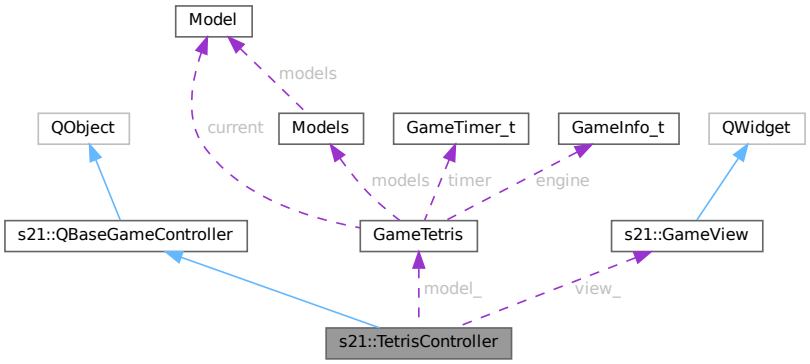
A controller for the Tetris game.

`#include <TetrisController.hpp>`

Inheritance diagram for s21::TetrisController:



Collaboration diagram for s21::TetrisController:



Public Slots

- void **updateView** () override

  Updates the game view.
- void **sendInputSignal** () override

  Sends an input signal to the game controller.
- void **onUserActionReceived** (const QKeyEvent ∗event) override

  Handles a user action received event for the Tetris game controller.

Public Slots inherited from s21::QBaseGameController

- virtual void **updateView** ()=0

  Updates the game view.
- virtual void **sendInputSignal** ()=0

  Sends an input signal.
- virtual void onUserActionReceived (const QKeyEvent ∗event)=0

  Handles a user action received event.

Public Member Functions

- TetrisController (GameTetris ∗model, GameView ∗view, QObject ∗parent=nullptr)

  The constructor for TetrisController.
- void run () override

  Runs the Tetris game controller.
- void stop () override

  Stops the Tetris game controller.

Public Member Functions inherited from s21::QBaseGameController

- QBaseGameController (QObject ∗parent=nullptr)

  The default constructor for QBaseGameController.
- virtual ∼**QBaseGameController** ()=default

  The destructor for QBaseGameController.

Protected Attributes

- GameTetris ∗ model_
- GameView ∗ view_
- UserAction_t action_

Protected Attributes inherited from s21::QBaseGameController

- QTimer ∗ timer_input
- QTimer ∗ timer_output

Additional Inherited Members

Signals inherited from s21::QBaseGameController

- void **finished** ()

  Signals that the game controller has finished.

7.19.1 Detailed Description

A controller for the Tetris game.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 TetrisController()

```
s21::TetrisController::TetrisController (
            GameTetris * model,
            GameView * view,
            QObject * parent = nullptr )  [explicit]
```

The constructor for TetrisController.

Parameters

model	The game model.
view	The game view.
parent	The parent object.

7.19.3 Member Function Documentation

7.19.3.1 run()

```
void s21::TetrisController::run ( )  [override], [virtual]
```

Runs the Tetris game controller.

Implements s21::QBaseGameController.

7.19.3.2 stop()

```
void s21::TetrisController::stop ( )  [override], [virtual]
```

Stops the Tetris game controller.

Implements s21::QBaseGameController.

7.19.4 Member Data Documentation

7.19.4.1 action_

```
UserAction_t s21::TetrisController::action_  [protected]
```

User input action

7.19.4.2 model_

```
GameTetris* s21::TetrisController::model_  [protected]
```

Game model

7.19.4.3 view_

```
GameView* s21::TetrisController::view_  [protected]
```

Game view

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

- src/gui/desktop/Controller/TetrisController.hpp
- src/gui/desktop/Controller/TetrisController.cpp

7.20 s21::UserInterface_t Struct Reference

A structure to hold the properties of the user interface.

`#include <GameView.hpp>`

Public Attributes

- int offsetY
- int offsetX
- int fieldOffsetX
- int fieldOffsetY
- int line
- int nextOffsetX
- int nextOffsetY
- int WWindow
- int HWindow

7.20.1 Detailed Description

A structure to hold the properties of the user interface.

7.20.2 Member Data Documentation

7.20.2.1 fieldOffsetX

`int s21::UserInterface_t::fieldOffsetX`

The x-coordinate offset for the field.

7.20.2.2 fieldOffsetY

`int s21::UserInterface_t::fieldOffsetY`

The y-coordinate offset for the field.

7.20.2.3 HWindow

`int s21::UserInterface_t::HWindow`

The height of the window.

7.20.2.4 line

`int s21::UserInterface_t::line`

The current line number.

7.20.2.5 nextOffsetX

```
int s21::UserInterface_t::nextOffsetX
```

The x-coordinate offset for the next section.

7.20.2.6 nextOffsetY

```
int s21::UserInterface_t::nextOffsetY
```

The y-coordinate offset for the next section.

7.20.2.7 offsetX

```
int s21::UserInterface_t::offsetX
```

The offset x-coordinate.

7.20.2.8 offsetY

```
int s21::UserInterface_t::offsetY
```

The offset y-coordinate.

7.20.2.9 WWindow

```
int s21::UserInterface_t::WWindow
```

The width of the window.

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

- src/gui/desktop/View/GameView.hpp

Chapter 8

File Documentation

8.1 src/brick_game/common/Action/Action.h File Reference

Header file with the main structure of the action.

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

Enumerations

- enum UserAction_t {
  Start , Pause , Terminate , Left ,
  Right , Up , Down , Action }

  Enumeration representing user actions in the game.

8.1.1 Detailed Description

Header file with the main structure of the action.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

Copyright (c) 2024

8.1.2 Enumeration Type Documentation

8.1.2.1 UserAction_t

```
enum UserAction_t
```

Enumeration representing user actions in the game.

Enumerator

Start	Start action
Pause	Pause action
Terminate	Terminate action
Left	Left arrow key action
Right	Right arrow key action
Up	Up arrow key action
Down	Down arrow key action
Action	Action key action

8.2 Action.h

Go to the documentation of this file.

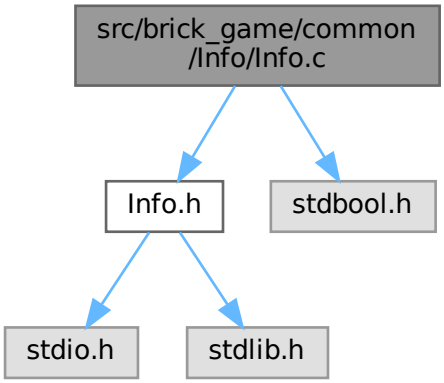
```
00001
00012 #pragma once
00013
00017 typedef enum {
00018   Start,
00019   Pause,
00020   Terminate,
00021   Left,
00022   Right,
00023   Up,
00024   Down,
00025   Action
00026 } UserAction_t;
```

8.3 src/brick_game/common/Info/Info.c File Reference

Source file with the main structure of the game.

```
#include "Info.h"
#include <stdbool.h>
```

Include dependency graph for Info.c:



Functions

- int ** allocateInt (const size_t rows, const size_t cols, int *code)

  Function to allocate memory for a two-dimensional array of type int.
- void freeIntDoubleArray (int ***data, const size_t rows)

  Function to free memory for a two-dimensional array of type int.
- int allocateField (GameInfo_t *engine)

  Allocates memory for the game field.
- int allocateNext (GameInfo_t *engine)

  Allocates memory for the next piece preview area.
- void freeNext (GameInfo_t *engine)

  Frees the memory allocated for the next piece preview area.
- void freeField (GameInfo_t *engine)

  Frees the memory allocated for the game field.
- void setScore (GameInfo_t *engine, const int score)

  Set the Score object.
- void setHigeScore (GameInfo_t *engine, const int hscore)

  Set the Hige Score object.
- void setSpeed (GameInfo_t *engine, const int speed)

  Set the Speed object.
- void setLevel (GameInfo_t *engine, const int level)

  Set the Level object.
- void setPause (GameInfo_t *engine, const int pause)

  Set the Pause object.
- int getScore (const GameInfo_t *engine)

  Gets the score from the GameInfo_t structure.
- int getHigeScore (const GameInfo_t *engine)

  Gets the high score from the GameInfo_t structure.
- int getSpeed (const GameInfo_t *engine)

  Gets the speed from the GameInfo_t structure.

- int getLevel (const GameInfo_t *engine)

  Gets the level from the GameInfo_t structure.
- int getPause (const GameInfo_t *engine)

  Gets the pause state from the GameInfo_t structure.

8.3.1 Detailed Description

Source file with the main structure of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

Copyright (c) 2024

8.3.2 Function Documentation

8.3.2.1 allocateField()

```
int allocateField (
            GameInfo_t * engine )
```

Allocates memory for the game field.

This function allocates memory for the game field array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

int Returns GOOD_ALLOCATE if memory allocation is successful, else returns BAD_ALLOCATE.

8.3.2.2 allocateInt()

```
int ** allocateInt (
            const size_t rows,
```

```
            const size_t cols,
            int * code )
```

Function to allocate memory for a two-dimensional array of type int.

Parameters

rows	number of lines.
cols	number of columns.
code	execution code.

Returns

two-dimensional array of type int.

8.3.2.3 allocateNext()

```
int allocateNext (
            GameInfo_t * engine )
```

Allocates memory for the next piece preview area.

This function allocates memory for the next piece preview area array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

int Returns GOOD_ALLOCATE if memory allocation is successful, else returns BAD_ALLOCATE.

8.3.2.4 freeField()

```
void freeField (
            GameInfo_t * engine )
```

Frees the memory allocated for the game field.

This function frees the memory allocated for the game field array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

8.3.2.5 freeIntDoubleArray()

```
void freeIntDoubleArray (
```

```
            int *** data,
            const size_t rows )
```

Function to free memory for a two-dimensional array of type int.

Parameters

data	Array pointer.
rows	Number of rows in the array.

8.3.2.6 freeNext()

```
void freeNext (
            GameInfo_t * engine )
```

Frees the memory allocated for the next piece preview area.

This function frees the memory allocated for the next piece preview area array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

8.3.2.7 getHigeScore()

```
int getHigeScore (
            const GameInfo_t * engine )
```

Gets the high score from the GameInfo_t structure.

This function retrieves the high score from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The high score value.

8.3.2.8 getLevel()

```
int getLevel (
            const GameInfo_t * engine )
```

Gets the level from the GameInfo_t structure.

This function retrieves the level from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The level value.

8.3.2.9 getPause()

```
int getPause (
            const GameInfo_t * engine )
```

Gets the pause state from the GameInfo_t structure.

This function retrieves the pause state from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The pause state value.

8.3.2.10 getScore()

```
int getScore (
            const GameInfo_t * engine )
```

Gets the score from the GameInfo_t structure.

This function retrieves the score from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The score value.

8.3.2.11 getSpeed()

```
int getSpeed (
            const GameInfo_t * engine )
```

Gets the speed from the GameInfo_t structure.

This function retrieves the speed from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The speed value.

8.3.2.12 setHigeScore()

```
void setHigeScore (
            GameInfo_t * engine,
            const int hscore )
```

Set the Hige Score object.

Parameters

engine	Pointer to the GameInfo_t structure.
hscore	The score value to set.

8.3.2.13 setLevel()

```
void setLevel (
            GameInfo_t * engine,
            const int level )
```

Set the Level object.

Parameters

engine	Pointer to the GameInfo_t structure.
level	The level value to set.

8.3.2.14 setPause()

```
void setPause (
            GameInfo_t * engine,
            const int pause )
```

Set the Pause object.

Parameters

engine	Pointer to the GameInfo_t structure.
pause	The pause value to set.

8.3.2.15 setScore()

```
void setScore (
            GameInfo_t * engine,
            const int score )
```

Set the Score object.

Parameters

engine	Pointer to the GameInfo_t structure.
score	The score value to set.

8.3.2.16 setSpeed()

```
void setSpeed (
            GameInfo_t * engine,
            const int speed )
```

Set the Speed object.

Parameters

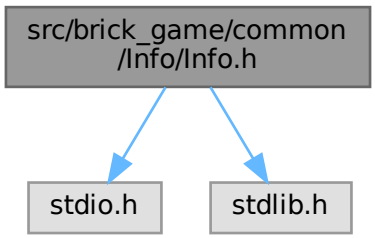
engine	Pointer to the GameInfo_t structure.
speed	The speed value to set.

8.4 src/brick_game/common/Info/Info.h File Reference

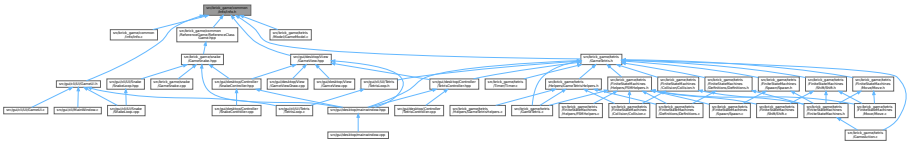
Header file with the main structure of the game.

```
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for Info.h:



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



Classes

- struct GameInfo_t

  A structure holding information about a game.

Macros

- #define WFIELD 10
- #define HFIELD 20
- #define WNEXT 4
- #define HNEXT 2
- #define ST_CODE_SNAKE 100
- #define ST_CODE_FRUIT 200
- #define BAD_SIZE 2
- #define GOOD_ALLOCATE 1
- #define BAD_ALLOCATE 0

Functions

- int ** allocateInt (const size_t rows, const size_t cols, int *code)

  Function to allocate memory for a two-dimensional array of type int.
- void freeIntDoubleArray (int ***data, const size_t rows)

  Function to free memory for a two-dimensional array of type int.
- int allocateField (GameInfo_t *engine)

  Allocates memory for the game field.
- int allocateNext (GameInfo_t *engine)

  Allocates memory for the next piece preview area.
- void freeField (GameInfo_t *engine)

  Frees the memory allocated for the game field.
- void freeNext (GameInfo_t *engine)

  Frees the memory allocated for the next piece preview area.
- void setScore (GameInfo_t *engine, const int score)

  Set the Score object.
- void setHigeScore (GameInfo_t *engine, const int hscore)

  Set the Hige Score object.
- void setSpeed (GameInfo_t *engine, const int speed)

  Set the Speed object.
- void setLevel (GameInfo_t *engine, const int level)

  Set the Level object.
- void setPause (GameInfo_t *engine, const int pause)

  Set the Pause object.

- int getScore (const GameInfo_t *engine)

  Gets the score from the GameInfo_t structure.
- int getHigeScore (const GameInfo_t *engine)

  Gets the high score from the GameInfo_t structure.
- int getSpeed (const GameInfo_t *engine)

  Gets the speed from the GameInfo_t structure.
- int getLevel (const GameInfo_t *engine)

  Gets the level from the GameInfo_t structure.
- int getPause (const GameInfo_t *engine)

  Gets the pause state from the GameInfo_t structure.

8.4.1 Detailed Description

Header file with the main structure of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.4.2 Macro Definition Documentation

8.4.2.1 BAD_ALLOCATE

```
#define BAD_ALLOCATE 0
```

Indicates a failed allocation.

8.4.2.2 BAD_SIZE

```
#define BAD_SIZE 2
```

Indicates a bad size value.

8.4.2.3 GOOD_ALLOCATE

```
#define GOOD_ALLOCATE 1
```

Indicates a successful allocation.

8.4.2.4 HFIELD

```
#define HFIELD 20
```

Height field

8.4.2.5 HNEXT

```
#define HNEXT 2
```

Height next

8.4.2.6 ST_CODE_FRUIT

```
#define ST_CODE_FRUIT 200
```

int code fruit

8.4.2.7 ST_CODE_SNAKE

```
#define ST_CODE_SNAKE 100
```

int code snake

8.4.2.8 WFIELD

```
#define WFIELD 10
```

Width field

8.4.2.9 WNEXT

```
#define WNEXT 4
```

Width next

8.4.3 Function Documentation

8.4.3.1 allocateField()

```
int allocateField (
            GameInfo_t * engine )
```

Allocates memory for the game field.

This function allocates memory for the game field array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

int Returns GOOD_ALLOCATE if memory allocation is successful, else returns BAD_ALLOCATE.

8.4.3.2 allocateInt()

```
int ** allocateInt (
            const size_t rows,
            const size_t cols,
            int * code )
```

Function to allocate memory for a two-dimensional array of type int.

Parameters

rows	number of lines.
cols	number of columns.
code	execution code.

Returns

two-dimensional array of type int.

8.4.3.3 allocateNext()

```
int allocateNext (
            GameInfo_t * engine )
```

Allocates memory for the next piece preview area.

This function allocates memory for the next piece preview area array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

int Returns GOOD_ALLOCATE if memory allocation is successful, else returns BAD_ALLOCATE.

8.4.3.4 freeField()

```
void freeField (
            GameInfo_t * engine )
```

Frees the memory allocated for the game field.

This function frees the memory allocated for the game field array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

8.4.3.5 freeIntDoubleArray()

```
void freeIntDoubleArray (
            int *** data,
            const size_t rows )
```

Function to free memory for a two-dimensional array of type int.

Parameters

data	Array pointer.
rows	Number of rows in the array.

8.4.3.6 freeNext()

```
void freeNext (
            GameInfo_t * engine )
```

Frees the memory allocated for the next piece preview area.

This function frees the memory allocated for the next piece preview area array in the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

8.4.3.7 getHigeScore()

```
int getHigeScore (
            const GameInfo_t * engine )
```

Gets the high score from the GameInfo_t structure.

This function retrieves the high score from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The high score value.

8.4.3.8 getLevel()

```
int getLevel (
            const GameInfo_t * engine )
```

Gets the level from the GameInfo_t structure.

This function retrieves the level from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The level value.

8.4.3.9 getPause()

```
int getPause (
            const GameInfo_t * engine )
```

Gets the pause state from the GameInfo_t structure.

This function retrieves the pause state from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The pause state value.

8.4.3.10 getScore()

```
int getScore (
            const GameInfo_t * engine )
```

Gets the score from the GameInfo_t structure.

This function retrieves the score from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The score value.

8.4.3.11 getSpeed()

```
int getSpeed (
            const GameInfo_t * engine )
```

Gets the speed from the GameInfo_t structure.

This function retrieves the speed from the GameInfo_t structure.

Parameters

engine	Pointer to the GameInfo_t structure.

Returns

The speed value.

8.4.3.12 setHigeScore()

```
void setHigeScore (
            GameInfo_t * engine,
            const int hscore )
```

Set the Hige Score object.

Parameters

engine	Pointer to the GameInfo_t structure.
hscore	The score value to set.

8.4.3.13 setLevel()

```
void setLevel (
            GameInfo_t * engine,
            const int level )
```

Set the Level object.

Parameters

engine	Pointer to the GameInfo_t structure.
level	The level value to set.

8.4.3.14 setPause()

```
void setPause (
            GameInfo_t * engine,
            const int pause )
```

Set the Pause object.

Parameters

engine	Pointer to the GameInfo_t structure.
pause	The pause value to set.

8.4.3.15 setScore()

```
void setScore (
            GameInfo_t * engine,
            const int score )
```

Set the Score object.

Parameters

engine	Pointer to the GameInfo_t structure.
score	The score value to set.

8.4.3.16 setSpeed()

```
void setSpeed (
            GameInfo_t * engine,
            const int speed )
```

Set the Speed object.

Parameters

engine	Pointer to the GameInfo_t structure.
speed	The speed value to set.

8.5 Info.h

Go to the documentation of this file.

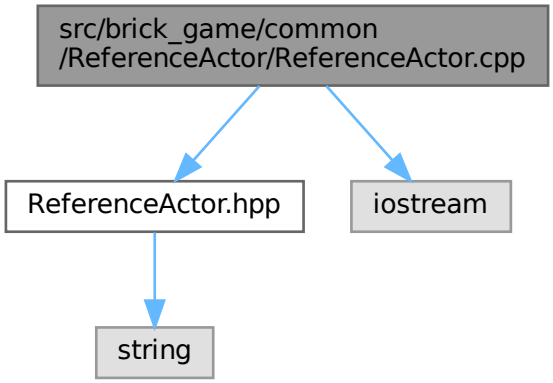
```
00001
00011 #pragma once
00012
00013 #ifdef __cplusplus
00014 extern "C" {
00015 #endif
00016
00017 #include <stdio.h>
00018 #include <stdlib.h>
00019
00020 #define WFIELD 10
00021 #define HFIELD 20
00023 #define WNEXT 4
00024 #define HNEXT 2
00026 #define ST_CODE_SNAKE 100
00027 #define ST_CODE_FRUIT 200
00029 #define BAD_SIZE 2
00030 #define GOOD_ALLOCATE 1
00031 #define BAD_ALLOCATE 0
00036 typedef struct {
00037   int **field;
00038   int **next;
00039   int score;
00040   int high_score;
00041   int level;
00042   int speed;
00043   int pause;
00044 } GameInfo_t;
00045
00046 int **allocateInt(const size_t rows, const size_t cols, int *code);
00047 void freeIntDoubleArray(int ***data, const size_t rows);
00048
00049 int allocateField(GameInfo_t *engine);
00050 int allocateNext(GameInfo_t *engine);
00051 void freeField(GameInfo_t *engine);
00052 void freeNext(GameInfo_t *engine);
00053
00054 void setScore(GameInfo_t *engine, const int score);
00055 void setHigeScore(GameInfo_t *engine, const int hscore);
00056 void setSpeed(GameInfo_t *engine, const int speed);
00057 void setLevel(GameInfo_t *engine, const int level);
00058 void setPause(GameInfo_t *engine, const int pause);
00059
00060 int getScore(const GameInfo_t *engine);
00061 int getHigeScore(const GameInfo_t *engine);
00062 int getSpeed(const GameInfo_t *engine);
00063 int getLevel(const GameInfo_t *engine);
00064 int getPause(const GameInfo_t *engine);
00065
00066 #ifdef __cplusplus
00067 }
00068 #endif
```

8.6 src/brick_game/common/ReferenceActor/ReferenceActor.cpp File Reference

Source file with the reference actor of the game.

```
#include "ReferenceActor.hpp"
#include <iostream>
```

Include dependency graph for ReferenceActor.cpp:



8.6.1 Detailed Description

Source file with the reference actor of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

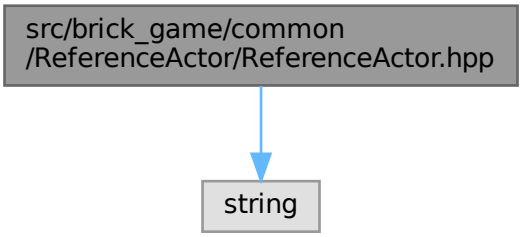
Copyright (c) 2024

8.7 src/brick_game/common/ReferenceActor/ReferenceActor.hpp File Reference

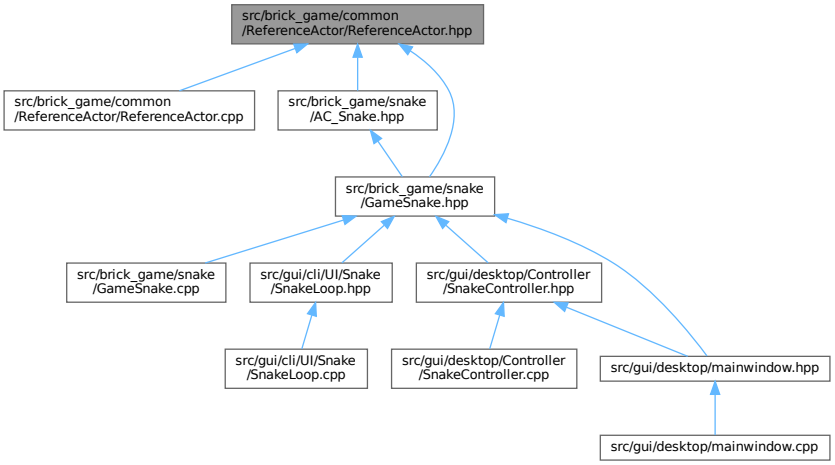
Header file with the reference actor of the game.

```
#include <string>
```

Include dependency graph for ReferenceActor.hpp:



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



Classes

- struct s21::Coordinate

  A structure representing a coordinate in a 2D space.
- class s21::ReferenceActor

  A base class for actors in a game.

Variables

- const std::string **s21::NOT_NAME** = "none"

  String: `none`

8.7.1 Detailed Description

Header file with the reference actor of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.8 ReferenceActor.hpp

Go to the documentation of this file.

```
00001
00011 #pragma once
00012 #include <string>
00013
00014 namespace s21 {
00015
00019 struct Coordinate {
00020   int x;
00021   int y;
00031   bool operator==(const Coordinate& other) const {
00032     return (this->x == other.x && this->y == other.y);
00033   }
00034
00043   bool operator!=(const Coordinate& other) const {
00044     return (this->x != other.x || this->y != other.y);
00045   }
00046
00055   Coordinate& operator+=(const Coordinate& other) {
00056     this->x += other.x;
00057     this->y += other.y;
00058     return *this;
00059   }
00060 };
00061
00065 const std::string NOT_NAME = "none";
00066
00070 class ReferenceActor {
00071  protected:
00072   std::string name;
00073   bool isAlive;
00074   bool movementBlocked;
00075   Coordinate location;
00076   Coordinate rotation;
00078  public:
00079   ReferenceActor(void);
00080   ReferenceActor(const std::string& name_, const bool alive, const bool block,
00081                  const Coordinate loc, const Coordinate rot);
00082   virtual ~ReferenceActor() = default;
00083
00084   virtual void move(void);
00085
00086   virtual void setRotation(const int x, const int y);
00087   virtual void setRotation(const Coordinate coord);
00088   virtual Coordinate getRotation(void);
00089
```

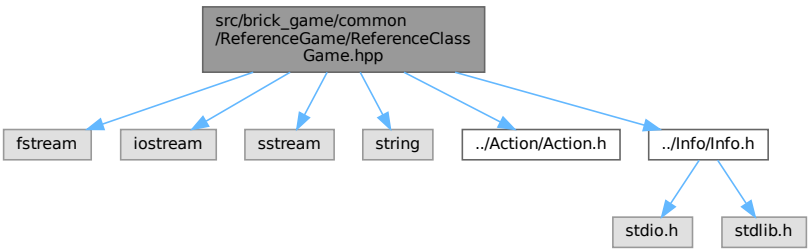
```
00090   virtual void setLocation(const int x, const int y);
00091   virtual void setLocation(const Coordinate coord);
00092   virtual Coordinate getLocation(void);
00093
00094   virtual void setIsAlive(bool alive);
00095   virtual bool getIsAlive(void);
00096
00097   virtual void setMovementBlocked(bool block);
00098   virtual bool getMovementBlocked(void);
00099 };
00100 }  // namespace s21
```

8.9 src/brick_game/common/ReferenceGame/ReferenceClassGame.hpp File Reference

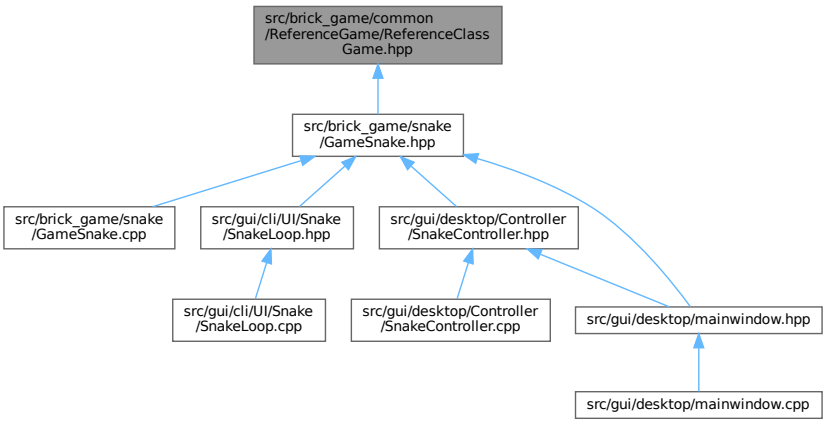
File in which the virtual class of the game is implemented.

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include "../Action/Action.h"
#include "../Info/Info.h"
```

Include dependency graph for ReferenceClassGame.hpp:



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



Classes

- class s21::Game

  The base class for the game.

Variables

- constexpr int s21::ST_SIZE_WIDTH = WFIELD

  Width of the game state.
- constexpr int s21::ST_SIZE_HEIGHT = HFIELD

  Height of the game state.
- constexpr int s21::ST_DIRACTION_LEFT = -1

  Action for moving left.
- constexpr int s21::ST_DIRACTION_RIGHT = 1

  Action for moving right.
- constexpr int s21::ST_DIRACTION_UP = -1

  Action for moving up.
- constexpr int s21::ST_DIRACTION_DOWN = 1

  Action for moving down.
- const std::string s21::FILE_SAVE = "BD_Snake.txt"

  File path for saving the game.
- const std::string s21::SAVING_INFO = "High Score: "

  Information string for saving high score.

8.9.1 Detailed Description

File in which the virtual class of the game is implemented.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.9.2 Variable Documentation

8.9.2.1 FILE_SAVE

```
const std::string s21::FILE_SAVE = "BD_Snake.txt"
```

File path for saving the game.

This constant represents the file path for saving the game state.

8.9.2.2 SAVING_INFO

```
const std::string s21::SAVING_INFO = "High Score:  "
```

Information string for saving high score.

This constant represents the string used to save high score information.

8.9.2.3 ST_DIRACTION_DOWN

```
constexpr int s21::ST_DIRACTION_DOWN = 1  [constexpr]
```

Action for moving down.

This constant represents the action for moving the game state down.

8.9.2.4 ST_DIRACTION_LEFT

```
constexpr int s21::ST_DIRACTION_LEFT = -1  [constexpr]
```

Action for moving left.

This constant represents the action for moving the game state to the left.

8.9.2.5 ST_DIRACTION_RIGHT

```
constexpr int s21::ST_DIRACTION_RIGHT = 1  [constexpr]
```

Action for moving right.

This constant represents the action for moving the game state to the right.

8.9.2.6 ST_DIRACTION_UP

```
constexpr int s21::ST_DIRACTION_UP = -1  [constexpr]
```

Action for moving up.

This constant represents the action for moving the game state up.

8.9.2.7 ST_SIZE_HEIGHT

```
constexpr int s21::ST_SIZE_HEIGHT = HFIELD  [constexpr]
```

Height of the game state.

This constant represents the height of the game state.

8.9.2.8 ST_SIZE_WIDTH

```
constexpr int s21::ST_SIZE_WIDTH = WFIELD  [constexpr]
```

Width of the game state.

This constant represents the width of the game state.

8.10 ReferenceClassGame.hpp

Go to the documentation of this file.

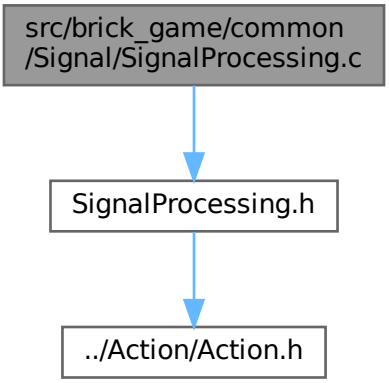
```
00001
00012 #pragma once
00013 #include <fstream>
00014 #include <iostream>
00015 #include <sstream>
00016 #include <string>
00017
00018 #include "../Action/Action.h"
00019 #include "../Info/Info.h"
00020
00021 namespace s21 {
00022
00028 constexpr int ST_SIZE_WIDTH = WFIELD;
00029
00035 constexpr int ST_SIZE_HEIGHT = HFIELD;
00036
00042 constexpr int ST_DIRACTION_LEFT = -1;
00043
00049 constexpr int ST_DIRACTION_RIGHT = 1;
00050
00056 constexpr int ST_DIRACTION_UP = -1;
00057
00063 constexpr int ST_DIRACTION_DOWN = 1;
00064
00070 const std::string FILE_SAVE = "BD_Snake.txt";
00071
00077 const std::string SAVING_INFO = "High Score: ";
00078
00082 class Game {
00083 #ifdef TESTING
00084  public:
00085 #else
00086  protected:
00087 #endif
00088   GameInfo_t engine;
00089   std::string filenameSaving;
00091  public:
00095   Game(void) : engine({}), filenameSaving(FILE_SAVE){};
00096
00102   explicit Game(const std::string &filename)
00103       : engine({}), filenameSaving(filename){};
00104   virtual ~Game() = default;
00105
00106 #ifdef TESTING
00107   virtual void userInput(UserAction_t action, bool hold) {
00108     (void)action;
00109     (void)hold;
00110   };
00111 #else
00118   virtual void userInput(UserAction_t action, bool hold) = 0;
00119 #endif
00128   virtual GameInfo_t updateCurrentState() { return this->engine; };
00129
00135   virtual void cleanField() {
00136     if (!engine.field) return;
00137     for (int i = 0; i < ST_SIZE_HEIGHT; i++)
00138       for (int j = 0; j < ST_SIZE_WIDTH; j++) engine.field[i][j] = 0;
00139   }
00140
00146   virtual void save() {
00147     std::ofstream file(filenameSaving, std::ios::out | std::ios::trunc);
00148     if (file.is_open()) {
00149       file « SAVING_INFO « engine.high_score « std::endl;
00150       file.close();
00151     }
00152   }
```

```
00153
00160   virtual void readSave() {
00161     std::ifstream file(filenameSaving);
00162     if (file.is_open()) {
00163       std::string line;
00164       std::getline(file, line);
00165       file.close();
00166
00167       std::string::size_type pos = line.find(SAVING_INFO);
00168       if (pos != std::string::npos) {
00169         std::string scoreStr = line.substr(pos + SAVING_INFO.size());
00170         std::istringstream scoreStream(scoreStr);
00171         scoreStream » engine.high_score;
00172
00173       } else
00174         engine.high_score = 0;
00175     } else
00176       engine.high_score = 0;
00177   }
00178 };
00179
00180 }  // namespace s21
```

8.11 src/brick_game/common/Signal/SignalProcessing.c File Reference

Source file with the signals of the game.

`#include "SignalProcessing.h"`
Include dependency graph for SignalProcessing.c:



Functions

- UserAction_t get_signal (int signal)

  Get the signal object.

8.11.1 Detailed Description

Source file with the signals of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.11.2 Function Documentation

8.11.2.1 get_signal()

```
UserAction_t get_signal (
             int signal )
```

Get the signal object.

Parameters

signal	User input signal

Returns

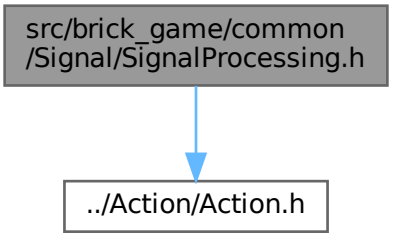
UserAction_t

8.12 src/brick_game/common/Signal/SignalProcessing.h File Reference

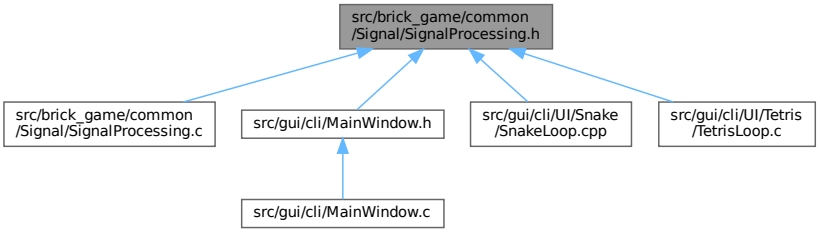
Header file with the signals of the game.

```
#include "../Action/Action.h"
```

Include dependency graph for SignalProcessing.h:



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



Macros

- #define KEY_DOWN_B 0402
- #define KEY_UP_B 0403
- #define KEY_LEFT_B 0404
- #define KEY_RIGHT_B 0405
- #define KEY_PAUSE_UPPER 'P'
- #define KEY_PAUSE_LOWER 'p'
- #define KEY_SPACE 32
- #define KEY_ENTER1 10
- #define KEY_ENTER2 13
- #define KEY_EXIT_BT 'q'

Functions

- UserAction_t get_signal (int signal)

  Get the signal object.

8.12.1 Detailed Description

Header file with the signals of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.12.2 Macro Definition Documentation

8.12.2.1 KEY_DOWN_B

```
#define KEY_DOWN_B 0402
```

Key code for the down arrow button.

8.12.2.2 KEY_ENTER1

```
#define KEY_ENTER1 10
```

Key code for the Enter button (variant 1).

8.12.2.3 KEY_ENTER2

```
#define KEY_ENTER2 13
```

Key code for the Enter button (variant 2).

8.12.2.4 KEY_EXIT_BT

```
#define KEY_EXIT_BT 'q'
```

Key code for the exit button.

8.12.2.5 KEY_LEFT_B

```
#define KEY_LEFT_B 0404
```

Key code for the left arrow button.

8.12.2.6 KEY_PAUSE_LOWER

```
#define KEY_PAUSE_LOWER 'p'
```

Lower case key code for the pause button.

8.12.2.7 KEY_PAUSE_UPPER

```
#define KEY_PAUSE_UPPER 'P'
```

Upper case key code for the pause button.

8.12.2.8 KEY_RIGHT_B

```
#define KEY_RIGHT_B 0405
```

Key code for the right arrow button.

8.12.2.9 KEY_SPACE

```
#define KEY_SPACE 32
```

Key code for the space button.

8.12.2.10 KEY_UP_B

```
#define KEY_UP_B 0403
```

Key code for the up arrow button.

8.12.3 Function Documentation

8.12.3.1 get_signal()

```
UserAction_t get_signal (
             int signal )
```

Get the signal object.

Parameters

signal	User input signal

Returns

UserAction_t

8.13 SignalProcessing.h

Go to the documentation of this file.

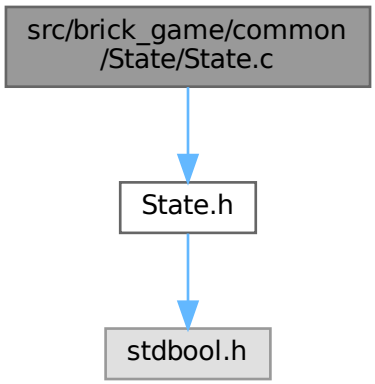
```
00001
00012 #pragma once
00013 #include "../Action/Action.h"
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 #define KEY_DOWN_B 0402
00020 #define KEY_UP_B 0403
00021 #define KEY_LEFT_B 0404
00022 #define KEY_RIGHT_B 0405
00023 #define KEY_PAUSE_UPPER 'P'
00024 #define KEY_PAUSE_LOWER 'p'
00026 #define KEY_SPACE 32
00028 #define KEY_ENTER1 10
00029 #define KEY_ENTER2 13
00031 #define KEY_EXIT_BT 'q'
00033 UserAction_t get_signal(int signal);
00034
00035 #ifdef __cplusplus
00036 }
00037 #endif
```

8.14 src/brick_game/common/State/State.c File Reference

Source File with game state.

`#include "State.h"`

Include dependency graph for State.c:



Functions

- bool isInfoState (GameState_t state)

  Checks if the game state represents an informational state.
- bool isGamingState (GameState_t state)

  Checks if the game state represents a gaming state.
- bool isGamingStateWithoutKey (GameState_t state)

  Checks if the game state represents a gaming state without user input.
- char * convertStateToStrInf (GameState_t state)

  Converts a game state to a string representation.

8.14.1 Detailed Description

Source File with game state.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.14.2 Function Documentation

8.14.2.1 convertStateToStrInf()

```
char * convertStateToStrInf (
            GameState_t state )
```

Converts a game state to a string representation.

This function takes a `GameState_t` enum value as input and returns a string representation of the state. The function uses a switch statement to determine which string to return based on the input state.

Parameters

state	The game state to convert to a string representation.

Returns

char* A pointer to a string representing the game state.

8.14.2.2 isGamingState()

```
bool isGamingState (
            GameState_t state )
```

Checks if the game state represents a gaming state.

Parameters

state	The game state to check.

Returns

true if the state is between SPAWN and SHIFTING (inclusive), false otherwise.

8.14.2.3 isGamingStateWithoutKey()

```
bool isGamingStateWithoutKey (
            GameState_t state )
```

Checks if the game state represents a gaming state without user input.

Parameters

state	The game state to check.

Returns

true if the state is either SPAWN, SHIFT, or COLLIDE, false otherwise.

8.14.2.4 isInfoState()

```
bool isInfoState (
            GameState_t state )
```

Checks if the game state represents an informational state.

Parameters

state	The game state to check.

Returns

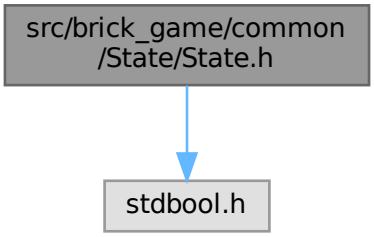
true if the state is either START or GAME_OVER, false otherwise.

8.15 src/brick_game/common/State/State.h File Reference

Header File with game state.

```
#include <stdbool.h>
```

Include dependency graph for State.h:



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

Macros

- #define DS_Start "start"
- #define DS_End "end"
- #define DS_Not "not"

Enumerations

- enum GameState_t {
  START , SPAWN , MOVE , SHIFT ,
  COLLISION , GAMEOVER , PAUSE , EXIT }

  Defines an enumeration of game states. This enumeration represents the different states that a game can be in.

Functions

- bool isInfoState (GameState_t state)
  Checks if the game state represents an informational state.
- bool isGamingState (GameState_t state)
  Checks if the game state represents a gaming state.
- bool isGamingStateWithoutKey (GameState_t state)
  Checks if the game state represents a gaming state without user input.
- char * convertStateToStrInf (GameState_t state)
  Converts a game state to a string representation.

8.15.1 Detailed Description

Header File with game state.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

Copyright (c) 2024

8.15.2 Macro Definition Documentation

8.15.2.1 DS_End

```
#define DS_End "end"
```

String: `end`

8.15.2.2 DS_Not

```
#define DS_Not "not"
```

String: `not`

8.15.2.3 DS_Start

```
#define DS_Start "start"
```

String: `start`

8.15.3 Enumeration Type Documentation

8.15.3.1 GameState_t

```
enum GameState_t
```

Defines an enumeration of game states. This enumeration represents the different states that a game can be in.

Enumerator

START	State START
SPAWN	State SPAWN
MOVE	State MOVE
SHIFT	State SHIFT
COLLISION	State COLLISION
GAMEOVER	State GAMEOVER
PAUSE	State PAUSE
EXIT	State EXIT

8.15.4 Function Documentation

8.15.4.1 convertStateToStrInf()

```
char * convertStateToStrInf (
            GameState_t state )
```

Converts a game state to a string representation.

This function takes a `GameState_t` enum value as input and returns a string representation of the state. The function uses a switch statement to determine which string to return based on the input state.

Parameters

state	The game state to convert to a string representation.

Returns

char* A pointer to a string representing the game state.

8.15.4.2 isGamingState()

```
bool isGamingState (
            GameState_t state )
```

Checks if the game state represents a gaming state.

Parameters

state	The game state to check.

Returns

true if the state is between SPAWN and SHIFTING (inclusive), false otherwise.

8.15.4.3 isGamingStateWithoutKey()

```
bool isGamingStateWithoutKey (
            GameState_t state )
```

Checks if the game state represents a gaming state without user input.

Parameters

state	The game state to check.

Returns

true if the state is either SPAWN, SHIFT, or COLLIDE, false otherwise.

8.15.4.4 isInfoState()

```
bool isInfoState (
            GameState_t state )
```

Checks if the game state represents an informational state.

Parameters

state	The game state to check.

Returns

true if the state is either START or GAME_OVER, false otherwise.

8.16 State.h

Go to the documentation of this file.

```
00001
00011 #pragma once
00012
00013 #ifdef __cplusplus
00014 extern "C" {
00015 #endif
00016
00017 #include <stdbool.h>
00018
00019 #define DS_Start "start"
00020 #define DS_End "end"
00021 #define DS_Not "not"
00027 typedef enum {
00028   START,
00029   SPAWN,
00030   MOVE,
00031   SHIFT,
00032   COLLISION,
00033   GAMEOVER,
00034   PAUSE,
00035   EXIT
00036 } GameState_t;
00037
00038 bool isInfoState(GameState_t state);
00039 bool isGamingState(GameState_t state);
```

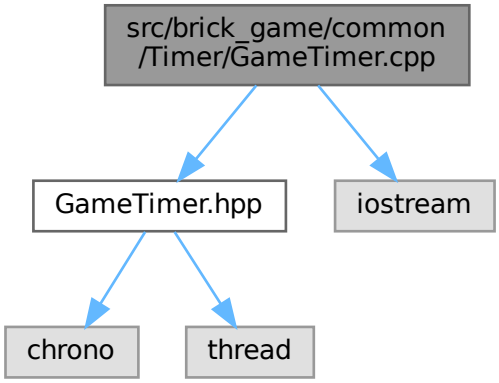
```
00040 bool isGamingStateWithoutKey(GameState_t state);
00041 char *convertStateToStrInf(GameState_t state);
00042
00043 #ifdef __cplusplus
00044 }
00045 #endif
```

8.17 src/brick_game/common/Timer/GameTimer.cpp File Reference

Source file with the timer of the game.

```
#include "GameTimer.hpp"
#include <iostream>
```

Include dependency graph for GameTimer.cpp:



8.17.1 Detailed Description

Source file with the timer of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

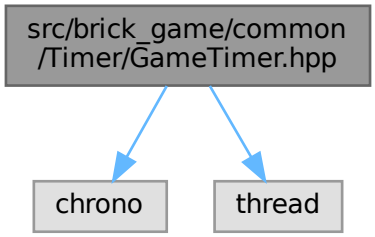
Copyright (c) 2024

8.18 src/brick_game/common/Timer/GameTimer.hpp File Reference

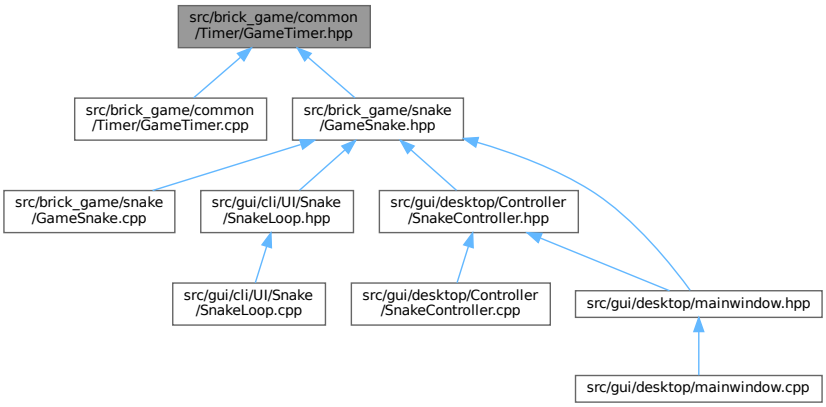
Header file with the timer of the game.

```
#include <chrono>
#include <thread>
```

Include dependency graph for GameTimer.hpp:



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



Classes

- class s21::GameTimer

  Game timer class.

8.18.1 Detailed Description

Header file with the timer of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.19 GameTimer.hpp

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #include <chrono>
00015 #include <thread>
00016
00017 namespace s21 {
00018
00022 class GameTimer {
00023  public:
00027   using type_time = std::chrono::time_point<std::chrono::steady_clock>;
00028   GameTimer(void);
00029   ~GameTimer() = default;
00030
00031   void updateStartTime(void);
00032   void updateEndTime(void);
00033
00034   void setDuration(const std::chrono::milliseconds time);
00035   const std::chrono::milliseconds getDuration(void) const;
00036
00037   void setActive(bool active_);
00038   bool getActive(void) const;
00039
00040  private:
00041   type_time start_time;
00042   type_time end_time;
00043   std::chrono::milliseconds duration;
00044   bool active;
00045 };
00046
00047 }  // namespace s21
```

8.20 AC_Snake.hpp

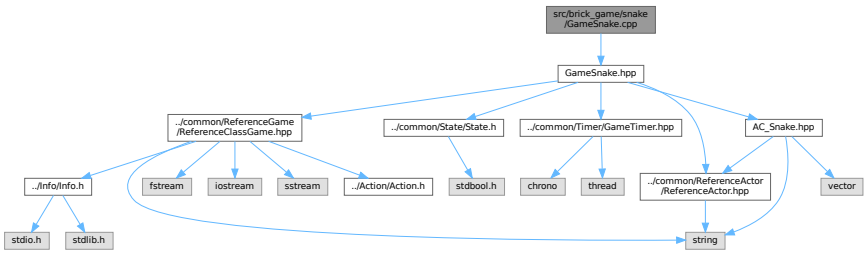
```
00001 #pragma once
00002 #include <string>
00003 #include <vector>
00004
00005 #include "../common/ReferenceActor/ReferenceActor.hpp"
00006
00007 namespace s21 {
00008
00012 constexpr int16_t ST_MAX_LENGTH_SNAKE = 200;
00013
00017 constexpr int16_t ST_INIT_LENGTH_SNAKE = 4;
00018
00022 constexpr int16_t ST_MAX_INIT_LENGTH_SNAKE = 5;
00023
00027 constexpr Coordinate ST_LOCATION_SNAKE = {5, 10};
```

```
00028
00032 constexpr Coordinate ST_ROTATION_SNAKE = {1, 0};
00033
00037 constexpr Coordinate ST_BAD_POSITION = {-100, -100};
00038
00042 const std::string CS_Wall = "Wall";
00043
00047 const std::string CS_Fruit = "Fruit";
00048
00052 const std::string CS_Snake = "Snake";
00053
00057 const std::string CS_NoCollide = "No Collide";
00058
00064 class AC_Snake : public ReferenceActor {
00065 #ifdef TESTING
00066  public:
00067 #else
00068  private:
00069 #endif
00070
00071   Coordinate blockAsix;
00072   Coordinate track;
00074   std::vector<Coordinate>
00075       body;
00078  public:
00079   AC_Snake(void);
00080   AC_Snake(int length, Coordinate location);
00081
00082   ~AC_Snake();
00083
00084   void move() override;
00085   void setRotation(const int x, const int y) override;
00086   void setRotation(const Coordinate coord) override;
00087   void IncreaseLength(void);
00088   const std::vector<Coordinate>& getSnake(void) const;
00089   int getLength(void) const;
00090 };
00091
00092 }  // namespace s21
```

8.21 src/brick_game/snake/GameSnake.cpp File Reference

Snake game source file.

`#include "GameSnake.hpp"`
Include dependency graph for GameSnake.cpp:



8.21.1 Detailed Description

Snake game source file.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

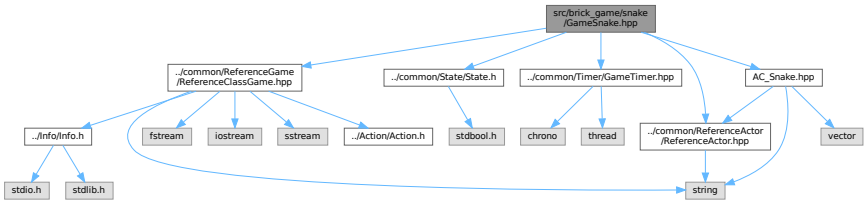
Copyright

8.22 src/brick_game/snake/GameSnake.hpp File Reference

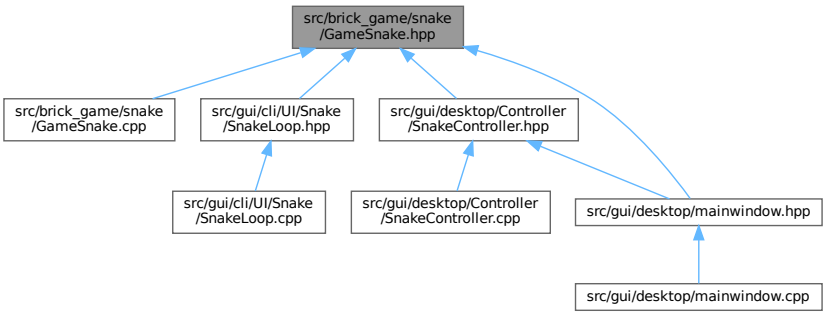
Snake game header file.

```
#include "../common/ReferenceActor/ReferenceActor.hpp"
#include "../common/ReferenceGame/ReferenceClassGame.hpp"
#include "../common/State/State.h"
#include "../common/Timer/GameTimer.hpp"
#include "AC_Snake.hpp"
```

Include dependency graph for GameSnake.hpp:



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



Classes

- class s21::GameSnake

  The GameSnake class represents the snake game.

Variables

- constexpr int16_t **s21::ST_MAX_LEVEL** = 10

  Maximum level for the game.
- constexpr int16_t **s21::ST_POINT_FOR_LEVEL** = 5

  Points required to reach a new level.
- constexpr int16_t **s21::ST_INIT_SPEED** = 200

  Initial speed for the game entities.
- constexpr int16_t **s21::ST_COEFICIENT_SPEED** = 10

  Coefficient for speed adjustment.
- const std::string **s21::NGAME_SNAKE** = "SNAKE"

  Name of the snake game.

8.22.1 Detailed Description

Snake game header file.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

8.23 GameSnake.hpp

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #include "../common/ReferenceActor/ReferenceActor.hpp"
00015 #include "../common/ReferenceGame/ReferenceClassGame.hpp"
00016 #include "../common/State/State.h"
00017 #include "../common/Timer/GameTimer.hpp"
00018 #include "AC_Snake.hpp"
00019
00020 namespace s21 {
00021
00025 constexpr int16_t ST_MAX_LEVEL = 10;
00026
00030 constexpr int16_t ST_POINT_FOR_LEVEL = 5;
00031
00035 constexpr int16_t ST_INIT_SPEED = 200;
00036
00040 constexpr int16_t ST_COEFICIENT_SPEED = 10;
00041
00045 const std::string NGAME_SNAKE = "SNAKE";
00046
```

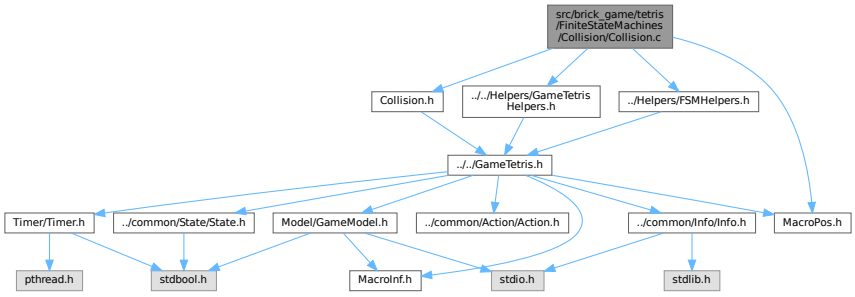
```
00053 class GameSnake : public Game {
00054 #ifdef TESTING
00055  public:
00056 #else
00057  private:
00058 #endif
00059   AC_Snake snake;
00060   ReferenceActor fruit;
00061   GameTimer timer;
00062   GameState_t state;
00064  public:
00065   GameSnake(void);
00066   ~GameSnake();
00067
00068   GameState_t getState() const;
00069   void setState(const GameState_t sstate);
00070   void reset(const GameState_t reset_state);
00071
00072   void userInput(UserAction_t action, bool hold) override;
00073   GameInfo_t updateCurrentState() override;
00074
00075 #ifdef TESTING
00076  public:
00077 #else
00078  private:
00079 #endif
00080   void addSnakeToField();
00081   void addFruitToField();
00082   void updateInfo();
00083
00084   void generateFruit(const int x, const int y);
00085   void generateFruitInRandomPosition();
00086   bool checkCollideSnake();
00087   bool checkCollide(std::string &object);
00088
00089   void helperFsmSpawn();
00090   void setDiraction(UserAction_t action);
00091   void caseMove(UserAction_t action);
00092   void helperFsmShift();
00093   void helperFsmCollision();
00094 };
00095
00096 }  // namespace s21
```

8.24 src/brick_game/tetris/FiniteStateMachines/Collision/Collision.c File Reference

COLLISION finite automaton.

```
#include "Collision.h"
#include "../../Helpers/GameTetrisHelpers.h"
#include "../../MacroPos.h"
#include "../Helpers/FSMHelpers.h"
```

Include dependency graph for Collision.c:



Functions

- unsigned int getTime (const GameInfo_t *engine)

  Calculates the time interval based on the game speed.
- void FSM_Collision (GameTetris *game)

  Handles collision events in the game state machine.

8.24.1 Detailed Description

COLLISION finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.24.2 Function Documentation

8.24.2.1 FSM_Collision()

```
void FSM_Collision (
            GameTetris * game )
```

Handles collision events in the game state machine.

Parameters

game	The game parameters.

8.24.2.2 getTime()

```
unsigned int getTime (
            const GameInfo_t * engine )
```

Calculates the time interval based on the game speed.

This function calculates the time interval (in milliseconds) based on the game speed specified in the GameTetris structure. The time interval decreases as the game speed increases.

Parameters

engine	Pointer to the GameInfo_t structure containing the game speed.

Returns

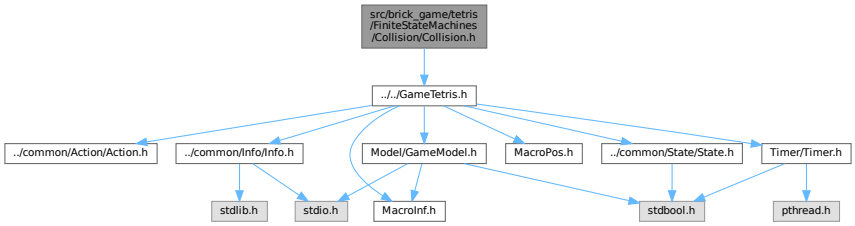
unsigned int The time interval in milliseconds.

8.25 src/brick_game/tetris/FiniteStateMachines/Collision/Collision.h File Reference

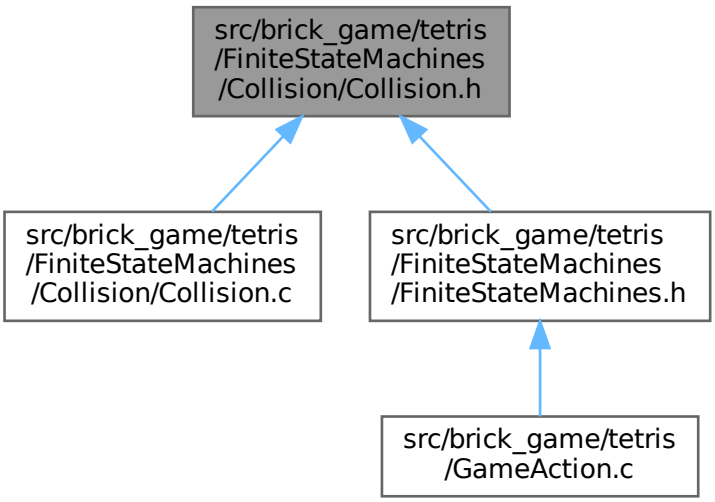
COLLISION finite automaton.

```
#include "../../GameTetris.h"
```

Include dependency graph for Collision.h:



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



Macros

- #define MAX_INDEX_Y HFIELD - 1
- #define POINT_ONE_LINE 100
- #define POINT_TWO_LINE 300
- #define POINT_THREE_LINE 700
- #define POINT_FOUR_LINE 1500
- #define PROCENT_MAX 100
- #define DIVISORTOGETANUMBER 10
- #define MAX_SCORE 999999
- #define MIN_SCORE 0
- #define LESSOREQUUAL(first, second) (first <= second)

  Macro to check if a number is less than or equal to another number.
- #define MORE(first, second) (first > second)

  Macro to check if the first number is greater than the second number.
- #define CHECKED_LEVEL(lfirst, lsecond, score) (LESSOREQUUAL(lfirst, score) && MORE(lsecond, score))

  Macro to check if a score is within a certain range of levels.

Enumerations

- enum LevelScore {
  FIRST = 600 , SECOND = 1200 , THIRD = 1800 , FOURTH = 2400 ,
  FIFTH = 3000 , SIXTH = 3600 , SEVENTH = 4200 , EIGHTH = 4800 ,
  NINETH = 5400 , TENTH = 6000 }

  Represents the score levels in the game.
- enum LEVEL {
  LFIRST = 1 , LSECOND = 2 , LTHIRD = 3 , LFOURTH = 4 ,
  LFIFTH = 5 , LSIXTH = 6 , LSEVENTH = 7 , LEIGHTH = 8 ,
  LNINETH = 9 , LTENTH = 10 }

  Represents the levels in the game.

Functions

- void FSM_Collision (GameTetris *game)

  Handles collision events in the game state machine.

8.25.1 Detailed Description

COLLISION finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

8.25.2 Macro Definition Documentation

8.25.2.1 CHECKED_LEVEL

```
#define CHECKED_LEVEL(
              lfirst,
              lsecond,
              score )  (LESSOREQUUAL(lfirst, score) && MORE(lsecond, score))
```

Macro to check if a score is within a certain range of levels.

Parameters

lfirst	The lower level bound.
lsecond	The upper level bound.
score	The score to check.

Returns

True if the score falls within the specified range, false otherwise.

8.25.2.2 DIVISORTOGETANUMBER

```
#define DIVISORTOGETANUMBER 10
```

The divisor to get a number.

8.25.2.3 LESSOREQUUAL

```
#define LESSOREQUUAL(
              first,
              second ) (first <= second)
```

Macro to check if a number is less than or equal to another number.

Parameters

first	The first number.
second	The second number.

Returns

True if the first number is less than or equal to the second number, false otherwise.

8.25.2.4 MAX_INDEX_Y

```
#define MAX_INDEX_Y HFIELD - 1
```

Maximum index value by y

8.25.2.5 MAX_SCORE

```
#define MAX_SCORE 999999
```

The maximum score.

8.25.2.6 MIN_SCORE

```
#define MIN_SCORE 0
```

The minimum score.

8.25.2.7 MORE

```
#define MORE(
             first,
             second ) (first > second)
```

Macro to check if the first number is greater than the second number.

Parameters

first	The first number.
second	The second number.

Returns

True if the first number is greater than the second number, false otherwise.

8.25.2.8 POINT_FOUR_LINE

```
#define POINT_FOUR_LINE 1500
```

Points for four collected levels

8.25.2.9 POINT_ONE_LINE

```
#define POINT_ONE_LINE 100
```

Points per level collected

8.25.2.10 POINT_THREE_LINE

```
#define POINT_THREE_LINE 700
```

Points for three levels collected

8.25.2.11 POINT_TWO_LINE

```
#define POINT_TWO_LINE 300
```

Points for two levels collected

8.25.2.12 PROCENT_MAX

```
#define PROCENT_MAX 100
```

The maximum value for percentage.

8.25.3 Enumeration Type Documentation

8.25.3.1 LEVEL

```
enum LEVEL
```

Represents the levels in the game.

Enumerator

LFIRST	First level.
LSECOND	Second level.
LTHIRD	Third level.
LFOURTH	Fourth level.
LFIFTH	Fifth level.
LSIXTH	Sixth level.
LSEVENTH	Seventh level.
LEIGHTH	Eighth level.
LNINETH	Ninth level.
LTENTH	Tenth level.

8.25.3.2 LevelScore

```
enum LevelScore
```

Represents the score levels in the game.

Enumerator

FIRST	Score level for the first tier.
SECOND	Score level for the second tier.
THIRD	Score level for the third tier.
FOURTH	Score level for the fourth tier.
FIFTH	Score level for the fifth tier.
SIXTH	Score level for the sixth tier.
SEVENTH	Score level for the seventh tier.

Enumerator

EIGHTH	Score level for the eighth tier.
NINETH	Score level for the ninth tier.
TENTH	Score level for the tenth tier.

8.25.4 Function Documentation

8.25.4.1 FSM_Collision()

```
void FSM_Collision (
            GameTetris * game )
```

Handles collision events in the game state machine.

Parameters

game	The game parameters.

8.26 Collision.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include "../../GameTetris.h"
00019
00020 #define MAX_INDEX_Y HFIELD - 1
00022 #define POINT_ONE_LINE 100
00023 #define POINT_TWO_LINE 300
00024 #define POINT_THREE_LINE 700
00025 #define POINT_FOUR_LINE 1500
00027 #define PROCENT_MAX 100
00028 #define DIVISORTOGETANUMBER 10
00029 #define MAX_SCORE 999999
00030 #define MIN_SCORE 0
00040 #define LESSOREQUUAL(first, second) (first <= second)
00041
00050 #define MORE(first, second) (first > second)
00051
00060 #define CHECKED_LEVEL(lfirst, lsecond, score) \
00061   (LESSOREQUUAL(lfirst, score) && MORE(lsecond, score))
00062
00066 typedef enum {
00067   FIRST = 600,
00068   SECOND = 1200,
00069   THIRD = 1800,
00070   FOURTH = 2400,
00071   FIFTH = 3000,
00072   SIXTH = 3600,
00073   SEVENTH = 4200,
00074   EIGHTH = 4800,
00075   NINETH = 5400,
00076   TENTH = 6000
00077 } LevelScore;
00078
00082 typedef enum {
00083   LFIRST = 1,
00084   LSECOND = 2,
00085   LTHIRD = 3,
```

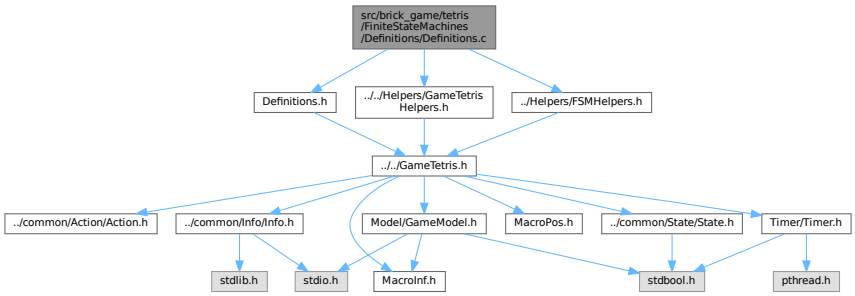
```
00086   LFOURTH = 4,
00087   LFIFTH = 5,
00088   LSIXTH = 6,
00089   LSEVENTH = 7,
00090   LEIGHTH = 8,
00091   LNINETH = 9,
00092   LTENTH = 10
00093 } LEVEL;
00094
00095 void FSM_Collision(GameTetris *game);
00096
00097 #ifdef TESTING
00098 unsigned int getTime(const GameInfo_t *engine);
00099 #else
00100 #endif
00101
00102 #ifdef __cplusplus
00103 }
00104 #endif
```

8.27 src/brick_game/tetris/FiniteStateMachines/Definitions/Definitions.c File Reference

Source file with the fsm `start`, fsm `gameover` of the game.

```
#include "Definitions.h"
#include "../../Helpers/GameTetrisHelpers.h"
#include "../Helpers/FSMHelpers.h"
```

Include dependency graph for Definitions.c:



Macros

- #define BEGIN 1
- #define END 2

Functions

- void FSM_Start (const UserAction_t action, GameTetris *game)

  Starts a new game or resumes a previous game.
- void FSM_GameOver (UserAction_t action, GameTetris *game)

  Ends a game or resets the game state.

8.27.1 Detailed Description

Source file with the fsm `start`, fsm `gameover` of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.27.2 Macro Definition Documentation

8.27.2.1 BEGIN

```
#define BEGIN 1
```

Code begin

8.27.2.2 END

```
#define END 2
```

Code end

8.27.3 Function Documentation

8.27.3.1 FSM_GameOver()

```
void FSM_GameOver (
            UserAction_t action,
            GameTetris * game )
```

Ends a game or resets the game state.

This function is called when the game ends or when the user wants to reset the game state. It updates the game state based on the given action and saves the high score if necessary.

Parameters

action	The action taken by the user (e.g. quit, restart, etc.).
game	The game object to be updated.

8.27.3.2 FSM_Start()

```
void FSM_Start (
            const UserAction_t action,
            GameTetris * game )
```

Starts a new game or resumes a previous game.

This function is called when the game starts or when the user wants to resume a previous game. It updates the game state based on the given action.

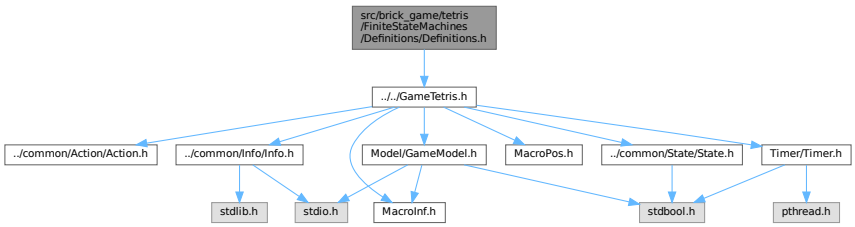
Parameters

action	The action taken by the user (e.g. start, resume, etc.).
game	The game object to be updated.

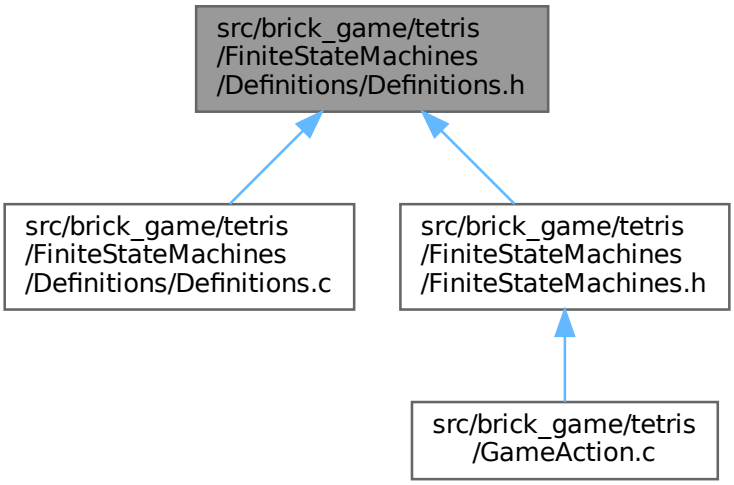
8.28 src/brick_game/tetris/FiniteStateMachines/Definitions/Definitions.h File Reference

Header file with the fsm `start`, fsm `gameover` of the game.

`#include "../../GameTetris.h"`
Include dependency graph for Definitions.h:



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



Functions

- void FSM_Start (const UserAction_t action, GameTetris *game)

  Starts a new game or resumes a previous game.
- void FSM_GameOver (UserAction_t action, GameTetris *game)

  Ends a game or resets the game state.

8.28.1 Detailed Description

Header file with the fsm `start`, fsm `gameover` of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

Copyright (c) 2024

8.28.2 Function Documentation

8.28.2.1 FSM_GameOver()

```
void FSM_GameOver (
            UserAction_t action,
            GameTetris * game )
```

Ends a game or resets the game state.

This function is called when the game ends or when the user wants to reset the game state. It updates the game state based on the given action and saves the high score if necessary.

Parameters

action	The action taken by the user (e.g. quit, restart, etc.).
game	The game object to be updated.

8.28.2.2 FSM_Start()

```
void FSM_Start (
            const UserAction_t action,
            GameTetris * game )
```

Starts a new game or resumes a previous game.

This function is called when the game starts or when the user wants to resume a previous game. It updates the game state based on the given action.

Parameters

action	The action taken by the user (e.g. start, resume, etc.).
game	The game object to be updated.

8.29 Definitions.h

Go to the documentation of this file.

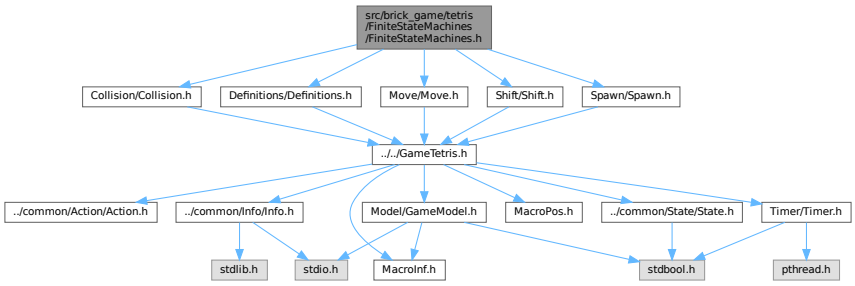
```
00001
00011 #pragma once
00012
00013 #ifdef __cplusplus
00014 extern "C" {
00015 #endif
00016
00017 #include "../../GameTetris.h"
00018 void FSM_Start(const UserAction_t action, GameTetris *game);
00019 void FSM_GameOver(UserAction_t action, GameTetris *game);
00020
00021 #ifdef __cplusplus
00022 }
00023 #endif
```

8.30 src/brick_game/tetris/FiniteStateMachines/FiniteStateMachines.h File Reference

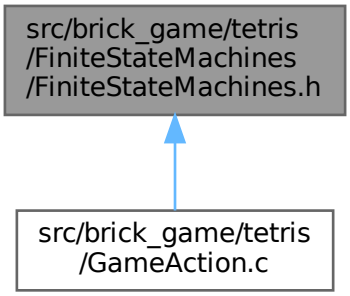
Header file with the reference to FSM.

```
#include "Collision/Collision.h"
#include "Definitions/Definitions.h"
#include "Move/Move.h"
#include "Shift/Shift.h"
#include "Spawn/Spawn.h"
```

Include dependency graph for FiniteStateMachines.h:



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



8.30.1 Detailed Description

Header file with the reference to FSM.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

Copyright (c) 2024

8.31 FiniteStateMachines.h

Go to the documentation of this file.

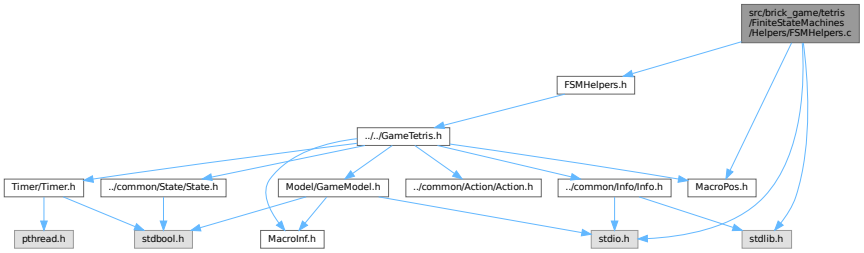
```
00001
00011 #pragma once
00012
00013 #include "Collision/Collision.h"
00014 #include "Definitions/Definitions.h"
00015 #include "Move/Move.h"
00016 #include "Shift/Shift.h"
00017 #include "Spawn/Spawn.h"
```

8.32 src/brick_game/tetris/FiniteStateMachines/Helpers/FSMHelpers.c File Reference

File in which all auxiliary functions for finite automata are stored.

```
#include "FSMHelpers.h"
#include <stdio.h>
#include <stdlib.h>
#include "../../MacroPos.h"
```

Include dependency graph for FSMHelpers.c:



Functions

- bool isNormalNextIndex (const GameTetris *game)
  Check if the next index is within the bounds of the models array.
- int getRandomIndex (const size_t maxIndex)
  Get a random index within the range of available models.
- void zeroingNext (GameInfo_t *engine)
  Reset the 'next' array in the game information structure.
- void zeroingField (GameInfo_t *engine)
  Reset the game field array in the game information structure.
- void zeroingInfo (GameTetris *game)
  Reset all information in the game parameters structure to initial values.
- void addedModelToField (GameTetris *game)
  Add current model to the game field.
- void deletModelInField (GameTetris *game)
  Delete current model from the game field.
- void setNextDooubleArray (GameTetris *game)
  Set the next double array based on the next model.
- bool isNormalCheckedPosition (const int x, const int y)
  Check if the given position is within the bounds of the field.

8.32.1 Detailed Description

File in which all auxiliary functions for finite automata are stored.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.32.2 Function Documentation

8.32.2.1 addedModelToField()

```
void addedModelToField (
            GameTetris * game )
```

Add current model to the game field.

This function adds the current model to the game field by calling the helperModelOperation function with the ADD mode.

Parameters

game	Pointer to the Brick Game parameters structure

8.32.2.2 deletModelInField()

```
void deletModelInField (
            GameTetris * game )
```

Delete current model from the game field.

This function deletes the current model from the game field by calling the helperModelOperation function with the DEL mode.

Parameters

game	Pointer to the Brick Game parameters structure

8.32.2.3 getRandomIndex()

```
int getRandomIndex (
            const size_t maxIndex )
```

Get a random index within the range of available models.

This function generates a random index within the range of available models.

Parameters

maxIndex	The number of available models

Returns

An integer representing the random index

8.32.2.4 isNormalCheckedPosition()

```
bool isNormalCheckedPosition (
            const int x,
            const int y )
```

Check if the given position is within the bounds of the field.

This function checks if the given position (x, y) is within the bounds of the game field.

Parameters

x	The x-coordinate of the position
y	The y-coordinate of the position

Returns

true if the position is within the bounds of the field, false otherwise

8.32.2.5 isNormalNextIndex()

```
bool isNormalNextIndex (
            const GameTetris * game )
```

Check if the next index is within the bounds of the models array.

Parameters

game	The GameTetris struct.

Returns

true If the next index is within the bounds of the models array.

false If the next index is outside the bounds of the models array.

8.32.2.6 setNextDooubleArray()

```
void setNextDooubleArray (
            GameTetris * game )
```

Set the next double array based on the next model.

This function sets the next double array in the Brick Game parameters structure based on the next model. It first clears the next double array by calling the zeroingNext function, then iterates through each cell of the next model and copies its values to the corresponding cells of the next double array.

Parameters

game	Pointer to the Brick Game parameters structure

8.32.2.7 zeroingField()

```
void zeroingField (
            GameInfo_t * engine )
```

Reset the game field array in the game information structure.

This function resets the game field array in the game information structure to false. It calls the zerroingDoubleArray function with the game field array pointer, along with the dimensions of the game field array.

Parameters

engine	Pointer to the game information structure

8.32.2.8 zeroingInfo()

```
void zeroingInfo (
            GameTetris * game )
```

Reset all information in the game parameters structure to initial values.

This function resets all information stored in the game parameters structure to its initial values. It deallocates memory for the current and next models, sets all cells in the game field to false, and resets the level, speed, and score.

Parameters

game	Pointer to the game parameters structure

8.32.2.9 zeroingNext()

```
void zeroingNext (
            GameInfo_t * engine )
```

Reset the 'next' array in the game information structure.

This function resets the 'next' array in the game information structure to false. It calls the zerroingDoubleArray function with the 'next' array pointer, along with the dimensions of the 'next' array.

Parameters

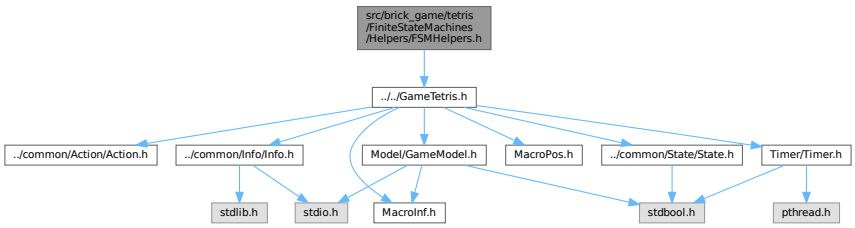
engine	Pointer to the game information structure

8.33 src/brick_game/tetris/FiniteStateMachines/Helpers/FSMHelpers.h File Reference

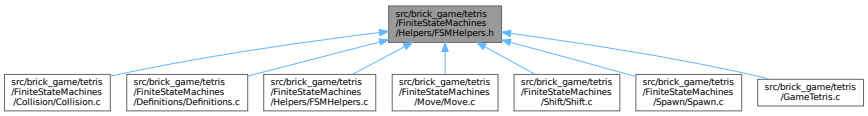
File in which all auxiliary functions for finite automata are stored.

```
#include "../../GameTetris.h"
```

Include dependency graph for FSMHelpers.h:



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



Macros

- #define ADD 100
- #define DEL -100

Functions

- int getRandomIndex (const size_t maxIndex)

  Get a random index within the range of available models.
- bool isNormalNextIndex (const GameTetris *game)

  Check if the next index is within the bounds of the models array.
- void zeroingNext (GameInfo_t *engine)

  Reset the 'next' array in the game information structure.
- void zeroingField (GameInfo_t *engine)

  Reset the game field array in the game information structure.
- void zeroingInfo (GameTetris *game)

  Reset all information in the game parameters structure to initial values.
- void setNextDooubleArray (GameTetris *game)

  Set the next double array based on the next model.
- void addedModelToField (GameTetris *game)

  Add current model to the game field.
- void deletModelInField (GameTetris *game)

  Delete current model from the game field.
- bool isNormalCheckedPosition (const int x, const int y)

  Check if the given position is within the bounds of the field.

8.33.1 Detailed Description

File in which all auxiliary functions for finite automata are stored.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.33.2 Macro Definition Documentation

8.33.2.1 ADD

```
#define ADD 100
```

Constant representing addition operation

8.33.2.2 DEL

```
#define DEL -100
```

Constant representing deletion operation

8.33.3 Function Documentation

8.33.3.1 addedModelToField()

```
void addedModelToField (
            GameTetris * game )
```

Add current model to the game field.

This function adds the current model to the game field by calling the helperModelOperation function with the `ADD` mode.

Parameters

game	Pointer to the Brick Game parameters structure

8.33.3.2 deletModelInField()

```
void deletModelInField (
            GameTetris * game )
```

Delete current model from the game field.

This function deletes the current model from the game field by calling the helperModelOperation function with the `DEL` mode.

Parameters

game	Pointer to the Brick Game parameters structure

8.33.3.3 getRandomIndex()

```
int getRandomIndex (
```

```
            const size_t maxIndex )
```

Get a random index within the range of available models.

This function generates a random index within the range of available models.

Parameters

maxIndex	The number of available models

Returns

An integer representing the random index

8.33.3.4 isNormalCheckedPosition()

```
bool isNormalCheckedPosition (
            const int x,
            const int y )
```

Check if the given position is within the bounds of the field.

This function checks if the given position (x, y) is within the bounds of the game field.

Parameters

x	The x-coordinate of the position
y	The y-coordinate of the position

Returns

true if the position is within the bounds of the field, false otherwise

8.33.3.5 isNormalNextIndex()

```
bool isNormalNextIndex (
            const GameTetris * game )
```

Check if the next index is within the bounds of the models array.

Parameters

game	The GameTetris struct.

Returns

true If the next index is within the bounds of the models array.

false If the next index is outside the bounds of the models array.

8.33.3.6 setNextDooubleArray()

```
void setNextDooubleArray (
            GameTetris * game )
```

Set the next double array based on the next model.

This function sets the next double array in the Brick Game parameters structure based on the next model. It first clears the next double array by calling the zeroingNext function, then iterates through each cell of the next model and copies its values to the corresponding cells of the next double array.

Parameters

game	Pointer to the Brick Game parameters structure

8.33.3.7 zeroingField()

```
void zeroingField (
            GameInfo_t * engine )
```

Reset the game field array in the game information structure.

This function resets the game field array in the game information structure to false. It calls the zerroingDoubleArray function with the game field array pointer, along with the dimensions of the game field array.

Parameters

engine	Pointer to the game information structure

8.33.3.8 zeroingInfo()

```
void zeroingInfo (
            GameTetris * game )
```

Reset all information in the game parameters structure to initial values.

This function resets all information stored in the game parameters structure to its initial values. It deallocates memory for the current and next models, sets all cells in the game field to false, and resets the level, speed, and score.

Parameters

game	Pointer to the game parameters structure

8.33.3.9 zeroingNext()

```
void zeroingNext (
            GameInfo_t * engine )
```

Reset the 'next' array in the game information structure.

This function resets the 'next' array in the game information structure to false. It calls the zerroingDoubleArray function with the 'next' array pointer, along with the dimensions of the 'next' array.

Parameters

engine	Pointer to the game information structure

8.34 FSMHelpers.h

Go to the documentation of this file.

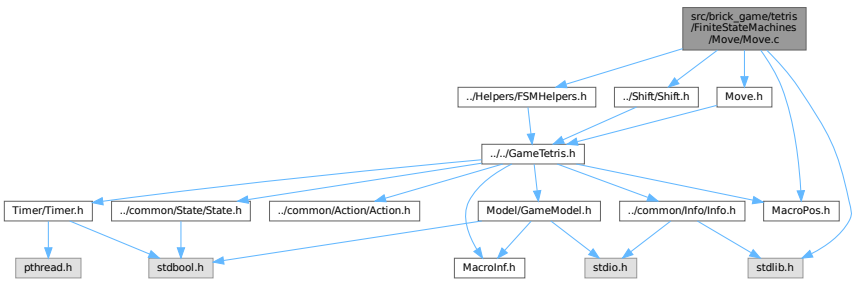
```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include "../../GameTetris.h"
00019
00020 #define ADD 100
00021 #define DEL -100
00023 int getRandomIndex(const size_t maxIndex);
00024 bool isNormalNextIndex(const GameTetris *game);
00025
00026 void zeroingNext(GameInfo_t *engine);
00027 void zeroingField(GameInfo_t *engine);
00028 void zeroingInfo(GameTetris *game);
00029
00030 void setNextDooubleArray(GameTetris *game);
00031 void addedModelToField(GameTetris *game);
00032 void deletModelInField(GameTetris *game);
00033 bool isNormalCheckedPosition(const int x, const int y);
00034
00035 #ifdef __cplusplus
00036 }
00037 #endif
```

8.35 src/brick_game/tetris/FiniteStateMachines/Move/Move.c File Reference

MOVE finite automaton.

```
#include "Move.h"
#include <stdlib.h>
#include "../../MacroPos.h"
#include "../Helpers/FSMHelpers.h"
#include "../Shift/Shift.h"
```

Include dependency graph for Move.c:



Functions

- void FSM_Move (const UserAction_t action, GameTetris *game)

  Handle move actions in the game state machine.

8.35.1 Detailed Description

MOVE finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-16

Copyright

Copyright (c) 2024

8.35.2 Function Documentation

8.35.2.1 FSM_Move()

```
void FSM_Move (
            const UserAction_t action,
            GameTetris * game )
```

Handle move actions in the game state machine.

This function handles move actions in the game state machine. It performs different actions based on the given action parameter:

- If the action is Left or Right, it moves the current model horizontally. - If the action is Action, it rotates the current model. - If the action is Down, it continuously moves the current model downward until it collides.

Parameters

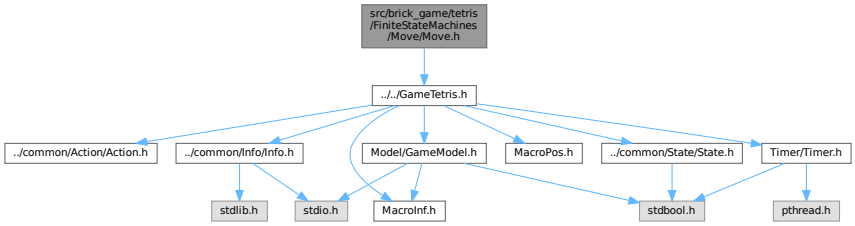
action	The action to handle.
game	The game parameters.

8.36 src/brick_game/tetris/FiniteStateMachines/Move/Move.h File Reference

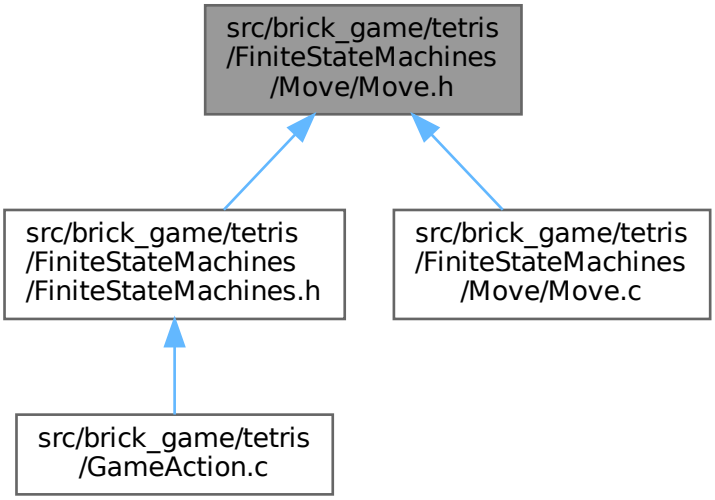
MOVE finite automaton.

```
#include "../../GameTetris.h"
```

Include dependency graph for Move.h:



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



Macros

- #define GET_COEF_MOVE(direction) (direction == Left) ? -1 : 1

  Get the coefficient for movement.
- #define BACK_POS_X(cols, i) cols - i - 1

  Calculates the backward position along the X-axis.
- #define BACK_POS_Y(rows, i) rows - i - 1

  Calculates the backward position along the Y-axis.
- #define HALF_ROWS(rows) (rows / 2)

  Calculates the half number of rows.
- #define HALF_COLS(cols) (cols / 2)

  Calculates the half number of columns.

Functions

- void FSM_Move (const UserAction_t action, GameTetris *game)

  Handle move actions in the game state machine.

8.36.1 Detailed Description

MOVE finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-16

Copyright

Copyright (c) 2024

8.36.2 Macro Definition Documentation

8.36.2.1 BACK_POS_X

```
#define BACK_POS_X(
              cols,
              i ) cols - i - 1
```

Calculates the backward position along the X-axis.

This macro calculates the backward position along the X-axis based on the total number of columns and the current index.

Parameters

cols	Total number of columns.
i	Current index.

Returns

int Backward position along the X-axis.

8.36.2.2 BACK_POS_Y

```
#define BACK_POS_Y(
              rows,
              i ) rows - i - 1
```

Calculates the backward position along the Y-axis.

This macro calculates the backward position along the Y-axis based on the total number of rows and the current index.

Parameters

rows	Total number of rows.
i	Current index.

Returns

int Backward position along the Y-axis.

8.36.2.3 GET_COEF_MOVE

```
#define GET_COEF_MOVE(
              direction ) (direction == Left) ?  -1 :  1
```

Get the coefficient for movement.

This macro calculates the coefficient for movement based on the direction. If the direction is Left, the coefficient is -1, otherwise it is 1.

Parameters

direction	The direction of movement.

Returns

int The coefficient for movement.

8.36.2.4 HALF_COLS

```
#define HALF_COLS(
              cols ) (cols / 2)
```

Calculates the half number of columns.

This macro calculates the half number of columns based on the total number of columns.

Parameters

cols	Total number of columns.

Returns

int Half number of columns.

8.36.2.5 HALF_ROWS

```
#define HALF_ROWS(
              rows ) (rows / 2)
```

Calculates the half number of rows.

This macro calculates the half number of rows based on the total number of rows.

Parameters

rows	Total number of rows.

Returns

int Half number of rows.

8.36.3 Function Documentation

8.36.3.1 FSM_Move()

```
void FSM_Move (
            const UserAction_t action,
            GameTetris * game )
```

Handle move actions in the game state machine.

This function handles move actions in the game state machine. It performs different actions based on the given action parameter:

− If the action is Left or Right, it moves the current model horizontally. − If the action is Action, it rotates the current model. − If the action is Down, it continuously moves the current model downward until it collides.

Parameters

action	The action to handle.
game	The game parameters.

8.37 Move.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013 #ifdef __cplusplus
00014 extern "C" {
```

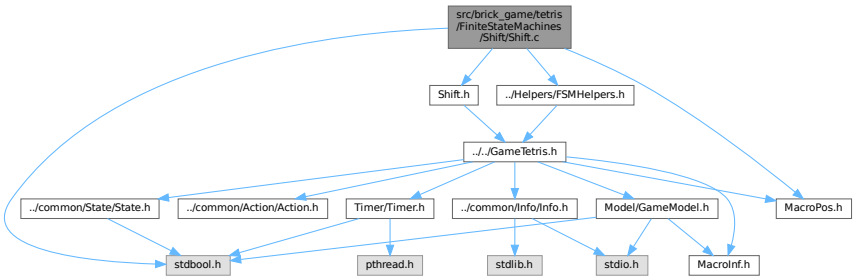
```
00015 #endif
00016
00017 #include "../../GameTetris.h"
00018
00028 #define GET_COEF_MOVE(direction) (direction == Left) ? -1 : 1
00029
00040 #define BACK_POS_X(cols, i) cols - i - 1
00041
00052 #define BACK_POS_Y(rows, i) rows - i - 1
00053
00063 #define HALF_ROWS(rows) (rows / 2)
00064
00074 #define HALF_COLS(cols) (cols / 2)
00075
00076 void FSM_Move(const UserAction_t action, GameTetris* game);
00077
00078 #ifdef __cplusplus
00079 }
00080 #endif
```

8.38 src/brick_game/tetris/FiniteStateMachines/Shift/Shift.c File Reference

SHIFT finite automaton.

```
#include "Shift.h"
#include <stdbool.h>
#include "../../MacroPos.h"
#include "../Helpers/FSMHelpers.h"
```

Include dependency graph for Shift.c:



Functions

- void FSM_Shift (GameTetris *game)

  Shifts the current model downwards if possible.

8.38.1 Detailed Description

SHIFT finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

8.38.2 Function Documentation

8.38.2.1 FSM_Shift()

```
void FSM_Shift (
            GameTetris * game )
```

Shifts the current model downwards if possible.

Parameters

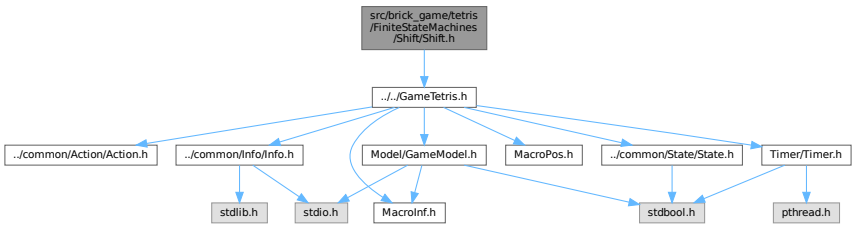
game	Pointer to the game parameters.

8.39 src/brick_game/tetris/FiniteStateMachines/Shift/Shift.h File Reference

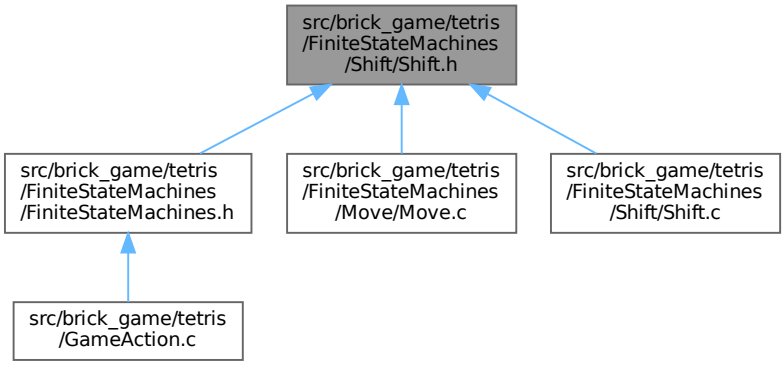
SHIFT finite automaton.

```
#include "../../GameTetris.h"
```

Include dependency graph for Shift.h:



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



Macros

- #define COEF_POS_SHIFT 1

Functions

- void FSM_Shift (GameTetris *game)

  Shifts the current model downwards if possible.

8.39.1 Detailed Description

SHIFT finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.39.2 Macro Definition Documentation

8.39.2.1 COEF_POS_SHIFT

```
#define COEF_POS_SHIFT 1
```

The coefficient for position shifting.

8.39.3 Function Documentation

8.39.3.1 FSM_Shift()

```
void FSM_Shift (
            GameTetris * game )
```

Shifts the current model downwards if possible.

Parameters

game	Pointer to the game parameters.

8.40 Shift.h

Go to the documentation of this file.

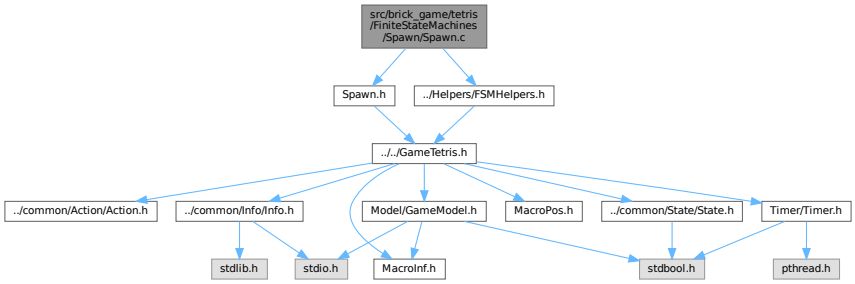
```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include "../../GameTetris.h"
00019
00020 #define COEF_POS_SHIFT 1
00022 void FSM_Shift(GameTetris* game);
00023
00024 #ifdef __cplusplus
00025 }
00026 #endif
```

8.41 src/brick_game/tetris/FiniteStateMachines/Spawn/Spawn.c File Reference

SPAWN finite automaton.

```
#include "Spawn.h"
#include "../Helpers/FSMHelpers.h"
```

Include dependency graph for Spawn.c:



Functions

- void FSM_Spawn (GameTetris *game)

  Handles the spawn state of the finite state machine.

8.41.1 Detailed Description

SPAWN finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.41.2 Function Documentation

8.41.2.1 FSM_Spawn()

```
void FSM_Spawn (
            GameTetris * game )
```

Handles the spawn state of the finite state machine.

This function updates the current and next models in the Brick Game Parameters (game) and transitions the state of the finite state machine based on the availability of the models.

Parameters

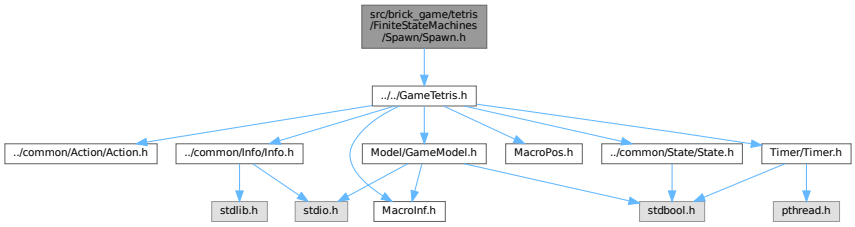
game	Pointer to the Brick Game Parameters structure.

8.42 src/brick_game/tetris/FiniteStateMachines/Spawn/Spawn.h File Reference

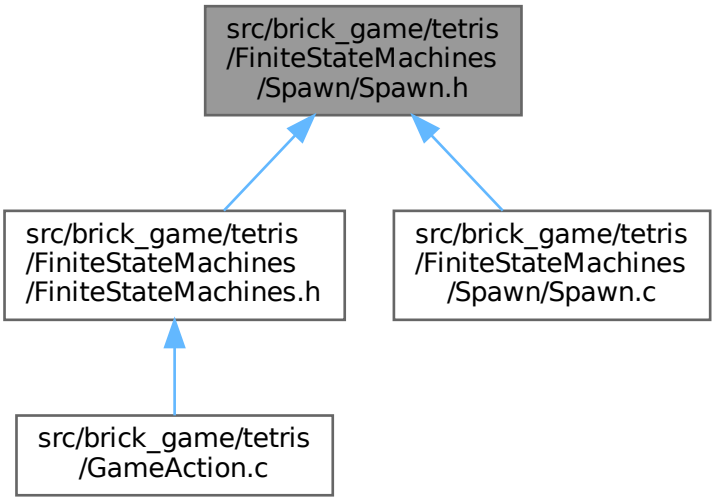
SPAWN finite automaton.

```
#include "../../GameTetris.h"
```

Include dependency graph for Spawn.h:



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



Macros

- #define START_X 4
- #define START_Y -1
- #define GET_RANDOM_MODEL(game) game->models.models[getRandomIndex(game->models.count)]

  Retrieves a random model from the available models.

Functions

- void FSM_Spawn (GameTetris *game)

  Handles the spawn state of the finite state machine.

8.42.1 Detailed Description

SPAWN finite automaton.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.42.2 Macro Definition Documentation

8.42.2.1 GET_RANDOM_MODEL

```
#define GET_RANDOM_MODEL(
              game )  game->models.models[getRandomIndex(game->models.count)]
```

Retrieves a random model from the available models.

This macro returns a randomly selected model from the models stored in the Brick Game Parameters (game).

Parameters

game	Pointer to the Brick Game Parameters structure containing the models.

Returns

The randomly selected model from the available models.

8.42.2.2 START_X

```
#define START_X 4
```

Start position x

8.42.2.3 START_Y

```
#define START_Y -1
```

Start position `y`

8.42.3 Function Documentation

8.42.3.1 FSM_Spawn()

```
void FSM_Spawn (
            GameTetris * game )
```

Handles the spawn state of the finite state machine.

This function updates the current and next models in the Brick Game Parameters (game) and transitions the state of the finite state machine based on the availability of the models.

Parameters

game	Pointer to the Brick Game Parameters structure.

8.43 Spawn.h

Go to the documentation of this file.

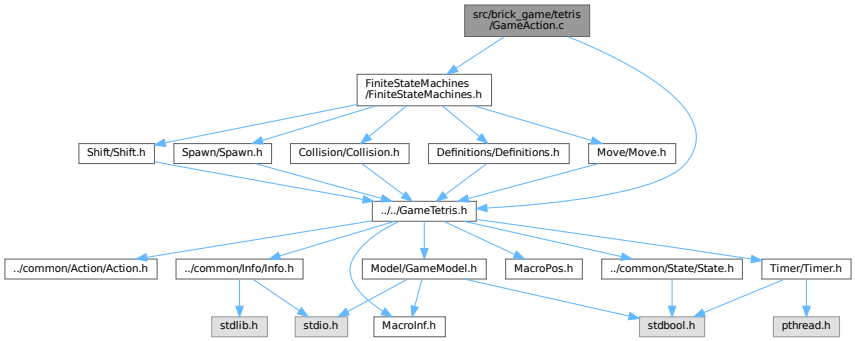
```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include "../../GameTetris.h"
00019
00020 #define START_X 4
00021 #define START_Y -1
00034 #define GET_RANDOM_MODEL(game) \
00035   game->models.models[getRandomIndex(game->models.count)]
00036
00037 void FSM_Spawn(GameTetris *game);
00038
00039 #ifdef __cplusplus
00040 }
00041 #endif
```

8.44 src/brick_game/tetris/GameAction.c File Reference

File with the basic game action.

```
#include "FiniteStateMachines/FiniteStateMachines.h"
#include "GameTetris.h"
```

Include dependency graph for GameAction.c:



Functions

- void userInput (UserAction_t action, bool hold)

  Processes user input based on the current game state.

8.44.1 Detailed Description

File with the basic game action.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.44.2 Function Documentation

8.44.2.1 userInput()

```
void userInput (
            UserAction_t action,
            bool hold )
```

Processes user input based on the current game state.

Parameters

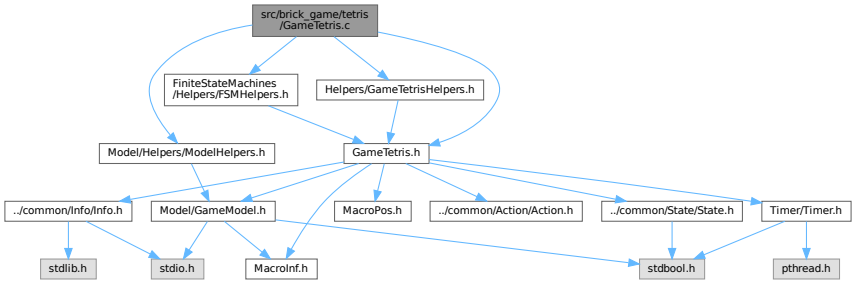
action	The user action to process.
hold	Indicates whether the action is being held.

8.45 src/brick_game/tetris/GameTetris.c File Reference

File with the basic structures of the game.

```
#include "GameTetris.h"
#include "FiniteStateMachines/Helpers/FSMHelpers.h"
#include "Helpers/GameTetrisHelpers.h"
#include "Model/Helpers/ModelHelpers.h"
```

Include dependency graph for GameTetris.c:



Functions

- int initializeGameInfo (GameInfo_t *engine)

  Initializes the GameInfo_t structure.
- int initializeGameTetris (GameTetris *game)

  Initializes the GameTetris structure.
- void cleanGameInfo (GameInfo_t *engine)

  Frees the memory allocated for the GameInfo_t structure.
- void cleanGameTetris (GameTetris *game)

  Frees the memory allocated for the GameTetris structure.
- GameTetris * updateParams (GameTetris *game)

  Updates the game parameters.
- GameInfo_t updateCurrentState (void)

  Updates and retrieves the current game state information.
- void reset (GameTetris *game, const GameState_t reset_state)

  Resets the game state to a specific state.

8.45.1 Detailed Description

File with the basic structures of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.45.2 Function Documentation

8.45.2.1 cleanGameInfo()

```
void cleanGameInfo (
            GameInfo_t * engine )
```

Frees the memory allocated for the GameInfo_t structure.

This function frees the memory allocated for the field and next arrays in the GameInfo_t structure and resets the score, high score, level, speed, and pause attributes.

Parameters

engine	Pointer to the GameInfo_t structure to be freed.

8.45.2.2 cleanGameTetris()

```
void cleanGameTetris (
            GameTetris * game )
```

Frees the memory allocated for the GameTetris structure.

This function frees the memory allocated for the GameInfo_t structure, current and next models, and the models array in the GameTetris structure. Additionally, it sets the state attribute to EXIT.

Parameters

game	Pointer to the GameTetris structure to be freed.

8.45.2.3 initializeGameInfo()

```
int initializeGameInfo (
            GameInfo_t * engine )
```

Initializes the GameInfo_t structure.

This function initializes the GameInfo_t structure by allocating memory for the game field and next figures, setting initial score, high score, level, speed, and pause state.

Parameters

engine	Pointer to the GameInfo_t structure to be initialized.

8.45.2.4 initializeGameTetris()

```
int initializeGameTetris (
            GameTetris * game )
```

Initializes the GameTetris structure.

This function initializes the GameTetris structure by initializing the game information, models, and setting the game state to START.

Parameters

game	Pointer to the GameTetris structure to be initialized.

8.45.2.5 reset()

```
void reset (
            GameTetris * game,
            const GameState_t reset_state )
```

Resets the game state to a specific state.

Parameters

game	The struct GameTetris.
reset_state	The state to which the game should be moved

8.45.2.6 updateCurrentState()

```
GameInfo_t updateCurrentState (
            void  )
```

Updates and retrieves the current game state information.

This function updates and retrieves the current game state information, including the field, next piece, score, level, speed, and pause state.

Returns

GameInfo_t The updated game state information.

8.45.2.7 updateParams()

```
GameTetris * updateParams (
            GameTetris * game )
```

Updates the game parameters.

This function updates the game parameters and returns a pointer to the updated GameTetris structure.

Parameters

game	Pointer to the GameTetris structure containing the updated parameters.

Returns

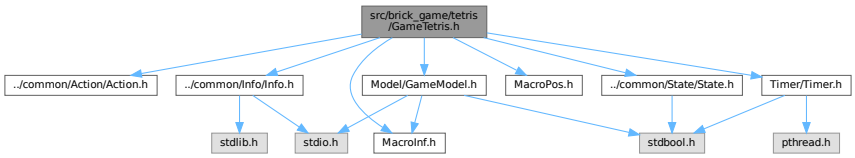
GameTetris* Pointer to the updated GameTetris structure.

8.46 src/brick_game/tetris/GameTetris.h File Reference

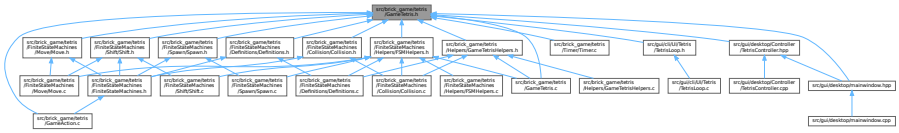
File with the basic structures of the game.

```
#include "../common/Action/Action.h"
#include "../common/Info/Info.h"
#include "../common/State/State.h"
#include "MacroInf.h"
#include "MacroPos.h"
#include "Model/GameModel.h"
#include "Timer/Timer.h"
```

Include dependency graph for GameTetris.h:



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



Classes

- struct GameTetris

  Structure representing game parameters.

Functions

- void userInput (UserAction_t action, bool hold)

  Processes user input based on the current game state.
- int initializeGameInfo (GameInfo_t *engine)

  Initializes the GameInfo_t structure.
- int initializeGameTetris (GameTetris *game)

  Initializes the GameTetris structure.
- void cleanGameInfo (GameInfo_t *engine)

  Frees the memory allocated for the GameInfo_t structure.
- void cleanGameTetris (GameTetris *game)

  Frees the memory allocated for the GameTetris structure.
- GameTetris * updateParams (GameTetris *game)

  Updates the game parameters.
- GameInfo_t updateCurrentState (void)

  Updates and retrieves the current game state information.
- void reset (GameTetris *game, const GameState_t reset_state)

  Resets the game state to a specific state.

8.46.1 Detailed Description

File with the basic structures of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.46.2 Function Documentation

8.46.2.1 cleanGameInfo()

```
void cleanGameInfo (
            GameInfo_t * engine )
```

Frees the memory allocated for the GameInfo_t structure.

This function frees the memory allocated for the field and next arrays in the GameInfo_t structure and resets the score, high score, level, speed, and pause attributes.

Parameters

engine	Pointer to the GameInfo_t structure to be freed.

8.46.2.2 cleanGameTetris()

```
void cleanGameTetris (
            GameTetris * game )
```

Frees the memory allocated for the GameTetris structure.

This function frees the memory allocated for the GameInfo_t structure, current and next models, and the models array in the GameTetris structure. Additionally, it sets the state attribute to EXIT.

Parameters

game	Pointer to the GameTetris structure to be freed.

8.46.2.3 initializeGameInfo()

```
int initializeGameInfo (
            GameInfo_t * engine )
```

Initializes the GameInfo_t structure.

This function initializes the GameInfo_t structure by allocating memory for the game field and next figures, setting initial score, high score, level, speed, and pause state.

Parameters

engine	Pointer to the GameInfo_t structure to be initialized.

8.46.2.4 initializeGameTetris()

```
int initializeGameTetris (
            GameTetris * game )
```

Initializes the GameTetris structure.

This function initializes the GameTetris structure by initializing the game information, models, and setting the game state to START.

Parameters

game	Pointer to the GameTetris structure to be initialized.

8.46.2.5 reset()

```
void reset (
            GameTetris * game,
            const GameState_t reset_state )
```

Resets the game state to a specific state.

Parameters

game	The struct GameTetris.
reset_state	The state to which the game should be moved

8.46.2.6 updateCurrentState()

```
GameInfo_t updateCurrentState (
            void  )
```

Updates and retrieves the current game state information.

This function updates and retrieves the current game state information, including the field, next piece, score, level, speed, and pause state.

Returns

GameInfo_t The updated game state information.

8.46.2.7 updateParams()

```
GameTetris * updateParams (
            GameTetris * game )
```

Updates the game parameters.

This function updates the game parameters and returns a pointer to the updated GameTetris structure.

Parameters

game	Pointer to the GameTetris structure containing the updated parameters.

Returns

GameTetris* Pointer to the updated GameTetris structure.

8.46.2.8 userInput()

```
void userInput (
            UserAction_t action,
            bool hold )
```

Processes user input based on the current game state.

Parameters

action	The user action to process.
hold	Indicates whether the action is being held.

8.47 GameTetris.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include "../common/Action/Action.h"
00019 #include "../common/Info/Info.h"
00020 #include "../common/State/State.h"
00021 #include "MacroInf.h"
00022 #include "MacroPos.h"
00023 #include "Model/GameModel.h"
00024 #include "Timer/Timer.h"
00025
00029 typedef struct {
00030   Models models;
00031   Model current;
00032   int current_color;
00033   size_t index_next;
00034   GameState_t state;
00035   GameTimer_t timer;
00036   GameInfo_t engine;
00037 } GameTetris;
00038
00039 void userInput(UserAction_t action, bool hold);
00040
00041 int initializeGameInfo(GameInfo_t *engine);
00042 int initializeGameTetris(GameTetris *game);
00043 void cleanGameInfo(GameInfo_t *engine);
00044 void cleanGameTetris(GameTetris *game);
00045 GameTetris *updateParams(GameTetris *game);
00046 GameInfo_t updateCurrentState(void);
00047 void reset(GameTetris *game, const GameState_t reset_state);
00048
00049 #ifdef __cplusplus
00050 }
00051 #endif
```

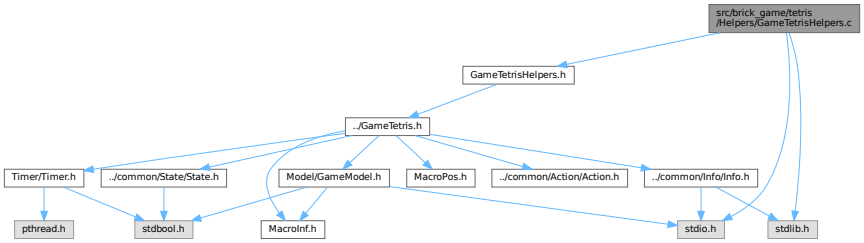
8.48 src/brick_game/tetris/Helpers/GameTetrisHelpers.c File Reference

This file describes auxiliary functions that are stored in the type directory, auxiliary functions for allocating and clearing memory and so on.

```
#include "GameTetrisHelpers.h"
#include <stdio.h>
```

```
#include <stdlib.h>
```

Include dependency graph for GameTetrisHelpers.c:



Functions

- void initializeHighScore (GameInfo_t *engine)

  Function to initialize the HighScore. Information is taken from a special file where information is stored, the function parses the file and retrieves information by a special literal.
- void saveHighScore (const GameInfo_t *engine)

  Function for saving HighScore to a file, saves only if the points scored exceed the current score record.

8.48.1 Detailed Description

This file describes auxiliary functions that are stored in the type directory, auxiliary functions for allocating and clearing memory and so on.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-13

Copyright

Copyright (c) 2024

8.48.2 Function Documentation

8.48.2.1 initializeHighScore()

```
void initializeHighScore (
            GameInfo_t * engine )
```

Function to initialize the HighScore. Information is taken from a special file where information is stored, the function parses the file and retrieves information by a special literal.

Parameters

engine	Pointer to the `GameInfo_t` structure, where the highScore will be written.

8.48.2.2 saveHighScore()

```
void saveHighScore (
            const GameInfo_t * engine )
```

Function for saving HighScore to a file, saves only if the points scored exceed the current score record.

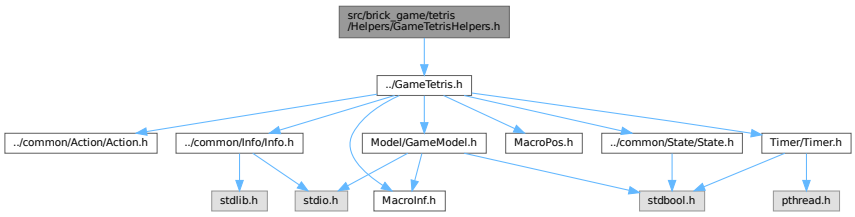
Parameters

engine	Pointer to the `GameInfo_t` structure from which the information is taken.

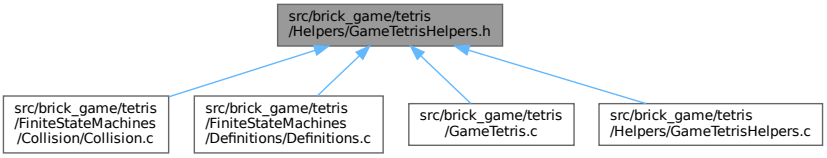
8.49 src/brick_game/tetris/Helpers/GameTetrisHelpers.h File Reference

This file describes auxiliary functions that are stored in the type directory, auxiliary functions for allocating and clearing memory and so on.

`#include "../GameTetris.h"`
Include dependency graph for GameTetrisHelpers.h:



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



Macros

- #define **FILENAME_BD** "BD_Tetris.txt"

  The name of the file to store the information.
- #define **HIGH_SCORE_LITTERAL** "High Score: %d"

  Literals for reading and writing information from a file.

Functions

- void saveHighScore (const GameInfo_t *engine)

  Function for saving HighScore to a file, saves only if the points scored exceed the current score record.
- void initializeHighScore (GameInfo_t *engine)

  Function to initialize the HighScore. Information is taken from a special file where information is stored, the function parses the file and retrieves information by a special literal.

8.49.1 Detailed Description

This file describes auxiliary functions that are stored in the type directory, auxiliary functions for allocating and clearing memory and so on.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-13

Copyright

Copyright (c) 2024

8.49.2 Function Documentation

8.49.2.1 initializeHighScore()

```
void initializeHighScore (
            GameInfo_t * engine )
```

Function to initialize the HighScore. Information is taken from a special file where information is stored, the function parses the file and retrieves information by a special literal.

Parameters

engine	Pointer to the GameInfo_t structure, where the highScore will be written.

8.49.2.2 saveHighScore()

```
void saveHighScore (
            const GameInfo_t * engine )
```

Function for saving HighScore to a file, saves only if the points scored exceed the current score record.

Parameters

engine	Pointer to the `GameInfo_t` structure from which the information is taken.

8.50 GameTetrisHelpers.h

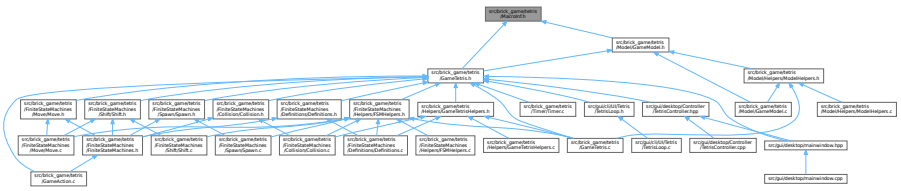
Go to the documentation of this file.

```
00001
00013 #pragma once
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 #include "../GameTetris.h"
00020
00024 #define FILENAME_BD "BD_Tetris.txt"
00025
00029 #define HIGH_SCORE_LITTERAL "High Score: %d"
00030
00031 void saveHighScore(const GameInfo_t *engine);
00032 void initializeHighScore(GameInfo_t *engine);
00033
00034 #ifdef __cplusplus
00035 }
00036 #endif
```

8.51 src/brick_game/tetris/MacroInf.h File Reference

Macro file.

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



Macros

- #define ST_MODELS_COUNT 7
- #define FAIL false
- #define SUCCESS true
- #define SIZE_BUFFER 1024
- #define SIZE_COORD 2
- #define **MAX_LEVEL** 10

  The maximum level of the game.
- #define **MIN_LEVEL** 1

  The minimum level of the game.
- #define **INIT_LEVEL** MIN_LEVEL

The initial level of the game.

- #define **INIT_SCORE** 0

  The initial score of the game.

- #define COEF_SPEED_LEVEL_1 250000000
- #define COEF_SPEED_LEVEL_2 240000000
- #define COEF_SPEED_LEVEL_3 230000000
- #define COEF_SPEED_LEVEL_4 220000000
- #define COEF_SPEED_LEVEL_5 210000000
- #define COEF_SPEED_LEVEL_6 200000000
- #define COEF_SPEED_LEVEL_7 190000000
- #define COEF_SPEED_LEVEL_8 180000000
- #define COEF_SPEED_LEVEL_9 170000000
- #define COEF_SPEED_LEVEL_10 160000000
- #define NGAME_TETRIS "TETRIS"
- #define NAME_FILE_TEMPLATE "Model_templates.txt"

8.51.1 Detailed Description

Macro file.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-24

Copyright

8.51.2 Macro Definition Documentation

8.51.2.1 COEF_SPEED_LEVEL_1

```
#define COEF_SPEED_LEVEL_1 250000000
```

Speed coefficient for level 1.

8.51.2.2 COEF_SPEED_LEVEL_10

```
#define COEF_SPEED_LEVEL_10 160000000
```

Speed coefficient for level 10.

8.51.2.3 COEF_SPEED_LEVEL_2

```
#define COEF_SPEED_LEVEL_2 240000000
```

Speed coefficient for level 2.

8.51.2.4 COEF_SPEED_LEVEL_3

```
#define COEF_SPEED_LEVEL_3 230000000
```

Speed coefficient for level 3.

8.51.2.5 COEF_SPEED_LEVEL_4

```
#define COEF_SPEED_LEVEL_4 220000000
```

Speed coefficient for level 4.

8.51.2.6 COEF_SPEED_LEVEL_5

```
#define COEF_SPEED_LEVEL_5 210000000
```

Speed coefficient for level 5.

8.51.2.7 COEF_SPEED_LEVEL_6

```
#define COEF_SPEED_LEVEL_6 200000000
```

Speed coefficient for level 6.

8.51.2.8 COEF_SPEED_LEVEL_7

```
#define COEF_SPEED_LEVEL_7 190000000
```

Speed coefficient for level 7.

8.51.2.9 COEF_SPEED_LEVEL_8

```
#define COEF_SPEED_LEVEL_8 180000000
```

Speed coefficient for level 8.

8.51.2.10 COEF_SPEED_LEVEL_9

```
#define COEF_SPEED_LEVEL_9 170000000
```

Speed coefficient for level 9.

8.51.2.11 FAIL

```
#define FAIL false
```

Represents a failed operation or function call.

8.51.2.12 NAME_FILE_TEMPLATE

```
#define NAME_FILE_TEMPLATE  "Model_templates.txt"
```

The name of the template file used in the model. \

8.51.2.13 NGAME_TETRIS

```
#define NGAME_TETRIS "TETRIS"
```

The name of the game TETRIS.

8.51.2.14 SIZE_BUFFER

```
#define SIZE_BUFFER 1024
```

The size of the buffer.

8.51.2.15 SIZE_COORD

```
#define SIZE_COORD 2
```

Number of coordinates

8.51.2.16 ST_MODELS_COUNT

```
#define ST_MODELS_COUNT 7
```

Standart count models

8.51.2.17 SUCCESS

```
#define SUCCESS  true
```

Represents a successful operation or function \ call.

8.52 MacroInf.h

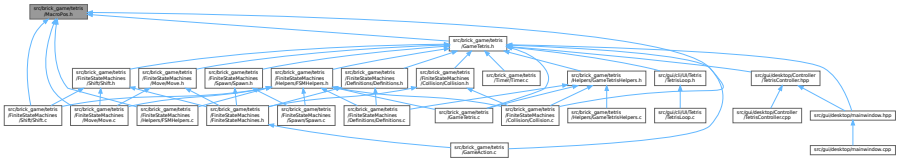
Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #define ST_MODELS_COUNT 7
00020 #define FAIL false
00021 #define SUCCESS                                                    \
00022   true
00025 #define SIZE_BUFFER 1024
00026 #define SIZE_COORD 2
00031 #define MAX_LEVEL 10
00032
00036 #define MIN_LEVEL 1
00037
00041 #define INIT_LEVEL MIN_LEVEL
00042
00046 #define INIT_SCORE 0
00047
00048 #define COEF_SPEED_LEVEL_1 250000000
00049 #define COEF_SPEED_LEVEL_2 240000000
00050 #define COEF_SPEED_LEVEL_3 230000000
00051 #define COEF_SPEED_LEVEL_4 220000000
00052 #define COEF_SPEED_LEVEL_5 210000000
00053 #define COEF_SPEED_LEVEL_6 200000000
00054 #define COEF_SPEED_LEVEL_7 190000000
00055 #define COEF_SPEED_LEVEL_8 180000000
00056 #define COEF_SPEED_LEVEL_9 170000000
00057 #define COEF_SPEED_LEVEL_10 160000000
00059 #define NGAME_TETRIS "TETRIS"
00060 #define NAME_FILE_TEMPLATE                                         \
00061   "Model_templates.txt"
00064 #ifdef __cplusplus
00065 }
00066 #endif
```

8.53 src/brick_game/tetris/MacroPos.h File Reference

Header file with the macros position of the game.

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



Macros

- #define GET_CURRENT_POS_X(game) game.current.position[X]

  Retrieves the X-coordinate of the current model's position.
- #define GET_CURRENT_POS_Y(game) game.current.position[Y]

  Retrieves the Y-coordinate of the current model's position.
- #define GET_CURRENT_MID_X(game) game.current.center[X]

  Retrieves the X-coordinate of the center of the current model.
- #define GET_CURRENT_MID_Y(game) game.current.center[Y]

  Retrieves the Y-coordinate of the center of the current model.

- #define GET_CURRENT_COLS(game) game.current.cols
  Retrieves the number of columns in the current model.
- #define GET_CURRENT_ROWS(game) game.current.rows
  Retrieves the number of rows in the current model.
- #define GET_NEXT_POS_X(game) game.next.position[X]
  Retrieves the X-coordinate of the next model's position.
- #define GET_NEXT_POS_Y(game) game.next.position[Y]
  Retrieves the Y-coordinate of the next model's position.
- #define GET_NEXT_MID_X(game) game.next.center[X]
  Retrieves the X-coordinate of the center of the next model.
- #define GET_NEXT_MID_Y(game) game.next.center[Y]
  Retrieves the Y-coordinate of the center of the next model.
- #define GET_NEXT_COLS(game) game.next.cols
  Retrieves the number of columns in the next model.
- #define GET_NEXT_ROWS(game) game.next.rows
  Retrieves the number of rows in the next model.
- #define CELL_NEXT_MODEL(i, j, game) game.next.model_[i][j]
  Retrieves the value of the cell in the next model at the specified position.
- #define CELL_CURRENT_MODEL(i, j, game) game.current.model_[i][j]
  Retrieves the value of the cell in the current model at the specified position.
- #define CELL_FIELD(i, j, game) game.engine.field[i][j]
  Retrieves the value of the cell in the game field at the specified position.
- #define CELL_NEXT(i, j, game) game.engine.next[i][j]
  Retrieves the value of the cell in the next model at the specified position in the game field.

8.53.1 Detailed Description

Header file with the macros position of the game.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-09

Copyright

8.53.2 Macro Definition Documentation

8.53.2.1 CELL_CURRENT_MODEL

```
#define CELL_CURRENT_MODEL(
              i,
              j,
              game ) game.current.model_[i][j]
```

Retrieves the value of the cell in the current model at the specified position.

Parameters

i	The row index.
j	The column index.
game	The GameTetris structure containing information about the current model.

Returns

bool The value of the cell in the current model at the specified position.

8.53.2.2 CELL_FIELD

```
#define CELL_FIELD(
                i,
                j,
                game ) game.engine.field[i][j]
```

Retrieves the value of the cell in the game field at the specified position.

Parameters

i	The row index.
j	The column index.
game	The GameTetris structure containing information about the game field.

Returns

bool The value of the cell in the game field at the specified position.

8.53.2.3 CELL_NEXT

```
#define CELL_NEXT(
                i,
                j,
                game ) game.engine.next[i][j]
```

Retrieves the value of the cell in the next model at the specified position in the game field.

Parameters

i	The row index.
j	The column index.
game	The GameTetris structure containing information about the game field.

Returns

bool The value of the cell in the next model at the specified position in the game field.

8.53.2.4 CELL_NEXT_MODEL

```
#define CELL_NEXT_MODEL(
              i,
              j,
              game ) game.next.model_[i][j]
```

Retrieves the value of the cell in the next model at the specified position.

Parameters

i	The row index.
j	The column index.
game	The GameTetris structure containing information about the next model.

Returns

bool The value of the cell in the next model at the specified position.

8.53.2.5 GET_CURRENT_COLS

```
#define GET_CURRENT_COLS(
              game ) game.current.cols
```

Retrieves the number of columns in the current model.

Parameters

game	The GameTetris structure containing information about the current model.

Returns

size_t The number of columns in the current model.

8.53.2.6 GET_CURRENT_MID_X

```
#define GET_CURRENT_MID_X(
              game ) game.current.center[X]
```

Retrieves the X-coordinate of the center of the current model.

Parameters

game	The GameTetris structure containing information about the current model.

Returns

int The X-coordinate of the center of the current model.

8.53.2.7 GET_CURRENT_MID_Y

```
#define GET_CURRENT_MID_Y(
              game ) game.current.center[Y]
```

Retrieves the Y-coordinate of the center of the current model.

Parameters

game	The GameTetris structure containing information about the current model.

Returns

int The Y-coordinate of the center of the current model.

8.53.2.8 GET_CURRENT_POS_X

```
#define GET_CURRENT_POS_X(
              game ) game.current.position[X]
```

Retrieves the X-coordinate of the current model's position.

Parameters

game	The GameTetris structure containing information about the current model.

Returns

int The X-coordinate of the current model's position.

8.53.2.9 GET_CURRENT_POS_Y

```
#define GET_CURRENT_POS_Y(
              game ) game.current.position[Y]
```

Retrieves the Y-coordinate of the current model's position.

Parameters

game	The GameTetris structure containing information about the current model.

Returns

int The Y-coordinate of the current model's position.

8.53.2.10 GET_CURRENT_ROWS

```
#define GET_CURRENT_ROWS(
```

```
            game ) game.current.rows
```

Retrieves the number of rows in the current model.

Parameters

game	The GameTetris structure containing information about the current model.

Returns

size_t The number of rows in the current model.

8.53.2.11 GET_NEXT_COLS

```
#define GET_NEXT_COLS(
            game ) game.next.cols
```

Retrieves the number of columns in the next model.

Parameters

game	The GameTetris structure containing information about the next model.

Returns

size_t The number of columns in the next model.

8.53.2.12 GET_NEXT_MID_X

```
#define GET_NEXT_MID_X(
            game ) game.next.center[X]
```

Retrieves the X-coordinate of the center of the next model.

Parameters

game	The GameTetris structure containing information about the next model.

Returns

int The X-coordinate of the center of the next model.

8.53.2.13 GET_NEXT_MID_Y

```
#define GET_NEXT_MID_Y(
            game ) game.next.center[Y]
```

Retrieves the Y-coordinate of the center of the next model.

Parameters

game	The GameTetris structure containing information about the next model.

Returns

int The Y-coordinate of the center of the next model.

8.53.2.14 GET_NEXT_POS_X

```
#define GET_NEXT_POS_X(
              game ) game.next.position[X]
```

Retrieves the X-coordinate of the next model's position.

Parameters

game	The GameTetris structure containing information about the next model.

Returns

int The X-coordinate of the next model's position.

8.53.2.15 GET_NEXT_POS_Y

```
#define GET_NEXT_POS_Y(
              game ) game.next.position[Y]
```

Retrieves the Y-coordinate of the next model's position.

Parameters

game	The GameTetris structure containing information about the next model.

Returns

int The Y-coordinate of the next model's position.

8.53.2.16 GET_NEXT_ROWS

```
#define GET_NEXT_ROWS(
              game ) game.next.rows
```

Retrieves the number of rows in the next model.

Parameters

game	The GameTetris structure containing information about the next model.

Returns

size_t The number of rows in the next model.

8.54 MacroPos.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00025 #define GET_CURRENT_POS_X(game) game.current.position[X]
00026
00034 #define GET_CURRENT_POS_Y(game) game.current.position[Y]
00035
00043 #define GET_CURRENT_MID_X(game) game.current.center[X]
00044
00052 #define GET_CURRENT_MID_Y(game) game.current.center[Y]
00053
00061 #define GET_CURRENT_COLS(game) game.current.cols
00062
00070 #define GET_CURRENT_ROWS(game) game.current.rows
00071
00079 #define GET_NEXT_POS_X(game) game.next.position[X]
00080
00088 #define GET_NEXT_POS_Y(game) game.next.position[Y]
00089
00097 #define GET_NEXT_MID_X(game) game.next.center[X]
00098
00106 #define GET_NEXT_MID_Y(game) game.next.center[Y]
00107
00115 #define GET_NEXT_COLS(game) game.next.cols
00116
00124 #define GET_NEXT_ROWS(game) game.next.rows
00125
00137 #define CELL_NEXT_MODEL(i, j, game) game.next.model_[i][j]
00138
00150 #define CELL_CURRENT_MODEL(i, j, game) game.current.model_[i][j]
00151
00163 #define CELL_FIELD(i, j, game) game.engine.field[i][j]
00164
00176 #define CELL_NEXT(i, j, game) game.engine.next[i][j]
00177
00178 #ifdef __cplusplus
00179 }
00180 #endif
```

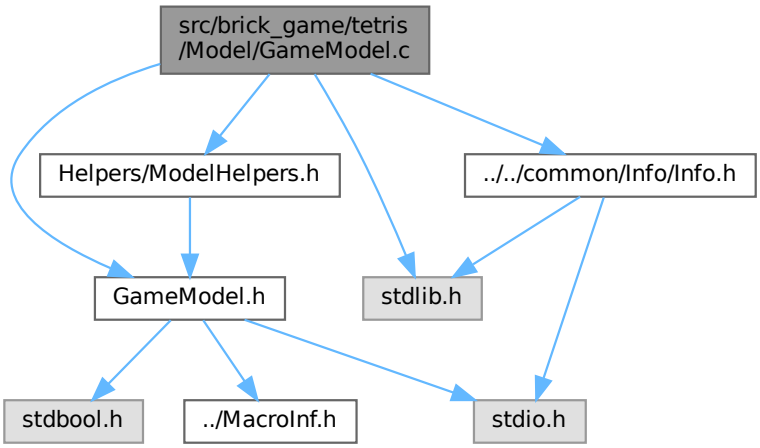
8.55 src/brick_game/tetris/Model/GameModel.c File Reference

A header file containing structure and function declarations for the game models.

```
#include "GameModel.h"
#include <stdlib.h>
#include "../../common/Info/Info.h"
```

```
#include "Helpers/ModelHelpers.h"
```

Include dependency graph for GameModel.c:



Functions

- int allocateModel (Model *model)

  Allocates memory for a game model.
- int allocateModels (Models *models)

  Allocates memory for the game models.
- void freeModel (Model *model)

  Frees memory allocated for a game model.
- void freeModels (Models *models)

  Frees memory allocated for multiple game models.
- void setRowsModel (Model *model, const size_t rows)

  Set the number of rows for a model.
- void setColsModel (Model *model, const size_t cols)

  Set the number of columns for a model.
- void setCountModels (Models *models, const size_t count)

  Set the count of models in a collection.
- size_t getRowsModel (const Model *model)

  Get the number of rows for a model.
- size_t getColsModel (const Model *model)

  Get the number of columns for a model.
- size_t getCountModels (const Models *models)

  Get the count of models in a collection.
- bool checkSizeModel (const Model *model)

  Check if the model size meets the minimum size requirement.
- bool isNormalModel (const Model *model)

  Check model.
- void copyModel (const Model *src, Model *dest)

  Copy the contents of one model to another.
- bool rotateModel (Model *model)

  Rotate the model clockwise by 90 degrees.

8.55.1 Detailed Description

A header file containing structure and function declarations for the game models.

Author

nenamaxi (an.veringe@gmail.com)

This file contains declarations of structures and functions used to work with game models.

Version

0.1

Date

2024-04-16

Copyright

Copyright (c) 2024

8.55.2 Function Documentation

8.55.2.1 allocateModel()

```
int allocateModel (
            Model * model )
```

Allocates memory for a game model.

This function allocates memory for the game model specified by the `Model` structure. It dynamically allocates memory for a 2D array representing the model's grid, based on the number of rows and columns specified in the `model` parameter. If successful, the function returns SUCCESS (true), otherwise, it returns an error code indicating the reason for failure.

Parameters

model	Pointer to the `Model` structure containing information about the model to allocate memory for.

Returns

int Returns SUCCESS (true) if memory allocation is successful, otherwise returns an error code.

8.55.2.2 allocateModels()

```
int allocateModels (
            Models * models )
```

Allocates memory for the game models.

This function allocates memory for the game models specified by the `Models` structure. It dynamically allocates memory for an array of `Model` structures based on the count provided in the `models` parameter. If successful, the function returns SUCCESS (true), otherwise, it returns FAIL (false).

Parameters

models	Pointer to the `Models` structure containing information about the models to allocate memory for.

Returns

int Returns SUCCESS (true) if memory allocation is successful, FAIL (false) otherwise.

8.55.2.3 checkSizeModel()

```
bool checkSizeModel (
            const Model * model )
```

Check if the model size meets the minimum size requirement.

This function checks if the size of the provided model meets the minimum size requirement.

Parameters

model	Pointer to the Model structure to check

Returns

true if the model size meets the minimum size requirement, false otherwise

8.55.2.4 copyModel()

```
void copyModel (
            const Model * src,
            Model * dest )
```

Copy the contents of one model to another.

This function copies the contents of one model to another model. It deallocates the memory of the destination model if it already contains data.

Parameters

src	The source model to copy from
dest	The destination model to copy to

8.55.2.5 freeModel()

```
void freeModel (
            Model * model )
```

Frees memory allocated for a game model.

This function frees the memory allocated for the game model specified by the Model structure. It releases the memory of the 2D array representing the model's grid and sets the relevant fields of the Model structure to NULL and 0. It is essential to call this function to avoid memory leaks after the model is no longer needed.

Parameters

model	Pointer to the Model structure containing the model to free memory for.

8.55.2.6 freeModels()

```
void freeModels (
            Models * models )
```

Frees memory allocated for multiple game models.

This function frees the memory allocated for an array of game models specified by the Models structure. It iterates through each model in the array, frees the memory of the 2D arrays representing their grids, and then frees the memory allocated for the array itself. Finally, it sets the relevant fields of the Models structure to NULL and 0. It is essential to call this function to avoid memory leaks after the models are no longer needed.

Parameters

models	Pointer to the Models structure containing the array of models to free memory for.

8.55.2.7 getColsModel()

```
size_t getColsModel (
            const Model * model )
```

Get the number of columns for a model.

Parameters

model	Pointer to the Model structure.

Returns

Number of columns.

8.55.2.8 getCountModels()

```
size_t getCountModels (
            const Models * models )
```

Get the count of models in a collection.

Parameters

models	Pointer to the Models structure.

Returns

Count of models.

8.55.2.9 getRowsModel()

```
size_t getRowsModel (
            const Model * model )
```

Get the number of rows for a model.

Parameters

model	Pointer to the Model structure.

Returns

Number of rows.

8.55.2.10 isNormalModel()

```
bool isNormalModel (
            const Model * model )
```

Check model.

Parameters

model	Pointer to the Model structure

Returns

true if model is correct

false if model is incorrect

8.55.2.11 rotateModel()

```
bool rotateModel (
            Model * model )
```

Rotate the model clockwise by 90 degrees.

This function rotates the given model clockwise by 90 degrees. It allocates memory for the rotated model, copies the rotated values from the original model, and updates the model's dimensions and center accordingly.

Parameters

model	Pointer to the model structure to be rotated.

Returns

true If the rotation operation succeeds.

false If the rotation operation fails, typically due to invalid input or memory allocation issues.

8.55.2.12 setColsModel()

```
void setColsModel (
            Model * model,
            const size_t cols )
```

Set the number of columns for a model.

Parameters

model	Pointer to the Model structure.
cols	Number of columns to set.

8.55.2.13 setCountModels()

```
void setCountModels (
            Models * models,
            const size_t count )
```

Set the count of models in a collection.

Parameters

models	Pointer to the Models structure.
count	Number of models to set.

8.55.2.14 setRowsModel()

```
void setRowsModel (
            Model * model,
            const size_t rows )
```

Set the number of rows for a model.

Parameters

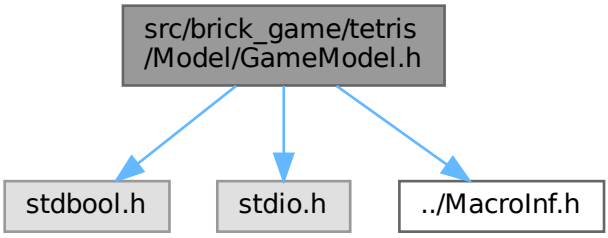
model	Pointer to the Model structure.
rows	Number of rows to set.

8.56 src/brick_game/tetris/Model/GameModel.h File Reference

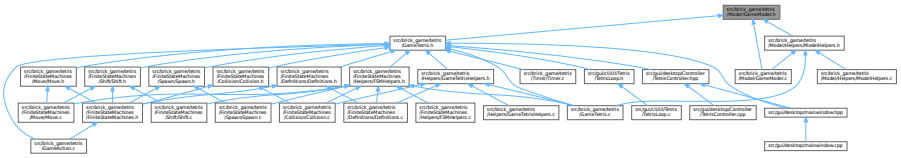
A header file containing structure and function declarations for the game models.

```
#include <stdbool.h>
#include <stdio.h>
#include "../MacroInf.h"
```

Include dependency graph for GameModel.h:



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



Classes

- struct Model

  Structure representing a game model.
- struct Models

  Structure representing a collection of game models.

Enumerations

- enum COORD { X = 0 , Y = 1 }

  Enumeration of coordinate directions.

Functions

- int allocateModel (Model *model)
  Allocates memory for a game model.
- int allocateModels (Models *models)
  Allocates memory for the game models.
- void freeModel (Model *model)
  Frees memory allocated for a game model.
- void freeModels (Models *models)
  Frees memory allocated for multiple game models.
- void setRowsModel (Model *model, const size_t rows)
  Set the number of rows for a model.
- void setColsModel (Model *model, const size_t cols)
  Set the number of columns for a model.
- void setCountModels (Models *models, const size_t count)
  Set the count of models in a collection.
- size_t getRowsModel (const Model *model)
  Get the number of rows for a model.
- size_t getColsModel (const Model *model)
  Get the number of columns for a model.
- size_t getCountModels (const Models *models)
  Get the count of models in a collection.
- bool checkSizeModel (const Model *model)
  Check if the model size meets the minimum size requirement.
- bool isNormalModel (const Model *model)
  Check model.
- void copyModel (const Model *src, Model *dest)
  Copy the contents of one model to another.
- bool rotateModel (Model *model)
  Rotate the model clockwise by 90 degrees.

8.56.1 Detailed Description

A header file containing structure and function declarations for the game models.

Author

nenamaxi (an.veringe@gmail.com)

This file contains declarations of structures and functions used to work with game models.

Version

0.1

Date

2024-04-16

Copyright

8.56.2 Enumeration Type Documentation

8.56.2.1 COORD

```
enum COORD
```

Enumeration of coordinate directions.

An enumeration defining the possible coordinate directions. X corresponds to the horizontal direction and Y corresponds to the vertical direction.

Enumerator

X	Horizontal direction.
Y	Vertical direction.

8.56.3 Function Documentation

8.56.3.1 allocateModel()

```
int allocateModel (
            Model * model )
```

Allocates memory for a game model.

This function allocates memory for the game model specified by the `Model` structure. It dynamically allocates memory for a 2D array representing the model's grid, based on the number of rows and columns specified in the `model` parameter. If successful, the function returns SUCCESS (true), otherwise, it returns an error code indicating the reason for failure.

Parameters

model	Pointer to the `Model` structure containing information about the model to allocate memory for.

Returns

int Returns SUCCESS (true) if memory allocation is successful, otherwise returns an error code.

8.56.3.2 allocateModels()

```
int allocateModels (
            Models * models )
```

Allocates memory for the game models.

This function allocates memory for the game models specified by the `Models` structure. It dynamically allocates memory for an array of `Model` structures based on the count provided in the `models` parameter. If successful, the function returns SUCCESS (true), otherwise, it returns FAIL (false).

Parameters

models	Pointer to the `Models` structure containing information about the models to allocate memory for.

Returns

int Returns SUCCESS (true) if memory allocation is successful, FAIL (false) otherwise.

8.56.3.3 checkSizeModel()

```
bool checkSizeModel (
            const Model * model )
```

Check if the model size meets the minimum size requirement.

This function checks if the size of the provided model meets the minimum size requirement.

Parameters

model	Pointer to the Model structure to check

Returns

true if the model size meets the minimum size requirement, false otherwise

8.56.3.4 copyModel()

```
void copyModel (
            const Model * src,
            Model * dest )
```

Copy the contents of one model to another.

This function copies the contents of one model to another model. It deallocates the memory of the destination model if it already contains data.

Parameters

src	The source model to copy from
dest	The destination model to copy to

8.56.3.5 freeModel()

```
void freeModel (
            Model * model )
```

Frees memory allocated for a game model.

This function frees the memory allocated for the game model specified by the `Model` structure. It releases the memory of the 2D array representing the model's grid and sets the relevant fields of the `Model` structure to NULL and 0. It is essential to call this function to avoid memory leaks after the model is no longer needed.

Parameters

model	Pointer to the `Model` structure containing the model to free memory for.

8.56.3.6 freeModels()

```
void freeModels (
            Models * models )
```

Frees memory allocated for multiple game models.

This function frees the memory allocated for an array of game models specified by the `Models` structure. It iterates through each model in the array, frees the memory of the 2D arrays representing their grids, and then frees the memory allocated for the array itself. Finally, it sets the relevant fields of the `Models` structure to NULL and 0. It is essential to call this function to avoid memory leaks after the models are no longer needed.

Parameters

models	Pointer to the `Models` structure containing the array of models to free memory for.

8.56.3.7 getColsModel()

```
size_t getColsModel (
            const Model * model )
```

Get the number of columns for a model.

Parameters

model	Pointer to the Model structure.

Returns

Number of columns.

8.56.3.8 getCountModels()

```
size_t getCountModels (
            const Models * models )
```

Get the count of models in a collection.

Parameters

models	Pointer to the Models structure.

Returns

Count of models.

8.56.3.9 getRowsModel()

```
size_t getRowsModel (
            const Model * model )
```

Get the number of rows for a model.

Parameters

model	Pointer to the Model structure.

Returns

Number of rows.

8.56.3.10 isNormalModel()

```
bool isNormalModel (
            const Model * model )
```

Check model.

Parameters

model	Pointer to the Model structure

Returns

true if model is correct

false if model is incorrect

8.56.3.11 rotateModel()

```
bool rotateModel (
            Model * model )
```

Rotate the model clockwise by 90 degrees.

This function rotates the given model clockwise by 90 degrees. It allocates memory for the rotated model, copies the rotated values from the original model, and updates the model's dimensions and center accordingly.

Parameters

model	Pointer to the model structure to be rotated.

Returns

true If the rotation operation succeeds.

false If the rotation operation fails, typically due to invalid input or memory allocation issues.

8.56.3.12 setColsModel()

```
void setColsModel (
            Model * model,
            const size_t cols )
```

Set the number of columns for a model.

Parameters

model	Pointer to the Model structure.
cols	Number of columns to set.

8.56.3.13 setCountModels()

```
void setCountModels (
            Models * models,
            const size_t count )
```

Set the count of models in a collection.

Parameters

models	Pointer to the Models structure.
count	Number of models to set.

8.56.3.14 setRowsModel()

```
void setRowsModel (
            Model * model,
            const size_t rows )
```

Set the number of rows for a model.

Parameters

model	Pointer to the Model structure.
rows	Number of rows to set.

8.57 GameModel.h

Go to the documentation of this file.

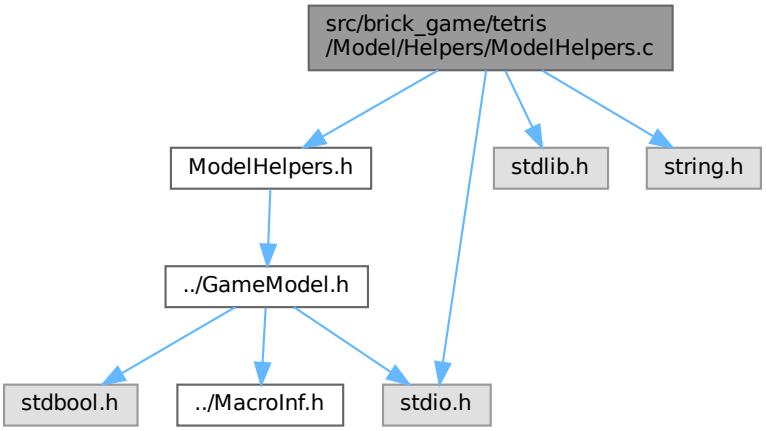
```
00001
00017 #pragma once
00018
00019 #ifdef __cplusplus
00020 extern "C" {
00021 #endif
00022
00023 #include <stdbool.h>
00024 #include <stdio.h>
00025
00026 #include "../MacroInf.h"
00027
00035 typedef enum {
00036   X = 0,
00037   Y = 1,
00038 } COORD;
00039
00048 typedef struct {
00049   size_t rows;
00050   size_t cols;
00051   int **model_;
00052   int position[SIZE_COORD];
00053   int center[SIZE_COORD];
00055 } Model;
00056
00064 typedef struct {
00065   Model *models;
00067   size_t count;
00068 } Models;
00069
00070 int allocateModel(Model *model);
00071 int allocateModels(Models *models);
00072
00073 void freeModel(Model *model);
00074 void freeModels(Models *models);
00075
00076 void setRowsModel(Model *model, const size_t rows);
00077 void setColsModel(Model *model, const size_t cols);
00078 void setCountModels(Models *models, const size_t count);
00079
00080 size_t getRowsModel(const Model *model);
00081 size_t getColsModel(const Model *model);
00082 size_t getCountModels(const Models *models);
00083
00084 bool checkSizeModel(const Model *model);
00085 bool isNormalModel(const Model *model);
00086 void copyModel(const Model *src, Model *dest);
00087
00088 bool rotateModel(Model *model);
00089
00090 #ifdef __cplusplus
00091 }
00092 #endif
```

8.58 src/brick_game/tetris/Model/Helpers/ModelHelpers.c File Reference

A file with constants for the parser and prototypes for model initialization.

```
#include "ModelHelpers.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for ModelHelpers.c:



Functions

- void obtainingModels (Models *models, int *code, const char *filename)

  Obtain models from a file template.
- int initializeModels (Models *models, const char *filename)

  Initialize models from a file template.

8.58.1 Detailed Description

A file with constants for the parser and prototypes for model initialization.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-13

Copyright

Copyright (c) 2024

8.58.2 Function Documentation

8.58.2.1 initializeModels()

```
int initializeModels (
            Models * models,
            const char * filename )
```

Initialize models from a file template.

This function initializes models by obtaining them from a file template. It calls the obtainingModels function to parse the file template and populate the provided Models structure with the parsed models. If the number of models obtained is less than a specified threshold, it sets the return code to indicate failure.

Parameters

models	Pointer to the Models structure to store the parsed models
filename	filename BD

Returns

int Return code indicating the success or failure of the initialization operation

8.58.2.2 obtainingModels()

```
void obtainingModels (
            Models * models,
            int * code,
            const char * filename )
```

Obtain models from a file template.

This function reads models from a file template, parses the content, and populates the provided Models structure with the parsed models. It sets the return code based on the success or failure of the parsing operation.

Parameters

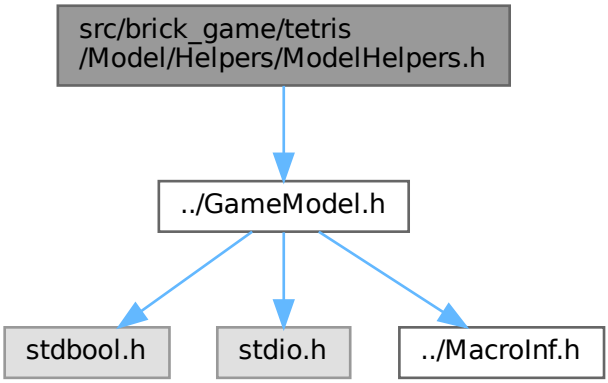
models	Pointer to the Models structure to store the parsed models
code	Pointer to an integer variable to store the return code
filename	filename template

8.59 src/brick_game/tetris/Model/Helpers/ModelHelpers.h File Reference

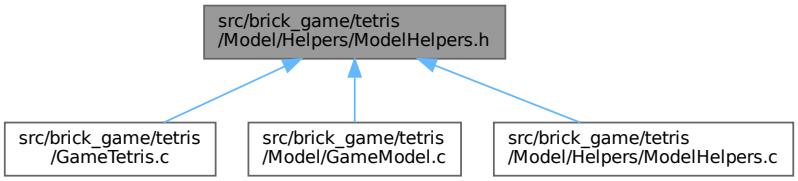
A file with constants for the parser and prototypes for model initialization.

```
#include "../GameModel.h"
```

Include dependency graph for ModelHelpers.h:



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



Macros

- #define SIGN_WIDTH 'w'
- #define SIGN_HEIGHT 'h'
- #define SIGN_TRUE_CELL '1'
- #define LITERAL_INFO_FIGURE "t: h-%lu w-%lu"
- #define **LITERAL_TEMPLATE** 't'

  Symbol indicating the beginning of a new model.
- #define MIN_SIZE 1

Functions

- int initializeModels (Models *models, const char *filename)

  Initialize models from a file template.
- void obtainingModels (Models *models, int *code, const char *filename)

  Obtain models from a file template.

8.59.1 Detailed Description

A file with constants for the parser and prototypes for model initialization.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-13

Copyright

8.59.2 Macro Definition Documentation

8.59.2.1 LITERAL_INFO_FIGURE

```
#define LITERAL_INFO_FIGURE  "t:  h-%lu w-%lu"
```

The literal string format for figure information.

8.59.2.2 MIN_SIZE

```
#define MIN_SIZE 1
```

Minimal correct size.

8.59.2.3 SIGN_HEIGHT

```
#define SIGN_HEIGHT 'h'
```

Symbol for the `height` of the model.

8.59.2.4 SIGN_TRUE_CELL

```
#define SIGN_TRUE_CELL '1'
```

Symbol for the `height` of the model.

8.59.2.5 SIGN_WIDTH

```
#define SIGN_WIDTH 'w'
```

Symbol for the `width` of the model.

8.59.3 Function Documentation

8.59.3.1 initializeModels()

```
int initializeModels (
            Models * models,
            const char * filename )
```

Initialize models from a file template.

This function initializes models by obtaining them from a file template. It calls the obtainingModels function to parse the file template and populate the provided Models structure with the parsed models. If the number of models obtained is less than a specified threshold, it sets the return code to indicate failure.

Parameters

models	Pointer to the Models structure to store the parsed models
filename	filename BD

Returns

int Return code indicating the success or failure of the initialization operation

8.59.3.2 obtainingModels()

```
void obtainingModels (
            Models * models,
            int * code,
            const char * filename )
```

Obtain models from a file template.

This function reads models from a file template, parses the content, and populates the provided Models structure with the parsed models. It sets the return code based on the success or failure of the parsing operation.

Parameters

models	Pointer to the Models structure to store the parsed models
code	Pointer to an integer variable to store the return code
filename	filename template

8.60 ModelHelpers.h

Go to the documentation of this file.

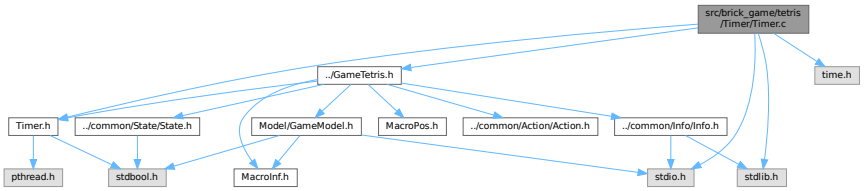
```
00001
00013 #pragma once
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 #include "../GameModel.h"
00020
00021 #define SIGN_WIDTH 'w'
00022 #define SIGN_HEIGHT 'h'
00023 #define SIGN_TRUE_CELL '1'
00024 #define LITERAL_INFO_FIGURE \
00025   "t: h-%lu w-%lu"
00030 #define LITERAL_TEMPLATE 't'
00031
00032 #define MIN_SIZE 1
00034 int initializeModels(Models *models, const char *filename);
00035 void obtainingModels(Models *models, int *code, const char *filename);
00036
00037 #ifdef __cplusplus
00038 }
00039 #endif
```

8.61 src/brick_game/tetris/Timer/Timer.c File Reference

The file that describes the thread for the timer.

```
#include "Timer.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "../GameTetris.h"
```

Include dependency graph for Timer.c:



Functions

- void stopTime (GameTimer_t *timer)

  Stops the timer thread.
- void runTime (GameTimer_t *timer)

  Runs the timer if the game is not paused and the state is MOVE.
- bool initializeTimer (GameTimer_t *timer)

  Initializes the game timer.
- void freeTimer (GameTimer_t *timer)

  Frees resources associated with the game timer.

8.61.1 Detailed Description

The file that describes the thread for the timer.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-24

Copyright

Copyright (c) 2024

8.61.2 Function Documentation

8.61.2.1 freeTimer()

```
void freeTimer (
            GameTimer_t * timer )
```

Frees resources associated with the game timer.

This function frees the memory allocated for the timer thread and releases any resources associated with the game timer.

Parameters

timer	A pointer to a GameTimer_t structure whose resources are to be freed.

8.61.2.2 initializeTimer()

```
bool initializeTimer (
            GameTimer_t * timer )
```

Initializes the game timer.

This function initializes the game timer by allocating memory for the timer thread and setting the indicator to false.

Parameters

timer	A pointer to a GameTimer_t structure to be initialized.

Returns

true if the initialization is successful, false otherwise.

8.61.2.3 runTime()

```
void runTime (
            GameTimer_t * timer )
```

Runs the timer if the game is not paused and the state is MOVE.

This function runs the timer if the game is not paused and the state is MOVE. It stops the existing timer thread and creates a new one to handle timer operations.

Parameters

timer	A pointer to a GameTimer_t structure containing timer information.

8.61.2.4 stopTime()

```
void stopTime (
            GameTimer_t * timer )
```

Stops the timer thread.

This function stops the timer thread associated with the given timer.

Parameters

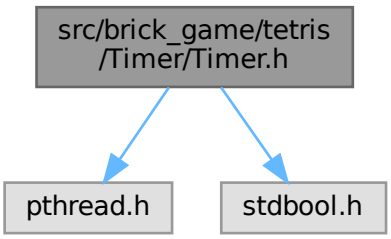
timer	A pointer to a GameTimer_t structure containing timer information.

8.62 src/brick_game/tetris/Timer/Timer.h File Reference

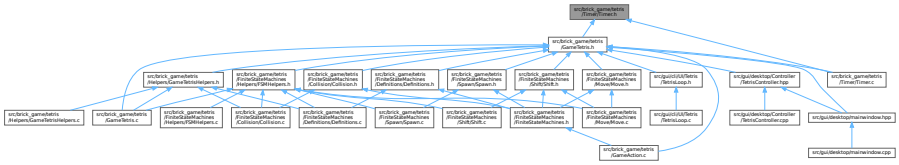
The file that describes the thread for the timer.

```
#include <pthread.h>
#include <stdbool.h>
```

Include dependency graph for Timer.h:



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



Classes

- struct GameTimer_t

  Structure representing a game timer.

Functions

- bool initializeTimer (GameTimer_t *timer)

  Initializes the game timer.
- void freeTimer (GameTimer_t *timer)

  Frees resources associated with the game timer.
- void stopTime (GameTimer_t *timer)

  Stops the timer thread.
- void runTime (GameTimer_t *timer)

  Runs the timer if the game is not paused and the state is MOVE.

8.62.1 Detailed Description

The file that describes the thread for the timer.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-24

Copyright

8.62.2 Function Documentation

8.62.2.1 freeTimer()

```
void freeTimer (
            GameTimer_t * timer )
```

Frees resources associated with the game timer.

This function frees the memory allocated for the timer thread and releases any resources associated with the game timer.

Parameters

timer	A pointer to a GameTimer_t structure whose resources are to be freed.

8.62.2.2 initializeTimer()

```
bool initializeTimer (
            GameTimer_t * timer )
```

Initializes the game timer.

This function initializes the game timer by allocating memory for the timer thread and setting the indicator to false.

Parameters

timer	A pointer to a GameTimer_t structure to be initialized.

Returns

true if the initialization is successful, false otherwise.

8.62.2.3 runTime()

```
void runTime (
            GameTimer_t * timer )
```

Runs the timer if the game is not paused and the state is MOVE.

This function runs the timer if the game is not paused and the state is MOVE. It stops the existing timer thread and creates a new one to handle timer operations.

Parameters

timer	A pointer to a GameTimer_t structure containing timer information.

8.62.2.4 stopTime()

```
void stopTime (
            GameTimer_t * timer )
```

Stops the timer thread.

This function stops the timer thread associated with the given timer.

Parameters

timer	A pointer to a GameTimer_t structure containing timer information.

8.63 Timer.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include <pthread.h>
00019 #include <stdbool.h>
00020
00021 // #include "../type/GameType.h"
00022
00026 typedef struct {
00027   bool indicator;
00028   pthread_t *thread;
00029 } GameTimer_t;
00030
00031 bool initializeTimer(GameTimer_t *timer);
00032 void freeTimer(GameTimer_t *timer);
00033 void stopTime(GameTimer_t *timer);
00034 void runTime(GameTimer_t *timer);
00035
00036 #ifdef __cplusplus
00037 }
00038 #endif
```

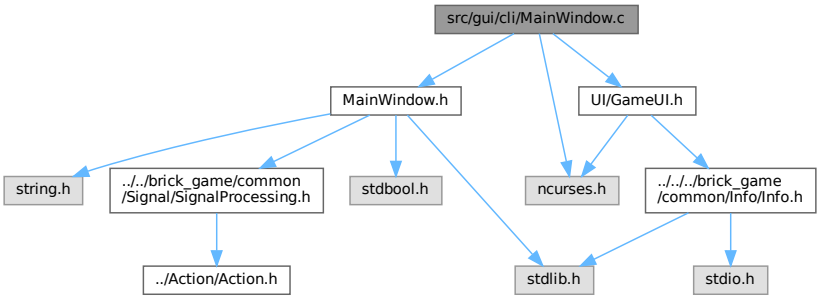
8.64 src/gui/cli/MainWindow.c File Reference

Source file Mainwindow.

```
#include "MainWindow.h"
#include <ncurses.h>
```

```
#include "UI/GameUI.h"
```

Include dependency graph for MainWindow.c:



Functions

- bool launch (Mainwindow *menu)

  Launches a game menu.
- bool initMainWindow (Mainwindow *mw)

  Initializes the main window structure.

8.64.1 Detailed Description

Source file Mainwindow.

Author

nenamaxi(an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

8.64.2 Function Documentation

8.64.2.1 initMainWindow()

```
bool initMainWindow (
            Mainwindow * mw )
```

Initializes the main window structure.

Parameters

mw	The main window structure to be initialized.

Returns

Whether the initialization was successful.

8.64.2.2 launch()

```
bool launch (
             Mainwindow * menu )
```

Launches a game menu.

Parameters

menu	The main window structure containing information about the menu.

Returns

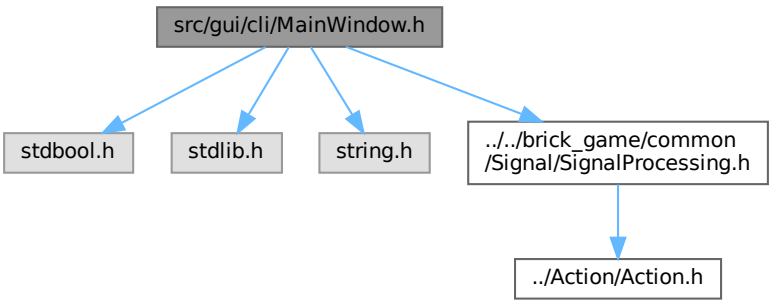
Whether the game was launched successfully.

8.65 src/gui/cli/MainWindow.h File Reference

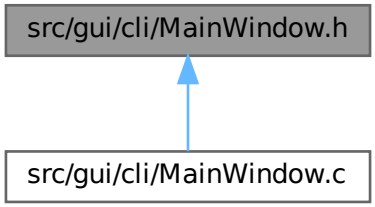
Header file Mainwindow.

```
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include "../../brick_game/common/Signal/SignalProcessing.h"
```

Include dependency graph for MainWindow.h:



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



Classes

- struct Mainwindow

  A structure holding information about a main window.

Macros

- #define MENU_HEIGHT 20
- #define MENU_WIDTH 40
- #define BUFF_STR 256
- #define BASE_COUNT 10
- #define INDEX_FIRST_GAME 0
- #define INDEX_SECOND_GAME 1
- #define COUNT_GAME 2
- #define COUNT_INFO 4

Functions

- bool launch (Mainwindow *menu)

  Launches a game menu.
- bool initMainWindow (Mainwindow *mw)

  Initializes the main window structure.

8.65.1 Detailed Description

Header file Mainwindow.

Author

nenamaxi(an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

8.65.2 Macro Definition Documentation

8.65.2.1 BASE_COUNT

```
#define BASE_COUNT 10
```

Base count

8.65.2.2 BUFF_STR

```
#define BUFF_STR 256
```

Size string

8.65.2.3 COUNT_GAME

```
#define COUNT_GAME 2
```

Number of games

8.65.2.4 COUNT_INFO

```
#define COUNT_INFO 4
```

Number of info

8.65.2.5 INDEX_FIRST_GAME

```
#define INDEX_FIRST_GAME 0
```

Index of the first game

8.65.2.6 INDEX_SECOND_GAME

```
#define INDEX_SECOND_GAME 1
```

Index of the second game

8.65.2.7 MENU_HEIGHT

```
#define MENU_HEIGHT 20
```

Menu Height

8.65.2.8 MENU_WIDTH

```
#define MENU_WIDTH 40
```

Menu WIDTH

8.65.3 Function Documentation

8.65.3.1 initMainWindow()

```
bool initMainWindow (
            Mainwindow * mw )
```

Initializes the main window structure.

Parameters

mw	The main window structure to be initialized.

Returns

Whether the initialization was successful.

8.65.3.2 launch()

```
bool launch (
             Mainwindow * menu )
```

Launches a game menu.

Parameters

menu	The main window structure containing information about the menu.

Returns

Whether the game was launched successfully.

8.66 MainWindow.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include <stdbool.h>
00019 #include <stdlib.h>
00020 #include <string.h>
00021
00022 #include "../../brick_game/common/Signal/SignalProcessing.h"
00023
00024 #define MENU_HEIGHT 20
00025 #define MENU_WIDTH 40
00027 #define BUFF_STR 256
00028 #define BASE_COUNT 10
00030 #define INDEX_FIRST_GAME 0
00031 #define INDEX_SECOND_GAME 1
00032 #define COUNT_GAME 2
00033 #define COUNT_INFO 4
00038 typedef struct {
00039   char title[BUFF_STR];
00041   char description[BUFF_STR];
00044   size_t sgame;
00045   size_t cgame;
00046   char games[BASE_COUNT]
00047             [BUFF_STR];
00049   size_t sinfo;
00050   size_t cinfo;
00051   char info[BASE_COUNT]
00052            [BUFF_STR];
00054   bool vertical;
00056   bool
00057       centered;
00058 } Mainwindow;
00059
```

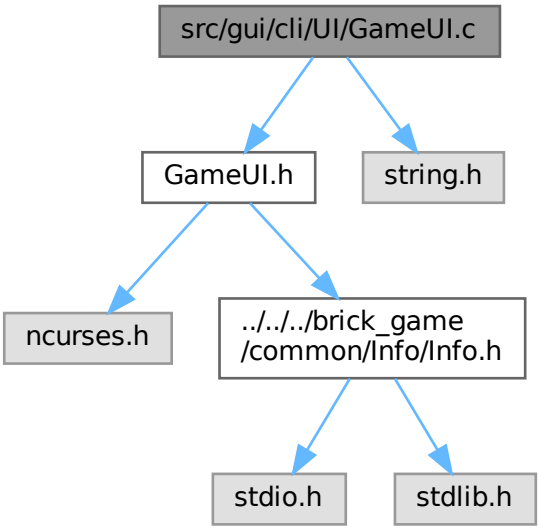
```
00060 bool launch(Mainwindow *menu);
00061 bool initMainWindow(Mainwindow *mw);
00062
00063 #ifdef __cplusplus
00064 }
00065 #endif
```

8.67 src/gui/cli/UI/GameUI.c File Reference

A file that describes the interface.

```
#include "GameUI.h"
#include <string.h>
```

Include dependency graph for GameUI.c:



Functions

- void drawUIBoard (WINDOW *win, const size_t x, const size_t y, const size_t width, const size_t height)

  Draw a board in the specified window.
- void initColor ()

  Initialize color pairs for curses.
- void initCli ()

  Initialize the command-line interface (CLI).
- void delCli ()

  Delete the command-line interface (CLI).
- bool initWindow (WINDOW **fieldW, WINDOW **infoW)

  Initialize windows.
- void deleteWindow (WINDOW **win)

  Delete a window.

- void printInfoWindow (int win, const char *gameName, const GameInfo_t gameInfo)

  Print info.
- void refreshUIWin (const char *gameName, GameInfo_t gameInfo, GameWindows *gameWin)

  Refresh the user interface windows.
- bool initGameWindow (GameWindows *gameWin)

  Initialize the game windows.
- void freeGameWindow (GameWindows *gameWin)

  Free the memory allocated for game windows.
- void update (const char *gameName, GameWindows *gameWin, GameInfo_t gameInfo, const char *addition)

  Update function for the game state.

8.67.1 Detailed Description

A file that describes the interface.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.67.2 Function Documentation

8.67.2.1 delCli()

```
void delCli ( )
```

Delete the command-line interface (CLI).

This function clears the screen, refreshes it, and closes the curses library.

8.67.2.2 deleteWindow()

```
void deleteWindow (
            WINDOW ** win )
```

Delete a window.

This function deletes the specified window.

Parameters

win	Pointer to the window to delete.

8.67.2.3 drawUIBoard()

```
void drawUIBoard (
            WINDOW * win,
            const size_t x,
            const size_t y,
            const size_t width,
            const size_t height )
```

Draw a board in the specified window.

This function draws a rectangular board.

Parameters

win	The window in which the board will be drawn.
x	The initial x-coordinate of the board.
y	The initial y-coordinate of the board.
width	The width of the board.
height	The height of the board.

8.67.2.4 freeGameWindow()

```
void freeGameWindow (
            GameWindows * gameWin )
```

Free the memory allocated for game windows.

This function frees the memory allocated for the game windows.

Parameters

gameWin	Pointer to the game windows structure.

8.67.2.5 initCli()

```
void initCli ( )
```

Initialize the command-line interface (CLI).

This function initializes the command-line interface (CLI) using the curses library. It sets up the necessary configurations for curses, such as colors, cursor visibility, echoing, and keypad input.

8.67.2.6 initColor()

```
void initColor ( )
```

Initialize color pairs for curses.

This function initializes color pairs for curses. It enables the use of colors in the terminal window.

8.67.2.7 initGameWindow()

```
bool initGameWindow (
            GameWindows * gameWin )
```

Initialize the game windows.

This function initializes the game windows and sets the infoDraw flag to false.

Parameters

gameWin	Pointer to the game windows structure.

Returns

true if initialization is successful, false otherwise.

8.67.2.8 initWindow()

```
bool initWindow (
            WINDOW ** fieldW,
            WINDOW ** infoW )
```

Initialize windows.

This function creates and initializes the field window and the information window.

Parameters

fieldW	Pointer to a pointer to the field window.
infoW	Pointer to a pointer to the information window.

8.67.2.9 printInfoWindow()

```
void printInfoWindow (
            int win,
            const char * gameName,
            const GameInfo_t gameInfo )
```

Print info.

This function prints game information.

Parameters

win	The window that will be drawn.
gameName	Name game.
gameInfo	Pointer to the GameInfo_t structure containing information about the game.

8.67.2.10 refreshUIWin()

```
void refreshUIWin (
            const char * gameName,
            GameInfo_t gameInfo,
            GameWindows * gameWin )
```

Refresh the user interface windows.

This function refreshes the game field window and the game information window.

Parameters

gameName	Name game.
gameInfo	The current game information.
gameWin	The struct window.

8.67.2.11 update()

```
void update (
            const char * gameName,
            GameWindows * gameWin,
            GameInfo_t gameInfo,
            const char * addition )
```

Update function for the game state.

This function updates the game state and refreshes the user interface window accordingly.

Parameters

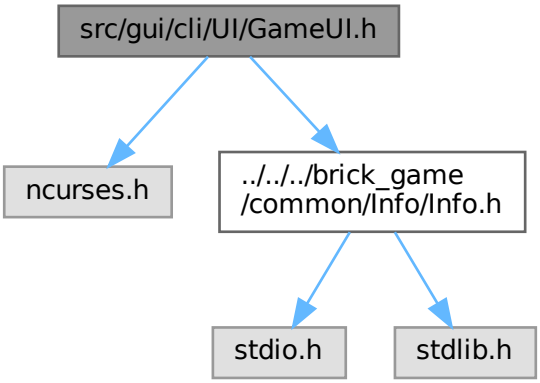
gameName	Game name.
gameWin	Pointer to the game windows.
gameInfo	The game information.
addition	The game addition information.

8.68 src/gui/cli/UI/GameUI.h File Reference

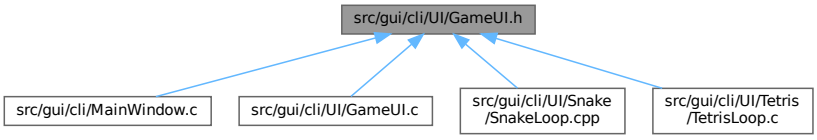
A file that describes the interface.

```
#include <ncurses.h>
#include "../../../brick_game/common/Info/Info.h"
```

Include dependency graph for GameUI.h:



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



Classes

- struct SizeText

  Structure for displaying terminal information.
- struct GameWindows

  Structure representing game windows.

Macros

- #define MAX_CELLS 200
- #define WINDOW_FIELD_HEIGHT HFIELD + 2
- #define HEIGHT_BOARD_NEXT 6
- #define HEIGHT_WIN_PAUSE 3
- #define HEIGHT_WIN_START 9
- #define HEIGHT_WIN_EXIT 9
- #define WINDOW_FIELD_WIDTH WFIELD ∗ 2 + 2
- #define WIDTH_BOARD_NEXT 15
- #define WIDTH_WIN_PAUSE 40

- #define WIDTH_WIN_START 45
- #define WIDTH_WIN_EXIT 40
- #define WIN_PAUSE 0
- #define WIN_START 1
- #define WIN_EXIT 2
- #define WIN_FIELD 3
- #define SIZE_BOX 1
- #define DRAW_POS_Y1 1
- #define DRAW_POS_Y2 3
- #define DRAW_POS_Y3 5
- #define DRAW_POS_Y4 7
- #define START_X_BOAR 3
- #define START_Y_BOAR 3
- #define GET_POS_X1(x) x ∗ 2 + 1

  Macro to return the first position of the square on the window.
- #define GET_POS_X2(x) GET_POS_X1(x) + 1

  Macro to return the second position of the square in the window.
- #define GETMAXWH(height, width) getmaxyx(stdscr, height, width);

  Macro to get terminal dimensions.
- #define SET_COLOR_PAIR(win, index) wbkgdset(win, COLOR_PAIR(index));

  Macro for setting window color pair.
- #define CHECK_WIN(win) (win != NULL)
- #define CHECK_FIELD(field) (field != NULL)
- #define DRAW_BOX(win) box(win, 0, 0);

  Macro for drawing a box.
- #define COLOR_BOX(win, index) wbkgd(win, COLOR_PAIR(index));

  Macro for setting box color pair.
- #define COLOR_TEXT(win, index) wattron(win, COLOR_PAIR(index));

  Macro for setting text color pair.
- #define DRAW_CELL(x, y, win)

  Macro for inserting a square.
- #define GET_INFO_PRINT(index)

  The macro returns the text according to the index, indices from the `INDEX_TEXT_INFO` enumeration.

Enumerations

- enum INDEX_TEXT_INFO {
  ITNAME = 0 , ISNAME = 1 , IHIGH_SCORE = 2 , ISCORE = 3 ,
  ILEVEL = 4 , INEXT = 5 , IPAUSE = 6 , INAMEBG = 7 ,
  IPRESSENTER = 8 , IPRESSESC = 9 , ILOSE = 10 , IWIN = 11 ,
  IEXITORREST = 12 , ISTART = 13 , IEXIT = 14 , ISELECT = 15 ,
  ISSTART = 16 , ISEND = 17 , ISNOT = 18 }

  Enumeration to get the text to display information in the window.
- enum Color {
  BLACK_MAGENTA = 1 , WHITE_BLACK = 2 , BLACK_CYAN = 3 , BLACK_RED = 4 ,
  BLACK_WHITE = 5 , BLACK_GREEN = 6 , RED_BLACK = 7 , BLACK_YELLOW = 8 ,
  BLACK_BLUE = 9 , GREEN_BLACK = 10 }

  Enumeration of color indices.

Functions

- void initCli ()

  Initialize the command-line interface (CLI).
- void initColor ()

  Initialize color pairs for curses.
- bool initWindow (WINDOW **fieldW, WINDOW **infoW)

  Initialize windows.
- bool initGameWindow (GameWindows *gameWin)

  Initialize the game windows.
- void delCli ()

  Delete the command-line interface (CLI).
- void deleteWindow (WINDOW **win)

  Delete a window.
- void freeGameWindow (GameWindows *gameWin)

  Free the memory allocated for game windows.
- void printInfoWindow (int win, const char *gameName, const GameInfo_t gameInfo)

  Print info.
- void refreshUIWin (const char *gameName, GameInfo_t gameInfo, GameWindows *gameWin)

  Refresh the user interface windows.
- void update (const char *gameName, GameWindows *gameWin, GameInfo_t gameInfo, const char *addition)

  Update function for the game state.

8.68.1 Detailed Description

A file that describes the interface.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-04-15

Copyright

Copyright (c) 2024

8.68.2 Enumeration Type Documentation

8.68.2.1 Color

```
enum Color
```

Enumeration of color indices.

Enumerator

BLACK_MAGENTA	`Text` - Black, `Bakcground` - Magenta
WHITE_BLACK	`Text` - White, `Bakcground` - Black
BLACK_CYAN	`Text` - Black, `Bakcground` - Cyan
BLACK_RED	`Text` - Black, `Bakcground` - Red
BLACK_WHITE	`Text` - Black, `Bakcground` - White
BLACK_GREEN	`Text` - Black, `Bakcground` - Green
RED_BLACK	`Text` - Red, `Bakcground` - Black
BLACK_YELLOW	`Text` - Black, `Bakcground` - Yellow
BLACK_BLUE	`Text` - Black, `Bakcground` - Blue
GREEN_BLACK	`Text` - Green, `Bakcground` - Black

8.68.2.2 INDEX_TEXT_INFO

```
enum INDEX_TEXT_INFO
```

Enumeration to get the text to display information in the window.

Enumerator

ITNAME	Game name - `TETRIS`
ISNAME	Game name - `SNAKE`
IHIGH_SCORE	High Score - `HIGH SCORE:`
ISCORE	Score - `SCORE:`
ILEVEL	Level - `LEVEL:`
INEXT	Next - `NEXT`
IPAUSE	Pause - `PAUSE`
INAMEBG	Brick Game text - `Brick Games`
IPRESSENTER	Press enter - `START Game press <Enter>`
IPRESSESC	Press esc - `EXIT Game press <ESC>`
ILOSE	Lose info - `You lose`
IWIN	Win info - `You win`
IEXITORREST	ent or esc - `Press ESC to exit or ENTER to restart`
ISTART	String - `START`
IEXIT	String - `EXIT`
ISELECT	String - `SELECT THE GAME AND THEN CLICK START`
ISSTART	String - `start`
ISEND	String - `end`
ISNOT	String - `not`

8.68.3 Function Documentation

8.68.3.1 delCli()

```
void delCli ( )
```

Delete the command-line interface (CLI).

This function clears the screen, refreshes it, and closes the curses library.

8.68.3.2 deleteWindow()

```
void deleteWindow (
            WINDOW ** win )
```

Delete a window.

This function deletes the specified window.

Parameters

win	Pointer to the window to delete.

8.68.3.3 freeGameWindow()

```
void freeGameWindow (
            GameWindows * gameWin )
```

Free the memory allocated for game windows.

This function frees the memory allocated for the game windows.

Parameters

gameWin	Pointer to the game windows structure.

8.68.3.4 initCli()

```
void initCli ( )
```

Initialize the command-line interface (CLI).

This function initializes the command-line interface (CLI) using the curses library. It sets up the necessary configurations for curses, such as colors, cursor visibility, echoing, and keypad input.

8.68.3.5 initColor()

```
void initColor ( )
```

Initialize color pairs for curses.

This function initializes color pairs for curses. It enables the use of colors in the terminal window.

8.68.3.6 initGameWindow()

```
bool initGameWindow (
            GameWindows * gameWin )
```

Initialize the game windows.

This function initializes the game windows and sets the infoDraw flag to false.

Parameters

gameWin	Pointer to the game windows structure.

Returns

true if initialization is successful, false otherwise.

8.68.3.7 initWindow()

```
bool initWindow (
            WINDOW ** fieldW,
            WINDOW ** infoW )
```

Initialize windows.

This function creates and initializes the field window and the information window.

Parameters

fieldW	Pointer to a pointer to the field window.
infoW	Pointer to a pointer to the information window.

8.68.3.8 printInfoWindow()

```
void printInfoWindow (
            int win,
            const char * gameName,
            const GameInfo_t gameInfo )
```

Print info.

This function prints game information.

Parameters

win	The window that will be drawn.
gameName	Name game.
gameInfo	Pointer to the GameInfo_t structure containing information about the game.

8.68.3.9 refreshUIWin()

```
void refreshUIWin (
            const char * gameName,
            GameInfo_t gameInfo,
            GameWindows * gameWin )
```

Refresh the user interface windows.

This function refreshes the game field window and the game information window.

Parameters

gameName	Name game.
gameInfo	The current game information.
gameWin	The struct window.

8.68.3.10 update()

```
void update (
            const char * gameName,
            GameWindows * gameWin,
            GameInfo_t gameInfo,
            const char * addition )
```

Update function for the game state.

This function updates the game state and refreshes the user interface window accordingly.

Parameters

gameName	Game name.
gameWin	Pointer to the game windows.
gameInfo	The game information.
addition	The game addition information.

8.69 GameUI.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include <ncurses.h>
00019
00020 #include "../../../brick_game/common/Info/Info.h"
00021
00028 #define MAX_CELLS 200
00029 #define WINDOW_FIELD_HEIGHT HFIELD + 2
00030 #define HEIGHT_BOARD_NEXT 6
00031 #define HEIGHT_WIN_PAUSE 3
```

```
00032 #define HEIGHT_WIN_START 9
00033 #define HEIGHT_WIN_EXIT 9
00035 #define WINDOW_FIELD_WIDTH WFIELD * 2 + 2
00036 #define WIDTH_BOARD_NEXT 15
00037 #define WIDTH_WIN_PAUSE 40
00038 #define WIDTH_WIN_START 45
00039 #define WIDTH_WIN_EXIT 40
00041 #define WIN_PAUSE 0
00042 #define WIN_START 1
00043 #define WIN_EXIT 2
00044 #define WIN_FIELD 3
00045 #define SIZE_BOX 1
00047 #define DRAW_POS_Y1 1
00048 #define DRAW_POS_Y2 3
00049 #define DRAW_POS_Y3 5
00050 #define DRAW_POS_Y4 7
00052 #define START_X_BOAR \
00053   3
00054 #define START_Y_BOAR \
00055   3
00062 #define GET_POS_X1(x) x * 2 + 1
00063
00069 #define GET_POS_X2(x) GET_POS_X1(x) + 1
00070
00076 #define GETMAXWH(height, width) getmaxyx(stdscr, height, width);
00082 #define SET_COLOR_PAIR(win, index) wbkgdset(win, COLOR_PAIR(index));
00083
00084 #define CHECK_WIN(win) (win != NULL)
00085 #define CHECK_FIELD(field) (field != NULL)
00091 #define DRAW_BOX(win) box(win, 0, 0);
00092
00098 #define COLOR_BOX(win, index) wbkgd(win, COLOR_PAIR(index));
00099
00105 #define COLOR_TEXT(win, index) wattron(win, COLOR_PAIR(index));
00106
00113 #define DRAW_CELL(x, y, win)            \
00114   mvwaddch(win, y, GET_POS_X1(x), ' '); \
00115   mvwaddch(win, y, GET_POS_X2(x), ' ');
00116
00121 #define GET_INFO_PRINT(index)                                          \
00122   (((index) == ITNAME)        ? "TETRIS"                               \
00123    : ((index) == ISNAME)      ? "SNAKE"                                \
00124    : ((index) == IHIGH_SCORE) ? "HIGH SCORE: "                         \
00125    : ((index) == ISCORE)      ? "SCORE: "                              \
00126    : ((index) == ILEVEL)      ? "LEVEL: "                              \
00127    : ((index) == INEXT)       ? "NEXT"                                 \
00128    : ((index) == IPAUSE)      ? "PAUSE"                                \
00129    : ((index) == INAMEBG)     ? "Brick Games"                          \
00130    : ((index) == IPRESSENTER) ? "START Game press <Enter>"             \
00131    : ((index) == IPRESSESC)   ? "EXIT Game press <q>"                  \
00132    : ((index) == ILOSE)       ? "You lose"                             \
00133    : ((index) == IWIN)        ? "You Win"                              \
00134    : ((index) == IEXITORREST) ? "Press <q> to exit or <ENTER> to restart" \
00135    : ((index) == ISTART)      ? "START"                                \
00136    : ((index) == IEXIT)       ? "EXIT"                                 \
00137    : ((index) == ISELECT)     ? "SELECT THE GAME AND THEN CLICK START" \
00138    : ((index) == ISSTART)     ? "start"                                \
00139    : ((index) == ISEND)       ? "end"                                  \
00140    : ((index) == ISNOT)       ? "not"                                  \
00141                               : NULL)
00142
00150 typedef enum {
00151   ITNAME = 0,
00152   ISNAME = 1,
00153   IHIGH_SCORE = 2,
00154   ISCORE = 3,
00155   ILEVEL = 4,
00156   INEXT = 5,
00157   IPAUSE = 6,
00158   INAMEBG = 7,
00159   IPRESSENTER = 8,
00160   IPRESSESC = 9,
00161   ILOSE = 10,
00162   IWIN = 11,
00163   IEXITORREST = 12,
00164   ISTART = 13,
00165   IEXIT = 14,
00166   ISELECT = 15,
00167   ISSTART = 16,
00168   ISEND = 17,
00169   ISNOT = 18
00170 } INDEX_TEXT_INFO;
00171
00175 typedef enum {
00176   BLACK_MAGENTA = 1,
00177   WHITE_BLACK = 2,
00178   BLACK_CYAN = 3,
```

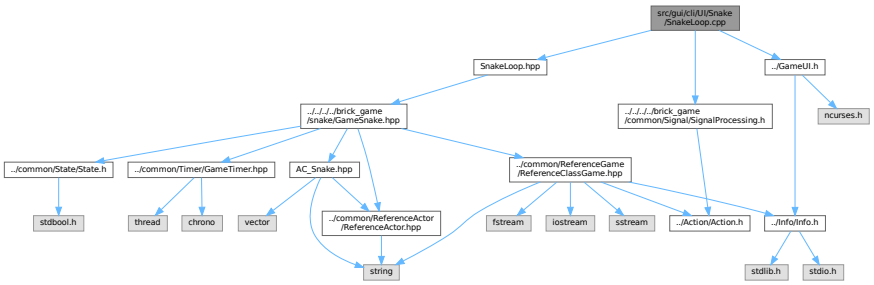
```
00179   BLACK_RED = 4,
00180   BLACK_WHITE = 5,
00181   BLACK_GREEN = 6,
00182   RED_BLACK = 7,
00183   BLACK_YELLOW = 8,
00184   BLACK_BLUE = 9,
00185   GREEN_BLACK = 10
00186 } Color;
00187
00191 typedef struct {
00192   int height;
00193   int width;
00195   int y;
00196   int x;
00197 } SizeText;
00198
00206 typedef struct {
00207   bool infoDraw;
00208   WINDOW *fieldw;
00209   WINDOW *infow;
00210 } GameWindows;
00211
00212 void initCli();
00213 void initColor();
00214 bool initWindow(WINDOW **fieldW, WINDOW **infoW);
00215 bool initGameWindow(GameWindows *gameWin);
00216
00217 void delCli();
00218 void deleteWindow(WINDOW **win);
00219 void freeGameWindow(GameWindows *gameWin);
00220
00221 void printInfoWindow(int win, const char *gameName, const GameInfo_t gameInfo);
00222 void refreshUIWin(const char *gameName, GameInfo_t gameInfo,
00223                   GameWindows *gameWin);
00224
00225 void update(const char *gameName, GameWindows *gameWin, GameInfo_t gameInfo,
00226             const char *addition);
00227
00228 #ifdef __cplusplus
00229 }
00230 #endif
```

8.70 src/gui/cli/UI/Snake/SnakeLoop.cpp File Reference

Snake game header file.

```
#include "SnakeLoop.hpp"
#include "../../../../brick_game/common/Signal/SignalProcessing.h"
#include "../GameUI.h"
```

Include dependency graph for SnakeLoop.cpp:



Functions

- void game_snake ()

  Starts the game loop.

8.70.1 Detailed Description

Snake game header file.

Author

nenamaxi (an.veringe@gmail.co)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

8.70.2 Function Documentation

8.70.2.1 game_snake()

```
void game_snake ( )
```

Starts the game loop.

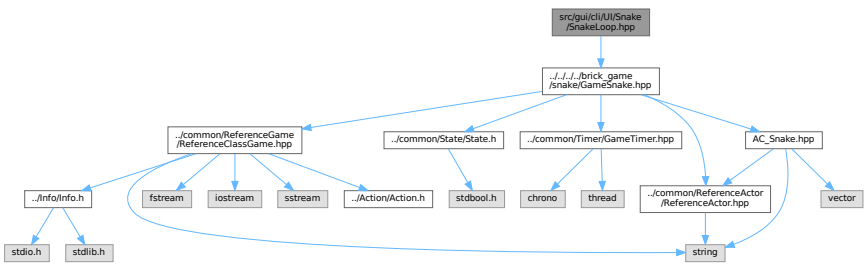
This function initializes game settings, initializes the command-line interface, initializes game components, and runs the game loop. It continuously updates the game state based on user input and thread operations until the game state becomes EXIT.

8.71 src/gui/cli/UI/Snake/SnakeLoop.hpp File Reference

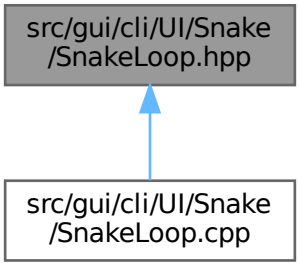
Snake game header file.

```
#include "../../../../brick_game/snake/GameSnake.hpp"
```

Include dependency graph for SnakeLoop.hpp:



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



Functions

- void game_snake ()
  Starts the game loop.

8.71.1 Detailed Description

Snake game header file.

Author

nenamaxi (an.veringe@gmail.co)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

8.71.2 Function Documentation

8.71.2.1 game_snake()

```
void game_snake ( )
```

Starts the game loop.

This function initializes game settings, initializes the command-line interface, initializes game components, and runs the game loop. It continuously updates the game state based on user input and thread operations until the game state becomes EXIT.

8.72 SnakeLoop.hpp

Go to the documentation of this file.

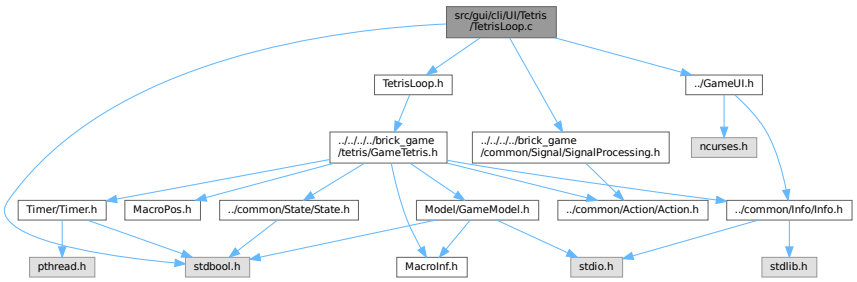
```
00001
00012 #pragma once
00013
00014 #include "../../../../brick_game/snake/GameSnake.hpp"
00015
00016 void game_snake();
```

8.73 src/gui/cli/UI/Tetris/TetrisLoop.c File Reference

Tetris game source file.

```
#include "TetrisLoop.h"
#include <stdbool.h>
#include "../../../../brick_game/common/Signal/SignalProcessing.h"
#include "../GameUI.h"
```

Include dependency graph for TetrisLoop.c:



Functions

- void game_tetris ()

  Starts the game loop.

8.73.1 Detailed Description

Tetris game source file.

Author

nenamaxi (an.veringe@gmail.co)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

8.73.2 Function Documentation

8.73.2.1 game_tetris()

```
void game_tetris ( )
```

Starts the game loop.

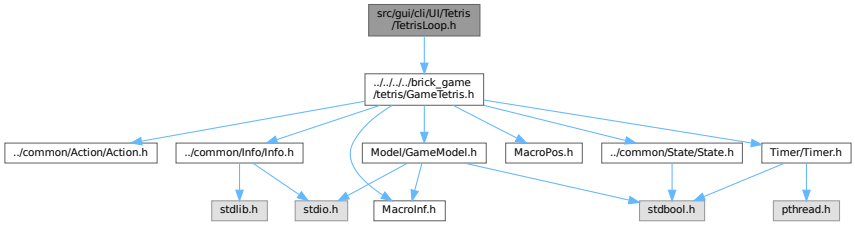
This function initializes game settings, initializes the command-line interface, initializes game components, and runs the game loop. It continuously updates the game state based on user input and thread operations until the game state becomes EXIT.

8.74 src/gui/cli/UI/Tetris/TetrisLoop.h File Reference

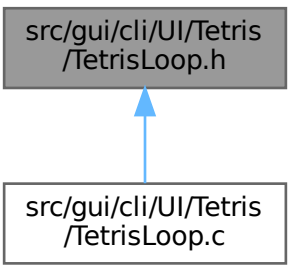
Tetris game header file.

```
#include "../../../../brick_game/tetris/GameTetris.h"
```

Include dependency graph for TetrisLoop.h:



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



Functions

- void game_tetris ()

  Starts the game loop.

8.74.1 Detailed Description

Tetris game header file.

Author

nenamaxi (an.veringe@gmail.co)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

8.74.2 Function Documentation

8.74.2.1 game_tetris()

```
void game_tetris ( )
```

Starts the game loop.

This function initializes game settings, initializes the command-line interface, initializes game components, and runs the game loop. It continuously updates the game state based on user input and thread operations until the game state becomes EXIT.

8.75 TetrisLoop.h

Go to the documentation of this file.

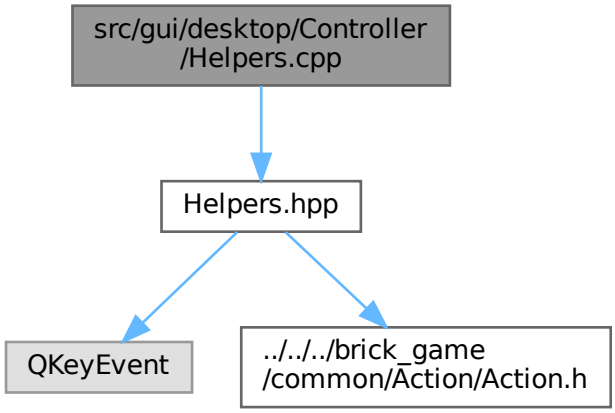
```
00001
00012 #pragma once
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 #include "../../../../brick_game/tetris/GameTetris.h"
00019
00020 void game_tetris();
00021
00022 #ifdef __cplusplus
00023 }
00024 #endif
```

8.76 src/gui/desktop/Controller/Helpers.cpp File Reference

Auxiliary file for the controller.

```
#include "Helpers.hpp"
```

Include dependency graph for Helpers.cpp:



Functions

- UserAction_t s21::QKeyEventToUserAction (const QKeyEvent *event)

  Converts a QKeyEvent to a UserAction_t.

8.76.1 Detailed Description

Auxiliary file for the controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-11

Copyright

8.76.2 Function Documentation

8.76.2.1 QKeyEventToUserAction()

```
UserAction_t s21::QKeyEventToUserAction (
            const QKeyEvent * event )
```

Converts a QKeyEvent to a UserAction_t.

Parameters

event	The QKeyEvent to convert.

Returns

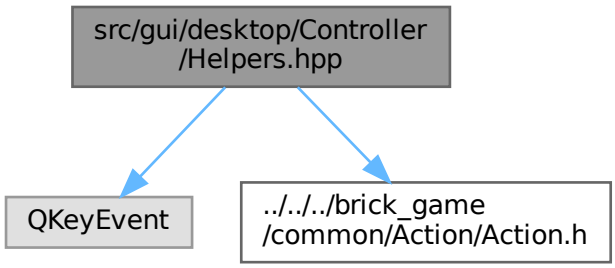
The converted UserAction_t.

8.77 src/gui/desktop/Controller/Helpers.hpp File Reference

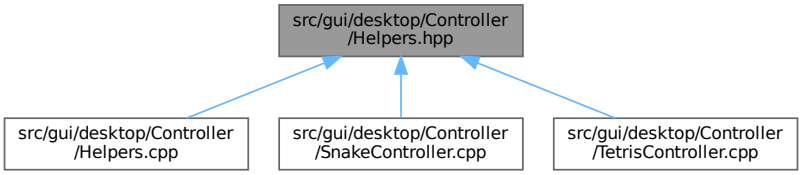
Auxiliary file for the controller.

```
#include <QKeyEvent>
#include "../../../brick_game/common/Action/Action.h"
```

Include dependency graph for Helpers.hpp:



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



Functions

- UserAction_t s21::QKeyEventToUserAction (const QKeyEvent *event)

  Converts a QKeyEvent to a UserAction_t.

8.77.1 Detailed Description

Auxiliary file for the controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-11

Copyright

Copyright (c) 2024

8.77.2 Function Documentation

8.77.2.1 QKeyEventToUserAction()

```
UserAction_t s21::QKeyEventToUserAction (
            const QKeyEvent * event )
```

Converts a QKeyEvent to a UserAction_t.

Parameters

event	The QKeyEvent to convert.

Returns

The converted UserAction_t.

8.78 Helpers.hpp

Go to the documentation of this file.

```
00001
```

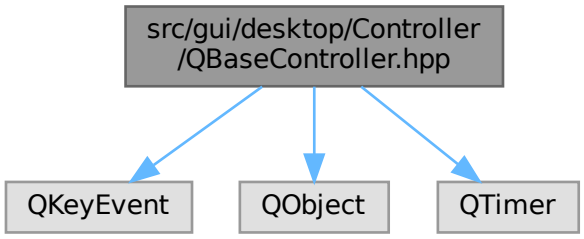
```
00012 #pragma once
00013
00014 #include <QKeyEvent>
00015
00016 #include "../../../brick_game/common/Action/Action.h"
00017 namespace s21 {
00018 UserAction_t QKeyEventToUserAction(const QKeyEvent *event);
00019 }
```

8.79 src/gui/desktop/Controller/QBaseController.hpp File Reference

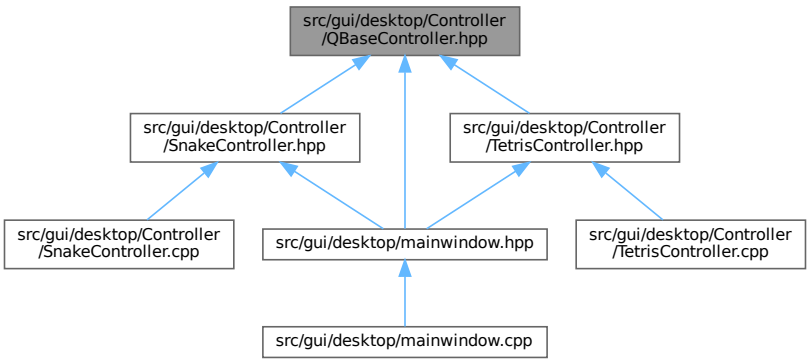
Header file base controller.

```
#include <QKeyEvent>
#include <QObject>
#include <QTimer>
```

Include dependency graph for QBaseController.hpp:



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



Classes

- class s21::QBaseGameController

  A base class for game controllers.

8.79.1 Detailed Description

Header file base controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

8.80 QBaseController.hpp

Go to the documentation of this file.

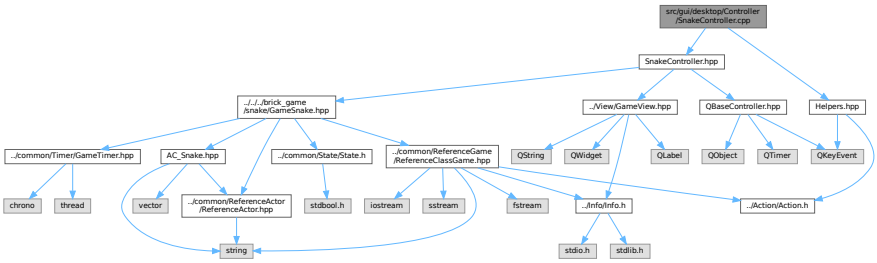
```
00001
00012 #pragma once
00013
00014 #include <QKeyEvent>
00015 #include <QObject>
00016 #include <QTimer>
00017
00018 namespace s21 {
00019
00023 class QBaseGameController : public QObject {
00024   Q_OBJECT
00025
00026  protected:
00027   QTimer *timer_input;
00028   QTimer *timer_output;
00030  public:
00036   explicit QBaseGameController(QObject *parent = nullptr)
00037       : QObject(parent),
00038         timer_input(new QTimer(this)),
00039         timer_output(new QTimer(this)){};
00040
00044   virtual ~QBaseGameController() = default;
00045
00049   virtual void run() = 0;
00050
00054   virtual void stop() = 0;
00055
00056  public slots:
00061   virtual void updateView() = 0;
00062
00067   virtual void sendInputSignal() = 0;
00068
00074   virtual void onUserActionReceived(const QKeyEvent *event) = 0;
00075
00076  signals:
00080   void finished();
00081 };
00082
00083 }  // namespace s21
```

8.81 src/gui/desktop/Controller/SnakeController.cpp File Reference

Source file Snake Controller.

```
#include "SnakeController.hpp"
#include "Helpers.hpp"
```

Include dependency graph for SnakeController.cpp:



8.81.1 Detailed Description

Source file Snake Controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

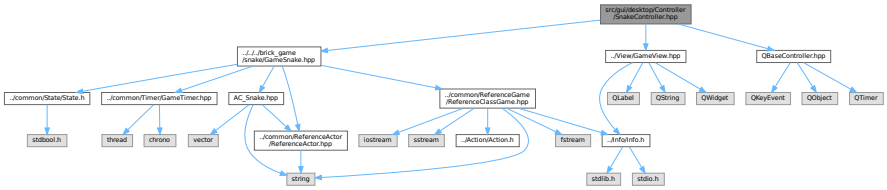
8.82 src/gui/desktop/Controller/SnakeController.hpp File Reference

Header file Snake Controller.

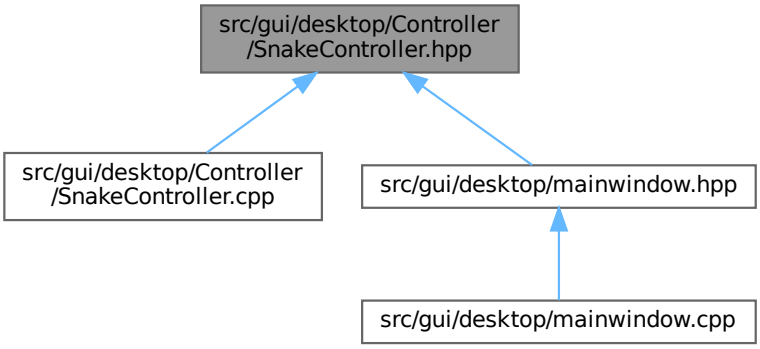
```
#include "../../../brick_game/snake/GameSnake.hpp"
#include "../View/GameView.hpp"
```

```
#include "QBaseController.hpp"
```

Include dependency graph for SnakeController.hpp:



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



Classes

- class s21::SnakeController

  A controller for the Snake game.

8.82.1 Detailed Description

Header file Snake Controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

8.83 SnakeController.hpp

Go to the documentation of this file.

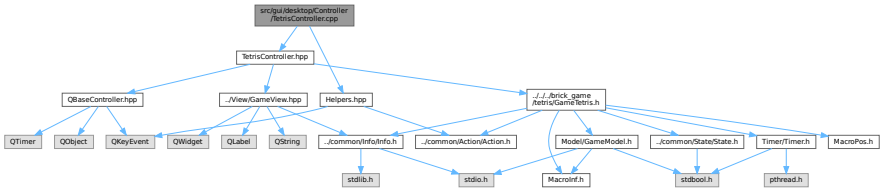
```
00001
00012 #pragma once
00013
00014 #include "../../../brick_game/snake/GameSnake.hpp"
00015 #include "../View/GameView.hpp"
00016 #include "QBaseController.hpp"
00017 namespace s21 {
00018
00022 class SnakeController : public QBaseGameController {
00023   Q_OBJECT
00024
00025  protected:
00026   GameSnake *model_;
00027   GameView *view_;
00028   UserAction_t action_;
00030  public:
00031   explicit SnakeController(GameSnake *model, GameView *view,
00032                            QObject *parent = nullptr);
00033
00034   ~SnakeController() = default;
00035   void run() override;
00036   void stop() override;
00037
00038  public slots:
00039   void updateView() override;
00040   void sendInputSignal() override;
00041   void onUserActionReceived(const QKeyEvent *event) override;
00042 };
00043 }  // namespace s21
```

8.84 src/gui/desktop/Controller/TetrisController.cpp File Reference

Source file Tetris Controller.

```
#include "TetrisController.hpp"
#include "Helpers.hpp"
```

Include dependency graph for TetrisController.cpp:



8.84.1 Detailed Description

Source file Tetris Controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

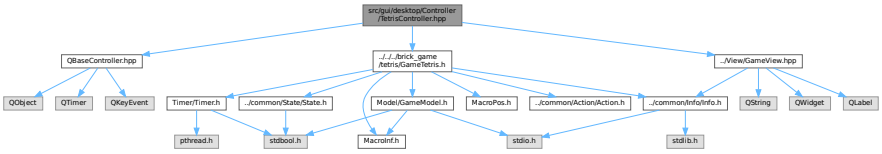
Copyright

8.85 src/gui/desktop/Controller/TetrisController.hpp File Reference

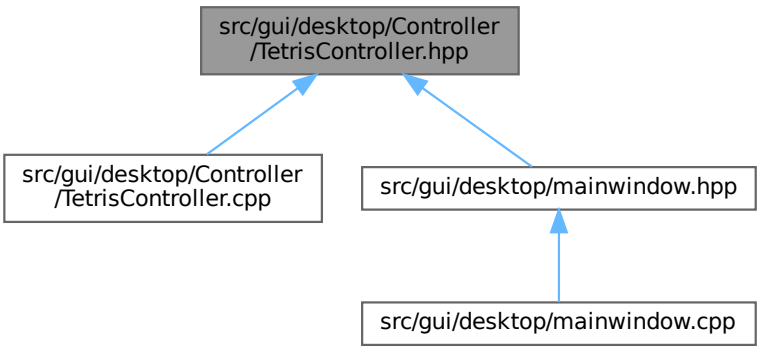
Header file Tetris Controller.

```
#include "QBaseController.hpp"
#include "../../../brick_game/tetris/GameTetris.h"
#include "../View/GameView.hpp"
```

Include dependency graph for TetrisController.hpp:



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



Classes

- class s21::TetrisController

  A controller for the Tetris game.

8.85.1 Detailed Description

Header file Tetris Controller.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

8.86 TetrisController.hpp

Go to the documentation of this file.

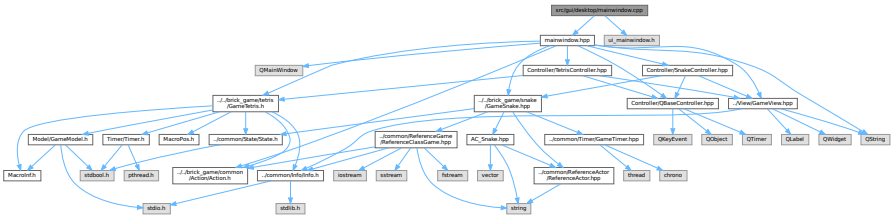
```
00001
00012 #pragma once
00013
00014 #include "QBaseController.hpp"
00015
00016 extern "C" {
00017 #include "../../../brick_game/tetris/GameTetris.h"
00018 }
00019 #include "../View/GameView.hpp"
00020
00021 namespace s21 {
00022
00026 class TetrisController : public QBaseGameController {
00027   Q_OBJECT
00028
00029  protected:
00030   GameTetris *model_;
00031   GameView *view_;
00032   UserAction_t action_;
00034  public:
00035   explicit TetrisController(GameTetris *model, GameView *view,
00036                             QObject *parent = nullptr);
00037   ~TetrisController() = default;
00038
00039   void run() override;
00040   void stop() override;
00041
00042  public slots:
00043   void updateView() override;
00044   void sendInputSignal() override;
00045   void onUserActionReceived(const QKeyEvent *event) override;
00046 };
00047
00048 }  // namespace s21
```

8.87 src/gui/desktop/mainwindow.cpp File Reference

Source file mainwindow desktop.

```
#include "mainwindow.hpp"
#include "ui_mainwindow.h"
```

Include dependency graph for mainwindow.cpp:



8.87.1 Detailed Description

Source file mainwindow desktop.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

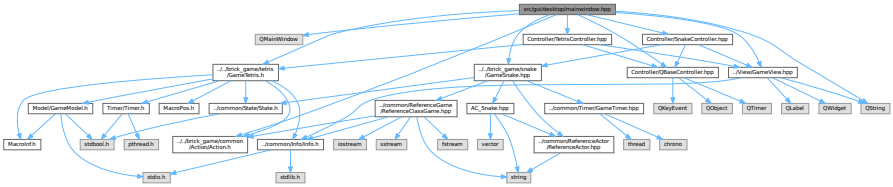
8.88 src/gui/desktop/mainwindow.hpp File Reference

Header file mainwindow desktop.

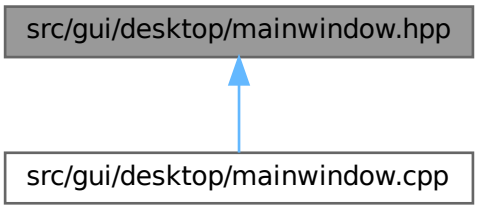
```
#include <QMainWindow>
#include <QString>
#include "../../brick_game/common/Action/Action.h"
#include "../../brick_game/tetris/GameTetris.h"
#include "../../brick_game/snake/GameSnake.hpp"
#include "Controller/QBaseController.hpp"
#include "Controller/SnakeController.hpp"
#include "Controller/TetrisController.hpp"
```

```
#include "View/GameView.hpp"
```

Include dependency graph for mainwindow.hpp:



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



Classes

- class s21::MainWindow

  The main window class for the game.

Variables

- constexpr int s21::BT_WIDTH = 300
- constexpr int s21::BT_HEIGHT = 50
- const QString s21::NOT_STYLE = ""
- const QString s21::PRESSED_STYLE

  Button style.

8.88.1 Detailed Description

Header file mainwindow desktop.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

8.88.2 Variable Documentation

8.88.2.1 BT_HEIGHT

```
constexpr int s21::BT_HEIGHT = 50  [constexpr]
```

Button height

8.88.2.2 BT_WIDTH

```
constexpr int s21::BT_WIDTH = 300  [constexpr]
```

Button width

8.88.2.3 NOT_STYLE

```
const QString s21::NOT_STYLE = ""
```

Without style

8.88.2.4 PRESSED_STYLE

```
const QString s21::PRESSED_STYLE
```

Initial value:

```
=
    "QPushButton {"
    "        background-color: #3a6cb1;"
    "        color: #282c34;"
    "        border: none;"
    "        border-radius: 10px;"
    "        padding: 10px;"
    "        font-size: 16px;"
    "}"
    "QPushButton:hover {"
    "        background-color: #61afef;"
    "}"
    "QPushButton:pressed {"
    "        background-color: #528bde;"
    "}"
```

Button style.

8.89 mainwindow.hpp

Go to the documentation of this file.

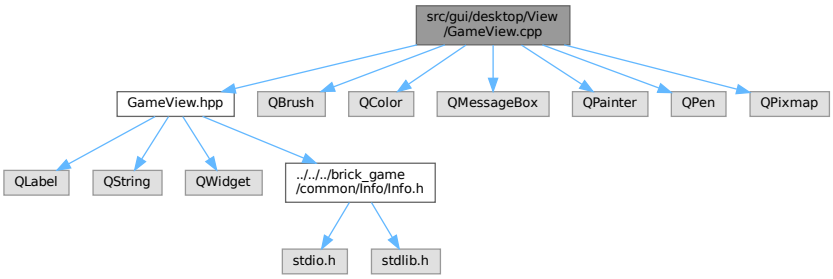
```
00001
00012 #pragma once
00013
00014 #include <QMainWindow>
00015 #include <QString>
00016
00017 extern "C" {
00018 #include "../../brick_game/common/Action/Action.h"
00019 #include "../../brick_game/tetris/GameTetris.h"
00020 }
00021
00022 #include "../../brick_game/snake/GameSnake.hpp"
00023 #include "Controller/QBaseController.hpp"
00024 #include "Controller/SnakeController.hpp"
00025 #include "Controller/TetrisController.hpp"
00026 #include "View/GameView.hpp"
00027
00028 namespace Ui {
00029 class MainWindow;
00030 }
00031
00032 namespace s21 {
00033
00034 constexpr int BT_WIDTH = 300;
00035 constexpr int BT_HEIGHT = 50;
00037 const QString NOT_STYLE = "";
00041 const QString PRESSED_STYLE =
00042     "QPushButton {"
00043     "        background-color: #3a6cb1;"
00044     "        color: #282c34;"
00045     "        border: none;"
00046     "        border-radius: 10px;"
00047     "        padding: 10px;"
00048     "        font-size: 16px;"
00049     "}"
00050     "QPushButton:hover {"
00051     "        background-color: #61afef;"
00052     "}"
00053     "QPushButton:pressed {"
00054     "        background-color: #528bde;"
00055     "}";
00056
00060 class MainWindow : public QMainWindow {
00061   Q_OBJECT
00062
00063  public:
00064   explicit MainWindow(QWidget *parent = nullptr);
00065   ~MainWindow();
00066
00067  private:
00068   void refreshFinishConnect(void);
00069   void setEnableMenu(bool enable);
00070   void createController();
00071
00072  private slots:
00073   void on_PB_Start_clicked();
00074   void on_PB_Tetris_clicked();
00075   void on_PB_Snake_clicked();
00076   void on_PB_Exit_clicked();
00077
00078   void isGameFinished();
00079
00080  private:
00081   Ui::MainWindow *ui;
00083   QString game_select;
00085   s21::GameSnake *GSnake;
00086   GameTetris *GTetris;
00088   GameView *View;
00089   QBaseGameController
00090       *controller;
00091 };
00092
00093 }  // namespace s21
```

8.90 src/gui/desktop/View/GameView.cpp File Reference

Source file game view.

```
#include "GameView.hpp"
#include <QBrush>
#include <QColor>
#include <QMessageBox>
#include <QPainter>
#include <QPen>
#include <QPixmap>
```

Include dependency graph for GameView.cpp:



8.90.1 Detailed Description

Source file game view.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

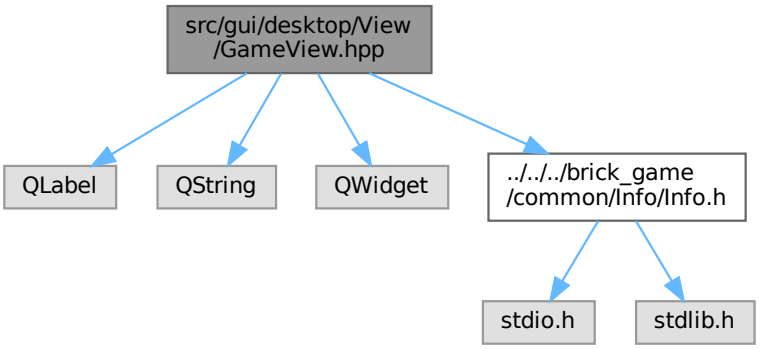
8.91 src/gui/desktop/View/GameView.hpp File Reference

Header file game view.

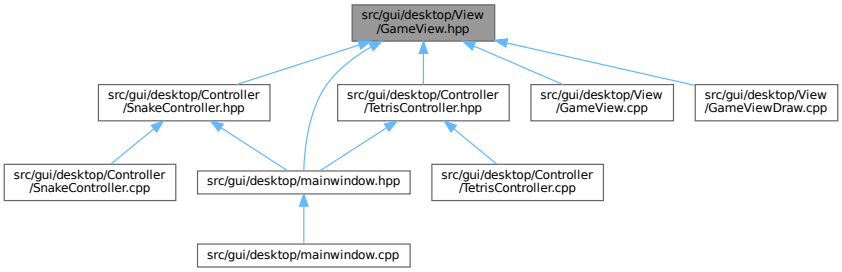
```
#include <QLabel>
#include <QString>
#include <QWidget>
```

```
#include "../../../brick_game/common/Info/Info.h"
```

Include dependency graph for GameView.hpp:



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



Classes

- struct s21::UserInterface_t

  A structure to hold the properties of the user interface.
- class s21::GameView

  A class for representing a game view.

Variables

- const QString **s21::CS_start** = "start"

  The start constant.
- const QString **s21::CS_end** = "end"

  The end constant.
- const QString **s21::CS_not** = "not"

  The not constant.
- const QString **s21::CS_pause** = "PAUSE"

  The pause constant.

- const QString **s21::CS_restart** = "restart"

  The restart constant.
- const QString **s21::CS_NBG** = "Brick Game"

  The name of the brick game constant.
- const QString **s21::CS_NSNAKE** = "SNAKE"

  The name of the snake game constant.
- const QString **s21::CS_NTETRIS** = "TETRIS"

  The name of the Tetris game constant.
- const QString **s21::CS_SLOGO** = ":/im/images/SnakeLogo.jpg"

  The logo path for the snake game constant.
- const QString **s21::CS_TLOGO** = ":/im/images/TetrisLogo.png"

  The logo path for the Tetris game constant.
- const QString **s21::CS_press_enter** = "Press <enter> to "

  The text for pressing enter to start a game constant.
- const QString **s21::CS_esc_or_exit** = "or <esc> to exit to menu"

  The text for pressing escape to exit to the menu constant.
- const QString **s21::CS_lose** = "You LOSE"

  The text for losing the game constant.
- const QString **s21::CS_win** = "You WIN"

  The text for winning the game constant.
- const QString **s21::CS_bad_im** = "Bad image"

  The text for bad image error constant.
- const QString **s21::CS_score** = "SCORE: %1"

  The score format string constant.
- const QString **s21::CS_hscore** = "HIGH SCORE: %1"

  The high score format string constant.
- const QString **s21::CS_level** = "LEVEL: %1"

  The level format string constant.

8.91.1 Detailed Description

Header file game view.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

8.92 GameView.hpp

Go to the documentation of this file.

```
00001
00012 #pragma once
00013
00014 #include <QLabel>
00015 #include <QString>
00016 #include <QWidget>
00017
00018 extern "C" {
00019 #include "../../../brick_game/common/Info/Info.h"
00020 }
00021
00022 namespace s21 {
00023
00027 const QString CS_start = "start";
00028
00032 const QString CS_end = "end";
00033
00037 const QString CS_not = "not";
00038
00042 const QString CS_pause = "PAUSE";
00043
00047 const QString CS_restart = "restart";
00048
00052 const QString CS_NBG = "Brick Game";
00053
00057 const QString CS_NSNAKE = "SNAKE";
00058
00062 const QString CS_NTETRIS = "TETRIS";
00063
00067 const QString CS_SLOGO = ":/im/images/SnakeLogo.jpg";
00068
00072 const QString CS_TLOGO = ":/im/images/TetrisLogo.png";
00073
00077 const QString CS_press_enter = "Press <enter> to ";
00078
00082 const QString CS_esc_or_exit = "or <esc> to exit to menu";
00083
00087 const QString CS_lose = "You LOSE";
00088
00092 const QString CS_win = "You WIN";
00093
00097 const QString CS_bad_im = "Bad image";
00098
00102 const QString CS_score = "SCORE: %1";
00103
00107 const QString CS_hscore = "HIGH SCORE: %1";
00108
00112 const QString CS_level = "LEVEL: %1";
00113
00117 static constexpr int cellSize = 30;
00118
00122 static constexpr int fieldWidth = WFIELD;
00123
00127 static constexpr int fieldHeight = HFIELD;
00128
00132 static constexpr int nextWidth = WNEXT;
00133
00137 static constexpr int nextHeight = HNEXT;
00138
00142 static constexpr int fieldWidthP = fieldWidth * cellSize;
00143
00147 static constexpr int fieldHeightP = fieldHeight * cellSize;
00148
00152 static constexpr int nextWidthP = nextWidth * cellSize;
00153
00157 static constexpr int nextHeightP = nextHeight * cellSize;
00158
00162 static constexpr int indentation1x = 10;
00163
00167 static constexpr int indentation2x = indentation1x * 2;
00168
00172 static constexpr int indentation3x = indentation1x * 3;
00173
00177 static constexpr int indentation4x = indentation1x * 4;
00178
00182 static constexpr int indentation5x = indentation1x * 5;
00183
00187 static constexpr int indentation7x = indentation1x * 7;
00188
00192 static constexpr int indentation11x = indentation1x * 11;
00193
00197 static constexpr int indentation13x = indentation1x * 13;
```

```
00198
00202 static constexpr int WIMAGE = 300;
00203
00207 static constexpr int HIMAGE = 230;
00208
00212 static constexpr int POS_X1 = 20;
00213
00217 static constexpr int POS_Y_3_CELLS = 10 + 3 * cellSize;
00218
00222 static constexpr int POS_Y_4_CELLS = 10 + 4 * cellSize;
00223
00227 static constexpr int MAX_COUNT_CELLS = 200;
00228
00232 struct UserInterface_t {
00233   int offsetY;
00234   int offsetX;
00236   int fieldOffsetX;
00237   int fieldOffsetY;
00239   int line;
00241   int nextOffsetX;
00242   int nextOffsetY;
00244   int WWindow;
00245   int HWindow;
00246 };
00247
00251 class GameView : public QWidget {
00252   Q_OBJECT
00253
00254  public:
00255   explicit GameView(QWidget *parent = nullptr);
00256
00257   void updateGameInfo(const GameInfo_t &info, const QString addition = "not");
00258
00259   void setGameSelected(const QString &gameName);
00260   QString getGameSelected(void);
00261
00262   void keyPressEvent(QKeyEvent *event) override;
00263
00264  signals:
00270   void userActionReceived(const QKeyEvent *event);
00271
00272  protected:
00273   void paintEvent(QPaintEvent *event) override;
00274
00275   void drawInfo(QPainter &painter, QBrush &brush);
00276   void drawField(QPainter &painter, QBrush &brush);
00277   void drawNext(QPainter &painter, QBrush &brush);
00278   void initSetting();
00279
00280   QColor getColor(const int code);
00281
00282   void drawInfoAddition(QPainter &painter, const int infoOffsetY);
00283   void drawInfoImage(QPainter &painter, QRect &infoRect, const int infoOffsetY);
00284
00285  private:
00286   GameInfo_t gameInfo;
00288   QString addition;
00290   QString gameSelected;
00292   QPixmap QP_image;
00294   UserInterface_t
00295       setting;
00296 };
00297
00298 }  // namespace s21
```

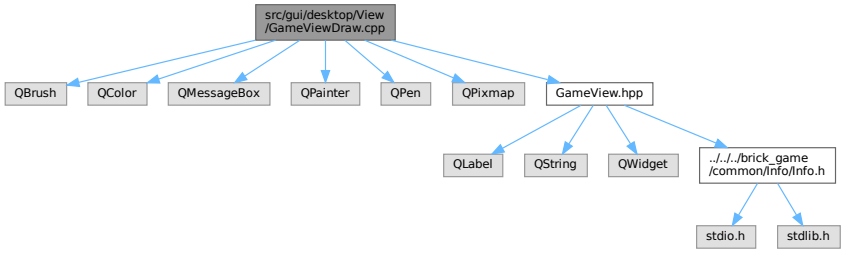
8.93 src/gui/desktop/View/GameViewDraw.cpp File Reference

Source file game view.

```
#include <QBrush>
#include <QColor>
#include <QMessageBox>
#include <QPainter>
#include <QPen>
#include <QPixmap>
```

```
#include "GameView.hpp"
```

Include dependency graph for GameViewDraw.cpp:



8.93.1 Detailed Description

Source file game view.

Author

nenamaxi (an.veringe@gmail.com)

Version

0.1

Date

2024-08-10

Copyright

Copyright (c) 2024

Index