# Liar's dice

Generated by Doxygen 1.12.0

# Chapter 1

# Liar-s-Dice-C-

Liar's Dice made using C++

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

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

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

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

# Chapter 5

# Class Documentation

## 5.1 AI_Player Class Reference

AI player derives everything from **Player** (p. 14) and has ai logic.

`#include <Player.h>`

Inherits **Player**.

### Public Member Functions

- **AI_Player** (int player_num)
  *Creates player but sets is_ai = true.*
- ~**AI_Player** ()=default
  *Default constructor.*
- void **evaluate_bid** (int bid[2], int prev_bid[2], int dice_count, int turn_player)
  *Evaluates the last bid and makes a new bid or returns false to call a liar.*
- void **reset_ai** ()
  *Resets ai components.*
- **Player** (int playerID)
  *Constructor for a player.*

### Public Member Functions inherited from Player

- **Player** (int playerID)
  *Constructor for a player.*
- void **reroll_cup** ()
  *Rerolls dices in cup.*
- int **contains_dice** (int dice)
  *Returns the number of dices that this player has. Counts 1's as the target dice since it's wild.*
- int **get_cup_size** ()
  *Get the number of dices in cup.*
- void **remove_a_die** ()
  *Removes a die from the cup.*
- void **print_dice** ()
  *Prints the die that are in the cup.*
- std::vector< int > **get_dices** () const
  *Get dices that this player has.*
- virtual ~**Player** ()=default
  *Default contructor.*

**Additional Inherited Members**

## Public Attributes inherited from Player

- bool **is_ai** = false
- int **playerID** = 0

## Protected Attributes inherited from Player

- std::unique_ptr< **Cup** > **_cup**

### 5.1.1 Detailed Description

AI player derives everything from **Player** (p. 14) and has ai logic.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 AI_Player()

```
AI_Player::AI_Player (
            int player_num)
```

Creates player but sets is_ai = true.

**Parameters**

| | |
|---|---|
| *player_num* | |

### 5.1.3 Member Function Documentation

#### 5.1.3.1 evaluate_bid()

```
void AI_Player::evaluate_bid (
            int bid[2],
            int prev_bid[2],
            int dice_count,
            int turn_player)
```

Evaluates the last bid and makes a new bid or returns false to call a liar.

**Parameters**

| | |
|---|---|
| *bid* | new bid |
| *prev_bid* | previous bid |
| *dice_count* | how many dices are left |

**Returns**

If ai will make a bid

#### 5.1.3.2 Player()

```
Player::Player (
            int playerID)
```

Constructor for a player.

**Parameters**

| | |
|---|---|
| *player_num* | |

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

- Player.h
- Player.cpp

## 5.2 Cup Class Reference

Contains dices that can be edited (reroll, etc.)

`#include <Cup.h>`

### Public Member Functions

- void **roll_dice** ()

  *Rolls all the dices in the cup.*
- void **reduce_size** ()

  *Reduced the number of dices in cup by one.*
- int **how_many_of_x_dice** (int dice)

  *Checks how many copies of given dice does the cup have.*

### Public Attributes

- int **cup_size** = 5
- std::vector< int > **dices**

### 5.2.1 Detailed Description

Contains dices that can be edited (reroll, etc.)

### 5.2.2 Member Function Documentation

#### 5.2.2.1 how_many_of_x_dice()

```
int Cup::how_many_of_x_dice (
            int dice)
```

Checks how many copies of given dice does the cup have.

**Parameters**

| | |
|---|---|
| *dice* | |

**Returns**

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

- Cup.h
- Cup.cpp

## 5.3 DrawDice Class Reference

Class used to render dice values.

`#include <DrawDice.h>`

#### Public Member Functions

- **DrawDice** (int relative_x, int relative_y, int dice_w, int dice_h, int padding=10)
- void **draw_shape** (SDL_Renderer ∗renderer, int type_id) const

### 5.3.1 Detailed Description

Class used to render dice values.

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

- DrawDice.h
- DrawDice.cpp

## 5.4 GameState Class Reference

Game state handles the entire game.

`#include <Gamestate.h>`

#### Public Member Functions

- **GameState** (int player_count)

  *Creates game state with given players players.*
- void **start_new_round** (int winner_id, int loser_id)

  *Start a new round of liar's dice. Removes a die from the winner if there was one.*
- bool **make_bid** (int count, int dice, int playerID)

  *Evaluates if the bid is legal and returns if it's legal.*
- int **call** ()

  *Calculates the how many copies of last bid dice there was.*
- void **next_player** ()

  *Updates the turnplayer.*
- int **get_previous_player_id** ()

  *Get the index of the previous player.*
- bool **is_game_over** () const

  *Check if game is over.*

#### Public Attributes

- std::vector< std::shared_ptr< **Player** > > **players**
- std::shared_ptr< **Player** > **turnplayer**
- int **d_count** = 0
- int **d_last_bid** [2] = { 0, 0 }

### 5.4.1 Detailed Description

Game state handles the entire game.

**Parameters**

| | |
|---|---|
| *player_count* | How many players can be created (I originally planned there to be multiple) |

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 GameState()

```
GameState::GameState (
            int player_count)
```

Creates game state with given players players.

**Parameters**

| | |
|---|---|
| *player_count* | |

### 5.4.3 Member Function Documentation

#### 5.4.3.1 call()

```
int GameState::call ()
```

Calculates the how many copies of last bid dice there was.

**Returns**

number of dices found

#### 5.4.3.2 get_previous_player_id()

```
int GameState::get_previous_player_id ()
```

Get the index of the previous player.

**Returns**

index of the previous player

#### 5.4.3.3 is_game_over()

```
bool GameState::is_game_over () const
```

Check if game is over.

**Returns**

if game is over

#### 5.4.3.4 make_bid()

```
bool GameState::make_bid (
            int count,
            int dice,
            int playerID)
```

Evaluates if the bid is legal and returns if it's legal.

**Parameters**

| | |
|---|---|
| *count* | number of dices bid |
| *dice* | dice value bid |
| *playerID* | player making the bid |

**Returns**

if the bid is legal

#### 5.4.3.5 start_new_round()

```
void GameState::start_new_round (
            int winner_id,
            int loser_id)
```

Start a new round of liar's dice. Removes a die from the winner if there was one.

**Parameters**

| | |
|---|---|
| *winner_id* | Who was the winner of last round (-1 = no winner) |
| *loser_id* | Who was the loser of the last round (-1 = no loser) |

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

- Gamestate.h
- Gamestate.cpp

## 5.5 Player Class Reference

**Player** (p. 14) class.

`#include <Player.h>`

Inherited by **AI_Player**.

### Public Member Functions

- **Player** (int playerID)
  *Constructor for a player.*
- void **reroll_cup** ()
  *Rerolls dices in cup.*
- int **contains_dice** (int dice)
  *Returns the number of dices that this player has. Counts 1's as the target dice since it's wild.*
- int **get_cup_size** ()
  *Get the number of dices in cup.*
- void **remove_a_die** ()
  *Removes a die from the cup.*
- void **print_dice** ()
  *Prints the die that are in the cup.*
- std::vector< int > **get_dices** () const
  *Get dices that this player has.*
- virtual ~**Player** ()=default
  *Default contructor.*

**Public Attributes**

- bool **is_ai** = false
- int **playerID** = 0

**Protected Attributes**

- std::unique_ptr< **Cup** > **_cup**

### 5.5.1 Detailed Description

**Player** (p. 14) class.

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 Player()

```
Player::Player (
            int playerID)
```

Constructor for a player.

**Parameters**

| | |
|---|---|
| *player_num* | |

### 5.5.3 Member Function Documentation

#### 5.5.3.1 contains_dice()

```
int Player::contains_dice (
            int dice)
```

Returns the number of dices that this player has. Counts 1's as the target dice since it's wild.

**Parameters**

| | |
|---|---|
| *dice* | the dice looked for |

**Returns**

Count of dices that the cup has

#### 5.5.3.2 get_cup_size()

```
int Player::get_cup_size ()
```

Get the number of dices in cup.

**Returns**

number of dices in cup

#### 5.5.3.3 get_dices()

```
std::vector< int > Player::get_dices () const
```

Get dices that this player has.

**Returns**

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

- Player.h
- Player.cpp

# Chapter 6

# File Documentation

## 6.1 Cup.h

```
00001 #pragma once
00002 #include <random>
00003 #include <vector>
00007 class Cup
00008 {
00009 public:
00010     // How many die can the cup contain
00011     int cup_size = 5;
00012     // Dices in cup
00013     std::vector<int> dices;
00014
00015     Cup();
00019     void roll_dice();
00023     void reduce_size();
00029     int how_many_of_x_dice(int dice);
00030 };
```

## 6.2 DrawDice.h

```
00001 #pragma once
00002 #include <SDL.h>
00003
00007 class DrawDice {
00008 public:
00009     DrawDice(int relative_x, int relative_y, int dice_w, int dice_h, int padding = 10)
00010         : _relative_x(relative_x), _relative_y(relative_y),
00011         _dice_w(dice_w), _dice_h(dice_h), _padding_div(padding) {};
00012
00013     void draw_shape(SDL_Renderer* renderer, int type_id) const;
00014
00015 private:
00016     int _relative_x;
00017     int _relative_y;
00018     int _dice_w;
00019     int _dice_h;
00020     int _padding_div;
00021
00022     void draw_dot(SDL_Renderer* renderer, int x_i, int y_i) const;
00023     void get_dot_relative_pos(int& pos, const int& i, int axis, int size) const;
00024     void draw_unknown(SDL_Renderer* renderer) const;
00025 };
```

## 6.3 Gamestate.h

```
00001 #pragma once
00002 #include "Player.h"
00003 #include <vector>
00004 #include <memory>
00005
```

```
00010 class GameState
00011 {
00012 public:
00013     // Players
00014     std::vector<std::shared_ptr<Player» players;
00015     // Turnplayer
00016     std::shared_ptr<Player> turnplayer;
00017     // Number of dices total between all players
00018     int d_count = 0;
00019     // Last bid (x copies of x dice)
00020     int d_last_bid[2] = { 0, 0 };
00021
00026     GameState(int player_count);
00027     ~GameState() = default;
00028
00034     void start_new_round(int winner_id, int loser_id);
00035
00043     bool make_bid(int count, int dice, int playerID);
00044
00045
00050     int call();
00051
00055     void next_player();
00056
00061     int get_previous_player_id();
00062
00067     bool is_game_over() const;
00068
00069 private:
00070
00071
00072     // Current player index
00073     int _current_player_i = 0;
00074     // How many rounds have been played
00075     int _round_num = 0;
00076     // _d_values prevents players of making the same exact bid
00077     std::vector<int> _d_values;
00078 };
00079
```

## 6.4 Player.h

```
00001 #pragma once
00002 #include <random>
00003 #include <iostream>
00004 #include <memory>
00005
00006 #include "Cup.h"
00010 class Player
00011 {
00012 public:
00013     // Is this player an ai
00014     bool is_ai = false;
00015     // Id of the player
00016     int playerID = 0;
00017
00022     Player(int playerID);
00023
00027     void reroll_cup();
00028
00035     int contains_dice(int dice);
00036
00041     int get_cup_size();
00042
00046     void remove_a_die();
00047
00051     void print_dice();
00052
00057     std::vector<int> get_dices() const;
00058
00062     virtual ~Player() = default;
00063
00064 protected:
00065     // Contains die and methods to interact with the die
00066     std::unique_ptr<Cup> _cup;
00067 };
00068
00072 class AI_Player : public Player
00073 {
00074 public:
00075     using Player::Player;
00080     AI_Player(int player_num);
00081
```

```
00085     ~AI_Player() = default;
00086
00094     void evaluate_bid(int bid[2], int prev_bid[2], int dice_count, int turn_player);
00095
00099     void reset_ai();
00100
00101 private:
00102     // Dice that the ai likes to bid this round
00103     int _dice_to_bid = 0;
00104     // Count of the die that the ai likes to bid this round
00105     int _dice_count_to_bid = 0;
00106
00113     void should_copy_bid(int has_prev_dices, int& target_dice, int  prev_bid[2]) const;
00114
00123     void select_dice_count_for_bid(float probability, int target_dice, int d_count, int  prev_bid[2],
      int& count) const;
00124
00133     float binomial_coefficient(int& n, int& x);
00134
00141     bool should_call_liar(float& probability, int& has_prev_dices);
00142
00147     void get_most_common_die();
00148
00158     void calculate_own_probability(int dice_count, int target_dice, int count, float& probability);
00159
00166     void calculate_probability(float& probability, int n, int x);
00167 };
00168
00169
```

## 6.5 Render_text.h

```
00001 #pragma once
00002 #pragma once
00003 #include <SDL.h>
00004 #include <SDL_ttf.h>
00005 #include <string>
00006 #include <vector>
00007
00008 // Define screen
00009 static const int SCREEN_WIDTH = 480;
00010 static const int SCREEN_HEIGHT = 640;
00011 static const int DICE_SIZE = 88;
00012 static const int BUTTON_POS = 400;
00013
00025 void render_text(SDL_Renderer* renderer, TTF_Font* font, const std::string& text, SDL_Color color, int
      x, int y, int w, int h);
00026
00036 void clear_area(SDL_Renderer* renderer, SDL_Color color, int x, int y, int w, int h);
```