# Casino

Generated by Doxygen 1.11.0

# Chapter 1

# Casino

Simple practising tool meant for fun and as a proof of concept

TODO list:

- Create games
  - ~~Black Jack~~
  - Poker
  - ~~Dices~~
  - Roulette
  - ~~Double or Nothing~~
- ~~Add currence/balance~~
- ~~Add the ability to save your balance for later log in~~
- **GUI :)**

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# File Index

## 3.1 File List

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

# Chapter 4

# Class Documentation

## 4.1 BJ Struct Reference

**Public Member Functions**

- **BJ** ( **Player** &player)

  *Constructs a new instance of the **BJ** (*p. 7*) class.*
- void **startGame** ( **Player** &player)

  *Starts the BlackJack game.*
- void **rndCard** (int[ ], **Hand** &)
- void **giveCards** (int[ ], **Hand** &, **Hand** &)
- void **showCards** ( **Hand** &, **Hand** &)
- void **printCard** (int)
- void **calculateScore** ( **Hand** &)
- void **dealersMove** (int[ ], **Hand** &, **Hand** &)

### 4.1.1 Constructor & Destructor Documentation

#### 4.1.1.1 BJ()

```
BJ::BJ (
              Player & player)
```

Constructs a new instance of the **BJ** (p. 7) class.

This constructor initializes a new instance of the **BJ** (p. 7) class and starts the game for the specified player.

**Parameters**

| | |
|---|---|
| *player* | The player object for whom the game is being started. |

### 4.1.2 Member Function Documentation

#### 4.1.2.1 startGame()

```
void BJ::startGame (
             Player & player)
```

Starts the BlackJack game.

This function allows the player to play the BlackJack game. It takes a reference to a **Player** (p. 12) object as a parameter. The function prompts the player to enter their bet, deals the cards, and allows the player to make choices (hit or stand). After the player's turn, the function determines the outcome of the game and updates the player's balance accordingly. The game continues until the player runs out of balance or chooses to quit. At the end of the game, the function displays the total wins and losses of the player.

**Parameters**

| | |
|---|---|
| *player* | The **Player** (p. 12) object representing the player. |

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

- BJ.h
- BJ.cpp

## 4.2 Dice Struct Reference

**Public Member Functions**

- **Dice** ( **Player** &player)

  *Constructs a new instance of the* ***Dice*** *(p. 8) class.*
- void **startGame** ( **Player** &player)

### 4.2.1 Constructor & Destructor Documentation

#### 4.2.1.1 Dice()

```
Dice::Dice (
             Player & player)
```

Constructs a new instance of the **Dice** (p. 8) class.

This constructor initializes a new instance of the **Dice** (p. 8) class and starts the game for the specified player.

**Parameters**

| | |
|---|---|
| *player* | The player object for whom the game is being started. |

### 4.2.2 Member Function Documentation

#### 4.2.2.1 startGame()

```
void Dice::startGame (
              Player & player)
```

Starts the dice game.

This function allows the player to play the dice game. It takes a reference to a **Player** (p. 12) object as a parameter. The player is prompted to enter their bet and choose what to bet on (red, blue, or tie). The dice are rolled and the outcome is determined. If the player wins, their balance is increased and the winnings are displayed. If the player loses, their balance is decreased and a message is displayed. The player can choose to play again or quit the game.

**Parameters**

| | |
|---|---|
| *player* | The player object for whom the game is being started. |

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

- Dice.h
- Dice.cpp

## 4.3 DorN Struct Reference

**Public Member Functions**

- **DorN** ( **Player** &player)

  *Constructs a new instance of the* ***DorN*** *(p. 9) (Double or nothing) class.*
- void **startGame** ( **Player** &player)

### 4.3.1 Constructor & Destructor Documentation

#### 4.3.1.1 DorN()

```
DorN::DorN (
              Player & player)
```

Constructs a new instance of the **DorN** (p. 9) (Double or nothing) class.

This constructor initializes a new instance of the **DorN** (p. 9) class and starts the game for the specified player.

**Parameters**

| | |
|---|---|
| *player* | The player object for whom the game is being started. |

### 4.3.2 Member Function Documentation

#### 4.3.2.1 startGame()

```
void DorN::startGame (
              Player & player)
```

Starts the Double or Nothing game.

This function allows the player to play the Double or Nothing game. The player is prompted to enter their bet and choose heads or tails. The outcome is determined randomly, and the player's balance is updated accordingly. The game continues until the player's balance reaches zero or the player chooses to exit. At the end of the game, the total number of wins and losses is displayed.

**Parameters**

| | |
|---|---|
| *player* | The player object for whom the game is being started. |

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

- DorN.h
- DorN.cpp

## 4.4 Hand Class Reference

**Public Member Functions**

- **Hand** ()

  *Constructs a new instance of the* ***Hand*** *(*p. *11) class. used to keep track of the player's hand in the game of BlackJack.*

**Public Attributes**

- int **nCards** [10]
- int **aces**
- int **score**
- int **altScore**
- int **n**
- bool **end**

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

- Hand.h
- Hand.cpp

## 4.5 Login Struct Reference

**Public Member Functions**

- tuple< std::string, double, int, int > **getPlayer** ()

  *Reads the player data from the CSV file.*
- void **changeCSV** (double bet, int additionW, int additionL)
- void **printList** ()

### 4.5.1 Member Function Documentation

#### 4.5.1.1 changeCSV()

```
void Login::changeCSV (
             double bet,
             int additionW,
             int additionL)
```

Changes the CSV according to the input of +- bet which reduces or increases the **Player** (p. 12)'s account balance. Additionaly increments wins or loses in the csv file.

#### 4.5.1.2 getPlayer()

```
tuple< std::string, double, int, int > Login::getPlayer ()
```

Reads the player data from the CSV file.

This function reads the player data from the CSV file and stores it in the respective vectors. the **Player** (p. 12) is prompted to enter their username, if the username is not in the names vector the code repeats until a recognised username is entered.

#### 4.5.1.3 printList()

```
void Login::printList ()
```

Replaces the current csv file with an updated one using vectors of names, balance, wins and losses.

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

- Login.h
- Login.cpp

## 4.6 Player Class Reference

**Public Member Functions**

- **Player** ()

  ***Player*** *(*p. *12) class to store player data of balance, wins and losses.*
- **Player** (std::string initialName, int initialBalance, int initialWins, int initialLosses)
- int **getBalance** () const
- int **getWins** () const
- int **getLosses** () const
- const std::string & **getName** () const
- void **setName** (const std::string &newName)
- void **setBalance** (int newBalance)
- void **setWins** (int newWins)
- void **setLosses** (int newLosses)
- void **increaseBalance** (int amount)
- void **decreaseBalance** (int amount)
- void **incrementWins** ()
- void **incrementLosses** ()

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

- Player.h
- Player.cpp

# Chapter 5

# File Documentation

## 5.1 BJ.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "Player.h"
#include "Hand.h"

struct BJ
{
public:
    BJ(Player& player);
    ~BJ();
    void startGame(Player& player);
    void rndCard(int[], Hand&);
    void giveCards(int[], Hand&, Hand&);
    void showCards(Hand&, Hand&);
    void printCard(int);
    void calculateScore(Hand&);
    void dealersMove(int[],Hand&, Hand&);
};
#pragma once
```

## 5.2 Dice.h

```
#include "Player.h"
#include <iostream>

struct Dice{
    public:
        Dice(Player& player);
        ~Dice();

        void startGame(Player& player);
};
```

## 5.3 DorN.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "Player.h"

struct DorN
{
public:
    DorN(Player& player);
    ~DorN();
    void startGame(Player& player);
};
```

## 5.4 Hand.h

```
#ifndef HAND_H
#define HAND_H

#include <string>

class Hand{
public:
    Hand();
    int nCards[10];
    int aces;
    int score;
    int altScore;
    int n;
    bool end;
};

#endif
```

## 5.5 Login.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include "Player.h"

using namespace std;

struct Login
{
private:
    int counter = 0;
    int playerNum;
    vector<string> names;
    vector<double> balance;
    vector<int> wins;
    vector<int> losses;
public:
    tuple<std::string, double, int, int> getPlayer();
    void changeCSV(double bet, int additionW, int additionL);
    void printList();
};
```

## 5.6 Player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include <string>

class Player {
public:
    Player();
    Player(std::string initialName, int initialBalance, int initialWins, int initialLosses);

    int getBalance() const;
    int getWins() const;
    int getLosses() const;
    const std::string& getName() const;

    void setName(const std::string& newName);
    void setBalance(int newBalance);
    void setWins(int newWins);
    void setLosses(int newLosses);

    void increaseBalance(int amount);
    void decreaseBalance(int amount);
    void incrementWins();
    void incrementLosses();

private:
    std::string name;
    int balance;
    int wins;
    int losses;
};

#endif
```

# Index