Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Programowanie Komputerów

Silnik gry Makao

autor	Bartłomiej Pogodziński
prowadzący	mgr inż. Grzegorz Kwiatkowski
rok akademicki	2020/2021
kierunek	informatyka
rodzaj studiów	SSI
semestr	3
termin laboratorium	czwartek, 11:15 – 12:45
	piątek, 8:15 – 9:45
sekcja	22
termin oddania sprawozdania	2020-11-12

1 Temat zadania

Stworzenie silnika gry karcianej Makao umożliwiającego implementację własnych algorytmów gry i ich porównywania.

2 Analiza zadania

Symulowanie gry w stopniu umożliwiającym porównywanie rzeczywistych algorytmów gry wymaga silnika umożliwiającego odzwierciedlenie przebiegu prawdziwej rozgrywki na satysfakcjonującym poziomie, a także intuicyjnego interfejsu do przenoszenia własnej taktyki na algorytm. W tym projekcie zbudowana została pierwsza część omawianego tematu.

Aby gra karciana tego typu mogła zostać przeniesiona do symulatora wymagane jest, aby odrzucone zostały wszystkie domysły i niejednoznaczności zasad - a tych w Makao jest sporo. Gra ta różni się zależnie od kręgów kulturowych - nie tylko funkcjami pojedynczych kart, ale też całymi mechanizmami rozgrywki. Z tego powodu poczynione zostały pewne założenia.

1. Karty funkcyjne to:
    - Dwójka
    - Trójka
    - Czwórka
    - Walet
    - Dama pik
    - Król kier, pik
    - As
2. Dama pik uwalnia od aktualnych żądań
3. Król pik nadaje żądanie pięciu kart graczowi poprzedniemu podczas jego najbliższej tury
4. Zestaw kart, na przykład 3 karty o tej samej figurze, jest egzekwowany po kolei. Stąd na przykład kartę trójkę kier można odbić zestawem kolejnych kart: Król kier, Król trefl. Król kier odbija karty, a Król trefl ląduje na stole, gdy żądanie zostało już ”odbite”. Niemożliwe jest odbicie trójki trefl sekwencją: Król trefl, Król kier - Król trefl lądowałby na stole przed odbiciem żądania pobrania kart.
5. Postój jest ruchem. Z tego powodu gracz w postoju w każdej swojej turze “decyduje się” na dalsze oczekiwanie. Możliwe jest więc odbicie wielokrotnego żądania postoju czwórką po odczekaniu pierwszej kolejki.

6. Aby zachować zgodność z ogólnie przyjętą zasadą, dotyczącej faktu iż gracz w postój jest pomijany podczas żądania pobrania kart, aktywny ruch oczekiwania podczas postoju jest równoważny z przepisaniem wszystkich aktualnych żądań na gracza następnego.

Postawienie takich założeń umożliwiło jednoznaczne zdefiniowanie struktury programu oraz jego możliwości.

2.1 Struktury danych

W celu realizacji zadania konieczne było utworzenie dynamicznego kontenera przechowującego karty, graczy, nazwy algorytmów, pasujące kolory oraz figury do poszczególnych kart. W tym celu zaimplementowana została szablonowa *Lista jednokierunkowa* rozbudowana o dodatkowe funkcjonalności, do których należy należy między innymi iterator pozwalający na przejście po wszystkich elementach listy bez konieczności naruszania prywatności klasy oraz niepotrzebnego zwiększania złożoności obliczeniowej iterowania. Dzięki temu złożoność obliczeniowa dostępu do kolejnych elementów listy jest rzędu O(n).

Inną przydatną metodą jest metoda *removeMatching*, realizująca usunięcie elementów listy, które również znajdują się wewnątrz drugiej listy przesłanej jako parametr. Dzięki zastosowaniu wyżej wspomnianego iteratora złożoność obliczeniowa tej metody jest rzędu O(n*m).

Same karty zostały zrealizowane w postaci klas dziedziczących po klasie *Card*, zawierającej podstawowe informacje (jak kolor czy symbol), listę pasujących kolorów i symboli, żądanych pobrań kart postojów. Zawiera również wirtualne metody odpowiedzialne za przywrócenie karty do swojego stanu podstawowego - na przykład zresetowanie żądania w przypadku Waleta, oraz za egzekucję funkcji karty - między innymi usunięcie żądań pobrania kart lub postoju w przypadku Damy Pik. Egzekucja odbywa się poprzez przesłanie jako parametr struktury kar *Punishments* przygotowanej dla aktualnego gracza oraz zwrócenie tej samej struktury po modyfikacji wynikającej z funkcji karty.

Struktura kar *Punishments* składa się z zestawu żądań pobrania kart oraz postojów wiszących nad graczem poprzednim, aktualnym i następnym. Umożliwia ona uniknięcie modyfikowania parametrów graczy bezpośrednio przez karty - co wiązałoby się z pętlą zależności.

Klasa *Deck* imituje stos potasowanych kart na stole - posiada więc listę wskaźników na przechowywane karty oraz silnik losujący odpowiedzialny za umieszczanie podanego wskaźnika na kartę w liście na przypadkowej pozycji.

Każdy algorytm biorący udział w symulacji jest klasą dziedziczącą po klasie *Player* z przeciążoną metodą wyboru karty oraz własną unikatową nazwą. Klasa *Player* posiada pola opisujące żądania pobrania kart oraz postojów wiszących nad graczem, listę kart znajdujących się w jego

ręce oraz jego nazwę. Dodatkowo klasa ta zawiera wiele metod odpowiedzialnych za sprawdzenie czy zaproponowany ruch jest poprawny oraz egzekwujący cały ruch jednego gracza.

W symulatorze znajduje się również klasa *Game* zajmująca się obsługą jednej rozgrywki. Zawiera ona listę nazw algorytmów, generowaną na jej podstawie listę algorytmów, *Deck* oraz listę kart położonych na stole.

Szczegółowy opis klas i struktur znajduje się w załączniku.

3 Specyfikacja zewnętrzna

Aplikacja uruchamiana jest z wiersza poleceń. Po uruchomieniu użytkownikowi wyświetlana jest lista dostępnych algorytmów. Użytkownik wpisuje kolejno nazwy algorytmów - w tej kolejności będą oni wykonywać ruchy w symulacjach. W przypadku niewłaściwego wpisu wyświetlony zostanie komunikat o błędzie i użytkownik dostanie możliwość ponownego wpisu. Po wybraniu algorytmów gracz zostanie poproszony o wybranie liczby symulacji. Po zakończonym wpisie uruchomione zostaną kolejne symulacje. Stopień ich realizacji sygnalizowany będzie przez pasek postępu. Po zakończonych symulacjach wyświetlone zostaną procentowo stosunki gier wygranych, do poprawnie rozegranych dla każdego algorytmu i program zakończy działanie.

Przykładowe uruchomienie zaprezentowano poniżej.

```
bpogodzinski@ubuntu $ ./makao_sim
MACAU SIMULATOR by Bartłomiej Pogodziński 2020

Select algorithms:
adv-placeholder
placeholder

Type the names separated by space: adv-placeholder adv-placeholder adv-placeholder
How many games shall be played: 100000

Starting simulation
Progress: ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

Results:
adv-placeholder: 35.16%
adv-placeholder: 33.34%
adv-placeholder: 31.50%
bpogodzinski@ubuntu $
```

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem obiektowym. Klasy oraz struktury rozdzielone zostały między pliki oraz podfoldery, celem zwiększenia czytelności oraz modularności projektu.

4.1 Ogólna struktura programu

Program posiada budowę warstwową. Pierwszą warstwą jest warstwa interakcji z użytkownikiem. Zajmuje się tym klasa *MainInterface*. Zapewnia ona komunikaty o błędach wpisu, pasek postępu oraz wypisanie rezultatów. Ona generuje również instancje klasy Game, która odpowiedzialna jest za rozgrywanie poszczególnych rozgrywek. Z uwagi na niezależność każdego z obiektów *Game*, możliwe jest zachowanie niezależności obciążenia pamięciowego komputera od liczby wykonywanych symulacji.

Klasa *Game* odpowiada za inicjację każdego z graczy (klasa *Player*), talii kart (klasa *Deck*) oraz stosu kart już położonych. Kolejno wywołuje ona graczy do wykonania ruchów zapewniając im prawidłowe parametry jako gracza następnego oraz poprzedniego, stosu kart na stole i talii. Rozgrywa ruchy (za pomocą *Player.makeMove*) do momentu wygrania któregoś z graczy lub przekroczenia limitu długości gry.

Wykonywaniem kolejnych ruchów zajmują się metody klasy abstrakcyjnej *Player*. Posiada ona metody umożliwiające testowanie czy wybrana karta lub sekwencja kart jest prawidłowa. Odpowiedzialna jest także za wykonywanie ruchu poprzez metodę *makeMove* testującą prawidłowość wybranej sekwencji oraz opcjonalnie modyfikującej parametry graczy - poprzedniego, aktualnego i następnego - wykorzystując polimorfizm klasy abstrakcyjnej *Card*. Sam wybór karty polega na wywołaniu metody *choose* nadpisywanej przez każdy algorytm.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5 Testowanie

Program był testowany za pomocą algorytmowych placeholderów *placeholder* oraz *adv-placeholder*. Pierwszy z nich imituje gracza pasywnego - który pobiera kartę lub akceptuje postój niezależnie od zawartości swoich kart. Niemożliwe jest aby wygrał rozgrywkę. Drugi zaimplementowany algorytm znajduje pierwszą pasującą kartę w ręce i stara się dopasować do niej inne. Dzięki temu może imitować gracza lekko roztargnionego, lecz świadomego ogólnych zasad gry.

Dodatkowo część funkcjonalności została przetestowana za pomocą testów jednostkowych zawartych w pliku *test.h*.

Program został również przetestowany pod kątem wycieków pamięci za pomocą narzędzia *Valgrind*.

6 Wnioski

Budowa tego projektu była wyzwaniem przede wszystkim ze względu na silnie ograniczający krótki czas realizacji spowodowany blokowym systemem prowadzenia zajęć. Mimo tego projekt był bardzo dobrą okazją do zapoznania się z mechanizmami obiektowości, ogromnymi możliwościami jakie daje polimorfizm, czy projektowania aplikacji w sposób odpowiadający rzeczywistym obiektom.

Dodatkowo pozwolił on na zmierzenie się z trudnościami takimi jak zapętlone zależności. Pierwotnie metody *execute* klasy *Card* miały operować na obiektach graczy poprzez settery i gettery - i bezpośrednio nadawać im żądania pobrania kart lub postoju. NIestety rozwiązanie takie wymagałoby, aby Gracze korzystali z metod klas kart, które również operowałyby na metodach klasy gracza. Takie rozwiązanie zawierało błąd pętli zależności. Ostatecznie dodana została struktura *Punishments*, pozwalająca przełożyć modyfikowanie pól graczy na klasę gracza i usunąć problematyczną zależność w klasie kart.

Dodatek
Szczegółowy opis typów i funkcji

Projekt zaliczeniowy z PK3-SSI

Wygenerowano przez Doxygen 1.8.17

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Rozdział 2

Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 3

Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

Rozdział 4

Dokumentacja klas

4.1 Dokumentacja klasy Ace

`#include <Ace.h>`

Diagram dziedziczenia dla Ace

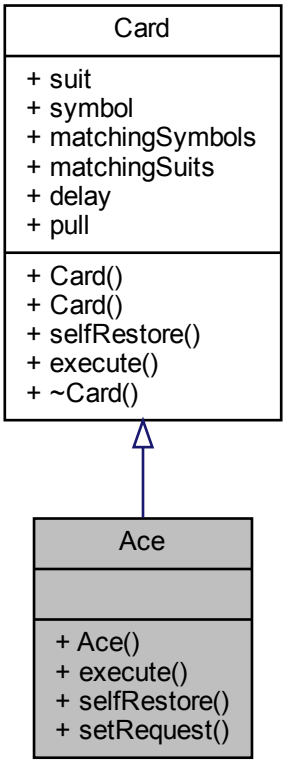
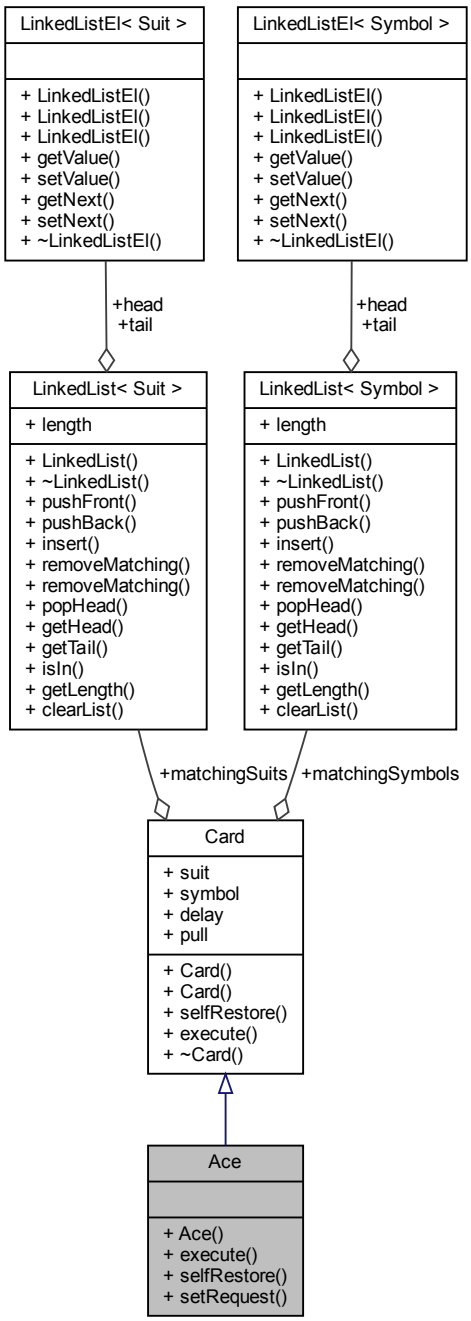


Diagram współpracy dla Ace:



Metody publiczne

- **Ace** (Suit _suit)
- Punishments execute (Punishments p) override
- void selfRestore () override
- void setRequest (Suit requestedSuit)

Dodatkowe Dziedziczone Składowe

4.1.1 Opis szczegółowy

Class representing the ace card - able to make card requests

4.1.2 Dokumentacja funkcji składowych

4.1.2.1 execute()

```
Punishments Ace::execute (
            Punishments p )  [inline], [override], [virtual]
```

Card execution placeholder

Parametry

p	punishments

Zwraca

unchanged punishments

Implementuje Card.

4.1.2.2 selfRestore()

```
void Ace::selfRestore ( )  [override], [virtual]
```

Restores the card to the basic state Removes the requests

Implementuje Card.

4.1.2.3 setRequest()

```
void Ace::setRequest (
            Suit requestedSuit )
```

Sets the card request

Parametry

requestedSuit	suit requested by the card placer

Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/game/Cards/Ace.h
- kod/game/Cards/Ace.cpp

4.2 Dokumentacja klasy AdvancedPlaceholder

```
#include <AdvancedPlaceholder.h>
```

Diagram dziedziczenia dla AdvancedPlaceholder

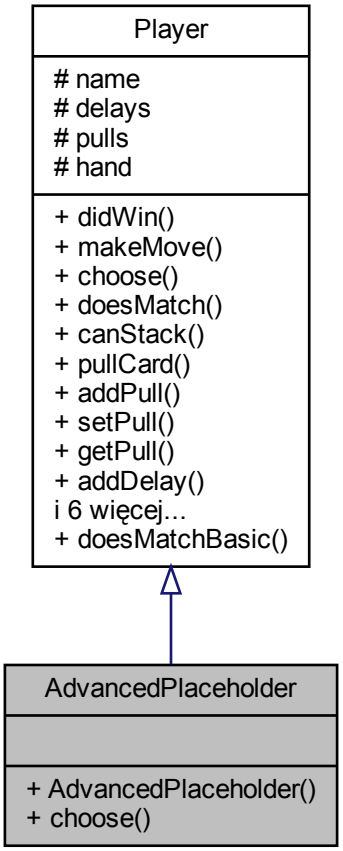
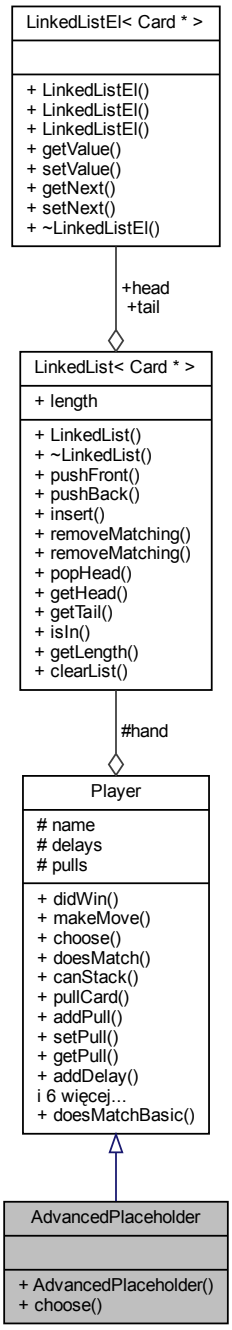


Diagram współpracy dla AdvancedPlaceholder:



Metody publiczne

- LinkedList< Card * > * **choose** (LinkedList< Card * > *cardsStack, int playersNumber, int *players↩ HandsLengths) override

Dodatkowe Dziedziczone Składowe

4.2.1 Opis szczegółowy

Advanced algorithm placeholder

Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/players/algorithms/AdvancedPlaceholder.h
- kod/players/algorithms/AdvancedPlaceholder.cpp

4.3 Dokumentacja klasy AgroCard

```
#include <AgroCard.h>
```

Diagram dziedziczenia dla AgroCard

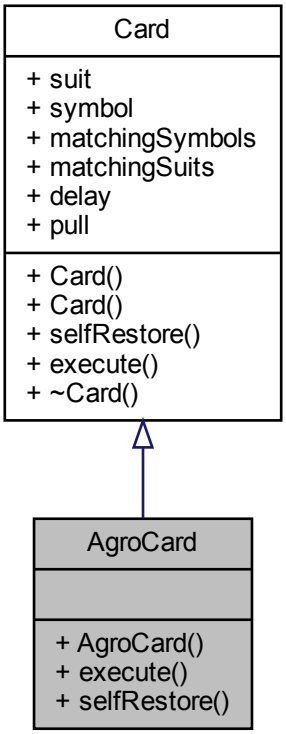
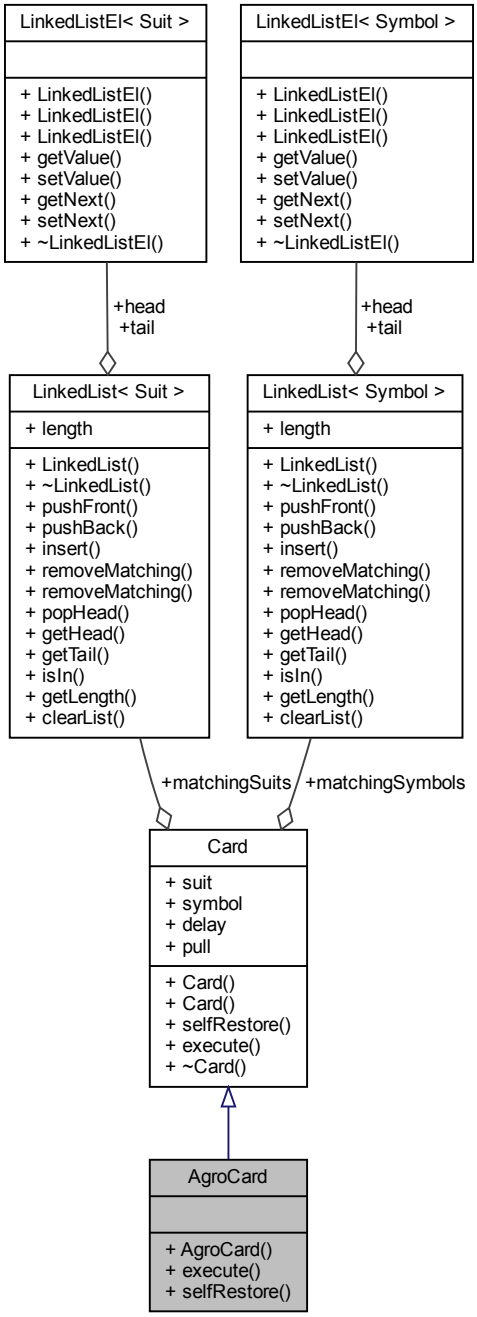


Diagram współpracy dla AgroCard:



Metody publiczne

- **AgroCard** (Suit _suit, Symbol _symbol, int _delay, int _pull, bool _forward=true)
- Punishments execute (Punishments) override
- void selfRestore () override

  Method restoring the card to basic state.

Dodatkowe Dziedziczone Składowe

4.3.1 Opis szczegółowy

Class representing the aggressive cards those setting the pull or delay on players

4.3.2 Dokumentacja funkcji składowych

4.3.2.1 execute()

```
Punishments AgroCard::execute (
            Punishments punishments )  [override], [virtual]
```

Executes the pull/delay

Zwraca

new set of punshments

Implementuje Card.

Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/game/Cards/AgroCard.h
- kod/game/Cards/AgroCard.cpp

4.4 Dokumentacja klasy BasicCard

Diagram dziedziczenia dla BasicCard

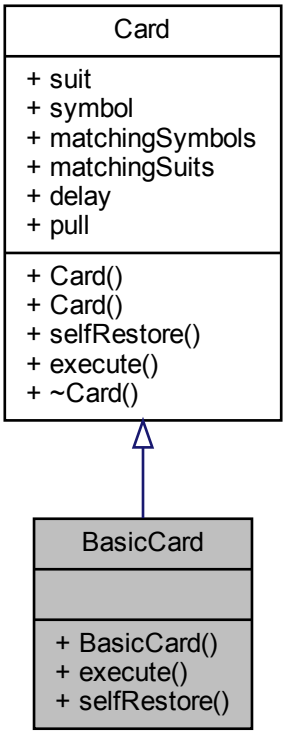
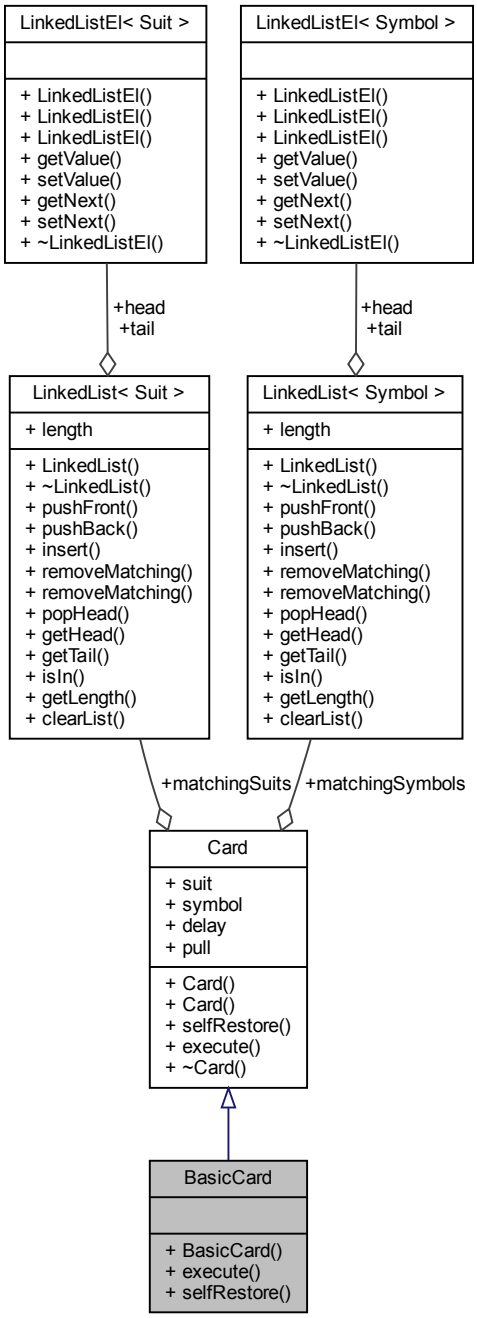


Diagram współpracy dla BasicCard:



Metody publiczne

- **BasicCard** (Suit _suit, Symbol _symbol)
- Punishments execute (Punishments p) override
- void selfRestore () override

Dodatkowe Dziedziczone Składowe

4.4.1 Dokumentacja funkcji składowych

4.4.1.1 execute()

```
Punishments BasicCard::execute (
            Punishments p )  [inline], [override], [virtual]
```

Blank execution

Implementuje Card.

4.4.1.2 selfRestore()

```
void BasicCard::selfRestore ( )  [inline], [override], [virtual]
```

Does not affect the card

Implementuje Card.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- kod/game/Cards/BasicCard.h

4.5 Dokumentacja klasy Card

```
#include <Card.h>
```

Diagram dziedziczenia dla Card

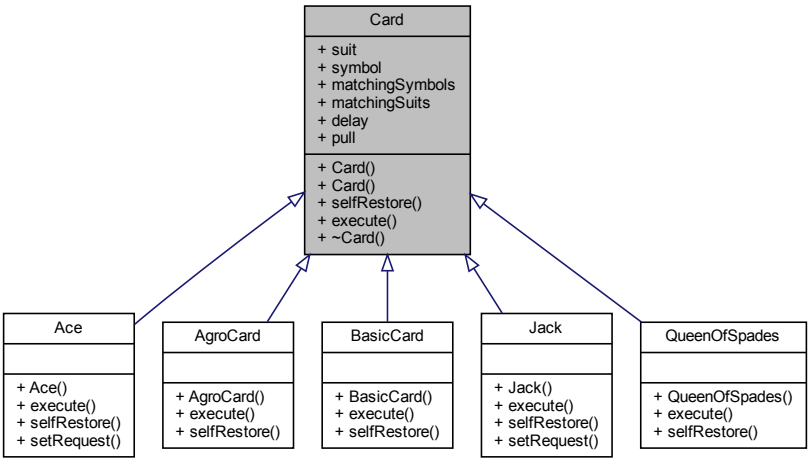
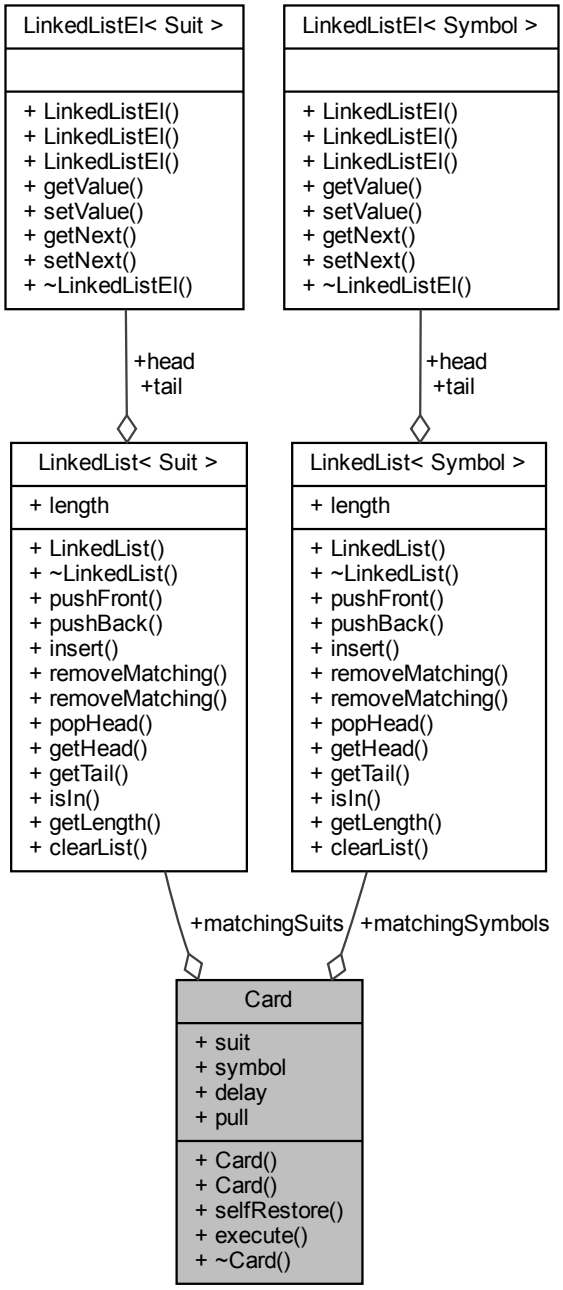


Diagram współpracy dla Card:



Metody publiczne

- Card (Suit _suit, Symbol _symbol)
- Card (Suit _suit, Symbol _symbol, int _delay, int _pull)
- virtual void selfRestore ()=0

  Method restoring the card to basic state.
- virtual Punishments execute (Punishments)=0

Method executing the extra card actions.

- virtual ~Card ()=default

  Default card destructor.

Atrybuty publiczne

- const Suit suit

  Suit or color of the card.

- const Symbol symbol

  Symbol, value of the card.

- LinkedList< Symbol > matchingSymbols

  List of matching symbols.

- LinkedList< Suit > matchingSuits

  List of matching suits.

- const int delay

  Delay given by card.

- const int pull

  Pull request given by card.

4.5.1 Opis szczegółowy

Card virtual base class

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 Card() [1/2]

```
Card::Card (
            Suit _suit,
            Symbol _symbol )  [inline]
```

Main card constructor for non aggresive cards

Parametry

_suit	Card suit
_symbol	Card symbol

4.5.2.2 Card() [2/2]

```
Card::Card (
            Suit _suit,
```

```
        Symbol _symbol,
        int _delay,
        int _pull )  [inline]
```

Card constructor for aggresive cards

Parametry

_suit	Card suit
_symbol	Card symbol
_delay	Delay given by the card
_pull	Pulls given by the card

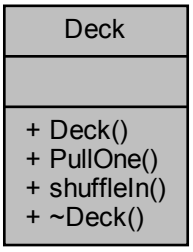
Dokumentacja dla tej klasy została wygenerowana z pliku:

- kod/game/Cards/Card.h

4.6 Dokumentacja klasy Deck

```
#include <Deck.h>
```

Diagram współpracy dla Deck:



Metody publiczne

- Deck (int seed)
- Card ∗ PullOne ()
- void shuffleIn (Card ∗)
- ∼Deck ()

4.6.1 Opis szczegółowy

Class representing the deck of cards in a game

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 Deck()

```
Deck::Deck (
            int seed )  [explicit]
```

Deck main construrctor

Parametry

seed	shuffling randomness seed

4.6.2.2 ~Deck()

```
Deck::~Deck ( )
```

Deck destructor Destructs only the elements which currently belong to the deck - other have to be destroyed by the players

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 PullOne()

```
Card * Deck::PullOne ( )
```

Pulls one card from card

Zwraca

card pointer

4.6.3.2 shuffleIn()

```
void Deck::shuffleIn (
            Card * cardPointer )
```

Shuffles one card into the deck

Parametry

card	pointer

Dokumentacja dla tej klasy została wygenerowana z plików:

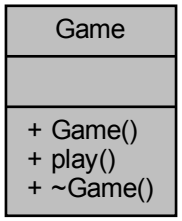
- kod/game/Deck.h
- kod/game/Deck.cpp

4.7 Dokumentacja klasy Game

ALGORITHMS.

`#include <Game.h>`

Diagram współpracy dla Game:



Metody publiczne

- Game (int randomSeed, int maxGameLength, LinkedList< std::string > &playerNames)
- int play ()
- ~Game ()

4.7.1 Opis szczegółowy

ALGORITHMS.

Class responsible for simulating one game

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 Game()

```
Game::Game (
            int randomSeed,
            int maxGameLength,
            LinkedList< std::string > & playerNames )
```

Game constructor

Parametry

randomSeed	Random number - has to be different throughout the games to enure results variability
maxGameLength	Max rounds to be played
playerNames	List of algorithm names

4.7.2.2 ~Game()

```
Game::~Game ( )
```

Game destructor

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 play()

```
int Game::play ( )
```

Runs the game simulation

Zwraca

ID of the winning player or -1 if the game was terminated due to timeout

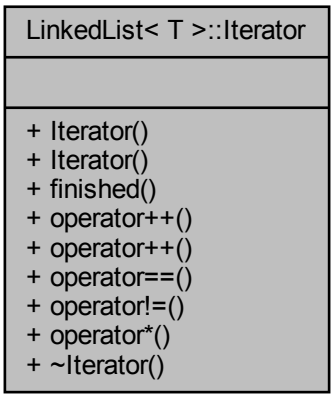
Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/game/Game.h
- kod/game/Game.cpp

4.8 Dokumentacja klasy LinkedList< T >::Iterator

`#include <LinkedList.h>`

Diagram współpracy dla LinkedList< T >::Iterator:



Metody publiczne

- **Iterator** (const LinkedList< T > &l)
- **Iterator** (LinkedListEl< T > *head)
- bool **finished** ()
- Iterator & **operator++** ()
- Iterator **operator++** (int)
- bool **operator==** (const Iterator &iter)
- bool **operator!=** (const Iterator &iter)
- T **operator*** ()

4.8.1 Opis szczegółowy

template<typename T>
class LinkedList< T >::Iterator

Class custom iterator

Dokumentacja dla tej klasy została wygenerowana z pliku:

- kod/utilities/LinkedList.h

4.9 Dokumentacja klasy Jack

```
#include <Jack.h>
```

Diagram dziedziczenia dla Jack

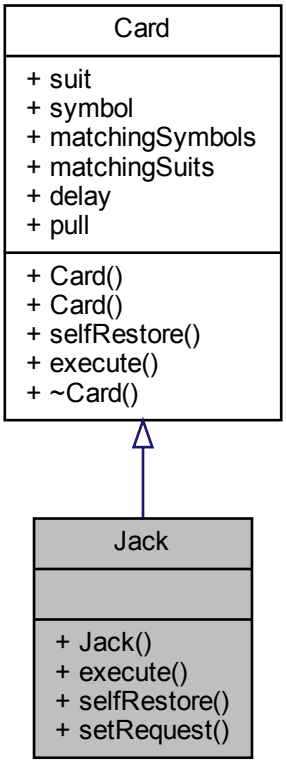
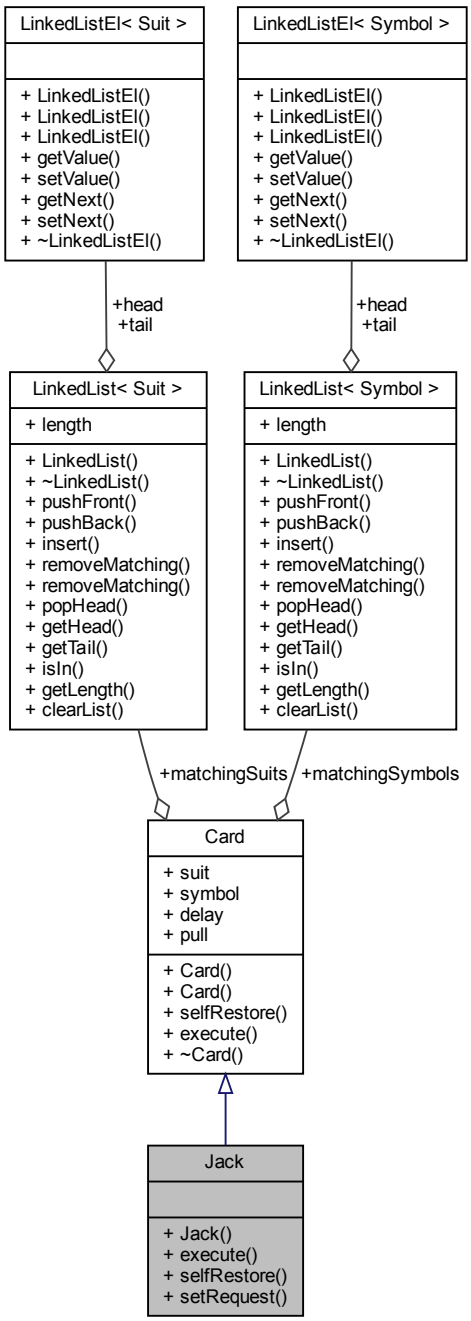


Diagram współpracy dla Jack:



Metody publiczne

- Jack (Suit _suit)
- Punishments execute (Punishments p) override
- void selfRestore () override
- void setRequest (Symbol requestedSymbol)

Dodatkowe Dziedziczone Składowe

4.9.1 Opis szczegółowy

Class representing Jack card

4.9.2 Dokumentacja konstruktora i destruktora

4.9.2.1 Jack()

```
Jack::Jack (
            Suit _suit )  [inline], [explicit]
```

Jack class constructor

Parametry

_suit	Jack's suit

4.9.3 Dokumentacja funkcji składowych

4.9.3.1 execute()

```
Punishments Jack::execute (
            Punishments p )  [inline], [override], [virtual]
```

Execute placeholder

Parametry

p	current punishments

Zwraca

unchanged punishments

Implementuje Card.

4.9.3.2 selfRestore()

```
void Jack::selfRestore ( )  [override], [virtual]
```

Restores the card to its base state

Implementuje Card.

4.9.3.3 setRequest()

```
void Jack::setRequest (
            Symbol requestedSymbol )
```

Sets the symbol request

Parametry

requestedSymbol	Requested symbol

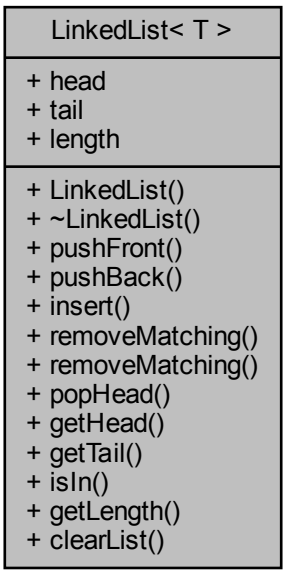
Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/game/Cards/Jack.h
- kod/game/Cards/Jack.cpp

4.10 Dokumentacja szablonu klasy LinkedList< T >

```
#include <LinkedList.h>
```

Diagram współpracy dla LinkedList< T >:



Komponenty

- class Iterator

Metody publiczne

- void pushFront (const T &element)
- void pushBack (const T &element)
- void insert (const T &element, int position)
- int removeMatching (const T &element)
- int removeMatching (const LinkedList &list)
- void popHead ()
- T getHead ()
- T getTail ()
- bool isIn (const T &element) const
- int getLength ()
- int clearList ()

Atrybuty publiczne

- LinkedListEl< T > * head = nullptr

  Head of the list.
- LinkedListEl< T > * tail = nullptr

  Tail of the list.
- int length = 0

  Length of the list.

4.10.1 Opis szczegółowy

template<typename T>
class LinkedList< T >

Template Linked List with extra functions and parameters

Parametry Szablonu

T	Type of the elements

4.10.2 Dokumentacja funkcji składowych

4.10.2.1 clearList()

```
template<typename T >
int LinkedList< T >::clearList ( )
```

Deletes all the list elements

Zwraca

number of deleted elements

4.10.2.2 getHead()

```
template<typename T >
T LinkedList< T >::getHead ( )
```

Returns top element if exists

Zwraca

top element

4.10.2.3 getLength()

```
template<typename T >
int LinkedList< T >::getLength ( )
```

List length getter

Zwraca

list length

4.10.2.4 getTail()

```
template<typename T >
T LinkedList< T >::getTail ( )
```

Returns last element if exists

Zwraca

the last element

4.10.2.5 insert()

```
template<typename T >
void LinkedList< T >::insert (
            const T & element,
            int position )
```

Inserts one element at a given position

Parametry

element	the inserted element
position	position in the list

4.10.2.6 isIn()

```
template<typename T >
bool LinkedList< T >::isIn (
            const T & element ) const
```

Checks if an element is present in the list

Parametry

element	element to check if present

Zwraca

the last element

4.10.2.7 popHead()

```
template<typename T >
void LinkedList< T >::popHead ( )
```

Removes top element if exists

4.10.2.8 pushBack()

```
template<typename T >
void LinkedList< T >::pushBack (
            const T & element )
```

Pushes one element to the back of the list

Parametry

element	

4.10.2.9 pushFront()

```
template<typename T >
void LinkedList< T >::pushFront (
            const T & element )
```

Pushes one element to the front of the list

Parametry

element	

4.10.2.10 removeMatching() [1/2]

```
template<typename T >
int LinkedList< T >::removeMatching (
            const LinkedList< T > & list )
```

Removes all matching elements in the list

Parametry

list	list of the elements to compare

Zwraca

number of deleted elements

4.10.2.11 removeMatching() [2/2]

```
template<typename T >
int LinkedList< T >::removeMatching (
            const T & element )
```

Removes matching elements in the list

Parametry

element	element to compare

Zwraca

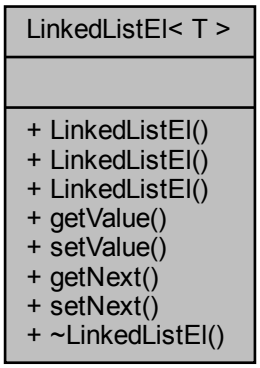
number of deleted elements

Dokumentacja dla tej klasy została wygenerowana z pliku:

- kod/utilities/LinkedList.h

4.11 Dokumentacja szablonu klasy LinkedListEl< T >

Diagram współpracy dla LinkedListEl< T >:



Metody publiczne

- **LinkedListEl** (const LinkedListEl< T > &el)
- **LinkedListEl** (const T val, LinkedListEl< T > *pNext)
- T **getValue** () const
- void **setValue** (T newValue)
- LinkedListEl< T > * **getNext** () const
- void **setNext** (LinkedListEl< T > *newNext)

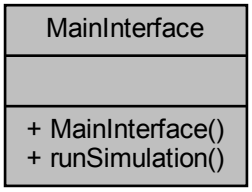
Dokumentacja dla tej klasy została wygenerowana z pliku:

- kod/utilities/LinkedListEl.h

4.12 Dokumentacja klasy MainInterface

```
#include <MainInterface.h>
```

Diagram współpracy dla MainInterface:



Metody publiczne

- MainInterface ()
- void runSimulation ()

4.12.1 Opis szczegółowy

Game wrapper responsible for human interactions and simulation running

4.12.2 Dokumentacja konstruktora i destruktora

4.12.2.1 MainInterface()

```
MainInterface::MainInterface ( )
```

Main interface constructor Handles all the input and prepares for the simulation run

4.12.3 Dokumentacja funkcji składowych

4.12.3.1 runSimulation()

```
void MainInterface::runSimulation ( )
```

Runs the simulation and prints the results

Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/interface/MainInterface.h
- kod/interface/MainInterface.cpp

4.13 Dokumentacja klasy Player

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

Diagram dziedziczenia dla Player

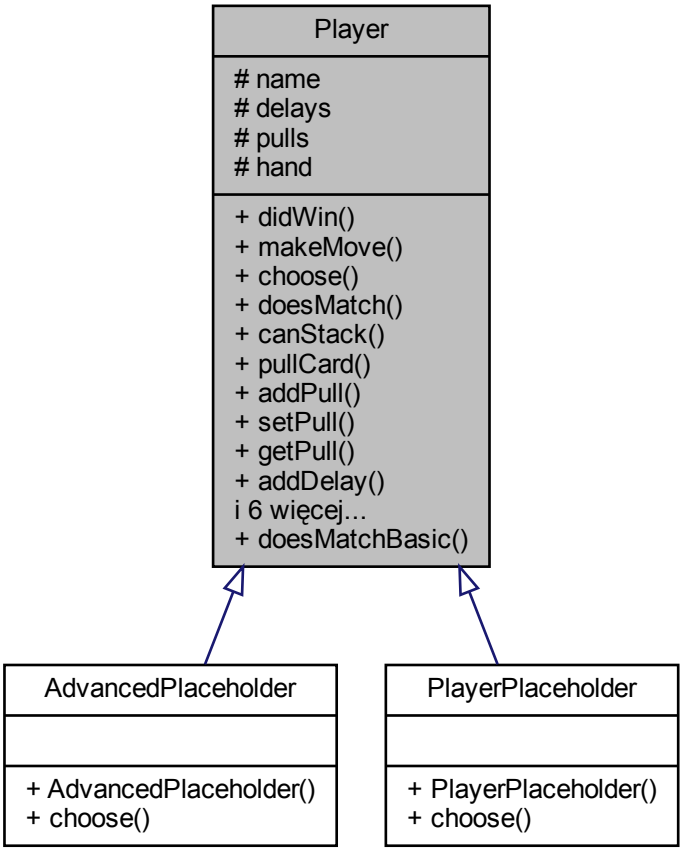
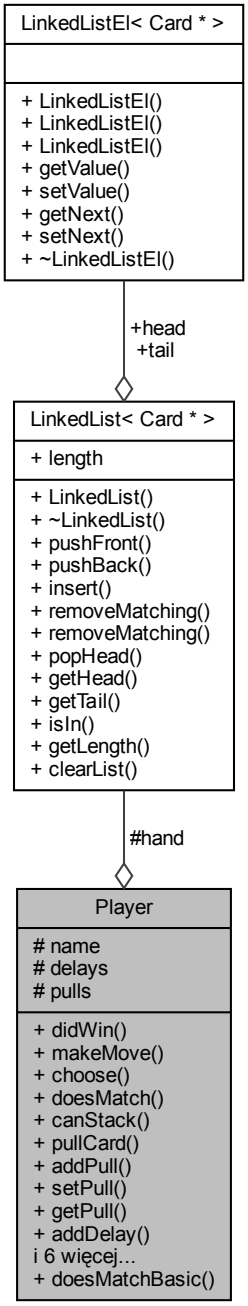


Diagram współpracy dla Player:



Metody publiczne

- bool didWin () const
- void makeMove (LinkedList< Card * > *cardsStack, int playersNumber, int playersHandsLengths[ ], Player *previousPlayer, Player *nextPlayer, Deck &deck)
- virtual LinkedList< Card * > * choose (LinkedList< Card * > *cardsStack, int playersNumber, int players↩ HandsLengths[ ])=0

- bool doesMatch (const Card *laying, const Card *putting) const
- bool canStack (const Card *first, const Card *second)
- void pullCard (Deck &deck)
- void addPull (int)
- void setPull (int)
- int getPull () const
- void addDelay (int)
- void setDelay (int)
- void executeDelay ()
- int getDelay () const
- int getHandLength () const
- std::string getName () const
- ~Player ()

Statyczne metody publiczne

- static bool doesMatchBasic (const Card *laying, const Card *putting)

Atrybuty chronione

- std::string name

  Name of the algorithm.
- int delays = 0

  Current number of delays.
- int pulls = 0

  Current number of pulls.
- LinkedList< Card * > hand = LinkedList<Card*>()

  Cards on hand.

4.13.1 Opis szczegółowy

Base virtual player/algorithm class

4.13.2 Dokumentacja konstruktora i destruktora

4.13.2.1 ~Player()

```
Player::~Player ( )
```

Player Destructor

4.13.3 Dokumentacja funkcji składowych

4.13.3.1 addDelay()

```
void Player::addDelay (
            int additional )
```

Adds given number of delays to player

Parametry

additional	delay

4.13.3.2 addPull()

```
void Player::addPull (
            int additional )
```

Adds given number of pull requests to player

Parametry

additional	pull

4.13.3.3 canStack()

```
bool Player::canStack (
            const Card * first,
            const Card * second )
```

Returns true if two cards can be stacked together

Parametry

first	First card to be stacked
second	Second card to be stacked

Zwraca

4.13.3.4 choose()

```
virtual LinkedList<Card*>* Player::choose (
            LinkedList< Card * > * cardsStack,
            int playersNumber,
            int playersHandsLengths[] )  [pure virtual]
```

Overridable method for choosing the cards. Cards shold be layed out in such order that the first card touching the card stack should lay at the head of the list. The card in the tail will lay at the top of the card stack after move. Also remember to allocate the memory for the cards stack. If making empty move return pointer to the allocated, but empty list

Parametry

cardsStack	Stack of cards - the top card is in the head
playersNumber	number of players
playersHandsLengths	array of players lengths - at position 0 is the current player

Zwraca

list of chosen cards

4.13.3.5 didWin()

```
bool Player::didWin ( ) const
```

Checks if player won

Zwraca

true if player won

4.13.3.6 doesMatch()

```
bool Player::doesMatch (
            const Card * laying,
            const Card * putting ) const
```

Checks if a card can be placed on a card stack It takes into consideration the user state

Parametry

laying	Card laying on the top of the stack
putting	Card willing to be put on the stack

Zwraca

true if card can be placed

4.13.3.7 doesMatchBasic()

```
bool Player::doesMatchBasic (
            const Card * laying,
            const Card * putting )  [static]
```

Checks if a card could be put on the table if no punishments were put on the player

Parametry

laying	Card laying on the table
putting	Card which is tested

Zwraca

true if card can be placed

4.13.3.8 executeDelay()

```
void Player::executeDelay ( )
```

Executes one delay

4.13.3.9 getDelay()

```
int Player::getDelay ( ) const
```

Delay number getter

4.13.3.10 getHandLength()

```
int Player::getHandLength ( ) const
```

Gets the length of a hand

4.13.3.11 getName()

```
std::string Player::getName ( ) const
```

Gets the algorithm name

4.13.3.12 getPull()

```
int Player::getPull ( ) const
```

Pull value getter

4.13.3.13 makeMove()

```
void Player::makeMove (
            LinkedList< Card * > * cardsStack,
            int playersNumber,
            int playersHandsLengths[],
            Player * previousPlayer,
            Player * nextPlayer,
            Deck & deck )
```

Handles full player move - from choice to execution

Parametry

cardsStack	///< Cards stacked on the table
playersNumber	///< Number of players in the game
playersHandsLengths	///< Players hands lengths - at position 0 is the current player
previousPlayer	///< Pointer to the previous player
nextPlayer	///< Pointer to the next player
deck	///< Reference to the deck on the table

4.13.3.14 pullCard()

```
void Player::pullCard (
            Deck & deck )
```

Pulls one card from deck

Parametry

deck	reference to the deck

4.13.3.15 setDelay()

```
void Player::setDelay (
            int newDelays )
```

Sets the delays to given

Parametry

delay	

4.13.3.16 setPull()

```
void Player::setPull (
            int newPulls )
```

Sets pull to given

Parametry

pull	to be set

Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/players/Player.h
- kod/players/Player.cpp

4.14 Dokumentacja klasy PlayerPlaceholder

```
#include <PlayerPlaceholder.h>
```

Diagram dziedziczenia dla PlayerPlaceholder

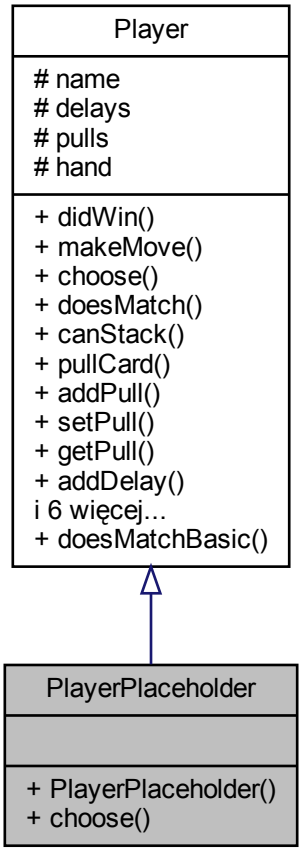
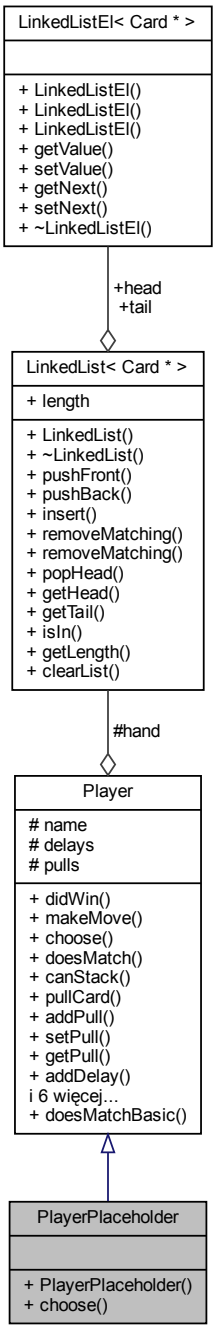


Diagram współpracy dla PlayerPlaceholder:



Metody publiczne

- LinkedList< Card * > * **choose** (LinkedList< Card * > *cardsStack, int playersNumber, int *players↩ HandsLengths) override

Dodatkowe Dziedziczone Składowe

4.14.1 Opis szczegółowy

Basic algorithm placeholder

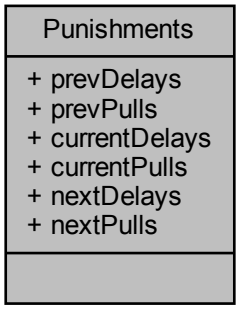
Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/players/algorithms/PlayerPlaceholder.h
- kod/players/algorithms/PlayerPlaceholder.cpp

4.15 Dokumentacja struktury Punishments

`#include <Punishments.h>`

Diagram współpracy dla Punishments:



Atrybuty publiczne

- int prevDelays
  delays of the previous player
- int prevPulls
  pulls of the previous player
- int currentDelays
  delays of the current player
- int currentPulls
  pulls of the current player
- int nextDelays
  delays of the next player
- int nextPulls
  pulls of the next player

4.15.1 Opis szczegółowy

Structure representing the punishments of players

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/game/Punishments.h

4.16 Dokumentacja klasy QueenOfSpades

```
#include <QueenOfSpades.h>
```

Diagram dziedziczenia dla QueenOfSpades

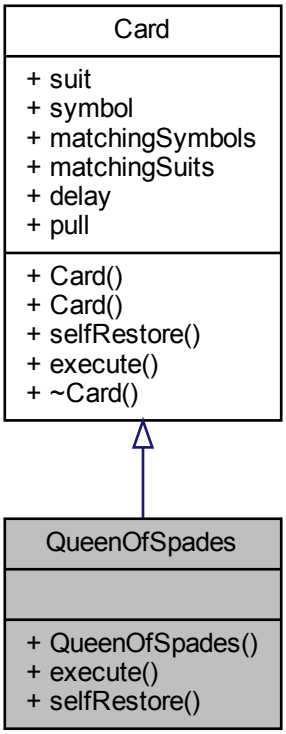
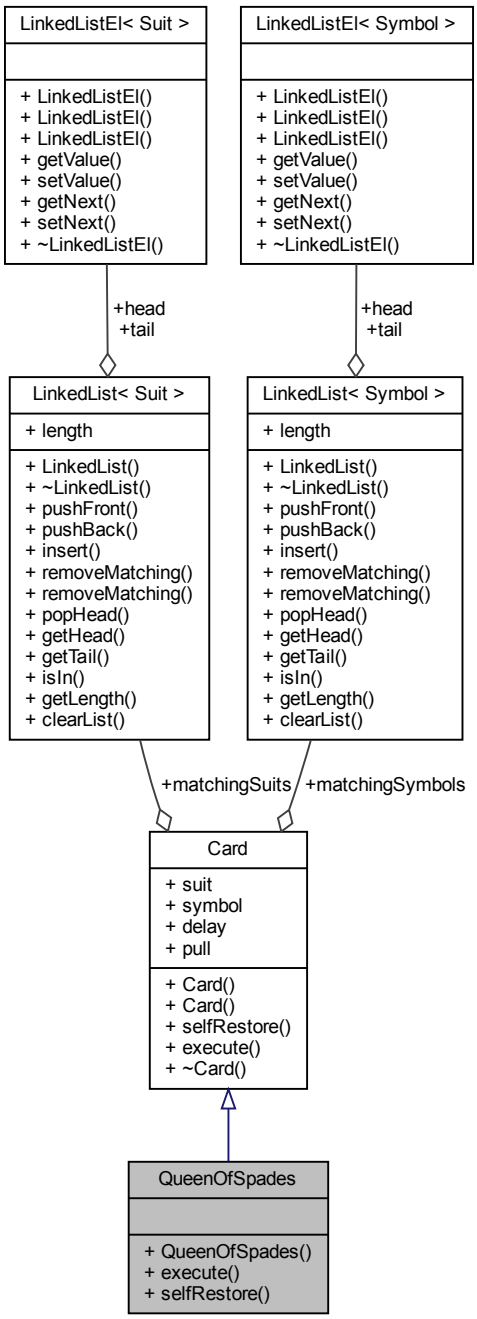


Diagram współpracy dla QueenOfSpades:



Metody publiczne

- Punishments execute (Punishments) override
- void selfRestore () override

Dodatkowe Dziedziczone Składowe

4.16.1 Opis szczegółowy

Class representing the queen of spades card

4.16.2 Dokumentacja funkcji składowych

4.16.2.1 execute()

```
Punishments QueenOfSpades::execute (
            Punishments punishments )  [override], [virtual]
```

Executes the card function Frees the card placer of any pulls or delays

Zwraca

new punishments

Implementuje Card.

4.16.2.2 selfRestore()

```
void QueenOfSpades::selfRestore ( )  [inline], [override], [virtual]
```

Changes the card to its base state In this case does nothing

Implementuje Card.

Dokumentacja dla tej klasy została wygenerowana z plików:

- kod/game/Cards/QueenOfSpades.h
- kod/game/Cards/QueenOfSpades.cpp

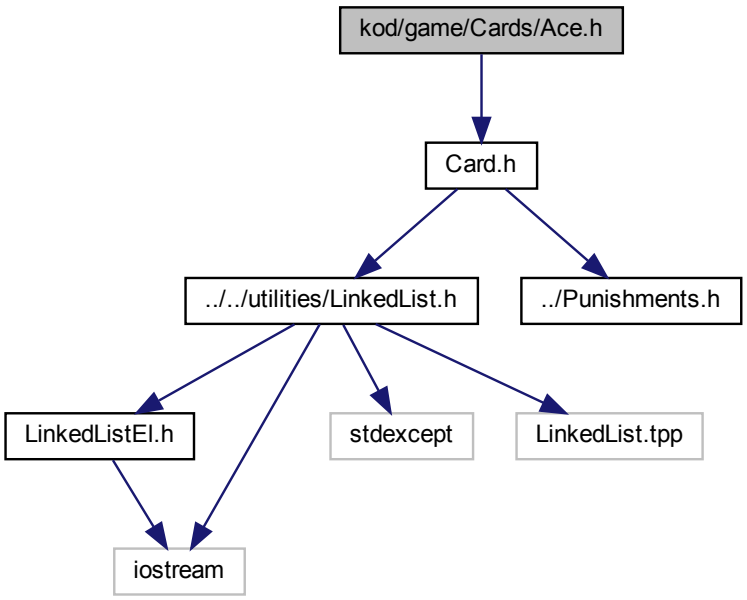
Rozdział 5

Dokumentacja plików

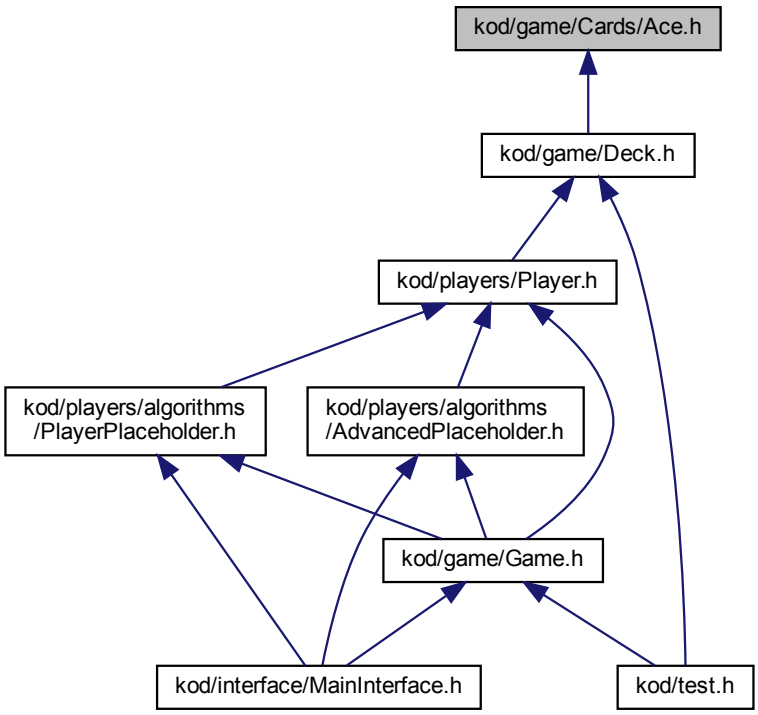
5.1 Dokumentacja pliku kod/game/Cards/Ace.h

```
#include "Card.h"
```

Wykres zależności załączania dla Ace.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



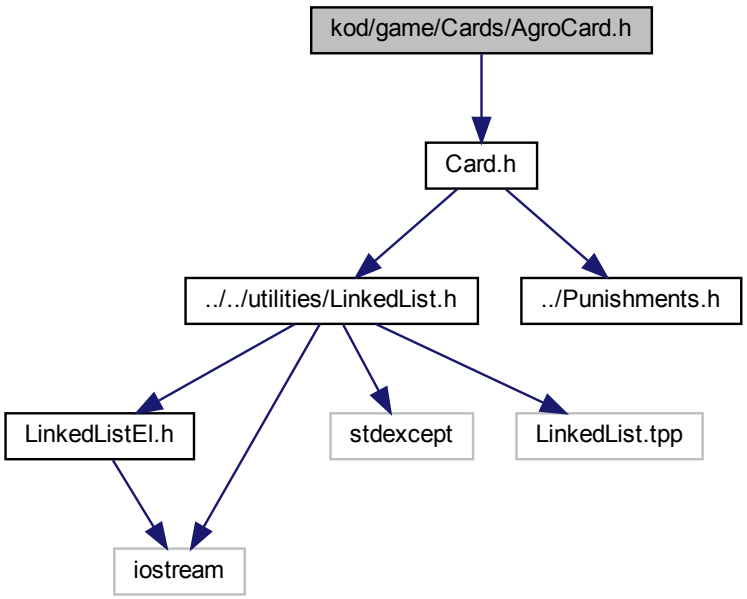
Komponenty

- class Ace

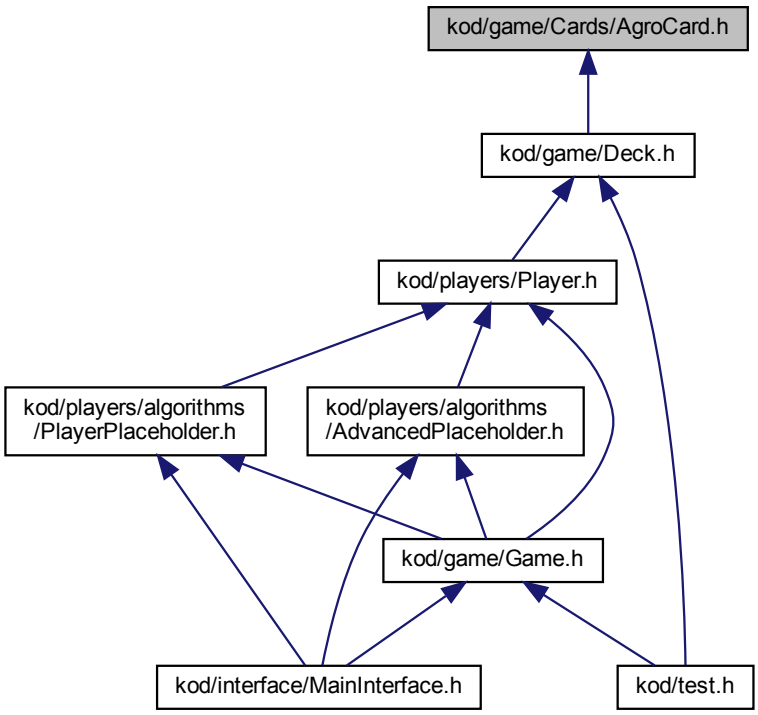
5.2 Dokumentacja pliku kod/game/Cards/AgroCard.h

```
#include "Card.h"
```

Wykres zależności załączania dla AgroCard.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



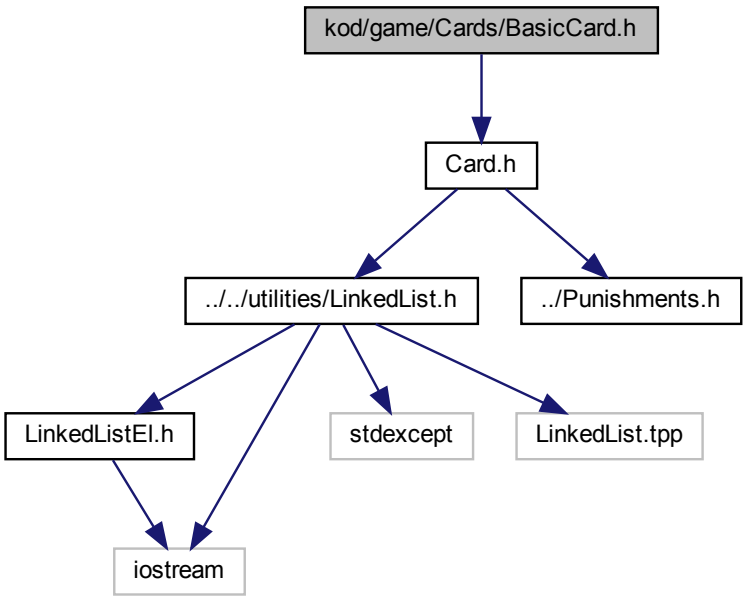
Komponenty

- class AgroCard

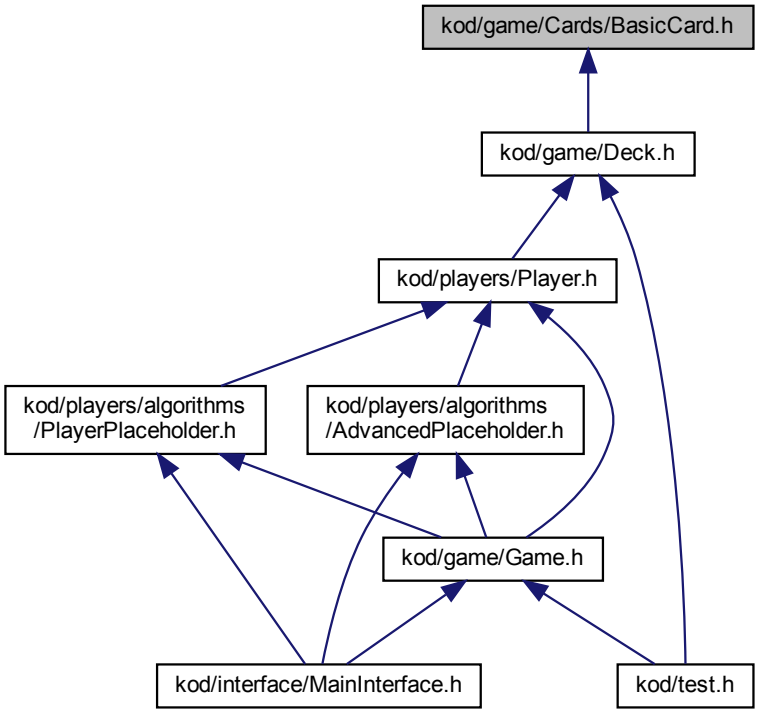
5.3 Dokumentacja pliku kod/game/Cards/BasicCard.h

```
#include "Card.h"
```

Wykres zależności załączania dla BasicCard.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



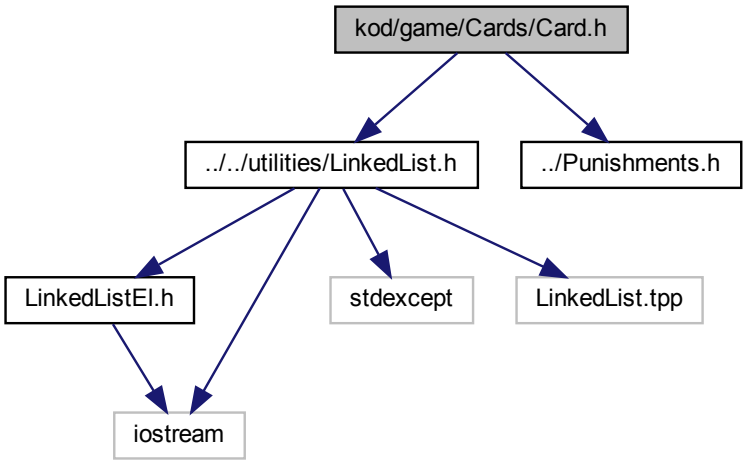
Komponenty

- class BasicCard

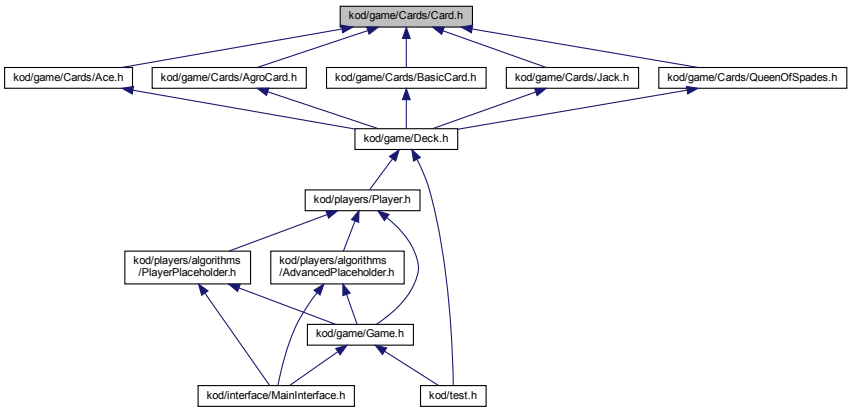
5.4 Dokumentacja pliku kod/game/Cards/Card.h

```
#include "../../utilities/LinkedList.h"
#include "../Punishments.h"
```

Wykres zależności załączania dla Card.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Card

Wyliczenia

- enum Symbol {
  ace, **two**, **three**, **four**,
  five, **six**, **seven**, **eight**,
  nine, **ten**, **jack**, **queen**,
  king }
- enum Suit { **clubs**, **hearts**, **spades**, **diamonds** }

5.4.1 Dokumentacja typów wyliczanych

5.4.1.1 Suit

```
enum Suit
```

All the suits available in the game

5.4.1.2 Symbol

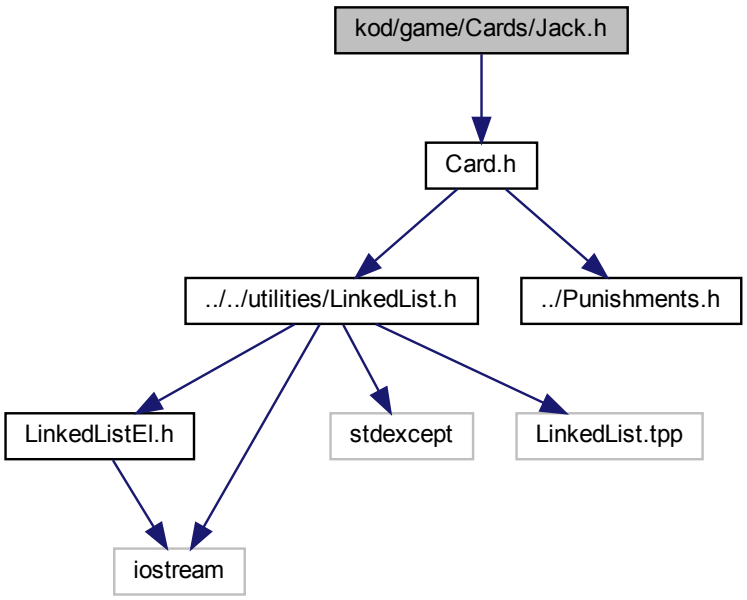
```
enum Symbol
```

All the Symbols avaliable in the simulator

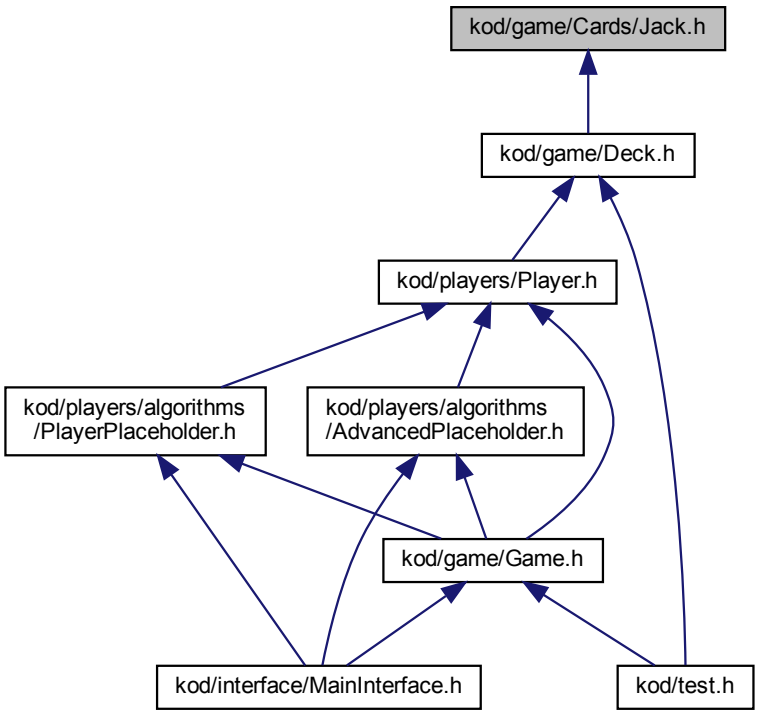
5.5 Dokumentacja pliku kod/game/Cards/Jack.h

```
#include "Card.h"
```

Wykres zależności załączania dla Jack.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



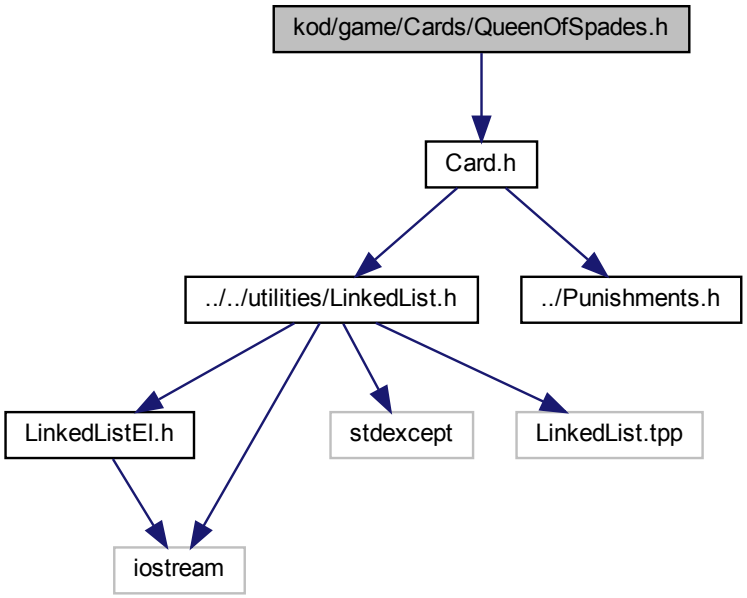
Komponenty

- class Jack

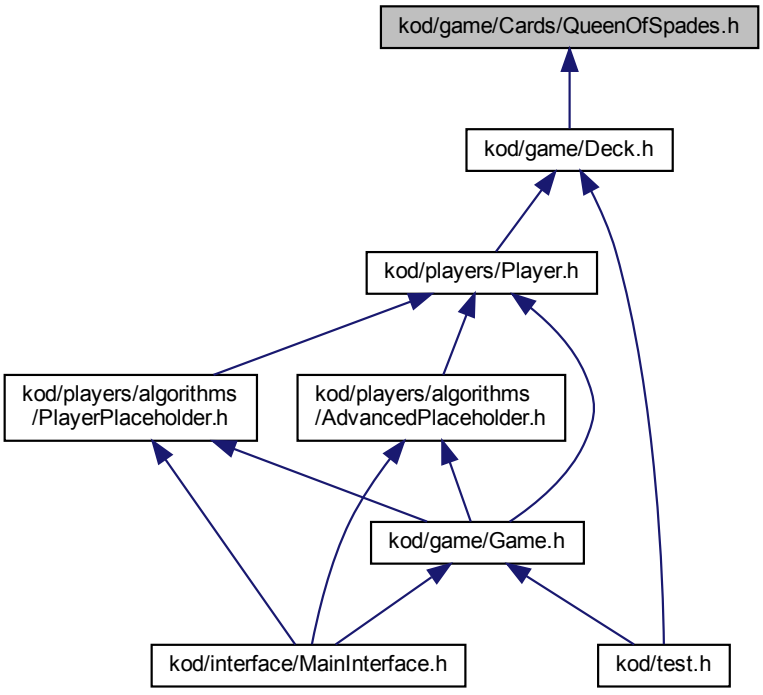
5.6 Dokumentacja pliku kod/game/Cards/QueenOfSpades.h

```
#include "Card.h"
```

Wykres zależności załączania dla QueenOfSpades.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



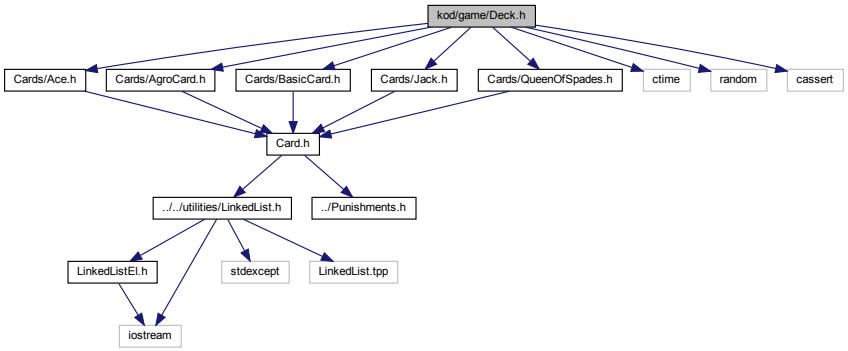
Komponenty

- class QueenOfSpades

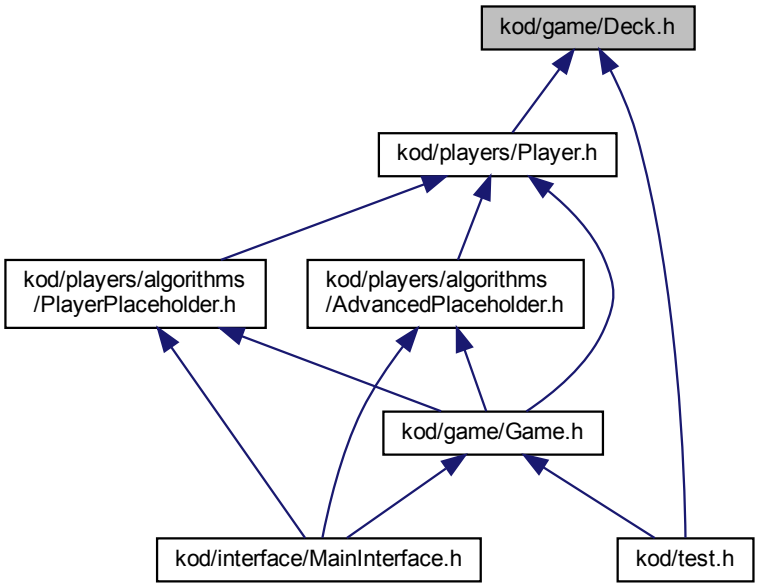
5.7 Dokumentacja pliku kod/game/Deck.h

```
#include "Cards/Ace.h"
#include "Cards/AgroCard.h"
#include "Cards/BasicCard.h"
#include "Cards/Jack.h"
#include "Cards/QueenOfSpades.h"
#include <ctime>
#include <random>
#include <cassert>
```

Wykres zależności załączania dla Deck.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



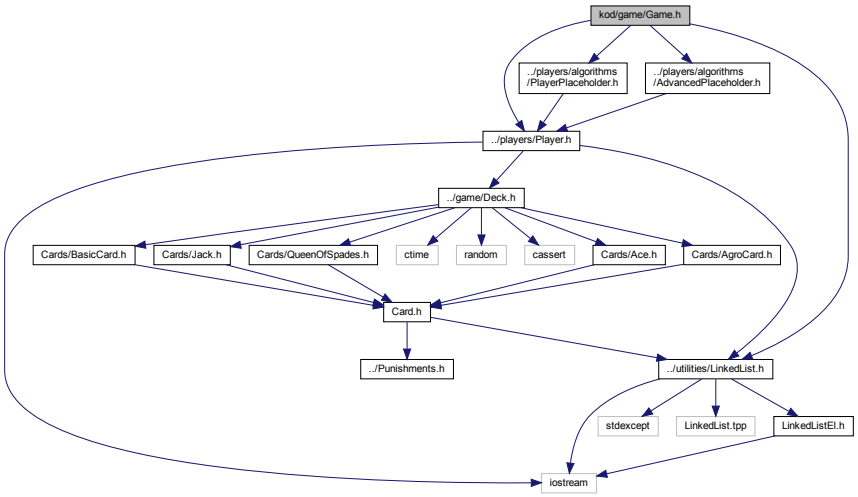
Komponenty

- class Deck

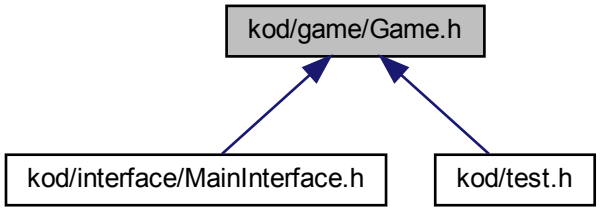
5.8 Dokumentacja pliku kod/game/Game.h

```
#include "../players/Player.h"
#include "../utilities/LinkedList.h"
#include "../players/algorithms/PlayerPlaceholder.h"
#include "../players/algorithms/AdvancedPlaceholder.h"
```

Wykres zależności załączania dla Game.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



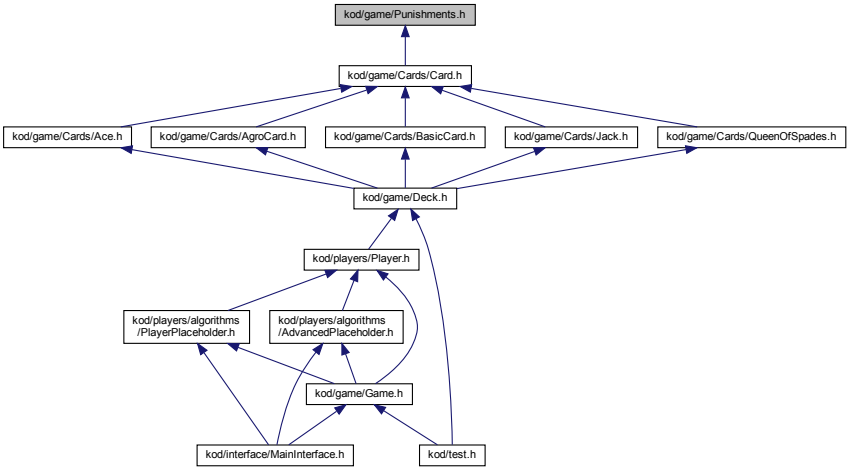
Komponenty

- class Game

  ALGORITHMS.

5.9 Dokumentacja pliku kod/game/Punishments.h

Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



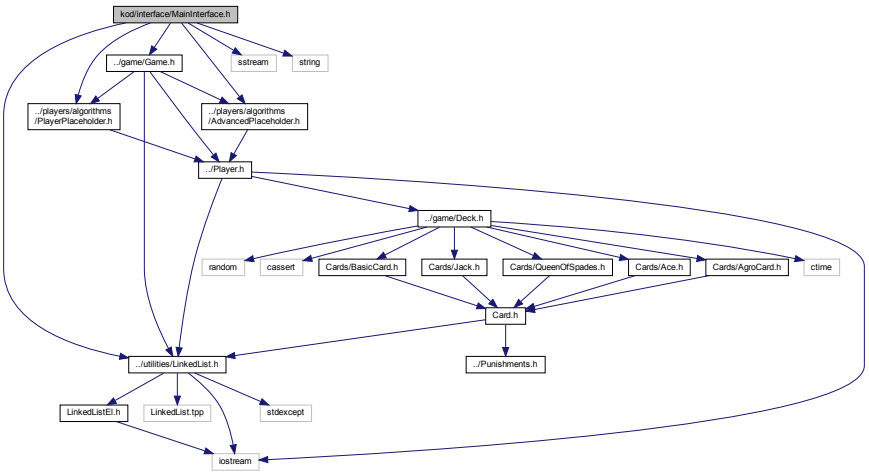
Komponenty

- struct Punishments

5.10 Dokumentacja pliku kod/interface/MainInterface.h

```
#include "../utilities/LinkedList.h"
#include "../players/algorithms/PlayerPlaceholder.h"
#include "../players/algorithms/AdvancedPlaceholder.h"
#include "../game/Game.h"
#include <sstream>
#include <string>
```

Wykres zależności załączania dla MainInterface.h:



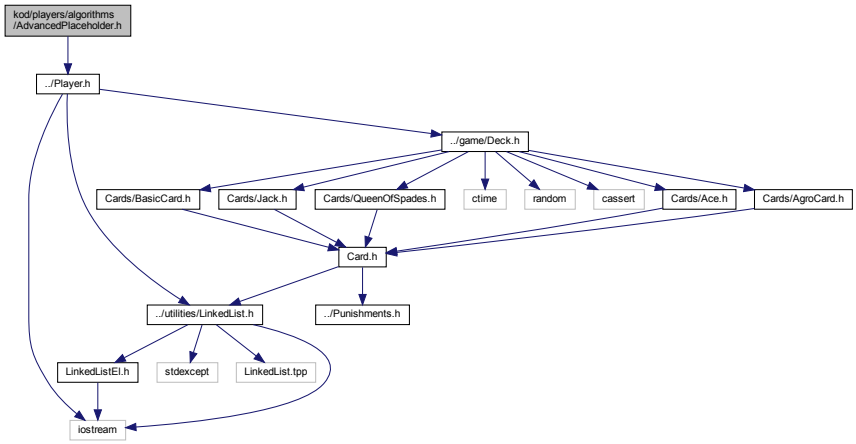
Komponenty

- class MainInterface

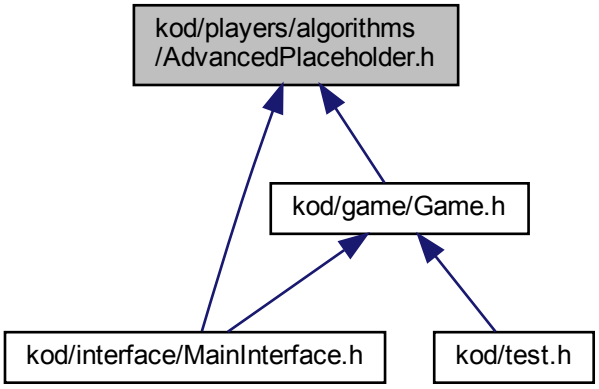
5.11 Dokumentacja pliku kod/players/algorithms/AdvancedPlaceholder.h

```
#include "../Player.h"
```

Wykres zależności załączania dla AdvancedPlaceholder.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



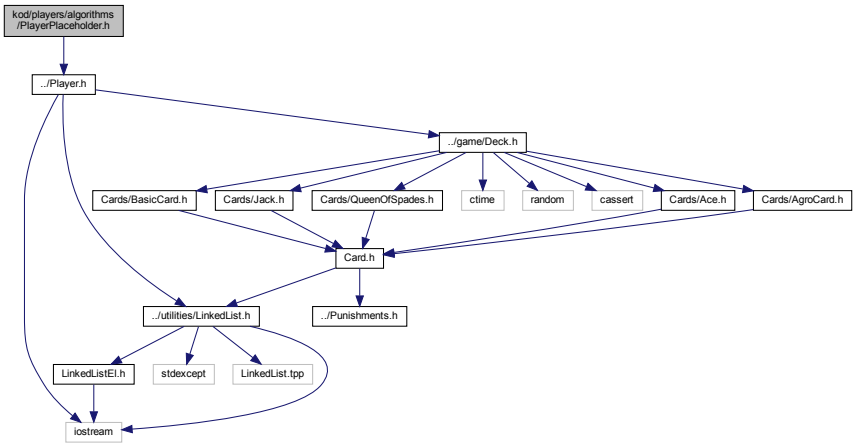
Komponenty

- class AdvancedPlaceholder

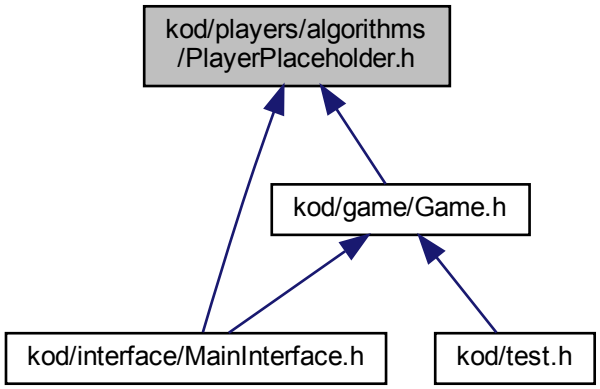
5.12 Dokumentacja pliku kod/players/algorithms/PlayerPlaceholder.h

```
#include "../Player.h"
```

Wykres zależności załączania dla PlayerPlaceholder.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

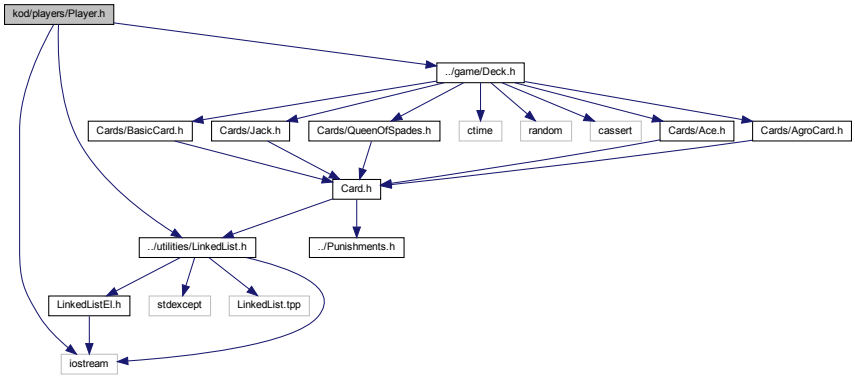
- class PlayerPlaceholder

5.13 Dokumentacja pliku kod/players/Player.h

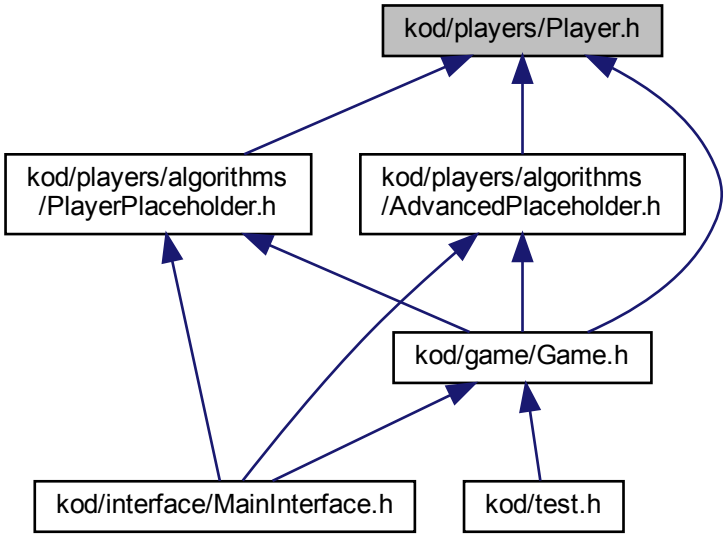
```
#include "../utilities/LinkedList.h"
#include "../game/Deck.h"
```

```
#include <iostream>
```

Wykres zależności załączania dla Player.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

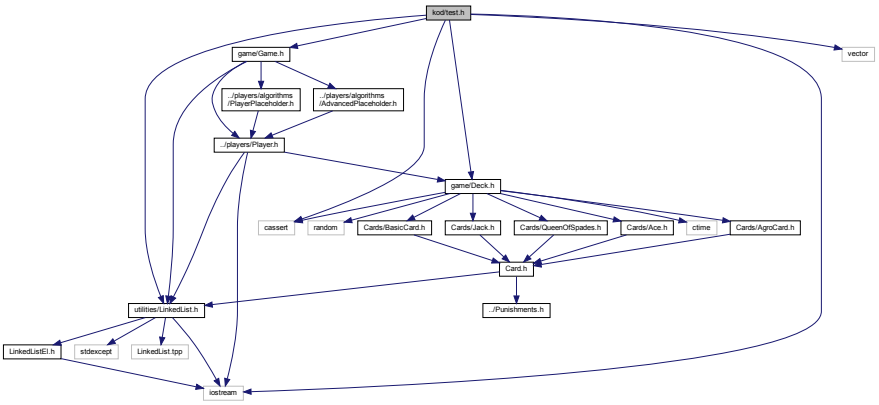
- class Player

5.14 Dokumentacja pliku kod/test.h

```
#include <cassert>
#include <iostream>
```

```
#include "utilities/LinkedList.h"
#include "game/Deck.h"
#include "game/Game.h"
#include <vector>
```

Wykres zależności załączania dla test.h:



Funkcje

- void testLinkedList ()
- void buildDeck ()
- void buildGame ()
- void unitTest ()

5.14.1 Dokumentacja funkcji

5.14.1.1 buildDeck()

```
void buildDeck ( )
```

Ensure Deck builds with no issues

5.14.1.2 buildGame()

```
void buildGame ( )
```

Ensure Game builds with no issues and terminates due to limit

5.14.1.3 testLinkedList()

```
void testLinkedList ( )
```

Ensures Linked List works correctly

5.14.1.4 unitTest()

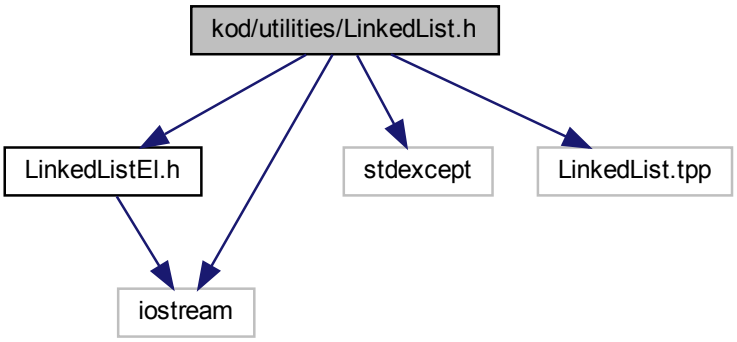
```
void unitTest ( )
```

Ensures all functions work correctly

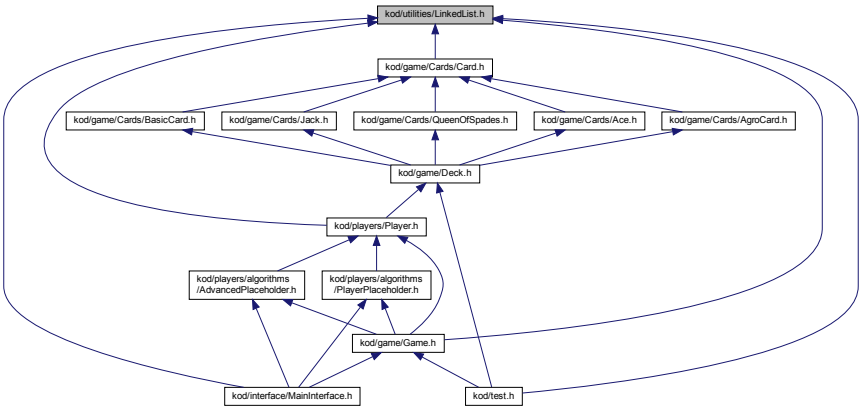
5.15 Dokumentacja pliku kod/utilities/LinkedList.h

```
#include "LinkedListEl.h"
#include <iostream>
#include <stdexcept>
#include "LinkedList.tpp"
```

Wykres zależności załączania dla LinkedList.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



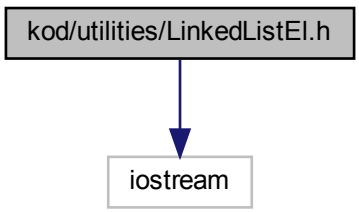
Komponenty

- class LinkedList< T >
- class LinkedList< T >::Iterator

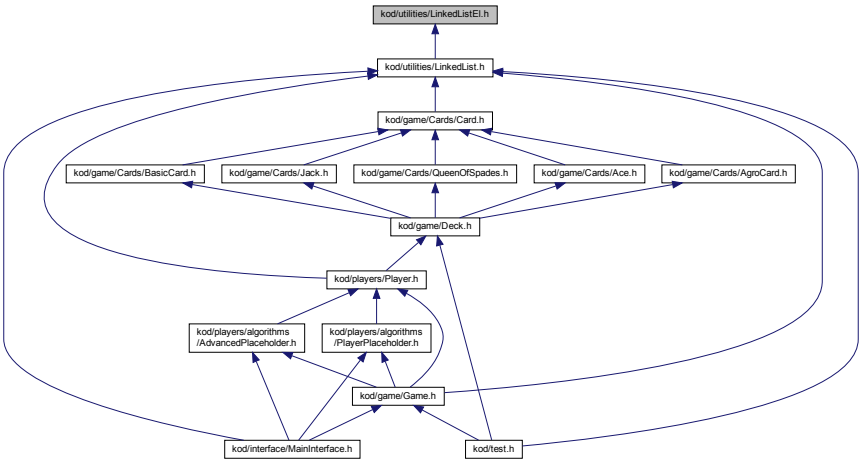
5.16 Dokumentacja pliku kod/utilities/LinkedListEl.h

```
#include <iostream>
```

Wykres zależności załączania dla LinkedListEl.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class LinkedListEl< T >

Indeks