My Project

Generated by Doxygen 1.9.8

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

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

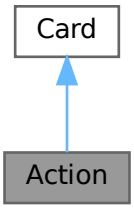
Class Documentation

4.1 Action Class Reference

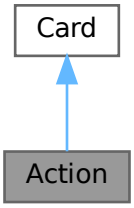
The Action class represents an action card in the game. This class inherits from the Card class and adds an ActionType attribute. It provides methods to get the action type and the index of the card.

`#include <Action.h>`

Inheritance diagram for Action:



Collaboration diagram for Action:



Public Member Functions

- Action (ActionType actionType)

  Constructs an Action object with the specified action type.
- ActionType getActionType () const

  Gets the action type of the card.
- int getIndex () override

  Gets the index of the card.

Public Member Functions inherited from Card

- Card (CardType type)

  Constructs a Card object with the specified type.
- CardType getType () const

  Gets the type of the card.
- virtual ~**Card** ()

  Destroys the Card object.

4.1.1 Detailed Description

The Action class represents an action card in the game. This class inherits from the Card class and adds an ActionType attribute. It provides methods to get the action type and the index of the card.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Action()

```
Action::Action (
            ActionType actionType )
```

Constructs an Action object with the specified action type.

Parameters

actionType	The type of the action.

4.1.3 Member Function Documentation

4.1.3.1 getActionType()

```
ActionType Action::getActionType ( ) const
```

Gets the action type of the card.

Returns

The action type of the card.

4.1.3.2 getIndex()

```
int Action::getIndex ( )  [override], [virtual]
```

Gets the index of the card.

Returns

The index of the card.

Reimplemented from Card.

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

- src/Card/Action/Action.h
- src/Card/Action/Action.cpp

4.2 Button Class Reference

Represents a graphical button element.

```
#include <Button.h>
```

Public Member Functions

- Button (sf::Vector2f position, sf::Vector2f size, sf::Font ∗font, std::string text, int fontSize)
  Constructs a Button object.
- sf::RectangleShape getShape () const
  Gets the shape of the button.
- bool isHovered (sf::RenderWindow &window) const
  Checks if the button is being hovered over.
- bool isClicked (sf::RenderWindow &window) const
  Checks if the button is being clicked.
- void draw (sf::RenderWindow &window)
  Draws the button on the specified render window.

4.2.1 Detailed Description

Represents a graphical button element.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Button()

```
Button::Button (
            sf::Vector2f position,
            sf::Vector2f size,
            sf::Font * font,
            std::string text,
            int fontSize )
```

Constructs a Button object.

Parameters

position	The position of the button.
size	The size of the button.
font	A pointer to the font used for the button text.
text	The text displayed on the button.
fontSize	The font size of the button text.

4.2.3 Member Function Documentation

4.2.3.1 draw()

```
void Button::draw (
            sf::RenderWindow & window )
```

Draws the button on the specified render window.

Parameters

window	The render window to draw on.

4.2.3.2 getShape()

```
sf::RectangleShape Button::getShape ( ) const
```

Gets the shape of the button.

Returns

The shape of the button.

4.2.3.3 isClicked()

```
bool Button::isClicked (
            sf::RenderWindow & window ) const
```

Checks if the button is being clicked.

Parameters

window	The render window to check against.

Returns

True if the button is being clicked, false otherwise.

4.2.3.4 isHovered()

```
bool Button::isHovered (
            sf::RenderWindow & window ) const
```

Checks if the button is being hovered over.

Parameters

window	The render window to check against.

Returns

True if the button is being hovered over, false otherwise.

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

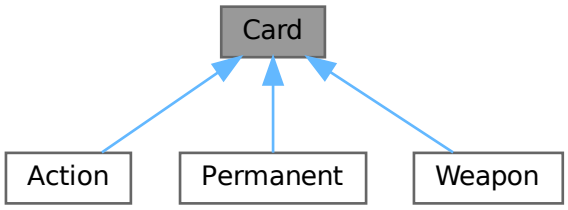
- src/UI/Button/Button.h
- src/UI/Button/Button.cpp

4.3 Card Class Reference

The Card class represents a card in the game.

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

Inheritance diagram for Card:



Public Member Functions

- Card (CardType type)

  Constructs a Card object with the specified type.
- CardType getType () const

  Gets the type of the card.
- virtual int getIndex ()

  Gets the index of the card.
- virtual ~**Card** ()

  Destroys the Card object.

4.3.1 Detailed Description

The Card class represents a card in the game.

This class provides the basic functionality of a card, such as the type of the card.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Card()

```
Card::Card (
            CardType type )
```

Constructs a Card object with the specified type.

Parameters

type	The type of the card.

Note

This constructor is used by the derived classes to set the type of the card.

4.3.3 Member Function Documentation

4.3.3.1 getIndex()

```
int Card::getIndex ( )  [virtual]
```

Gets the index of the card.

Returns

The index of the card.

Reimplemented in Action, Permanent, and Weapon.

4.3.3.2 getType()

```
CardType Card::getType ( ) const
```

Gets the type of the card.

Returns

The type of the card.

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

- src/Card/Card.h
- src/Card/Card.cpp

4.4 Character Class Reference

The Character class represents a character in the game.

```
#include <Character.h>
```

Public Member Functions

- Character (CharacterType type, int HP)
  Constructs a new Character object.
- CharacterType getType () const
  Gets the type of the character.
- int getIndex () const
  Gets the index of the character.
- int getHP () const
  Gets the hit points of the character.
- std::string getName () const
  Gets the name of the character.

4.4.1 Detailed Description

The Character class represents a character in the game.

This class stores information about the character's type, HP (hit points), and provides methods to access and modify these attributes.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Character()

```
Character::Character (
            CharacterType type,
            int HP )
```

Constructs a new Character object.

Parameters

type	The type of the character.
HP	The initial hit points of the character.

4.4.3 Member Function Documentation

4.4.3.1 getHP()

```
int Character::getHP ( ) const
```

Gets the hit points of the character.

Returns

The character's hit points.

4.4.3.2 getIndex()

```
int Character::getIndex ( ) const
```

Gets the index of the character.

Returns

The character index.

4.4.3.3 getName()

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

Gets the name of the character.

Returns

The character's name.

4.4.3.4 getType()

```
CharacterType Character::getType ( ) const
```

Gets the type of the character.

Returns

The character type.

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

- src/Character/Character.h
- src/Character/Character.cpp

4.5 Game Class Reference

Represents the game state and logic.

```
#include <Game.h>
```

Public Member Functions

- **Game** ()

  Constructs a new Game object.
- void **initRole** ()

  Initializes the roles in the game.
- void **initCharacter** ()

  Initializes the characters in the game.
- void **initCard** ()

  Initializes the cards in the game.
- void **initPlayer** ()

  Initializes the players in the game.
- void setNbPlayers (int nbPlayers)

  set the number of players in the game.
- int getNbPlayers () const

  get the number of players in the game.
- int getIndexActualPlayer () const

  get the index of the actual player.
- int getIndexPlayerAttacked () const

  get the index of the player attacked.
- std::vector< Player * > * getPlayers () const

  get the players in the game.
- std::vector< Card * > * getCards () const

  get the cards in the stack
- std::vector< Card * > * getDiscards () const

  get the discards in the game.
- std::vector< std::string * > * getLogs () const

  get the logs of the game.
- void **updateHonorPointsHP** ()

  update the honor points and health points of the players.
- void **recoverHP** ()

  recover the players' health points.
- void pick (Player *player, int nbCard)

  pick a card from the stack.
- bool attack (Weapon *card, Player *player)

  attack the player with a weapon card
- int calculateDistance (Player *playerTarget)

  calculate the distance between the actual player and the target player.
- bool canBlock (Player *player)

  if the player can block the attack.
- void discard (Player *player, Card *card)

  discard a card from the player's hand.
- void **recoverCards** ()

  take the cards from the discard stack.
- void **changePlayer** ()

  change the actual player.
- void **criDeGuerreFunction** ()

  action of cri de guerre action card.
- void **daimyoFunction** ()

  action of the daimyo action card.
- void geishaFunction (Player *player=nullptr)

action of the geisha action card.

- void **ceremonieDuTheFunction** ()

  action of the ceremonie du the action card.

- void **juJitsuFunction** ()

  action of the ju jitsu action card.

- void **codeDuBushidoFunction** ()

  action of the code du bushido permanent card.

- bool isGameOver ()

  verification of the end of the game.

- ~**Game** ()

  destructs a Game object.

Public Attributes

- bool isCarteDuBushidoInGame

4.5.1 Detailed Description

Represents the game state and logic.

The Game class manages the game state and provides methods for initializing the game, performing game actions, and checking game conditions.

4.5.2 Member Function Documentation

4.5.2.1 attack()

```
bool Game::attack (
            Weapon * card,
            Player * player )
```

attack the player with a weapon card

Parameters

card	The weapon card used to attack.
player	The player attacked.

Returns

True if the attack is successful, false otherwise.

4.5.2.2 calculateDistance()

```
int Game::calculateDistance (
            Player * playerTarget )
```

calculate the distance between the actual player and the target player.

Parameters

playerTarget	The target player.

Returns

The distance between the actual player and the target player.

4.5.2.3 canBlock()

```
bool Game::canBlock (
            Player * player )
```

if the player can block the attack.

Parameters

player	The player who can block the attack.

Returns

True if the player can block the attack, false otherwise.

4.5.2.4 discard()

```
void Game::discard (
            Player * player,
            Card * card )
```

discard a card from the player's hand.

Parameters

player	The player who discards the card.
card	The card to discard.

4.5.2.5 geishaFunction()

```
void Game::geishaFunction (
            Player * player = nullptr )
```

action of the geisha action card.

Parameters

player	The player who uses the geisha card.

4.5.2.6 getCards()

```
std::vector< Card * > * Game::getCards ( ) const
```

get the cards in the stack

Returns

The list of cards.

4.5.2.7 getDiscards()

```
std::vector< Card * > * Game::getDiscards ( ) const
```

get the discards in the game.

Returns

The list of discarded cards.

4.5.2.8 getIndexActualPlayer()

```
int Game::getIndexActualPlayer ( ) const
```

get the index of the actual player.

Returns

The index of the actual player.

4.5.2.9 getIndexPlayerAttacked()

```
int Game::getIndexPlayerAttacked ( ) const
```

get the index of the player attacked.

Returns

The index of the player attacked.

4.5.2.10 getLogs()

```
std::vector< std::string * > * Game::getLogs ( ) const
```

get the logs of the game.

Returns

The list of logs.

4.5.2.11 getNbPlayers()

```
int Game::getNbPlayers ( ) const
```

get the number of players in the game.

Returns

The number of players.

4.5.2.12 getPlayers()

```
std::vector< Player * > * Game::getPlayers ( ) const
```

get the players in the game.

Returns

The list of players.

4.5.2.13 isGameOver()

```
bool Game::isGameOver ( )
```

verification of the end of the game.

Returns

True if the game is over, false otherwise.

4.5.2.14 pick()

```
void Game::pick (
            Player * player,
            int nbCard )
```

pick a card from the stack.

Parameters

player	The player who picks the card.
nbCard	The number of cards to pick.

4.5.2.15 setNbPlayers()

```
void Game::setNbPlayers (
```

```
            int nbPlayers )
```

set the number of players in the game.

Parameters

nbPlayers	The number of players.

4.5.3 Member Data Documentation

4.5.3.1 isCarteDuBushidoInGame

```
bool Game::isCarteDuBushidoInGame
```

Boolean to know if the Carte du Bushido is in the game

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

- src/Game/Game.h
- src/Game/Game.cpp

4.6 Menu Class Reference

Represents a menu in the game.

```
#include <Menu.h>
```

Public Member Functions

- Menu (sf::Font *font, sf::Image *leftImage, sf::Image *rightImage)
  Constructs a new Menu object.
- int getNbPlayers () const
  Get the number of players selected.
- void display (sf::RenderWindow &window)
  Display the menu on a given window.
- ~**Menu** ()
  Destructor for the Menu object.

4.6.1 Detailed Description

Represents a menu in the game.

The Menu class is responsible for displaying a menu with buttons and images. It allows the user to interact with the menu and retrieve the number of players selected.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Menu()

```
Menu::Menu (
            sf::Font * font,
            sf::Image * leftImage,
            sf::Image * rightImage )
```

Constructs a new Menu object.

Parameters

font	The font used for the title and buttons.
leftImage	The image for the left side of the menu.
rightImage	The image for the right side of the menu.

4.6.3 Member Function Documentation

4.6.3.1 display()

```
void Menu::display (
            sf::RenderWindow & window )
```

Display the menu on a given window.

Parameters

window	The window to display the menu on.

4.6.3.2 getNbPlayers()

```
int Menu::getNbPlayers ( ) const
```

Get the number of players selected.

Returns

int The number of players.

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

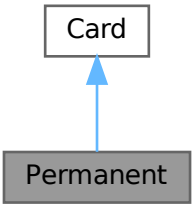
- src/UI/Menu/Menu.h
- src/UI/Menu/Menu.cpp

4.7 Permanent Class Reference

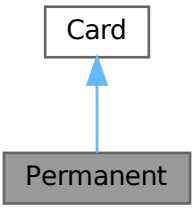
The Permanent class represents a permanent card in the game. This class inherits from the Card class and adds functionality specific to permanent cards.

```
#include <Permanent.h>
```

Inheritance diagram for Permanent:



Collaboration diagram for Permanent:



Public Member Functions

- Permanent (PermanentType permanentType)

  Constructs a new Permanent object with the specified permanent type.
- PermanentType getPermanentType () const

  Gets the type of the permanent card.
- int getIndex () override

  Gets the index of the card.

Public Member Functions inherited from Card

- Card (CardType type)

  Constructs a Card object with the specified type.
- CardType getType () const

  Gets the type of the card.
- virtual ~**Card** ()

  Destroys the Card object.

4.7.1 Detailed Description

The Permanent class represents a permanent card in the game. This class inherits from the Card class and adds functionality specific to permanent cards.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Permanent()

```
Permanent::Permanent (
            PermanentType permanentType )
```

Constructs a new Permanent object with the specified permanent type.

Parameters

permanentType	The type of the permanent card.

4.7.3 Member Function Documentation

4.7.3.1 getIndex()

```
int Permanent::getIndex ( )  [override], [virtual]
```

Gets the index of the card.

Returns

The index of the card.

Reimplemented from Card.

4.7.3.2 getPermanentType()

```
PermanentType Permanent::getPermanentType ( ) const
```

Gets the type of the permanent card.

Returns

The permanent type.

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

- src/Card/Permanent/Permanent.h
- src/Card/Permanent/Permanent.cpp

4.8 Player Class Reference

Represents a player in the game.

`#include <Player.h>`

Public Member Functions

- Player (Role *role, Character *character)
  Constructs a new Player object.
- Role * getRole () const
  Get the role of the player.
- Character * getCharacter () const
  Get the character of the player.
- std::vector< Card * > * getHand () const
  Get the hand of the player.
- std::vector< Permanent * > * getPermanentCardsPlayed () const
  Get the permanent cards played by the player.
- int getMaxNbAttack () const
  Get the maximum number of attacks the player can perform.
- void **meditationFunction** ()
  the action of the mediation action card
- int attackRapideFunction ()
  the action of the attack rapide permanent card
- std::pair< Permanent *, int > asCodeDuBushido ()
  the action of the code du bushido permanent card
- int armureFunction ()
  the action of the armure permanent card
- void **concentrationFunction** ()
  the action of the concentration permanent card
- bool isDown () const
  if the player is down
- void **recover** ()
  recover all the HP of is character maxHP
- ~**Player** ()
  Destructor for the Player object.

Public Attributes

- int HP
- int honorPoints
- bool asAttacked
- int nbAttack

4.8.1 Detailed Description

Represents a player in the game.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Player()

```
Player::Player (
            Role * role,
            Character * character )
```

Constructs a new Player object.

Parameters

role	The role of the player.
character	The character of the player.

4.8.3 Member Function Documentation

4.8.3.1 armureFunction()

```
int Player::armureFunction ( )
```

the action of the armure permanent card

Returns

int The number of armor points the player has.

4.8.3.2 asCodeDuBushido()

```
std::pair< Permanent *, int > Player::asCodeDuBushido ( )
```

the action of the code du bushido permanent card

Returns

std::pair<Permanent*, int> The code du bushido card and its index in the permanent cards played.

4.8.3.3 attackRapideFunction()

```
int Player::attackRapideFunction ( )
```

the action of the attack rapide permanent card

Returns

int The number of damage the player can deal with all the attack rapide cards played.

4.8.3.4 getCharacter()

```
Character * Player::getCharacter ( ) const
```

Get the character of the player.

Returns

Character* The character of the player.

4.8.3.5 getHand()

```
std::vector< Card * > * Player::getHand ( ) const
```

Get the hand of the player.

Returns

std::vector<Card*>* The hand of the player.

4.8.3.6 getMaxNbAttack()

```
int Player::getMaxNbAttack ( ) const
```

Get the maximum number of attacks the player can perform.

Returns

int The maximum number of attacks the player can perform.

4.8.3.7 getPermanentCardsPlayed()

```
std::vector< Permanent * > * Player::getPermanentCardsPlayed ( ) const
```

Get the permanent cards played by the player.

Returns

std::vector<Permanent*>* The permanent cards played by the player.

4.8.3.8 getRole()

```
Role * Player::getRole ( ) const
```

Get the role of the player.

Returns

Role* The role of the player.

4.8.3.9 isDown()

```
bool Player::isDown ( ) const
```

if the player is down

Returns

bool if the player is down

4.8.4 Member Data Documentation

4.8.4.1 asAttacked

```
bool Player::asAttacked
```

Whether the player has attacked or not

4.8.4.2 honorPoints

```
int Player::honorPoints
```

Honor points of the player

4.8.4.3 HP

```
int Player::HP
```

Health points of the player

4.8.4.4 nbAttack

```
int Player::nbAttack
```

Number of attacks the player can perform

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

- src/Player/Player.h
- src/Player/Player.cpp

4.9 Role Class Reference

The Role class represents a role in the game.

```
#include <Role.h>
```

Public Member Functions

- Role (RoleType type, int level=0)

  Constructs a new Role object.
- RoleType getType () const

  Gets the type of the role.
- int getLevel () const

  Gets the level of the role.
- int getIndex () const

  Gets the index of the role.
- std::string getName () const

  Gets the name of the role.

4.9.1 Detailed Description

The Role class represents a role in the game.

A role has a type and a level. The type determines the role's abilities and characteristics, while the level represents the role's progression or experience.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Role()

```
Role::Role (
            RoleType type,
            int level = 0 )
```

Constructs a new Role object.

Parameters

type	The type of the role.
level	The level of the role (default is 0).

4.9.3 Member Function Documentation

4.9.3.1 getIndex()

```
int Role::getIndex ( ) const
```

Gets the index of the role.

Returns

The index of the role.

4.9.3.2 getLevel()

```
int Role::getLevel ( ) const
```

Gets the level of the role.

Returns

The level of the role.

4.9.3.3 getName()

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

Gets the name of the role.

Returns

The name of the role.

4.9.3.4 getType()

```
RoleType Role::getType ( ) const
```

Gets the type of the role.

Returns

The type of the role.

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

- src/Role/Role.h
- src/Role/Role.cpp

4.10 UI Class Reference

Represents the user interface of the game.

```
#include <UI.h>
```

Public Member Functions

- UI (sf::Font ∗font, sf::Image ∗HPImage, sf::Image ∗HonorImage, sf::Image ∗backRoleImage, sf::Image ∗backCardImage, std::vector< sf::Image ∗ > ∗roleImages, std::vector< sf::Image ∗ > ∗cardImages, std↩ ::vector< sf::Image ∗ > ∗characterImages)

  Constructs a new UI object.
- void menu (sf::Image ∗leftImage, sf::Image ∗rightImage)

  the menu screen
- void **start** ()

  the game screen
- void **update** ()

  update the UI
- void **display** ()

  display the UI
- void **setPlayersSprites** ()

  set the player sprite other than the actual player
- void setSpriteHonorCharactersHP (int index)

  set the honor points and health points of the players other than the actual player
- void **setActualPlayerSprite** ()

  set the actual player sprite, honor points and health points
- void **setHandSprite** ()

  set the actual player card sprites
- void **setStackSprite** ()

  set the stack sprites
- void **setDiscardStackSprite** ()

  set the discard stack sprites
- void **setLogsTexts** ()

  set the logs texts and logs background
- void **setPermanentCircleSprites** ()

  set the red circle that indicates the shogun player if is not is turn
- void handleClickLogBtn (sf::Event event)

  handle the click event on the logs button
- void handleClickHandCard (sf::Event event)

  handle the click event on the hand cards
- void handleClickPassParadeBtn (sf::Event event)

  handle the click event on the parade pass button
- void handleClickEndTurnBtn (sf::Event event)

  handle the click event on the end turn button
- void handleClickDiscardingBtn (sf::Event event)

  handle the click event on the discard button
- void handleClickNobunaga (sf::Event event)

  handle the click event on the Nobunage character if the actual player is Nobunaga
- void **score** ()

  score screen at end of the game
- ∼**UI** ()

  destroy the UI

4.10.1 Detailed Description

Represents the user interface of the game.

The UI class handles the rendering and interaction with the game's graphical user interface. It manages the window, fonts, textures, sprites, buttons, and other UI elements. It also provides methods for updating and displaying the UI, as well as handling user input.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 UI()

```
UI::UI (
            sf::Font * font,
            sf::Image * HPImage,
            sf::Image * HonorImage,
            sf::Image * backRoleImage,
            sf::Image * backCardImage,
            std::vector< sf::Image * > * roleImages,
            std::vector< sf::Image * > * cardImages,
            std::vector< sf::Image * > * characterImages )
```

Constructs a new UI object.

Parameters

font	The font used for the text.
HPImage	The image for the health points.
HonorImage	The image for the honor points.
backRoleImage	The image for the back of the role cards.
backCardImage	The image for the back of the cards.
roleImages	The images for the role cards.
cardImages	The images for the cards.
characterImages	The images for the characters.

4.10.3 Member Function Documentation

4.10.3.1 handleClickDiscardingBtn()

```
void UI::handleClickDiscardingBtn (
            sf::Event event )
```

handle the click event on the discard button

Parameters

event	The sfml event

4.10.3.2 handleClickEndTurnBtn()

```
void UI::handleClickEndTurnBtn (
            sf::Event event )
```

handle the click event on the end turn button

Parameters

event	The sfml event

4.10.3.3 handleClickHandCard()

```
void UI::handleClickHandCard (
            sf::Event event )
```

handle the click event on the hand cards

Parameters

event	The sfml event

4.10.3.4 handleClickLogBtn()

```
void UI::handleClickLogBtn (
            sf::Event event )
```

handle the click event on the logs button

Parameters

event	The sfml event

4.10.3.5 handleClickNobunaga()

```
void UI::handleClickNobunaga (
            sf::Event event )
```

handle the click event on the Nobunage character if the actual player is Nobunaga

Parameters

event	The sfml event

4.10.3.6 handleClickPassParadeBtn()

```
void UI::handleClickPassParadeBtn (
            sf::Event event )
```

handle the click event on the parade pass button

Parameters

event	The sfml event

4.10.3.7 menu()

```
void UI::menu (
            sf::Image * leftImage,
            sf::Image * rightImage )
```

the menu screen

Parameters

leftImage	The image for the left side of the menu
rightImage	The image for the right side of the menu

4.10.3.8 setSpriteHonorCharactersHP()

```
void UI::setSpriteHonorCharactersHP (
            int index )
```

set the honor points and health points of the players other than the actual player

Parameters

index	The index of the player

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

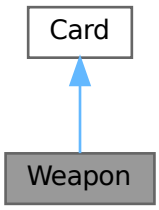
- src/UI/UI.h
- src/UI/UI.cpp

4.11 Weapon Class Reference

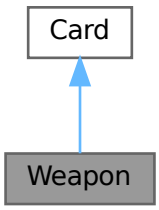
The Weapon class represents a weapon card in the game.

```
#include <Weapon.h>
```

Inheritance diagram for Weapon:



Collaboration diagram for Weapon:



Public Member Functions

- Weapon (WeaponType weaponType, int damage, int range)

  Constructs a new Weapon object.
- WeaponType getWeaponType () const

  Gets the weapon type.
- int getDamage () const

  Gets the damage value of the weapon.
- int getRange () const

  Gets the range value of the weapon.
- int getIndex () override

  Gets the index of the weapon.

Public Member Functions inherited from Card

- Card (CardType type)

  Constructs a Card object with the specified type.
- CardType getType () const

  Gets the type of the card.
- virtual ~**Card** ()

  Destroys the Card object.

4.11.1 Detailed Description

The Weapon class represents a weapon card in the game.

This class inherits from the Card class and provides additional functionality specific to weapons, such as weapon type, damage, and range.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Weapon()

```
Weapon::Weapon (
            WeaponType weaponType,
            int damage,
            int range )
```

Constructs a new Weapon object.

Parameters

weaponType	The type of the weapon.
damage	The damage value of the weapon.
range	The range value of the weapon.

4.11.3 Member Function Documentation

4.11.3.1 getDamage()

```
int Weapon::getDamage ( ) const
```

Gets the damage value of the weapon.

Returns

The damage value.

4.11.3.2 getIndex()

```
int Weapon::getIndex ( )  [override], [virtual]
```

Gets the index of the weapon.

Returns

The index of the weapon.

Reimplemented from Card.

4.11.3.3 getRange()

```
int Weapon::getRange ( ) const
```

Gets the range value of the weapon.

Returns

The range value.

4.11.3.4 getWeaponType()

```
WeaponType Weapon::getWeaponType ( ) const
```

Gets the weapon type.

Returns

The weapon type.

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

- src/Card/Weapon/Weapon.h
- src/Card/Weapon/Weapon.cpp

Chapter 5

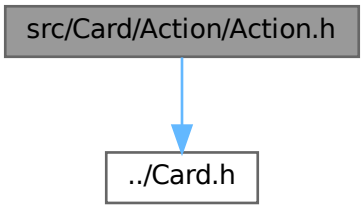
File Documentation

5.1 src/Card/Action/Action.h File Reference

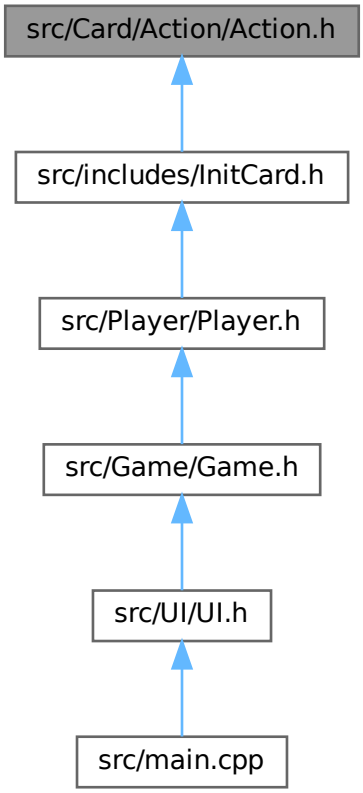
This file contains the declaration of the Action class and the enum class ActionType.

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

Include dependency graph for Action.h:



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



Classes

- class Action

  The Action class represents an action card in the game. This class inherits from the Card class and adds an Action↩ Type attribute. It provides methods to get the action type and the index of the card.

Enumerations

- enum class ActionType {
  CRI_DE_GUERRE , DAIMYO , DIVERSION , GEISHA ,
  MEDITATION , PARADE , CEREMONIE_DU_THE , JU_JITSU }

  Represents the type of an Action.

5.1.1 Detailed Description

This file contains the declaration of the Action class and the enum class ActionType.

5.1.2 Enumeration Type Documentation

5.1.2.1 ActionType

```
enum class ActionType  [strong]
```

Represents the type of an Action.

Note

The type of an Action is also the index of the Action in the Image List and as the name of the Action

Enumerator

CRI_DE_GUERRE	CRI_DE_GUERRE action type
DAIMYO	DAIMYO action type
DIVERSION	DIVERSION action type
GEISHA	GEISHA action type
MEDITATION	MEDITATION action type
PARADE	PARADE action type
CEREMONIE_DU_THE	CEREMONIE_DUTHE action type
JU_JITSU	JU_JITSU action type

5.2 Action.h

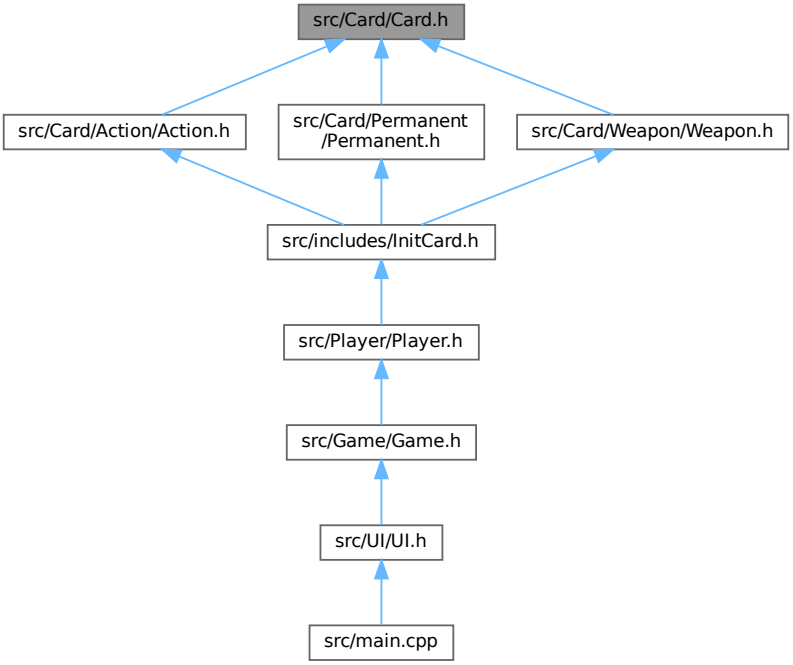
Go to the documentation of this file.

```
00001
00006 #ifndef ACTION_H
00007 #define ACTION_H
00008
00009 #include "../Card.h"
00010
00017 enum class ActionType {
00018     CRI_DE_GUERRE,
00019     DAIMYO,
00020     DIVERSION,
00021     GEISHA,
00022     MEDITATION,
00023     PARADE,
00024     CEREMONIE_DU_THE,
00025     JU_JITSU
00026 };
00027
00034 class Action : public Card {
00035     private:
00036         ActionType actionType;
00038     public:
00044         Action(ActionType actionType);
00045
00051         ActionType getActionType() const;
00052
00058         int getIndex() override;
00059 };
00060
00061 #endif // ACTION_H
```

5.3 src/Card/Card.h File Reference

This file contains the declaration of the Card class and the enum class CardType.

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



Classes

- class Card

  The Card class represents a card in the game.

Enumerations

- enum class CardType { ACTION , PERMANENT , WEAPON }

  Represents the type of a card.

5.3.1 Detailed Description

This file contains the declaration of the Card class and the enum class CardType.

5.3.2 Enumeration Type Documentation

5.3.2.1 CardType

```
enum class CardType  [strong]
```

Represents the type of a card.

Enumerator

ACTION	Action card type
PERMANENT	Permanent card type
WEAPON	Weapon card type

5.4 Card.h

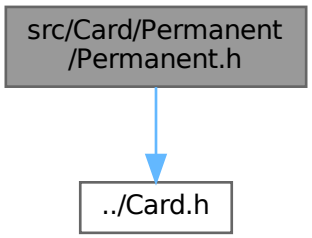
Go to the documentation of this file.

```
00001
00006 #ifndef CARD_H
00007 #define CARD_H
00008
00013 enum class CardType {
00014     ACTION,
00015     PERMANENT,
00016     WEAPON
00017 };
00018
00026 class Card {
00027     private:
00028         CardType type;
00030     public:
00036         Card(CardType type);
00037
00042         CardType getType() const;
00043
00048         virtual int getIndex();
00049
00053         virtual ~Card();
00054 };
00055
00056 #endif // CARD_H
```

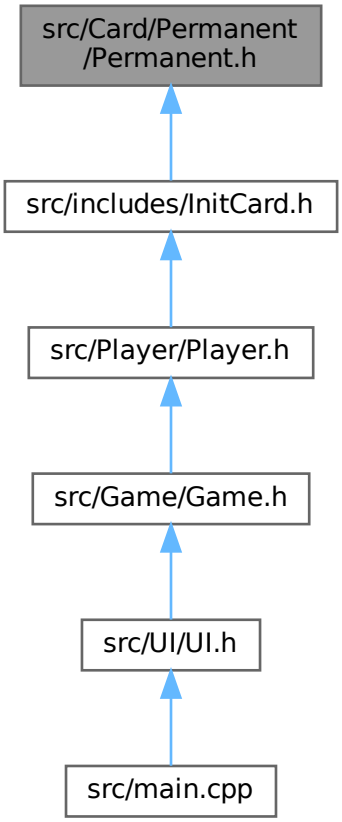
5.5 src/Card/Permanent/Permanent.h File Reference

This file contains the declaration of the Permanent class and the enum class PermanentType.

`#include "../Card.h"`
Include dependency graph for Permanent.h:



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



Classes

- class Permanent

  The Permanent class represents a permanent card in the game. This class inherits from the Card class and adds functionality specific to permanent cards.

Enumerations

- enum class PermanentType { ATTAQUE_RAPIDE , CODE_DU_BUSHIDO , ARMURE , CONCENTRATION }

  Represents the type of a Permanent.

5.5.1 Detailed Description

This file contains the declaration of the Permanent class and the enum class PermanentType.

5.5.2 Enumeration Type Documentation

5.5.2.1 PermanentType

```
enum class PermanentType  [strong]
```

Represents the type of a Permanent.

Note

The type of a Permanent is also the index of the Permanent in the Image List and as the name of the Permanent

Enumerator

ATTAQUE_RAPIDE	ATTAQUE_RAPIDE permanent type
CODE_DU_BUSHIDO	CODE_DU_BUSHIDO permanent type
ARMURE	ARMURE permanent type
CONCENTRATION	CONCENTRATION permanent type

5.6 Permanent.h

Go to the documentation of this file.

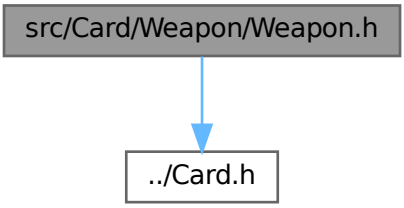
```
00001
00006 #ifndef PERMANENT_H
00007 #define PERMANENT_H
00008
00009 #include "../Card.h"
00010
00017 enum class PermanentType {
00018     ATTAQUE_RAPIDE,
00019     CODE_DU_BUSHIDO,
00020     ARMURE,
00021     CONCENTRATION
00022 };
00023
00029 class Permanent : public Card {
00030     private:
00031         PermanentType permanentType;
00033     public:
00039         Permanent(PermanentType permanentType);
00040
00046         PermanentType getPermanentType() const;
00047
00053         int getIndex() override;
00054 };
00055
00056 #endif // PERMANENT_H
```

5.7 src/Card/Weapon/Weapon.h File Reference

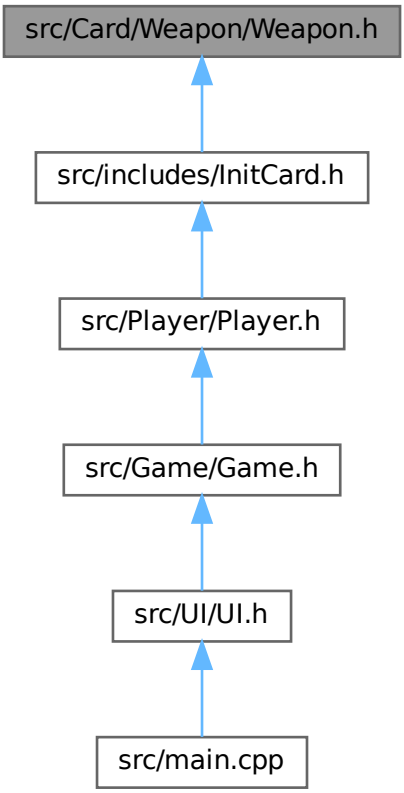
This file contains the declaration of the Weapon class and the enum class WeaponType.

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

Include dependency graph for Weapon.h:



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



Classes

- class Weapon

  The Weapon class represents a weapon card in the game.

Enumerations

- enum class WeaponType {
  NODACHI , NAGINATA , NAGAYARI , TANEGASHIMA ,
  DAIKYU , BO , KUSARIGAMA , KATANA ,
  SHURIKEN , KANABO , BOKKEN , KISERU ,
  WAKIZASHI }

  Represents the type of a Weapon.

5.7.1 Detailed Description

This file contains the declaration of the Weapon class and the enum class WeaponType.

5.7.2 Enumeration Type Documentation

5.7.2.1 WeaponType

```
enum class WeaponType  [strong]
```

Represents the type of a Weapon.

Note

The type of a Weapon is also the index of the Weapon in the Image List and as the name of the Weapon

Enumerator

NODACHI	NODACHI weapon type
NAGINATA	NAGINATA weapon type
NAGAYARI	NAGAYARI weapon type
TANEGASHIMA	TANEGASHIMA weapon type
DAIKYU	DAIKYU weapon type
BO	BO weapon type
KUSARIGAMA	KUSARIGAMA weapon type
KATANA	KATANA weapon type
SHURIKEN	SHURIKEN weapon type
KANABO	KANABO weapon type
BOKKEN	BOKKEN weapon type
KISERU	KISERU weapon type
WAKIZASHI	WAKIZASHI weapon type

5.8 Weapon.h

Go to the documentation of this file.

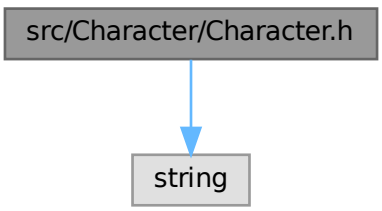
```
00001
00006 #ifndef WEAPON_H
```

```
00007 #define WEAPON_H
00008
00009 #include "../Card.h"
00010
00017 enum class WeaponType {
00018     NODACHI,
00019     NAGINATA,
00020     NAGAYARI,
00021     TANEGASHIMA,
00022     DAIKYU,
00023     BO,
00024     KUSARIGAMA,
00025     KATANA,
00026     SHURIKEN,
00027     KANABO,
00028     BOKKEN,
00029     KISERU,
00030     WAKIZASHI
00031 };
00032
00041 class Weapon : public Card {
00042     private:
00043         WeaponType weaponType;
00044         int damage;
00045         int range;
00047     public:
00055         Weapon(WeaponType weaponType, int damage, int range);
00056
00062         WeaponType getWeaponType() const;
00063
00069         int getDamage() const;
00070
00076         int getRange() const;
00077
00083         int getIndex() override;
00084 };
00085
00086 #endif // WEAPON_H
```

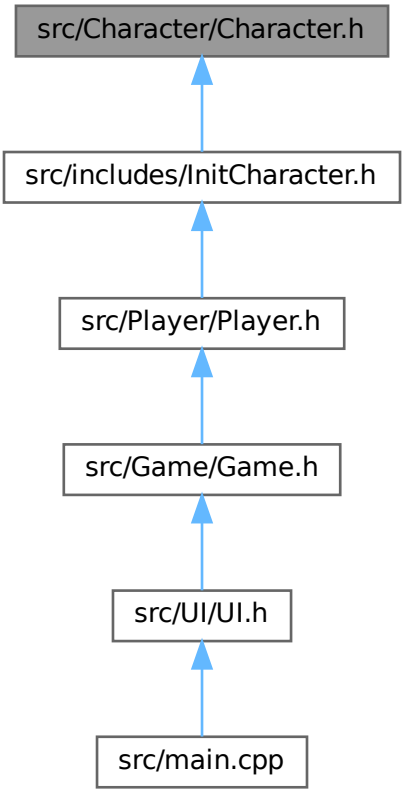
5.9 src/Character/Character.h File Reference

This file contains the declaration of the Character class and the enum class CharacterType.

`#include <string>`
Include dependency graph for Character.h:



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



Classes

- class Character

  The Character class represents a character in the game.

Enumerations

- enum class CharacterType {
  HANZO , USHIWAKA , CHIYOME , HIDEYOSHI ,
  GINCHIYO , GOEMON , NOBUNAGA , TOMOE ,
  IEYASU , BENKEI , MUSASHI , KOJIRO }

  Represents the type of a Character.

5.9.1 Detailed Description

This file contains the declaration of the Character class and the enum class CharacterType.

5.9.2 Enumeration Type Documentation

5.9.2.1 CharacterType

```
enum class CharacterType  [strong]
```

Represents the type of a Character.

Note

The type of a Character is also the index of the Character in the Image List and as the name of the Character

Enumerator

HANZO	HANZO character type
USHIWAKA	USHIWAKA character type
CHIYOME	CHIYOME character type
HIDEYOSHI	HIDEYOSHI character type
GINCHIYO	GINCHIYO character type
GOEMON	GOEMON character type
NOBUNAGA	NOBUNAGA character type
TOMOE	TOMOE character type
IEYASU	IEYASU character type
BENKEI	BENKEI character type
MUSASHI	MUSASHI character type
KOJIRO	KOJIRO character type

5.10 Character.h

Go to the documentation of this file.

```
00001
00006 #ifndef CHARACTER_H
00007 #define CHARACTER_H
00008
00009 #include <string>
00010
00017 enum class CharacterType {
00018     HANZO,
00019     USHIWAKA,
00020     CHIYOME,
00021     HIDEYOSHI,
00022     GINCHIYO,
00023     GOEMON,
00024     NOBUNAGA,
00025     TOMOE,
00026     IEYASU,
00027     BENKEI,
00028     MUSASHI,
00029     KOJIRO
00030 };
00031
00039 class Character {
00040     private:
00041         CharacterType type;
00042         int HP;
00044     public:
00051         Character(CharacterType type, int HP);
00052
00058         CharacterType getType() const;
```

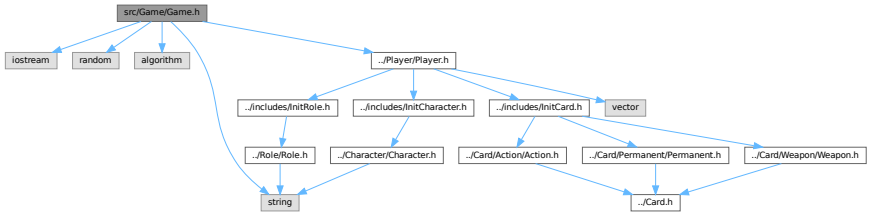
```
00059
00065         int getIndex() const;
00066
00072         int getHP() const;
00073
00079         std::string getName() const;
00080 };
00081
00082 #endif // CHARACTER_H
```

5.11 src/Game/Game.h File Reference

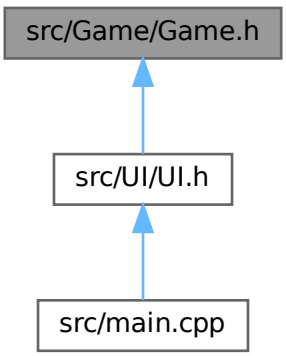
This file contains the declaration of the Game class.

```
#include <iostream>
#include <random>
#include <algorithm>
#include <string>
#include "../Player/Player.h"
```

Include dependency graph for Game.h:



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



Classes

- class Game

  Represents the game state and logic.

5.11.1 Detailed Description

This file contains the declaration of the Game class.

5.12 Game.h

Go to the documentation of this file.

```
00001
00006 #ifndef GAME_H
00007 #define GAME_H
00008
00009 #include <iostream>
00010 #include <random>
00011 #include <algorithm>
00012 #include <string>
00013 #include "../Player/Player.h"
00014
00022 class Game {
00023     private:
00024         std::vector<Role*> *roles;
00025         std::vector<Character*> *characters;
00026         std::vector<Card*> *cards;
00027         std::vector<Card*> *discards;
00028         std::vector<Player*> *players;
00029         std::vector<std::string*> *logs;
00030         int nbPlayers;
00031         int indexActualPlayer;
00033     public:
00034         bool isCarteDuBushidoInGame;
00039         Game();
00040
00044         void initRole();
00045
00049         void initCharacter();
00050
00054         void initCard();
00055
00059         void initPlayer();
00060
00065         void setNbPlayers(int nbPlayers);
00066
00071         int getNbPlayers() const;
00072
00077         int getIndexActualPlayer() const;
00078
00083         int getIndexPlayerAttacked() const;
00084
00089         std::vector<Player*> *getPlayers() const;
00090
00095         std::vector<Card*> *getCards() const;
00096
00101         std::vector<Card*> *getDiscards() const;
00102
00107         std::vector<std::string*> *getLogs() const;
00108
00112         void updateHonorPointsHP();
00113
00117         void recoverHP();
00118
00124         void pick(Player *player, int nbCard);
00125
00132         bool attack(Weapon *card, Player *player);
00133
00139         int calculateDistance(Player *playerTarget);
00140
00146         bool canBlock(Player *player);
00147
00153         void discard(Player *player, Card* card);
00154
00158         void recoverCards();
00159
00163         void changePlayer();
00164
00168         void criDeGuerreFunction();
00169
00173         void daimyoFunction();
00174
00179         void geishaFunction(Player *player = nullptr);
00180
```

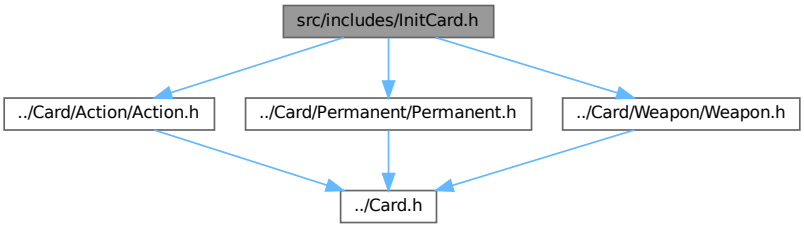
```
00184         void ceremonieDuTheFunction();
00185
00189         void juJitsuFunction();
00190
00194         void codeDuBushidoFunction();
00195
00200         bool isGameOver();
00201
00205         ~Game();
00206 };
00207
00208 #endif // GAME_H
```

5.13 src/includes/InitCard.h File Reference

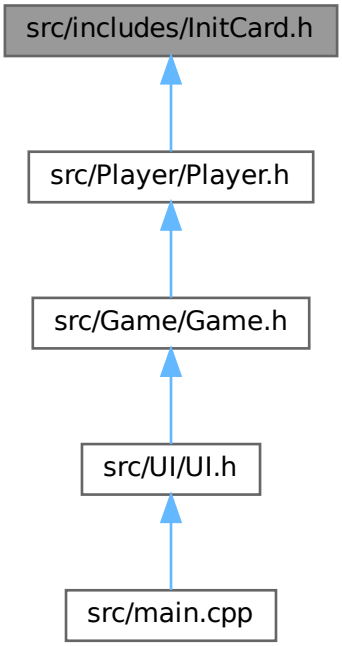
This file contains the initialization of various cards, actions, permanents, and weapons.

```
#include "../Card/Action/Action.h"
#include "../Card/Permanent/Permanent.h"
#include "../Card/Weapon/Weapon.h"
```

Include dependency graph for InitCard.h:



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



Macros

- #define CriDeGuerre Action(ActionType::CRI_DE_GUERRE)
- #define Daimyo Action(ActionType::DAIMYO)
- #define Diversion Action(ActionType::DIVERSION)
- #define Geisha Action(ActionType::GEISHA)
- #define Meditation Action(ActionType::MEDITATION)
- #define Parade Action(ActionType::PARADE)
- #define CeremonieDuThe Action(ActionType::CEREMONIE_DU_THE)
- #define JuJitsu Action(ActionType::JU_JITSU)
- #define AttaqueRapide Permanent(PermanentType::ATTAQUE_RAPIDE)
- #define CodeDuBushido Permanent(PermanentType::CODE_DU_BUSHIDO)
- #define Armure Permanent(PermanentType::ARMURE)
- #define Concentration Permanent(PermanentType::CONCENTRATION)
- #define Nodachi Weapon(WeaponType::NODACHI, 3, 3)
- #define Naginata Weapon(WeaponType::NAGINATA, 1, 4)
- #define Nagayari Weapon(WeaponType::NAGAYARI, 2, 4)
- #define Tanegashima Weapon(WeaponType::TANEGASHIMA, 1, 5)
- #define Daikyu Weapon(WeaponType::DAIKYU, 2, 5)
- #define Bo Weapon(WeaponType::BO, 1, 2)
- #define Kusarigama Weapon(WeaponType::KUSARIGAMA, 2, 2)
- #define Katana Weapon(WeaponType::KATANA, 3, 2)
- #define Shuriken Weapon(WeaponType::SHURIKEN, 1, 3)
- #define Kanabo Weapon(WeaponType::KANABO, 2, 3)
- #define Bokken Weapon(WeaponType::BOKKEN, 1, 1)

- #define Kiseru Weapon(WeaponType::KISERU, 2, 1)
- #define Wakizashi Weapon(WeaponType::WAKIZASHI, 3, 1)
- #define NB_COPY_CRI_DE_GUERRE 4
- #define NB_COPY_DAIMYO 3
- #define NB_COPY_DIVERSION 5
- #define NB_COPY_GEISHA 6
- #define NB_COPY_MEDITATION 3
- #define NB_COPY_PARADE 15
- #define NB_COPY_CEREMONIE_DU_THE 4
- #define NB_COPY_JU_JITSU 3
- #define NB_COPY_ATTAQUE_RAPIDE 3
- #define NB_COPY_CODE_DU_BUSHIDO 2
- #define NB_COPY_ARMURE 4
- #define NB_COPY_CONCENTRATION 6
- #define NB_COPY_NODACHI 1
- #define NB_COPY_NAGINATA 2
- #define NB_COPY_NAGAYARI 1
- #define NB_COPY_TANEGASHIMA 1
- #define NB_COPY_DAIKYU 1
- #define NB_COPY_BO 5
- #define NB_COPY_KUSARIGAMA 4
- #define NB_COPY_KATANA 1
- #define NB_COPY_SHURIKEN 3
- #define NB_COPY_KANABO 1
- #define NB_COPY_BOKKEN 6
- #define NB_COPY_KISERU 5
- #define NB_COPY_WAKIZASHI 1

5.13.1 Detailed Description

This file contains the initialization of various cards, actions, permanents, and weapons.

5.13.2 Macro Definition Documentation

5.13.2.1 Armure

```
#define Armure Permanent(PermanentType::ARMURE)
```

ARMURE permanent card

5.13.2.2 AttaqueRapide

```
#define AttaqueRapide Permanent(PermanentType::ATTAQUE_RAPIDE)
```

ATTAQUE_RAPIDE permanent card

5.13.2.3 Bo

```
#define Bo Weapon(WeaponType::BO, 1, 2)
```

BO weapon card

5.13.2.4 Bokken

```
#define Bokken Weapon(WeaponType::BOKKEN, 1, 1)
```

BOKKEN weapon card

5.13.2.5 CeremonieDuThe

```
#define CeremonieDuThe Action(ActionType::CEREMONIE_DU_THE)
```

CEREMONIE_DU_THE action card

5.13.2.6 CodeDuBushido

```
#define CodeDuBushido Permanent(PermanentType::CODE_DU_BUSHIDO)
```

CODE_DU_BUSHIDO permanent card

5.13.2.7 Concentration

```
#define Concentration Permanent(PermanentType::CONCENTRATION)
```

CONCENTRATION permanent card

5.13.2.8 CriDeGuerre

```
#define CriDeGuerre Action(ActionType::CRI_DE_GUERRE)
```

CRI_DE_GUERRE action card

5.13.2.9 Daikyu

```
#define Daikyu Weapon(WeaponType::DAIKYU, 2, 5)
```

DAIKYU weapon card

5.13.2.10 Daimyo

```
#define Daimyo Action(ActionType::DAIMYO)
```

DAIMYO action card

5.13.2.11 Diversion

```
#define Diversion Action(ActionType::DIVERSION)
```

DIVERSION action card

5.13.2.12 Geisha

```
#define Geisha Action(ActionType::GEISHA)
```

GEISHA action card

5.13.2.13 JuJitsu

```
#define JuJitsu Action(ActionType::JU_JITSU)
```

JU_JITSU action card

5.13.2.14 Kanabo

```
#define Kanabo Weapon(WeaponType::KANABO, 2, 3)
```

KANABO weapon card

5.13.2.15 Katana

```
#define Katana Weapon(WeaponType::KATANA, 3, 2)
```

KATANA weapon card

5.13.2.16 Kiseru

```
#define Kiseru Weapon(WeaponType::KISERU, 2, 1)
```

KISERU weapon card

5.13.2.17 Kusarigama

```
#define Kusarigama Weapon(WeaponType::KUSARIGAMA, 2, 2)
```

KUSARIGAMA weapon card

5.13.2.18 Meditation

```
#define Meditation Action(ActionType::MEDITATION)
```

MEDITATION action card

5.13.2.19 Nagayari

```
#define Nagayari Weapon(WeaponType::NAGAYARI, 2, 4)
```

NAGAYARI weapon card

5.13.2.20 Naginata

```
#define Naginata Weapon(WeaponType::NAGINATA, 1, 4)
```

NAGINATA weapon card

5.13.2.21 NB_COPY_ARMURE

```
#define NB_COPY_ARMURE 4
```

Number of copies of ARMURE permanent card

5.13.2.22 NB_COPY_ATTAQUE_RAPIDE

```
#define NB_COPY_ATTAQUE_RAPIDE 3
```

Number of copies of ATTAQUE_RAPIDE permanent card

5.13.2.23 NB_COPY_BO

```
#define NB_COPY_BO 5
```

Number of copies of BO weapon card

5.13.2.24 NB_COPY_BOKKEN

```
#define NB_COPY_BOKKEN 6
```

Number of copies of BOKKEN weapon card

5.13.2.25 NB_COPY_CEREMONIE_DU_THE

```
#define NB_COPY_CEREMONIE_DU_THE 4
```

Number of copies of CEREMONIE_DU_THE action card

5.13.2.26 NB_COPY_CODE_DU_BUSHIDO

```
#define NB_COPY_CODE_DU_BUSHIDO 2
```

Number of copies of CODE_DU_BUSHIDO permanent card

5.13.2.27 NB_COPY_CONCENTRATION

```
#define NB_COPY_CONCENTRATION 6
```

Number of copies of CONCENTRATION permanent card

5.13.2.28 NB_COPY_CRI_DE_GUERRE

```
#define NB_COPY_CRI_DE_GUERRE 4
```

Number of copies of CRI_DE_GUERRE action card

5.13.2.29 NB_COPY_DAIKYU

```
#define NB_COPY_DAIKYU 1
```

Number of copies of DAIKYU weapon card

5.13.2.30 NB_COPY_DAIMYO

```
#define NB_COPY_DAIMYO 3
```

Number of copies of DAIMYO action card

5.13.2.31 NB_COPY_DIVERSION

```
#define NB_COPY_DIVERSION 5
```

Number of copies of DIVERSION action card

5.13.2.32 NB_COPY_GEISHA

```
#define NB_COPY_GEISHA 6
```

Number of copies of GEISHA action card

5.13.2.33 NB_COPY_JU_JITSU

```
#define NB_COPY_JU_JITSU 3
```

Number of copies of JU_JITSU action card

5.13.2.34 NB_COPY_KANABO

```
#define NB_COPY_KANABO 1
```

Number of copies of KANABO weapon card

5.13.2.35 NB_COPY_KATANA

```
#define NB_COPY_KATANA 1
```

Number of copies of KATANA weapon card

5.13.2.36 NB_COPY_KISERU

```
#define NB_COPY_KISERU 5
```

Number of copies of KISERU weapon card

5.13.2.37 NB_COPY_KUSARIGAMA

```
#define NB_COPY_KUSARIGAMA 4
```

Number of copies of KUSARIGAMA weapon card

5.13.2.38 NB_COPY_MEDITATION

```
#define NB_COPY_MEDITATION 3
```

Number of copies of MEDITATION action card

5.13.2.39 NB_COPY_NAGAYARI

```
#define NB_COPY_NAGAYARI 1
```

Number of copies of NAGAYARI weapon card

5.13.2.40 NB_COPY_NAGINATA

```
#define NB_COPY_NAGINATA 2
```

Number of copies of NAGINATA weapon card

5.13.2.41 NB_COPY_NODACHI

```
#define NB_COPY_NODACHI 1
```

Number of copies of NODACHI weapon card

5.13.2.42 NB_COPY_PARADE

```
#define NB_COPY_PARADE 15
```

Number of copies of PARADE action card

5.13.2.43 NB_COPY_SHURIKEN

```
#define NB_COPY_SHURIKEN 3
```

Number of copies of SHURIKEN weapon card

5.13.2.44 NB_COPY_TANEGASHIMA

```
#define NB_COPY_TANEGASHIMA 1
```

Number of copies of TANEGASHIMA weapon card

5.13.2.45 NB_COPY_WAKIZASHI

```
#define NB_COPY_WAKIZASHI 1
```

Number of copies of WAKIZASHI weapon card

5.13.2.46 Nodachi

```
#define Nodachi Weapon(WeaponType::NODACHI, 3, 3)
```

NODACHI weapon card

5.13.2.47 Parade

```
#define Parade Action(ActionType::PARADE)
```

PARADE action card

5.13.2.48 Shuriken

```
#define Shuriken Weapon(WeaponType::SHURIKEN, 1, 3)
```

SHURIKEN weapon card

5.13.2.49 Tanegashima

```
#define Tanegashima Weapon(WeaponType::TANEGASHIMA, 1, 5)
```

TANEGASHIMA weapon card

5.13.2.50 Wakizashi

```
#define Wakizashi Weapon(WeaponType::WAKIZASHI, 3, 1)
```

WAKIZASHI weapon card

5.14 InitCard.h

Go to the documentation of this file.

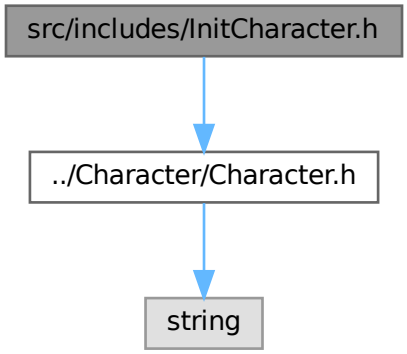
```
00001
00006 #ifndef INIT_CARD_H
00007 #define INIT_CARD_H
00008
00009 #include "../Card/Action/Action.h"
00010 #include "../Card/Permanent/Permanent.h"
00011 #include "../Card/Weapon/Weapon.h"
00012
00013 #define CriDeGuerre Action(ActionType::CRI_DE_GUERRE)
00014 #define Daimyo Action(ActionType::DAIMYO)
00015 #define Diversion Action(ActionType::DIVERSION)
00016 #define Geisha Action(ActionType::GEISHA)
00017 #define Meditation Action(ActionType::MEDITATION)
00018 #define Parade Action(ActionType::PARADE)
00019 #define CeremonieDuThe Action(ActionType::CEREMONIE_DU_THE)
00020 #define JuJitsu Action(ActionType::JU_JITSU)
00022 #define AttaqueRapide Permanent(PermanentType::ATTAQUE_RAPIDE)
00023 #define CodeDuBushido Permanent(PermanentType::CODE_DU_BUSHIDO)
00024 #define Armure Permanent(PermanentType::ARMURE)
00025 #define Concentration Permanent(PermanentType::CONCENTRATION)
00027 #define Nodachi Weapon(WeaponType::NODACHI, 3, 3)
00028 #define Naginata Weapon(WeaponType::NAGINATA, 1, 4)
00029 #define Nagayari Weapon(WeaponType::NAGAYARI, 2, 4)
00030 #define Tanegashima Weapon(WeaponType::TANEGASHIMA, 1, 5)
00031 #define Daikyu Weapon(WeaponType::DAIKYU, 2, 5)
00032 #define Bo Weapon(WeaponType::BO, 1, 2)
00033 #define Kusarigama Weapon(WeaponType::KUSARIGAMA, 2, 2)
00034 #define Katana Weapon(WeaponType::KATANA, 3, 2)
00035 #define Shuriken Weapon(WeaponType::SHURIKEN, 1, 3)
00036 #define Kanabo Weapon(WeaponType::KANABO, 2, 3)
00037 #define Bokken Weapon(WeaponType::BOKKEN, 1, 1)
00038 #define Kiseru Weapon(WeaponType::KISERU, 2, 1)
00039 #define Wakizashi Weapon(WeaponType::WAKIZASHI, 3, 1)
00041 #define NB_COPY_CRI_DE_GUERRE 4
00042 #define NB_COPY_DAIMYO 3
00043 #define NB_COPY_DIVERSION 5
00044 #define NB_COPY_GEISHA 6
00045 #define NB_COPY_MEDITATION 3
00046 #define NB_COPY_PARADE 15
00047 #define NB_COPY_CEREMONIE_DU_THE 4
00048 #define NB_COPY_JU_JITSU 3
00050 #define NB_COPY_ATTAQUE_RAPIDE 3
00051 #define NB_COPY_CODE_DU_BUSHIDO 2
00052 #define NB_COPY_ARMURE 4
00053 #define NB_COPY_CONCENTRATION 6
00055 #define NB_COPY_NODACHI 1
00056 #define NB_COPY_NAGINATA 2
00057 #define NB_COPY_NAGAYARI 1
00058 #define NB_COPY_TANEGASHIMA 1
00059 #define NB_COPY_DAIKYU 1
00060 #define NB_COPY_BO 5
00061 #define NB_COPY_KUSARIGAMA 4
00062 #define NB_COPY_KATANA 1
00063 #define NB_COPY_SHURIKEN 3
00064 #define NB_COPY_KANABO 1
00065 #define NB_COPY_BOKKEN 6
00066 #define NB_COPY_KISERU 5
00067 #define NB_COPY_WAKIZASHI 1
00069 #endif // INIT_CARD_H
```

5.15 src/includes/InitCharacter.h File Reference

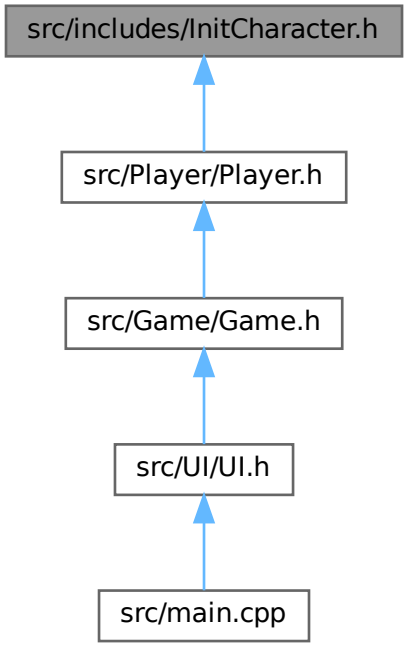
This file contains the initialization of character objects.

```
#include "../Character/Character.h"
```

Include dependency graph for InitCharacter.h:



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



Macros

- #define Hanzo Character(CharacterType::HANZO, 4)

- #define Ushiwaka Character(CharacterType::USHIWAKA, 4)
- #define Chiyome Character(CharacterType::CHIYOME, 4)
- #define Hideyoshi Character(CharacterType::HIDEYOSHI, 4)
- #define Ginchyo Character(CharacterType::GINCHIYO, 4)
- #define Goemon Character(CharacterType::GOEMON, 5)
- #define Nobunaga Character(CharacterType::NOBUNAGA, 5)
- #define Tomoe Character(CharacterType::TOMOE, 5)
- #define Ieyasu Character(CharacterType::IEYASU, 5)
- #define Benkei Character(CharacterType::BENKEI, 5)
- #define Musashi Character(CharacterType::MUSASHI, 5)
- #define Kojiro Character(CharacterType::KOJIRO, 5)

5.15.1 Detailed Description

This file contains the initialization of character objects.

5.15.2 Macro Definition Documentation

5.15.2.1 Benkei

```
#define Benkei Character(CharacterType::BENKEI, 5)
```

Benkei character

5.15.2.2 Chiyome

```
#define Chiyome Character(CharacterType::CHIYOME, 4)
```

Chiyome character

5.15.2.3 Ginchyo

```
#define Ginchyo Character(CharacterType::GINCHIYO, 4)
```

Ginchyo character

5.15.2.4 Goemon

```
#define Goemon Character(CharacterType::GOEMON, 5)
```

Goemon character

5.15.2.5 Hanzo

```
#define Hanzo Character(CharacterType::HANZO, 4)
```

Hanzo character

5.15.2.6 Hideyoshi

```
#define Hideyoshi Character(CharacterType::HIDEYOSHI, 4)
```

Hideyoshi character

5.15.2.7 Ieyasu

```
#define Ieyasu Character(CharacterType::IEYASU, 5)
```

Ieyasu character

5.15.2.8 Kojiro

```
#define Kojiro Character(CharacterType::KOJIRO, 5)
```

Kojiro character

5.15.2.9 Musashi

```
#define Musashi Character(CharacterType::MUSASHI, 5)
```

Musashi character

5.15.2.10 Nobunaga

```
#define Nobunaga Character(CharacterType::NOBUNAGA, 5)
```

Nobunaga character

5.15.2.11 Tomoe

```
#define Tomoe Character(CharacterType::TOMOE, 5)
```

Tomoe character

5.15.2.12 Ushiwaka

```
#define Ushiwaka Character(CharacterType::USHIWAKA, 4)
```

Ushiwaka character

5.16 InitCharacter.h

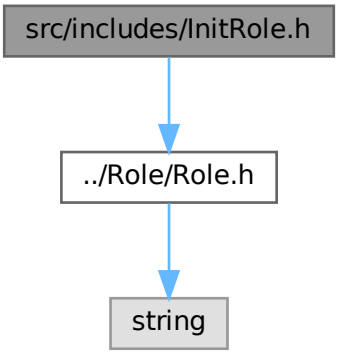
Go to the documentation of this file.

```
00001
00006 #ifndef INIT_CHARACTER_H
00007 #define INIT_CHARACTER_H
00008
00009 #include "../Character/Character.h"
00010
00011 #define Hanzo Character(CharacterType::HANZO, 4)
00012 #define Ushiwaka Character(CharacterType::USHIWAKA, 4)
00013 #define Chiyome Character(CharacterType::CHIYOME, 4)
00014 #define Hideyoshi Character(CharacterType::HIDEYOSHI, 4)
00015 #define Ginchyo Character(CharacterType::GINCHIYO, 4)
00016 #define Goemon Character(CharacterType::GOEMON, 5)
00017 #define Nobunaga Character(CharacterType::NOBUNAGA, 5)
00018 #define Tomoe Character(CharacterType::TOMOE, 5)
00019 #define Ieyasu Character(CharacterType::IEYASU, 5)
00020 #define Benkei Character(CharacterType::BENKEI, 5)
00021 #define Musashi Character(CharacterType::MUSASHI, 5)
00022 #define Kojiro Character(CharacterType::KOJIRO, 5)
00024 #endif // INIT_CHARACTER_H
```

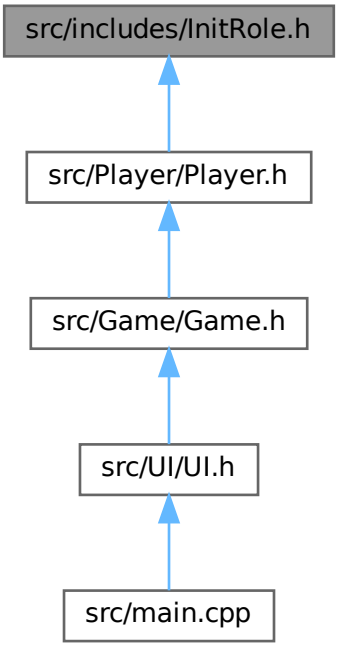
5.17 src/includes/InitRole.h File Reference

This file contains the initialization of roles.

`#include "../Role/Role.h"`
Include dependency graph for InitRole.h:



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



Macros

- #define Shogun Role(RoleType::SHOGUN)
- #define Samurai Role(RoleType::SAMURAI)
- #define NinjaOne Role(RoleType::NINJA, 1)
- #define NinjaTwo Role(RoleType::NINJA, 2)
- #define NinjaThree Role(RoleType::NINJA, 3)
- #define Ronin Role(RoleType::RONIN)

5.17.1 Detailed Description

This file contains the initialization of roles.

5.17.2 Macro Definition Documentation

5.17.2.1 NinjaOne

```
#define NinjaOne Role(RoleType::NINJA, 1)
```

Ninja role level 1

5.17.2.2 NinjaThree

```
#define NinjaThree Role(RoleType::NINJA, 3)
```

Ninja role level 3

5.17.2.3 NinjaTwo

```
#define NinjaTwo Role(RoleType::NINJA, 2)
```

Ninja role level 2

5.17.2.4 Ronin

```
#define Ronin Role(RoleType::RONIN)
```

Ronin role

5.17.2.5 Samurai

```
#define Samurai Role(RoleType::SAMURAI)
```

Samurai role

5.17.2.6 Shogun

```
#define Shogun Role(RoleType::SHOGUN)
```

Shogun role

5.18 InitRole.h

Go to the documentation of this file.

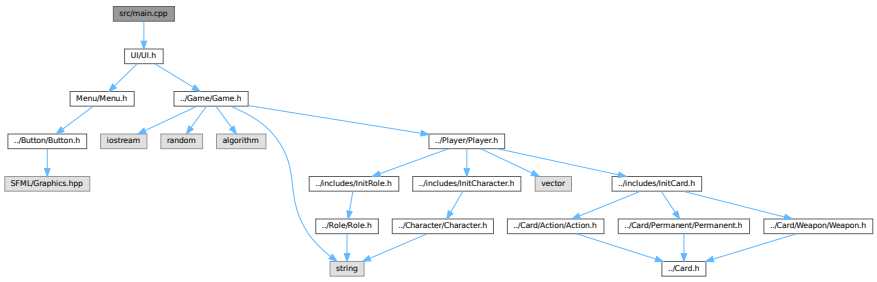
```
00001
00006 #ifndef INIT_ROLE_H
00007 #define INIT_ROLE_H
00008
00009 #include "../Role/Role.h"
00010
00011 #define Shogun Role(RoleType::SHOGUN)
00012 #define Samurai Role(RoleType::SAMURAI)
00013 #define NinjaOne Role(RoleType::NINJA, 1)
00014 #define NinjaTwo Role(RoleType::NINJA, 2)
00015 #define NinjaThree Role(RoleType::NINJA, 3)
00016 #define Ronin Role(RoleType::RONIN)
00018 #endif // INIT_ROLE_H
```

5.19 src/main.cpp File Reference

The main function of the game.

```
#include "UI/UI.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

  The main function of the program.

5.19.1 Detailed Description

The main function of the game.

5.19.2 Function Documentation

5.19.2.1 main()

```
int main ( )
```

The main function of the program.

This function is the entry point of the program. It initializes various objects, loads images and fonts, creates UI elements, and manages the flow of the game. It also deallocates memory before exiting.

Returns

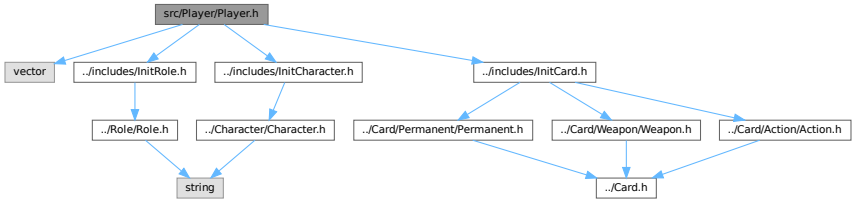
int The exit status of the program.

5.20 src/Player/Player.h File Reference

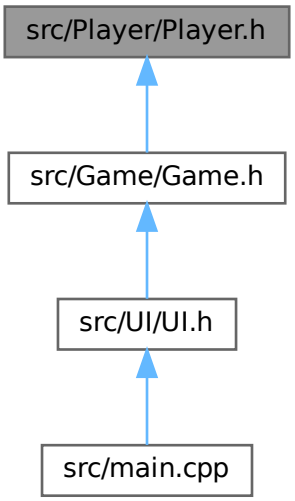
This file contains the declaration of the Player class.

```
#include <vector>
#include "../includes/InitRole.h"
#include "../includes/InitCard.h"
#include "../includes/InitCharacter.h"
```

Include dependency graph for Player.h:



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



Classes

- class Player

  Represents a player in the game.

5.20.1 Detailed Description

This file contains the declaration of the Player class.

5.21 Player.h

Go to the documentation of this file.

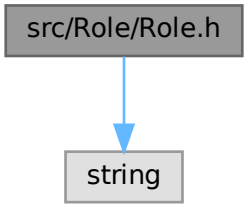
```
00001
00006 #ifndef PLAYER_H
00007 #define PLAYER_H
00008
00009 #include <vector>
00010 #include "../includes/InitRole.h"
00011 #include "../includes/InitCard.h"
00012 #include "../includes/InitCharacter.h"
00013
00018 class Player {
00019     private:
00020         Role *role;
00021         Character *character;
00022         std::vector<Card*> *hand;
00023         std::vector<Permanent*> *permanentCardsPlayed;
00025         int maxNbAttack;
00026
00027     public:
00028         int HP;
00029         int honorPoints;
00030         bool asAttacked;
00031         int nbAttack;
00039         Player(Role *role, Character *character);
00040
00046         Role* getRole() const;
00047
00053         Character* getCharacter() const;
00054
00060         std::vector<Card*>* getHand() const;
00061
00067         std::vector<Permanent*>* getPermanentCardsPlayed() const;
00068
00074         int getMaxNbAttack() const;
00075
00079         void meditationFunction();
00080
00085         int attackRapideFunction();
00086
00091         std::pair<Permanent*, int> asCodeDuBushido();
00092
00097         int armureFunction();
00098
00102         void concentrationFunction();
00103
00108         bool isDown() const;
00109
00113         void recover();
00114
00118         ~Player();
00119 };
00120
00121 #endif // PLAYER_H
```

5.22 src/Role/Role.h File Reference

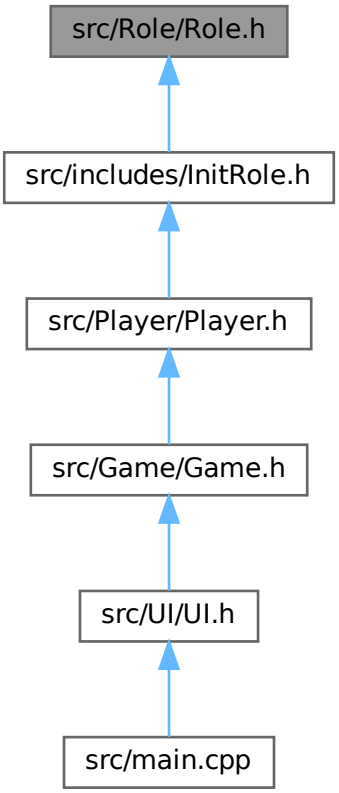
This file contains the declaration of the Role class and the enum class RoleType.

```
#include <string>
```

Include dependency graph for Role.h:



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



Classes

- class Role

  The Role class represents a role in the game.

Enumerations

- enum class RoleType { SHOGUN , SAMURAI , NINJA , RONIN }

  Represents the type of a Role.

5.22.1 Detailed Description

This file contains the declaration of the Role class and the enum class RoleType.

5.22.2 Enumeration Type Documentation

5.22.2.1 RoleType

```
enum class RoleType  [strong]
```

Represents the type of a Role.

Note

The type of a Role is also the index of the Role in the Image List and as the name of the Role

Enumerator

SHOGUN	SHOGUN role type
SAMURAI	SAMURAI role type
NINJA	NINJA role type
RONIN	RONIN role type

5.23 Role.h

Go to the documentation of this file.

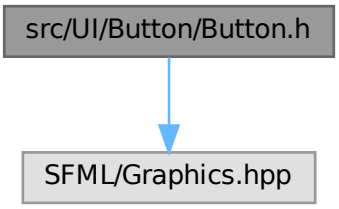
```
00001
00006 #ifndef ROLE_H
00007 #define ROLE_H
00008
00009 #include <string>
00010
00017 enum class RoleType {
00018     SHOGUN,
00019     SAMURAI,
00020     NINJA,
00021     RONIN
00022 };
00023
00031 class Role {
00032     private:
00033         RoleType type;
00034         int level;
00036     public:
00043         Role(RoleType type, int level = 0);
00044
00050         RoleType getType() const;
00051
00057         int getLevel() const;
00058
```

```
00064         int getIndex() const;
00065
00071         std::string getName() const;
00072 };
00073
00074 #endif // ROLE_H
```

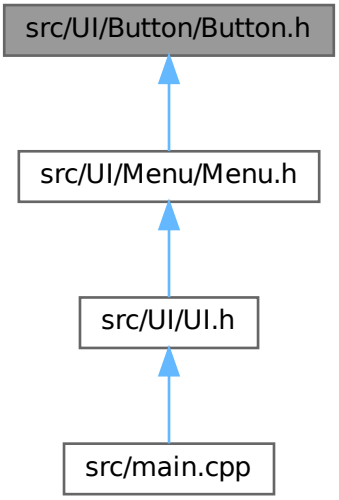
5.24 src/UI/Button/Button.h File Reference

This file contains the declaration of the Button class.

`#include <SFML/Graphics.hpp>`
Include dependency graph for Button.h:



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



Classes

- class Button

  Represents a graphical button element.

5.24.1 Detailed Description

This file contains the declaration of the Button class.

5.25 Button.h

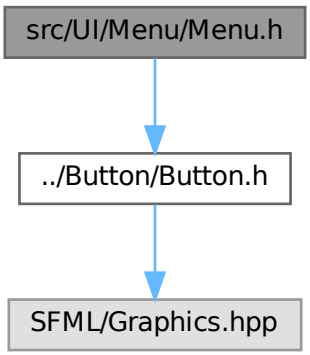
Go to the documentation of this file.

```
00001
00006 #ifndef BUTTON_H
00007 #define BUTTON_H
00008
00009 #include <SFML/Graphics.hpp>
00010
00015 class Button {
00016     private:
00017         sf::RectangleShape shape;
00018         sf::Text text;
00020     public:
00029         Button(sf::Vector2f position, sf::Vector2f size, sf::Font *font, std::string text, int
      fontSize);
00030
00035         sf::RectangleShape getShape() const;
00036
00042         bool isHovered(sf::RenderWindow &window) const;
00043
00049         bool isClicked(sf::RenderWindow &window) const;
00050
00055         void draw(sf::RenderWindow &window);
00056 };
00057
00058 #endif // BUTTON_H
```

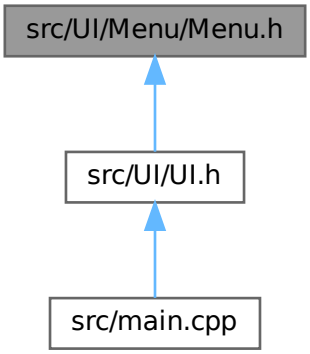
5.26 src/UI/Menu/Menu.h File Reference

This file contains the declaration of the Menu class and of the constants SCREEN_WIDTH and SCREEN_HEIGHT.

`#include "../Button/Button.h"`
Include dependency graph for Menu.h:



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



Classes

- class Menu

  Represents a menu in the game.

Macros

- #define SCREEN_WIDTH 1920
- #define SCREEN_HEIGHT 1080

5.26.1 Detailed Description

This file contains the declaration of the Menu class and of the constants SCREEN_WIDTH and SCREEN_HEIGHT.

5.26.2 Macro Definition Documentation

5.26.2.1 SCREEN_HEIGHT

```
#define SCREEN_HEIGHT 1080
```

Height of the window

5.26.2.2 SCREEN_WIDTH

```
#define SCREEN_WIDTH 1920
```

Width of the window

5.27 Menu.h

Go to the documentation of this file.

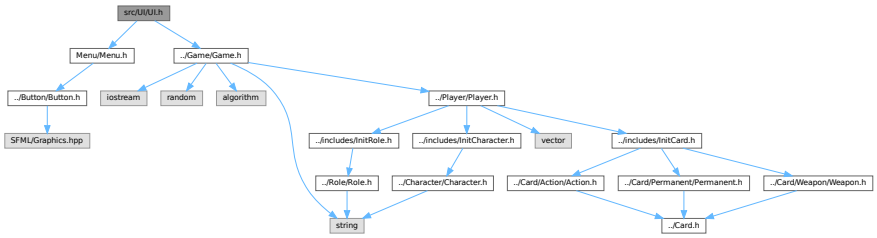
```
00001
00006 #ifndef MENU_H
00007 #define MENU_H
00008
00009 #include "../Button/Button.h"
00010
00011 #define SCREEN_WIDTH 1920
00012 #define SCREEN_HEIGHT 1080
00021 class Menu {
00022     private:
00023         sf::Font *font;
00024         sf::Text *title;
00025         std::vector<Button*> *buttons;
00026         sf::Texture *leftTexture;
00027         sf::Texture *rightTexture;
00028         sf::Sprite *leftSprite;
00029         sf::Sprite *rightSprite;
00030         int nbPlayers;
00032     public:
00040         Menu(sf::Font *font, sf::Image *leftImage, sf::Image *rightImage);
00041
00047         int getNbPlayers() const;
00048
00054         void display(sf::RenderWindow &window);
00055
00059         ~Menu();
00060 };
00061
00062 #endif // MENU_H
```

5.28 src/UI/UI.h File Reference

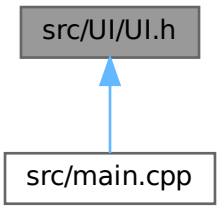
This file contains the declaration of the UI class.

```
#include "Menu/Menu.h"
#include "../Game/Game.h"
```

Include dependency graph for UI.h:



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



Classes

- class UI

  Represents the user interface of the game.

Macros

- #define FPS 60

5.28.1 Detailed Description

This file contains the declaration of the UI class.

5.28.2 Macro Definition Documentation

5.28.2.1 FPS

```
#define FPS 60
```

Frames per second

5.29 UI.h

Go to the documentation of this file.

```
00001
00006 #ifndef UI_H
00007 #define UI_H
00008
00009 #include "Menu/Menu.h"
00010 #include "../Game/Game.h"
00011
00012 #define FPS 60
00022 class UI {
00023     private:
00024         sf::RenderWindow *window;
00025         sf::Font *font;
```

```
00027         sf::Texture *HPTexture;
00028         sf::Texture *HonorTexture;
00029         sf::Texture *backRoleTexture;
00030         sf::Texture *backCardTexture;
00031         std::vector<sf::Texture*> *roleTextures;
00032         std::vector<sf::Texture*> *cardTextures;
00033         std::vector<sf::Texture*> *characterTextures;
00035         sf::Sprite *backRoleSprite;
00036         std::vector<sf::Sprite*> *playersSprites;
00037         std::vector<sf::Sprite*> *HPSprites;
00038         std::vector<sf::Text*> *HPTexts;
00039         std::vector<sf::Sprite*> *HonorSprites;
00040         std::vector<sf::Text*> *HonorTexts;
00042         sf::Sprite *actualPlayerSprite;
00043         sf::Sprite *actualPlayerRoleSprite;
00044         std::vector<sf::Sprite*> *actualPlayerCardSprites;
00045         sf::Sprite *HPSpritePlayer;
00046         sf::Text *HPTextPlayer;
00047         sf::Sprite *HonorSpritePlayer;
00048         sf::Text *HonorTextPlayer;
00050         std::vector<sf::Sprite*> *stackSprites;
00051         std::vector<sf::Sprite*> *discardStackSprites;
00053         std::vector<Player*> *players;
00054         std::vector<Card*> *hand;
00055         std::vector<Card*> *stack;
00056         std::vector<Card*> *discardStack;
00058         std::vector<std::string*> *logs;
00059         std::vector<sf::Text*> *logsTexts;
00060         sf::RectangleShape *logsBackground;
00061         sf::Text *openLogsText;
00062         sf::RectangleShape *openLogsBtn;
00063         sf::Text *closeLogsText;
00064         sf::RectangleShape *closeLogsBtn;
00065         bool isOpenLogsText;
00067         Button *passParadeBtn;
00068         Button *endTurnBtn;
00069         Button *discardingBtn;
00070         bool isDiscarding;
00072         Game *game;
00073         int indexActualPlayer;
00074         bool blocking;
00075         int nbPlayers;
00076         int spriteShogunIndex;
00077         int indexSelectedCard;
00078         int indexSelectedPlayer;
00080         int attackRapideNbDamage;
00082     public:
00095         UI(sf::Font *font, sf::Image *HPImage, sf::Image *HonorImage, sf::Image *backRoleImage,
      sf::Image *backCardImage, std::vector<sf::Image*> *roleImages, std::vector<sf::Image*> *cardImages,
      std::vector<sf::Image*> *characterImages);
00096
00102         void menu(sf::Image *leftImage, sf::Image *rightImage);
00103
00107         void start();
00108
00112         void update();
00113
00117         void display();
00118
00122         void setPlayersSprites();
00123
00128         void setSpriteHonorCharactersHP(int index);
00129
00133         void setActualPlayerSprite();
00134
00138         void setHandSprite();
00139
00143         void setStackSprite();
00144
00148         void setDiscardStackSprite();
00149
00153         void setLogsTexts();
00154
00158         void setPermanentCircleSprites();
00159
00164         void handleClickLogBtn(sf::Event event);
00165
00170         void handleClickHandCard(sf::Event event);
00171
00176         void handleClickPassParadeBtn(sf::Event event);
00177
00182         void handleClickEndTurnBtn(sf::Event event);
00183
00188         void handleClickDiscardingBtn(sf::Event event);
00189
00194         void handleClickNobunaga(sf::Event event);
00195
```

```
00199         void score();
00200
00204         ~UI();
00205 };
00206
00207 #endif // UI_H
```

Index