

Wizard Poker

0.0.1

Généré par Doxygen 1.8.10



# Table des matières

<b>1</b>	<b>Index des espaces de nommage</b>	<b>1</b>
1.1	Liste des espaces de nommage . . . . .	1
<b>2</b>	<b>Index hiérarchique</b>	<b>3</b>
2.1	Hiérarchie des classes . . . . .	3
<b>3</b>	<b>Index des classes</b>	<b>5</b>
3.1	Liste des classes . . . . .	5
<b>4</b>	<b>Index des fichiers</b>	<b>7</b>
4.1	Liste des fichiers . . . . .	7
<b>5</b>	<b>Documentation des espaces de nommage</b>	<b>9</b>
5.1	Référence de l'espace de nommage PacketManager . . . . .	9
5.1.1	Documentation des fonctions . . . . .	9
5.1.1.1	askDefausse(Player *, int) . . . . .	9
5.1.1.2	collectionResult(const Packet : :collectionListPacket *) . . . . .	9
5.1.1.3	initGame(Player *, std : :string) . . . . .	9
5.1.1.4	loginResult(const Packet : :loginResultPacket *) . . . . .	9
5.1.1.5	makeLoginRequest(const char *, const char *) . . . . .	9
5.1.1.6	makeRegistrationRequest(const char *, const char *) . . . . .	9
5.1.1.7	manageCollectionRequest(Player *, Packet : :packet *) . . . . .	9
5.1.1.8	manageDisconnectRequest(Player *, Packet : :packet *) . . . . .	9
5.1.1.9	managePacket(Packet : :packet *) . . . . .	9
5.1.1.10	managePacket(Player *, Packet : :packet *) . . . . .	9
5.1.1.11	requestCollection() . . . . .	10
5.1.1.12	sendAttack(Player *, std : :string, int, int, bool, bool, unsigned int) . . . . .	10
5.1.1.13	sendCard(Player *, Card *) . . . . .	10
5.1.1.14	sendDisconnection() . . . . .	10
5.1.1.15	sendEndGame(Player *, bool) . . . . .	10
5.1.1.16	sendInfoStartTurn(Player *, int, int) . . . . .	10
5.1.1.17	setTurn(Player *, std : :string) . . . . .	10
5.2	Référence de l'espace de nommage WizardLogger . . . . .	10

5.2.1	Documentation des fonctions . . . . .	10
5.2.1.1	error(std : :string) . . . . .	10
5.2.1.2	error(std : :string, std : :string) . . . . .	10
5.2.1.3	fatal(std : :string) . . . . .	10
5.2.1.4	fatal(std : :string, std : :string) . . . . .	11
5.2.1.5	info(std : :string) . . . . .	11
5.2.1.6	initLogger(bool, std : :string) . . . . .	11
5.2.1.7	warning(std : :string) . . . . .	11
<b>6</b>	<b>Documentation des classes</b>	<b>13</b>
6.1	Référence de la classe Card . . . . .	13
6.1.1	Description détaillée . . . . .	13
6.1.2	Documentation des constructeurs et destructeur . . . . .	13
6.1.2.1	Card(int id, bool isMonster, std : :string name, std : :string description, std : :size← _t energy, int HP) . . . . .	14
6.1.2.2	~Card()=default . . . . .	14
6.1.2.3	Card(unsigned int id, std : :string name, unsigned int energy, int effect, bool) . . . .	14
6.1.2.4	Card(Card &card)=default . . . . .	14
6.1.2.5	~Card() . . . . .	14
6.1.3	Documentation des fonctions membres . . . . .	14
6.1.3.1	applyEffect(CardMonster &cardmonster, Game &) . . . . .	14
6.1.3.2	applyEffect(PlayerInGame &player, Game &) . . . . .	14
6.1.3.3	canBeApplyOnCard() . . . . .	14
6.1.3.4	canBeApplyOnPlayer() . . . . .	15
6.1.3.5	getDescription() . . . . .	15
6.1.3.6	getEffectID() . . . . .	15
6.1.3.7	getEnergyCost() . . . . .	15
6.1.3.8	getEnergyCost() . . . . .	15
6.1.3.9	getHP() . . . . .	15
6.1.3.10	getId() . . . . .	15
6.1.3.11	getId() const . . . . .	15
6.1.3.12	getMaxHP() . . . . .	15
6.1.3.13	getName() . . . . .	15
6.1.3.14	getName() const . . . . .	15
6.1.3.15	gotEffect() . . . . .	15
6.1.3.16	isMonster() . . . . .	15
6.1.3.17	isMonster() . . . . .	15
6.1.3.18	operator=(const Card &)=default . . . . .	15
6.2	Référence de la classe CardManager . . . . .	16
6.2.1	Documentation des fonctions membres . . . . .	16

6.2.1.1	chooseCardWin()	16
6.2.1.2	getCardById(unsigned int id)	16
6.2.1.3	loadAllCards()	16
6.3	Référence de la classe CardMonster	16
6.3.1	Documentation des constructeurs et destructeur	17
6.3.1.1	CardMonster(unsigned int id, std::string name, unsigned int energy, int effect, bool, unsigned int life, unsigned int attack, unsigned int nbrTour=0)	17
6.3.1.2	CardMonster(CardMonster &otherMonster)	17
6.3.1.3	~CardMonster()	17
6.3.2	Documentation des fonctions membres	17
6.3.2.1	dealDamage(CardMonster &otherMonster)	17
6.3.2.2	dealDamage(PlayerInGame &player)	17
6.3.2.3	getAttack()	18
6.3.2.4	getLife()	18
6.3.2.5	getMaxLife()	18
6.3.2.6	getNbrTourPose()	18
6.3.2.7	incrementTour()	18
6.3.2.8	isDead()	18
6.3.2.9	isMonster() override	18
6.3.2.10	isTaunt()	18
6.3.2.11	setAttack(unsigned int newAttack)	18
6.3.2.12	setLife(unsigned int newLife)	18
6.3.2.13	setMaxLife(unsigned int newMax)	18
6.3.2.14	setTaunt(bool value)	18
6.4	Référence de la structure Packet : :carteInfosPacket	18
6.4.1	Documentation des données membres	19
6.4.1.1	carteID	19
6.4.1.2	cartesDescription	19
6.4.1.3	ID	19
6.4.1.4	size	19
6.5	Référence de la structure Packet : :carteRequestPacket	19
6.5.1	Documentation des données membres	19
6.5.1.1	carteID	19
6.5.1.2	ID	19
6.5.1.3	size	19
6.6	Référence de la classe ChatManager	19
6.6.1	Documentation des fonctions membres	19
6.6.1.1	sendMessage(Player player, std::string message)	19
6.7	Référence de la classe CLI	20
6.7.1	Documentation des constructeurs et destructeur	20

6.7.1.1	CLI()	20
6.7.1.2	~CLI()	20
6.7.2	Documentation des fonctions membres	20
6.7.2.1	displayCollectionWindow()	20
6.7.2.2	displayDeckWindow()	20
6.7.2.3	displayFatalError(std : :string)	20
6.7.2.4	displayFriendsWindow()	20
6.7.2.5	displayGame()	20
6.7.2.6	displayLoginPrompt()	21
6.7.2.7	displayLoginResult(std : :string)	21
6.7.2.8	displayMainWindow()	21
6.7.2.9	displayWait()	21
6.7.2.10	focusTchat()	21
6.7.2.11	valideLogin()	21
6.8	Référence de la classe Collection	21
6.8.1	Documentation des constructeurs et destructeur	22
6.8.1.1	Collection()=default	22
6.8.1.2	Collection(std : :vector< Card * >)	22
6.8.1.3	Collection(std : :vector< unsigned >)	22
6.8.1.4	Collection(const Collection &)=default	22
6.8.1.5	~Collection()=default	22
6.8.2	Documentation des fonctions membres	22
6.8.2.1	addCard(Card *)	22
6.8.2.2	addCard(int cardId)	22
6.8.2.3	getCardIndex(Card *)	23
6.8.2.4	getCardOnIndex(const unsigned index)	23
6.8.2.5	getCardsId() const	23
6.8.2.6	indexOfCard(int cardId)	23
6.8.2.7	operator=(const Collection &)=default	23
6.8.2.8	removeCard(int i)	23
6.8.2.9	removeCard(Card *)	23
6.8.2.10	removeCardId(int cardId)	23
6.8.3	Documentation des données membres	23
6.8.3.1	_listCard	24
6.9	Référence de la structure Packet : :collectionListPacket	24
6.9.1	Documentation des données membres	24
6.9.1.1	cartesList	24
6.9.1.2	ID	24
6.9.1.3	size	24
6.10	Référence de la classe Connection	24

6.10.1	Documentation des constructeurs et destructeur	24
6.10.1.1	Connection(char *)	24
6.10.1.2	~Connection()	24
6.10.1.3	Connection()	24
6.10.1.4	~Connection()	24
6.10.2	Documentation des fonctions membres	24
6.10.2.1	mainLoop()	24
6.10.2.2	sendPacket(Packet *, size_t)	25
6.11	Référence de la structure dataIGPlayer	25
6.11.1	Documentation des données membres	25
6.11.1.1	cardsInHand	25
6.11.1.2	cardsPlaced	25
6.11.1.3	energy	25
6.11.1.4	limitEnergy	25
6.11.1.5	maxEnergy	25
6.11.1.6	playerHeal	25
6.11.1.7	turn	25
6.12	Référence de la classe Deck	25
6.12.1	Description détaillée	26
6.12.2	Documentation des constructeurs et destructeur	26
6.12.2.1	Deck(std : :string name, std : :vector< Card * > listCard)	26
6.12.2.2	Deck(std : :string name, std : :vector< unsigned > listCard)	26
6.12.2.3	Deck(const Deck &)	26
6.12.3	Documentation des fonctions membres	26
6.12.3.1	addCard(Card *card) override	26
6.12.3.2	addCard(int cardId) override	27
6.12.3.3	copyDeck()	27
6.12.3.4	deleteDeck(PlayerInGame *)	27
6.12.3.5	getDeck(std : :string, std : :vector< Deck * >)	27
6.12.3.6	getName() const	27
6.12.3.7	isValide()	27
6.12.3.8	operator=(const Deck &)=default	28
6.12.3.9	operator==(const std : :string &) const	28
6.12.3.10	pickup()	28
6.13	Référence de la structure Packet : :deckContentPacket	28
6.13.1	Documentation des données membres	28
6.13.1.1	cartesList	28
6.13.1.2	deckID	28
6.13.1.3	ID	28
6.13.1.4	size	28

6.14	Référence de la structure Packet : :deckRequestPacket . . . . .	28
6.14.1	Documentation des données membres . . . . .	28
6.14.1.1	deckID . . . . .	28
6.14.1.2	ID . . . . .	29
6.14.1.3	size . . . . .	29
6.15	Référence de la classe Display . . . . .	29
6.15.1	Documentation des constructeurs et destructeur . . . . .	29
6.15.1.1	Display()=default . . . . .	29
6.15.1.2	~Display()=default . . . . .	29
6.15.2	Documentation des fonctions membres . . . . .	29
6.15.2.1	displayCollectionWindow()=0 . . . . .	29
6.15.2.2	displayDeckWindow()=0 . . . . .	29
6.15.2.3	displayFatalError(std : :string)=0 . . . . .	29
6.15.2.4	displayFriendsWindow()=0 . . . . .	30
6.15.2.5	displayGame()=0 . . . . .	30
6.15.2.6	displayLoginPrompt()=0 . . . . .	30
6.15.2.7	displayLoginResult(std : :string)=0 . . . . .	30
6.15.2.8	displayMainWindow()=0 . . . . .	30
6.15.2.9	displayWait()=0 . . . . .	30
6.15.2.10	focusTchat()=0 . . . . .	30
6.15.2.11	valideLogin()=0 . . . . .	30
6.16	Référence de la classe Effect . . . . .	30
6.16.1	Documentation des constructeurs et destructeur . . . . .	31
6.16.1.1	Effect() . . . . .	31
6.16.1.2	~Effect() . . . . .	31
6.16.2	Documentation des fonctions membres . . . . .	31
6.16.2.1	apply(CardMonster *, Game *)=0 . . . . .	31
6.16.2.2	apply(PlayerInGame *player, Game *game) . . . . .	31
6.16.2.3	canBeApplyOnCard() . . . . .	31
6.16.2.4	canBeApplyOnPlayer() . . . . .	31
6.16.2.5	getEffectByID(unsigned) . . . . .	31
6.16.2.6	getId() . . . . .	31
6.16.2.7	isTaunt() . . . . .	31
6.16.2.8	loadAllEffect() . . . . .	31
6.17	Référence de la classe FriendsManager . . . . .	31
6.17.1	Description détaillée . . . . .	31
6.18	Référence de la classe Game . . . . .	32
6.18.1	Description détaillée . . . . .	32
6.18.2	Documentation des constructeurs et destructeur . . . . .	32
6.18.2.1	Game() . . . . .	32



6.18.2.2	Game(const Game &)	32
6.18.2.3	~Game()=default	32
6.18.3	Documentation des fonctions membres	32
6.18.3.1	addPlayerWaitGame(Player player)	32
6.18.3.2	attackWithCard(PlayerInGame *, CardMonster *, CardMonster *)	33
6.18.3.3	attackWithCardAffectPlayer(PlayerInGame *, CardMonster *)	33
6.18.3.4	checkDeckAndStart()	33
6.18.3.5	draw()	33
6.18.3.6	draw(PlayerInGame *)	33
6.18.3.7	nextPlayer()	33
6.18.3.8	operator=(const Game &)=default	34
6.18.3.9	placeCard(PlayerInGame *, Card *, CardMonster *)	34
6.18.3.10	placeCardAffectPlayer(PlayerInGame *, Card *)	35
6.18.3.11	sendInfoAction(PlayerInGame *, int, int, bool, bool, unsigned)	35
6.19	Référence de la structure Packet : :loginRequestPacket	35
6.19.1	Documentation des données membres	35
6.19.1.1	ID	35
6.19.1.2	password	35
6.19.1.3	pseudo	35
6.19.1.4	size	36
6.20	Référence de la structure Packet : :loginResultPacket	36
6.20.1	Documentation des données membres	36
6.20.1.1	ID	36
6.20.1.2	resultCode	36
6.20.1.3	size	36
6.21	Référence de la classe Packet	36
6.21.1	Documentation des énumérations membres	37
6.21.1.1	IDList	37
6.21.2	Documentation des données membres	37
6.21.2.1	packetMaxSize	37
6.21.2.2	packetSize	37
6.22	Référence de la structure Packet : :packet	37
6.22.1	Documentation des données membres	37
6.22.1.1	ID	37
6.22.1.2	size	37
6.23	Référence de la classe Player	38
6.23.1	Description détaillée	38
6.23.2	Documentation des constructeurs et destructeur	38
6.23.2.1	Player(nlohmann : :json &, int sockfd=0)	38
6.23.2.2	~Player()	39

6.23.3	Documentation des fonctions membres	39
6.23.3.1	addCardCollection(Card *c)	39
6.23.3.2	adjudicateDefeat()	39
6.23.3.3	adjudicateVictory()	39
6.23.3.4	getCollection()	39
6.23.3.5	getDeck(std : :string)	39
6.23.3.6	getDefeats() const	39
6.23.3.7	getListDeck()	39
6.23.3.8	getName() const	39
6.23.3.9	getPass() const	39
6.23.3.10	getVictories() const	39
6.23.3.11	logout()	39
6.23.3.12	operator<(const Player &) const	39
6.23.3.13	operator==(const std : :string &) const	39
6.23.3.14	operator>(const Player &) const	39
6.23.3.15	recvLoop()	39
6.23.3.16	removeDeck(Deck *)	39
6.23.3.17	sendPacket(Packet : :packet *, size_t)	40
6.23.3.18	serialise() const	40
6.23.3.19	updateSockfd(int a)	40
6.23.4	Documentation des fonctions amies et associées	40
6.23.4.1	operator<<	40
6.23.4.2	operator<<	40
6.23.5	Documentation des données membres	40
6.23.5.1	_defeats	40
6.23.5.2	_victories	40
6.24	Référence de la classe PlayerInGame	40
6.24.1	Documentation des constructeurs et destructeur	41
6.24.1.1	PlayerInGame(const PlayerInGame &)=default	41
6.24.1.2	PlayerInGame(const Player &, Game *)	41
6.24.1.3	~PlayerInGame()=default	41
6.24.2	Documentation des fonctions membres	41
6.24.2.1	addDefeat()	41
6.24.2.2	addMaxEnergy()	41
6.24.2.3	addWin()	41
6.24.2.4	defausseCardPlaced(CardMonster *)	41
6.24.2.5	draw()	41
6.24.2.6	getCardsInHand()	41
6.24.2.7	getCardsPlaced()	41
6.24.2.8	getDataPlayer()	41

6.24.2.9	getHeal()	42
6.24.2.10	getHealed(unsigned int)	42
6.24.2.11	haveEnoughEnergy(Card *card)	42
6.24.2.12	isDead()	42
6.24.2.13	isDeckDefined()	42
6.24.2.14	nbrCardInHand()	42
6.24.2.15	operator=(const PlayerInGame &)=default	42
6.24.2.16	placeCard(CardMonster *)	42
6.24.2.17	resetEnergy()	42
6.24.2.18	setDeck(Deck *deck)	42
6.24.2.19	takeDamage(unsigned int)	42
6.25	Référence de la classe PlayerManager	42
6.25.1	Documentation des constructeurs et destructeur	43
6.25.1.1	PlayerManager()=default	43
6.25.1.2	~PlayerManager()=default	43
6.25.2	Documentation des fonctions membres	43
6.25.2.1	getRanking()	43
6.25.2.2	loadPlayers()	43
6.25.2.3	logIn(std : :string, std : :string, int)	43
6.25.2.4	savePlayers() const	43
6.25.2.5	signUp(std : :string, std : :string, int)	43
<b>7</b>	<b>Documentation des fichiers</b>	<b>45</b>
7.1	Référence du fichier client/Card.hpp	45
7.2	Référence du fichier server/Card.hpp	45
7.3	Référence du fichier client/CLI.cpp	45
7.4	Référence du fichier client/CLI.hpp	46
7.4.1	Documentation des macros	46
7.4.1.1	AMIS_LABEL	46
7.4.1.2	CARD_HEIGHT	46
7.4.1.3	CARD_INFO_HEIGHT	46
7.4.1.4	CARD_INFO_WIDTH	46
7.4.1.5	CARD_WIDTH	46
7.4.1.6	COLL_LABEL	47
7.4.1.7	COLLONNES	47
7.4.1.8	COMMANDE_PANEL_HEIGHT	47
7.4.1.9	DECK_LABEL	47
7.4.1.10	DEFAULT_EMPTY_SPACE	47
7.4.1.11	GAME_HEIGHT	47
7.4.1.12	GAME_LABEL	47

7.4.1.13	GAME_WIDTH	47
7.4.1.14	LINES	47
7.4.1.15	MAIN_LABEL	47
7.4.1.16	PANEL_TOTAL_NUMBER	48
7.4.1.17	PLAYER_INFO_HEIGHT	48
7.4.1.18	PLAYER_INFO_WIDTH	48
7.4.1.19	TCHAT_HEIGHT	48
7.4.1.20	TCHAT_INPUT_HEIGHT	48
7.4.1.21	TCHAT_INPUT_WIDTH	48
7.4.1.22	TCHAT_WIDTH	48
7.4.1.23	WAIT_LABEL	48
7.5	Référence du fichier client/Connection.cpp	48
7.6	Référence du fichier server/Connection.cpp	48
7.7	Référence du fichier client/Connection.hpp	48
7.7.1	Documentation des macros	49
7.7.1.1	PORT	49
7.7.2	Documentation des variables	49
7.7.2.1	display	49
7.8	Référence du fichier server/Connection.hpp	49
7.8.1	Documentation des macros	49
7.8.1.1	BACKLOG	49
7.8.1.2	PORT	50
7.8.2	Documentation des variables	50
7.8.2.1	pm	50
7.9	Référence du fichier client/Display.hpp	50
7.10	Référence du fichier client/main.cpp	50
7.10.1	Documentation des fonctions	50
7.10.1.1	main(int argc, char **argv)	50
7.10.2	Documentation des variables	50
7.10.2.1	conn	50
7.10.2.2	display	50
7.11	Référence du fichier server/main.cpp	51
7.11.1	Documentation des fonctions	51
7.11.1.1	main()	51
7.11.2	Documentation des variables	51
7.11.2.1	pm	51
7.12	Référence du fichier client/PackageManager.cpp	51
7.12.1	Documentation des variables	51
7.12.1.1	conn	51
7.13	Référence du fichier server/PackageManager.cpp	51

7.14	Référence du fichier client/PackageManager.hpp . . . . .	52
7.14.1	Documentation des macros . . . . .	52
7.14.1.1	MAX_CARTES . . . . .	52
7.14.2	Documentation des variables . . . . .	52
7.14.2.1	display . . . . .	52
7.15	Référence du fichier server/PackageManager.hpp . . . . .	52
7.16	Référence du fichier common/Error.hpp . . . . .	53
7.16.1	Documentation du type de l'énumération . . . . .	53
7.16.1.1	Error . . . . .	53
7.17	Référence du fichier common/Packet.hpp . . . . .	53
7.17.1	Documentation des macros . . . . .	54
7.17.1.1	DECK_SIZE . . . . .	54
7.17.1.2	MAX_CARTE_DESCRIPTION_SIZE . . . . .	54
7.17.1.3	MAX_CARTES . . . . .	54
7.17.1.4	MAX_PSEUDO_SIZE . . . . .	54
7.18	Référence du fichier common/WizardLogger.cpp . . . . .	54
7.19	Référence du fichier common/WizardLogger.hpp . . . . .	54
7.19.1	Documentation des macros . . . . .	54
7.19.1.1	CLIENT_LOGFILE . . . . .	54
7.19.1.2	LOGGER . . . . .	54
7.19.1.3	SERVER_LOGFILE . . . . .	55
7.20	Référence du fichier server/Card.cpp . . . . .	55
7.21	Référence du fichier server/CardManager.cpp . . . . .	55
7.22	Référence du fichier server/CardManager.hpp . . . . .	55
7.22.1	Documentation des définitions de type . . . . .	55
7.22.1.1	json . . . . .	55
7.23	Référence du fichier server/CardMonster.cpp . . . . .	55
7.24	Référence du fichier server/CardMonster.hpp . . . . .	55
7.25	Référence du fichier server/ChatManager.hpp . . . . .	56
7.26	Référence du fichier server/Collection.cpp . . . . .	56
7.27	Référence du fichier server/Collection.hpp . . . . .	56
7.28	Référence du fichier server/Deck.cpp . . . . .	56
7.29	Référence du fichier server/Deck.hpp . . . . .	56
7.29.1	Documentation des macros . . . . .	57
7.29.1.1	LIMITNAME . . . . .	57
7.30	Référence du fichier server/Effect.cpp . . . . .	57
7.31	Référence du fichier server/Effect.hpp . . . . .	57
7.32	Référence du fichier server/FriendsManager.hpp . . . . .	57
7.33	Référence du fichier server/Game.cpp . . . . .	57
7.34	Référence du fichier server/Game.hpp . . . . .	58

7.34.1	Documentation du type de l'énumération . . . . .	58
7.34.1.1	GameStatut . . . . .	58
7.35	Référence du fichier server/Player.cpp . . . . .	58
7.35.1	Documentation des fonctions . . . . .	58
7.35.1.1	operator<<(std::ostream &os, const Player &c) . . . . .	58
7.35.1.2	operator<<(std::string &str, const Player &c) . . . . .	58
7.36	Référence du fichier server/Player.hpp . . . . .	59
7.37	Référence du fichier server/PlayerInGame.cpp . . . . .	59
7.38	Référence du fichier server/PlayerInGame.hpp . . . . .	59
7.39	Référence du fichier server/PlayerManager.cpp . . . . .	59
7.40	Référence du fichier server/PlayerManager.hpp . . . . .	60
7.40.1	Documentation des macros . . . . .	60
7.40.1.1	PLAYERS_DB . . . . .	60
<b>Index</b>		<b>61</b>

# Chapitre 1

## Index des espaces de nommage

### 1.1 Liste des espaces de nommage

Liste de tous les espaces de nommage avec une brève description :

<a href="#">PacketManager</a> . . . . .	9
<a href="#">WizardLogger</a> . . . . .	10





## Chapitre 2

# Index hiérarchique

### 2.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Card . . . . .	13
CardMonster . . . . .	16
CardManager . . . . .	16
Packet : :carteInfosPacket . . . . .	18
Packet : :carteRequestPacket . . . . .	19
ChatManager . . . . .	19
Collection . . . . .	21
Deck . . . . .	25
Packet : :collectionListPacket . . . . .	24
Connection . . . . .	24
dataIGPlayer . . . . .	25
Packet : :deckContentPacket . . . . .	28
Packet : :deckRequestPacket . . . . .	28
Display . . . . .	29
CLI . . . . .	20
Effect . . . . .	30
FriendsManager . . . . .	31
Game . . . . .	32
Packet : :loginRequestPacket . . . . .	35
Packet : :loginResultPacket . . . . .	36
Packet . . . . .	36
Packet : :packet . . . . .	37
Player . . . . .	38
PlayerInGame . . . . .	40
PlayerManager . . . . .	42



## Chapitre 3

# Index des classes

### 3.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Card	13
CardManager	16
CardMonster	16
Packet : :carteInfosPacket	18
Packet : :carteRequestPacket	19
ChatManager	19
CLI	20
Collection	21
Packet : :collectionListPacket	24
Connection	24
dataIGPlayer	25
Deck	25
Packet : :deckContentPacket	28
Packet : :deckRequestPacket	28
Display	29
Effect	30
FriendsManager	31
Game	32
Packet : :loginRequestPacket	35
Packet : :loginResultPacket	36
Packet	36
Packet : :packet	37
Player	38
PlayerInGame	40
PlayerManager	42



## Chapitre 4

# Index des fichiers

### 4.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

client/Card.hpp	45
client/CLI.cpp	45
client/CLI.hpp	46
client/Connection.cpp	48
client/Connection.hpp	48
client/Display.hpp	50
client/main.cpp	50
client/PackageManager.cpp	51
client/PackageManager.hpp	52
common/Error.hpp	53
common/Packet.hpp	53
common/WizardLogger.cpp	54
common/WizardLogger.hpp	54
server/Card.cpp	55
server/Card.hpp	45
server/CardManager.cpp	55
server/CardManager.hpp	55
server/CardMonster.cpp	55
server/CardMonster.hpp	55
server/ChatManager.hpp	56
server/Collection.cpp	56
server/Collection.hpp	56
server/Connection.cpp	48
server/Connection.hpp	49
server/Deck.cpp	56
server/Deck.hpp	56
server/Effect.cpp	57
server/Effect.hpp	57
server/FriendsManager.hpp	57
server/Game.cpp	57
server/Game.hpp	58
server/main.cpp	51
server/PackageManager.cpp	51
server/PackageManager.hpp	52
server/Player.cpp	58
server/Player.hpp	59
server/PlayerInGame.cpp	59
server/PlayerInGame.hpp	59

server/ <a href="#">PlayerManager.cpp</a>	59
server/ <a href="#">PlayerManager.hpp</a>	60

## Chapitre 5

# Documentation des espaces de nommage

### 5.1 Référence de l'espace de nommage PacketManager

#### Fonctions

- void `managePacket` (`Packet` : `:packet` \*)
- void `makeLoginRequest` (`const char` \*, `const char` \*)
- void `makeRegistrationRequest` (`const char` \*, `const char` \*)
- void `sendDisconnection` ()
- void `requestCollection` ()
- void `loginResult` (`const Packet` : `:loginResultPacket` \*)
- void `collectionResult` (`const Packet` : `:collectionListPacket` \*)
- void `manageDisconnectRequest` (`Player` \*, `Packet` : `:packet` \*)
- void `manageCollectionRequest` (`Player` \*, `Packet` : `:packet` \*)
- void `managePacket` (`Player` \*, `Packet` : `:packet` \*)
- void `initGame` (`Player` \*, `std` : `:string`)
- void `sendCard` (`Player` \*, `Card` \*)
- void `setTurn` (`Player` \*, `std` : `:string`)
- void `sendInfoStartTurn` (`Player` \*, `int`, `int`)
- void `sendAttack` (`Player` \*, `std` : `:string`, `int`, `int`, `bool`, `bool`, `unsigned int`)
- void `askDefausse` (`Player` \*, `int`)
- void `sendEndGame` (`Player` \*, `bool`)

#### 5.1.1 Documentation des fonctions

5.1.1.1 void `PacketManager` : `:askDefausse` ( `Player` \* *player*, `int` *amount* )

5.1.1.2 void `PacketManager` : `:collectionResult` ( `const Packet` : `:collectionListPacket` \* *collectionPacket* )

5.1.1.3 void `PacketManager` : `:initGame` ( `Player` \* *player*, `std` : `:string` *enemyPseudo* )

5.1.1.4 void `PacketManager` : `:loginResult` ( `const Packet` : `:loginResultPacket` \* *resultPacket* )

5.1.1.5 void `PacketManager` : `:makeLoginRequest` ( `const char` \* *pseudo*, `const char` \* *password* )

5.1.1.6 void `PacketManager` : `:makeRegistrationRequest` ( `const char` \* *pseudo*, `const char` \* *password* )

5.1.1.7 void `PacketManager` : `:manageCollectionRequest` ( `Player` \* *player*, `Packet` : `:packet` \* *collectionReqPacket* )

5.1.1.8 void `PacketManager` : `:manageDisconnectRequest` ( `Player` \* *player*, `Packet` : `:packet` \* *disconnectReqPacket* )

5.1.1.9 void `PacketManager` : `:managePacket` ( `Packet` : `:packet` \* *customPacket* )

5.1.1.10 void `PacketManager` : `:managePacket` ( `Player` \* *player*, `Packet` : `:packet` \* *customPacket* )

5.1.1.11 void PacketManager : :requestCollection ( )

5.1.1.12 void PacketManager : :sendAttack ( Player \* *player*, std : :string *pseudo*, int *cardWichAttack*, int *targetID*, bool *isEffect*, bool *newCard*, unsigned int *finalLife* )

Send attack informations

Paramètres

<i>player</i>	: the players who to send this packet
<i>pseudo</i>	: attacking player's pseudo
<i>targetID</i>	: ID of the target (-1 for player, other for cardMonster)
<i>cardWichAttack</i>	: <a href="#">Card</a> which attack the other
<i>isEffect</i>	: if the attack is an effect (false if it is attack)
<i>newCard</i>	: true if it's a new card wich make attack
<i>finalLife</i>	: final life of the target after attack

5.1.1.13 void PacketManager : :sendCard ( Player \* *player*, Card \* *card* )

5.1.1.14 void PacketManager : :sendDisconnection ( )

5.1.1.15 void PacketManager : :sendEndGame ( Player \* *player*, bool *victory* )

5.1.1.16 void PacketManager : :sendInfoStartTurn ( Player \* *player*, int *energy*, int *adverseCardInHand* )

Send informations when the player start he turn

Paramètres

<i>player</i>	: the players who to send this packet
<i>energy</i>	: energy that have the player (it is egal at he max energy)
<i>adverseCardInHand</i>	: number of card that the adverse player have

5.1.1.17 void PacketManager : :setTurn ( Player \* *player*, std : :string *pseudo* )

## 5.2 Référence de l'espace de nommage WizardLogger

### Fonctions

- void [initLogger](#) (bool, std : :string)
- void [info](#) (std : :string)
- void [warning](#) (std : :string)
- void [error](#) (std : :string)
- void [error](#) (std : :string, std : :string)
- void [fatal](#) (std : :string)
- void [fatal](#) (std : :string, std : :string)

### 5.2.1 Documentation des fonctions

5.2.1.1 void WizardLogger : :error ( std : :string *message* )

5.2.1.2 void WizardLogger : :error ( std : :string *message*, std : :string *ex* )

5.2.1.3 void WizardLogger : :fatal ( std : :string *message* )



5.2.1.4 void WizardLogger : :fatal ( std : :string *message*, std : :string *ex* )

5.2.1.5 void WizardLogger : :info ( std : :string *message* )

5.2.1.6 void WizardLogger : :initLogger ( bool *useConsole*, std : :string *logFileName* )

5.2.1.7 void WizardLogger : :warning ( std : :string *message* )



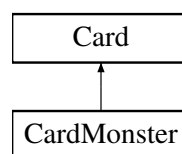
## Chapitre 6

# Documentation des classes

### 6.1 Référence de la classe Card

```
#include <Card.hpp>
```

Graphe d'héritage de Card :



#### Fonctions membres publiques

- `Card` (int id, bool `isMonster`, std : :string name, std : :string description, std : :size\_t energy, int HP)
- `~Card` ()=default
- int `getId` ()
- bool `isMonster` ()
- std : :string `getName` ()
- std : :string `getDescription` ()
- int `getEnergyCost` ()
- int `getMaxHP` ()
- int `getHP` ()
- `Card` (unsigned int id, std : :string name, unsigned int energy, int effect, bool)
- `Card` (`Card` &card)=default
- `Card` & `operator=` (const `Card` &)=default
- virtual void `applyEffect` (`CardMonster` &cardmonster, `Game` &)
- virtual void `applyEffect` (`PlayerInGame` &player, `Game` &)
- virtual `~Card` ()
- unsigned int `getId` () const
- std : :string `getName` () const
- unsigned int `getEnergyCost` ()
- bool `gotEffect` ()
- int `getEffectID` ()
- virtual bool `isMonster` ()
- virtual bool `canBeApplyOnCard` ()
- virtual bool `canBeApplyOnPlayer` ()

#### 6.1.1 Description détaillée

One class per card. Contain all information of the card

#### 6.1.2 Documentation des constructeurs et destructeur

6.1.2.1 `Card::Card ( int id, bool isMonster, std::string name, std::string description, std::size_t energy, int HP )`  
`[inline]`

6.1.2.2 `Card::~~Card ( )` `[default]`

6.1.2.3 `Card::Card ( unsigned int id, std::string name, unsigned int energy, int effect, bool save = true )`

Constructor

Paramètres

<i>id</i>	of the card
<i>name</i>	of the card
<i>energy</i>	of the card
<i>effect</i>	name of the specific effect
<i>save</i>	True if save in cache

6.1.2.4 `Card::Card ( Card & card )` `[default]`

6.1.2.5 `virtual Card::~~Card ( )` `[virtual]`

### 6.1.3 Documentation des fonctions membres

6.1.3.1 `void Card::applyEffect ( CardMonster & cardmonster, Game & game )` `[virtual]`

Apply the effect on a monster

Paramètres

<i>the</i>	monster targeted
<i>the</i>	game where the effect will be apply

Renvoie

void

6.1.3.2 `void Card::applyEffect ( PlayerInGame & player, Game & game )` `[virtual]`

Apply the effect on a player

Paramètres

<i>the</i>	player targeted
<i>the</i>	game where the effect will be apply

Renvoie

void

6.1.3.3 `bool Card::canBeApplyOnCard ( )` `[virtual]`

Check if the effect can be apply on a player

Renvoie

true if yes, false if not

6.1.3.4 `bool Card::canBeApplyOnPlayer( ) [virtual]`

Check if the effect can be apply on a player

Renvoie

true if yes, false if not

6.1.3.5 `std::string Card::getDescription( ) [inline]`

6.1.3.6 `int Card::getEffectID( )`

Return the effect id

Renvoie

-1 if no effect or the effect id

6.1.3.7 `int Card::getEnergyCost( ) [inline]`

6.1.3.8 `unsigned int Card::getEnergyCost( ) [inline]`

6.1.3.9 `int Card::getHP( ) [inline]`

6.1.3.10 `int Card::getId( ) [inline]`

6.1.3.11 `unsigned int Card::getId( ) const [inline]`

6.1.3.12 `int Card::getMaxHP( ) [inline]`

6.1.3.13 `std::string Card::getName( ) [inline]`

6.1.3.14 `std::string Card::getName( ) const [inline]`

6.1.3.15 `bool Card::gotEffect( )`

Check if the card got an effect

Renvoie

true if yes, false if not

6.1.3.16 `bool Card::isMonster( ) [inline]`

6.1.3.17 `virtual bool Card::isMonster( ) [inline],[virtual]`

Réimplémentée dans [CardMonster](#).

6.1.3.18 `Card& Card::operator=( const Card & ) [default]`

La documentation de cette classe a été générée à partir des fichiers suivants :

- [client/Card.hpp](#)
- [server/Card.cpp](#)

## 6.2 Référence de la classe CardManager

```
#include <CardManager.hpp>
```

### Fonctions membres publiques statiques

- static [Card](#) \* [getCardById](#) (unsigned int id)
- static void [loadAllCards](#) ()
- static [Card](#) \* [chooseCardWin](#) ()

### 6.2.1 Documentation des fonctions membres

#### 6.2.1.1 [Card](#) \* [CardManager](#) : :[chooseCardWin](#) ( ) [static]

Select a random card

Renvoie

[Card](#)\*

#### 6.2.1.2 [Card](#) \* [CardManager](#) : :[getCardById](#) ( unsigned int *id* ) [static]

#### 6.2.1.3 void [CardManager](#) : :[loadAllCards](#) ( ) [static]

Create and keep all the cards in a dictionnary witj the id as key

Renvoie

void

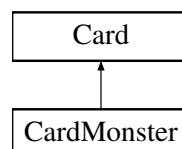
La documentation de cette classe a été générée à partir des fichiers suivants :

- server/[CardManager.hpp](#)
- server/[CardManager.cpp](#)

## 6.3 Référence de la classe CardMonster

```
#include <CardMonster.hpp>
```

Graphe d'héritage de CardMonster :



### Fonctions membres publiques

- virtual void [dealDamage](#) ([CardMonster](#) &otherMonster)
- virtual void [dealDamage](#) ([PlayerInGame](#) &player)
- virtual void [incrementTour](#) ()
- [CardMonster](#) (unsigned int id, std : :string name, unsigned int energy, int effect, bool, unsigned int life, unsigned int attack, unsigned int nbrTour=0)
- [CardMonster](#) ([CardMonster](#) &otherMonster)
- virtual ~[CardMonster](#) ()
- unsigned int [getLife](#) ()
- unsigned int [getAttack](#) ()

- unsigned int [getMaxLife](#) ()
- unsigned int [getNbrTourPose](#) ()
- virtual bool [isMonster](#) () override
- bool [isTaunt](#) ()
- void [setTaunt](#) (bool value)
- void [setLife](#) (unsigned int newLife)
- void [setAttack](#) (unsigned int newAttack)
- void [setMaxLife](#) (unsigned int newMax)
- bool [isDead](#) ()

### 6.3.1 Documentation des constructeurs et destructeur

6.3.1.1 **CardMonster** : **CardMonster** ( unsigned int *id*, std : :string *name*, unsigned int *energy*, int *effect*, bool *aBool*, unsigned int *life*, unsigned int *attack*, unsigned int *nbrTour* = 0 )

Constructor

Paramètres

<i>id</i>	of the card
<i>name</i>	of the card
<i>energy</i>	of the card
<i>effect</i>	id of the specific effect
<i>save</i>	True if save in cache
<i>life</i>	of the card
<i>attack</i>	of the card
<i>number</i>	of tour that the card is placed

6.3.1.2 **CardMonster** : **CardMonster** ( **CardMonster** & *otherMonster* )

6.3.1.3 **CardMonster** : **~CardMonster** ( ) [virtual]

### 6.3.2 Documentation des fonctions membres

6.3.2.1 void **CardMonster** : **dealDamage** ( **CardMonster** & *otherMonster* ) [virtual]

The monster attack an other monster

Paramètres

<i>card</i>	(other monster)
-------------	-----------------

Renvoie

void

6.3.2.2 void **CardMonster** : **dealDamage** ( **PlayerInGame** & *player* ) [virtual]

The monster attack a player

Paramètres

<i>player</i>	
---------------	--

Renvoie

void

6.3.2.3 `unsigned int CardMonster::getAttack ( ) [inline]`

6.3.2.4 `unsigned int CardMonster::getLife ( ) [inline]`

6.3.2.5 `unsigned int CardMonster::getMaxLife ( ) [inline]`

6.3.2.6 `unsigned int CardMonster::getNbrTourPose ( ) [inline]`

6.3.2.7 `void CardMonster::incrementTour ( ) [virtual]`

Increment the tour of the monster

Renvoie

`void`

6.3.2.8 `bool CardMonster::isDead ( )`

Check if the monster is dead

Renvoie

`true` if the monster is dead

6.3.2.9 `virtual bool CardMonster::isMonster ( ) [inline],[override],[virtual]`

Réimplémentée à partir de [Card](#).

6.3.2.10 `bool CardMonster::isTaunt ( ) [inline]`

6.3.2.11 `void CardMonster::setAttack ( unsigned int newAttack ) [inline]`

6.3.2.12 `void CardMonster::setLife ( unsigned int newLife ) [inline]`

6.3.2.13 `void CardMonster::setMaxLife ( unsigned int newMax ) [inline]`

6.3.2.14 `void CardMonster::setTaunt ( bool value ) [inline]`

La documentation de cette classe a été générée à partir des fichiers suivants :

- [server/CardMonster.hpp](#)
- [server/CardMonster.cpp](#)

## 6.4 Référence de la structure Packet : `carteInfosPacket`

```
#include <Packet.hpp>
```

### Attributs publics

- `int ID` = [CARTE\\_INFO\\_ID](#)
- `int size` = `sizeof(int)+sizeof(char)*MAX_CARTE_DESCRIPTION_SIZE`
- `int carteID`
- `char cartesDescription` [[MAX\\_CARTE\\_DESCRIPTION\\_SIZE](#)]



### 6.4.1 Documentation des données membres

6.4.1.1 int Packet : :carteInfosPacket : :carteID

6.4.1.2 char Packet : :carteInfosPacket : :cartesDescription[MAX\_CARTE\_DESCRIPTION\_SIZE]

6.4.1.3 int Packet : :carteInfosPacket : :ID = CARTE\_INFO\_ID

6.4.1.4 int Packet : :carteInfosPacket : :size = sizeof(int)+sizeof(char)\*MAX\_CARTE\_DESCRIPTION\_SIZE

La documentation de cette structure a été générée à partir du fichier suivant :

— common/[Packet.hpp](#)

## 6.5 Référence de la structure Packet : :carteRequestPacket

```
#include <Packet.hpp>
```

### Attributs publics

- int ID = CARTE\_REQ\_ID
- int size = sizeof(int)
- int carteID

### 6.5.1 Documentation des données membres

6.5.1.1 int Packet : :carteRequestPacket : :carteID

6.5.1.2 int Packet : :carteRequestPacket : :ID = CARTE\_REQ\_ID

6.5.1.3 int Packet : :carteRequestPacket : :size = sizeof(int)

La documentation de cette structure a été générée à partir du fichier suivant :

— common/[Packet.hpp](#)

## 6.6 Référence de la classe ChatManager

```
#include <ChatManager.hpp>
```

### Fonctions membres publiques

- [sendMessage](#) ([Player](#) player, std : :string message)

### 6.6.1 Documentation des fonctions membres

6.6.1.1 ChatManager : :sendMessage ( [Player](#) *player*, std : :string *message* )

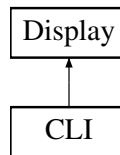
La documentation de cette classe a été générée à partir du fichier suivant :

— server/[ChatManager.hpp](#)

## 6.7 Référence de la classe CLI

```
#include <CLI.hpp>
```

Graphe d'héritage de CLI :



### Fonctions membres publiques

- [CLI](#) ()
- [~CLI](#) ()
- void [displayFatalError](#) (std : :string)
- void [displayLoginPrompt](#) ()
- void [displayLoginResult](#) (std : :string)
- void [valideLogin](#) ()
- void [displayMainWindow](#) ()
- void [displayFriendsWindow](#) ()
- void [displayCollectionWindow](#) ()
- void [displayDeckWindow](#) ()
- void [displayWait](#) ()
- void [displayGame](#) ()
- void [focusTchat](#) ()

#### 6.7.1 Documentation des constructeurs et destructeur

6.7.1.1 `CLI::CLI ( )`

6.7.1.2 `CLI::~~CLI ( )`

#### 6.7.2 Documentation des fonctions membres

6.7.2.1 `void CLI::displayCollectionWindow ( ) [virtual]`

Implémente [Display](#).

6.7.2.2 `void CLI::displayDeckWindow ( ) [virtual]`

Implémente [Display](#).

6.7.2.3 `void CLI::displayFatalError ( std : :string error ) [virtual]`

Implémente [Display](#).

6.7.2.4 `void CLI::displayFriendsWindow ( ) [virtual]`

Implémente [Display](#).

6.7.2.5 `void CLI::displayGame ( ) [virtual]`

Implémente [Display](#).

6.7.2.6 void CLI::displayLoginPrompt ( ) [virtual]

Implémente [Display](#).

6.7.2.7 void CLI::displayLoginResult ( std::string errorMessage ) [virtual]

Implémente [Display](#).

6.7.2.8 void CLI::displayMainWindow ( ) [virtual]

Implémente [Display](#).

6.7.2.9 void CLI::displayWait ( ) [virtual]

Implémente [Display](#).

6.7.2.10 void CLI::focusTchat ( ) [virtual]

Implémente [Display](#).

6.7.2.11 void CLI::valideLogin ( ) [virtual]

Implémente [Display](#).

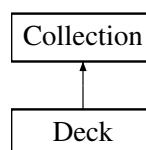
La documentation de cette classe a été générée à partir des fichiers suivants :

- client/CLI.hpp
- client/CLI.cpp

## 6.8 Référence de la classe Collection

```
#include <Collection.hpp>
```

Graphe d'héritage de Collection :



### Fonctions membres publiques

- [Collection](#) ()=default
- [Collection](#) (std::vector< [Card](#) \* >)
- [Collection](#) (std::vector< unsigned >)
- [Collection](#) (const [Collection](#) &)=default
- [Collection](#) & operator= (const [Collection](#) &)=default
- virtual [Error](#) addCard ([Card](#) \*)
- virtual [Error](#) addCard (int cardId)
- void removeCard (int i)
- void removeCard ([Card](#) \*)
- void removeCardId (int cardId)
- int indexOfCard (int cardId)
- int getCardIndex ([Card](#) \*)
- [Card](#) \* getCardOnIndex (const unsigned index)
- std::vector< unsigned > getCardsId () const

— virtual [~Collection](#) ()=default

### Attributs protégés

— std : :vector< [Card](#) \* > [\\_listCard](#)

## 6.8.1 Documentation des constructeurs et destructeur

6.8.1.1 [Collection](#) : :[Collection](#) ( ) [default]

6.8.1.2 [Collection](#) : :[Collection](#) ( std : :vector< [Card](#) \* > [listCard](#) )

Constructor

Paramètres

<i>listCard</i>	list of card
-----------------	--------------

6.8.1.3 [Collection](#) : :[Collection](#) ( std : :vector< unsigned > [listIdCard](#) )

Constructor

Paramètres

<i>listIdCard</i>	the list of card ID
-------------------	---------------------

6.8.1.4 [Collection](#) : :[Collection](#) ( const [Collection](#) & ) [default]

6.8.1.5 virtual [Collection](#) : :~[Collection](#) ( ) [virtual],[default]

## 6.8.2 Documentation des fonctions membres

6.8.2.1 [Error Collection](#) : :addCard ( [Card](#) \* [card](#) ) [virtual]

Add a [Card](#) to the collection

Paramètres

<i>card</i>	the card to add
-------------	-----------------

Renvoie

True if we can add [Card](#) (false if there is more than two cards the same)

Réimplémentée dans [Deck](#).

6.8.2.2 [Error Collection](#) : :addCard ( int [cardId](#) ) [virtual]

Add a [Card](#) to the collection

Paramètres

<i>cardId</i>	the id of the card
---------------	--------------------

Renvoie

True if we can add [Card](#) (false if there is more than two cards the same)

Réimplémentée dans [Deck](#).

**6.8.2.3** `int Collection : :getCardIndex ( Card * card )`

Get the index of one card

Paramètres

<i>card</i>	
-------------	--

Renvoie

index or -1 if not found

**6.8.2.4** `Card * Collection : :getCardOnIndex ( const unsigned index )`

Get the card on the specific index

Paramètres

<i>index</i>	of the card
--------------	-------------

Renvoie

the [Card](#) at the specific index

**6.8.2.5** `std : :vector< unsigned > Collection : :getCardsId ( ) const`**6.8.2.6** `int Collection : :indexOfCard ( int cardId )`**6.8.2.7** `Collection& Collection : :operator= ( const Collection & ) [default]`**6.8.2.8** `void Collection : :removeCard ( int i )`

Remove a [Card](#) on a specific position

Paramètres

<i>i</i>	index of the card
----------	-------------------

**6.8.2.9** `void Collection : :removeCard ( Card * card )`

Remove a [Card](#) from the collection

Paramètres

<i>card</i>	the card to remove
-------------	--------------------

**6.8.2.10** `void Collection : :removeCardId ( int cardId )`

Remove a [Card](#) from the collection

Paramètres

<i>cardId</i>	the id of the card
---------------	--------------------

**6.8.3** Documentation des données membres

6.8.3.1 `std::vector<Card*> Collection::_listCard` [protected]

La documentation de cette classe a été générée à partir des fichiers suivants :

- [server/Collection.hpp](#)
- [server/Collection.cpp](#)

## 6.9 Référence de la structure Packet : `:collectionListPacket`

```
#include <Packet.hpp>
```

### Attributs publics

- `int ID = COLLECTION_LIST_ID`
- `int size = sizeof(int)*MAX_CARTES`
- `int cartesList [MAX_CARTES]`

### 6.9.1 Documentation des données membres

6.9.1.1 `int Packet::collectionListPacket::cartesList[MAX_CARTES]`

6.9.1.2 `int Packet::collectionListPacket::ID = COLLECTION_LIST_ID`

6.9.1.3 `int Packet::collectionListPacket::size = sizeof(int)*MAX_CARTES`

La documentation de cette structure a été générée à partir du fichier suivant :

- [common/Packet.hpp](#)

## 6.10 Référence de la classe Connection

```
#include <Connection.hpp>
```

### Fonctions membres publiques

- `Connection (char *)`
- `~Connection ()`
- `void sendPacket (Packet *, size_t)`
- `Connection ()`
- `~Connection ()`
- `void mainLoop ()`

### 6.10.1 Documentation des constructeurs et destructeur

6.10.1.1 `Connection::Connection ( char * hostName )`

6.10.1.2 `Connection::~~Connection ( )`

6.10.1.3 `Connection::Connection ( )`

6.10.1.4 `Connection::~~Connection ( )`

### 6.10.2 Documentation des fonctions membres

6.10.2.1 `void Connection::mainLoop ( )`

6.10.2.2 void Connection : :sendPacket ( Packet \* packet, size\_t size )

La documentation de cette classe a été générée à partir des fichiers suivants :

- client/[Connection.hpp](#)
- client/[Connection.cpp](#)

## 6.11 Référence de la structure dataIGPlayer

```
#include <PlayerInGame.hpp>
```

### Attributs publics

- int [playerHeal](#)
- int [energy](#)
- int [maxEnergy](#)
- int [limitEnergy](#)
- std : :vector< [Card](#) \* > [cardsInHand](#)
- std : :vector< [CardMonster](#) \* > [cardsPlaced](#)
- bool [turn](#)

### 6.11.1 Documentation des données membres

6.11.1.1 std : :vector< [Card](#)\* > dataIGPlayer : :cardsInHand

6.11.1.2 std : :vector< [CardMonster](#)\* > dataIGPlayer : :cardsPlaced

6.11.1.3 int dataIGPlayer : :energy

6.11.1.4 int dataIGPlayer : :limitEnergy

6.11.1.5 int dataIGPlayer : :maxEnergy

6.11.1.6 int dataIGPlayer : :playerHeal

6.11.1.7 bool dataIGPlayer : :turn

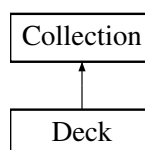
La documentation de cette structure a été générée à partir du fichier suivant :

- server/[PlayerInGame.hpp](#)

## 6.12 Référence de la classe Deck

```
#include <Deck.hpp>
```

Graphe d'héritage de Deck :



### Fonctions membres publiques

- [Deck](#) (std : :string name, std : :vector< [Card](#) \* > listCard)

- `Deck` (`std : :string name`, `std : :vector< unsigned > listCard`)
- `Deck` (`const Deck &`)
- `Deck & operator=` (`const Deck &`)=default
- `std : :string getName` () const
- `bool isValid` ()
- `Card * pickup` ()
- `virtual Error addCard` (`Card *card`) override
- `virtual Error addCard` (`int cardId`) override
- `bool deleteDeck` (`PlayerInGame *`)
- `Deck * copyDeck` ()
- `bool operator==` (`const std : :string &`) const

### Fonctions membres publiques statiques

- static `Deck * getDeck` (`std : :string`, `std : :vector< Deck * >`)

### Membres hérités additionnels

#### 6.12.1 Description détaillée

A deck is a list of 20 Cards (or less)

#### 6.12.2 Documentation des constructeurs et destructeur

6.12.2.1 `Deck : :Deck ( std : :string name, std : :vector< Card * > listCard )`

Constructor

Paramètres

<i>name</i>	of the deck
<i>listCard</i>	the must be add on deck

6.12.2.2 `Deck : :Deck ( std : :string name, std : :vector< unsigned > listCard )`

Constructor

Paramètres

<i>name</i>	of the deck
<i>listCard</i>	the must be add on deck

6.12.2.3 `Deck : :Deck ( const Deck & deck )`

Copy Constructor

Paramètres

<i>deck</i>	to copy
-------------	---------

#### 6.12.3 Documentation des fonctions membres

6.12.3.1 `Error Deck : :addCard ( Card * card )` [override],[virtual]

Adds a card in the deck



## Paramètres

<i>card</i>	the card to add
-------------	-----------------

## Renvoie

Error and NoError if all is ok

Réimplémentée à partir de [Collection](#).

**6.12.3.2 Error Deck : :addCard ( int *cardId* )** [override],[virtual]

Adds a card in the deck

## Paramètres

<i>cardId</i>	the id of the card to add
---------------	---------------------------

## Renvoie

Error and NoError if all is ok

Réimplémentée à partir de [Collection](#).

**6.12.3.3 Deck \* Deck : :copyDeck ( )**

Copy the deck AND all the card !

## Renvoie

the new deck

**6.12.3.4 bool Deck : :deleteDeck ( PlayerInGame \* )****6.12.3.5 Deck \* Deck : :getDeck ( std : :string *name*, std : :vector< Deck \* > *listDeck* )** [static]

Get the deck with a specific name

## Paramètres

<i>name</i>	of the deck
<i>listDeck</i>	to find the deck with the specific name

## Renvoie

the deck or nullptr

**6.12.3.6 std : :string Deck : :getName ( ) const** [inline]**6.12.3.7 bool Deck : :isValide ( )**

Checks if the deck is valide. He must have 20 cards

## Renvoie

True if the deck is valide

6.12.3.8 **Deck** & **Deck** : :operator= ( const **Deck** & ) [default]

6.12.3.9 **bool** **Deck** : :operator== ( const std : :string & *deckName* ) const

6.12.3.10 **Card** \* **Deck** : :pickup ( )

Returns the last card of the deck

Renvoie

the id of the card or -1 if empty

La documentation de cette classe a été générée à partir des fichiers suivants :

- server/[Deck.hpp](#)
- server/[Deck.cpp](#)

## 6.13 Référence de la structure **Packet** : :deckContentPacket

```
#include <Packet.hpp>
```

### Attributs publics

- int **ID** = [DECK\\_CONT\\_ID](#)
- int **size** = sizeof(int)\*[DECK\\_SIZE](#)+sizeof(int)
- int **deckID**
- int **cartesList** [[DECK\\_SIZE](#)]

### 6.13.1 Documentation des données membres

6.13.1.1 **int** **Packet** : :deckContentPacket : :cartesList[[DECK\\_SIZE](#)]

6.13.1.2 **int** **Packet** : :deckContentPacket : :deckID

6.13.1.3 **int** **Packet** : :deckContentPacket : :ID = [DECK\\_CONT\\_ID](#)

6.13.1.4 **int** **Packet** : :deckContentPacket : :size = sizeof(int)\*[DECK\\_SIZE](#)+sizeof(int)

La documentation de cette structure a été générée à partir du fichier suivant :

- common/[Packet.hpp](#)

## 6.14 Référence de la structure **Packet** : :deckRequestPacket

```
#include <Packet.hpp>
```

### Attributs publics

- int **ID** = [DECK\\_REQ\\_ID](#)
- int **size** = sizeof(int)
- int **deckID**

### 6.14.1 Documentation des données membres

6.14.1.1 **int** **Packet** : :deckRequestPacket : :deckID

6.14.1.2 `int Packet : :deckRequestPacket : :ID = DECK_REQ_ID`

6.14.1.3 `int Packet : :deckRequestPacket : :size = sizeof(int)`

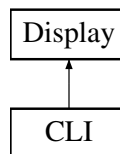
La documentation de cette structure a été générée à partir du fichier suivant :

— `common/`[Packet.hpp](#)

## 6.15 Référence de la classe Display

```
#include <Display.hpp>
```

Graphe d'héritage de Display :



### Fonctions membres publiques

- `Display()`=default
- `virtual ~Display()`=default
- `virtual void displayFatalError(std::string)=0`
- `virtual void displayLoginPrompt()`=0
- `virtual void displayLoginResult(std::string)=0`
- `virtual void valideLogin()`=0
- `virtual void displayMainWindow()`=0
- `virtual void displayFriendsWindow()`=0
- `virtual void displayCollectionWindow()`=0
- `virtual void displayDeckWindow()`=0
- `virtual void displayWait()`=0
- `virtual void displayGame()`=0
- `virtual void focusTchat()`=0

### 6.15.1 Documentation des constructeurs et destructeur

6.15.1.1 `Display : :Display ( )` [default]

6.15.1.2 `virtual Display : :~Display ( )` [virtual],[default]

### 6.15.2 Documentation des fonctions membres

6.15.2.1 `virtual void Display : :displayCollectionWindow ( )` [pure virtual]

Implémenté dans [CLI](#).

6.15.2.2 `virtual void Display : :displayDeckWindow ( )` [pure virtual]

Implémenté dans [CLI](#).

6.15.2.3 `virtual void Display : :displayFatalError ( std::string )` [pure virtual]

Implémenté dans [CLI](#).

6.15.2.4 `virtual void Display : :displayFriendsWindow ( ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.5 `virtual void Display : :displayGame ( ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.6 `virtual void Display : :displayLoginPrompt ( ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.7 `virtual void Display : :displayLoginResult ( std : :string ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.8 `virtual void Display : :displayMainWindow ( ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.9 `virtual void Display : :displayWait ( ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.10 `virtual void Display : :focusTchat ( ) [pure virtual]`

Implémenté dans [CLI](#).

6.15.2.11 `virtual void Display : :valideLogin ( ) [pure virtual]`

Implémenté dans [CLI](#).

La documentation de cette classe a été générée à partir du fichier suivant :

— [client/Display.hpp](#)

## 6.16 Référence de la classe Effect

```
#include <Effect.hpp>
```

### Fonctions membres publiques

- [Effect](#) ()
- unsigned [getId](#) ()
- virtual [~Effect](#) ()
- virtual void [apply](#) ([CardMonster](#) \*, [Game](#) \*)=0
- virtual void [apply](#) ([PlayerInGame](#) \*player, [Game](#) \*game)
- virtual bool [isTaunt](#) ()
- virtual bool [canBeApplyOnPlayer](#) ()
- virtual bool [canBeApplyOnCard](#) ()

## Fonctions membres publiques statiques

- static void [loadAllEffect](#) ( )
- static [Effect](#) \* [getEffectById](#) ( unsigned )

## 6.16.1 Documentation des constructeurs et destructeur

6.16.1.1 [Effect](#) : [Effect](#) ( )6.16.1.2 virtual [Effect](#) : [~Effect](#) ( ) [inline],[virtual]

## 6.16.2 Documentation des fonctions membres

6.16.2.1 virtual void [Effect](#) : [apply](#) ( [CardMonster](#) \*, [Game](#) \* ) [pure virtual]6.16.2.2 virtual void [Effect](#) : [apply](#) ( [PlayerInGame](#) \* *player*, [Game](#) \* *game* ) [inline],[virtual]6.16.2.3 virtual bool [Effect](#) : [canBeApplyOnCard](#) ( ) [inline],[virtual]6.16.2.4 virtual bool [Effect](#) : [canBeApplyOnPlayer](#) ( ) [inline],[virtual]6.16.2.5 [Effect](#) \* [Effect](#) : [getEffectById](#) ( unsigned *id* ) [static]

Get the effect that match with the id

Paramètres

<i>id</i>	: the id of an effect
-----------	-----------------------

Renvoie

[Effect](#)\*6.16.2.6 unsigned [Effect](#) : [getId](#) ( ) [inline]6.16.2.7 virtual bool [Effect](#) : [isTaunt](#) ( ) [inline],[virtual]6.16.2.8 void [Effect](#) : [loadAllEffect](#) ( ) [static]

Create and save all the effects in a vector

Renvoie

void

La documentation de cette classe a été générée à partir des fichiers suivants :

- [server/Effect.hpp](#)
- [server/Effect.cpp](#)

## 6.17 Référence de la classe FriendsManager

#include &lt;FriendsManager.hpp&gt;

## 6.17.1 Description détaillée

One class per [Player](#)

La documentation de cette classe a été générée à partir du fichier suivant :  
 — server/[FriendsManager.hpp](#)

## 6.18 Référence de la classe Game

```
#include <Game.hpp>
```

### Fonctions membres publiques

- [Game](#) ()
- [Game](#) (const [Game](#) &)
- [Game](#) & [operator=](#) (const [Game](#) &)=default
- virtual [~Game](#) ()=default
- void [checkDeckAndStart](#) ()
- void [draw](#) ()
- void [draw](#) ([PlayerInGame](#) \*)
- [Error](#) [placeCardAffectPlayer](#) ([PlayerInGame](#) \*, [Card](#) \*)
- [Error](#) [placeCard](#) ([PlayerInGame](#) \*, [Card](#) \*, [CardMonster](#) \*)
- [Error](#) [attackWithCard](#) ([PlayerInGame](#) \*, [CardMonster](#) \*, [CardMonster](#) \*)
- [Error](#) [attackWithCardAffectPlayer](#) ([PlayerInGame](#) \*, [CardMonster](#) \*)
- void [nextPlayer](#) ()
- void [sendInfoAction](#) ([PlayerInGame](#) \*, int, int, bool, bool, unsigned)

### Fonctions membres publiques statiques

- static void [addPlayerWaitGame](#) ([Player](#) player)

#### 6.18.1 Description détaillée

One class per game. Contains the two players and all other information for the game

#### 6.18.2 Documentation des constructeurs et destructeur

##### 6.18.2.1 [Game](#) : :[Game](#) ( )

Default constructor

##### 6.18.2.2 [Game](#) : :[Game](#) ( const [Game](#) & *game* )

Copy constructor

Paramètres

<i>game</i>	who must be copied
-------------	--------------------

##### 6.18.2.3 virtual [Game](#) : :[~Game](#) ( ) [virtual],[default]

#### 6.18.3 Documentation des fonctions membres

##### 6.18.3.1 void [Game](#) : :[addPlayerWaitGame](#) ( [Player](#) *player* ) [static]

Adds a player to the waiting list If there is more than one player who is waiting, then it creates a [Game](#)

## Paramètres

<i>player</i>	the new player waiting
---------------	------------------------

6.18.3.2 Error Game : :attackWithCard ( PlayerInGame \* *pIG*, CardMonster \* *card*, CardMonster \* *targetCard* )

Funciton when player attack a card

## Paramètres

<i>pIG</i>	who play
<i>card</i>	which play
<i>targetCard</i>	card which is attack

## Renvoie

Error or "NoError" if all is ok

6.18.3.3 Error Game : :attackWithCardAffectPlayer ( PlayerInGame \* *pIG*, CardMonster \* *card* )

Funciton when player attack a card

## Paramètres

<i>pIG</i>	who play
<i>card</i>	which play
<i>targetCard</i>	card which is attack

## Renvoie

Error or "NoError" if all is ok

## 6.18.3.4 void Game : :checkDeckAndStart ( )

Checks if the player have set his deck If all is ok, the game starts

## 6.18.3.5 void Game : :draw ( )

Current player draw a card

6.18.3.6 void Game : :draw ( PlayerInGame \* *pIG* )

Specific player draw a card

## Paramètres

<i>pIG</i>	who must draw
------------	---------------

## 6.18.3.7 void Game : :nextPlayer ( )

Switches player turn

6.18.3.8 **Game& Game : :operator= ( const Game & )** [default]

6.18.3.9 **Error Game : :placeCard ( PlayerInGame \* pIG, Card \* cardPlaced, CardMonster \* targetCard )**

Function when player place card



## Paramètres

<i>pIG</i>	player who place the card
<i>cardPlaced</i>	the card the must be place
<i>targetCard</i>	the card which will have the effect if the placed card have it

6.18.3.10 Error Game : :placeCardAffectPlayer ( PlayerInGame \* *pIG*, Card \* *cardPlaced* )

Function when player place card and attack player

## Paramètres

<i>pIG</i>	player who place the card
<i>cardPlaced</i>	the card the must be place
<i>targetPlayer</i>	player who will have the effect if the placed card have it

## Renvoie

Error or "NoError" if all ok

6.18.3.11 void Game : :sendInfoAction ( PlayerInGame \* *pIG*, int *cardWichAttack*, int *attackCard*, bool *isEffect*, bool *newCard*, unsigned *heal* )

Send information

## Paramètres

<i>pIG</i>	who play
<i>cardWichAttack</i>	
<i>attackCard</i>	card which is attack (-1 if player)
<i>heal</i>	of the attack entity

La documentation de cette classe a été générée à partir des fichiers suivants :

- [server/Game.hpp](#)
- [server/Game.cpp](#)

## 6.19 Référence de la structure Packet : :loginRequestPacket

```
#include <Packet.hpp>
```

## Attributs publics

- int *ID* = LOGIN\_REQ\_ID
- int *size* = sizeof(char)\*MAX\_PSEUDO\_SIZE\*2
- char *pseudo* [MAX\_PSEUDO\_SIZE]
- char *password* [MAX\_PSEUDO\_SIZE]

## 6.19.1 Documentation des données membres

## 6.19.1.1 int Packet : :loginRequestPacket : :ID = LOGIN\_REQ\_ID

## 6.19.1.2 char Packet : :loginRequestPacket : :password[MAX\_PSEUDO\_SIZE]

## 6.19.1.3 char Packet : :loginRequestPacket : :pseudo[MAX\_PSEUDO\_SIZE]

6.19.1.4 `int Packet : :loginRequestPacket : :size = sizeof(char)*MAX_PSEUDO_SIZE*2`

La documentation de cette structure a été générée à partir du fichier suivant :

— `common/`[Packet.hpp](#)

## 6.20 Référence de la structure `Packet : :loginResultPacket`

```
#include <Packet.hpp>
```

### Attributs publics

- `int ID = LOGIN_RES_ID`
- `int size = sizeof(int)`
- `int resultCode`

### 6.20.1 Documentation des données membres

6.20.1.1 `int Packet : :loginResultPacket : :ID = LOGIN_RES_ID`

6.20.1.2 `int Packet : :loginResultPacket : :resultCode`

6.20.1.3 `int Packet : :loginResultPacket : :size = sizeof(int)`

La documentation de cette structure a été générée à partir du fichier suivant :

— `common/`[Packet.hpp](#)

## 6.21 Référence de la classe `Packet`

```
#include <Packet.hpp>
```

### Classes

- `struct carteInfosPacket`
- `struct carteRequestPacket`
- `struct collectionListPacket`
- `struct deckContentPacket`
- `struct deckRequestPacket`
- `struct loginRequestPacket`
- `struct loginResultPacket`
- `struct packet`

### Types publics

- `enum IDList {`  
`LOGIN_REQ_ID = 11, REGIST_REQ_ID = 12, LOGIN_RES_ID = 13, DISCONNECT_ID = 14,`  
`COLLECTION_REQ_ID = 21, COLLECTION_LIST_ID = 22, DECK_REQ_ID = 23, DECK_CONT_ID = 24,`  
`CARTE_REQ_ID = 25, CARTE_INFO_ID = 26, TCHAT_CONV_REQ_ID = 31, TCHAT_NEW_CONV_ID =`  
`32,`  
`TCHAT_MESSAGE_ID = 33, TCHAT_END_REQ_ID = 34, TCHAT_END_CONV_ID = 35 }`

### Attributs publics statiques

- `static const int packetSize = sizeof(int)*2`
- `static const int packetMaxSize = sizeof(int[200])+packetSize`

### 6.21.1 Documentation des énumérations membres

#### 6.21.1.1 enum Packet : :IDList

Valeurs énumérées

*LOGIN\_REQ\_ID*  
*REGIST\_REQ\_ID*  
*LOGIN\_RES\_ID*  
*DISCONNECT\_ID*  
*COLLECTION\_REQ\_ID*  
*COLLECTION\_LIST\_ID*  
*DECK\_REQ\_ID*  
*DECK\_CONT\_ID*  
*CARTE\_REQ\_ID*  
*CARTE\_INFO\_ID*  
*TCHAT\_CONV\_REQ\_ID*  
*TCHAT\_NEW\_CONV\_ID*  
*TCHAT\_MESSAGE\_ID*  
*TCHAT\_END\_REQ\_ID*  
*TCHAT\_END\_CONV\_ID*

### 6.21.2 Documentation des données membres

6.21.2.1 `const int Packet : :packetMaxSize = sizeof(int[200])+packetSize` [static]

6.21.2.2 `const int Packet : :packetSize = sizeof(int)*2` [static]

La documentation de cette classe a été générée à partir du fichier suivant :

— common/[Packet.hpp](#)

## 6.22 Référence de la structure Packet : :packet

```
#include <Packet.hpp>
```

### Attributs publics

— int [ID](#)  
— int [size](#) = 0

### 6.22.1 Documentation des données membres

6.22.1.1 `int Packet : :packet : :ID`

6.22.1.2 `int Packet : :packet : :size = 0`

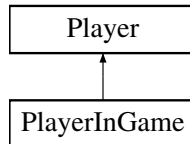
La documentation de cette structure a été générée à partir du fichier suivant :

— common/[Packet.hpp](#)

## 6.23 Référence de la classe Player

```
#include <Player.hpp>
```

Graphe d'héritage de Player :



### Fonctions membres publiques

- `Player` (`nlohmann : :json &`, `int sockfd=0`)
- `void adjudicateVictory ()`
- `void adjudicateDefeat ()`
- `Error addCardCollection (Card *c)`
- `Collection * getCollection ()`
- `void updateSockfd (int a)`
- `std : :string getName () const`
- `std : :string getPass () const`
- `unsigned getVictories () const`
- `unsigned getDefeats () const`
- `Deck * getDeck (std : :string)`
- `bool removeDeck (Deck *)`
- `void sendPacket (Packet : :packet *, size_t)`
- `void recvLoop ()`
- `void logout ()`
- `std : :string serialise () const`
- `bool operator== (const std : :string &) const`
- `bool operator< (const Player &) const`
- `bool operator> (const Player &) const`
- `virtual ~Player ()`

### Fonctions membres protégées

- `std : :vector< Deck * > getListDeck ()`

### Attributs protégés

- `unsigned _victories`
- `unsigned _defeats`

### Amis

- `std : :ostream & operator<< (std : :ostream &, const Player &)`
- `std : :string & operator<< (std : :string &, const Player &)`

#### 6.23.1 Description détaillée

One class per player. This object stocks the socket to communicate with player When the server starts, he must load all Players

#### 6.23.2 Documentation des constructeurs et destructeur

##### 6.23.2.1 `Player : :Player ( nlohmann : :json & info, int sockfd = 0 )`

6.23.2.2 `virtual Player : ~Player ( ) [inline], [virtual]`

### 6.23.3 Documentation des fonctions membres

6.23.3.1 `Error Player : addCardCollection ( Card * c ) [inline]`

6.23.3.2 `void Player : adjudicateDefeat ( ) [inline]`

6.23.3.3 `void Player : adjudicateVictory ( ) [inline]`

6.23.3.4 `Collection* Player : getCollection ( ) [inline]`

6.23.3.5 `Deck * Player : getDeck ( std : string deckName )`

Get the deck with this name

Paramètres

<i>deckname</i>	the name of the deck
-----------------	----------------------

Renvoie

the deck (or nullptr if not found)

6.23.3.6 `unsigned Player : getDefeats ( ) const [inline]`

6.23.3.7 `std : vector<Deck*> Player : getListDeck ( ) [inline], [protected]`

6.23.3.8 `std : string Player : getName ( ) const [inline]`

6.23.3.9 `std : string Player : getPass ( ) const [inline]`

6.23.3.10 `unsigned Player : getVictories ( ) const [inline]`

6.23.3.11 `void Player : logout ( )`

6.23.3.12 `bool Player : operator< ( const Player & other ) const`

6.23.3.13 `bool Player : operator== ( const std : string & other_name ) const`

6.23.3.14 `bool Player : operator> ( const Player & other ) const`

6.23.3.15 `void Player : recvLoop ( )`

6.23.3.16 `bool Player : removeDeck ( Deck * deck )`

Remove a [Deck](#)

Paramètres

<i>deck</i>	to remove
-------------	-----------

Renvoie

True if all is ok (cann't delete if one deck)

6.23.3.17 void Player::sendPacket ( Packet : :packet \* packet, size\_t size )

6.23.3.18 std::string Player::serialise ( ) const

6.23.3.19 void Player::updateSockfd ( int a ) [inline]

## 6.23.4 Documentation des fonctions amies et associées

6.23.4.1 std::ostream& operator<< ( std::ostream & os, const Player & c ) [friend]

6.23.4.2 std::string& operator<< ( std::string & str, const Player & c ) [friend]

## 6.23.5 Documentation des données membres

6.23.5.1 unsigned Player::\_defeats [protected]

6.23.5.2 unsigned Player::\_victories [protected]

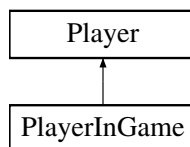
La documentation de cette classe a été générée à partir des fichiers suivants :

- server/Player.hpp
- server/Player.cpp

## 6.24 Référence de la classe PlayerInGame

```
#include <PlayerInGame.hpp>
```

Graphe d'héritage de PlayerInGame :



### Fonctions membres publiques

- PlayerInGame (const PlayerInGame &)=default
- PlayerInGame & operator= (const PlayerInGame &)=default
- PlayerInGame (const Player &, Game \*)
- dataIGPlayer getDataPlayer ()
- std::vector< CardMonster \* > getCardsPlaced ()
- std::vector< Card \* > getCardsInHand ()
- unsigned int nbrCardInHand ()
- void setDeck (Deck \*deck)
- bool isDeckDefined ()
- Card \* draw ()
- bool haveEnoughEnergy (Card \*card)
- void addMaxEnergy ()
- int resetEnergy ()
- void defausseCardPlaced (CardMonster \*)
- void placeCard (CardMonster \*)
- void takeDamage (unsigned int)
- void getHealed (unsigned int)
- int getHeal ()
- void addDefeat ()
- void addWin ()
- bool isDead ()
- virtual ~PlayerInGame ()=default

## Membres hérités additionnels

## 6.24.1 Documentation des constructeurs et destructeur

6.24.1.1 `PlayerInGame : :PlayerInGame ( const PlayerInGame & ) [default]`6.24.1.2 `PlayerInGame : :PlayerInGame ( const Player & player, Game * game )`Creates a `PlayerInGame` and asks to the player which `Deck` he would like to play with6.24.1.3 `virtual PlayerInGame : :~PlayerInGame ( ) [virtual],[default]`

## 6.24.2 Documentation des fonctions membres

6.24.2.1 `void PlayerInGame : :addDefeat ( )`

Adds a defeat to the player

6.24.2.2 `void PlayerInGame : :addMaxEnergy ( )`6.24.2.3 `void PlayerInGame : :addWin ( )`

Adds a win to the player

6.24.2.4 `void PlayerInGame : :defausseCardPlaced ( CardMonster * card )`6.24.2.5 `Card * PlayerInGame : :draw ( )`

Gets random card of deck and place it in his hand !

Renvoie

the card or nullptr

6.24.2.6 `std : :vector< Card * > PlayerInGame : :getCardsInHand ( )`

Returns the cards in hand

6.24.2.7 `std : :vector< CardMonster * > PlayerInGame : :getCardsPlaced ( )`

Returns the placed cards

6.24.2.8 `dataIGPlayer PlayerInGame : :getDataPlayer ( )`

Gets data information from this player to send it

Paramètres

<i>isTurn</i>	of the current player
---------------	-----------------------

6.24.2.9 `int PlayerInGame : :getHeal ( )`

6.24.2.10 `void PlayerInGame : :getHealed ( unsigned int heal )`

6.24.2.11 `bool PlayerInGame : :haveEnoughEnergy ( Card * card )`

6.24.2.12 `bool PlayerInGame : :isDead ( )`

Checks if the player is Dead

6.24.2.13 `bool PlayerInGame : :isDeckDefined ( )`

Checks if the deck of this player is defined

Renvoie

True if the deck is defined

6.24.2.14 `unsigned PlayerInGame : :nbrCardInHand ( )`

6.24.2.15 `PlayerInGame& PlayerInGame : :operator= ( const PlayerInGame & )` [default]

6.24.2.16 `void PlayerInGame : :placeCard ( CardMonster * card )`

6.24.2.17 `int PlayerInGame : :resetEnergy ( )`

Permet de remettre l'énergie au maximum

Renvoie

la nouvelle valeur d'énergie

6.24.2.18 `void PlayerInGame : :setDeck ( Deck * deck )`

Sets the player deck and notifies it at the game object

Paramètres

<i>deck</i>	The selected deck
-------------	-------------------

6.24.2.19 `void PlayerInGame : :takeDamage ( unsigned int damage )`

La documentation de cette classe a été générée à partir des fichiers suivants :

- [server/PlayerInGame.hpp](#)
- [server/PlayerInGame.cpp](#)

## 6.25 Référence de la classe PlayerManager

```
#include <PlayerManager.hpp>
```

### Fonctions membres publiques

- [PlayerManager](#) ()=default
- `std : :string` [getRanking](#) ()



- void [loadPlayers](#) ()
- void [savePlayers](#) () const
- [Player](#) \* [signUp](#) (std : :string, std : :string, int)
- [Player](#) \* [login](#) (std : :string, std : :string, int)
- virtual [~PlayerManager](#) ()=default

## 6.25.1 Documentation des constructeurs et destructeur

6.25.1.1 [PlayerManager](#) : :[PlayerManager](#) ( ) [default]

6.25.1.2 virtual [PlayerManager](#) : :~[PlayerManager](#) ( ) [virtual],[default]

## 6.25.2 Documentation des fonctions membres

6.25.2.1 std : :string [PlayerManager](#) : :[getRanking](#) ( )

6.25.2.2 void [PlayerManager](#) : :[loadPlayers](#) ( )

6.25.2.3 [Player](#) \* [PlayerManager](#) : :[login](#) ( std : :string *username*, std : :string *password*, int *sockfd* )

6.25.2.4 void [PlayerManager](#) : :[savePlayers](#) ( ) const

6.25.2.5 [Player](#) \* [PlayerManager](#) : :[signUp](#) ( std : :string *username*, std : :string *password*, int *sockfd* )

La documentation de cette classe a été générée à partir des fichiers suivants :

- server/[PlayerManager.hpp](#)
- server/[PlayerManager.cpp](#)



## Chapitre 7

# Documentation des fichiers

### 7.1 Référence du fichier client/Card.hpp

```
#include <string>
```

#### Classes

— class [Card](#)

### 7.2 Référence du fichier server/Card.hpp

```
#include <string>
#include <map>
#include <cstdint>
#include "Effect.hpp"
```

#### Classes

— class [Card](#)

### 7.3 Référence du fichier client/CLI.cpp

```
#include "CLI.hpp"
```

## 7.4 Référence du fichier client/CLI.hpp

```
#include <ncurses.h>
#include <panel.h>
#include <string>
#include <iostream>
#include "common/WizardLogger.hpp"
#include "Display.hpp"
#include "client/CLI/CLIPanel.hpp"
#include "client/CLI/LoginPanel.hpp"
#include "client/CLI/MainPanel.hpp"
#include "client/CLI/TchatPanel.hpp"
#include "client/CLI/FriendPanel.hpp"
#include "client/CLI/CollectionPanel.hpp"
```

### Classes

— class [CLI](#)

### Macros

```
— #define MAIN\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : VALIDER F10 : QUITTER "
— #define AMIS\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : AJOUTER F4 : RETIRER F10 :
  RETOUR "
— #define COLL\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : AFFICHER DECK F10 : RETOUR "
— #define DECK\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : CREER F4 : SUPPRIMER F10 :
  RETOUR "
— #define WAIT\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F10 : ANNULER "
— #define GAME\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : PIOCHER F4 : JOUER F5 : JETER
  F6 : FIN DE TOUR F10 : ABANDONNER"
— #define DEFAULT\_EMPTY\_SPACE 2
— #define LINES 37
— #define COLLONNES 96
— #define COMMANDE\_PANEL\_HEIGHT 1
— #define GAME\_HEIGHT 21
— #define GAME\_WIDTH 63
— #define CARD\_HEIGHT (GAME\_HEIGHT%)
— #define CARD\_WIDTH (GAME\_WIDTH%)
— #define PLAYER\_INFO\_HEIGHT (3+2*DEFAULT\_EMPTY\_SPACE)
— #define PLAYER\_INFO\_WIDTH GAME\_WIDTH
— #define TCHAT\_INPUT\_HEIGHT PLAYER\_INFO\_HEIGHT
— #define TCHAT\_INPUT\_WIDTH 27
— #define TCHAT\_HEIGHT 14
— #define TCHAT\_WIDTH TCHAT\_INPUT\_WIDTH
— #define CARD\_INFO\_HEIGHT 5
— #define CARD\_INFO\_WIDTH TCHAT\_INPUT\_WIDTH
— #define PANEL\_TOTAL\_NUMBER 4
```

### 7.4.1 Documentation des macros

7.4.1.1 `#define AMIS\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : AJOUTER F4 : RETIRER F10 : RETOUR "`

7.4.1.2 `#define CARD\_HEIGHT (GAME\_HEIGHT%)`

7.4.1.3 `#define CARD\_INFO\_HEIGHT 5`

7.4.1.4 `#define CARD\_INFO\_WIDTH TCHAT\_INPUT\_WIDTH`

7.4.1.5 `#define CARD\_WIDTH (GAME\_WIDTH%)`

7.4.1.6 #define COLL\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : AFFICHER DECK F10 : RETOUR "

7.4.1.7 #define COLLONNES 96

7.4.1.8 #define COMMANDE\_PANEL\_HEIGHT 1

7.4.1.9 #define DECK\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : CREER F4 : SUPPRIMER F10 : RETOUR "

7.4.1.10 #define DEFAULT\_EMPTY\_SPACE 2

7.4.1.11 #define GAME\_HEIGHT 21

7.4.1.12 #define GAME\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : PIOCHER F4 : JOUER F5 : JETER F6 : FIN DE TOUR  
F10 : ABANDONNER"

7.4.1.13 #define GAME\_WIDTH 63

7.4.1.14 #define LINES 37

7.4.1.15 #define MAIN\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F3 : VALIDER F10 : QUITTER "

in-game layout

# # cardInfo

gamePanel

# # tchat

# # tchat

playersInfo # # input

commandList (key shortcut)

7.4.1.16 #define PANEL\_TOTAL\_NUMBER 4

7.4.1.17 #define PLAYER\_INFO\_HEIGHT (3+2\*DEFAULT\_EMPTY\_SPACE)

7.4.1.18 #define PLAYER\_INFO\_WIDTH GAME\_WIDTH

7.4.1.19 #define TCHAT\_HEIGHT 14

7.4.1.20 #define TCHAT\_INPUT\_HEIGHT PLAYER\_INFO\_HEIGHT

7.4.1.21 #define TCHAT\_INPUT\_WIDTH 27

7.4.1.22 #define TCHAT\_WIDTH TCHAT\_INPUT\_WIDTH

7.4.1.23 #define WAIT\_LABEL "F1 : TCHAT F2 : ENVOYER (TCHAT) F10 : ANNULER "

## 7.5 Référence du fichier client/Connection.cpp

```
#include "Connection.hpp"
```

## 7.6 Référence du fichier server/Connection.cpp

```
#include "Connection.hpp"
```

## 7.7 Référence du fichier client/Connection.hpp

```
#include <netdb.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/tcp.h>
#include <pthread.h>
#include "common/Packet.hpp"
#include "PacketManager.hpp"
#include "common/WizardLogger.hpp"
#include "Display.hpp"
```

## Classes

— class [Connection](#)

## Macros

— #define [PORT](#) 5555

## Variables

— [Display](#) \* [display](#)

### 7.7.1 Documentation des macros

#### 7.7.1.1 #define PORT 5555

### 7.7.2 Documentation des variables

#### 7.7.2.1 Display\* display

## 7.8 Référence du fichier server/Connection.hpp

```
#include <vector>
#include <system_error>
#include <cstdlib>
#include <unistd.h>
#include <cstring>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/tcp.h>
#include <pthread.h>
#include "common/Packet.hpp"
#include "PacketManager.hpp"
#include "Player.hpp"
#include "PlayerManager.hpp"
```

## Classes

— class [Connection](#)

## Macros

— #define [PORT](#) 5555  
— #define [BACKLOG](#) 5 /\* Pending connections the queue will hold \*/

## Variables

— [PlayerManager](#) \* [pm](#)

### 7.8.1 Documentation des macros

#### 7.8.1.1 #define BACKLOG 5 /\* Pending connections the queue will hold \*/

7.8.1.2 `#define PORT 5555`

## 7.8.2 Documentation des variables

7.8.2.1 `PlayerManager* pm`

## 7.9 Référence du fichier client/Display.hpp

```
#include <string>
```

### Classes

— class [Display](#)

## 7.10 Référence du fichier client/main.cpp

```
#include <cstdlib>
#include <iostream>
#include <system_error>
#include "Connection.hpp"
#include "common/WizardLogger.hpp"
#include "Display.hpp"
#include "CLI.hpp"
```

### Fonctions

— int [main](#) (int argc, char \*\*argv)

### Variables

— [Connection](#) \* conn  
— [Display](#) \* display

## 7.10.1 Documentation des fonctions

7.10.1.1 `int main ( int argc, char ** argv )`

## 7.10.2 Documentation des variables

7.10.2.1 `Connection* conn`

7.10.2.2 `Display* display`



## 7.11 Référence du fichier server/main.cpp

```
#include <cstdlib>
#include <iostream>
#include <exception>
#include "Connection.hpp"
#include "CardManager.hpp"
#include "common/WizardLogger.hpp"
#include "Effect.hpp"
```

### Fonctions

— int [main](#) ()

### Variables

— [PlayerManager](#) \* [pm](#)

#### 7.11.1 Documentation des fonctions

7.11.1.1 int [main](#) ( )

#### 7.11.2 Documentation des variables

7.11.2.1 [PlayerManager](#)\* [pm](#)

## 7.12 Référence du fichier client/PackageManager.cpp

```
#include "PackageManager.hpp"
```

### Variables

— [Connection](#) \* [conn](#)

#### 7.12.1 Documentation des variables

7.12.1.1 [Connection](#)\* [conn](#)

## 7.13 Référence du fichier server/PackageManager.cpp

```
#include "PackageManager.hpp"
```

## 7.14 Référence du fichier client/PackageManager.hpp

```
#include <string>
#include <vector>
#include "common/Packet.hpp"
#include "common/WizardLogger.hpp"
#include "Connection.hpp"
#include "Display.hpp"
```

### Espaces de nommage

— [PackageManager](#)

### Macros

— `#define` [MAX\\_CARTES](#) 200

### Fonctions

— void [PackageManager::managePacket](#) ([Packet](#) : [:packet](#) \*)  
— void [PackageManager::makeLoginRequest](#) (const char \*, const char \*)  
— void [PackageManager::makeRegistrationRequest](#) (const char \*, const char \*)  
— void [PackageManager::sendDisconnection](#) ()  
— void [PackageManager::requestCollection](#) ()  
— void [PackageManager::loginResult](#) (const [Packet](#) : [:loginResultPacket](#) \*)  
— void [PackageManager::collectionResult](#) (const [Packet](#) : [:collectionListPacket](#) \*)

### Variables

— [Display](#) \* [display](#)

### 7.14.1 Documentation des macros

7.14.1.1 `#define` [MAX\\_CARTES](#) 200

### 7.14.2 Documentation des variables

7.14.2.1 [Display](#)\* [display](#)

## 7.15 Référence du fichier server/PackageManager.hpp

```
#include <string>
#include "common/Packet.hpp"
#include "common/WizardLogger.hpp"
#include "Card.hpp"
#include "Collection.hpp"
#include "Player.hpp"
```

### Espaces de nommage

— [PackageManager](#)

## Fonctions

- void `PacketManager::manageDisconnectRequest` (`Player *`, `Packet : :packet *`)
- void `PacketManager::manageCollectionRequest` (`Player *`, `Packet : :packet *`)
- void `PacketManager::managePacket` (`Player *`, `Packet : :packet *`)
- void `PacketManager::initGame` (`Player *`, `std : :string`)
- void `PacketManager::sendCard` (`Player *`, `Card *`)
- void `PacketManager::setTurn` (`Player *`, `std : :string`)
- void `PacketManager::sendInfoStartTurn` (`Player *`, `int`, `int`)
- void `PacketManager::sendAttack` (`Player *`, `std : :string`, `int`, `int`, `bool`, `bool`, `unsigned int`)
- void `PacketManager::askDefausse` (`Player *`, `int`)
- void `PacketManager::sendEndGame` (`Player *`, `bool`)

## 7.16 Référence du fichier common/Error.hpp

## Énumérations

- enum `Error` {  
`NoError` = 0, `NotEnoughEnergy` = 1, `NotEnoughPlace` = 2, `NotHisTurn` = 3,  
`MustAttackTaunt` = 4, `UnknowError` = 5, `NotEffectForPlayer` = 6, `NotEffectForMonster` = 7,  
`DeckFull` = 8, `TwoSameCardMax` = 9 }

## 7.16.1 Documentation du type de l'énumération

## 7.16.1.1 enum Error

Valeurs énumérées

***NoError******NotEnoughEnergy******NotEnoughPlace******NotHisTurn******MustAttackTaunt******UnknowError******NotEffectForPlayer******NotEffectForMonster******DeckFull******TwoSameCardMax***

## 7.17 Référence du fichier common/Packet.hpp

```
#include <string>
#include <list>
```

## Classes

- class `Packet`
- struct `Packet : :packet`
- struct `Packet : :loginRequestPacket`
- struct `Packet : :loginResultPacket`
- struct `Packet : :collectionListPacket`
- struct `Packet : :deckRequestPacket`
- struct `Packet : :deckContentPacket`
- struct `Packet : :carteRequestPacket`
- struct `Packet : :carteInfosPacket`

## Macros

- #define [MAX\\_PSEUDO\\_SIZE](#) 30
- #define [MAX\\_CARTES](#) 200
- #define [DECK\\_SIZE](#) 20
- #define [MAX\\_CARTE\\_DESCRIPTION\\_SIZE](#) 120

### 7.17.1 Documentation des macros

7.17.1.1 #define [DECK\\_SIZE](#) 20

7.17.1.2 #define [MAX\\_CARTE\\_DESCRIPTION\\_SIZE](#) 120

7.17.1.3 #define [MAX\\_CARTES](#) 200

7.17.1.4 #define [MAX\\_PSEUDO\\_SIZE](#) 30

## 7.18 Référence du fichier common/WizardLogger.cpp

```
#include "WizardLogger.hpp"
```

## 7.19 Référence du fichier common/WizardLogger.hpp

```
#include <iostream>
#include <stdexcept>
#include <string>
#include "include/spdlog/spdlog.h"
```

## Espaces de nommage

- [WizardLogger](#)

## Macros

- #define [SERVER\\_LOGFILE](#) "WizardPokerServer"
- #define [CLIENT\\_LOGFILE](#) "WizardPoker"
- #define [LOGGER](#) "MainLogger"

## Fonctions

- void [WizardLogger::initLogger](#) (bool, std::string)
- void [WizardLogger::info](#) (std::string)
- void [WizardLogger::warning](#) (std::string)
- void [WizardLogger::error](#) (std::string)
- void [WizardLogger::error](#) (std::string, std::string)
- void [WizardLogger::fatal](#) (std::string)
- void [WizardLogger::fatal](#) (std::string, std::string)

### 7.19.1 Documentation des macros

7.19.1.1 #define [CLIENT\\_LOGFILE](#) "WizardPoker"

7.19.1.2 #define [LOGGER](#) "MainLogger"

7.19.1.3 `#define SERVER_LOGFILE "WizardPokerServer"`

## 7.20 Référence du fichier server/Card.cpp

```
#include "Card.hpp"
```

## 7.21 Référence du fichier server/CardManager.cpp

```
#include "CardManager.hpp"  
#include "Card.hpp"  
#include "CardMonster.hpp"
```

## 7.22 Référence du fichier server/CardManager.hpp

```
#include <map>  
#include "include/json.hpp"  
#include <fstream>  
#include <time.h>  
#include <stdlib.h>  
#include "common/WizardLogger.hpp"
```

### Classes

— class [CardManager](#)

### Définitions de type

— using [json](#) = `nlohmann : json`

### 7.22.1 Documentation des définitions de type

7.22.1.1 using [json](#) = `nlohmann : json`

## 7.23 Référence du fichier server/CardMonster.cpp

```
#include "CardMonster.hpp"
```

## 7.24 Référence du fichier server/CardMonster.hpp

```
#include "Card.hpp"  
#include "Player.hpp"  
#include "PlayerInGame.hpp"
```

## Classes

— class [CardMonster](#)

## 7.25 Référence du fichier server/ChatManager.hpp

```
#include <string>
```

## Classes

— class [ChatManager](#)

## 7.26 Référence du fichier server/Collection.cpp

```
#include "Collection.hpp"
```

## 7.27 Référence du fichier server/Collection.hpp

```
#include <vector>
#include "common/Error.hpp"
#include "CardManager.hpp"
#include "Card.hpp"
```

## Classes

— class [Collection](#)

## 7.28 Référence du fichier server/Deck.cpp

```
#include <algorithm>
#include <stdexcept>
#include "Deck.hpp"
```

## 7.29 Référence du fichier server/Deck.hpp

```
#include <string>
#include <vector>
#include "Collection.hpp"
#include "common/WizardLogger.hpp"
#include "common/Error.hpp"
```

## Classes

— class [Deck](#)

## Macros

— #define [LIMITNAME](#) 20

### 7.29.1 Documentation des macros

7.29.1.1 #define [LIMITNAME](#) 20

## 7.30 Référence du fichier server/Effect.cpp

```
#include "Effect.hpp"
#include "CardMonster.hpp"
#include "effect/Taunt.hpp"
#include "effect/Heal.hpp"
#include "effect/Damage.hpp"
#include "effect/LifeBlessing.hpp"
#include "effect/LifeCurse.hpp"
#include "effect/AttackBlessing.hpp"
#include "effect/AttackCurse.hpp"
#include "effect/Draw.hpp"
```

## 7.31 Référence du fichier server/Effect.hpp

```
#include <vector>
```

## Classes

— class [Effect](#)

## 7.32 Référence du fichier server/FriendsManager.hpp

```
#include <vector>
```

## Classes

— class [FriendsManager](#)

## 7.33 Référence du fichier server/Game.cpp

```
#include "Game.hpp"
```

## 7.34 Référence du fichier server/Game.hpp

```
#include <queue>
#include <vector>
#include "Player.hpp"
#include "PlayerInGame.hpp"
#include "CardMonster.hpp"
#include "common/WizardLogger.hpp"
#include "PacketManager.hpp"
#include "common/Error.hpp"
```

### Classes

— class [Game](#)

### Énumérations

— enum [GameStatut](#) { [WAIT\\_DEC](#), [IN\\_GAME](#) }

#### 7.34.1 Documentation du type de l'énumération

##### 7.34.1.1 enum [GameStatut](#)

Valeurs énumérées

***WAIT\_DEC***

***IN\_GAME***

## 7.35 Référence du fichier server/Player.cpp

```
#include "include/json.hpp"
#include "Player.hpp"
```

### Fonctions

— `std::ostream & operator<< (std::ostream &os, const Player &c)`  
— `std::string & operator<< (std::string &str, const Player &c)`

#### 7.35.1 Documentation des fonctions

##### 7.35.1.1 `std::ostream& operator<< ( std::ostream & os, const Player & c )`

##### 7.35.1.2 `std::string& operator<< ( std::string & str, const Player & c )`



## 7.36 Référence du fichier server/Player.hpp

```
#include <string>
#include <vector>
#include <fstream>
#include <sys/socket.h>
#include "include/json.hpp"
#include "common/Error.hpp"
#include "Collection.hpp"
#include "Deck.hpp"
#include "common/Packet.hpp"
#include "PacketManager.hpp"
#include "common/WizardLogger.hpp"
```

### Classes

— class [Player](#)

## 7.37 Référence du fichier server/PlayerInGame.cpp

```
#include "PlayerInGame.hpp"
#include "Card.hpp"
#include "CardMonster.hpp"
```

## 7.38 Référence du fichier server/PlayerInGame.hpp

```
#include <vector>
#include <algorithm>
#include "Deck.hpp"
#include "Player.hpp"
#include "common/Packet.hpp"
#include "CardManager.hpp"
#include "Game.hpp"
```

### Classes

— struct [dataIGPlayer](#)  
— class [PlayerInGame](#)

## 7.39 Référence du fichier server/PlayerManager.cpp

```
#include "PlayerManager.hpp"
```

## 7.40 Référence du fichier server/PlayerManager.hpp

```
#include <cstdlib>
#include <iostream>
#include <algorithm>
#include <random>
#include "Player.hpp"
#include "include/json.hpp"
#include "common/Packet.hpp"
```

### Classes

— class [PlayerManager](#)

### Macros

— #define [PLAYERS\\_DB](#) "server/assets/players.json"

#### 7.40.1 Documentation des macros

7.40.1.1 #define [PLAYERS\\_DB](#) "server/assets/players.json"

# Index

- `_defeats`
    - Player, [40](#)
  - `_listCard`
    - Collection, [23](#)
  - `_victories`
    - Player, [40](#)
  - `~CLI`
    - CLI, [20](#)
  - `~Card`
    - Card, [14](#)
  - `~CardMonster`
    - CardMonster, [17](#)
  - `~Collection`
    - Collection, [22](#)
  - `~Connection`
    - Connection, [24](#)
  - `~Display`
    - Display, [29](#)
  - `~Effect`
    - Effect, [31](#)
  - `~Game`
    - Game, [32](#)
  - `~Player`
    - Player, [38](#)
  - `~PlayerInGame`
    - PlayerInGame, [41](#)
  - `~PlayerManager`
    - PlayerManager, [43](#)
- `AMIS_LABEL`
  - CLI.hpp, [46](#)
- `addCard`
  - Collection, [22](#)
  - Deck, [26](#), [27](#)
- `addCardCollection`
  - Player, [39](#)
- `addDefeat`
  - PlayerInGame, [41](#)
- `addMaxEnergy`
  - PlayerInGame, [41](#)
- `addPlayerWaitGame`
  - Game, [32](#)
- `addWin`
  - PlayerInGame, [41](#)
- `adjudicateDefeat`
  - Player, [39](#)
- `adjudicateVictory`
  - Player, [39](#)
- `apply`
  - Effect, [31](#)
- `applyEffect`
  - Card, [14](#)
- `askDefausse`
  - PacketManager, [9](#)
- `attackWithCard`
  - Game, [33](#)
- `attackWithCardAffectPlayer`
  - Game, [33](#)
- `BACKLOG`
  - server/Connection.hpp, [49](#)
- `CARD_HEIGHT`
  - CLI.hpp, [46](#)
- `CARD_INFO_HEIGHT`
  - CLI.hpp, [46](#)
- `CARD_INFO_WIDTH`
  - CLI.hpp, [46](#)
- `CARD_WIDTH`
  - CLI.hpp, [46](#)
- `CARTE_INFO_ID`
  - Packet, [37](#)
- `CARTE_REQ_ID`
  - Packet, [37](#)
- `CLI`, [20](#)
  - `~CLI`, [20](#)
  - CLI, [20](#)
  - `displayCollectionWindow`, [20](#)
  - `displayDeckWindow`, [20](#)
  - `displayFatalError`, [20](#)
  - `displayFriendsWindow`, [20](#)
  - `displayGame`, [20](#)
  - `displayLoginPrompt`, [20](#)
  - `displayLoginResult`, [21](#)
  - `displayMainWindow`, [21](#)
  - `displayWait`, [21](#)
  - `focusTchat`, [21](#)
  - `valideLogin`, [21](#)
- CLI.hpp
  - `AMIS_LABEL`, [46](#)
  - `CARD_HEIGHT`, [46](#)
  - `CARD_INFO_HEIGHT`, [46](#)
  - `CARD_INFO_WIDTH`, [46](#)
  - `CARD_WIDTH`, [46](#)
  - `COLL_LABEL`, [46](#)
  - `COLLONNES`, [47](#)
  - `COMMANDE_PANEL_HEIGHT`, [47](#)
  - `DECK_LABEL`, [47](#)
  - `DEFAULT_EMPTY_SPACE`, [47](#)
  - `GAME_HEIGHT`, [47](#)

- GAME\_LABEL, [47](#)
- GAME\_WIDTH, [47](#)
- LINES, [47](#)
- MAIN\_LABEL, [47](#)
- PANEL\_TOTAL\_NUMBER, [48](#)
- PLAYER\_INFO\_HEIGHT, [48](#)
- PLAYER\_INFO\_WIDTH, [48](#)
- TCHAT\_HEIGHT, [48](#)
- TCHAT\_INPUT\_HEIGHT, [48](#)
- TCHAT\_INPUT\_WIDTH, [48](#)
- TCHAT\_WIDTH, [48](#)
- WAIT\_LABEL, [48](#)
- CLIENT\_LOGFILE
  - WizardLogger.hpp, [54](#)
- COLL\_LABEL
  - CLI.hpp, [46](#)
- COLLECTION\_LIST\_ID
  - Packet, [37](#)
- COLLECTION\_REQ\_ID
  - Packet, [37](#)
- COLLONNES
  - CLI.hpp, [47](#)
- COMMANDE\_PANEL\_HEIGHT
  - CLI.hpp, [47](#)
- canBeApplyOnCard
  - Card, [14](#)
  - Effect, [31](#)
- canBeApplyOnPlayer
  - Card, [14](#)
  - Effect, [31](#)
- Card, [13](#)
  - ~Card, [14](#)
  - applyEffect, [14](#)
  - canBeApplyOnCard, [14](#)
  - canBeApplyOnPlayer, [14](#)
  - Card, [13](#), [14](#)
  - getDescription, [15](#)
  - getEffectID, [15](#)
  - getEnergyCost, [15](#)
  - getHP, [15](#)
  - getId, [15](#)
  - getMaxHP, [15](#)
  - getName, [15](#)
  - gotEffect, [15](#)
  - isMonster, [15](#)
  - operator=, [15](#)
- CardManager, [16](#)
  - chooseCardWin, [16](#)
  - getCardById, [16](#)
  - loadAllCards, [16](#)
- CardManager.hpp
  - json, [55](#)
- CardMonster, [16](#)
  - ~CardMonster, [17](#)
  - CardMonster, [17](#)
  - dealDamage, [17](#)
  - getAttack, [17](#)
  - getLife, [18](#)
  - getMaxLife, [18](#)
  - getNbrTourPose, [18](#)
  - incrementTour, [18](#)
  - isDead, [18](#)
  - isMonster, [18](#)
  - isTaunt, [18](#)
  - setAttack, [18](#)
  - setLife, [18](#)
  - setMaxLife, [18](#)
  - setTaunt, [18](#)
- cardsInHand
  - dataIGPlayer, [25](#)
- cardsPlaced
  - dataIGPlayer, [25](#)
- carteID
  - Packet : :carteInfosPacket, [19](#)
  - Packet : :carteRequestPacket, [19](#)
- cartesDescription
  - Packet : :carteInfosPacket, [19](#)
- cartesList
  - Packet : :collectionListPacket, [24](#)
  - Packet : :deckContentPacket, [28](#)
- ChatManager, [19](#)
  - sendMessage, [19](#)
- checkDeckAndStart
  - Game, [33](#)
- chooseCardWin
  - CardManager, [16](#)
- client/CLI.cpp, [45](#)
- client/CLI.hpp, [46](#)
- client/Card.hpp, [45](#)
- client/Connection.cpp, [48](#)
- client/Connection.hpp, [48](#)
  - display, [49](#)
  - PORT, [49](#)
- client/Display.hpp, [50](#)
- client/PackageManager.cpp, [51](#)
  - conn, [51](#)
- client/PackageManager.hpp, [52](#)
  - display, [52](#)
  - MAX\_CARTES, [52](#)
- client/main.cpp, [50](#)
  - conn, [50](#)
  - display, [50](#)
  - main, [50](#)
- Collection, [21](#)
  - \_listCard, [23](#)
  - ~Collection, [22](#)
  - addCard, [22](#)
  - Collection, [22](#)
  - getCardIndex, [22](#)
  - getCardOnIndex, [23](#)
  - getCardsId, [23](#)
  - indexOfCard, [23](#)
  - operator=, [23](#)
  - removeCard, [23](#)
  - removeCardId, [23](#)
- collectionResult

- PacketManager, 9
- common/Error.hpp, 53
- common/Packet.hpp, 53
- common/WizardLogger.cpp, 54
- common/WizardLogger.hpp, 54
- conn
  - client/PacketManager.cpp, 51
  - client/main.cpp, 50
- Connection, 24
  - ~Connection, 24
  - Connection, 24
  - mainLoop, 24
  - sendPacket, 24
- copyDeck
  - Deck, 27
- DECK\_CONT\_ID
  - Packet, 37
- DECK\_LABEL
  - CLI.hpp, 47
- DECK\_REQ\_ID
  - Packet, 37
- DECK\_SIZE
  - Packet.hpp, 54
- DEFAULT\_EMPTY\_SPACE
  - CLI.hpp, 47
- DISCONNECT\_ID
  - Packet, 37
- dataGPlayer, 25
  - cardsInHand, 25
  - cardsPlaced, 25
  - energy, 25
  - limitEnergy, 25
  - maxEnergy, 25
  - playerHeal, 25
  - turn, 25
- dealDamage
  - CardMonster, 17
- Deck, 25
  - addCard, 26, 27
  - copyDeck, 27
  - Deck, 26
  - deleteDeck, 27
  - getDeck, 27
  - getName, 27
  - isValide, 27
  - operator=, 27
  - operator==, 28
  - pickup, 28
- Deck.hpp
  - LIMITNAME, 57
- DeckFull
  - Error.hpp, 53
- deckID
  - Packet : :deckContentPacket, 28
  - Packet : :deckRequestPacket, 28
- defausseCardPlaced
  - PlayerInGame, 41
- deleteDeck
  - Deck, 27
- Display, 29
  - ~Display, 29
  - Display, 29
  - displayCollectionWindow, 29
  - displayDeckWindow, 29
  - displayFatalError, 29
  - displayFriendsWindow, 29
  - displayGame, 30
  - displayLoginPrompt, 30
  - displayLoginResult, 30
  - displayMainWindow, 30
  - displayWait, 30
  - focusTchat, 30
  - valideLogin, 30
- display
  - client/Connection.hpp, 49
  - client/PacketManager.hpp, 52
  - client/main.cpp, 50
- displayCollectionWindow
  - CLI, 20
  - Display, 29
- displayDeckWindow
  - CLI, 20
  - Display, 29
- displayFatalError
  - CLI, 20
  - Display, 29
- displayFriendsWindow
  - CLI, 20
  - Display, 29
- displayGame
  - CLI, 20
  - Display, 30
- displayLoginPrompt
  - CLI, 20
  - Display, 30
- displayLoginResult
  - CLI, 21
  - Display, 30
- displayMainWindow
  - CLI, 21
  - Display, 30
- displayWait
  - CLI, 21
  - Display, 30
- draw
  - Game, 33
  - PlayerInGame, 41
- Effect, 30
  - ~Effect, 31
  - apply, 31
  - canBeApplyOnCard, 31
  - canBeApplyOnPlayer, 31
  - Effect, 31
  - getEffectByID, 31
  - getId, 31
  - isTaunt, 31

- loadAllEffect, [31](#)
- energy
  - dataIGPlayer, [25](#)
- Error
  - Error.hpp, [53](#)
- error
  - WizardLogger, [10](#)
- Error.hpp
  - DeckFull, [53](#)
  - Error, [53](#)
  - MustAttackTaunt, [53](#)
  - NoError, [53](#)
  - NotEffectForMonster, [53](#)
  - NotEffectForPlayer, [53](#)
  - NotEnoughEnergy, [53](#)
  - NotEnoughPlace, [53](#)
  - NotHisTurn, [53](#)
  - TwoSameCardMax, [53](#)
  - UnknowError, [53](#)
- fatal
  - WizardLogger, [10](#)
- focusTchat
  - CLI, [21](#)
  - Display, [30](#)
- FriendsManager, [31](#)
- GAME\_HEIGHT
  - CLI.hpp, [47](#)
- GAME\_LABEL
  - CLI.hpp, [47](#)
- GAME\_WIDTH
  - CLI.hpp, [47](#)
- Game, [32](#)
  - ~Game, [32](#)
  - addPlayerWaitGame, [32](#)
  - attackWithCard, [33](#)
  - attackWithCardAffectPlayer, [33](#)
  - checkDeckAndStart, [33](#)
  - draw, [33](#)
  - Game, [32](#)
  - nextPlayer, [33](#)
  - operator=, [33](#)
  - placeCard, [34](#)
  - placeCardAffectPlayer, [35](#)
  - sendInfoAction, [35](#)
- Game.hpp
  - GameStatut, [58](#)
  - IN\_GAME, [58](#)
  - WAIT\_DEC, [58](#)
- GameStatut
  - Game.hpp, [58](#)
- getAttack
  - CardMonster, [17](#)
- getCardById
  - CardManager, [16](#)
- getCardIndex
  - Collection, [22](#)
- getCardOnIndex
  - Collection, [23](#)
- getCardsId
  - Collection, [23](#)
- getCardsInHand
  - PlayerInGame, [41](#)
- getCardsPlaced
  - PlayerInGame, [41](#)
- getCollection
  - Player, [39](#)
- getDataPlayer
  - PlayerInGame, [41](#)
- getDeck
  - Deck, [27](#)
  - Player, [39](#)
- getDefeats
  - Player, [39](#)
- getDescription
  - Card, [15](#)
- getEffectById
  - Effect, [31](#)
- getEffectID
  - Card, [15](#)
- getEnergyCost
  - Card, [15](#)
- getHP
  - Card, [15](#)
- getHeal
  - PlayerInGame, [41](#)
- getHealed
  - PlayerInGame, [42](#)
- getId
  - Card, [15](#)
  - Effect, [31](#)
- getLife
  - CardMonster, [18](#)
- getListDeck
  - Player, [39](#)
- getMaxHP
  - Card, [15](#)
- getMaxLife
  - CardMonster, [18](#)
- getName
  - Card, [15](#)
  - Deck, [27](#)
  - Player, [39](#)
- getNbrTourPose
  - CardMonster, [18](#)
- getPass
  - Player, [39](#)
- getRanking
  - PlayerManager, [43](#)
- getVictories
  - Player, [39](#)
- gotEffect
  - Card, [15](#)
- haveEnoughEnergy
  - PlayerInGame, [42](#)

- ID
  - Packet : :carteInfosPacket, [19](#)
  - Packet : :carteRequestPacket, [19](#)
  - Packet : :collectionListPacket, [24](#)
  - Packet : :deckContentPacket, [28](#)
  - Packet : :deckRequestPacket, [28](#)
  - Packet : :loginRequestPacket, [35](#)
  - Packet : :loginResultPacket, [36](#)
  - Packet : :packet, [37](#)
- IDList
  - Packet, [37](#)
- IN\_GAME
  - Game.hpp, [58](#)
- incrementTour
  - CardMonster, [18](#)
- indexOfCard
  - Collection, [23](#)
- info
  - WizardLogger, [11](#)
- initGame
  - PacketManager, [9](#)
- initLogger
  - WizardLogger, [11](#)
- isDead
  - CardMonster, [18](#)
  - PlayerInGame, [42](#)
- isDeckDefined
  - PlayerInGame, [42](#)
- isMonster
  - Card, [15](#)
  - CardMonster, [18](#)
- isTaunt
  - CardMonster, [18](#)
  - Effect, [31](#)
- isValide
  - Deck, [27](#)
- json
  - CardManager.hpp, [55](#)
- LIMITNAME
  - Deck.hpp, [57](#)
- LINES
  - CLI.hpp, [47](#)
- LOGGER
  - WizardLogger.hpp, [54](#)
- LOGIN\_REQ\_ID
  - Packet, [37](#)
- LOGIN\_RES\_ID
  - Packet, [37](#)
- limitEnergy
  - dataIGPlayer, [25](#)
- loadAllCards
  - CardManager, [16](#)
- loadAllEffect
  - Effect, [31](#)
- loadPlayers
  - PlayerManager, [43](#)
- login
  - PlayerManager, [43](#)
  - loginResult
    - PacketManager, [9](#)
  - logout
    - Player, [39](#)
- MAIN\_LABEL
  - CLI.hpp, [47](#)
- MAX\_CARTE\_DESCRIPTION\_SIZE
  - Packet.hpp, [54](#)
- MAX\_CARTES
  - client/PacketManager.hpp, [52](#)
  - Packet.hpp, [54](#)
- MAX\_PSEUDO\_SIZE
  - Packet.hpp, [54](#)
- main
  - client/main.cpp, [50](#)
  - server/main.cpp, [51](#)
- mainLoop
  - Connection, [24](#)
- makeLoginRequest
  - PacketManager, [9](#)
- makeRegistrationRequest
  - PacketManager, [9](#)
- manageCollectionRequest
  - PacketManager, [9](#)
- manageDisconnectRequest
  - PacketManager, [9](#)
- managePacket
  - PacketManager, [9](#)
- maxEnergy
  - dataIGPlayer, [25](#)
- MustAttackTaunt
  - Error.hpp, [53](#)
- nbrCardInHand
  - PlayerInGame, [42](#)
- nextPlayer
  - Game, [33](#)
- NoError
  - Error.hpp, [53](#)
- NotEffectForMonster
  - Error.hpp, [53](#)
- NotEffectForPlayer
  - Error.hpp, [53](#)
- NotEnoughEnergy
  - Error.hpp, [53](#)
- NotEnoughPlace
  - Error.hpp, [53](#)
- NotHisTurn
  - Error.hpp, [53](#)
- operator<
  - Player, [39](#)
- operator<<
  - Player, [40](#)
  - Player.cpp, [58](#)
- operator>
  - Player, [39](#)

- operator=
  - Card, [15](#)
  - Collection, [23](#)
  - Deck, [27](#)
  - Game, [33](#)
  - PlayerInGame, [42](#)
- operator==
  - Deck, [28](#)
  - Player, [39](#)
- PANEL\_TOTAL\_NUMBER
  - CLI.hpp, [48](#)
- PLAYER\_INFO\_HEIGHT
  - CLI.hpp, [48](#)
- PLAYER\_INFO\_WIDTH
  - CLI.hpp, [48](#)
- PLAYERS\_DB
  - PlayerManager.hpp, [60](#)
- PORT
  - client/Connection.hpp, [49](#)
  - server/Connection.hpp, [49](#)
- Packet, [36](#)
  - CARTE\_INFO\_ID, [37](#)
  - CARTE\_REQ\_ID, [37](#)
  - COLLECTION\_LIST\_ID, [37](#)
  - COLLECTION\_REQ\_ID, [37](#)
  - DECK\_CONT\_ID, [37](#)
  - DECK\_REQ\_ID, [37](#)
  - DISCONNECT\_ID, [37](#)
  - IDList, [37](#)
  - LOGIN\_REQ\_ID, [37](#)
  - LOGIN\_RES\_ID, [37](#)
  - packetMaxSize, [37](#)
  - packetSize, [37](#)
  - REGIST\_REQ\_ID, [37](#)
  - TCHAT\_CONV\_REQ\_ID, [37](#)
  - TCHAT\_END\_CONV\_ID, [37](#)
  - TCHAT\_END\_REQ\_ID, [37](#)
  - TCHAT\_MESSAGE\_ID, [37](#)
  - TCHAT\_NEW\_CONV\_ID, [37](#)
- Packet.hpp
  - DECK\_SIZE, [54](#)
  - MAX\_CARTE\_DESCRIPTION\_SIZE, [54](#)
  - MAX\_CARTES, [54](#)
  - MAX\_PSEUDO\_SIZE, [54](#)
- Packet : :carteInfosPacket, [18](#)
  - carteID, [19](#)
  - cartesDescription, [19](#)
  - ID, [19](#)
  - size, [19](#)
- Packet : :carteRequestPacket, [19](#)
  - carteID, [19](#)
  - ID, [19](#)
  - size, [19](#)
- Packet : :collectionListPacket, [24](#)
  - cartesList, [24](#)
  - ID, [24](#)
  - size, [24](#)
- Packet : :deckContentPacket, [28](#)
  - cartesList, [28](#)
  - deckID, [28](#)
  - ID, [28](#)
  - size, [28](#)
- Packet : :deckRequestPacket, [28](#)
  - deckID, [28](#)
  - ID, [28](#)
  - size, [29](#)
- Packet : :loginRequestPacket, [35](#)
  - ID, [35](#)
  - password, [35](#)
  - pseudo, [35](#)
  - size, [35](#)
- Packet : :loginResultPacket, [36](#)
  - ID, [36](#)
  - resultCode, [36](#)
  - size, [36](#)
- Packet : :packet, [37](#)
  - ID, [37](#)
  - size, [37](#)
- PacketManager, [9](#)
  - askDefausse, [9](#)
  - collectionResult, [9](#)
  - initGame, [9](#)
  - loginResult, [9](#)
  - makeLoginRequest, [9](#)
  - makeRegistrationRequest, [9](#)
  - manageCollectionRequest, [9](#)
  - manageDisconnectRequest, [9](#)
  - managePacket, [9](#)
  - requestCollection, [9](#)
  - sendAttack, [10](#)
  - sendCard, [10](#)
  - sendDisconnection, [10](#)
  - sendEndGame, [10](#)
  - sendInfoStartTurn, [10](#)
  - setTurn, [10](#)
- packetMaxSize
  - Packet, [37](#)
- packetSize
  - Packet, [37](#)
- password
  - Packet : :loginRequestPacket, [35](#)
- pickup
  - Deck, [28](#)
- placeCard
  - Game, [34](#)
  - PlayerInGame, [42](#)
- placeCardAffectPlayer
  - Game, [35](#)
- Player, [38](#)
  - \_defeats, [40](#)
  - \_victories, [40](#)
  - ~Player, [38](#)
  - addCardCollection, [39](#)
  - adjudicateDefeat, [39](#)
  - adjudicateVictory, [39](#)
  - getCollection, [39](#)



- getDeck, [39](#)
- getDefeats, [39](#)
- getListDeck, [39](#)
- getName, [39](#)
- getPass, [39](#)
- getVictories, [39](#)
- logout, [39](#)
- operator<, [39](#)
- operator<<, [40](#)
- operator>, [39](#)
- operator==, [39](#)
- Player, [38](#)
- recvLoop, [39](#)
- removeDeck, [39](#)
- sendPacket, [39](#)
- serialise, [40](#)
- updateSockfd, [40](#)
- Player.cpp
  - operator<<, [58](#)
- playerHeal
  - dataIGPlayer, [25](#)
- PlayerInGame, [40](#)
  - ~PlayerInGame, [41](#)
  - addDefeat, [41](#)
  - addMaxEnergy, [41](#)
  - addWin, [41](#)
  - defausseCardPlaced, [41](#)
  - draw, [41](#)
  - getCardsInHand, [41](#)
  - getCardsPlaced, [41](#)
  - getDataPlayer, [41](#)
  - getHeal, [41](#)
  - getHealed, [42](#)
  - haveEnoughEnergy, [42](#)
  - isDead, [42](#)
  - isDeckDefined, [42](#)
  - nbrCardInHand, [42](#)
  - operator=, [42](#)
  - placeCard, [42](#)
  - PlayerInGame, [41](#)
  - resetEnergy, [42](#)
  - setDeck, [42](#)
  - takeDamage, [42](#)
- PlayerManager, [42](#)
  - ~PlayerManager, [43](#)
  - getRanking, [43](#)
  - loadPlayers, [43](#)
  - logIn, [43](#)
  - PlayerManager, [43](#)
  - savePlayers, [43](#)
  - signUp, [43](#)
- PlayerManager.hpp
  - PLAYERS\_DB, [60](#)
- pm
  - server/Connection.hpp, [50](#)
  - server/main.cpp, [51](#)
- pseudo
  - Packet : :loginRequestPacket, [35](#)
- REGIST\_REQ\_ID
  - Packet, [37](#)
- recvLoop
  - Player, [39](#)
- removeCard
  - Collection, [23](#)
- removeCardId
  - Collection, [23](#)
- removeDeck
  - Player, [39](#)
- requestCollection
  - PacketManager, [9](#)
- resetEnergy
  - PlayerInGame, [42](#)
- resultCode
  - Packet : :loginResultPacket, [36](#)
- SERVER\_LOGFILE
  - WizardLogger.hpp, [54](#)
- savePlayers
  - PlayerManager, [43](#)
- sendAttack
  - PacketManager, [10](#)
- sendCard
  - PacketManager, [10](#)
- sendDisconnection
  - PacketManager, [10](#)
- sendEndGame
  - PacketManager, [10](#)
- sendInfoAction
  - Game, [35](#)
- sendInfoStartTurn
  - PacketManager, [10](#)
- sendMessage
  - ChatManager, [19](#)
- sendPacket
  - Connection, [24](#)
  - Player, [39](#)
- serialise
  - Player, [40](#)
- server/Card.cpp, [55](#)
- server/Card.hpp, [45](#)
- server/CardManager.cpp, [55](#)
- server/CardManager.hpp, [55](#)
- server/CardMonster.cpp, [55](#)
- server/CardMonster.hpp, [55](#)
- server/ChatManager.hpp, [56](#)
- server/Collection.cpp, [56](#)
- server/Collection.hpp, [56](#)
- server/Connection.cpp, [48](#)
- server/Connection.hpp, [49](#)
  - BACKLOG, [49](#)
  - PORT, [49](#)
  - pm, [50](#)
- server/Deck.cpp, [56](#)
- server/Deck.hpp, [56](#)
- server/Effect.cpp, [57](#)
- server/Effect.hpp, [57](#)
- server/FriendsManager.hpp, [57](#)

- server/Game.cpp, [57](#)
- server/Game.hpp, [58](#)
- server/PackageManager.cpp, [51](#)
- server/PackageManager.hpp, [52](#)
- server/Player.cpp, [58](#)
- server/Player.hpp, [59](#)
- server/PlayerInGame.cpp, [59](#)
- server/PlayerInGame.hpp, [59](#)
- server/PlayerManager.cpp, [59](#)
- server/PlayerManager.hpp, [60](#)
- server/main.cpp, [51](#)
  - main, [51](#)
  - pm, [51](#)
- setAttack
  - CardMonster, [18](#)
- setDeck
  - PlayerInGame, [42](#)
- setLife
  - CardMonster, [18](#)
- setMaxLife
  - CardMonster, [18](#)
- setTaunt
  - CardMonster, [18](#)
- setTurn
  - PackageManager, [10](#)
- signUp
  - PlayerManager, [43](#)
- size
  - Packet : :carteInfosPacket, [19](#)
  - Packet : :carteRequestPacket, [19](#)
  - Packet : :collectionListPacket, [24](#)
  - Packet : :deckContentPacket, [28](#)
  - Packet : :deckRequestPacket, [29](#)
  - Packet : :loginRequestPacket, [35](#)
  - Packet : :loginResultPacket, [36](#)
  - Packet : :packet, [37](#)
- TCHAT\_CONV\_REQ\_ID
  - Packet, [37](#)
- TCHAT\_END\_CONV\_ID
  - Packet, [37](#)
- TCHAT\_END\_REQ\_ID
  - Packet, [37](#)
- TCHAT\_HEIGHT
  - CLI.hpp, [48](#)
- TCHAT\_INPUT\_HEIGHT
  - CLI.hpp, [48](#)
- TCHAT\_INPUT\_WIDTH
  - CLI.hpp, [48](#)
- TCHAT\_MESSAGE\_ID
  - Packet, [37](#)
- TCHAT\_NEW\_CONV\_ID
  - Packet, [37](#)
- TCHAT\_WIDTH
  - CLI.hpp, [48](#)
- takeDamage
  - PlayerInGame, [42](#)
- turn
  - dataIGPlayer, [25](#)
- TwoSameCardMax
  - Error.hpp, [53](#)
- UnknowError
  - Error.hpp, [53](#)
- updateSockfd
  - Player, [40](#)
- valideLogin
  - CLI, [21](#)
  - Display, [30](#)
- WAIT\_DEC
  - Game.hpp, [58](#)
- WAIT\_LABEL
  - CLI.hpp, [48](#)
- warning
  - WizardLogger, [11](#)
- WizardLogger, [10](#)
  - error, [10](#)
  - fatal, [10](#)
  - info, [11](#)
  - initLogger, [11](#)
  - warning, [11](#)
- WizardLogger.hpp
  - CLIENT\_LOGFILE, [54](#)
  - LOGGER, [54](#)
  - SERVER\_LOGFILE, [54](#)