

MonopolyVsAi

1.0.0

Generated by Doxygen 1.9.1



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 ann Namespace Reference	9
5.1.1 Enumeration Type Documentation	9
5.1.1.1 type	9
5.2 neat Namespace Reference	9
<b>6 Class Documentation</b>	<b>11</b>
6.1 ActiveScreen Class Reference	11
6.1.1 Detailed Description	12
6.1.2 Constructor & Destructor Documentation	12
6.1.2.1 ActiveScreen()	12
6.1.3 Member Function Documentation	12
6.1.3.1 addButton()	12
6.1.3.2 addText()	13
6.1.3.3 buttonSetColors()	13
6.1.3.4 draw()	14
6.1.3.5 getButtons()	14
6.1.3.6 getContextWindow()	15
6.1.3.7 getFont()	15
6.1.3.8 getPlayersResult()	16
6.1.3.9 getPlayersSettings()	16
6.1.3.10 getScreenType()	16
6.1.3.11 getTexts()	17
6.1.3.12 setContextWindow()	17
6.1.3.13 setFont()	18
6.1.3.14 setScreenType()	18
6.1.3.15 worker()	19
6.2 AiAdapter Class Reference	19
6.2.1 Detailed Description	20
6.2.2 Constructor & Destructor Documentation	20
6.2.2.1 AiAdapter()	21
6.2.3 Member Function Documentation	21

6.2.3.1 clearSelectionState()	21
6.2.3.2 convertCard()	21
6.2.3.3 convertHouse()	22
6.2.3.4 convertHouseValue()	22
6.2.3.5 convertMoney()	23
6.2.3.6 convertMoneyValue()	23
6.2.3.7 convertPosition()	24
6.2.3.8 getInputs()	25
6.2.3.9 setCard()	25
6.2.3.10 setHouse()	26
6.2.3.11 setJail()	26
6.2.3.12 setMoney()	27
6.2.3.13 setMoneyContext()	27
6.2.3.14 setMortgage()	28
6.2.3.15 setOwner()	28
6.2.3.16 setPosition()	28
6.2.3.17 setSelection()	29
6.2.3.18 setSelectionState()	29
6.2.3.19 setTurn()	29
6.3 AiPlayer Class Reference	30
6.3.1 Detailed Description	31
6.3.2 Constructor & Destructor Documentation	31
6.3.2.1 AiPlayer() [1/3]	31
6.3.2.2 AiPlayer() [2/3]	31
6.3.2.3 AiPlayer() [3/3]	32
6.3.3 Member Function Documentation	32
6.3.3.1 decideAcceptTrade()	33
6.3.3.2 decideAuctionBid()	33
6.3.3.3 decideBuildHouse()	34
6.3.3.4 decideBuy()	35
6.3.3.5 decideJail()	35
6.3.3.6 decideMortgage()	36
6.3.3.7 decideOfferTrade()	37
6.3.3.8 decideSellHouse()	38
6.3.3.9 decideUnmortgage()	38
6.3.3.10 getAdapter()	39
6.3.3.11 getNeuralNetwork()	39
6.3.3.12 setNeuralNetwork()	39
6.4 Board Class Reference	40
6.4.1 Detailed Description	40
6.4.2 Constructor & Destructor Documentation	40
6.4.2.1 Board()	40

6.4.3 Member Function Documentation	41
6.4.3.1 clearBoard()	41
6.4.3.2 getBoard()	41
6.4.3.3 getBoardPosition()	42
6.4.3.4 getFieldById()	42
6.4.3.5 getFieldNumber()	42
6.4.3.6 getFieldPositon()	43
6.4.3.7 getFieldRotation()	43
6.5 Button Class Reference	44
6.5.1 Detailed Description	45
6.5.2 Constructor & Destructor Documentation	45
6.5.2.1 Button()	46
6.5.3 Member Function Documentation	47
6.5.3.1 draw()	47
6.5.3.2 getEventType()	47
6.5.3.3 getIsActive()	47
6.5.3.4 getIsClicked()	48
6.5.3.5 getIsFocus()	48
6.5.3.6 getIsVisible()	48
6.5.3.7 getSize()	48
6.5.3.8 getText()	49
6.5.3.9 getWasReleased()	49
6.5.3.10 isMouseOver()	49
6.5.3.11 mouselsNotOver()	49
6.5.3.12 mouselsOver()	50
6.5.3.13 setActiveBackColor()	50
6.5.3.14 setActiveTextColor()	50
6.5.3.15 setButtonFocus()	51
6.5.3.16 setButtonUnfocus()	51
6.5.3.17 setEventType()	52
6.5.3.18 setFocusBackColor()	52
6.5.3.19 setFocusTextColor()	52
6.5.3.20 setFont()	53
6.5.3.21 setInactiveBackColor()	53
6.5.3.22 setInactiveTextColor()	53
6.5.3.23 setIsActive()	53
6.5.3.24 setIsClicked()	54
6.5.3.25 setIsFocus()	54
6.5.3.26 setIsVisible()	54
6.5.3.27 setPosition()	55
6.5.3.28 setWasReleased()	55
6.5.3.29 updateColors()	55

6.6 ChanceCard Class Reference . . . . .	55
6.6.1 Detailed Description . . . . .	56
6.6.2 Constructor & Destructor Documentation . . . . .	56
6.6.2.1 ChanceCard() . . . . .	57
6.6.3 Member Function Documentation . . . . .	58
6.6.3.1 createSprite() . . . . .	58
6.6.3.2 getGraphicPath() . . . . .	58
6.6.3.3 getHeight() . . . . .	58
6.6.3.4 getId() . . . . .	59
6.6.3.5 getPosition() . . . . .	59
6.6.3.6 getRotation() . . . . .	59
6.6.3.7 getSprite() . . . . .	59
6.6.3.8 getText() . . . . .	60
6.6.3.9 getTexture() . . . . .	60
6.6.3.10 getType() . . . . .	60
6.6.3.11 getValue() . . . . .	61
6.6.3.12 getWidth() . . . . .	61
6.6.3.13 setPosition() . . . . .	61
6.6.3.14 setRotation() . . . . .	62
6.7 ContextWindow Class Reference . . . . .	62
6.7.1 Detailed Description . . . . .	63
6.7.2 Constructor & Destructor Documentation . . . . .	63
6.7.2.1 ContextWindow() . . . . .	63
6.7.3 Member Function Documentation . . . . .	63
6.7.3.1 clear() . . . . .	63
6.7.3.2 display() . . . . .	64
6.7.3.3 GetInstance() . . . . .	64
6.7.3.4 getView() . . . . .	65
6.7.3.5 getWindow() . . . . .	65
6.7.3.6 isOpen() . . . . .	66
6.7.3.7 operator=() . . . . .	66
6.7.4 Member Data Documentation . . . . .	66
6.7.4.1 view_ . . . . .	66
6.7.4.2 window_ . . . . .	66
6.8 DimensionException Class Reference . . . . .	67
6.8.1 Detailed Description . . . . .	67
6.8.2 Constructor & Destructor Documentation . . . . .	67
6.8.2.1 DimensionException() [1/2] . . . . .	68
6.8.2.2 DimensionException() [2/2] . . . . .	68
6.8.3 Member Function Documentation . . . . .	68
6.8.3.1 getBadDimension() . . . . .	68
6.9 Field Class Reference . . . . .	68

6.9.1 Detailed Description . . . . .	69
6.9.2 Constructor & Destructor Documentation . . . . .	69
6.9.2.1 Field() . . . . .	70
6.9.3 Member Function Documentation . . . . .	70
6.9.3.1 createSprite() . . . . .	70
6.9.3.2 getContextWindow() . . . . .	71
6.9.3.3 getGraphicPath() . . . . .	71
6.9.3.4 getHeight() . . . . .	72
6.9.3.5 getId() . . . . .	72
6.9.3.6 getName() . . . . .	73
6.9.3.7 getNameText() . . . . .	73
6.9.3.8 getPosition() . . . . .	74
6.9.3.9 getRotation() . . . . .	74
6.9.3.10 getSprite() . . . . .	75
6.9.3.11 getTexture() . . . . .	75
6.9.3.12 getType() . . . . .	75
6.9.3.13 getWidth() . . . . .	76
6.9.3.14 setHeight() . . . . .	76
6.9.3.15 setPosition() . . . . .	76
6.9.3.16 setRotation() . . . . .	77
6.9.3.17 setWidth() . . . . .	77
6.10 GameEngine Class Reference . . . . .	77
6.10.1 Detailed Description . . . . .	78
6.10.2 Constructor & Destructor Documentation . . . . .	78
6.10.2.1 GameEngine() [1/2] . . . . .	78
6.10.2.2 GameEngine() [2/2] . . . . .	79
6.10.3 Member Function Documentation . . . . .	79
6.10.3.1 clear() . . . . .	80
6.10.3.2 display() . . . . .	80
6.10.3.3 getContextWindow() . . . . .	81
6.10.3.4 getWindow_height() . . . . .	81
6.10.3.5 getWindow_width() . . . . .	82
6.10.3.6 pollForEvents() . . . . .	82
6.10.3.7 worker() . . . . .	83
6.11 GameMenuScreen Class Reference . . . . .	83
6.11.1 Detailed Description . . . . .	85
6.11.2 Constructor & Destructor Documentation . . . . .	85
6.11.2.1 GameMenuScreen() . . . . .	85
6.11.3 Member Function Documentation . . . . .	85
6.11.3.1 buttonClickHandle() . . . . .	85
6.11.3.2 draw() . . . . .	86
6.11.3.3 gameMenuCreate() . . . . .	87

6.11.3.4 getPlayerNumFromEventType()	87
6.11.3.5 getPlayersSettings()	88
6.11.3.6 isEventTypeAILevel()	88
6.11.3.7 isEventTypeSetAI()	89
6.11.3.8 setAILevelColumnVisibility()	89
6.11.3.9 setDefaultAILevelButtonsFocus()	90
6.11.3.10 setOtherButtonsInactive()	91
6.11.3.11 setPlayerSettings()	92
6.11.3.12 worker()	92
6.12 GameScreen Class Reference	93
6.12.1 Constructor & Destructor Documentation	94
6.12.1.1 GameScreen()	94
6.12.2 Member Function Documentation	95
6.12.2.1 draw()	95
6.12.2.2 getPlayersResult()	97
6.12.2.3 worker()	97
6.13 neat::gene Struct Reference	98
6.13.1 Member Data Documentation	99
6.13.1.1 enabled	99
6.13.1.2 from_node	99
6.13.1.3 innovation_num	99
6.13.1.4 to_node	99
6.13.1.5 weight	99
6.14 neat::genome Class Reference	100
6.14.1 Constructor & Destructor Documentation	100
6.14.1.1 genome() [1/2]	100
6.14.1.2 genome() [2/2]	101
6.14.2 Member Data Documentation	101
6.14.2.1 adjusted_fitness	101
6.14.2.2 can_be_recurrent	101
6.14.2.3 fitness	101
6.14.2.4 genes	101
6.14.2.5 global_rank	101
6.14.2.6 max_neuron	101
6.14.2.7 mutation_rates	102
6.14.2.8 network_info	102
6.15 HouseException Class Reference	102
6.15.1 Detailed Description	103
6.15.2 Constructor & Destructor Documentation	103
6.15.2.1 HouseException() [1/2]	103
6.15.2.2 HouseException() [2/2]	103
6.15.3 Member Function Documentation	103



6.15.3.1 getInvalidNumber()	103
6.16 neat::innovation_container Class Reference	103
6.16.1 Constructor & Destructor Documentation	104
6.16.1.1 innovation_container()	104
6.16.2 Member Function Documentation	104
6.16.2.1 add_gene()	104
6.16.2.2 number()	104
6.16.2.3 reset()	104
6.16.3 Friends And Related Function Documentation	104
6.16.3.1 pool	105
6.17 MainMenuScreen Class Reference	105
6.17.1 Detailed Description	106
6.17.2 Constructor & Destructor Documentation	106
6.17.2.1 MainMenuScreen()	106
6.17.3 Member Function Documentation	106
6.17.3.1 draw()	107
6.17.3.2 mainMenuCreate()	107
6.17.3.3 worker()	108
6.18 MonopolyGameEngine Class Reference	108
6.18.1 Detailed Description	112
6.18.2 Constructor & Destructor Documentation	112
6.18.2.1 MonopolyGameEngine()	112
6.18.3 Member Function Documentation	112
6.18.3.1 addAuctionButton()	112
6.18.3.2 addAuctionText()	113
6.18.3.3 addButton()	114
6.18.3.4 addHotels()	115
6.18.3.5 addHouses()	115
6.18.3.6 addText()	116
6.18.3.7 aiBuildingsMangingWorker()	117
6.18.3.8 buildingsManagingWorker()	118
6.18.3.9 calculateGroupFieldsOwned()	119
6.18.3.10 calculateRent()	120
6.18.3.11 clearBoard()	121
6.18.3.12 clearPlayers()	122
6.18.3.13 colorGroupEmpty()	122
6.18.3.14 createAuctionBidButton()	123
6.18.3.15 createAuctionOfferButtons()	124
6.18.3.16 createAuctionResignButton()	124
6.18.3.17 createAvailableHousesHotelText()	125
6.18.3.18 createBoard()	126
6.18.3.19 createButtonBuyResign()	127

6.18.3.20 createButtonNextProperty()	127
6.18.3.21 createButtonPerviousProperty()	128
6.18.3.22 createButtonRollDice()	129
6.18.3.23 createButtonsBankrupt()	130
6.18.3.24 createButtonsBuySellHouseHotel()	130
6.18.3.25 createButtonsJailPay()	131
6.18.3.26 createButtonsNextTurn()	132
6.18.3.27 createButtonWithdraw()	133
6.18.3.28 createChanceCards()	134
6.18.3.29 createCurrentOfferBidderInfo()	135
6.18.3.30 createMortagingButton()	136
6.18.3.31 createPlayers()	136
6.18.3.32 createResultScreenStuff()	137
6.18.3.33 createTextBiddedProperty()	138
6.18.3.34 createTextBidderInfo()	139
6.18.3.35 createTextHighestBidInfo()	139
6.18.3.36 createTextLeadingBidderInfo()	140
6.18.3.37 createTextPlayersInfo()	141
6.18.3.38 createTextrolled_value()	142
6.18.3.39 createTextTurnInfo()	142
6.18.3.40 getAllPropertyDataSprite()	143
6.18.3.41 getAllPropertyDataTexts()	144
6.18.3.42 getAuctionButtons()	144
6.18.3.43 getAuctionState()	145
6.18.3.44 getAuctionTexts()	145
6.18.3.45 getBoard()	146
6.18.3.46 getButtons()	146
6.18.3.47 getFont()	147
6.18.3.48 getFontSize()	148
6.18.3.49 getHotelCount()	149
6.18.3.50 getHotelSprite()	150
6.18.3.51 getHotelTexture()	151
6.18.3.52 getHouseCount()	151
6.18.3.53 getHouseSize()	152
6.18.3.54 getHouseSprite()	152
6.18.3.55 getHouseTexture()	153
6.18.3.56 getNotificationsWall()	154
6.18.3.57 getPlayers()	155
6.18.3.58 getPlayersResult()	155
6.18.3.59 getPropertyDataSprite()	156
6.18.3.60 getPropertyDataTexts()	156
6.18.3.61 getPropertyNamesToDraw()	156

6.18.3.62 getResultTexts()	157
6.18.3.63 getScreenType()	158
6.18.3.64 getTexts()	158
6.18.3.65 getTurnState()	159
6.18.3.66 getWithdraw()	159
6.18.3.67 groupCompleted()	160
6.18.3.68 isBuildingLegal()	161
6.18.3.69 isDestroyingLegal()	162
6.18.3.70 isHotelBuildingLegal()	163
6.18.3.71 isHotelDestroyingLegal()	164
6.18.3.72 monopolyGameWorker()	165
6.18.3.73 performAuction()	167
6.18.3.74 setAuctionState()	168
6.18.3.75 setFont()	168
6.18.3.76 setHotelCount()	168
6.18.3.77 setHouseCount()	169
6.18.3.78 setplayer_index_turn()	169
6.18.3.79 setScreenType()	170
6.18.3.80 showPropertyData()	170
6.18.3.81 shuffleChanceCards()	172
6.18.3.82 subtractHotels()	172
6.18.3.83 subtractHouses()	173
6.18.3.84 turnInfoTextShow()	174
6.18.3.85 updateAvailableHousesHotelText()	174
6.18.3.86 updateResultScreenStuff()	175
6.18.3.87 updateTextPlayersInfo()	176
6.19 neat::mutation_rate_container Struct Reference	176
6.19.1 Member Function Documentation	177
6.19.1.1 read()	177
6.19.1.2 write()	177
6.19.2 Member Data Documentation	177
6.19.2.1 bias_mutation_chance	178
6.19.2.2 connection_mutate_chance	178
6.19.2.3 crossover_chance	178
6.19.2.4 disable_mutation_chance	178
6.19.2.5 enable_mutation_chance	178
6.19.2.6 link_mutation_chance	178
6.19.2.7 node_mutation_chance	178
6.19.2.8 perturb_chance	178
6.19.2.9 step_size	179
6.20 neat::network_info_container Struct Reference	179
6.20.1 Member Data Documentation	179

6.20.1.1 bias_size	179
6.20.1.2 functional_nodes	179
6.20.1.3 input_size	179
6.20.1.4 output_size	180
6.20.1.5 recurrent	180
6.21 ann::neuralnet Class Reference	180
6.21.1 Constructor & Destructor Documentation	180
6.21.1.1 neuralnet()	180
6.21.2 Member Function Documentation	180
6.21.2.1 evaluate()	181
6.21.2.2 export_tofile()	181
6.21.2.3 from_genome()	181
6.21.2.4 import_fromfile()	182
6.22 ann::neuron Class Reference	182
6.22.1 Constructor & Destructor Documentation	182
6.22.1.1 neuron()	182
6.22.1.2 ~neuron()	183
6.22.2 Member Data Documentation	183
6.22.2.1 in_nodes	183
6.22.2.2 type	183
6.22.2.3 value	183
6.22.2.4 visited	183
6.23 NotificationWall Class Reference	183
6.23.1 Detailed Description	184
6.23.2 Constructor & Destructor Documentation	184
6.23.2.1 NotificationWall()	184
6.23.3 Member Function Documentation	184
6.23.3.1 addToWall()	184
6.23.3.2 clearWall()	185
6.23.3.3 getFont()	185
6.23.3.4 getFontSize()	186
6.23.3.5 getWall()	186
6.23.3.6 setFont()	187
6.24 Player Class Reference	187
6.24.1 Detailed Description	189
6.24.2 Constructor & Destructor Documentation	189
6.24.2.1 Player() [1/2]	189
6.24.2.2 Player() [2/2]	190
6.24.3 Member Function Documentation	190
6.24.3.1 addFieldOwnedId()	190
6.24.3.2 addMoney()	191
6.24.3.3 clearFieldOwnedId()	191

6.24.3.4 createSprite()	192
6.24.3.5 decideAcceptTrade()	192
6.24.3.6 decideAuctionBid()	192
6.24.3.7 decideBuildHouse()	192
6.24.3.8 decideBuy()	193
6.24.3.9 decideJail()	193
6.24.3.10 decideMortgage()	193
6.24.3.11 decideOfferTrade()	194
6.24.3.12 decideSellHouse()	194
6.24.3.13 decideUnmortgage()	194
6.24.3.14 getAdapter()	195
6.24.3.15 getAiLevel()	195
6.24.3.16 getColor()	196
6.24.3.17 getFieldOwnedId()	196
6.24.3.18 getId()	196
6.24.3.19 getIsAi()	196
6.24.3.20 getJailCards()	196
6.24.3.21 getJailStatus()	197
6.24.3.22 getMoney()	197
6.24.3.23 getNeuralNetwork()	197
6.24.3.24 getPosition()	198
6.24.3.25 getResultPlace()	198
6.24.3.26 getSprite()	198
6.24.3.27 getSpriteOffsetX()	198
6.24.3.28 getSpriteOffsetY()	198
6.24.3.29 getTexture()	198
6.24.3.30 hasFieldOwnedId()	198
6.24.3.31 reduceJailStatus()	199
6.24.3.32 removeFieldOwnedId()	199
6.24.3.33 setAiLevel()	199
6.24.3.34 setColor()	200
6.24.3.35 setId()	200
6.24.3.36 setIsAi()	201
6.24.3.37 setJailCards()	202
6.24.3.38 setJailStatus()	202
6.24.3.39 setMoney()	203
6.24.3.40 setPosition()	203
6.24.3.41 setResultPlace()	204
6.24.3.42 setSpriteOffsetX()	204
6.24.3.43 setSpriteOffsetY()	205
6.24.3.44 setSpritePosition()	205
6.24.3.45 subtractMoney()	205

6.25 playerSettings Struct Reference	206
6.25.1 Detailed Description	206
6.25.2 Member Data Documentation	206
6.25.2.1 isHuman	206
6.25.2.2 isNone	206
6.25.2.3 level	206
6.26 neat::pool Class Reference	207
6.26.1 Constructor & Destructor Documentation	207
6.26.1.1 pool()	207
6.26.2 Member Function Documentation	208
6.26.2.1 export_tofile()	208
6.26.2.2 generation()	208
6.26.2.3 get_genomes()	208
6.26.2.4 import_fromfile()	208
6.26.2.5 new_generation()	208
6.26.3 Member Data Documentation	209
6.26.3.1 generator	209
6.26.3.2 max_fitness	209
6.26.3.3 mutation_rates	209
6.26.3.4 network_info	209
6.26.3.5 rd	209
6.26.3.6 speciating_parameters	209
6.26.3.7 species	209
6.27 PropertyField Class Reference	210
6.27.1 Detailed Description	211
6.27.2 Constructor & Destructor Documentation	211
6.27.2.1 PropertyField()	211
6.27.3 Member Function Documentation	212
6.27.3.1 createFlagSprite()	212
6.27.3.2 getGroupMembers()	213
6.27.3.3 getIsMortgaged()	213
6.27.3.4 getMortgage()	214
6.27.3.5 getOwner()	215
6.27.3.6 getOwnerFlag()	215
6.27.3.7 getPrice()	216
6.27.3.8 getUnmortgageValue()	216
6.27.3.9 resetDefault()	217
6.27.3.10 resetOwner()	217
6.27.3.11 setIsMortgaged()	217
6.27.3.12 setOwner()	218
6.28 RotationException Class Reference	218
6.28.1 Detailed Description	219

6.28.2 Constructor & Destructor Documentation	219
6.28.2.1 RotationException() [1/2]	219
6.28.2.2 RotationException() [2/2]	220
6.28.3 Member Function Documentation	220
6.28.3.1 getBadRotation()	220
6.29 neat::speciating_parameter_container Struct Reference	220
6.29.1 Member Function Documentation	220
6.29.1.1 read()	220
6.29.1.2 write()	221
6.29.2 Member Data Documentation	221
6.29.2.1 delta_disjoint	221
6.29.2.2 delta_threshold	221
6.29.2.3 delta_weights	221
6.29.2.4 population	221
6.29.2.5 stale_species	221
6.30 neat::specie Struct Reference	222
6.30.1 Member Data Documentation	222
6.30.1.1 average_fitness	222
6.30.1.2 genomes	222
6.30.1.3 staleness	222
6.30.1.4 top_fitness	222
6.31 SpriteOffsetException Class Reference	223
6.31.1 Detailed Description	223
6.31.2 Constructor & Destructor Documentation	223
6.31.2.1 SpriteOffsetException() [1/2]	224
6.31.2.2 SpriteOffsetException() [2/2]	224
6.31.3 Member Function Documentation	224
6.31.3.1 getBadOffset()	224
6.32 StationField Class Reference	224
6.32.1 Detailed Description	225
6.32.2 Constructor & Destructor Documentation	225
6.32.2.1 StationField()	225
6.32.3 Member Function Documentation	226
6.32.3.1 calculateRent()	226
6.32.3.2 getRentValues()	226
6.33 StreetField Class Reference	227
6.33.1 Detailed Description	228
6.33.2 Constructor & Destructor Documentation	229
6.33.2.1 StreetField()	229
6.33.3 Member Function Documentation	229
6.33.3.1 calculateRent()	229
6.33.3.2 getHotelPrice()	230

6.33.3.3 getHouseNumber()	230
6.33.3.4 getHousePrice()	231
6.33.3.5 getIsHotel()	231
6.33.3.6 getRentValues()	232
6.33.3.7 resetDefault()	232
6.33.3.8 setHouseNumber()	232
6.33.3.9 setIsHotel()	233
6.34 TaxField Class Reference	233
6.34.1 Detailed Description	234
6.34.2 Constructor & Destructor Documentation	234
6.34.2.1 TaxField()	235
6.34.3 Member Function Documentation	235
6.34.3.1 getTaxValue()	235
6.35 UtilityField Class Reference	236
6.35.1 Constructor & Destructor Documentation	237
6.35.1.1 UtilityField()	237
6.35.2 Member Function Documentation	237
6.35.2.1 calculateRent()	237
6.35.2.2 getRentMultipliers()	238
6.36 Withdraw Class Reference	238
6.36.1 Detailed Description	242
6.36.2 Constructor & Destructor Documentation	242
6.36.2.1 Withdraw()	242
6.36.3 Member Function Documentation	242
6.36.3.1 addPropertyPlayerShowed()	242
6.36.3.2 createChoosePlayerScreen()	243
6.36.3.3 createDecisionPlayerScreen()	243
6.36.3.4 createValuePlayerScreen()	244
6.36.3.5 getAcceptDecisionButton()	244
6.36.3.6 getButtons()	245
6.36.3.7 getFont()	245
6.36.3.8 getFontSize()	246
6.36.3.9 getPlayer1AddButton()	246
6.36.3.10 getPlayer1Button()	246
6.36.3.11 getPlayer1IndexNextButton()	247
6.36.3.12 getPlayer1IndexPreviousButton()	247
6.36.3.13 getPlayer1IndexProperties()	247
6.36.3.14 getPlayer1minus1()	247
6.36.3.15 getPlayer1minus10()	248
6.36.3.16 getPlayer1minus100()	248
6.36.3.17 getPlayer1NextButton()	248
6.36.3.18 getPlayer1plus1()	248



6.36.3.19 getPlayer1plus10()	249
6.36.3.20 getPlayer1plus100()	249
6.36.3.21 getPlayer1PreviousButton()	249
6.36.3.22 getPlayer1 RemoveButton()	249
6.36.3.23 getPlayer1 ToWithdraw()	250
6.36.3.24 getPlayer2AddButton()	250
6.36.3.25 getPlayer2Button()	250
6.36.3.26 getPlayer2IndexNextButton()	250
6.36.3.27 getPlayer2IndexPreviousButton()	251
6.36.3.28 getPlayer2IndexProperties()	251
6.36.3.29 getPlayer2minus1()	251
6.36.3.30 getPlayer2minus10()	251
6.36.3.31 getPlayer2minus100()	252
6.36.3.32 getPlayer2NextButton()	252
6.36.3.33 getPlayer2plus1()	252
6.36.3.34 getPlayer2plus10()	252
6.36.3.35 getPlayer2plus100()	253
6.36.3.36 getPlayer2PreviousButton()	253
6.36.3.37 getPlayer2RemoveButton()	253
6.36.3.38 getPlayer2ToWithdraw()	253
6.36.3.39 getPlayer3Button()	254
6.36.3.40 getPlayer4Button()	254
6.36.3.41 getResignButton()	254
6.36.3.42 getResignDecisionButton()	254
6.36.3.43 getResignValueButton()	255
6.36.3.44 getSpritePropertyPlayer1()	255
6.36.3.45 getSpritePropertyPlayer1 Index()	255
6.36.3.46 getSpritePropertyPlayer2()	256
6.36.3.47 getSpritePropertyPlayer2 Index()	256
6.36.3.48 getSubmitValueButton()	257
6.36.3.49 getTextures()	257
6.36.3.50 getTexturesPropertyPlayer1()	257
6.36.3.51 getTexturesPropertyPlayer1 Index()	258
6.36.3.52 getTexturesPropertyPlayer2()	258
6.36.3.53 getTexturesPropertyPlayer2 Index()	259
6.36.3.54 getTexturePropertyPlayer1()	259
6.36.3.55 getTexturePropertyPlayer1 Index()	260
6.36.3.56 getTexturePropertyPlayer2()	260
6.36.3.57 getTexturePropertyPlayer2 Index()	261
6.36.3.58 getTurnState()	261
6.36.3.59 isNonZeroValue()	261
6.36.3.60 makeWithdraw()	262

6.36.3.61 moneyTextUpdate()	262
6.36.3.62 moneyTransferIndex()	262
6.36.3.63 propertyPlayerMoveIndex()	263
6.36.3.64 setBoard()	263
6.36.3.65 setChooseScreenVisible()	264
6.36.3.66 setDecisionScreenVisible()	264
6.36.3.67 setFont()	265
6.36.3.68 setPlayer1IndexProperties()	266
6.36.3.69 setPlayer1ToWithdraw()	266
6.36.3.70 setPlayer2IndexProperties()	266
6.36.3.71 setPlayer2ToWithdraw()	266
6.36.3.72 setTurnState()	267
6.36.3.73 setValueScreenVisible()	267
6.36.3.74 showProperty()	268
<b>7 File Documentation</b>	<b>271</b>
7.1 /home/kamil/zpr/Monopoly/ActiveScreen.cc File Reference	271
7.1.1 Detailed Description	271
7.2 /home/kamil/zpr/Monopoly/ActiveScreen.h File Reference	272
7.2.1 Detailed Description	273
7.3 /home/kamil/zpr/Monopoly/AiAdapter.cc File Reference	273
7.3.1 Detailed Description	273
7.4 /home/kamil/zpr/Monopoly/AiAdapter.h File Reference	274
7.4.1 Detailed Description	274
7.5 /home/kamil/zpr/Monopoly/Board.cc File Reference	275
7.5.1 Detailed Description	275
7.5.2 Typedef Documentation	275
7.5.2.1 json	275
7.5.3 Function Documentation	276
7.5.3.1 jsonToStationRent()	276
7.5.3.2 jsonToStreetRent()	276
7.5.3.3 jsonToUtilityRent()	277
7.6 /home/kamil/zpr/Monopoly/Board.h File Reference	277
7.6.1 Detailed Description	279
7.6.2 Typedef Documentation	279
7.6.2.1 json	279
7.6.2.2 PossibleFields	279
7.6.3 Function Documentation	279
7.6.3.1 jsonToStationRent()	279
7.6.3.2 jsonToStreetRent()	280
7.6.3.3 jsonToUtilityRent()	281
7.7 /home/kamil/zpr/Monopoly/Button.cc File Reference	281

7.8 /home/kamil/zpr/Monopoly/Button.h File Reference . . . . .	282
7.8.1 Detailed Description . . . . .	283
7.9 /home/kamil/zpr/Monopoly/Chance.cc File Reference . . . . .	283
7.9.1 Detailed Description . . . . .	283
7.10 /home/kamil/zpr/Monopoly/Chance.h File Reference . . . . .	284
7.10.1 Detailed Description . . . . .	285
7.10.2 Enumeration Type Documentation . . . . .	285
7.10.2.1 ChanceType . . . . .	285
7.11 /home/kamil/zpr/Monopoly/ContextWindow.cc File Reference . . . . .	285
7.11.1 Detailed Description . . . . .	286
7.12 /home/kamil/zpr/Monopoly/ContextWindow.h File Reference . . . . .	286
7.12.1 Detailed Description . . . . .	287
7.13 /home/kamil/zpr/Monopoly/Field.cc File Reference . . . . .	287
7.13.1 Detailed Description . . . . .	288
7.14 /home/kamil/zpr/Monopoly/Field.h File Reference . . . . .	288
7.14.1 Detailed Description . . . . .	289
7.15 /home/kamil/zpr/Monopoly/GameEngine.cc File Reference . . . . .	290
7.15.1 Detailed Description . . . . .	290
7.16 /home/kamil/zpr/Monopoly/GameEngine.h File Reference . . . . .	290
7.16.1 Detailed Description . . . . .	291
7.17 /home/kamil/zpr/Monopoly/GameScreen.cc File Reference . . . . .	291
7.17.1 Detailed Description . . . . .	292
7.18 /home/kamil/zpr/Monopoly/GameScreen.h File Reference . . . . .	292
7.18.1 Detailed Description . . . . .	293
7.19 /home/kamil/zpr/Monopoly/main.cc File Reference . . . . .	293
7.19.1 Detailed Description . . . . .	293
7.19.2 Function Documentation . . . . .	294
7.19.2.1 main() . . . . .	294
7.19.3 Variable Documentation . . . . .	294
7.19.3.1 FRAMES_PER_SEC_MAX . . . . .	294
7.19.3.2 GAMES_IN_ROUND . . . . .	294
7.19.3.3 height . . . . .	294
7.19.3.4 HEIGHT_MAX . . . . .	294
7.19.3.5 TRAIN . . . . .	294
7.19.3.6 width . . . . .	295
7.19.3.7 WIDTH_MAX . . . . .	295
7.20 /home/kamil/zpr/Monopoly/main.h File Reference . . . . .	295
7.20.1 Detailed Description . . . . .	296
7.20.2 Enumeration Type Documentation . . . . .	297
7.20.2.1 ActiveScreenType . . . . .	297
7.20.2.2 BuyDecision . . . . .	297
7.20.2.3 Decision . . . . .	297

7.20.2.4 FieldType	297
7.20.2.5 GameScreenType	299
7.20.2.6 JailDecision	299
7.20.2.7 ScreenEventType	299
7.20.2.8 StationTiers	300
7.20.2.9 StreetTiers	301
7.20.2.10 TurnState	301
7.20.2.11 UtilityTiers	301
7.21 /home/kamil/zpr/Monopoly/MonopolyGameEngine.cc File Reference	302
7.21.1 Detailed Description	302
7.22 /home/kamil/zpr/Monopoly/MonopolyGameEngine.h File Reference	302
7.22.1 Detailed Description	303
7.22.2 Enumeration Type Documentation	304
7.22.2.1 AuctionState	304
7.23 /home/kamil/zpr/Monopoly/NotificationWall.cc File Reference	304
7.23.1 Detailed Description	304
7.24 /home/kamil/zpr/Monopoly/NotificationWall.h File Reference	305
7.24.1 Detailed Description	306
7.25 /home/kamil/zpr/Monopoly/Player.cc File Reference	306
7.25.1 Detailed Description	306
7.26 /home/kamil/zpr/Monopoly/Player.h File Reference	306
7.26.1 Detailed Description	307
7.27 /home/kamil/zpr/Monopoly/Tinyann.cc File Reference	308
7.28 /home/kamil/zpr/Monopoly/Tinyann.h File Reference	308
7.28.1 Detailed Description	309
7.29 /home/kamil/zpr/Monopoly/Tinyneat.cc File Reference	310
7.29.1 Detailed Description	310
7.30 /home/kamil/zpr/Monopoly/Tinyneat.h File Reference	310
7.30.1 Detailed Description	311
7.31 /home/kamil/zpr/Monopoly/Withdraw.cc File Reference	312
7.31.1 Detailed Description	312
7.32 /home/kamil/zpr/Monopoly/Withdraw.h File Reference	312
7.32.1 Detailed Description	313
<b>Index</b>	<b>315</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">ann</a>	.....	<a href="#">9</a>
<a href="#">neat</a>	.....	<a href="#">9</a>



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

ActiveScreen . . . . .	11
GameMenuScreen . . . . .	83
GameScreen . . . . .	93
MainMenuScreen . . . . .	105
AiAdapter . . . . .	19
Board . . . . .	40
Button . . . . .	44
ChanceCard . . . . .	55
ContextWindow . . . . .	62
std::exception	
DimensionException . . . . .	67
HouseException . . . . .	102
RotationException . . . . .	218
SpriteOffsetException . . . . .	223
Field . . . . .	68
PropertyField . . . . .	210
StationField . . . . .	224
StreetField . . . . .	227
UtilityField . . . . .	236
TaxField . . . . .	233
GameEngine . . . . .	77
neat::gene . . . . .	98
neat::genome . . . . .	100
neat::innovation_container . . . . .	103
MonopolyGameEngine . . . . .	108
neat::mutation_rate_container . . . . .	176
neat::network_info_container . . . . .	179
ann::neuralnet . . . . .	180
ann::neuron . . . . .	182
NotificationWall . . . . .	183
Player . . . . .	187
AiPlayer . . . . .	30
playerSettings . . . . .	206
neat::pool . . . . .	207
neat::speciating_parameter_container . . . . .	220
neat::specie . . . . .	222
Withdraw . . . . .	238





## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">ActiveScreen</a>	Represents the base class for handling displayed screens in the project . . . . .	11
<a href="#">AiAdapter</a>	Represents an adapter for AI input data . . . . .	19
<a href="#">AiPlayer</a>	Represents an AI player in a Monopoly game, inheriting from <a href="#">Player</a> . . . . .	30
<a href="#">Board</a>	Class representing the monopoly game board . . . . .	40
<a href="#">Button</a>	Represents a button for handling user interactions . . . . .	44
<a href="#">ChanceCard</a>	Represents a chance card in the monopoly game . . . . .	55
<a href="#">ContextWindow</a>	Represents a Singleton class for handling SFML window operations . . . . .	62
<a href="#">DimensionException</a>	Exception for handling passing wrong dimensions to any displayed object . . . . .	67
<a href="#">Field</a>	Base class representing a generic game field . . . . .	68
<a href="#">GameEngine</a>	Handles low-level program operations, including input interactions and window display . . . . .	77
<a href="#">GameMenuScreen</a>	Represents the screen for the game menu . . . . .	83
<a href="#">GameScreen</a>	. . . . .	93
<a href="#">neat::gene</a>	. . . . .	98
<a href="#">neat::genome</a>	. . . . .	100
<a href="#">HouseException</a>	Custom exception class for handling invalid house numbers . . . . .	102
<a href="#">neat::innovation_container</a>	. . . . .	103
<a href="#">MainMenuScreen</a>	Represents the screen for the main menu . . . . .	105
<a href="#">MonopolyGameEngine</a>	Class representing the main game engine for the Monopoly game . . . . .	108
<a href="#">neat::mutation_rate_container</a>	. . . . .	176
<a href="#">neat::network_info_container</a>	. . . . .	179
<a href="#">ann::neuralnet</a>	. . . . .	180

<a href="#">ann::neuron</a>	182
<a href="#">NotificationWall</a>	
Represents a notification wall that displays messages	183
<a href="#">Player</a>	
Represents a player in a Monopoly game	187
<a href="#">playerSettings</a>	
Struct describing player settings manipulated from game menu	206
<a href="#">neat::pool</a>	207
<a href="#">PropertyField</a>	
Derived class representing a property field on the game board	210
<a href="#">RotationException</a>	
Exception for handling passing wrong rotation to any displayed object	218
<a href="#">neat::speciating_parameter_container</a>	220
<a href="#">neat::specie</a>	222
<a href="#">SpriteOffsetException</a>	
Exception for handling wrong offset to any displayed object	223
<a href="#">StationField</a>	
Derived class representing a station field on the game board	224
<a href="#">StreetField</a>	
Derived class representing a street field on the game board	227
<a href="#">TaxField</a>	
Derived class representing a tax field on the monopoly board	233
<a href="#">UtilityField</a>	236
<a href="#">Withdraw</a>	
Represents the trade and withdraw mechanism in a monopoly game	238

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

/home/kamil/zpr/Monopoly/ <a href="#">ActiveScreen.cc</a>	
Source file handling displayed screens of project Base class is <a href="#">ActiveScreen</a> , then derived class are used to work with specific screen shown . . . . .	271
/home/kamil/zpr/Monopoly/ <a href="#">ActiveScreen.h</a>	
Header file handling displayed screens of project Base class is <a href="#">ActiveScreen</a> , then derived class are used to work with specific screen shown . . . . .	272
/home/kamil/zpr/Monopoly/ <a href="#">AiAdapter.cc</a>	
TODO . . . . .	273
/home/kamil/zpr/Monopoly/ <a href="#">AiAdapter.h</a>	
TODO . . . . .	274
/home/kamil/zpr/Monopoly/ <a href="#">Board.cc</a>	
Source file for monopoly game board, creation is based on json file 'board.json'. Mainly handles all types of fields and their usage . . . . .	275
/home/kamil/zpr/Monopoly/ <a href="#">Board.h</a>	
Header file for monopoly game board, creation is based on json file 'board.json'. Mainly handles all types of fields and their usage . . . . .	277
/home/kamil/zpr/Monopoly/ <a href="#">Button.cc</a>	281
/home/kamil/zpr/Monopoly/ <a href="#">Button.h</a>	
Source file for handling button objects actions used to communicate with user . . . . .	282
/home/kamil/zpr/Monopoly/ <a href="#">Chance.cc</a>	
Source file for monopoly game chance cards, their types, actions.. . . .	283
/home/kamil/zpr/Monopoly/ <a href="#">Chance.h</a>	
Header file for monopoly game chance cards, their types, actions.. . . .	284
/home/kamil/zpr/Monopoly/ <a href="#">ContextWindow.cc</a>	
Source file for context Window class It is Singleton class type used mainly for handling SFML window operations between other classes . . . . .	285
/home/kamil/zpr/Monopoly/ <a href="#">ContextWindow.h</a>	
Source file for context Window class It is Singleton class type used mainly for handling SFML window operations between other classes . . . . .	286
/home/kamil/zpr/Monopoly/ <a href="#">Field.cc</a>	
Source file containing staff for single monopoly board game field. Separated to many deriving classes types each for specific field type . . . . .	287
/home/kamil/zpr/Monopoly/ <a href="#">Field.h</a>	
Header file containing staff for single monopoly board game field. Separated to many deriving classes types each for specific field type . . . . .	288

/home/kamil/zpr/Monopoly/ <a href="#">GameEngine.cc</a>	Source file for <a href="#">GameEngine</a> class, used to handle low level program operations as input interactions (mouse, keyboard) or display window . . . . .	290
/home/kamil/zpr/Monopoly/ <a href="#">GameEngine.h</a>	Header file for <a href="#">GameEngine</a> class, used to handle low level program operations as input interactions (mouse, keyboard) or display window . . . . .	290
/home/kamil/zpr/Monopoly/ <a href="#">GameScreen.cc</a>	Source file for game screen class deriving from <a href="#">ActiveScreen</a> class. Used to handle monopoly game activities and drawing . . . . .	291
/home/kamil/zpr/Monopoly/ <a href="#">GameScreen.h</a>	Header file for game screen class deriving from <a href="#">ActiveScreen</a> class. Used to handle monopoly game activities and drawing . . . . .	292
/home/kamil/zpr/Monopoly/ <a href="#">main.cc</a>	Source file launching monopoly game vs AI project . . . . .	293
/home/kamil/zpr/Monopoly/ <a href="#">main.h</a>	Header file containing structures shared between project files . . . . .	295
/home/kamil/zpr/Monopoly/ <a href="#">MonopolyGameEngine.cc</a>	Source file of class used to handle whole monopoly game process, turns, actions with players, board etc . . . . .	302
/home/kamil/zpr/Monopoly/ <a href="#">MonopolyGameEngine.h</a>	Header file of class used to handle whole monopoly game process, turns, actions with players, board etc . . . . .	302
/home/kamil/zpr/Monopoly/ <a href="#">NotificationWall.cc</a>	Source file for the <a href="#">NotificationWall</a> class . . . . .	304
/home/kamil/zpr/Monopoly/ <a href="#">NotificationWall.h</a>	Header file for the <a href="#">NotificationWall</a> class . . . . .	305
/home/kamil/zpr/Monopoly/ <a href="#">Player.cc</a>	Implementation file for <a href="#">Player</a> class and AI <a href="#">Player</a> class, containing data and methods for a player in a Monopoly game . . . . .	306
/home/kamil/zpr/Monopoly/ <a href="#">Player.h</a>	Implementation file for <a href="#">Player</a> class and AI <a href="#">Player</a> class, containing data and methods for a player in a Monopoly game . . . . .	306
/home/kamil/zpr/Monopoly/ <a href="#">Tinyann.cc</a>		308
/home/kamil/zpr/Monopoly/ <a href="#">Tinyann.h</a>	Header file for AI implementation of tiny library from github user hav4ik, repository: <a href="https://github.com/hav4ik/tinyai">https://github.com/hav4ik/tinyai</a> . . . . .	308
/home/kamil/zpr/Monopoly/ <a href="#">Tinyneat.cc</a>	Source file for AI implementation of tiny library from github user hav4ik, repository: <a href="https://github.com/hav4ik/tinyai">https://github.com/hav4ik/tinyai</a> . . . . .	310
/home/kamil/zpr/Monopoly/ <a href="#">Tinyneat.h</a>	Header file for AI implementation of tiny library from github user hav4ik, repository: <a href="https://github.com/hav4ik/tinyai">https://github.com/hav4ik/tinyai</a> . . . . .	310
/home/kamil/zpr/Monopoly/ <a href="#">Withdraw.cc</a>	Source file for trade/withdraw mechanism in monopoly game between players . . . . .	312
/home/kamil/zpr/Monopoly/ <a href="#">Withdraw.h</a>	Header file for trade/withdraw mechanism in the monopoly game between players . . . . .	312

## Chapter 5

# Namespace Documentation

### 5.1 ann Namespace Reference

#### Classes

- class [neuron](#)
- class [neuralnet](#)

#### Enumerations

- enum [type](#) { [RECURRENT](#) , [NON\\_RECURRENT](#) }

#### 5.1.1 Enumeration Type Documentation

##### 5.1.1.1 type

enum [ann::type](#)

##### Enumerator

RECURRENT	
NON_RECURRENT	

### 5.2 neat Namespace Reference

#### Classes

- struct [mutation\\_rate\\_container](#)

- struct [speciating\\_parameter\\_container](#)
- struct [network\\_info\\_container](#)
- struct [gene](#)
- class [genome](#)
- struct [specie](#)
- class [innovation\\_container](#)
- class [pool](#)

## Chapter 6

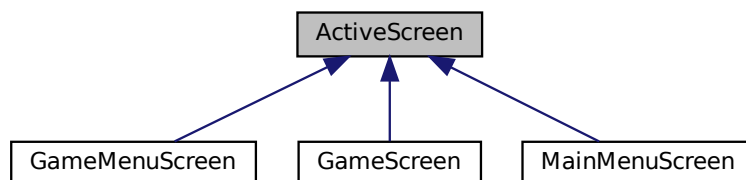
# Class Documentation

### 6.1 ActiveScreen Class Reference

Represents the base class for handling displayed screens in the project.

```
#include <ActiveScreen.h>
```

Inheritance diagram for ActiveScreen:



#### Public Member Functions

- **ActiveScreen** ()  
*Constructor for the **ActiveScreen** class.*
- virtual **ScreenEventType worker** ()=0  
*Pure virtual function representing the worker function for the screen.*
- virtual void **draw** ()=0  
*Pure virtual function to draw the screen.*
- sf::Font & **getFont** ()  
*Gets the SFML font used for text on the screen.*
- void **setFont** (sf::Font font)  
*Sets the SFML font used for text on the screen.*
- void **addButton** (std::shared\_ptr< **Button** > button\_tmp)  
*Adds a button to the screen.*
- void **addText** (std::shared\_ptr< sf::Text > text\_tmp)

- Adds a text object to the screen.*
  - `std::vector< std::shared_ptr< Button > > & getButtons ()`
- Gets the vector of buttons displayed on the screen.*
  - `std::vector< std::shared_ptr< sf::Text > > & getTexts ()`
- Gets the vector of SFML Text objects displayed on the screen.*
  - `ContextWindow * getContextWindow ()`
- Gets the pointer to the [ContextWindow](#).*
  - `void setContextWindow (ContextWindow *)`
- Sets the pointer to the [ContextWindow](#).*
  - `ActiveScreenType getScreenType ()`
- Gets the type of the active screen.*
  - `void setScreenType (ActiveScreenType type)`
- Sets the type of the active screen.*
  - `void buttonSetColors (std::shared_ptr< Button > buttonPtr)`
- Sets the colors of a button based on its state.*
  - `virtual std::vector< std::shared_ptr< playerSettings > > getPlayersSettings () const`
- Virtual function to get players' settings.*
  - `virtual std::vector< std::shared_ptr< Player > > getPlayersResult ()`
- Virtual function to get players' results.*

### 6.1.1 Detailed Description

Represents the base class for handling displayed screens in the project.

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 [ActiveScreen\(\)](#)

```
ActiveScreen::ActiveScreen ( )
```

Constructor for the [ActiveScreen](#) class.

### 6.1.3 Member Function Documentation

#### 6.1.3.1 [addButton\(\)](#)

```
void ActiveScreen::addButton (
    std::shared_ptr< Button > button_tmp )
```

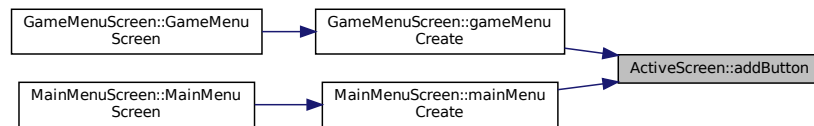
Adds a button to the screen.



## Parameters

<i>button_tmp</i>	The button to add.
-------------------	--------------------

Here is the caller graph for this function:



## 6.1.3.2 addText()

```
void ActiveScreen::addText (
    std::shared_ptr< sf::Text > text_tmp )
```

Adds a text object to the screen.

## Parameters

<i>text_tmp</i>	The SFML Text object to add.
-----------------	------------------------------

## 6.1.3.3 buttonSetColors()

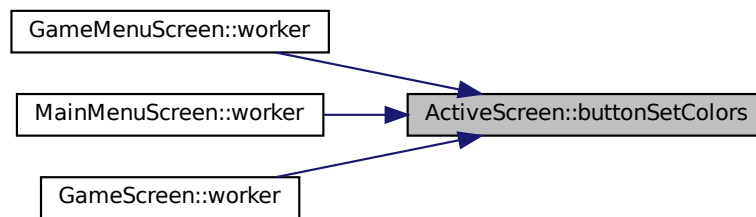
```
void ActiveScreen::buttonSetColors (
    std::shared_ptr< Button > buttonPtr )
```

Sets the colors of a button based on its state.

## Parameters

<i>buttonPtr</i>	The shared pointer to the <a href="#">Button</a> .
------------------	--

Here is the caller graph for this function:



#### 6.1.3.4 draw()

```
virtual void ActiveScreen::draw ( ) [pure virtual]
```

Pure virtual function to draw the screen.

Implemented in [GameScreen](#), [MainMenuScreen](#), and [GameMenuScreen](#).

#### 6.1.3.5 getButtons()

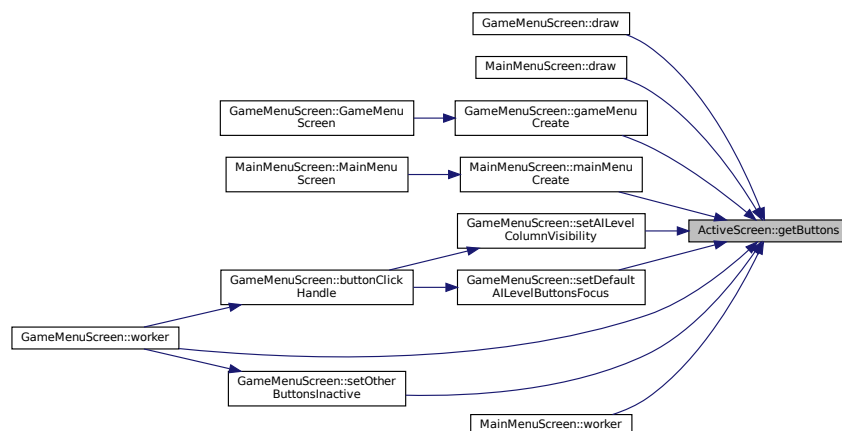
```
std::vector< std::shared_ptr< Button > > & ActiveScreen::getButtons ( )
```

Gets the vector of buttons displayed on the screen.

##### Returns

Vector of shared pointers to [Button](#) objects.

Here is the caller graph for this function:



### 6.1.3.6 getContextWindow()

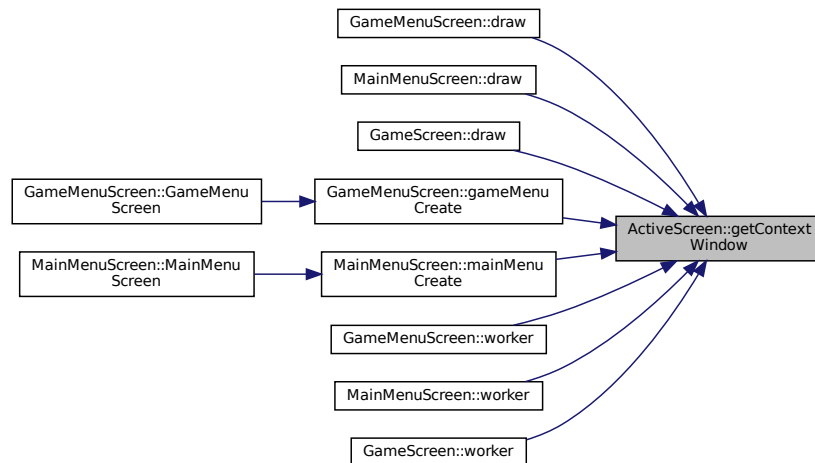
```
ContextWindow * ActiveScreen::getContextWindow ( )
```

Gets the pointer to the [ContextWindow](#).

#### Returns

Pointer to the [ContextWindow](#).

Here is the caller graph for this function:



### 6.1.3.7 getFont()

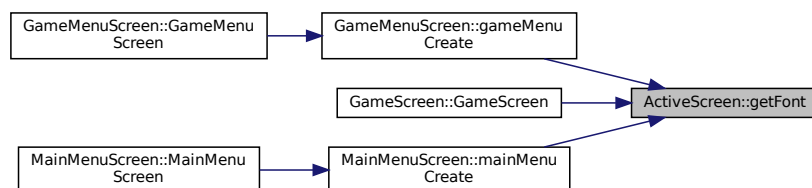
```
sf::Font & ActiveScreen::getFont ( )
```

Gets the SFML font used for text on the screen.

#### Returns

The SFML font.

Here is the caller graph for this function:



#### 6.1.3.8 getPlayersResult()

```
std::vector< std::shared_ptr< Player > > ActiveScreen::getPlayersResult ( ) [virtual]
```

Virtual function to get players' results.

##### Returns

Vector of shared pointers to [Player](#) objects.

Reimplemented in [GameScreen](#).

#### 6.1.3.9 getPlayersSettings()

```
std::vector< std::shared_ptr< playerSettings > > ActiveScreen::getPlayersSettings ( ) const  
[virtual]
```

Virtual function to get players' settings.

##### Returns

Vector of shared pointers to [playerSettings](#) objects.

Reimplemented in [GameMenuScreen](#).

#### 6.1.3.10 getScreenType()

```
ActiveScreenType ActiveScreen::getScreenType ( )
```

Gets the type of the active screen.

##### Returns

The [ActiveScreenType](#).

### 6.1.3.11 getTexts()

```
std::vector< std::shared_ptr< sf::Text > > & ActiveScreen::getTexts ( )
```

Gets the vector of SFML Text objects displayed on the screen.

#### Returns

Vector of shared pointers to SFML Text objects.

Here is the caller graph for this function:



### 6.1.3.12 setContextWindow()

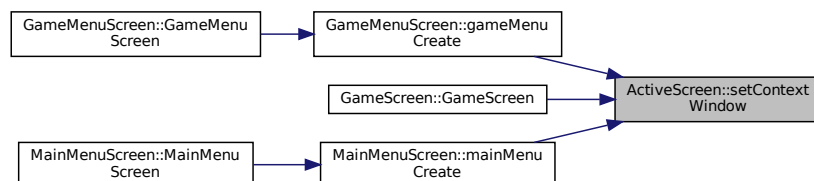
```
void ActiveScreen::setContextWindow (
    ContextWindow * cw )
```

Sets the pointer to the [ContextWindow](#).

#### Parameters

<code>context_window</code>	Pointer to the <a href="#">ContextWindow</a> to set.
-----------------------------	--

Here is the caller graph for this function:



### 6.1.3.13 setFont()

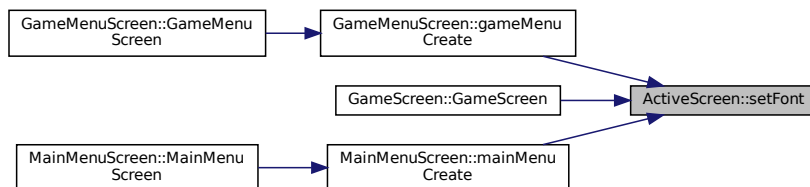
```
void ActiveScreen::setFont (
    sf::Font font )
```

Sets the SFML font used for text on the screen.

#### Parameters

<i>font</i>	The SFML font to set.
-------------	-----------------------

Here is the caller graph for this function:



### 6.1.3.14 setScreenType()

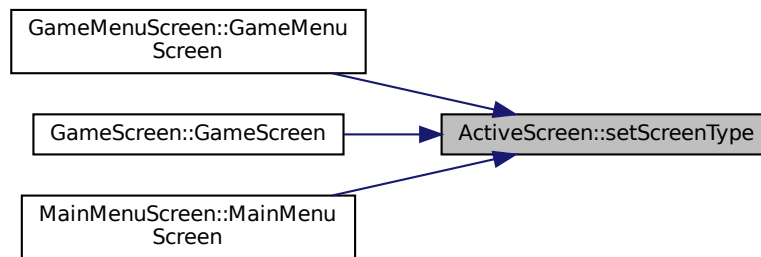
```
void ActiveScreen::setScreenType (
    ActiveScreenType type )
```

Sets the type of the active screen.

#### Parameters

<i>type</i>	The ActiveScreenType to set.
-------------	------------------------------

Here is the caller graph for this function:



### 6.1.3.15 worker()

```
virtual ScreenEventType ActiveScreen::worker ( ) [pure virtual]
```

Pure virtual function representing the worker function for the screen.

#### Returns

The `ScreenEventType` associated with the user interaction.

Implemented in [GameScreen](#), [MainMenuScreen](#), and [GameMenuScreen](#).

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

- [/home/kamil/zpr/Monopoly/ActiveScreen.h](#)
- [/home/kamil/zpr/Monopoly/ActiveScreen.cc](#)

## 6.2 AiAdapter Class Reference

Represents an adapter for AI input data.

```
#include <AiAdapter.h>
```

## Public Member Functions

- [AiAdapter](#) ()  
*Constructor for the [AiAdapter](#) class.*
- `std::vector< double >` [getInputs](#) ()  
*Gets the AI network inputs.*
- `double` [convertMoney](#) (unsigned int money)  
*Converts money into a format suitable for AI input.*
- `double` [convertMoneyValue](#) (double value)  
*Converts money value into a format suitable for AI input.*
- `double` [convertHouseValue](#) (double value)  
*Converts house value into a format suitable for AI input.*
- `double` [convertPosition](#) (unsigned int position)  
*Converts position into a format suitable for AI input.*
- `double` [convertCard](#) (unsigned int cards)  
*Converts card value into a format suitable for AI input.*
- `double` [convertHouse](#) (bool isHotel, unsigned int houseNumber)  
*Converts house state into a format suitable for AI input.*
- `void` [setTurn](#) (unsigned int index)  
*Sets the turn for a given index.*
- `void` [setSelection](#) (unsigned int index)  
*Sets the selection for a given index.*
- `void` [setSelectionState](#) (unsigned int index, int state)  
*Sets the selection state for a given index.*
- `void` [setMoneyContext](#) (int state)  
*Sets the money context state.*
- `void` [clearSelectionState](#) ()  
*Clears the selection state.*
- `void` [setPosition](#) (unsigned int index, unsigned int position)  
*Sets the position for a given index.*
- `void` [setMoney](#) (unsigned int index, unsigned int money)  
*Sets the money for a given index.*
- `void` [setCard](#) (unsigned int index, unsigned int cards)  
*Sets the card for a given index.*
- `void` [setJail](#) (unsigned int index, unsigned int state)  
*Sets the jail state for a given index.*
- `void` [setOwner](#) (unsigned int property, unsigned int state)  
*Sets the owner state for a given property.*
- `void` [setMortgage](#) (unsigned int property, unsigned int state)  
*Sets the mortgage state for a given property.*
- `void` [setHouse](#) (bool isHotel, unsigned int houseNumber, unsigned int id)  
*Sets the house state for a given house.*

### 6.2.1 Detailed Description

Represents an adapter for AI input data.

### 6.2.2 Constructor & Destructor Documentation



### 6.2.2.1 AiAdapter()

```
AiAdapter::AiAdapter ( )
```

Constructor for the [AiAdapter](#) class.

## 6.2.3 Member Function Documentation

### 6.2.3.1 clearSelectionState()

```
void AiAdapter::clearSelectionState ( )
```

Clears the selection state.

### 6.2.3.2 convertCard()

```
double AiAdapter::convertCard (
    unsigned int cards )
```

Converts card value into a format suitable for AI input.

#### Parameters

<i>cards</i>	The card value to convert.
--------------	----------------------------

#### Returns

The converted card value.

Here is the caller graph for this function:



### 6.2.3.3 convertHouse()

```
double AiAdapter::convertHouse (
    bool isHotel,
    unsigned int houseNumber )
```

Converts house state into a format suitable for AI input.

#### Parameters

<i>isHotel</i>	Flag indicating if it's a hotel.
<i>houseNumber</i>	The house number.

#### Returns

The converted house state.

Here is the caller graph for this function:



### 6.2.3.4 convertHouseValue()

```
double AiAdapter::convertHouseValue (
    double value )
```

Converts house value into a format suitable for AI input.

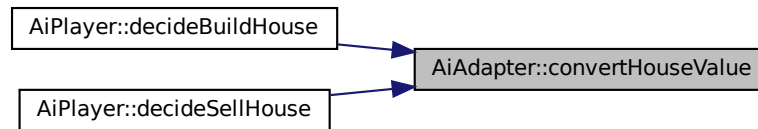
#### Parameters

<i>value</i>	The house value to convert.
--------------	-----------------------------

**Returns**

The converted house value.

Here is the caller graph for this function:

**6.2.3.5 convertMoney()**

```
double AiAdapter::convertMoney (
    unsigned int money )
```

Converts money into a format suitable for AI input.

**Parameters**

<i>money</i>	The money value to convert.
--------------	-----------------------------

**Returns**

The converted money value.

Here is the caller graph for this function:

**6.2.3.6 convertMoneyValue()**

```
double AiAdapter::convertMoneyValue (
    double value )
```

Converts money value into a format suitable for AI input.

**Parameters**

<i>value</i>	The money value to convert.
--------------	-----------------------------

**Returns**

The converted money value.

Here is the caller graph for this function:

**6.2.3.7 convertPosition()**

```
double AiAdapter::convertPosition (
    unsigned int position )
```

Converts position into a format suitable for AI input.

**Parameters**

<i>position</i>	The position to convert.
-----------------	--------------------------

**Returns**

The converted position.

Here is the caller graph for this function:



### 6.2.3.8 getInputs()

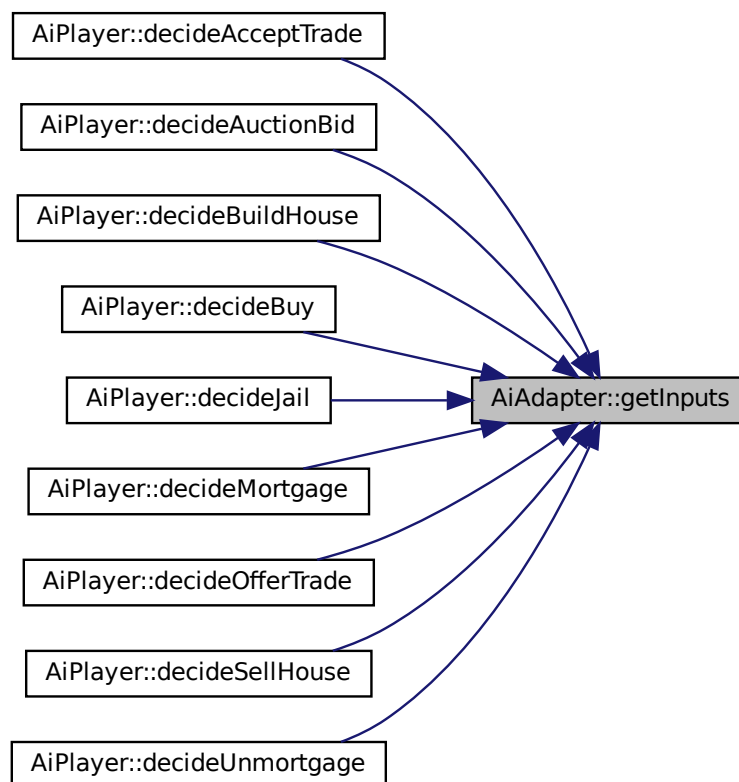
```
std::vector< double > AiAdapter::getInputs ( )
```

Gets the AI network inputs.

#### Returns

Vector of AI network inputs.

Here is the caller graph for this function:



### 6.2.3.9 setCard()

```
void AiAdapter::setCard (
    unsigned int index,
    unsigned int cards )
```

Sets the card for a given index.

**Parameters**

<i>index</i>	The index for which to set the card.
<i>cards</i>	The card to set.

Here is the call graph for this function:

**6.2.3.10 setHouse()**

```
void AiAdapter::setHouse (
    bool isHotel,
    unsigned int houseNumber,
    unsigned int id )
```

Sets the house state for a given house.

**Parameters**

<i>isHotel</i>	Flag indicating if it's a hotel.
<i>houseNumber</i>	The house number.
<i>id</i>	The ID to set.

Here is the call graph for this function:

**6.2.3.11 setJail()**

```
void AiAdapter::setJail (
```

```
    unsigned int index,  
    unsigned int state )
```

Sets the jail state for a given index.

#### Parameters

<i>index</i>	The index for which to set the jail state.
<i>state</i>	The state to set.

#### 6.2.3.12 setMoney()

```
void AiAdapter::setMoney (  
    unsigned int index,  
    unsigned int money )
```

Sets the money for a given index.

#### Parameters

<i>index</i>	The index for which to set the money.
<i>money</i>	The money to set.

Here is the call graph for this function:



#### 6.2.3.13 setMoneyContext()

```
void AiAdapter::setMoneyContext (  
    int state )
```

Sets the money context state.

#### Parameters

<i>state</i>	The state to set.
--------------	-------------------

#### 6.2.3.14 setMortgage()

```
void AiAdapter::setMortgage (
    unsigned int property,
    unsigned int state )
```

Sets the mortgage state for a given property.

##### Parameters

<i>property</i>	The property for which to set the mortgage state.
<i>state</i>	The state to set.

#### 6.2.3.15 setOwner()

```
void AiAdapter::setOwner (
    unsigned int property,
    unsigned int state )
```

Sets the owner state for a given property.

##### Parameters

<i>property</i>	The property for which to set the owner state.
<i>state</i>	The state to set.

#### 6.2.3.16 setPosition()

```
void AiAdapter::setPosition (
    unsigned int index,
    unsigned int position )
```

Sets the position for a given index.

##### Parameters

<i>index</i>	The index for which to set the position.
<i>position</i>	The position to set.



Here is the call graph for this function:



#### 6.2.3.17 setSelection()

```
void AiAdapter::setSelection (
    unsigned int index )
```

Sets the selection for a given index.

##### Parameters

<i>index</i>	The index for which to set the selection.
--------------	---

#### 6.2.3.18 setSelectionState()

```
void AiAdapter::setSelectionState (
    unsigned int index,
    int state )
```

Sets the selection state for a given index.

##### Parameters

<i>index</i>	The index for which to set the selection state.
<i>state</i>	The state to set.

#### 6.2.3.19 setTurn()

```
void AiAdapter::setTurn (
    unsigned int index )
```

Sets the turn for a given index.

#### Parameters

<i>index</i>	The index for which to set the turn.
--------------	--------------------------------------

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

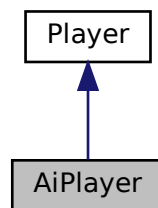
- [/home/kamil/zpr/Monopoly/AiAdapter.h](#)
- [/home/kamil/zpr/Monopoly/AiAdapter.cc](#)

## 6.3 AiPlayer Class Reference

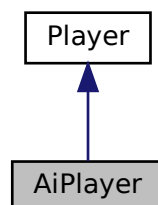
Represents an AI player in a Monopoly game, inheriting from [Player](#).

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

Inheritance diagram for AiPlayer:



Collaboration diagram for AiPlayer:



## Public Member Functions

- [AiPlayer](#) ()
- [AiPlayer](#) (unsigned int money)
- [AiPlayer](#) (unsigned int money, [ann::neuralnet](#) nn)
- [AiAdapter](#) & [getAdapter](#) ()
- [ann::neuralnet](#) & [getNeuralNetwork](#) ()
- void [setNeuralNetwork](#) ([ann::neuralnet](#) new\_nn)
- [BuyDecision](#) [decideBuy](#) (unsigned int index) override
- [JailDecision](#) [decideJail](#) () override
- [Decision](#) [decideMortgage](#) (unsigned int index) override
- [Decision](#) [decideUnmortgage](#) (unsigned int index) override
- unsigned int [decideAuctionBid](#) (unsigned int price) override
- unsigned int [decideBuildHouse](#) () override
- unsigned int [decideSellHouse](#) () override
- [Decision](#) [decideOfferTrade](#) () override
- [Decision](#) [decideAcceptTrade](#) () override

### 6.3.1 Detailed Description

Represents an AI player in a Monopoly game, inheriting from [Player](#).

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 AiPlayer() [1/3]

```
AiPlayer::AiPlayer ( )
```

Default constructor for [AiPlayer](#) class. Here is the call graph for this function:



#### 6.3.2.2 AiPlayer() [2/3]

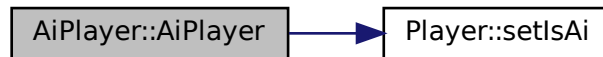
```
AiPlayer::AiPlayer (
    unsigned int money )
```

Constructor for [AiPlayer](#) class with initial money.

## Parameters

<i>money</i>	Initial amount of money for the AI player.
--------------	--

Here is the call graph for this function:



### 6.3.2.3 AiPlayer() [3/3]

```

AiPlayer::AiPlayer (
    unsigned int money,
    ann::neuralnet nn )
  
```

Constructor for [AiPlayer](#) class with initial money and neural\_network.

## Parameters

<i>money</i>	Initial amount of money for the AI player.
<i>n</i>	Neural network used by a <a href="#">Player</a> .

Here is the call graph for this function:



## 6.3.3 Member Function Documentation

### 6.3.3.1 decideAcceptTrade()

`Decision` AiPlayer::decideAcceptTrade ( ) [override], [virtual]

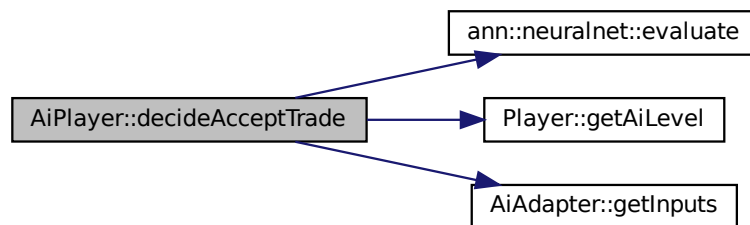
Make a decision for accepting a trade for the AI player (override from base class).

#### Returns

Decision object representing the acceptance of the trade.

Reimplemented from [Player](#).

Here is the call graph for this function:



### 6.3.3.2 decideAuctionBid()

`unsigned int` AiPlayer::decideAuctionBid (   
 `unsigned int price` ) [override], [virtual]

Make a decision for auction bidding for the AI player (override from base class).

#### Parameters

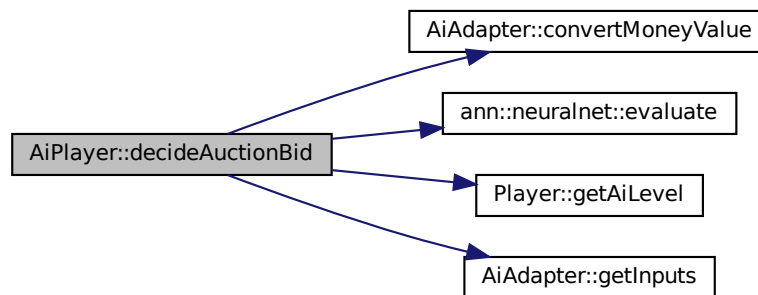
<i>price</i>	Current price in the auction.
--------------	-------------------------------

#### Returns

The bid amount decided by the AI player.

Reimplemented from [Player](#).

Here is the call graph for this function:



#### 6.3.3.3 decideBuildHouse()

```
unsigned int AiPlayer::decideBuildHouse ( ) [override], [virtual]
```

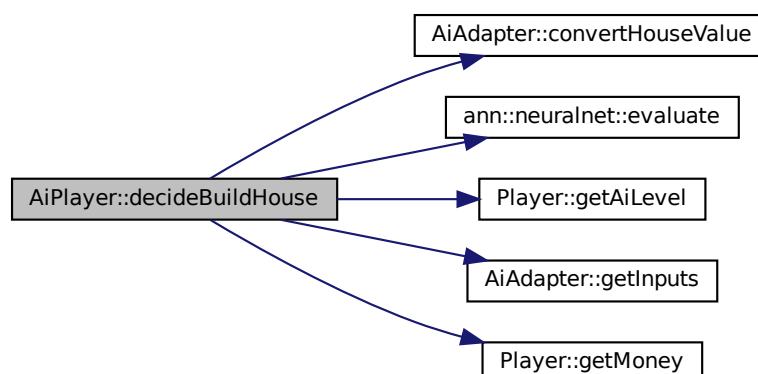
Make a decision for building a house for the AI player (override from base class).

##### Returns

The index of the property on which the AI player decides to build a house.

Reimplemented from [Player](#).

Here is the call graph for this function:



#### 6.3.3.4 decideBuy()

```
BuyDecision AiPlayer::decideBuy (
    unsigned int index ) [override], [virtual]
```

Make a buying decision for the AI player (override from base class).

##### Parameters

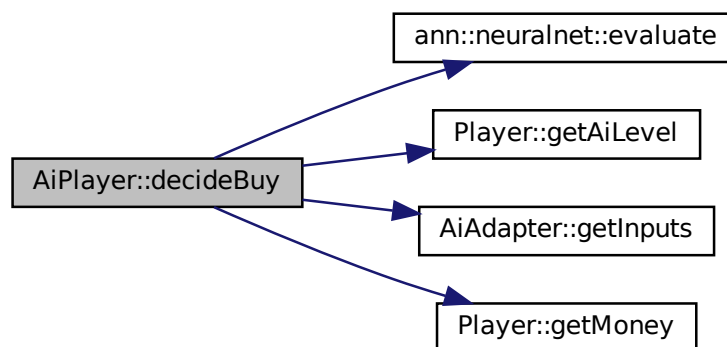
<i>index</i>	Index of the property to consider.
--------------	------------------------------------

##### Returns

BuyDecision object representing the decision.

Reimplemented from [Player](#).

Here is the call graph for this function:



#### 6.3.3.5 decideJail()

```
JailDecision AiPlayer::decideJail ( ) [override], [virtual]
```

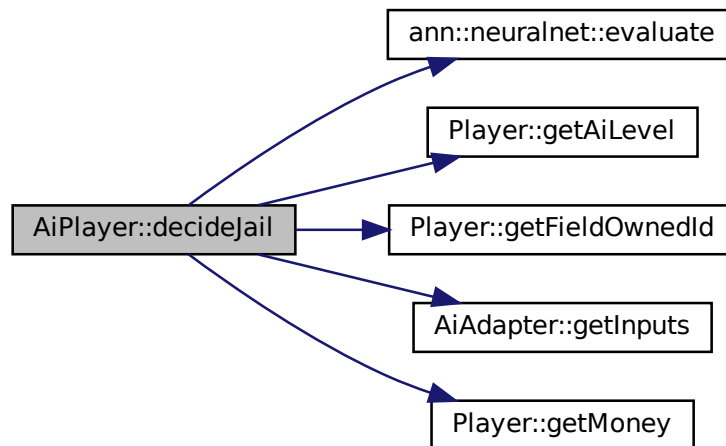
Make a jail decision for the AI player (override from base class).

**Returns**

JailDecision object representing the decision.

Reimplemented from [Player](#).

Here is the call graph for this function:

**6.3.3.6 decideMortgage()**

```

Decision AiPlayer::decideMortgage (
    unsigned int index ) [override], [virtual]
  
```

Make a mortgage decision for the AI player (override from base class).

**Parameters**

<i>index</i>	Index of the property to consider.
--------------	------------------------------------

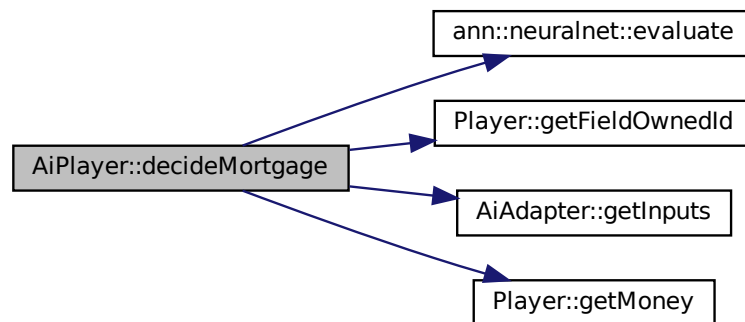
**Returns**

Decision object representing the mortgage decision.

Reimplemented from [Player](#).



Here is the call graph for this function:



#### 6.3.3.7 decideOfferTrade()

`Decision` `AiPlayer::decideOfferTrade ( )` `[override]`, `[virtual]`

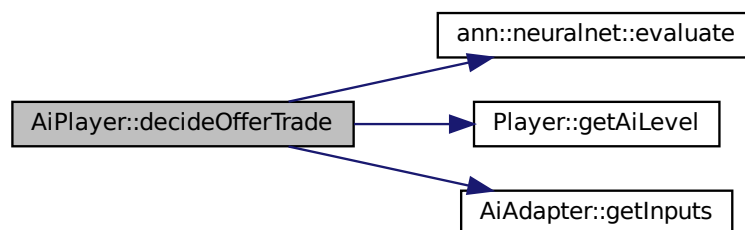
Make a decision for offering a trade for the AI player (override from base class).

##### Returns

Decision object representing the trade offer.

Reimplemented from [Player](#).

Here is the call graph for this function:



### 6.3.3.8 decideSellHouse()

```
unsigned int AiPlayer::decideSellHouse ( ) [override], [virtual]
```

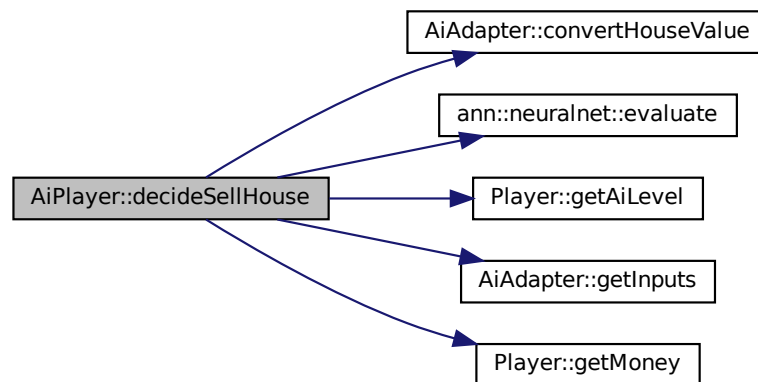
Make a decision for selling a house for the AI player (override from base class).

#### Returns

The index of the property from which the AI player decides to sell a house.

Reimplemented from [Player](#).

Here is the call graph for this function:



### 6.3.3.9 decideUnmortgage()

```
Decision AiPlayer::decideUnmortgage (
    unsigned int index ) [override], [virtual]
```

Make an unmortgage decision for the AI player (override from base class).

#### Parameters

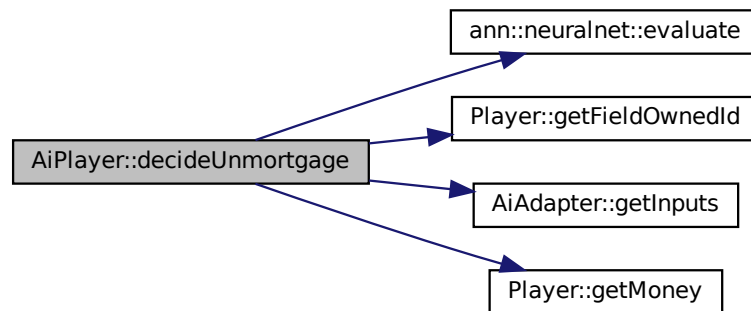
<i>index</i>	Index of the property to consider.
--------------	------------------------------------

#### Returns

Decision object representing the unmortgage decision.

Reimplemented from [Player](#).

Here is the call graph for this function:



#### 6.3.3.10 getAdapter()

```
AiAdapter & AiPlayer::getAdapter ( ) [virtual]
```

Get the AI adapter associated with the AI player.

Reimplemented from [Player](#).

#### 6.3.3.11 getNeuralNetwork()

```
ann::neuralnet & AiPlayer::getNeuralNetwork ( ) [virtual]
```

Get the neural network associated with the AI player.

Reimplemented from [Player](#).

#### 6.3.3.12 setNeuralNetwork()

```
void AiPlayer::setNeuralNetwork (
    ann::neuralnet new_nn )
```

Set the neural network used by the AI player. Here is the caller graph for this function:



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

- [/home/kamil/zpr/Monopoly/Player.h](#)
- [/home/kamil/zpr/Monopoly/Player.cc](#)

## 6.4 Board Class Reference

Class representing the monopoly game board.

```
#include <Board.h>
```

### Public Member Functions

- [Board](#) (const std::string file\_path)  
*Constructor for the [Board](#) class.*
- const std::vector< [PossibleFields](#) > & [getBoard](#) ()  
*Getter for the entire game board.*
- unsigned int [getFieldNumber](#) ()  
*Getter for the number of fields on the board.*
- const sf::Vector2i [getBoardPosition](#) ()  
*Getter for the position of the entire game board.*
- void [clearBoard](#) ()  
*Clears the entire game board.*
- sf::Vector2i [getFieldPositon](#) (unsigned int id, sf::Vector2i prevPos, unsigned int x, unsigned int y)  
*Getter for the position of a specific field on the board.*
- float [getFieldRotation](#) (unsigned int id)  
*Getter for the rotation angle of a specific field on the board.*
- [PossibleFields](#) & [getFieldById](#) (unsigned int wanted\_id)  
*Getter for a specific field on the board by its unique identifier.*

### 6.4.1 Detailed Description

Class representing the monopoly game board.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 Board()

```
Board::Board (
    const std::string file_path )
```

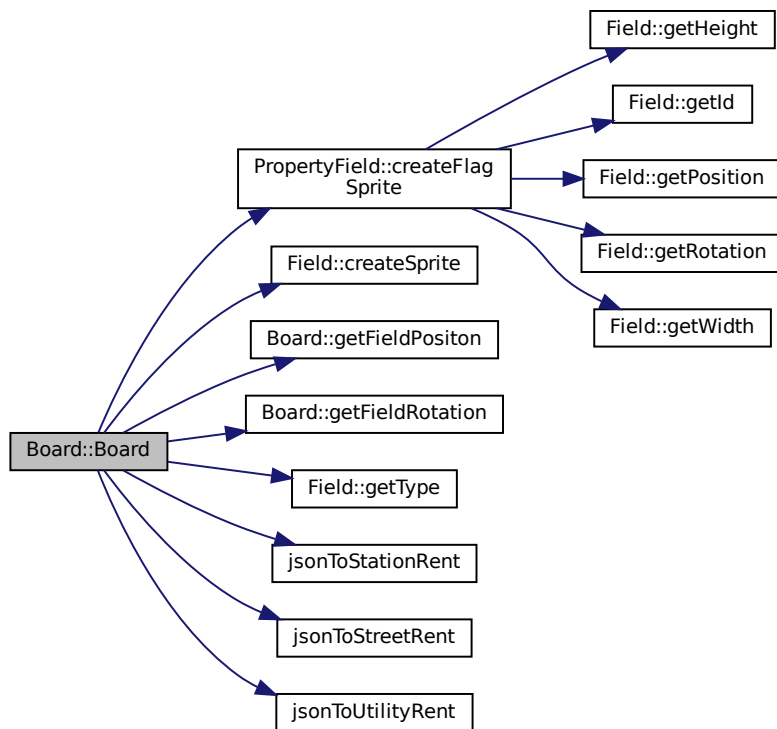
Constructor for the [Board](#) class.

Scale for displaying board stuff

#### Parameters

<i>file_path</i>	The file path to the 'board.json' file.
------------------	---

Here is the call graph for this function:



### 6.4.3 Member Function Documentation

#### 6.4.3.1 clearBoard()

```
void Board::clearBoard ( )
```

Clears the entire game board.

#### 6.4.3.2 getBoard()

```
const std::vector< PossibleFields > & Board::getBoard ( )
```

Getter for the entire game board.

##### Returns

A vector containing all types of fields on the board.

#### 6.4.3.3 getBoardPosition()

```
const sf::Vector2i Board::getBoardPosition ( )
```

Getter for the position of the entire game board.

##### Returns

The position as an sf::Vector2i.

#### 6.4.3.4 getFieldById()

```
PossibleFields & Board::getFieldById (
    unsigned int wanted_id )
```

Getter for a specific field on the board by its unique identifier.

##### Parameters

<i>wanted_id</i>	The unique identifier of the field.
------------------	-------------------------------------

##### Returns

A reference to the field specified by the identifier.

Here is the call graph for this function:



#### 6.4.3.5 getFieldNumber()

```
unsigned int Board::getFieldNumber ( )
```

Getter for the number of fields on the board.

##### Returns

The number of fields.

### 6.4.3.6 getFieldPositon()

```
sf::Vector2i Board::getFieldPositon (
    unsigned int id,
    sf::Vector2i prevPos,
    unsigned int x,
    unsigned int y )
```

Getter for the position of a specific field on the board.

#### Parameters

<i>id</i>	The unique identifier of the field.
<i>prevPos</i>	The previous position of the field.
<i>x</i>	The horizontal position of the field.
<i>y</i>	The vertical position of the field.

#### Returns

The position of the field as an sf::Vector2i.

Here is the caller graph for this function:



### 6.4.3.7 getFieldRotation()

```
float Board::getFieldRotation (
    unsigned int id )
```

Getter for the rotation angle of a specific field on the board.

#### Parameters

<i>id</i>	The unique identifier of the field.
-----------	-------------------------------------

#### Returns

The rotation angle of the field.

Here is the caller graph for this function:



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

- [/home/kamil/zpr/Monopoly/Board.h](#)
- [/home/kamil/zpr/Monopoly/Board.cc](#)

## 6.5 Button Class Reference

Represents a button for handling user interactions.

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

### Public Member Functions

- [Button](#) ([ScreenEventType](#) type, sf::String btnText, sf::Vector2f buttonSize, int charSize)  
*Constructor for the [Button](#) class.*
- void [draw](#) (sf::RenderWindow &window)  
*Draws the button on the specified window.*
- bool [isMouseOver](#) (sf::RenderWindow &window)  
*Checks if the mouse is over the button.*
- virtual void [mouselsOver](#) ()  
*Virtual function called when the mouse is over the button.*
- virtual void [mouselsNotOver](#) ()  
*Virtual function called when the mouse is not over the button.*
- [ScreenEventType](#) [getEventType](#) ()  
*Gets the type of screen event associated with the button.*
- void [setEventType](#) ([ScreenEventType](#) event)  
*Sets the type of screen event associated with the button.*
- void [setFont](#) (sf::Font &fonts)  
*Sets the font for the button text.*
- void [setPosition](#) (sf::Vector2f point)  
*Sets the position of the button.*
- void [setIsClicked](#) (bool state)  
*Sets the state of the button as clicked or not clicked.*
- bool [getIsClicked](#) ()  
*Gets the state of the button (clicked or not clicked).*
- sf::Vector2f [getSize](#) ()  
*Gets the size of the button.*



- void `setVisible` (bool state)  
*Sets the visibility state of the button.*
- bool `isVisible` ()  
*Gets the visibility state of the button.*
- void `setIsActive` (bool state)  
*Sets the active state of the button.*
- bool `isActive` ()  
*Gets the active state of the button.*
- void `setIsFocus` (bool state)  
*Sets the focus state of the button.*
- bool `isFocus` ()  
*Gets the focus state of the button.*
- void `setWasReleased` (bool state)  
*Sets the state of the button as released or not released.*
- bool `getWasReleased` ()  
*Gets the state of the button (released or not released).*
- sf::Text & `getText` ()  
*Gets the SFML Text object associated with the button.*
- void `setActiveBackColor` (sf::Color color)  
*Sets the background color and text color when the button is active.*
- void `setActiveTextColor` (sf::Color color)  
*Sets the text color when the button is active.*
- void `setInactiveBackColor` (sf::Color color)  
*Sets the background color and text color when the button is inactive.*
- void `setInactiveTextColor` (sf::Color color)  
*Sets the text color when the button is inactive.*
- void `setFocusBackColor` (sf::Color color)  
*Sets the background color and text color when the button is in focus.*
- void `setFocusTextColor` (sf::Color color)  
*Sets the text color when the button is in focus.*
- void `setButtonFocus` ()  
*Sets the button state to focused.*
- void `setButtonUnfocus` ()  
*Sets the button state to unfocused.*
- void `updateColors` ()  
*Updates the colors of the button based on its state.*

### 6.5.1 Detailed Description

Represents a button for handling user interactions.

### 6.5.2 Constructor & Destructor Documentation

### 6.5.2.1 Button()

```
Button::Button (
    ScreenEventType type,
    sf::String btnText,
    sf::Vector2f buttonSize,
    int charSize )
```

Constructor for the [Button](#) class.

## Parameters

<i>type</i>	The type of screen event associated with the button.
<i>btnText</i>	The text displayed on the button.
<i>buttonSize</i>	The size of the button.
<i>charSize</i>	The character size of the text.

## 6.5.3 Member Function Documentation

### 6.5.3.1 draw()

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

Draws the button on the specified window.

## Parameters

<i>window</i>	The SFML window to draw the button on.
---------------	--

### 6.5.3.2 getEventType()

```
ScreenEventType Button::getEventType ( )
```

Gets the type of screen event associated with the button.

## Returns

The screen event type.

### 6.5.3.3 getIsActive()

```
bool Button::getIsActive ( )
```

Gets the active state of the button.

## Returns

True if the button is active, false otherwise.

#### 6.5.3.4 `getIsClicked()`

```
bool Button::getIsClicked ( )
```

Gets the state of the button (clicked or not clicked).

##### Returns

True if the button is clicked, false otherwise.

#### 6.5.3.5 `getIsFocus()`

```
bool Button::getIsFocus ( )
```

Gets the focus state of the button.

##### Returns

True if the button is in focus, false otherwise.

#### 6.5.3.6 `getIsVisible()`

```
bool Button::getIsVisible ( )
```

Gets the visibility state of the button.

##### Returns

True if the button is visible, false otherwise.

#### 6.5.3.7 `getSize()`

```
sf::Vector2f Button::getSize ( )
```

Gets the size of the button.

##### Returns

The size of the button.

### 6.5.3.8 getText()

```
sf::Text & Button::getText ( )
```

Gets the SFML Text object associated with the button.

#### Returns

The SFML Text object.

### 6.5.3.9 getWasReleased()

```
bool Button::getWasReleased ( )
```

Gets the state of the button (released or not released).

#### Returns

True if the button was released, false otherwise.

### 6.5.3.10 isMouseOver()

```
bool Button::isMouseOver (
    sf::RenderWindow & window )
```

Checks if the mouse is over the button.

#### Parameters

<i>window</i>	The SFML window.
---------------	------------------

#### Returns

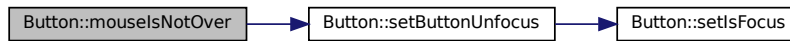
True if the mouse is over the button, false otherwise.

### 6.5.3.11 mouseIsNotOver()

```
void Button::mouseIsNotOver ( ) [virtual]
```

Virtual function called when the mouse is not over the button.

Here is the call graph for this function:

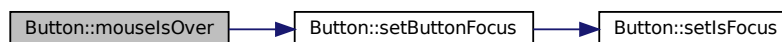


#### 6.5.3.12 `mouseIsOver()`

```
void Button::mouseIsOver ( ) [virtual]
```

Virtual function called when the mouse is over the button.

Here is the call graph for this function:



#### 6.5.3.13 `setActiveBackColor()`

```
void Button::setActiveBackColor (
    sf::Color color )
```

Sets the background color and text color when the button is active.

##### Parameters

<i>color</i>	The background color to set.
--------------	------------------------------

#### 6.5.3.14 `setActiveTextColor()`

```
void Button::setActiveTextColor (
    sf::Color color )
```

Sets the text color when the button is active.

## Parameters

<i>color</i>	The text color to set.
--------------	------------------------

**6.5.3.15 setButtonFocus()**

```
void Button::setButtonFocus ( )
```

Sets the button state to focused.

Here is the call graph for this function:



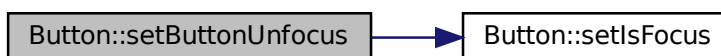
Here is the caller graph for this function:

**6.5.3.16 setButtonUnfocus()**

```
void Button::setButtonUnfocus ( )
```

Sets the button state to unfocused.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.5.3.17 `setEventType()`

```
void Button::setEventType (
    ScreenEventType event )
```

Sets the type of screen event associated with the button.

##### Parameters

<i>event</i>	The screen event type to set.
--------------	-------------------------------

#### 6.5.3.18 `setFocusBackColor()`

```
void Button::setFocusBackColor (
    sf::Color color )
```

Sets the background color and text color when the button is in focus.

##### Parameters

<i>color</i>	The background color to set.
--------------	------------------------------

#### 6.5.3.19 `setFocusTextColor()`

```
void Button::setFocusTextColor (
    sf::Color color )
```

Sets the text color when the button is in focus.

##### Parameters

<i>color</i>	The text color to set.
--------------	------------------------



### 6.5.3.20 setFont()

```
void Button::setFont (
    sf::Font & fonts )
```

Sets the font for the button text.

#### Parameters

<i>fonts</i>	The font to set.
--------------	------------------

### 6.5.3.21 setInactiveBackColor()

```
void Button::setInactiveBackColor (
    sf::Color color )
```

Sets the background color and text color when the button is inactive.

#### Parameters

<i>color</i>	The background color to set.
--------------	------------------------------

### 6.5.3.22 setInactiveTextColor()

```
void Button::setInactiveTextColor (
    sf::Color color )
```

Sets the text color when the button is inactive.

#### Parameters

<i>color</i>	The text color to set.
--------------	------------------------

### 6.5.3.23 setIsActive()

```
void Button::setIsActive (
    bool state )
```

Sets the active state of the button.

## Parameters

<i>state</i>	The active state to set.
--------------	--------------------------

**6.5.3.24 setClicked()**

```
void Button::setClicked (
    bool state )
```

Sets the state of the button as clicked or not clicked.

## Parameters

<i>state</i>	The state to set.
--------------	-------------------

**6.5.3.25 setFocus()**

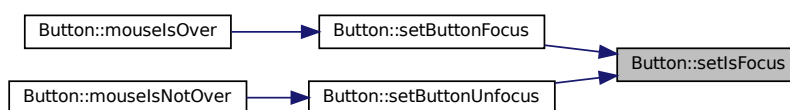
```
void Button::setIsFocus (
    bool state )
```

Sets the focus state of the button.

## Parameters

<i>state</i>	The focus state to set.
--------------	-------------------------

Here is the caller graph for this function:

**6.5.3.26 setIsVisible()**

```
void Button::setIsVisible (
    bool state )
```

Sets the visibility state of the button.

## Parameters

<i>state</i>	The visibility state to set.
--------------	------------------------------

**6.5.3.27 setPosition()**

```
void Button::setPosition (
    sf::Vector2f point )
```

Sets the position of the button.

## Parameters

<i>point</i>	The position to set.
--------------	----------------------

**6.5.3.28 setWasReleased()**

```
void Button::setWasReleased (
    bool state )
```

Sets the state of the button as released or not released.

## Parameters

<i>state</i>	The state to set.
--------------	-------------------

**6.5.3.29 updateColors()**

```
void Button::updateColors ( )
```

Updates the colors of the button based on its state.

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

- /home/kamil/zpr/Monopoly/[Button.h](#)
- /home/kamil/zpr/Monopoly/[Button.cc](#)

**6.6 ChanceCard Class Reference**

Represents a chance card in the monopoly game.

```
#include <Chance.h>
```

## Public Member Functions

- **ChanceCard** (const unsigned int id, const **ChanceType** type, unsigned int value, const sf::String text, const std::string graphic\_path, const unsigned int width, const unsigned int height, const sf::Vector2f position)  
*Constructor for the **ChanceCard** class.*
- unsigned int **getId** ()  
*Getter for the unique identifier of the chance card.*
- **ChanceType** **getType** ()  
*Getter for the type of the chance card.*
- int **getValue** ()  
*Getter for the value associated with the chance card.*
- const sf::String **getText** ()  
*Getter for the text describing the action of the chance card.*
- unsigned int **getWidth** ()  
*Getter for the width of the graphic associated with the chance card.*
- unsigned int **getHeight** ()  
*Getter for the height of the graphic associated with the chance card.*
- const std::string **getGraphicPath** ()  
*Getter for the path to the graphic associated with the chance card.*
- float **getRotation** ()  
*Getter for the rotation of the chance card graphic.*
- const sf::Sprite & **getSprite** ()  
*Getter for the sprite associated with the chance card graphic.*
- const sf::Texture & **getTexture** ()  
*Getter for the texture associated with the chance card graphic.*
- const sf::Vector2f & **getPosition** ()  
*Getter for the position of the chance card on the screen.*
- void **setRotation** (float new\_roation)  
*Setter for the rotation of the chance card graphic.*
- void **setPosition** (sf::Vector2f pos)  
*Setter for the position of the chance card on the screen.*
- void **createSprite** ()  
*Creates the sprite for the chance card using its graphic.*

### 6.6.1 Detailed Description

Represents a chance card in the monopoly game.

### 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 ChanceCard()

```
ChanceCard::ChanceCard (
    const unsigned int id,
    const ChanceType type,
    unsigned int value,
    const sf::String text,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,
    const sf::Vector2f position )
```

Constructor for the [ChanceCard](#) class.

**Parameters**

<i>id</i>	Unique identifier for the chance card.
<i>type</i>	Type of the chance card.
<i>value</i>	Value associated with the chance card.
<i>text</i>	Text describing the action of the chance card.
<i>graphic_path</i>	Path to the graphic associated with the chance card.
<i>width</i>	Width of the graphic.
<i>height</i>	Height of the graphic.
<i>position</i>	Position of the chance card on the screen.

## 6.6.3 Member Function Documentation

### 6.6.3.1 createSprite()

```
void ChanceCard::createSprite ( )
```

Creates the sprite for the chance card using its graphic.

### 6.6.3.2 getGraphicPath()

```
const std::string ChanceCard::getGraphicPath ( )
```

Getter for the path to the graphic associated with the chance card.

**Returns**

const std::string& Path to the graphic.

### 6.6.3.3 getHeight()

```
unsigned int ChanceCard::getHeight ( )
```

Getter for the height of the graphic associated with the chance card.

**Returns**

unsigned int Height of the graphic.

#### 6.6.3.4 getId()

```
unsigned int ChanceCard::getId ( )
```

Getter for the unique identifier of the chance card.

##### Returns

unsigned int Unique identifier for the chance card.

#### 6.6.3.5 getPosition()

```
const sf::Vector2f & ChanceCard::getPosition ( )
```

Getter for the position of the chance card on the screen.

##### Returns

const sf::Vector2f& Position of the chance card.

#### 6.6.3.6 getRotation()

```
float ChanceCard::getRotation ( )
```

Getter for the rotation of the chance card graphic.

##### Returns

float Rotation of the chance card graphic.

#### 6.6.3.7 getSprite()

```
const sf::Sprite & ChanceCard::getSprite ( )
```

Getter for the sprite associated with the chance card graphic.

##### Returns

const sf::Sprite& Sprite of the chance card graphic.

### 6.6.3.8 getText()

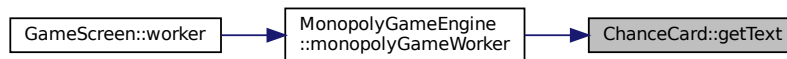
```
const sf::String ChanceCard::getText ( )
```

Getter for the text describing the action of the chance card.

#### Returns

const sf::String& Text describing the action of the chance card.

Here is the caller graph for this function:



### 6.6.3.9 getTexture()

```
const sf::Texture & ChanceCard::getTexture ( )
```

Getter for the texture associated with the chance card graphic.

#### Returns

const sf::Texture& Texture of the chance card graphic.

### 6.6.3.10 getType()

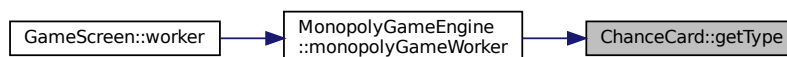
```
ChanceType ChanceCard::getType ( )
```

Getter for the type of the chance card.

#### Returns

ChanceType Type of the chance card.

Here is the caller graph for this function:





### 6.6.3.11 getValue()

```
int ChanceCard::getValue ( )
```

Getter for the value associated with the chance card.

#### Returns

int Value associated with the chance card.

Here is the caller graph for this function:



### 6.6.3.12 getWidth()

```
unsigned int ChanceCard::getWidth ( )
```

Getter for the width of the graphic associated with the chance card.

#### Returns

unsigned int Width of the graphic.

### 6.6.3.13 setPosition()

```
void ChanceCard::setPosition (
    sf::Vector2f pos )
```

Setter for the position of the chance card on the screen.

#### Parameters

<i>pos</i>	New position value.
------------	---------------------

### 6.6.3.14 setRotation()

```
void ChanceCard::setRotation (
    float new_roation )
```

Setter for the rotation of the chance card graphic.

#### Parameters

<i>new_roation</i>	New rotation value.
--------------------	---------------------

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

- /home/kamil/zpr/Monopoly/[Chance.h](#)
- /home/kamil/zpr/Monopoly/[Chance.cc](#)

## 6.7 ContextWindow Class Reference

Represents a Singleton class for handling SFML window operations.

```
#include <ContextWindow.h>
```

### Public Member Functions

- [ContextWindow](#) ([ContextWindow](#) &other)=delete  
*Deleted constructor for proper Singleton class implementation.*
- void [operator=](#) (const [ContextWindow](#) &)=delete  
*Deleted = operator for proper Singleton class implementation.*
- void [display](#) ()  
*Displays the contents of the window.*
- void [clear](#) ()  
*Clears the contents of the window.*
- bool [isOpen](#) ()  
*Checks if the window is open.*
- sf::RenderWindow & [getWindow](#) ()  
*Gets the SFML RenderWindow object.*
- sf::View & [getView](#) ()  
*Gets the SFML View object.*

### Static Public Member Functions

- static [ContextWindow](#) \* [GetInstance](#) ()  
*Gets the pointer to the [ContextWindow](#) instance.*

### Public Attributes

- sf::RenderWindow [window\\_](#)
- sf::View [view\\_](#)

### 6.7.1 Detailed Description

Represents a Singleton class for handling SFML window operations.

The [ContextWindow](#) class is a Singleton class used mainly for handling SFML window operations between other classes.

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 ContextWindow()

```
ContextWindow::ContextWindow (  
    ContextWindow & other ) [delete]
```

Deleted constructor for proper Singleton class implementation.

### 6.7.3 Member Function Documentation

#### 6.7.3.1 clear()

```
void ContextWindow::clear ( )
```

Clears the contents of the window.

Here is the call graph for this function:



### 6.7.3.2 display()

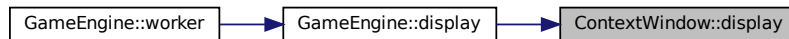
```
void ContextWindow::display ( )
```

Displays the contents of the window.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.7.3.3 GetInstance()

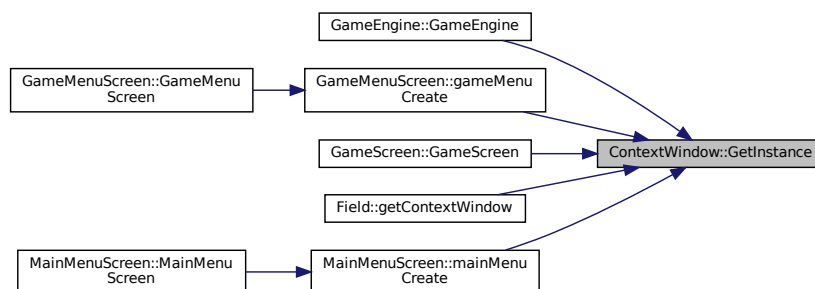
```
ContextWindow * ContextWindow::GetInstance ( ) [static]
```

Gets the pointer to the `ContextWindow` instance.

#### Returns

Pointer to the `ContextWindow` instance.

Here is the caller graph for this function:



#### 6.7.3.4 getView()

```
sf::View & ContextWindow::getView ( )
```

Gets the SFML View object.

##### Returns

Reference to the SFML View object.

Here is the caller graph for this function:



#### 6.7.3.5 getWindow()

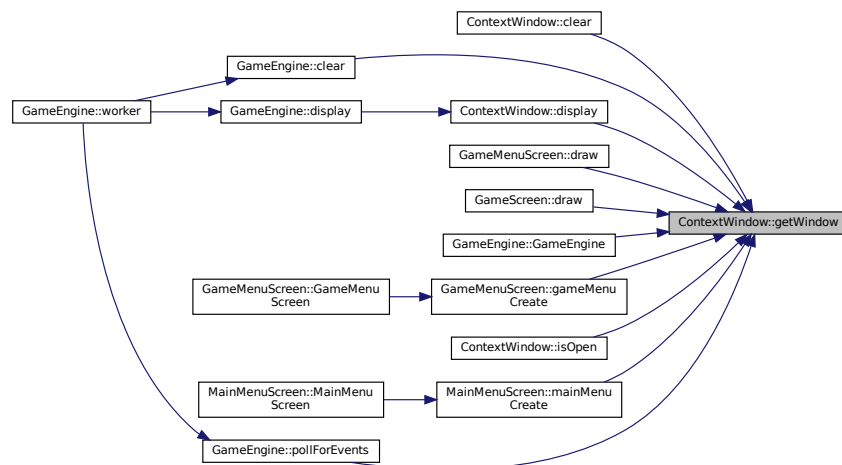
```
sf::RenderWindow & ContextWindow::getWindow ( )
```

Gets the SFML RenderWindow object.

##### Returns

Reference to the SFML RenderWindow object.

Here is the caller graph for this function:



#### 6.7.3.6 isOpen()

```
bool ContextWindow::isOpen ( )
```

Checks if the window is open.

##### Returns

True if the window is open, false otherwise.

Here is the call graph for this function:



#### 6.7.3.7 operator=()

```
void ContextWindow::operator= (
    const ContextWindow & ) [delete]
```

Deleted = operator for proper Singleton class implementation.

### 6.7.4 Member Data Documentation

#### 6.7.4.1 view\_

```
sf::View ContextWindow::view_
```

SFML View object for defining a camera in the 2D scene.

#### 6.7.4.2 window\_

```
sf::RenderWindow ContextWindow::window_
```

SFML RenderWindow object for rendering graphics.

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

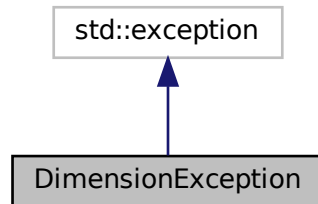
- [/home/kamil/zpr/Monopoly/ContextWindow.h](#)
- [/home/kamil/zpr/Monopoly/ContextWindow.cc](#)

## 6.8 DimensionException Class Reference

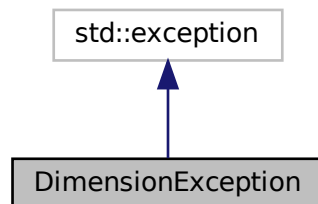
Exception for handling passing wrong dimensions to any displayed object.

```
#include <main.h>
```

Inheritance diagram for DimensionException:



Collaboration diagram for DimensionException:



### Public Member Functions

- [DimensionException](#) (unsigned int dimension)
- [DimensionException](#) (const [DimensionException](#) &e) throw ()
- unsigned int [getBadDimension](#) ()

### 6.8.1 Detailed Description

Exception for handling passing wrong dimensions to any displayed object.

### 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 DimensionException() [1/2]

```
DimensionException::DimensionException (
    unsigned int dimension ) [inline]
```

### 6.8.2.2 DimensionException() [2/2]

```
DimensionException::DimensionException (
    const DimensionException & e ) throw ( ) [inline]
```

## 6.8.3 Member Function Documentation

### 6.8.3.1 getBadDimension()

```
unsigned int DimensionException::getBadDimension ( ) [inline]
```

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

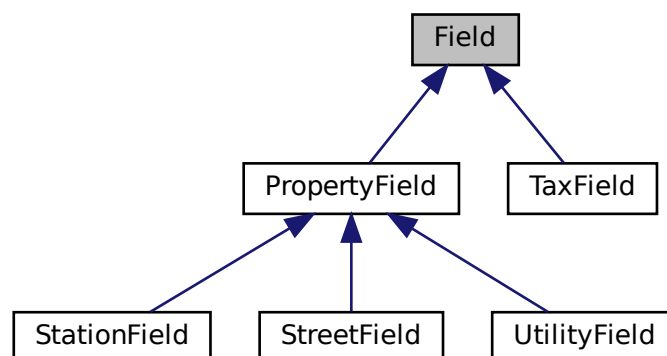
- </home/kamil/zpr/Monopoly/main.h>

## 6.9 Field Class Reference

Base class representing a generic game field.

```
#include <Field.h>
```

Inheritance diagram for Field:





## Public Member Functions

- **Field** (const unsigned int id, const **FieldType** type, const std::string name, const std::string graphic\_path, const unsigned int width, const unsigned int height, const float rotation, const sf::Vector2i position)  
*Constructor for the **Field** class.*
- **ContextWindow** \* **getContextWindow** ()  
*Gets the pointer to the context window.*
- unsigned int **getId** ()  
*Gets the ID of the field.*
- **FieldType** **getType** ()
- const std::string **getName** ()  
*Gets the name of the field.*
- const std::string **getGraphicPath** ()  
*Gets the file path to the field's graphic.*
- unsigned int **getWidth** ()  
*Gets the width of the field.*
- unsigned int **getHeight** ()  
*Gets the height of the field.*
- float **getRotation** ()  
*Gets the rotation angle of the field.*
- const sf::Sprite & **getSprite** ()  
*Gets the sprite representing the field.*
- const sf::Texture & **getTexture** ()  
*Gets the texture of the field.*
- const sf::Vector2i & **getPosition** ()  
*Gets the position of the field on the board.*
- const sf::Text & **getNameText** ()  
*Gets the text representing the name of the field.*
- void **createSprite** ()  
*Creates the sprite for the field.*
- void **setHeight** (unsigned int new\_height)  
*Sets the height of the field.*
- void **setWidth** (unsigned int new\_width)  
*Sets the width of the field.*
- void **setRotation** (float new\_rotation)  
*Sets the rotation angle of the field.*
- void **setPosition** (sf::Vector2i pos)  
*Sets the position of the field on the board.*

### 6.9.1 Detailed Description

Base class representing a generic game field.

### 6.9.2 Constructor & Destructor Documentation

### 6.9.2.1 Field()

```
Field::Field (
    const unsigned int id,
    const FieldType type,
    const std::string name,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,
    const float rotation,
    const sf::Vector2i position )
```

Constructor for the [Field](#) class.

#### Parameters

<i>id</i>	The ID of the field.
<i>type</i>	The type of the field.
<i>name</i>	The name of the field.
<i>graphic_path</i>	The file path to the field's graphic.
<i>width</i>	The width of the field.
<i>height</i>	The height of the field.
<i>rotation</i>	The rotation angle of the field.
<i>position</i>	The position of the field on the board.

## 6.9.3 Member Function Documentation

### 6.9.3.1 createSprite()

```
void Field::createSprite ( )
```

Creates the sprite for the field.

Here is the caller graph for this function:



### 6.9.3.2 getContextWindow()

```
ContextWindow * Field::getContextWindow ( )
```

Gets the pointer to the context window.

#### Returns

A pointer to the context window.

Here is the call graph for this function:



### 6.9.3.3 getGraphicPath()

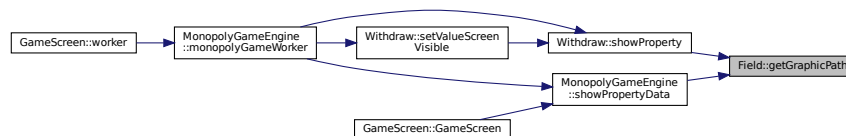
```
const std::string Field::getGraphicPath ( )
```

Gets the file path to the field's graphic.

#### Returns

The file path to the field's graphic.

Here is the caller graph for this function:



### 6.9.3.4 getHeight()

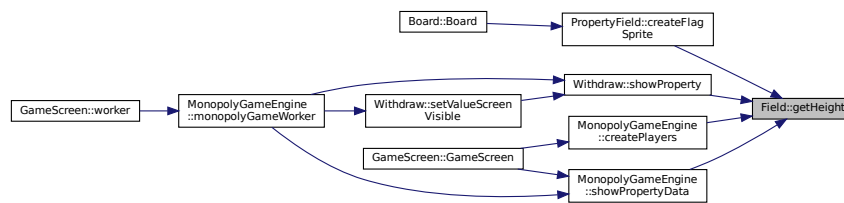
```
unsigned int Field::getHeight ( )
```

Gets the height of the field.

#### Returns

The height of the field.

Here is the caller graph for this function:



### 6.9.3.5 getId()

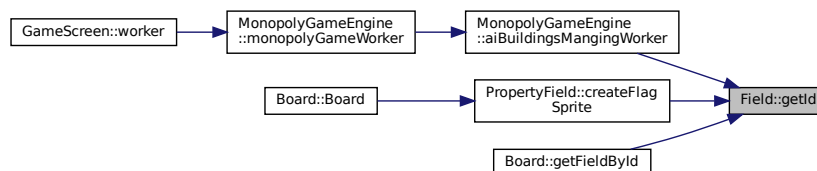
```
unsigned int Field::getId ( )
```

Gets the ID of the field.

#### Returns

The ID of the field.

Here is the caller graph for this function:



### 6.9.3.6 getName()

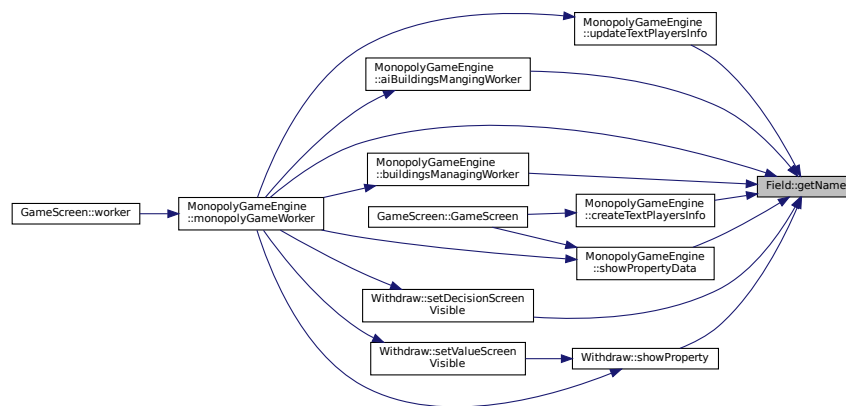
```
const std::string Field::getName ( )
```

Gets the name of the field.

#### Returns

The name of the field.

Here is the caller graph for this function:



### 6.9.3.7 getNameText()

```
const sf::Text & Field::getNameText ( )
```

Gets the text representing the name of the field.

#### Returns

The text representing the name of the field.

Here is the caller graph for this function:



### 6.9.3.8 getPosition()

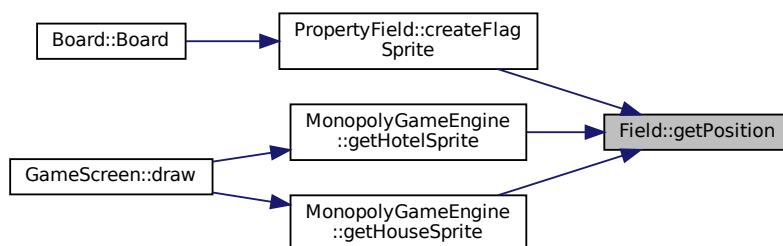
```
const sf::Vector2i & Field::getPosition ( )
```

Gets the position of the field on the board.

#### Returns

The position of the field on the board.

Here is the caller graph for this function:



### 6.9.3.9 getRotation()

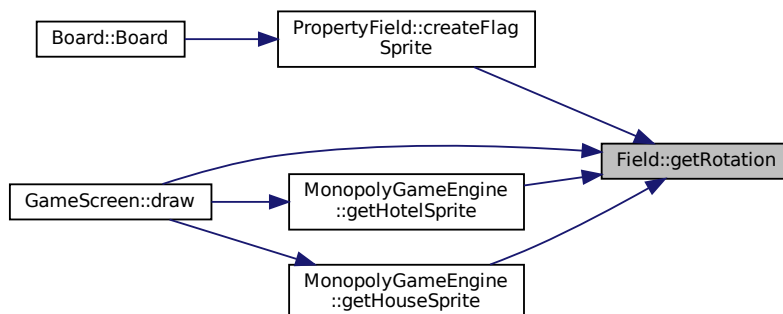
```
float Field::getRotation ( )
```

Gets the rotation angle of the field.

#### Returns

The rotation angle of the field.

Here is the caller graph for this function:



**6.9.3.10 getSprite()**

```
const sf::Sprite & Field::getSprite ( )
```

Gets the sprite representing the field.

**Returns**

The sprite representing the field.

Here is the caller graph for this function:

**6.9.3.11 getTexture()**

```
const sf::Texture & Field::getTexture ( )
```

Gets the texture of the field.

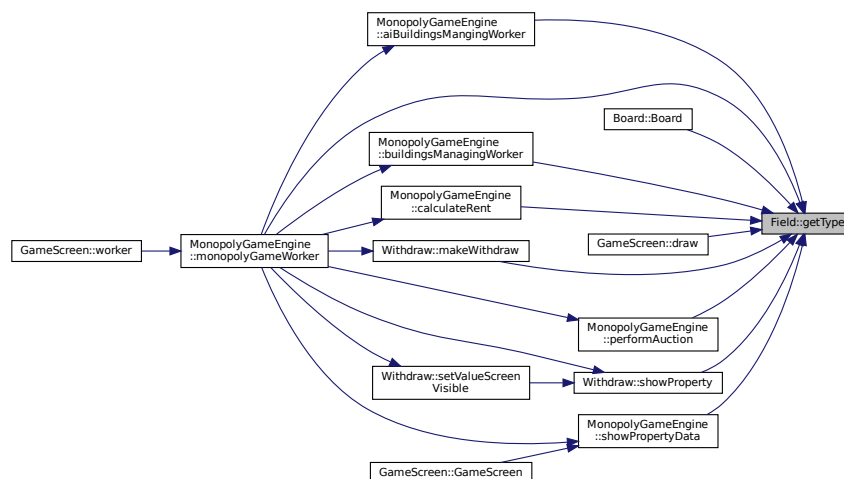
**Returns**

The texture of the field.

**6.9.3.12 getType()**

```
FieldType Field::getType ( )
```

Here is the caller graph for this function:



### 6.9.3.13 getWidth()

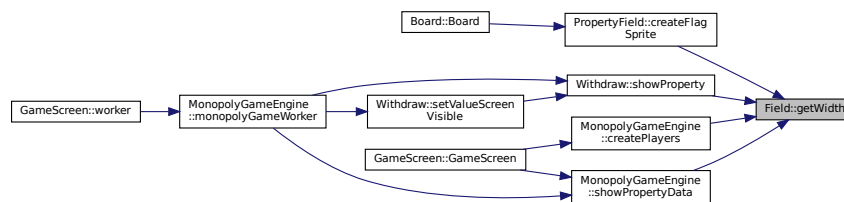
```
unsigned int Field::getWidth ( )
```

Gets the width of the field.

#### Returns

The width of the field.

Here is the caller graph for this function:



### 6.9.3.14 setHeight()

```
void Field::setHeight (
    unsigned int new_height )
```

Sets the height of the field.

#### Parameters

<i>new_height</i>	The new height of the field.
-------------------	------------------------------

### 6.9.3.15 setPosition()

```
void Field::setPosition (
    sf::Vector2i pos )
```

Sets the position of the field on the board.

#### Parameters

<i>pos</i>	The new position of the field.
------------	--------------------------------



### 6.9.3.16 setRotation()

```
void Field::setRotation (
    float new_rotation )
```

Sets the rotation angle of the field.

#### Parameters

<i>new_rotation</i>	The new rotation angle of the field.
---------------------	--------------------------------------

### 6.9.3.17 setWidth()

```
void Field::setWidth (
    unsigned int new_width )
```

Sets the width of the field.

#### Parameters

<i>new_width</i>	The new width of the field.
------------------	-----------------------------

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

- [/home/kamil/zpr/Monopoly/Field.h](#)
- [/home/kamil/zpr/Monopoly/Field.cc](#)

## 6.10 GameEngine Class Reference

Handles low-level program operations, including input interactions and window display.

```
#include <GameEngine.h>
```

### Public Member Functions

- [GameEngine](#) (double frame\_rate\_hz, uint window\_width, uint window\_height)  
*Constructor for the [GameEngine](#) class.*
- [GameEngine](#) (double frame\_rate\_hz)  
*Additional constructor for the [GameEngine](#) class. Used only to train AI players.*
- void [clear](#) ()  
*Clears content on the displayed window.*
- void [display](#) ()

- Displays the content of the context window.*
- void [pollForEvents](#) (sf::Event &event)  
*Polls for events such as mouse and keyboard interactions.*
- std::vector< std::shared\_ptr< [Player](#) > > [worker](#) (std::vector< std::shared\_ptr< [Player](#) > > &players\_vec)  
*Worker function for processing player-related tasks.*
- unsigned int [getwindow\\_width](#) () const  
*Gets the width of the game window.*
- unsigned int [getwindow\\_height](#) () const  
*Gets the height of the game window.*
- [ContextWindow](#) \* [getContextWindow](#) ()  
*Gets the pointer to the context window.*

### 6.10.1 Detailed Description

Handles low-level program operations, including input interactions and window display.

The [GameEngine](#) class manages fundamental program operations, such as handling input interactions (mouse, keyboard) and displaying the window. It incorporates an [ActiveScreen](#) to manage various game screens. Also connects main monopolyGame engine with main loop in main.c to exchange data about results.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 GameEngine() [1/2]

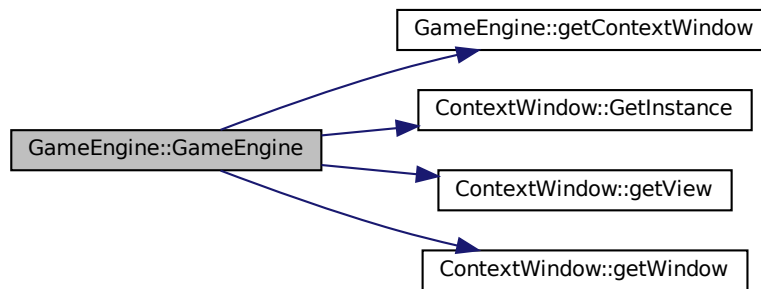
```
GameEngine::GameEngine (
    double frame_rate_hz,
    uint window_width,
    uint window_height )
```

Constructor for the [GameEngine](#) class.

#### Parameters

<i>frame_rate_hz</i>	Desired frame rate in Hertz (frames per second).
<i>window_width</i>	Width of the game window in pixels.
<i>window_height</i>	Height of the game window in pixels.

Here is the call graph for this function:



### 6.10.2.2 GameEngine() [2/2]

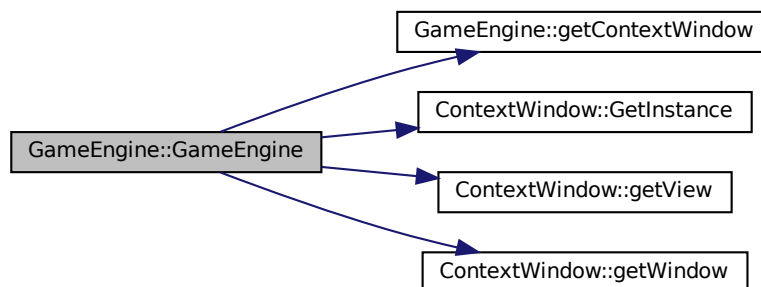
```
GameEngine::GameEngine (
    double frame_rate_hz )
```

Additional constructor for the [GameEngine](#) class. Used only to train AI players.

#### Parameters

<i>frame_rate_hz</i>	Desired frame rate in Hertz (frames per second).
----------------------	--

Here is the call graph for this function:



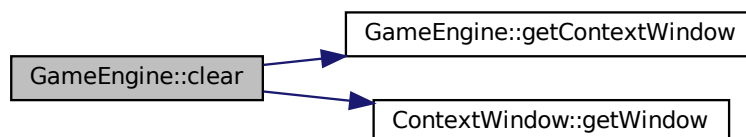
## 6.10.3 Member Function Documentation

### 6.10.3.1 clear()

```
void GameEngine::clear ( )
```

Clears content on the displayed window.

Here is the call graph for this function:



Here is the caller graph for this function:

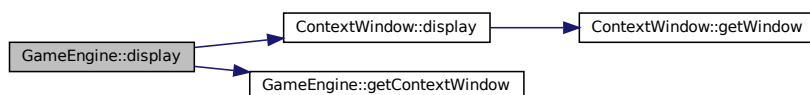


### 6.10.3.2 display()

```
void GameEngine::display ( )
```

Displays the content of the context window.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.10.3.3 getContextWindow()

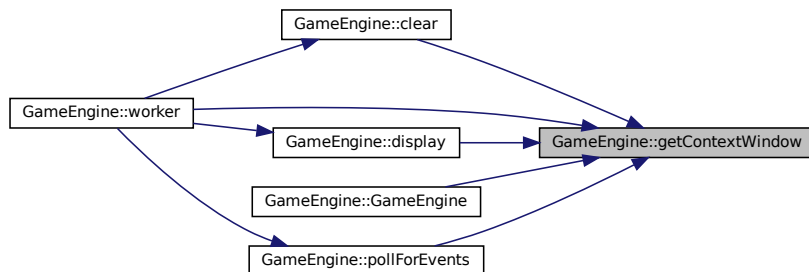
```
ContextWindow * GameEngine::getContextWindow ( )
```

Gets the pointer to the context window.

#### Returns

Pointer to the context window used for rendering graphics.

Here is the caller graph for this function:



### 6.10.3.4 getWindow\_height()

```
unsigned int GameEngine::getWindow_height ( ) const
```

Gets the height of the game window.

#### Returns

Height of the game window in pixels.

### 6.10.3.5 `getwindow_width()`

```
unsigned int GameEngine::getwindow_width ( ) const
```

Gets the width of the game window.

#### Returns

Width of the game window in pixels.

### 6.10.3.6 `pollForEvents()`

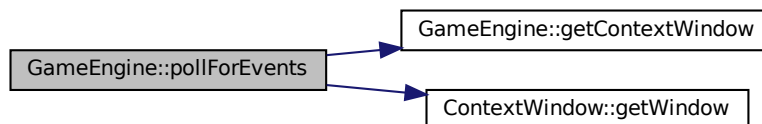
```
void GameEngine::pollForEvents (
    sf::Event & event )
```

Polls for events such as mouse and keyboard interactions.

#### Parameters

<i>event</i>	Reference to an <code>sf::Event</code> object to store the polled event.
--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.10.3.7 worker()

```
std::vector< std::shared_ptr< Player > > GameEngine::worker (
    std::vector< std::shared_ptr< Player >> & players_vec )
```

Worker function for processing player-related tasks.

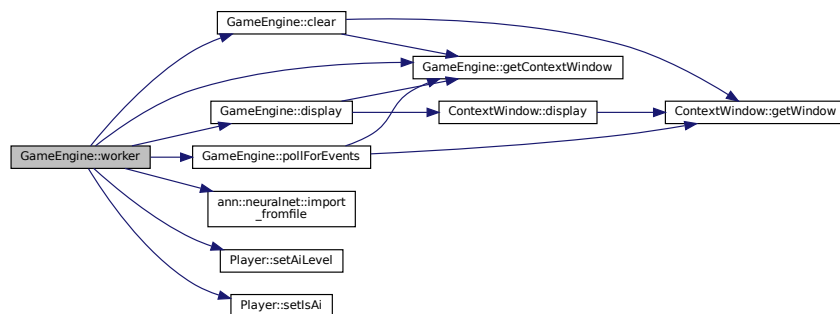
## Parameters

<i>players_vec</i>	Vector of shared pointers to <a href="#">Player</a> objects representing game players.
--------------------	--

## Returns

Vector of shared pointers to [Player](#) objects after processing.

Here is the call graph for this function:



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

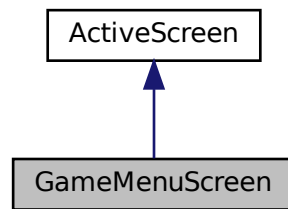
- [/home/kamil/zpr/Monopoly/GameEngine.h](#)
- [/home/kamil/zpr/Monopoly/GameEngine.cc](#)

## 6.11 GameMenuScreen Class Reference

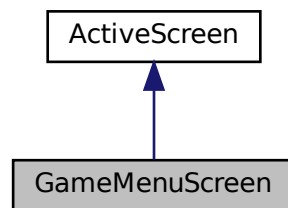
Represents the screen for the game menu.

```
#include <ActiveScreen.h>
```

Inheritance diagram for GameMenuScreen:



Collaboration diagram for GameMenuScreen:



## Public Member Functions

- [GameMenuScreen](#) ()  
*Constructor for the [GameMenuScreen](#) class.*
- void [gameMenuCreate](#) ()  
*Function to create the game menu.*
- [ScreenEventType](#) worker ()  
*Worker function for the game menu screen.*
- void [setPlayerSettings](#) (unsigned int index, bool isNone, bool isHuman, int level)  
*Sets the player settings.*
- void [buttonClickHandle](#) (std::shared\_ptr< [Button](#) > buttonPtr)  
*Handles button clicks in the game menu.*
- void [setOtherButtonsInactive](#) (std::shared\_ptr< [Button](#) > buttonPtr)  
*Sets other buttons inactive when a button is clicked.*
- int [getPlayerNumFromEventType](#) ([ScreenEventType](#) event)  
*Gets the player number from the ScreenEventType.*
- void [setAILevelColumnVisibility](#) (int playerNum, bool visible)  
*Sets the AI level column visibility.*
- bool [isEventTypeAllLevel](#) (int playerNum, [ScreenEventType](#) event)



- Checks if the `ScreenEventType` is related to AI level.
- bool `isEventTypeSetAI` (int playerNum, `ScreenEventType` event)
- Checks if the `ScreenEventType` is related to setting AI.
- void `setDefaultAILevelButtonsFocus` (int playerNum)
- Sets the default AI level buttons' focus.
- std::vector< std::shared\_ptr< `playerSettings` > > `getPlayersSettings` () const
- Gets the players' settings.
- void `draw` ()
- Draws the game menu screen.

### 6.11.1 Detailed Description

Represents the screen for the game menu.

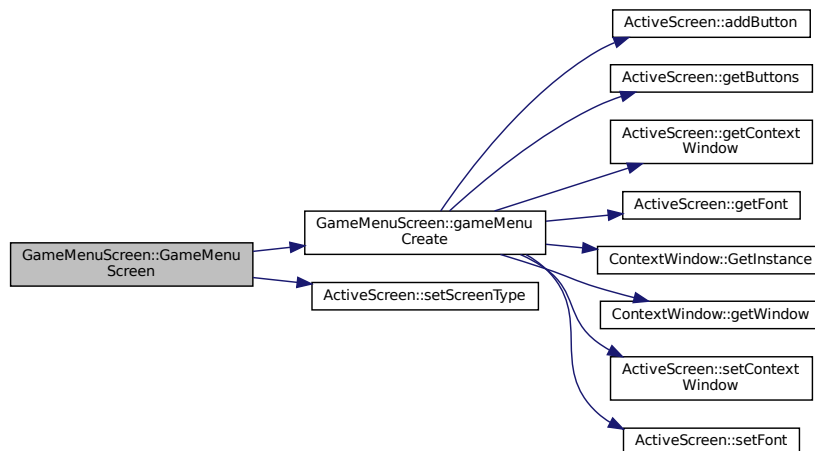
### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 GameMenuScreen()

```
GameMenuScreen::GameMenuScreen ( )
```

Constructor for the `GameMenuScreen` class.

Here is the call graph for this function:



### 6.11.3 Member Function Documentation

#### 6.11.3.1 buttonClickHandle()

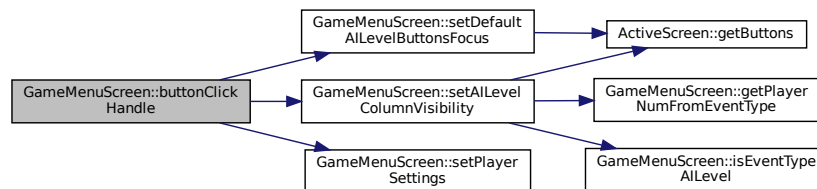
```
void GameMenuScreen::buttonClickHandle (
    std::shared_ptr< Button > buttonPtr )
```

Handles button clicks in the game menu.

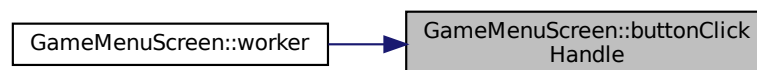
## Parameters

<i>buttonPtr</i>	The shared pointer to the <a href="#">Button</a> clicked.
------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



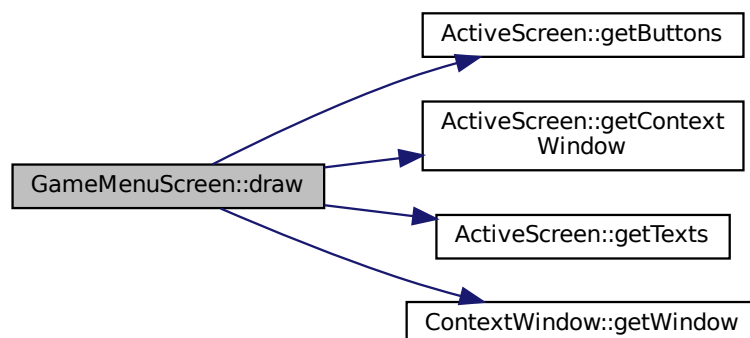
### 6.11.3.2 draw()

```
void GameMenuScreen::draw ( ) [virtual]
```

Draws the game menu screen.

Implements [ActiveScreen](#).

Here is the call graph for this function:

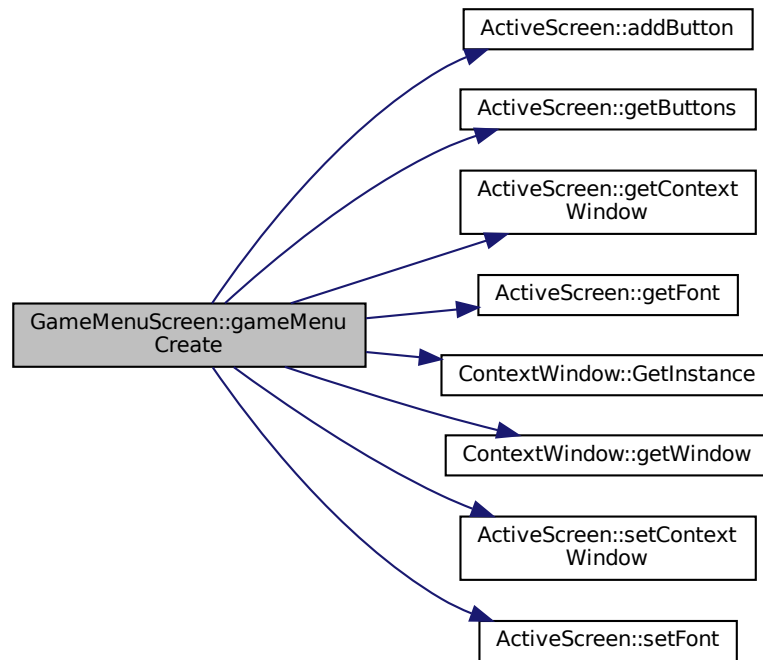


### 6.11.3.3 gameMenuCreate()

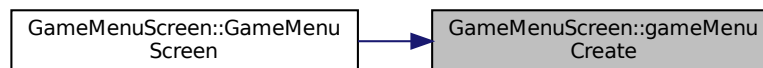
```
void GameMenuScreen::gameMenuCreate ( )
```

Function to create the game menu.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.4 getPlayerNumFromEventType()

```
int GameMenuScreen::getPlayerNumFromEventType (
    ScreenEventType event )
```

Gets the player number from the `ScreenEventType`.

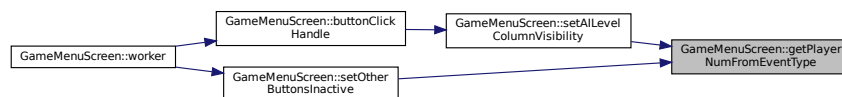
**Parameters**

<i>event</i>	The ScreenEventType to analyze.
--------------	---------------------------------

**Returns**

The player number.

Here is the caller graph for this function:

**6.11.3.5 getPlayerSettings()**

```
std::vector< std::shared_ptr< playerSettings > > GameMenuScreen::getPlayerSettings ( ) const
[virtual]
```

Gets the players' settings.

**Returns**

Vector of shared pointers to [playerSettings](#) objects.

Reimplemented from [ActiveScreen](#).

**6.11.3.6 isEventTypeAllLevel()**

```
bool GameMenuScreen::isEventTypeAllLevel (
    int playerNum,
    ScreenEventType event )
```

Checks if the ScreenEventType is related to AI level.

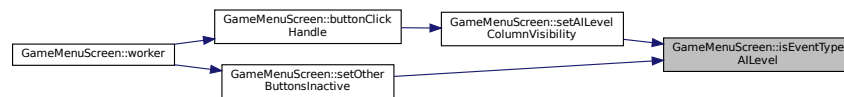
**Parameters**

<i>playerNum</i>	The player number.
<i>event</i>	The ScreenEventType to analyze.

**Returns**

True if the event is related to AI level, false otherwise.

Here is the caller graph for this function:

**6.11.3.7 isEventTypeSetAI()**

```
bool GameMenuScreen::isEventTypeSetAI (
    int playerNum,
    ScreenEventType event )
```

Checks if the ScreenEventType is related to setting AI.

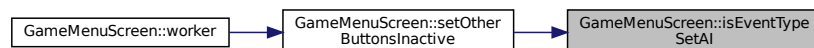
**Parameters**

<i>playerNum</i>	The player number.
<i>event</i>	The ScreenEventType to analyze.

**Returns**

True if the event is related to setting AI, false otherwise.

Here is the caller graph for this function:

**6.11.3.8 setAILevelColumnVisibility()**

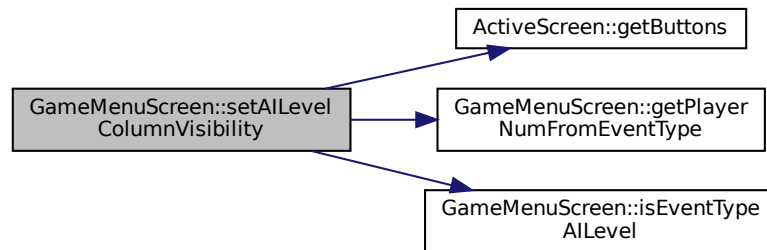
```
void GameMenuScreen::setAILevelColumnVisibility (
    int playerNum,
    bool visible )
```

Sets the AI level column visibility.

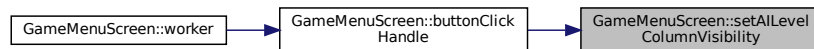
## Parameters

<i>playerNum</i>	The player number.
<i>visible</i>	Flag indicating if the column should be visible.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.9 setDefaultAILevelButtonsFocus()

```
void GameMenuScreen::setDefaultAILevelButtonsFocus (
    int playerNum )
```

Sets the default AI level buttons' focus.

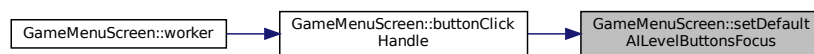
## Parameters

<i>playerNum</i>	The player number.
------------------	--------------------

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.11.3.10 setOtherButtonsInactive()

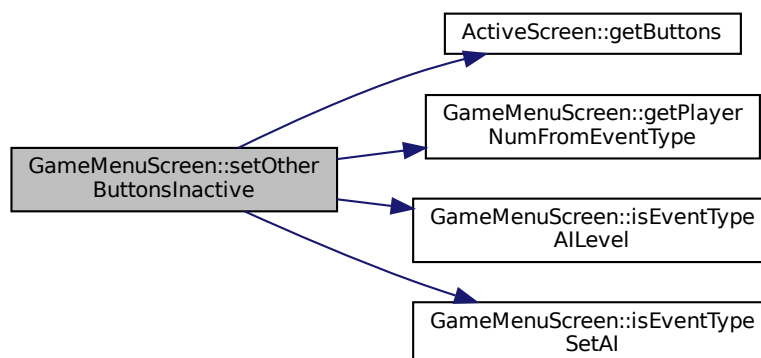
```
void GameMenuScreen::setOtherButtonsInactive (
    std::shared_ptr< Button > buttonPtr )
```

Sets other buttons inactive when a button is clicked.

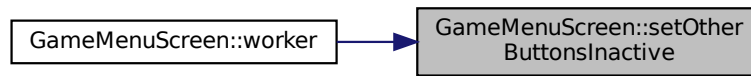
##### Parameters

<i>buttonPtr</i>	The shared pointer to the <a href="#">Button</a> clicked.
------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.11 setPlayerSettings()

```

void GameMenuScreen::setPlayerSettings (
    unsigned int index,
    bool isNone,
    bool isHuman,
    int level )
  
```

Sets the player settings.

#### Parameters

<i>index</i>	The index of the player.
<i>isNone</i>	Flag indicating if the player is set to None.
<i>isHuman</i>	Flag indicating if the player is human.
<i>level</i>	The AI level of the player.

Here is the caller graph for this function:



### 6.11.3.12 worker()

```

ScreenEventType GameMenuScreen::worker ( ) [virtual]
  
```

Worker function for the game menu screen.

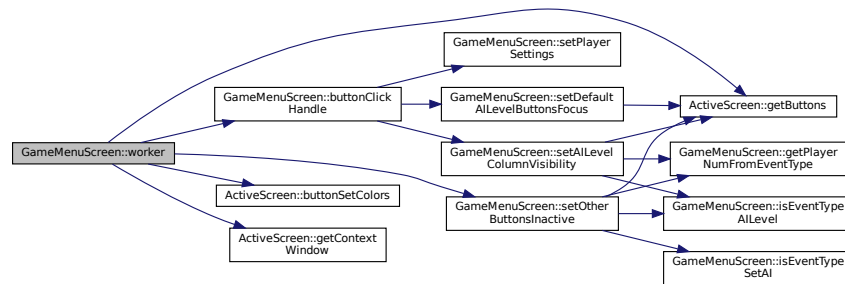


**Returns**

The ScreenEventType associated with the user interaction.

Implements [ActiveScreen](#).

Here is the call graph for this function:



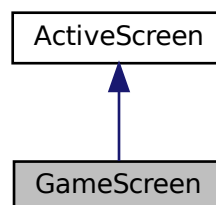
The documentation for this class was generated from the following files:

- [/home/kamil/zpr/Monopoly/ActiveScreen.h](#)
- [/home/kamil/zpr/Monopoly/ActiveScreen.cc](#)

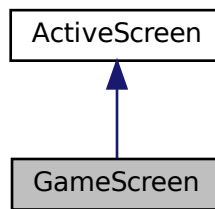
## 6.12 GameScreen Class Reference

```
#include <GameScreen.h>
```

Inheritance diagram for GameScreen:



Collaboration diagram for GameScreen:



## Public Member Functions

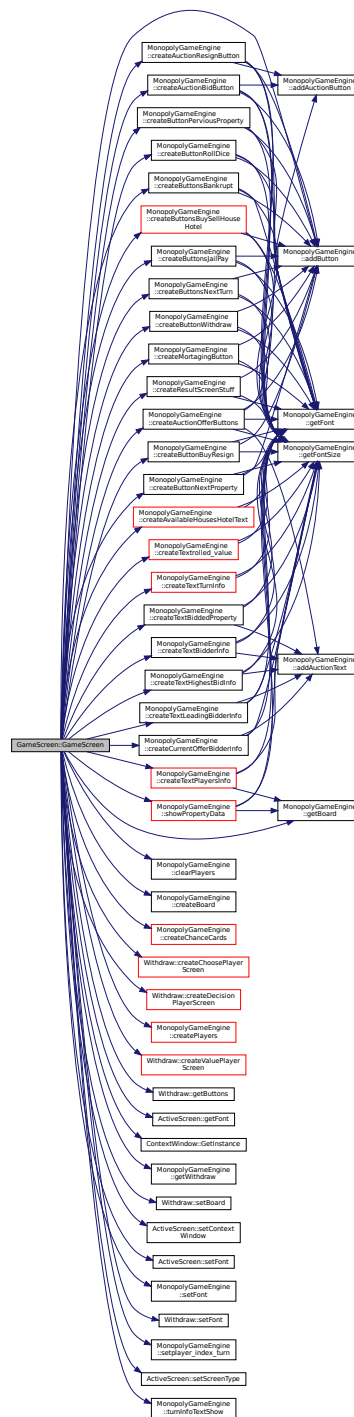
- [GameScreen](#) (std::vector< std::shared\_ptr< [Player](#) >> &players\_)
- [ScreenEventType worker](#) ()  
*Pure virtual function representing the worker function for the screen.*
- void [draw](#) ()  
*Pure virtual function to draw the screen.*
- std::vector< std::shared\_ptr< [Player](#) > > [getPlayersResult](#) ()  
*Virtual function to get players' results.*

### 6.12.1 Constructor & Destructor Documentation

#### 6.12.1.1 GameScreen()

```
GameScreen::GameScreen (  
    std::vector< std::shared_ptr< Player >> & players_ )
```

Here is the call graph for this function:



## 6.12.2 Member Function Documentation

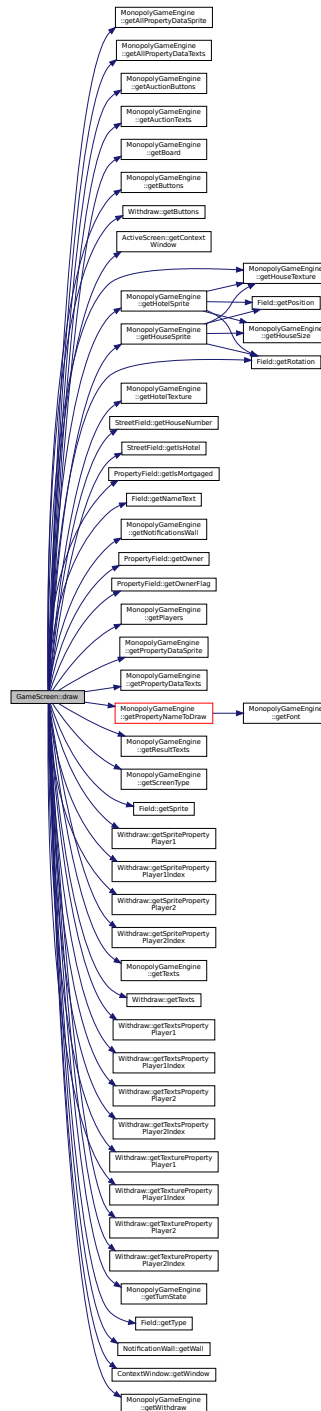
### 6.12.2.1 draw()

```
void GameScreen::draw ( ) [virtual]
```

Pure virtual function to draw the screen.

Implements [ActiveScreen](#).

Here is the call graph for this function:



### 6.12.2.2 getPlayersResult()

```
std::vector< std::shared_ptr< Player > > GameScreen::getPlayersResult ( ) [virtual]
```

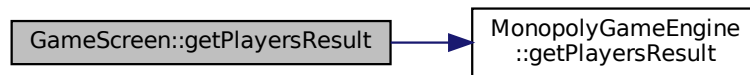
Virtual function to get players' results.

#### Returns

Vector of shared pointers to [Player](#) objects.

Reimplemented from [ActiveScreen](#).

Here is the call graph for this function:



### 6.12.2.3 worker()

```
ScreenEventType GameScreen::worker ( ) [virtual]
```

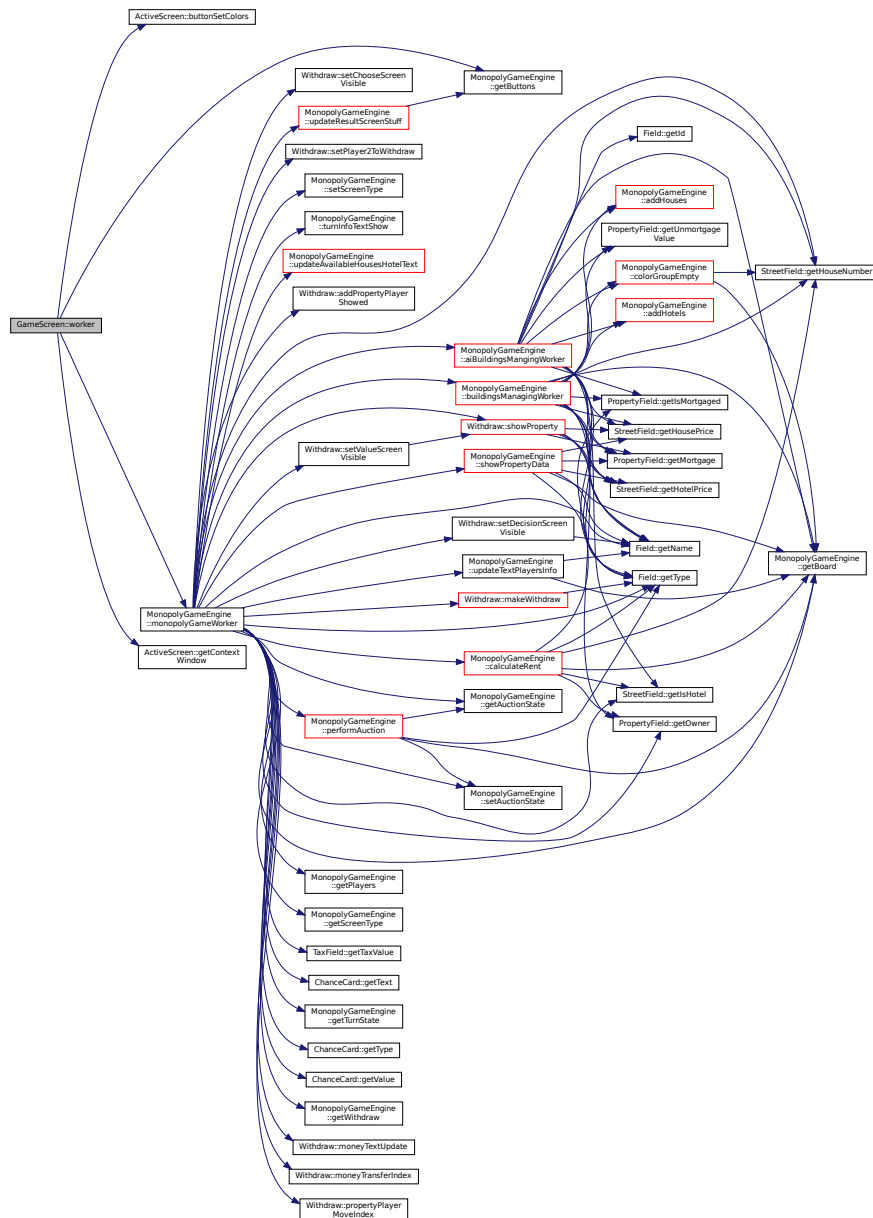
Pure virtual function representing the worker function for the screen.

#### Returns

The `ScreenEventType` associated with the user interaction.

Implements [ActiveScreen](#).

Here is the call graph for this function:



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

- /home/kamil/zpr/Monopoly/GameScreen.h
- /home/kamil/zpr/Monopoly/GameScreen.cc

## 6.13 neat::gene Struct Reference

```
#include <Tinyneat.h>
```

## Public Attributes

- unsigned int [innovation\\_num](#) = -1
- unsigned int [from\\_node](#) = -1
- unsigned int [to\\_node](#) = -1
- double [weight](#) = 0.0
- bool [enabled](#) = true

### 6.13.1 Member Data Documentation

#### 6.13.1.1 enabled

```
bool neat::gene::enabled = true
```

#### 6.13.1.2 from\_node

```
unsigned int neat::gene::from_node = -1
```

#### 6.13.1.3 innovation\_num

```
unsigned int neat::gene::innovation_num = -1
```

#### 6.13.1.4 to\_node

```
unsigned int neat::gene::to_node = -1
```

#### 6.13.1.5 weight

```
double neat::gene::weight = 0.0
```

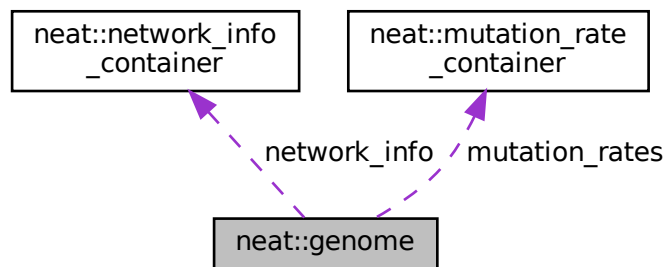
The documentation for this struct was generated from the following file:

- [/home/kamil/zpr/Monopoly/Tinyneat.h](#)

## 6.14 neat::genome Class Reference

```
#include <Tinyneat.h>
```

Collaboration diagram for neat::genome:



### Public Member Functions

- [genome](#) ([network\\_info\\_container](#) &info, [mutation\\_rate\\_container](#) &rates)
- [genome](#) (const [genome](#) &)=default

### Public Attributes

- unsigned int [fitness](#) = 0
- unsigned int [adjusted\\_fitness](#) = 0
- unsigned int [global\\_rank](#) = 0
- unsigned int [max\\_neuron](#)
- unsigned int [can\\_be\\_recurrent](#) = false
- [mutation\\_rate\\_container](#) [mutation\\_rates](#)
- [network\\_info\\_container](#) [network\\_info](#)
- std::map< unsigned int, [gene](#) > [genes](#)

### 6.14.1 Constructor & Destructor Documentation

#### 6.14.1.1 genome() [1/2]

```
neat::genome::genome (
    network\_info\_container & info,
    mutation\_rate\_container & rates )
```



### 6.14.1.2 genome() [2/2]

```
neat::genome::genome (  
    const genome & ) [default]
```

## 6.14.2 Member Data Documentation

### 6.14.2.1 adjusted\_fitness

```
unsigned int neat::genome::adjusted_fitness = 0
```

### 6.14.2.2 can\_be\_recurrent

```
unsigned int neat::genome::can_be_recurrent = false
```

### 6.14.2.3 fitness

```
unsigned int neat::genome::fitness = 0
```

### 6.14.2.4 genes

```
std::map<unsigned int, gene> neat::genome::genes
```

### 6.14.2.5 global\_rank

```
unsigned int neat::genome::global_rank = 0
```

### 6.14.2.6 max\_neuron

```
unsigned int neat::genome::max_neuron
```

### 6.14.2.7 mutation\_rates

`mutation_rate_container` `neat::genome::mutation_rates`

### 6.14.2.8 network\_info

`network_info_container` `neat::genome::network_info`

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

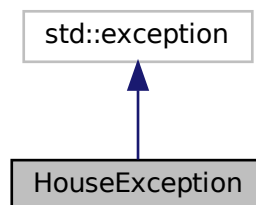
- </home/kamil/zpr/Monopoly/Tinyneat.h>
- </home/kamil/zpr/Monopoly/Tinyneat.cc>

## 6.15 HouseException Class Reference

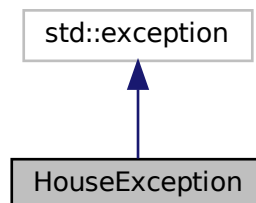
Custom exception class for handling invalid house numbers.

```
#include <Field.h>
```

Inheritance diagram for HouseException:



Collaboration diagram for HouseException:



## Public Member Functions

- [HouseException](#) (unsigned int houses)
- [HouseException](#) (const [HouseException](#) &e) throw ()
- unsigned int [getInvalidNumber](#) ()

### 6.15.1 Detailed Description

Custom exception class for handling invalid house numbers.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 [HouseException\(\)](#) [1/2]

```
HouseException::HouseException (  
    unsigned int houses ) [inline]
```

#### 6.15.2.2 [HouseException\(\)](#) [2/2]

```
HouseException::HouseException (  
    const HouseException & e ) throw ( ) [inline]
```

### 6.15.3 Member Function Documentation

#### 6.15.3.1 [getInvalidNumber\(\)](#)

```
unsigned int HouseException::getInvalidNumber ( )
```

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

- /home/kamil/zpr/Monopoly/[Field.h](#)
- /home/kamil/zpr/Monopoly/[Field.cc](#)

## 6.16 neat::innovation\_container Class Reference

```
#include <Tinyneat.h>
```

## Public Member Functions

- [innovation\\_container](#) ()
- void [reset](#) ()
- unsigned int [add\\_gene](#) ([gene](#) &g)
- unsigned int [number](#) ()

## Friends

- class [pool](#)

## 6.16.1 Constructor & Destructor Documentation

### 6.16.1.1 innovation\_container()

```
neat::innovation_container::innovation_container ( ) [inline]
```

## 6.16.2 Member Function Documentation

### 6.16.2.1 add\_gene()

```
unsigned int neat::innovation_container::add_gene (
    gene & g ) [inline]
```

### 6.16.2.2 number()

```
unsigned int neat::innovation_container::number ( ) [inline]
```

### 6.16.2.3 reset()

```
void neat::innovation_container::reset ( ) [inline]
```

## 6.16.3 Friends And Related Function Documentation

### 6.16.3.1 pool

```
friend class pool [friend]
```

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

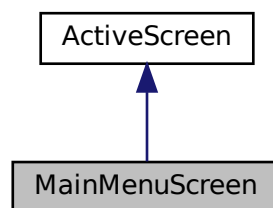
- /home/kamil/zpr/Monopoly/Tinyneat.h

## 6.17 MainMenuScreen Class Reference

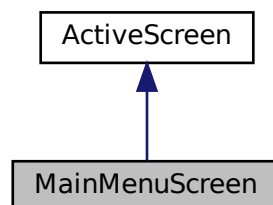
Represents the screen for the main menu.

```
#include <ActiveScreen.h>
```

Inheritance diagram for MainMenuScreen:



Collaboration diagram for MainMenuScreen:



## Public Member Functions

- [MainMenuScreen](#) ()  
Constructor for the [MainMenuScreen](#) class.
- void [mainMenuCreate](#) ()  
Function to create the main menu.
- [ScreenEventType](#) worker ()  
Worker function for the main menu screen.
- void [draw](#) ()  
Draws the main menu screen.

### 6.17.1 Detailed Description

Represents the screen for the main menu.

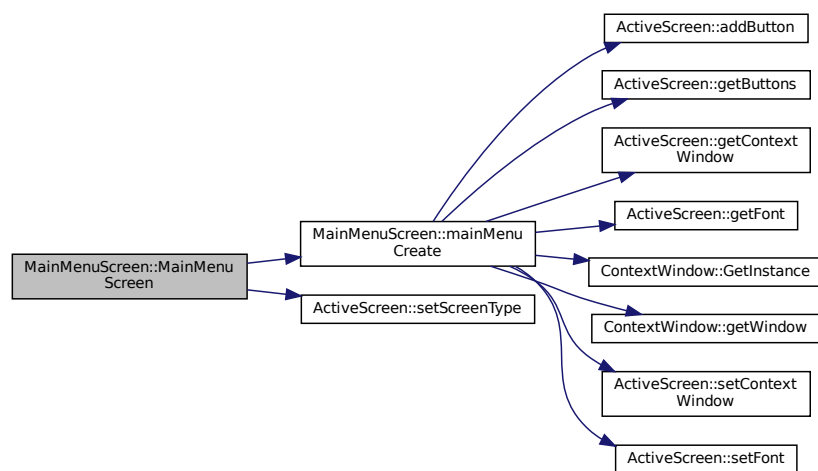
### 6.17.2 Constructor & Destructor Documentation

#### 6.17.2.1 MainMenuScreen()

```
MainMenuScreen::MainMenuScreen ( )
```

Constructor for the [MainMenuScreen](#) class.

Here is the call graph for this function:



### 6.17.3 Member Function Documentation

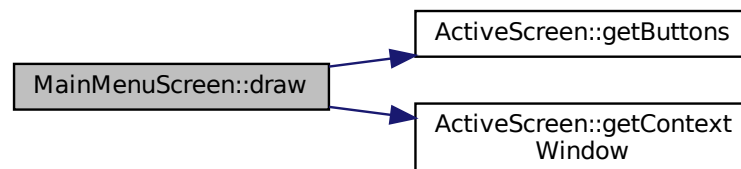
### 6.17.3.1 draw()

```
void MainMenuScreen::draw ( ) [virtual]
```

Draws the main menu screen.

Implements [ActiveScreen](#).

Here is the call graph for this function:

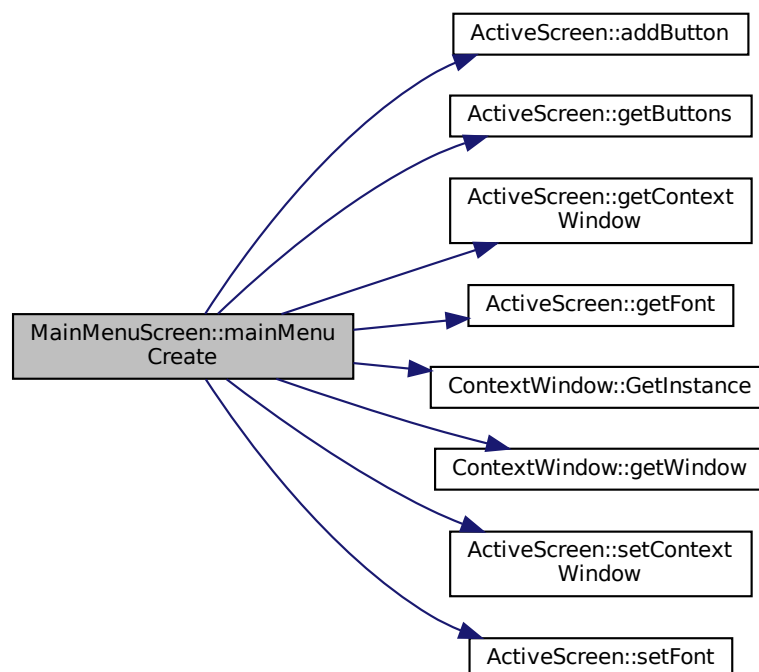


### 6.17.3.2 mainMenuCreate()

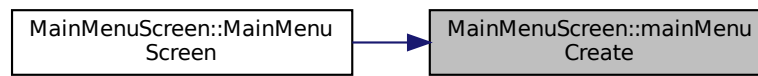
```
void MainMenuScreen::mainMenuCreate ( )
```

Function to create the main menu.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.17.3.3 worker()

`ScreenEventType` MainMenuScreen::worker ( ) [virtual]

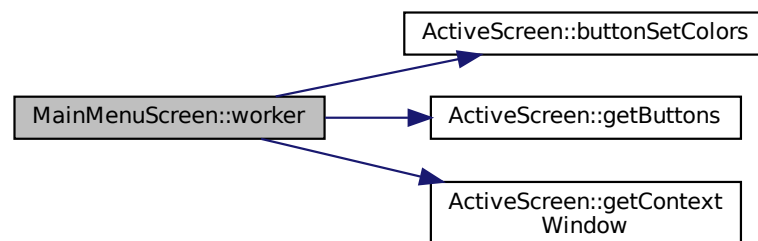
Worker function for the main menu screen.

#### Returns

The `ScreenEventType` associated with the user interaction.

Implements [ActiveScreen](#).

Here is the call graph for this function:



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

- [/home/kamil/zpr/Monopoly/ActiveScreen.h](#)
- [/home/kamil/zpr/Monopoly/ActiveScreen.cc](#)

## 6.18 MonopolyGameEngine Class Reference

Class representing the main game engine for the Monopoly game.

```
#include <MonopolyGameEngine.h>
```



## Public Member Functions

- [MonopolyGameEngine](#) ()  
*Constructor for the `monopolyGameEngine` class.*
- void [setScreenType](#) ([GameScreenType](#) new\_screen\_type)  
*Sets the screen type to the specified type.*
- [GameScreenType](#) [getScreenType](#) () const  
*Gets the current screen type.*
- void [createButtonRollDice](#) ()  
*Creates the button for rolling the dice.*
- void [createTextTurnInfo](#) ()  
*Creates text for displaying turn information.*
- void [createTextrolled\\_value](#) ()  
*Creates text for displaying the rolled dice value.*
- void [createTextPlayersInfo](#) ()  
*Creates text for displaying players' information.*
- void [updateTextPlayersInfo](#) ()  
*Updates the text displaying players' information.*
- void [createTextBiddedProperty](#) ()  
*Creates text for displaying information about the bidded property.*
- void [createTextBidderInfo](#) ()  
*Creates text for displaying information about the bidder.*
- void [createTextHighestBidInfo](#) ()  
*Creates text for displaying the highest bid in an auction.*
- void [createTextLeadingBidderInfo](#) ()  
*Creates text for displaying information about the leading bidder in an auction.*
- void [createCurrentOfferBidderInfo](#) ()  
*Creates text for displaying the current offer in an auction.*
- void [createButtonBuyResign](#) ()  
*Creates a button for buying or resigning from a property.*
- void [createButtonNextProperty](#) ()  
*Creates a button for moving to the next property.*
- void [createButtonPerviousProperty](#) ()  
*Creates a button for moving to the previous property.*
- void [createButtonsBuySellHouseHotel](#) ()  
*Creates buttons for buying, selling houses, and hotels.*
- void [createButtonsBankrupt](#) ()  
*Creates buttons for handling bankruptcy.*
- void [createButtonsNextTurn](#) ()  
*Creates buttons for moving to the next turn.*
- void [createButtonsJailPay](#) ()  
*Creates buttons for handling actions related to the jail (paying to get out).*
- void [createAuctionOfferButtons](#) ()  
*Creates buttons for handling auction offers.*
- void [createAuctionBidButton](#) ()  
*Creates a button for participating in an auction by bidding.*
- void [createAuctionResignButton](#) ()  
*Creates a button for resigning from an auction.*
- void [createButtonWithdraw](#) ()  
*Creates a button for withdrawing from the game.*
- void [createMortagingButton](#) ()

- Creates buttons for mortgaging properties.*

  - void [createResultScreenStuff](#) ()

*Creates various elements for the result screen.*

  - void [updateResultScreenStuff](#) ()

*Updates elements on the result screen.*

  - void [showPropertyData](#) (unsigned int pos, bool is\_property\_shown\_to\_buy)

*Shows property data on the GUI based on the position and whether it is shown to buy.*

  - void [turnInfoTextShow](#) ()

*Displays text related to turn information.*

  - sf::Font & [getFont](#) ()

*Returns the font used in the GUI.*

  - unsigned int [getFontSize](#) () const

*Returns the font size used in the GUI.*

  - void [setFont](#) (sf::Font font)

*Sets the font to be used in the GUI.*

  - void [addButton](#) (std::shared\_ptr< [Button](#) > button\_tmp)

*Adds a button to the GUI.*

  - void [addText](#) (std::shared\_ptr< sf::Text > text\_tmp)

*Adds text to the GUI.*

  - void [addAuctionButton](#) (std::shared\_ptr< [Button](#) > button\_tmp)

*Adds an auction button to the GUI.*

  - void [addAuctionText](#) (std::shared\_ptr< sf::Text > text\_tmp)

*Adds auction text to the GUI.*

  - std::vector< std::shared\_ptr< [Button](#) > > & [getButtons](#) ()

*Returns a vector of pointers to buttons in the GUI.*

  - std::vector< std::shared\_ptr< sf::Text > > & [getTexts](#) ()

*Returns a vector of pointers to text elements in the GUI.*

  - std::vector< std::shared\_ptr< [Button](#) > > & [getAuctionButtons](#) ()

*Returns a vector of pointers to auction buttons in the GUI.*

  - std::vector< std::shared\_ptr< sf::Text > > & [getAuctionTexts](#) ()

*Returns a vector of pointers to auction text elements in the GUI.*

  - std::vector< std::shared\_ptr< sf::Text > > & [getResultTexts](#) ()

*Returns a vector of pointers to text elements used in the result screen.*

  - sf::Sprite & [getPropertyDataSprite](#) ()

*Returns the sprite representing property data in the GUI.*

  - std::vector< std::shared\_ptr< sf::Text > > & [getPropertyDataTexts](#) ()

*Returns a vector of pointers to text elements representing property data in the GUI.*

  - sf::Sprite & [getAllPropertyDataSprite](#) ()

*Returns the sprite representing all property data in the GUI.*

  - std::vector< std::shared\_ptr< sf::Text > > & [getAllPropertyDataTexts](#) ()

*Returns a vector of pointers to text elements representing all property data in the GUI.*

  - [NotificationWall](#) & [getNotificationsWall](#) ()

*Returns the notification wall used in the GUI.*

  - sf::Text [getPropertyNameToDraw](#) (sf::Text text, sf::Sprite &sprite, float rotation)

*Gets the text suitable for drawing a property name.*

  - sf::Texture & [getHouseTexture](#) ()

*Returns the texture of the house used in the GUI.*

  - sf::Texture & [getHotelTexture](#) ()

*Returns the texture of the hotel used in the GUI.*

  - sf::Vector2f & [getHouseSize](#) ()

*Returns the size of a house in the GUI.*

- void `createPlayers` (std::vector< std::shared\_ptr< `Player` >> &players\_from\_game\_engine)  
*Creates player objects based on input vector.*
- void `clearPlayers` ()  
*Clears player objects from the game.*
- void `createBoard` ()  
*Creates the game board.*
- void `clearBoard` ()  
*Clears the game board.*
- std::shared\_ptr< `Board` > `getBoard` ()  
*Returns a pointer to the game board.*
- std::vector< std::shared\_ptr< `Player` > > & `getPlayers` ()  
*Returns a vector of pointers to players in the game.*
- std::vector< std::shared\_ptr< `Player` > > `getPlayersResult` ()  
*Returns a vector of pointers to players based on game result.*
- void `setplayer_index_turn` (unsigned int indx)  
*Sets the index of the player whose turn it is.*
- `TurnState` `getTurnState` () const  
*Gets the current turn state.*
- void `setAuctionState` (`AuctionState` new\_state)  
*Sets the state of the auction.*
- `AuctionState` `getAuctionState` ()  
*Gets the current state of the auction.*
- unsigned int `getHouseCount` ()  
*Gets the count of houses available for purchase.*
- unsigned int `getHotelCount` ()  
*Gets the count of hotels available for purchase.*
- void `setHouseCount` (unsigned int new\_count)  
*Sets the count of houses available for purchase.*
- void `setHotelCount` (unsigned int new\_count)  
*Sets the count of hotels available for purchase.*
- void `addHouses` (unsigned int added\_amount)  
*Adds houses to the available count.*
- void `subtractHouses` (unsigned int subtracted\_amount)  
*Subtracts houses from the available count.*
- void `addHotels` (unsigned int added\_amount)  
*Adds hotels to the available count.*
- void `subtractHotels` (unsigned int subtracted\_amount)  
*Subtracts hotels from the available count.*
- void `performAuction` ()  
*Performs the auction.*
- unsigned int `calculateGroupFieldsOwned` (std::vector< unsigned int > player\_fields, `PropertyField` &field) const  
*Calculates the number of group fields owned by a player.*
- bool `groupCompleted` (std::vector< unsigned int > player\_fields, `PropertyField` &field) const  
*Checks if a group of properties is completed by a player.*
- bool `isBuildingLegal` (std::shared\_ptr< `Player` > builder, `StreetField` field)  
*Checks if building houses on a street is legal for a player.*
- bool `isDestroyingLegal` (std::shared\_ptr< `Player` > builder, `StreetField` field)  
*Checks if destroying houses on a street is legal for a player.*
- bool `isHotelBuildingLegal` (std::shared\_ptr< `Player` > builder, `StreetField` &field)  
*Checks if building a hotel on a street is legal for a player.*

- bool `isHotelDestroyingLegal` (std::shared\_ptr< [Player](#) > builder, [StreetField](#) &field)  
*Checks if destroying a hotel on a street is legal for a player.*
- bool `colorGroupEmpty` (std::shared\_ptr< [Player](#) > mortgaging, [StreetField](#) &field)  
*Checks if a color group is empty (no properties owned) for a player.*
- sf::Sprite `getHouseSprite` ([StreetField](#) &field, unsigned int houses\_number)  
*Gets the sprite for a house on a street.*
- sf::Sprite `getHotelSprite` ([StreetField](#) &field)  
*Gets the sprite for a hotel on a street.*
- void `createAvailableHousesHotelText` ()  
*Creates text displaying the available count of houses and hotels.*
- void `updateAvailableHousesHotelText` ()  
*Updates the text displaying the available count of houses and hotels.*
- unsigned int `calculateRent` (unsigned int rolled\_val, int pos)  
*Calculates the rent to be paid based on the rolled dice value and property position.*
- void `buildingsManagingWorker` ()  
*Worker method for managing building and destroying houses/hotels.*
- void `aiBuildingsManagingWorker` ()  
*AI worker method for managing building and destroying houses/hotels.*
- bool `monopolyGameWorker` ()  
*Main Worker method for whole monopoly game engine.*
- [Withdraw](#) & `getWithdraw` ()  
*Returns a reference to the [Withdraw](#) object.*
- void `createChanceCards` ()  
*Creates the chance cards for the game.*
- void `shuffleChanceCards` ()  
*Shuffles the chance cards.*

### 6.18.1 Detailed Description

Class representing the main game engine for the Monopoly game.

The [MonopolyGameEngine](#) class handles the overall game flow, including player turns, actions, and interactions with the game board. It also manages the graphical user interface (GUI) components.

### 6.18.2 Constructor & Destructor Documentation

#### 6.18.2.1 MonopolyGameEngine()

```
MonopolyGameEngine::MonopolyGameEngine ( )
```

Constructor for the `monopolyGameEngine` class.

### 6.18.3 Member Function Documentation

#### 6.18.3.1 addAuctionButton()

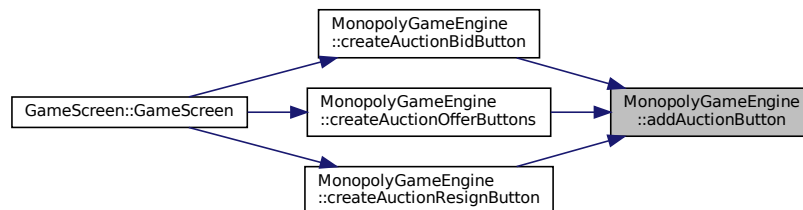
```
void MonopolyGameEngine::addAuctionButton (
    std::shared_ptr< Button > button_tmp )
```

Adds an auction button to the GUI.

## Parameters

<i>button_tmp</i>	Pointer to the auction button to be added.
-------------------	--

Here is the caller graph for this function:



## 6.18.3.2 addAuctionText()

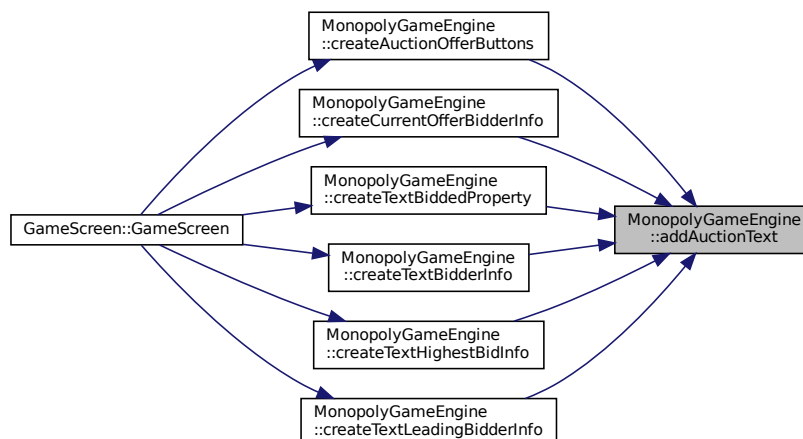
```
void MonopolyGameEngine::addAuctionText (
    std::shared_ptr< sf::Text > text_tmp )
```

Adds auction text to the GUI.

## Parameters

<i>text_tmp</i>	Pointer to the auction text to be added.
-----------------	--

Here is the caller graph for this function:



### 6.18.3.3 addButton()

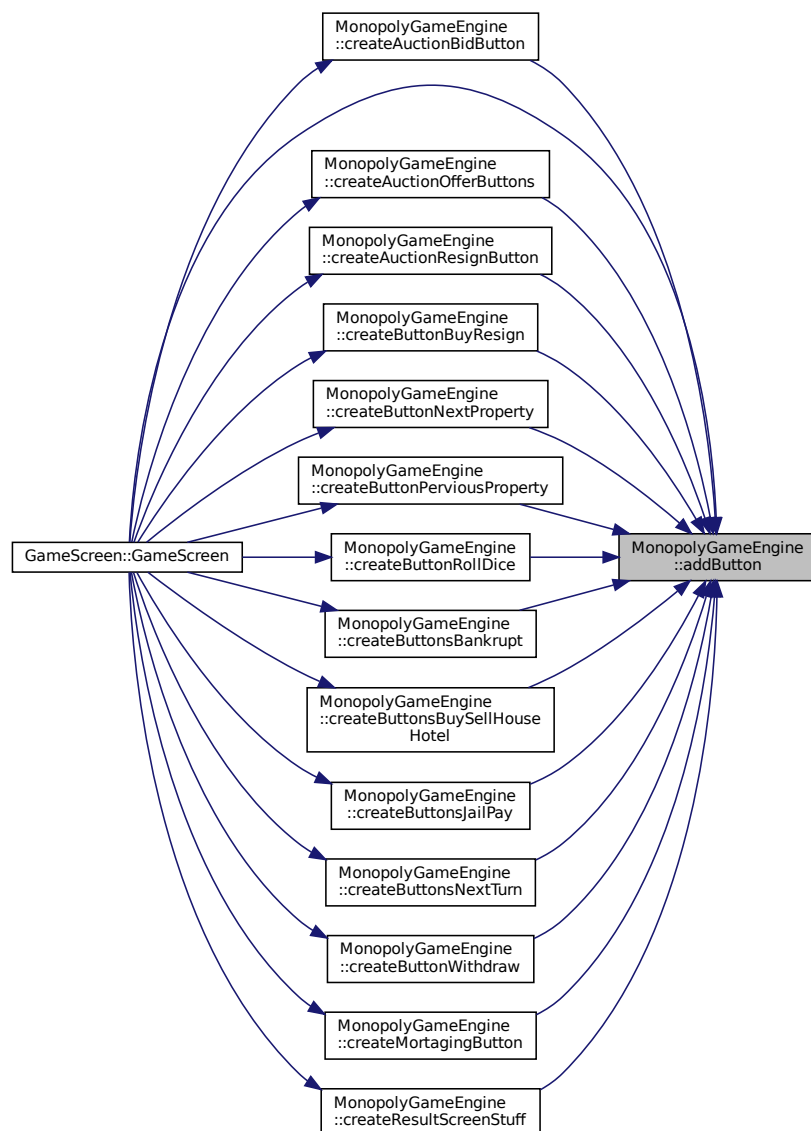
```
void MonopolyGameEngine::addButton (
    std::shared_ptr< Button > button_tmp )
```

Adds a button to the GUI.

#### Parameters

<i>button_tmp</i>	Pointer to the button to be added.
-------------------	------------------------------------

Here is the caller graph for this function:



### 6.18.3.4 addHotels()

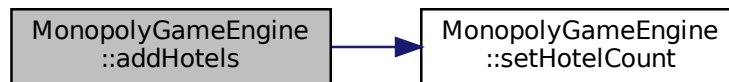
```
void MonopolyGameEngine::addHotels (
    unsigned int added_amount )
```

Adds hotels to the available count.

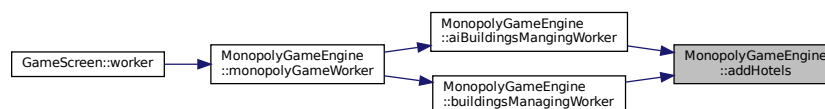
#### Parameters

<i>added_amount</i>	Amount to be added.
---------------------	---------------------

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.5 addHouses()

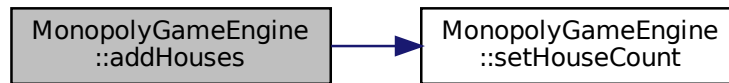
```
void MonopolyGameEngine::addHouses (
    unsigned int added_amount )
```

Adds houses to the available count.

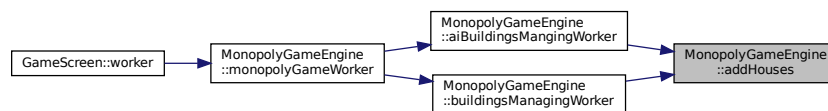
#### Parameters

<i>added_amount</i>	Amount to be added.
---------------------	---------------------

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.18.3.6 addText()

```
void MonopolyGameEngine::addText (
    std::shared_ptr< sf::Text > text_tmp )
```

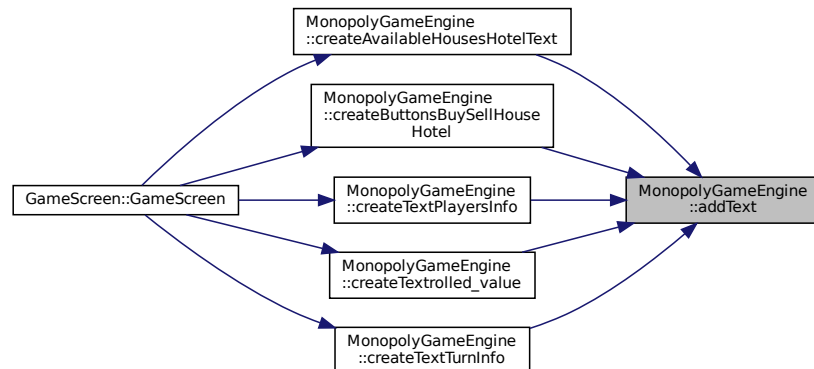
Adds text to the GUI.

##### Parameters

<i>text_tmp</i>	Pointer to the text to be added.
-----------------	----------------------------------



Here is the caller graph for this function:

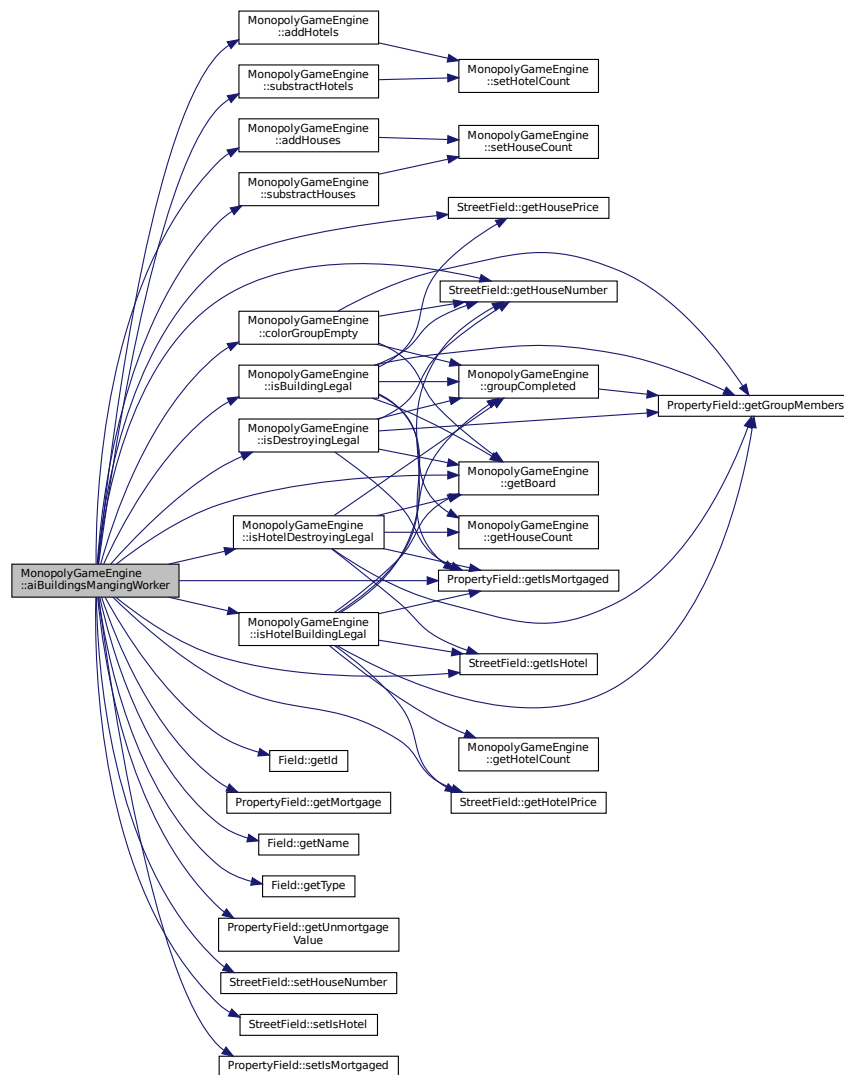


#### 6.18.3.7 aiBuildingsMangingWorker()

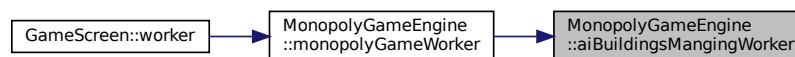
```
void MonopolyGameEngine::aiBuildingsMangingWorker ( )
```

AI worker method for managing building and destroying houses/hotels.

Here is the call graph for this function:



Here is the caller graph for this function:

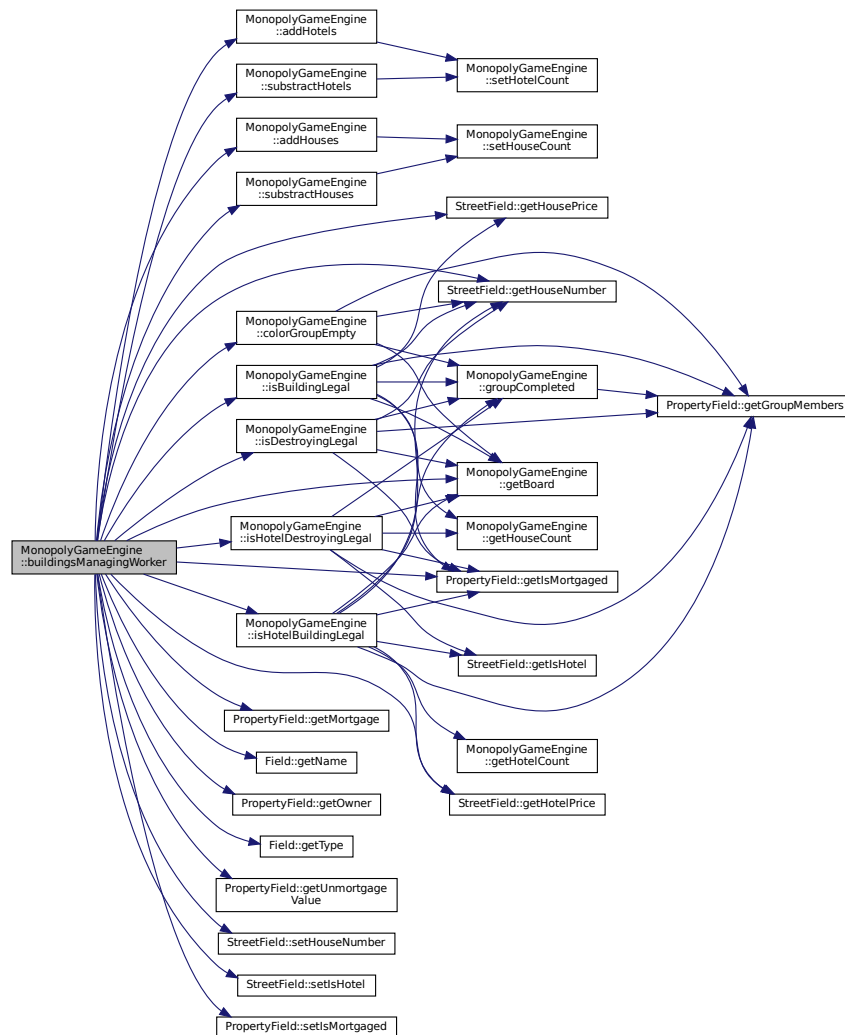


### 6.18.3.8 buildingsManagingWorker()

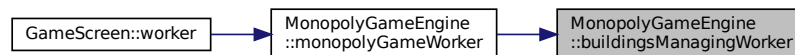
```
void MonopolyGameEngine::buildingsManagingWorker ( )
```

Worker method for managing building and destroying houses/hotels.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.9 calculateGroupFieldsOwned()

```

unsigned int MonopolyGameEngine::calculateGroupFieldsOwned (
    std::vector< unsigned int > player_fields,
    PropertyField & field ) const

```

Calculates the number of group fields owned by a player.

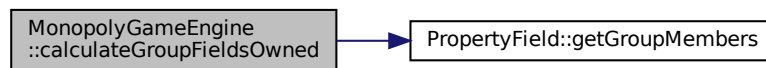
#### Parameters

<i>player_fields</i>	Vector of field positions owned by the player.
<i>field</i>	Property field to check for group completion.

#### Returns

Number of group fields owned.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.18.3.10 calculateRent()

```

unsigned int MonopolyGameEngine::calculateRent (
    unsigned int rolled_val,
    int pos )
  
```

Calculates the rent to be paid based on the rolled dice value and property position.

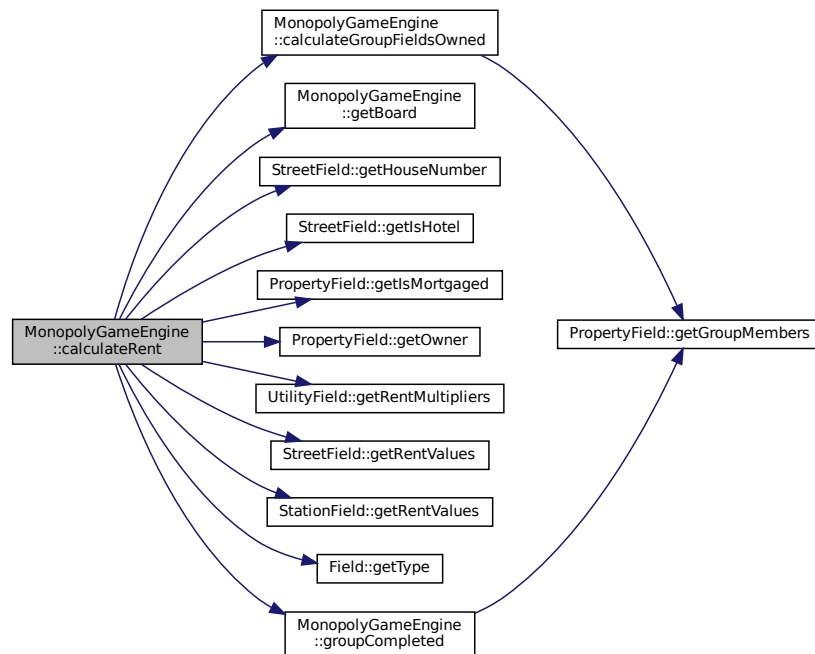
#### Parameters

<i>rolled_val</i>	Rolled dice value.
<i>pos</i>	Property position.

## Returns

Calculated rent amount.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.11 clearBoard()

```
void MonopolyGameEngine::clearBoard ( )
```

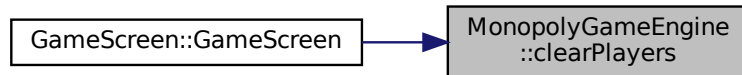
Clears the game board.

### 6.18.3.12 clearPlayers()

```
void MonopolyGameEngine::clearPlayers ( )
```

Clears player objects from the game.

Here is the caller graph for this function:



### 6.18.3.13 colorGroupEmpty()

```
bool MonopolyGameEngine::colorGroupEmpty (
    std::shared_ptr< Player > mortgaging,
    StreetField & field )
```

Checks if a color group is empty (no properties owned) for a player.

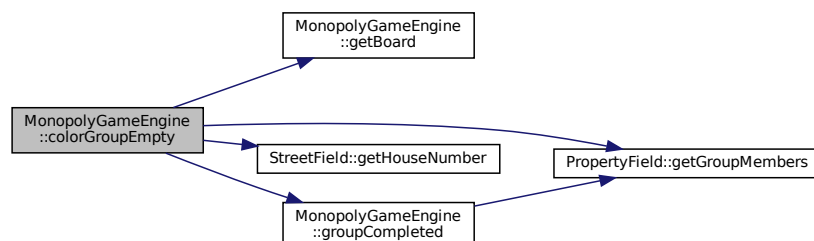
#### Parameters

<i>mortgaging</i>	<code>Player</code> attempting to mortgage properties.
<i>field</i>	Street field to check for a color group.

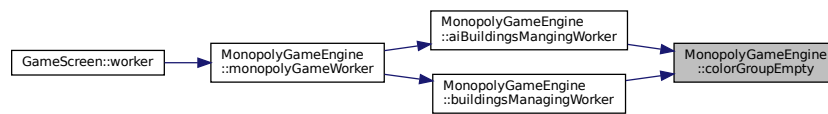
#### Returns

True if the color group is empty, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:

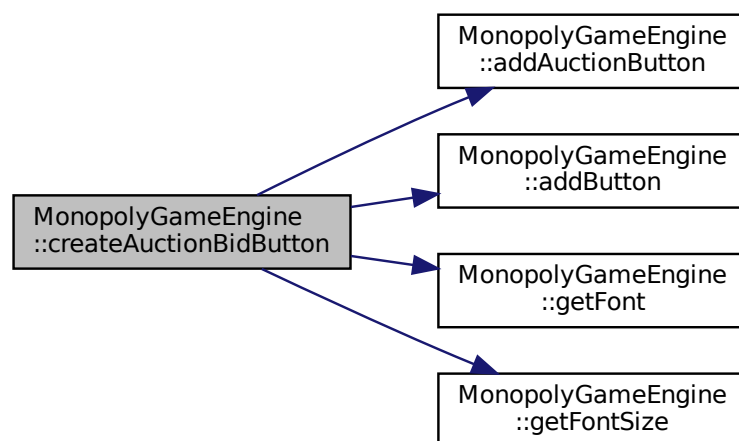


### 6.18.3.14 createAuctionBidButton()

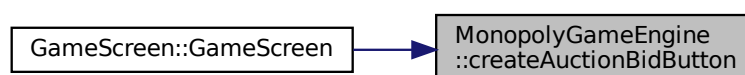
```
void MonopolyGameEngine::createAuctionBidButton ( )
```

Creates a button for participating in an auction by bidding.

Here is the call graph for this function:



Here is the caller graph for this function:

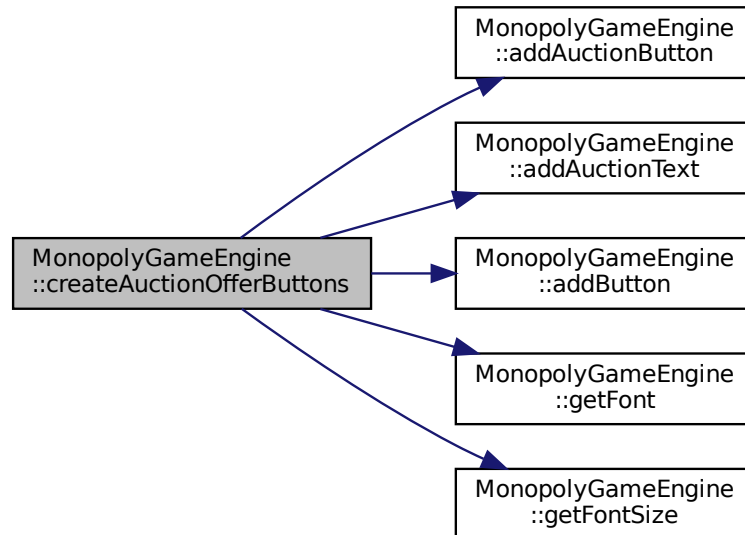


### 6.18.3.15 createAuctionOfferButtons()

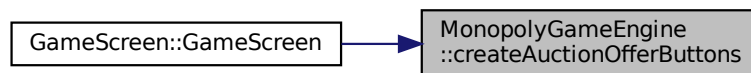
```
void MonopolyGameEngine::createAuctionOfferButtons ( )
```

Creates buttons for handling auction offers.

Here is the call graph for this function:



Here is the caller graph for this function:



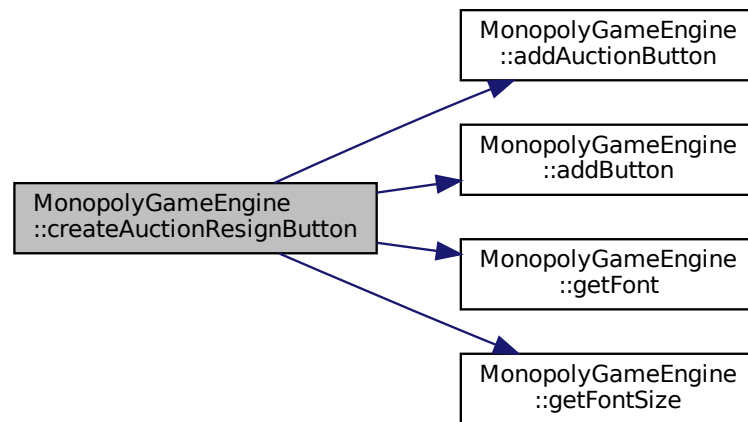
### 6.18.3.16 createAuctionResignButton()

```
void MonopolyGameEngine::createAuctionResignButton ( )
```

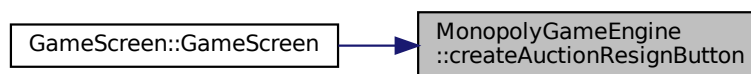
Creates a button for resigning from an auction.



Here is the call graph for this function:



Here is the caller graph for this function:

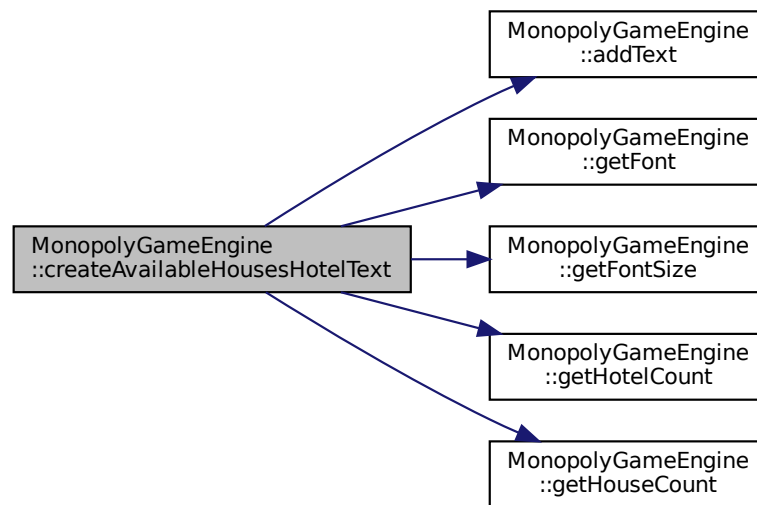


### 6.18.3.17 createAvailableHousesHotelText()

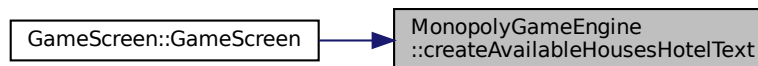
```
void MonopolyGameEngine::createAvailableHousesHotelText ( )
```

Creates text displaying the available count of houses and hotels.

Here is the call graph for this function:



Here is the caller graph for this function:

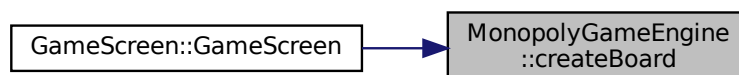


#### 6.18.3.18 createBoard()

```
void MonopolyGameEngine::createBoard ( )
```

Creates the game board.

Here is the caller graph for this function:

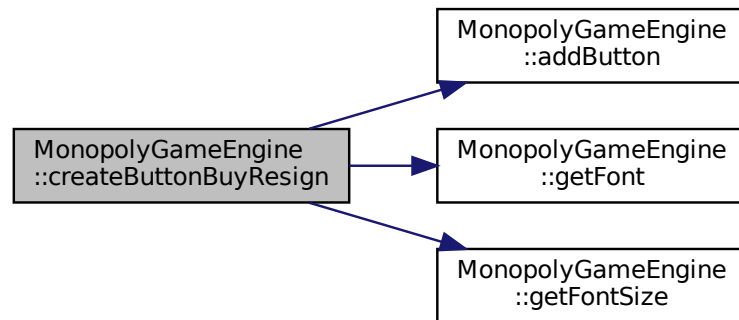


### 6.18.3.19 createButtonBuyResign()

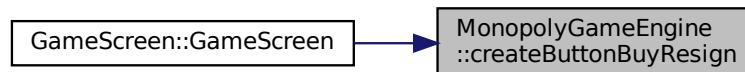
```
void MonopolyGameEngine::createButtonBuyResign ( )
```

Creates a button for buying or resigning from a property.

Here is the call graph for this function:



Here is the caller graph for this function:

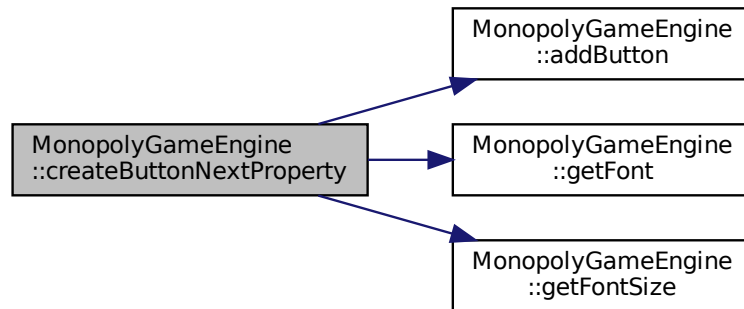


### 6.18.3.20 createButtonNextProperty()

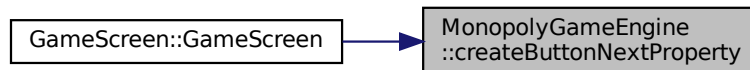
```
void MonopolyGameEngine::createButtonNextProperty ( )
```

Creates a button for moving to the next property.

Here is the call graph for this function:



Here is the caller graph for this function:

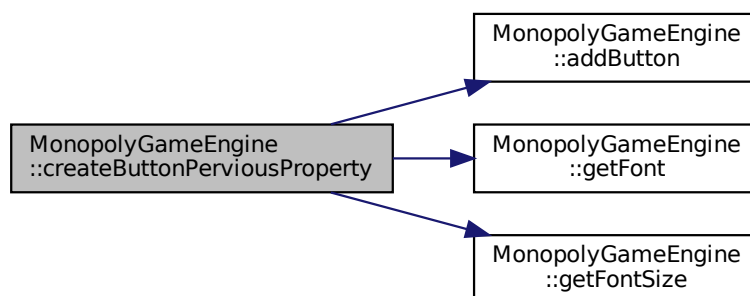


#### 6.18.3.21 createButtonPerviousProperty()

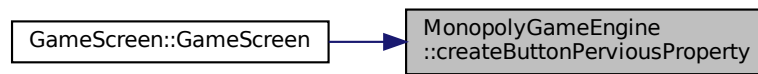
```
void MonopolyGameEngine::createButtonPerviousProperty ( )
```

Creates a button for moving to the previous property.

Here is the call graph for this function:



Here is the caller graph for this function:

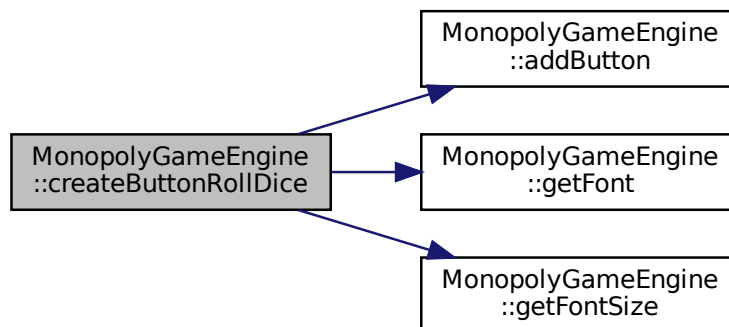


### 6.18.3.22 createButtonRollDice()

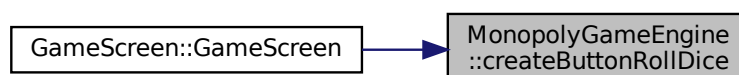
```
void MonopolyGameEngine::createButtonRollDice ( )
```

Creates the button for rolling the dice.

Here is the call graph for this function:



Here is the caller graph for this function:

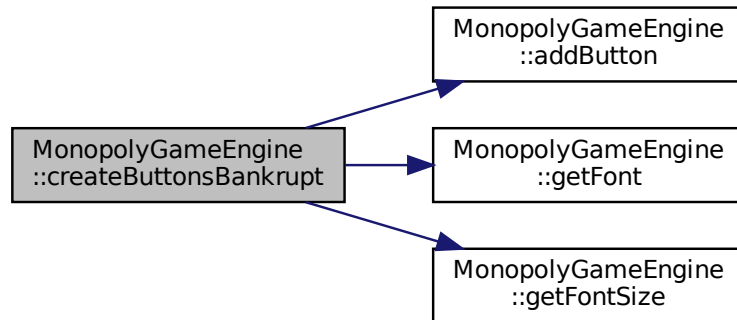


### 6.18.3.23 createButtonsBankrupt()

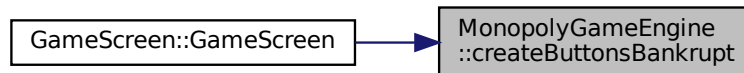
```
void MonopolyGameEngine::createButtonsBankrupt ( )
```

Creates buttons for handling bankruptcy.

Here is the call graph for this function:



Here is the caller graph for this function:

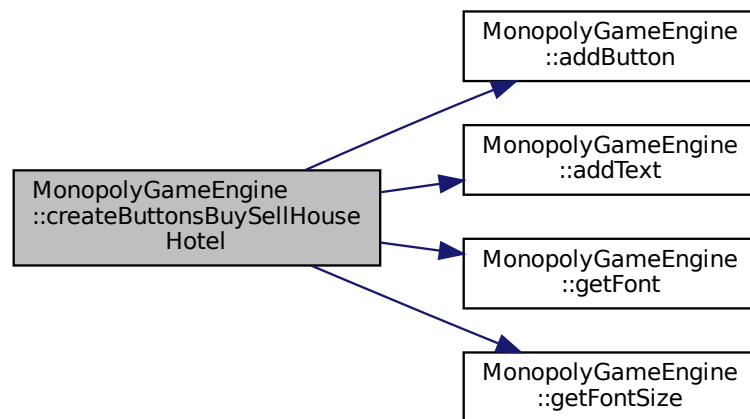


### 6.18.3.24 createButtonsBuySellHouseHotel()

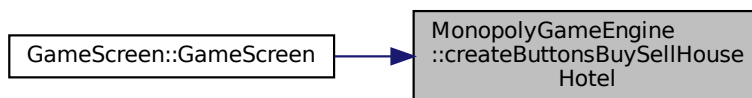
```
void MonopolyGameEngine::createButtonsBuySellHouseHotel ( )
```

Creates buttons for buying, selling houses, and hotels.

Here is the call graph for this function:



Here is the caller graph for this function:

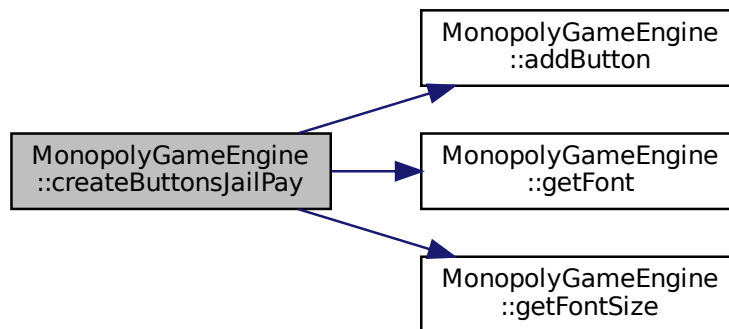


### 6.18.3.25 createButtonsJailPay()

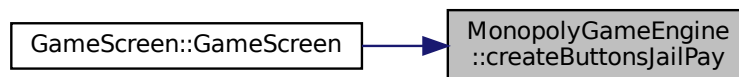
```
void MonopolyGameEngine::createButtonsJailPay ( )
```

Creates buttons for handling actions related to the jail (paying to get out).

Here is the call graph for this function:



Here is the caller graph for this function:



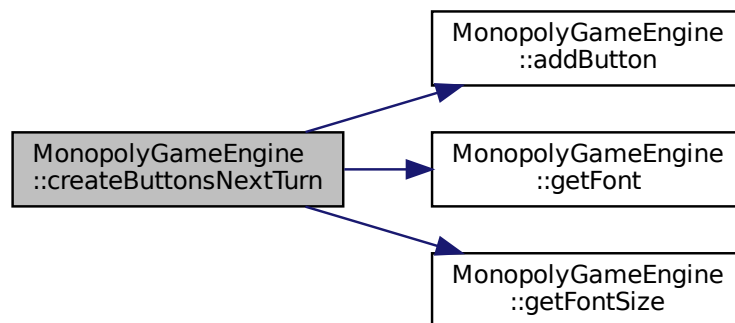
#### 6.18.3.26 createButtonsNextTurn()

```
void MonopolyGameEngine::createButtonsNextTurn ( )
```

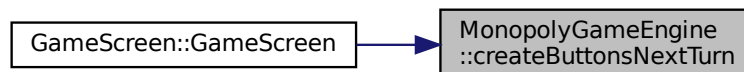
Creates buttons for moving to the next turn.



Here is the call graph for this function:



Here is the caller graph for this function:

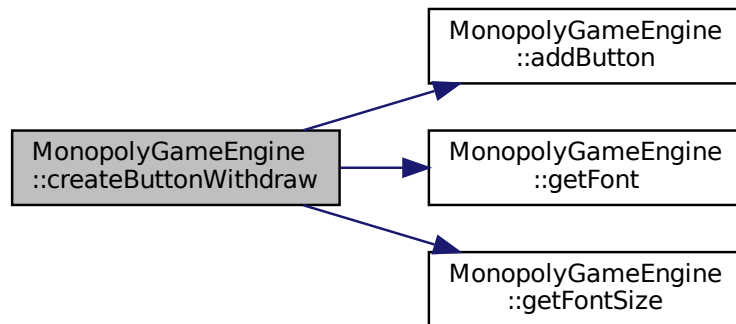


#### 6.18.3.27 createButtonWithdraw()

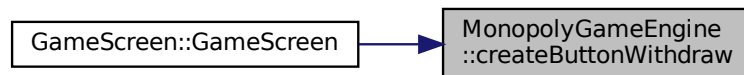
```
void MonopolyGameEngine::createButtonWithdraw ( )
```

Creates a button for withdrawing from the game.

Here is the call graph for this function:



Here is the caller graph for this function:

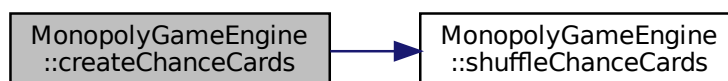


#### 6.18.3.28 createChanceCards()

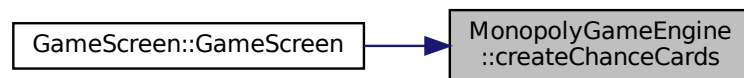
```
void MonopolyGameEngine::createChanceCards ( )
```

Creates the chance cards for the game.

This method initializes and populates the collection of [ChanceCard](#) objects used in the game. Here is the call graph for this function:



Here is the caller graph for this function:

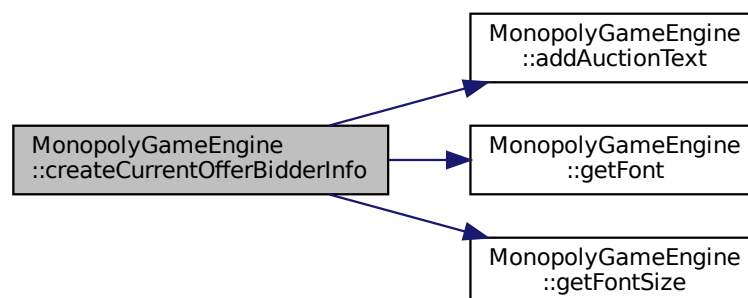


### 6.18.3.29 createCurrentOfferBidderInfo()

```
void MonopolyGameEngine::createCurrentOfferBidderInfo ( )
```

Creates text for displaying the current offer in an auction.

Here is the call graph for this function:



Here is the caller graph for this function:

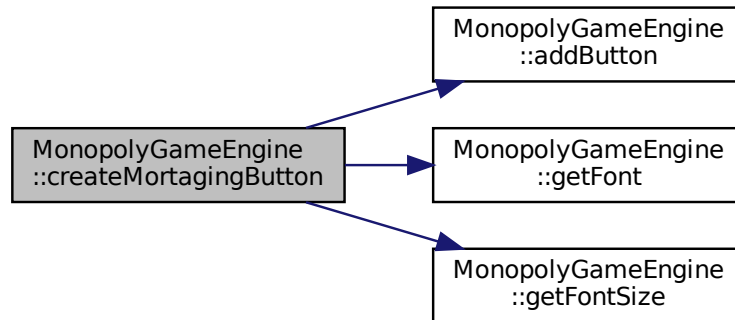


### 6.18.3.30 createMortagingButton()

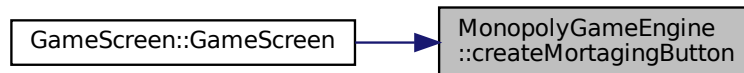
```
void MonopolyGameEngine::createMortagingButton ( )
```

Creates buttons for mortgaging properties.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.31 createPlayers()

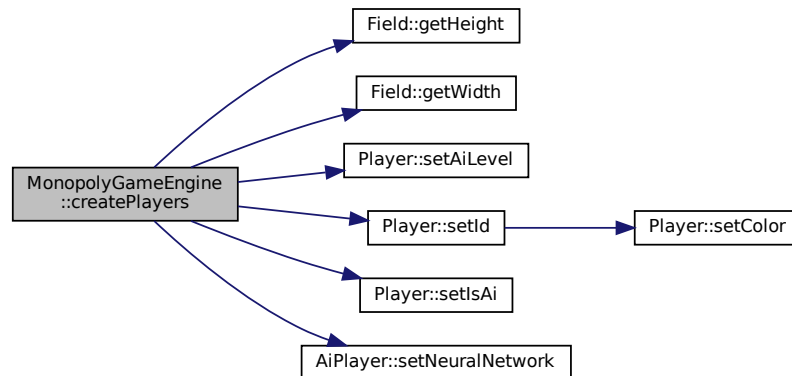
```
void MonopolyGameEngine::createPlayers (
    std::vector< std::shared_ptr< Player >> & players_from_game_engine )
```

Creates player objects based on input vector.

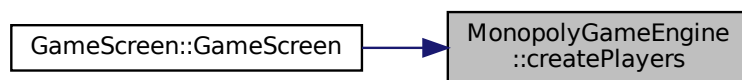
#### Parameters

<i>players_from_game_engine</i>	Vector of pointers to players.
---------------------------------	--------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:

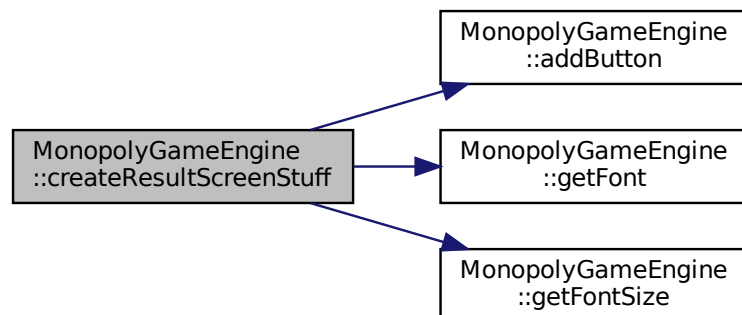


### 6.18.3.32 createResultScreenStuff()

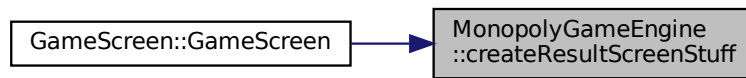
```
void MonopolyGameEngine::createResultScreenStuff ( )
```

Creates various elements for the result screen.

Here is the call graph for this function:



Here is the caller graph for this function:

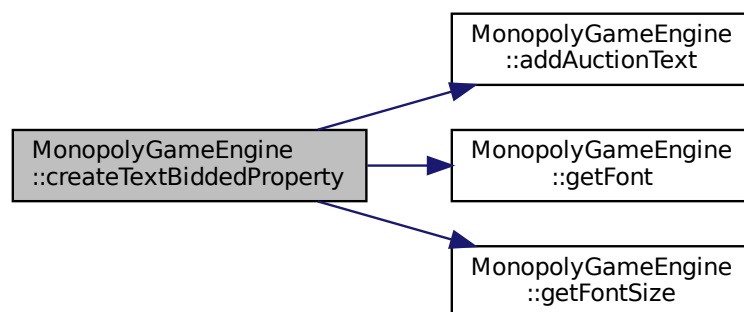


#### 6.18.3.33 createTextBiddedProperty()

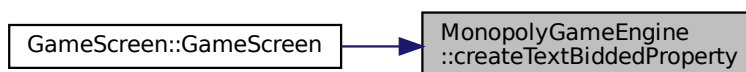
```
void MonopolyGameEngine::createTextBiddedProperty ( )
```

Creates text for displaying information about the bidded property.

Here is the call graph for this function:



Here is the caller graph for this function:

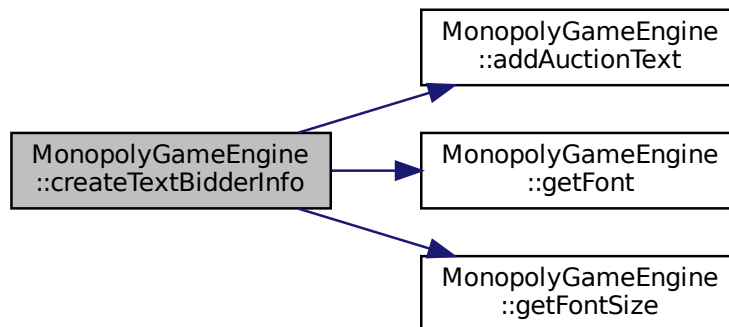


### 6.18.3.34 createTextBidderInfo()

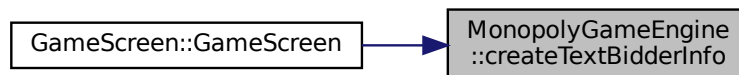
```
void MonopolyGameEngine::createTextBidderInfo ( )
```

Creates text for displaying information about the bidder.

Here is the call graph for this function:



Here is the caller graph for this function:

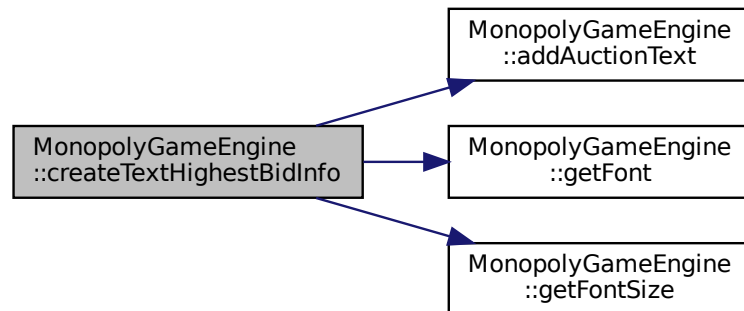


### 6.18.3.35 createTextHighestBidInfo()

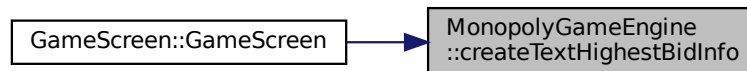
```
void MonopolyGameEngine::createTextHighestBidInfo ( )
```

Creates text for displaying the highest bid in an auction.

Here is the call graph for this function:



Here is the caller graph for this function:

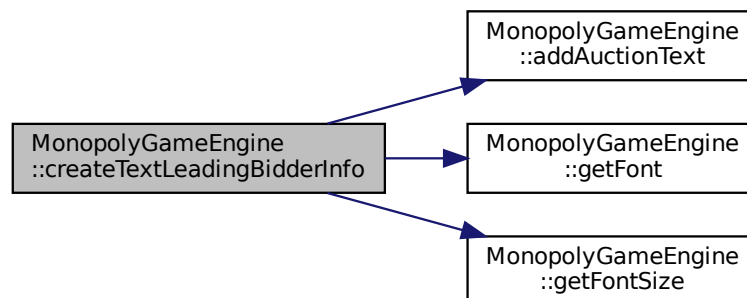


#### 6.18.3.36 createTextLeadingBidderInfo()

```
void MonopolyGameEngine::createTextLeadingBidderInfo ( )
```

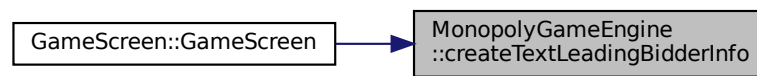
Creates text for displaying information about the leading bidder in an auction.

Here is the call graph for this function:





Here is the caller graph for this function:

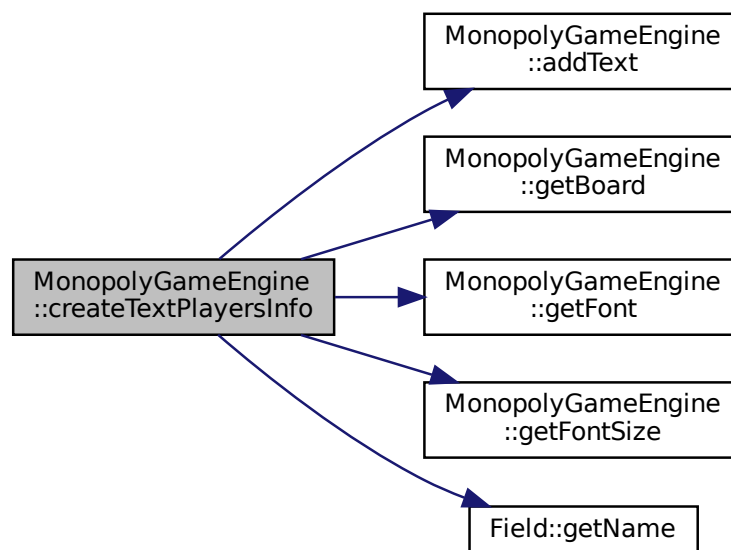


### 6.18.3.37 createTextPlayersInfo()

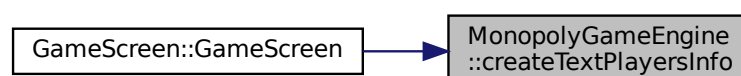
```
void MonopolyGameEngine::createTextPlayersInfo ( )
```

Creates text for displaying players' information.

Here is the call graph for this function:



Here is the caller graph for this function:

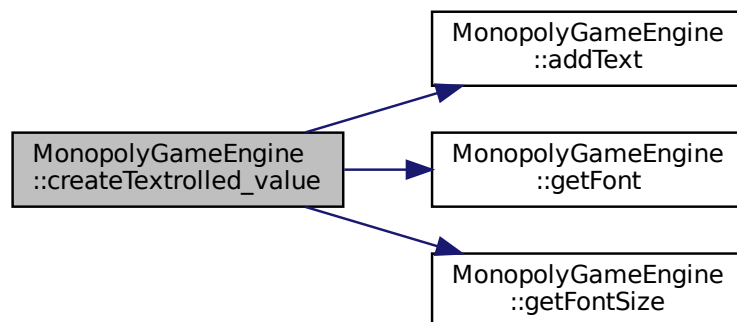


### 6.18.3.38 createTextrolled\_value()

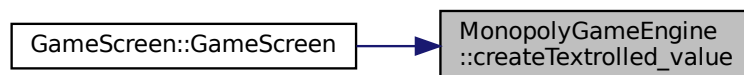
```
void MonopolyGameEngine::createTextrolled_value ( )
```

Creates text for displaying the rolled dice value.

Here is the call graph for this function:



Here is the caller graph for this function:

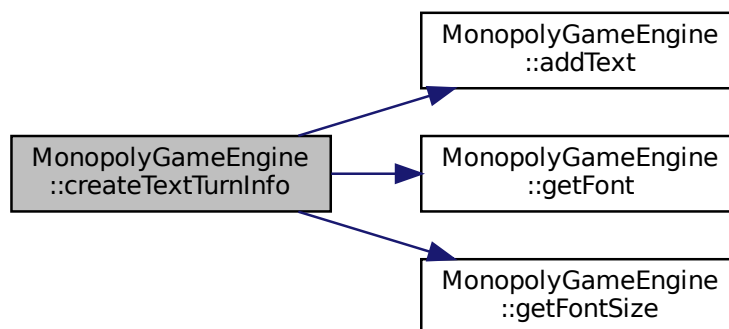


### 6.18.3.39 createTextTurnInfo()

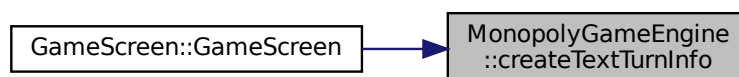
```
void MonopolyGameEngine::createTextTurnInfo ( )
```

Creates text for displaying turn information.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.18.3.40 getAllPropertyDataSprite()

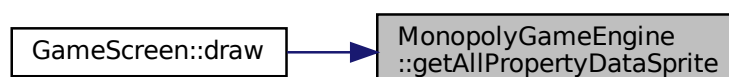
```
sf::Sprite & MonopolyGameEngine::getAllPropertyDataSprite ( )
```

Returns the sprite representing all property data in the GUI.

##### Returns

Reference to the all property data sprite.

Here is the caller graph for this function:



#### 6.18.3.41 getAllPropertyDataTexts()

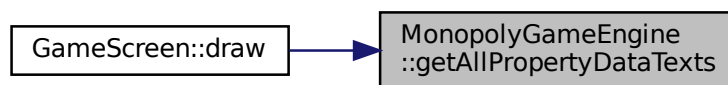
```
std::vector< std::shared_ptr< sf::Text > > & MonopolyGameEngine::getAllPropertyDataTexts ( )
```

Returns a vector of pointers to text elements representing all property data in the GUI.

##### Returns

Vector of pointers to all property data text elements.

Here is the caller graph for this function:



#### 6.18.3.42 getAuctionButtons()

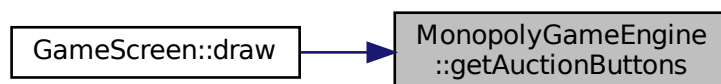
```
std::vector< std::shared_ptr< Button > > & MonopolyGameEngine::getAuctionButtons ( )
```

Returns a vector of pointers to auction buttons in the GUI.

##### Returns

Vector of pointers to auction buttons.

Here is the caller graph for this function:



### 6.18.3.43 getAuctionState()

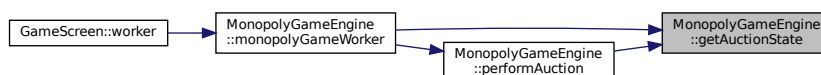
```
AuctionState MonopolyGameEngine::getAuctionState ( )
```

Gets the current state of the auction.

#### Returns

Current state of the auction.

Here is the caller graph for this function:



### 6.18.3.44 getAuctionTexts()

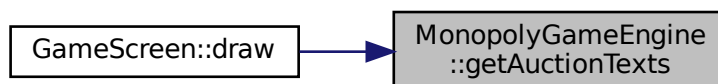
```
std::vector< std::shared_ptr< sf::Text > > & MonopolyGameEngine::getAuctionTexts ( )
```

Returns a vector of pointers to auction text elements in the GUI.

#### Returns

Vector of pointers to auction text elements.

Here is the caller graph for this function:





### 6.18.3.47 getFont()

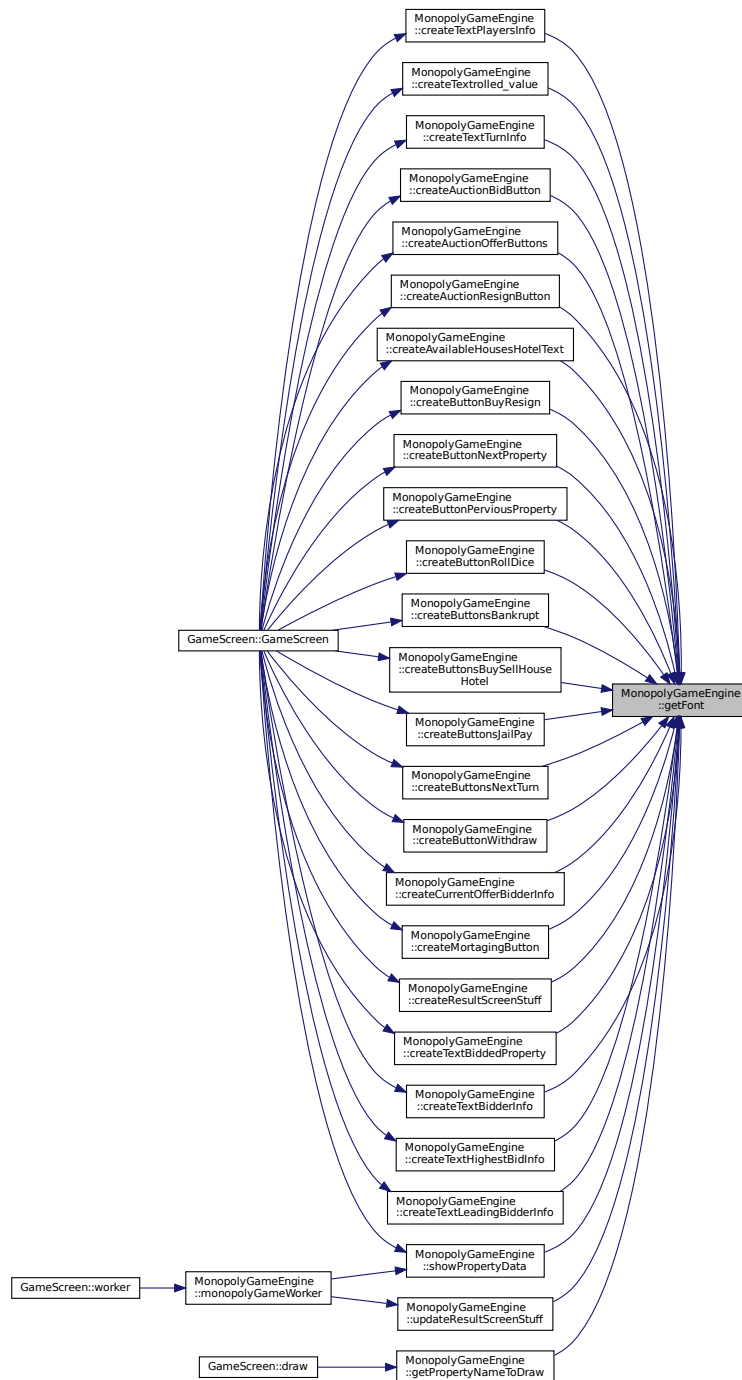
```
sf::Font & MonopolyGameEngine::getFont ( )
```

Returns the font used in the GUI.

#### Returns

Reference to the font.

Here is the caller graph for this function:



#### 6.18.3.48 getFontSize()

```
unsigned int MonopolyGameEngine::getFontSize ( ) const
```

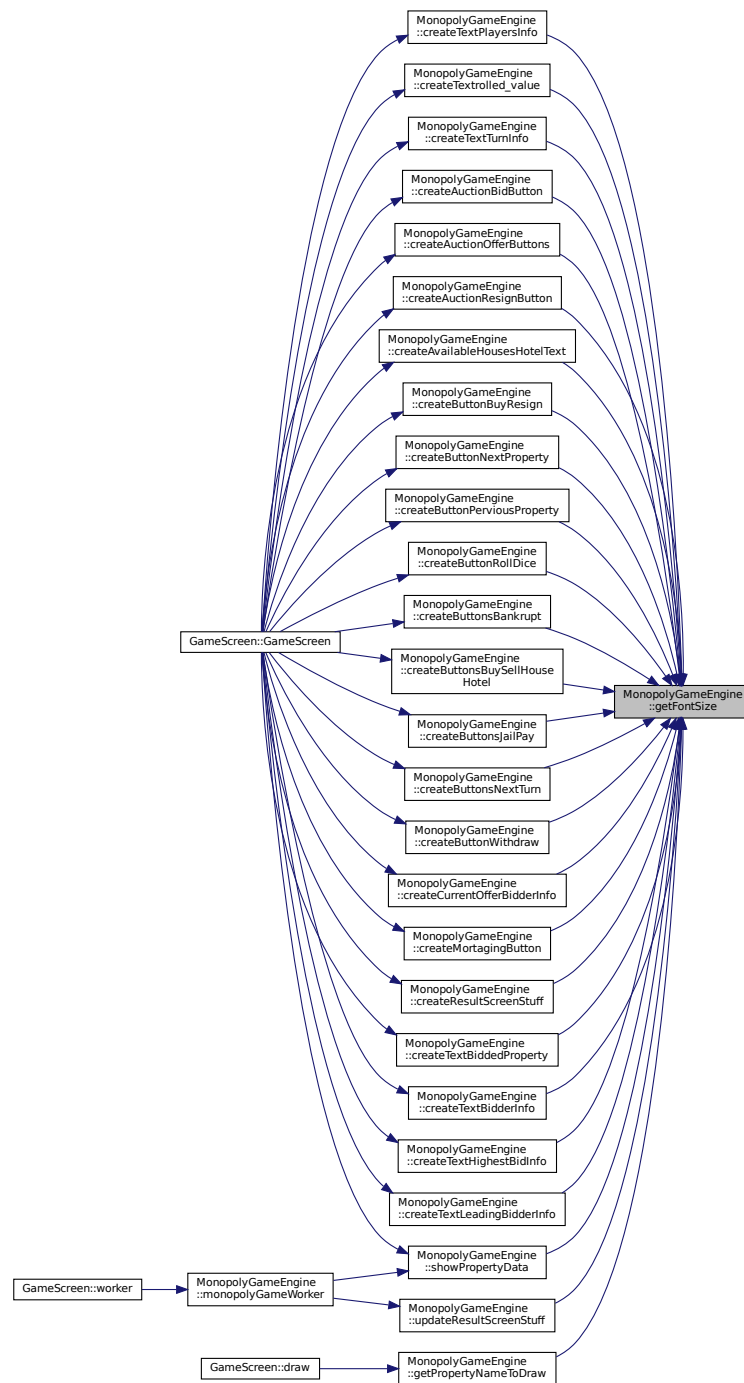
Returns the font size used in the GUI.

##### Returns

Font size.



Here is the caller graph for this function:



### 6.18.3.49 getHotelCount()

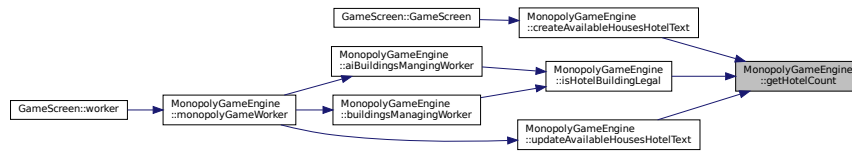
```
unsigned int MonopolyGameEngine::getHotelCount ( )
```

Gets the count of hotels available for purchase.

**Returns**

Count of hotels.

Here is the caller graph for this function:

**6.18.3.50 getHotelSprite()**

```
sf::Sprite MonopolyGameEngine::getHotelSprite (
    StreetField & field )
```

Gets the sprite for a hotel on a street.

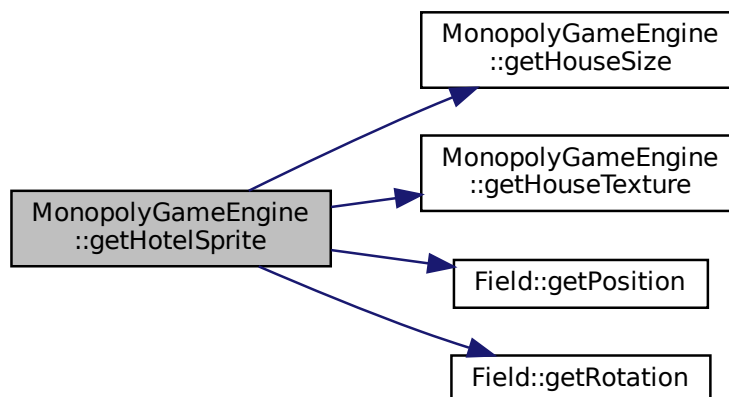
**Parameters**

<i>field</i>	Street field to get the hotel sprite for.
--------------	---

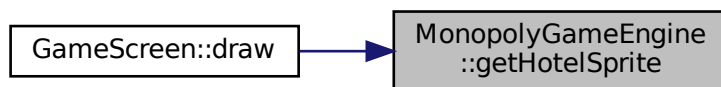
**Returns**

Sprite of the hotel.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.18.3.51 `getHotelTexture()`

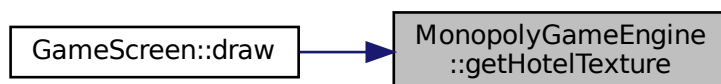
```
sf::Texture & MonopolyGameEngine::getHotelTexture ( )
```

Returns the texture of the hotel used in the GUI.

##### Returns

Reference to the hotel texture.

Here is the caller graph for this function:



#### 6.18.3.52 `getHouseCount()`

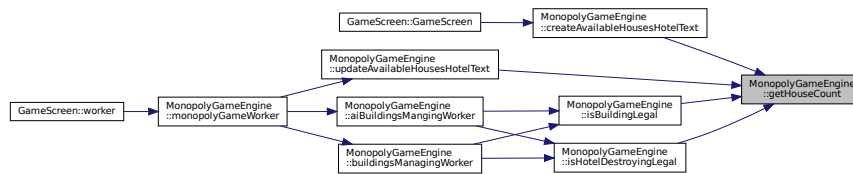
```
unsigned int MonopolyGameEngine::getHouseCount ( )
```

Gets the count of houses available for purchase.

**Returns**

Count of houses.

Here is the caller graph for this function:

**6.18.3.53 getHouseSize()**

```
sf::Vector2f & MonopolyGameEngine::getHouseSize ( )
```

Returns the size of a house in the GUI.

**Returns**

Reference to the house size.

Here is the caller graph for this function:

**6.18.3.54 getHouseSprite()**

```
sf::Sprite MonopolyGameEngine::getHouseSprite (
    StreetField & field,
    unsigned int houses_number )
```

Gets the sprite for a house on a street.

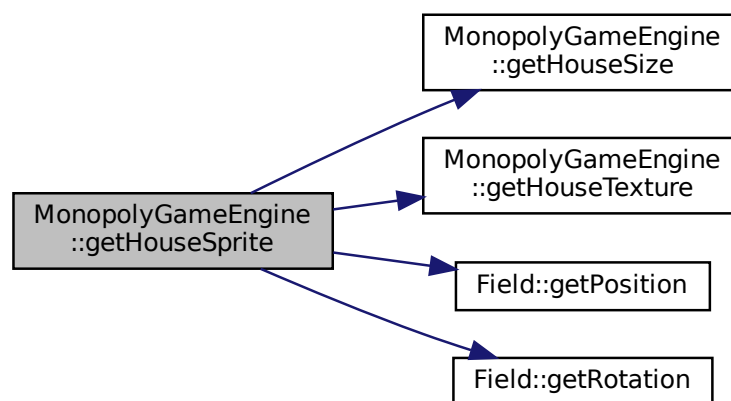
## Parameters

<i>field</i>	Street field to get the house sprite for.
<i>houses_number</i>	Number of houses on the street.

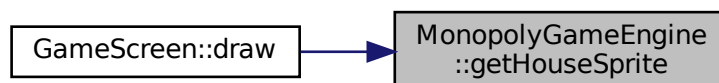
## Returns

Sprite of the house.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.55 getHouseTexture()

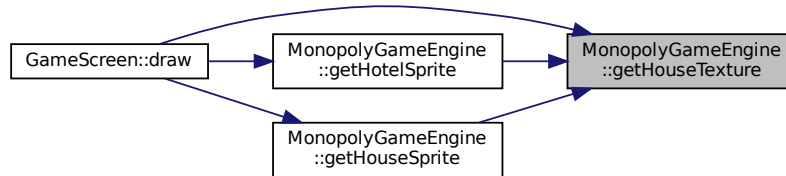
```
sf::Texture & MonopolyGameEngine::getHouseTexture ( )
```

Returns the texture of the house used in the GUI.

**Returns**

Reference to the house texture.

Here is the caller graph for this function:

**6.18.3.56 getNotificationsWall()**

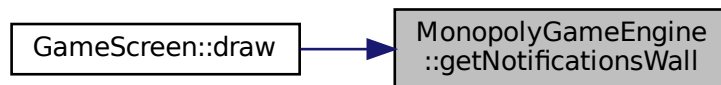
```
NotificationWall & MonopolyGameEngine::getNotificationsWall ( )
```

Returns the notification wall used in the GUI.

**Returns**

Reference to the notification wall.

Here is the caller graph for this function:



### 6.18.3.57 getPlayers()

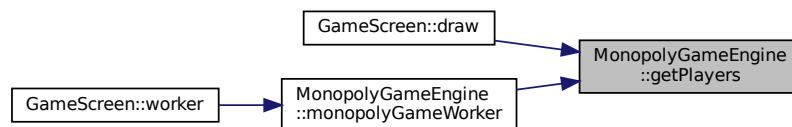
```
std::vector< std::shared_ptr< Player > > & MonopolyGameEngine::getPlayers ( )
```

Returns a vector of pointers to players in the game.

#### Returns

Vector of pointers to players.

Here is the caller graph for this function:



### 6.18.3.58 getPlayersResult()

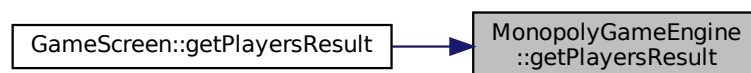
```
std::vector< std::shared_ptr< Player > > MonopolyGameEngine::getPlayersResult ( )
```

Returns a vector of pointers to players based on game result.

#### Returns

Vector of pointers to players.

Here is the caller graph for this function:



### 6.18.3.59 getPropertyDataSprite()

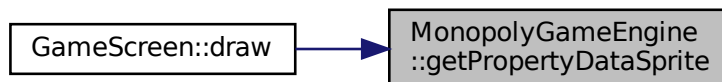
```
sf::Sprite & MonopolyGameEngine::getPropertyDataSprite ( )
```

Returns the sprite representing property data in the GUI.

#### Returns

Reference to the property data sprite.

Here is the caller graph for this function:



### 6.18.3.60 getPropertyDataTexts()

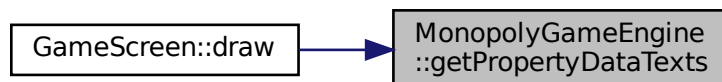
```
std::vector< std::shared_ptr< sf::Text > > & MonopolyGameEngine::getPropertyDataTexts ( )
```

Returns a vector of pointers to text elements representing property data in the GUI.

#### Returns

Vector of pointers to property data text elements.

Here is the caller graph for this function:



### 6.18.3.61 getPropertyNameToDraw()

```
sf::Text MonopolyGameEngine::getPropertyNameToDraw (
    sf::Text text,
    sf::Sprite & sprite,
    float rotation )
```

Gets the text suitable for drawing a property name.



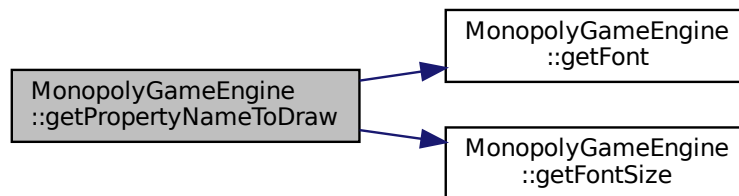
## Parameters

<i>text</i>	Original text to be drawn.
<i>sprite</i>	Reference to the sprite associated with the property.
<i>rotation</i>	Rotation angle of the text.

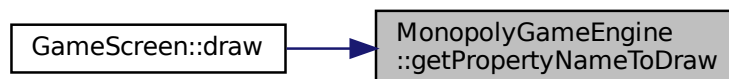
## Returns

Modified text for drawing.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.62 getResultTexts()

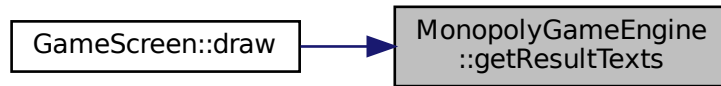
```
std::vector< std::shared_ptr< sf::Text > > & MonopolyGameEngine::getResultTexts ( )
```

Returns a vector of pointers to text elements used in the result screen.

**Returns**

Vector of pointers to result screen text elements.

Here is the caller graph for this function:

**6.18.3.63 getScreenType()**

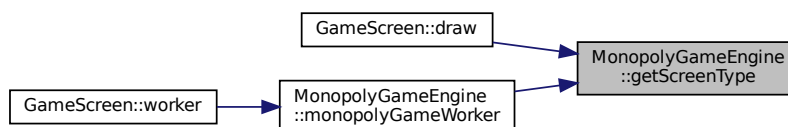
```
GameScreenType MonopolyGameEngine::getScreenType ( ) const
```

Gets the current screen type.

**Returns**

The current screen type.

Here is the caller graph for this function:

**6.18.3.64 getTexts()**

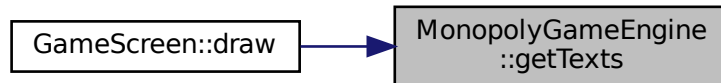
```
std::vector< std::shared_ptr< sf::Text > > & MonopolyGameEngine::getTexts ( )
```

Returns a vector of pointers to text elements in the GUI.

**Returns**

Vector of pointers to text elements.

Here is the caller graph for this function:

**6.18.3.65 getTurnState()**

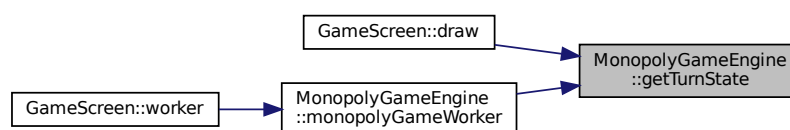
```
TurnState MonopolyGameEngine::getTurnState ( ) const
```

Gets the current turn state.

**Returns**

Current turn state.

Here is the caller graph for this function:

**6.18.3.66 getWithdraw()**

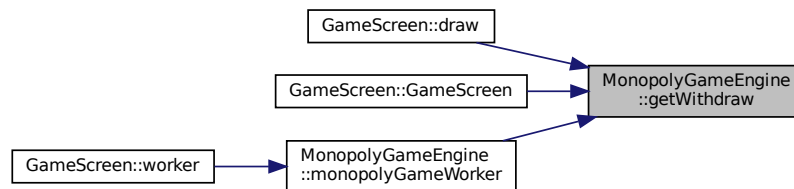
```
Withdraw & MonopolyGameEngine::getWithdraw ( )
```

Returns a reference to the [Withdraw](#) object.

**Returns**

[Withdraw](#) & Reference to the [Withdraw](#) object.

Here is the caller graph for this function:

**6.18.3.67 groupCompleted()**

```

bool MonopolyGameEngine::groupCompleted (
    std::vector< unsigned int > player_fields,
    PropertyField & field ) const
  
```

Checks if a group of properties is completed by a player.

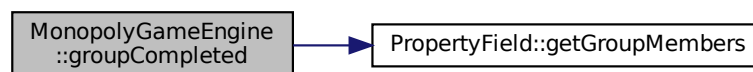
**Parameters**

<i>player_fields</i>	Vector of field positions owned by the player.
<i>field</i>	Property field to check for group completion.

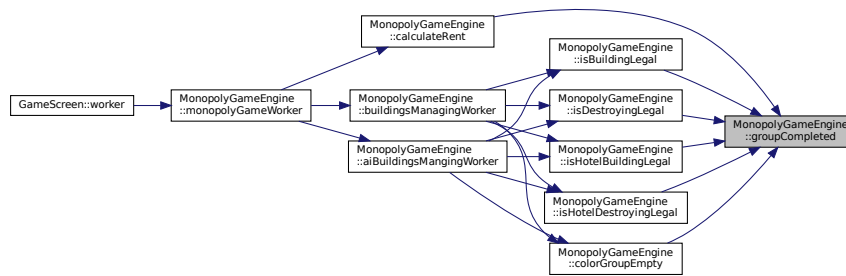
**Returns**

True if the group is completed, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.68 isBuildingLegal()

```

bool MonopolyGameEngine::isBuildingLegal (
    std::shared_ptr< Player > builder,
    StreetField field )

```

Checks if building houses on a street is legal for a player.

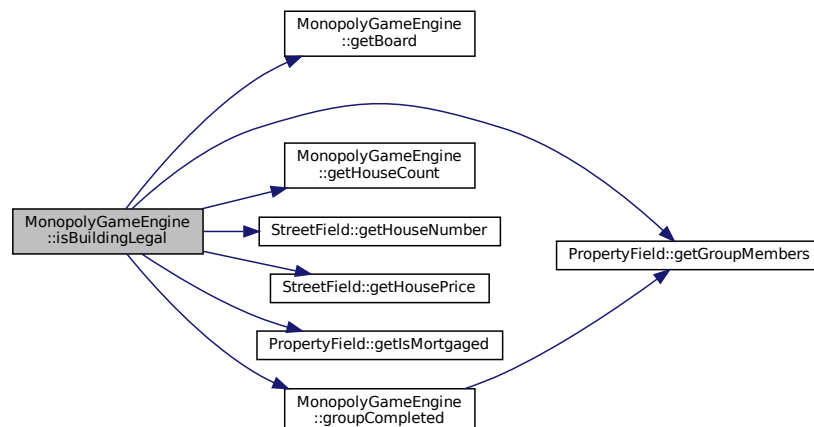
#### Parameters

<i>builder</i>	<a href="#">Player</a> attempting to build houses.
<i>field</i>	Street field to build houses on.

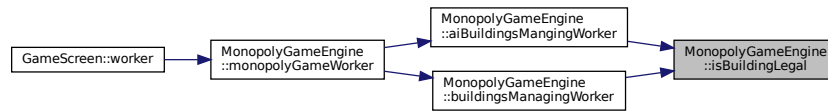
#### Returns

True if building is legal, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.69 isDestroyingLegal()

```

bool MonopolyGameEngine::isDestroyingLegal (
    std::shared_ptr< Player > builder,
    StreetField field )
  
```

Checks if destroying houses on a street is legal for a player.

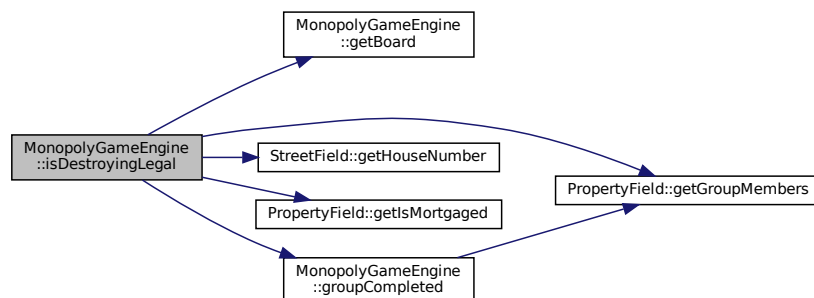
#### Parameters

<i>builder</i>	<a href="#">Player</a> attempting to destroy houses.
<i>field</i>	Street field to destroy houses on.

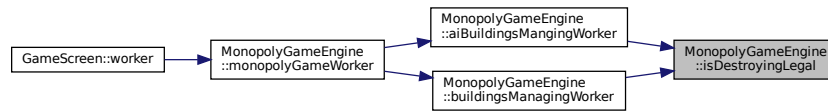
#### Returns

True if destroying is legal, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.70 isHotelBuildingLegal()

```

bool MonopolyGameEngine::isHotelBuildingLegal (
    std::shared_ptr< Player > builder,
    StreetField & field )
  
```

Checks if building a hotel on a street is legal for a player.

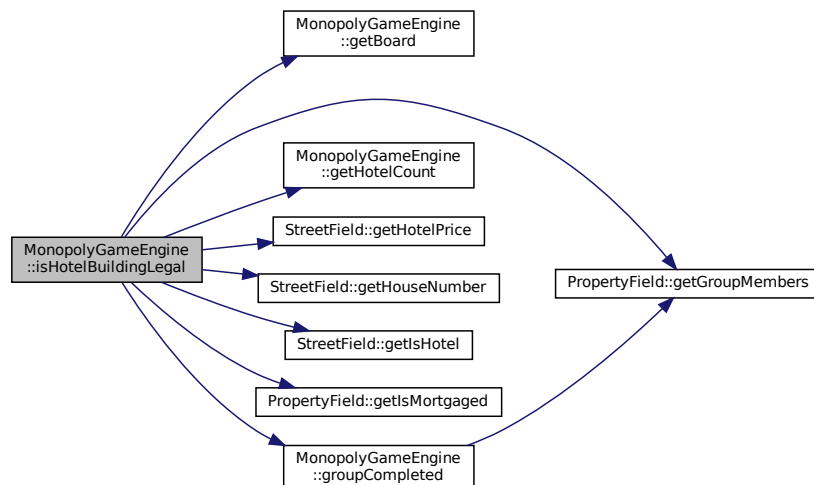
#### Parameters

<i>builder</i>	<a href="#">Player</a> attempting to build a hotel.
<i>field</i>	Street field to build a hotel on.

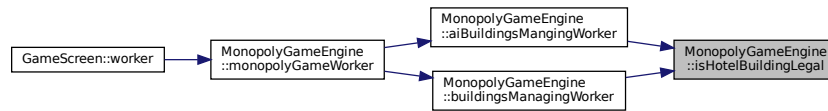
#### Returns

True if building a hotel is legal, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.71 isHotelDestroyingLegal()

```

bool MonopolyGameEngine::isHotelDestroyingLegal (
    std::shared_ptr< Player > builder,
    StreetField & field )
  
```

Checks if destroying a hotel on a street is legal for a player.

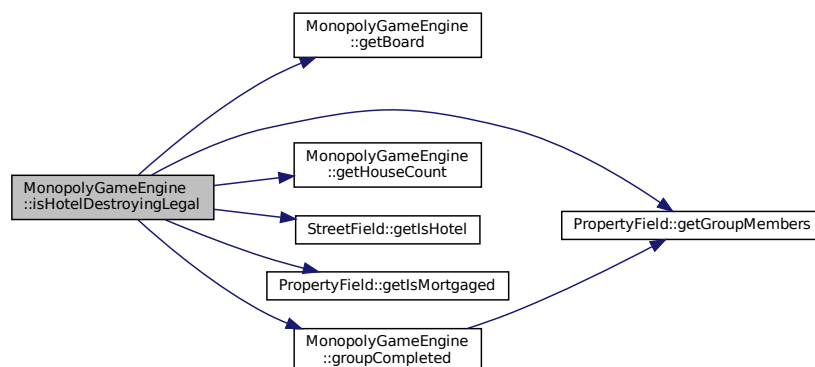
#### Parameters

<i>builder</i>	<a href="#">Player</a> attempting to destroy a hotel.
<i>field</i>	Street field to destroy a hotel on.

#### Returns

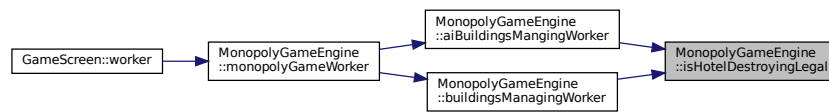
True if destroying a hotel is legal, false otherwise.

Here is the call graph for this function:





Here is the caller graph for this function:

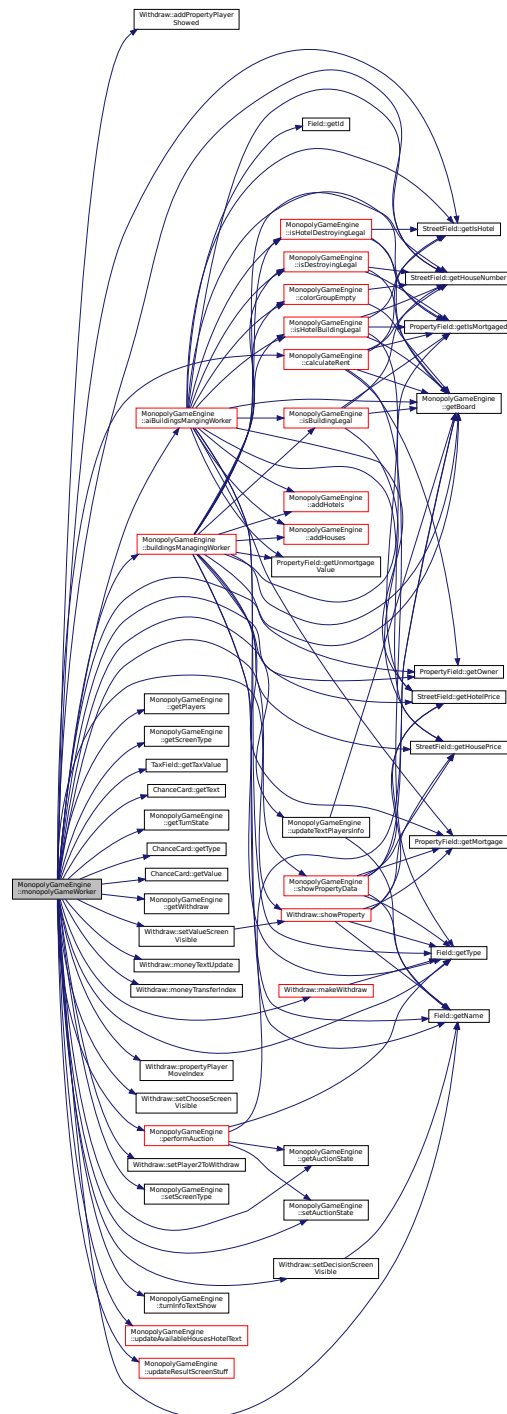


### 6.18.3.72 monopolyGameWorker()

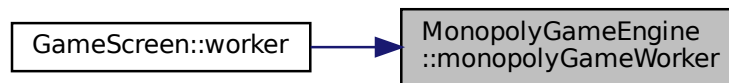
```
bool MonopolyGameEngine::monopolyGameWorker ( )
```

Main Worker method for whole monopoly game engine.

Here is the call graph for this function:



Here is the caller graph for this function:

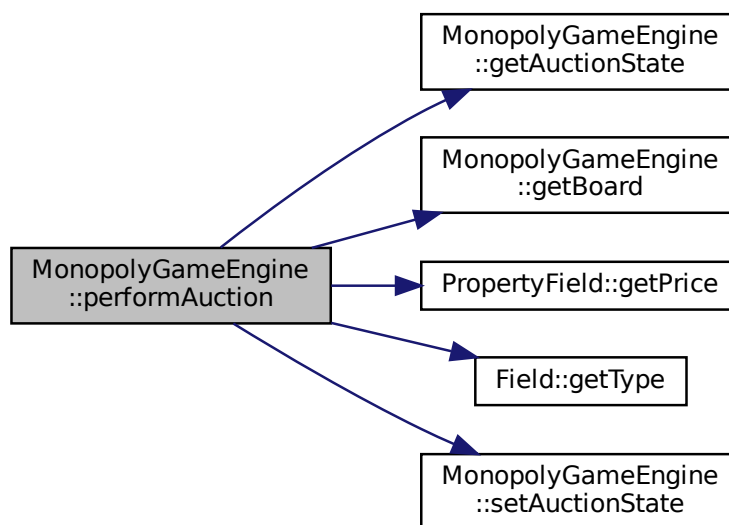


### 6.18.3.73 performAuction()

```
void MonopolyGameEngine::performAuction ( )
```

Performs the auction.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.74 setAuctionState()

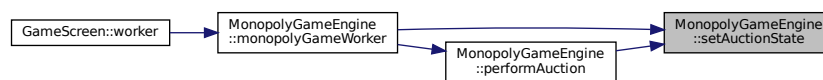
```
void MonopolyGameEngine::setAuctionState (
    AuctionState new_state )
```

Sets the state of the auction.

#### Parameters

<i>new_state</i>	New state of the auction.
------------------	---------------------------

Here is the caller graph for this function:



### 6.18.3.75 setFont()

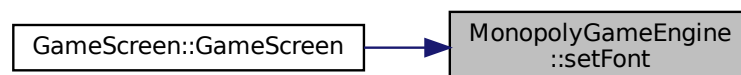
```
void MonopolyGameEngine::setFont (
    sf::Font font )
```

Sets the font to be used in the GUI.

#### Parameters

<i>font</i>	Font to be set.
-------------	-----------------

Here is the caller graph for this function:



### 6.18.3.76 setHotelCount()

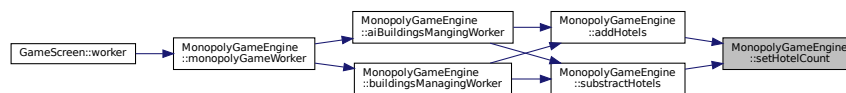
```
void MonopolyGameEngine::setHotelCount (
    unsigned int new_count )
```

Sets the count of hotels available for purchase.

#### Parameters

<i>new_count</i>	New count of hotels.
------------------	----------------------

Here is the caller graph for this function:



### 6.18.3.77 setHouseCount()

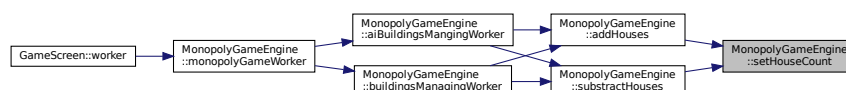
```
void MonopolyGameEngine::setHouseCount (
    unsigned int new_count )
```

Sets the count of houses available for purchase.

#### Parameters

<i>new_count</i>	New count of houses.
------------------	----------------------

Here is the caller graph for this function:



### 6.18.3.78 setplayer\_index\_turn()

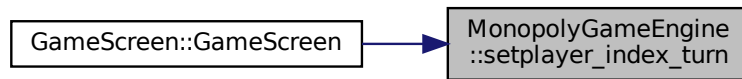
```
void MonopolyGameEngine::setplayer_index_turn (
    unsigned int indx )
```

Sets the index of the player whose turn it is.

#### Parameters

<i>indx</i>	Index of the player.
-------------	----------------------

Here is the caller graph for this function:



### 6.18.3.79 setScreenType()

```
void MonopolyGameEngine::setScreenType (
    GameScreenType new_screen_type )
```

Sets the screen type to the specified type.

#### Parameters

<i>new_screen_type</i>	The new screen type.
------------------------	----------------------

Here is the caller graph for this function:



### 6.18.3.80 showPropertyData()

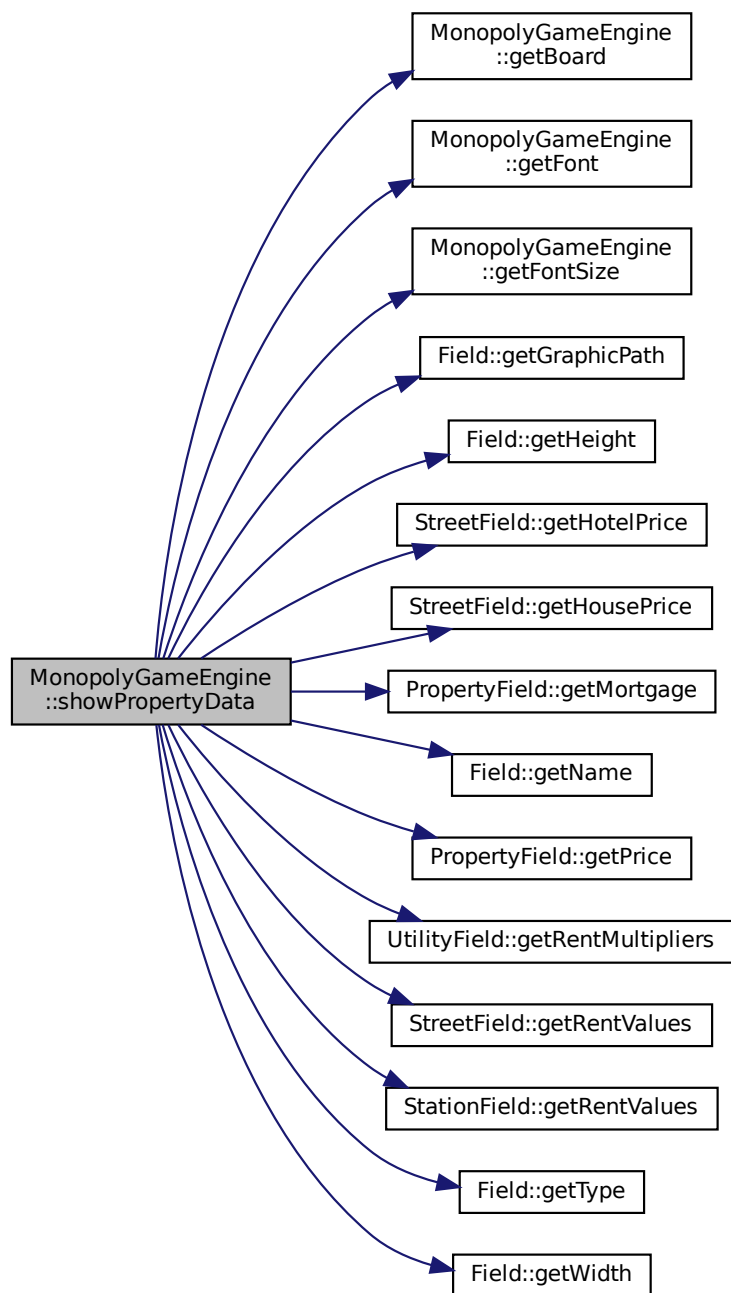
```
void MonopolyGameEngine::showPropertyData (
    unsigned int pos,
    bool is_property_shown_to_buy )
```

Shows property data on the GUI based on the position and whether it is shown to buy.

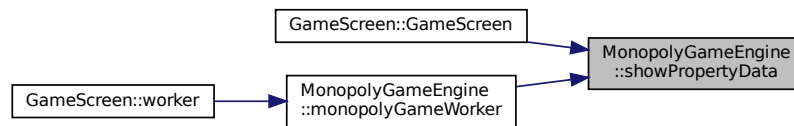
#### Parameters

<i>pos</i>	The position of the property.
<i>is_property_shown_to_buy</i>	Indicates whether the property information is shown for buying.

Here is the call graph for this function:



Here is the caller graph for this function:

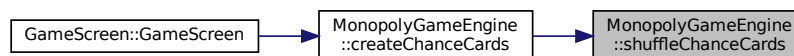


### 6.18.3.81 shuffleChanceCards()

```
void MonopolyGameEngine::shuffleChanceCards ( )
```

Shuffles the chance cards.

This method shuffles the collection of [ChanceCard](#) objects to randomize the order in which they will be drawn during the game. Here is the caller graph for this function:



### 6.18.3.82 subtractHotels()

```
void MonopolyGameEngine::subtractHotels (
    unsigned int subtracted_amount )
```

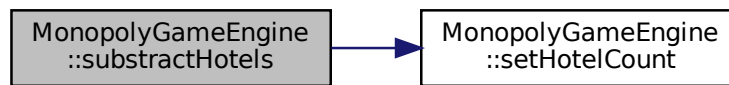
Subtracts hotels from the available count.

#### Parameters

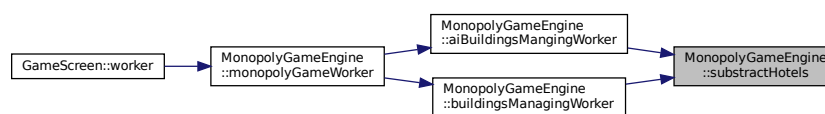
<i>subtracted_amount</i>	Amount to be subtracted.
--------------------------	--------------------------



Here is the call graph for this function:



Here is the caller graph for this function:



### 6.18.3.83 subtractHouses()

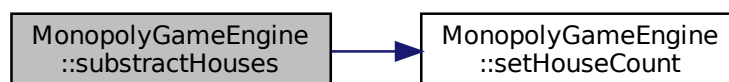
```
void MonopolyGameEngine::subtractHouses (
    unsigned int subtracted_amount )
```

Subtracts houses from the available count.

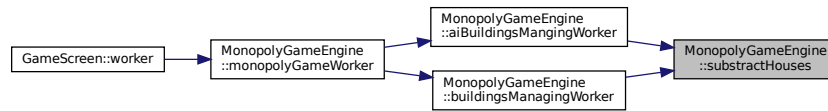
#### Parameters

<i>subtracted_amount</i>	Amount to be subtracted.
--------------------------	--------------------------

Here is the call graph for this function:



Here is the caller graph for this function:

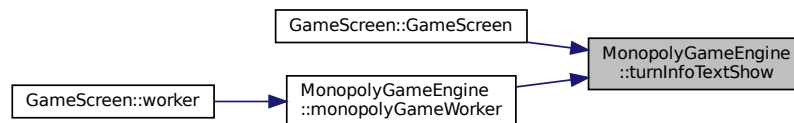


#### 6.18.3.84 turnInfoTextShow()

```
void MonopolyGameEngine::turnInfoTextShow ( )
```

Displays text related to turn information.

Here is the caller graph for this function:

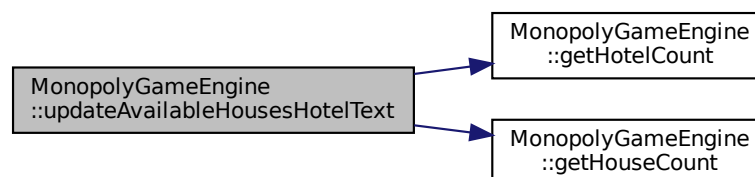


#### 6.18.3.85 updateAvailableHousesHotelText()

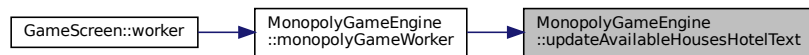
```
void MonopolyGameEngine::updateAvailableHousesHotelText ( )
```

Updates the text displaying the available count of houses and hotels.

Here is the call graph for this function:



Here is the caller graph for this function:

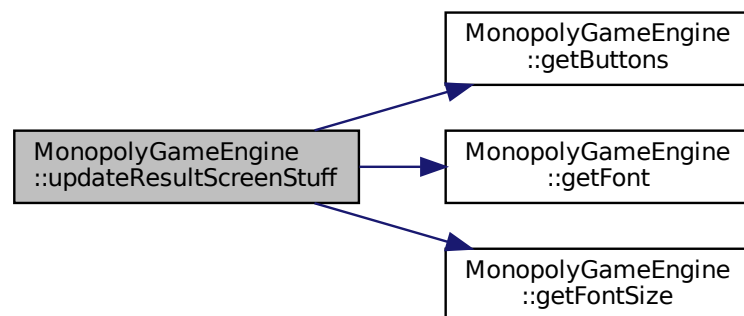


### 6.18.3.86 updateResultScreenStuff()

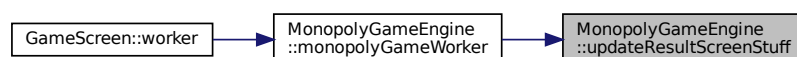
```
void MonopolyGameEngine::updateResultScreenStuff ( )
```

Updates elements on the result screen.

Here is the call graph for this function:



Here is the caller graph for this function:

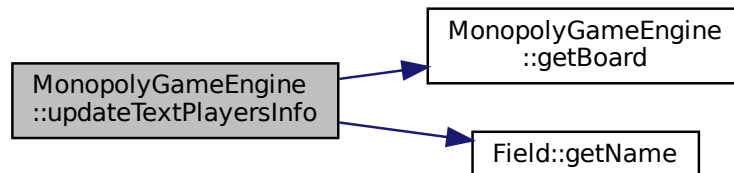


### 6.18.3.87 updateTextPlayersInfo()

```
void MonopolyGameEngine::updateTextPlayersInfo ( )
```

Updates the text displaying players' information.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- [/home/kamil/zpr/Monopoly/MonopolyGameEngine.h](#)
- [/home/kamil/zpr/Monopoly/MonopolyGameEngine.cc](#)

## 6.19 neat::mutation\_rate\_container Struct Reference

```
#include <Tinyneat.h>
```

### Public Member Functions

- void [read](#) (std::ifstream &o)
- void [write](#) (std::ofstream &o, std::string prefix)

## Public Attributes

- double `connection_mutate_chance` = 0.25
- double `perturb_chance` = 0.90
- double `crossover_chance` = 0.75
- double `link_mutation_chance` = 2.0
- double `node_mutation_chance` = 0.50
- double `bias_mutation_chance` = 0.40
- double `step_size` = 0.1
- double `disable_mutation_chance` = 0.4
- double `enable_mutation_chance` = 0.2

## 6.19.1 Member Function Documentation

### 6.19.1.1 read()

```
void neat::mutation_rate_container::read (
    std::ifstream & o )
```

Here is the caller graph for this function:



### 6.19.1.2 write()

```
void neat::mutation_rate_container::write (
    std::ofstream & o,
    std::string prefix )
```

## 6.19.2 Member Data Documentation

#### 6.19.2.1 bias\_mutation\_chance

```
double neat::mutation_rate_container::bias_mutation_chance = 0.40
```

#### 6.19.2.2 connection\_mutate\_chance

```
double neat::mutation_rate_container::connection_mutate_chance = 0.25
```

#### 6.19.2.3 crossover\_chance

```
double neat::mutation_rate_container::crossover_chance = 0.75
```

#### 6.19.2.4 disable\_mutation\_chance

```
double neat::mutation_rate_container::disable_mutation_chance = 0.4
```

#### 6.19.2.5 enable\_mutation\_chance

```
double neat::mutation_rate_container::enable_mutation_chance = 0.2
```

#### 6.19.2.6 link\_mutation\_chance

```
double neat::mutation_rate_container::link_mutation_chance = 2.0
```

#### 6.19.2.7 node\_mutation\_chance

```
double neat::mutation_rate_container::node_mutation_chance = 0.50
```

#### 6.19.2.8 perturb\_chance

```
double neat::mutation_rate_container::perturb_chance = 0.90
```

### 6.19.2.9 step\_size

```
double neat::mutation_rate_container::step_size = 0.1
```

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

- [/home/kamil/zpr/Monopoly/Tinyneat.h](#)
- [/home/kamil/zpr/Monopoly/Tinyneat.cc](#)

## 6.20 neat::network\_info\_container Struct Reference

```
#include <Tinyneat.h>
```

### Public Attributes

- unsigned int [input\\_size](#)
- unsigned int [bias\\_size](#)
- unsigned int [output\\_size](#)
- unsigned int [functional\\_nodes](#)
- bool [recurrent](#)

### 6.20.1 Member Data Documentation

#### 6.20.1.1 bias\_size

```
unsigned int neat::network_info_container::bias_size
```

#### 6.20.1.2 functional\_nodes

```
unsigned int neat::network_info_container::functional_nodes
```

#### 6.20.1.3 input\_size

```
unsigned int neat::network_info_container::input_size
```

#### 6.20.1.4 output\_size

```
unsigned int neat::network_info_container::output_size
```

#### 6.20.1.5 recurrent

```
bool neat::network_info_container::recurrent
```

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

- [/home/kamil/zpr/Monopoly/Tinyneat.h](#)

## 6.21 ann::neuralnet Class Reference

```
#include <Tinyann.h>
```

### Public Member Functions

- [neuralnet](#) ()
- void [from\\_genome](#) (const [neat::genome](#) &a)
- void [evaluate](#) (const std::vector< double > &input, std::vector< double > &output)
- void [import\\_fromfile](#) (std::string filename)
- void [export\\_tofile](#) (std::string filename)

### 6.21.1 Constructor & Destructor Documentation

#### 6.21.1.1 neuralnet()

```
ann::neuralnet::neuralnet ( )
```

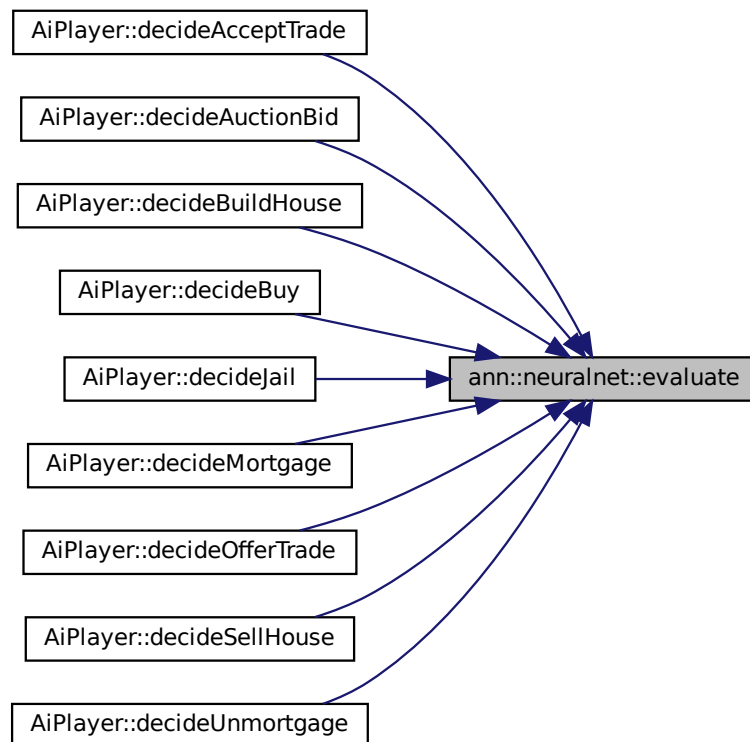
### 6.21.2 Member Function Documentation



### 6.21.2.1 evaluate()

```
void ann::neuralnet::evaluate (
    const std::vector< double > & input,
    std::vector< double > & output )
```

Here is the caller graph for this function:



### 6.21.2.2 export\_tofile()

```
void ann::neuralnet::export_tofile (
    std::string filename )
```

### 6.21.2.3 from\_genome()

```
void ann::neuralnet::from_genome (
    const neat::genome & a )
```

#### 6.21.2.4 import\_fromfile()

```
void ann::neuralnet::import_fromfile (
    std::string filename )
```

Here is the caller graph for this function:



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

- [/home/kamil/zpr/Monopoly/Tinyann.h](#)
- [/home/kamil/zpr/Monopoly/Tinyann.cc](#)

## 6.22 ann::neuron Class Reference

```
#include <Tinyann.h>
```

### Public Member Functions

- [neuron](#) ()
- [~neuron](#) ()

### Public Attributes

- int [type](#) = 0
- double [value](#) = 0.0
- bool [visited](#) = false
- std::vector< std::pair< size\_t, double > > [in\\_nodes](#)

### 6.22.1 Constructor & Destructor Documentation

#### 6.22.1.1 neuron()

```
ann::neuron::neuron ( ) [inline]
```

### 6.22.1.2 ~neuron()

```
ann::neuron::~~neuron ( ) [inline]
```

## 6.22.2 Member Data Documentation

### 6.22.2.1 in\_nodes

```
std::vector<std::pair<size_t, double> > ann::neuron::in_nodes
```

### 6.22.2.2 type

```
int ann::neuron::type = 0
```

### 6.22.2.3 value

```
double ann::neuron::value = 0.0
```

### 6.22.2.4 visited

```
bool ann::neuron::visited = false
```

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

- /home/kamil/zpr/Monopoly/[Tinyann.h](#)

## 6.23 NotificationWall Class Reference

Represents a notification wall that displays messages.

```
#include <NotificationWall.h>
```

## Public Member Functions

- [NotificationWall](#) ()
- void [clearWall](#) ()
- std::vector< std::shared\_ptr< sf::Text > > & [getWall](#) ()
- void [addToWall](#) (std::string text)
- unsigned int [getFontSize](#) () const
- void [setFont](#) (sf::Font font)
- sf::Font & [getFont](#) ()

### 6.23.1 Detailed Description

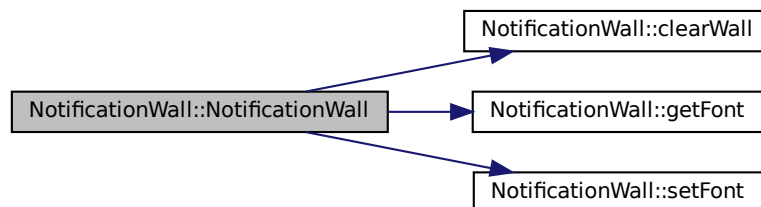
Represents a notification wall that displays messages.

### 6.23.2 Constructor & Destructor Documentation

#### 6.23.2.1 NotificationWall()

```
NotificationWall::NotificationWall ( )
```

Default constructor for the [NotificationWall](#) class. Here is the call graph for this function:



### 6.23.3 Member Function Documentation

#### 6.23.3.1 addToWall()

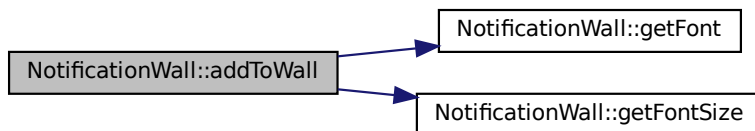
```
void NotificationWall::addToWall (
    std::string text )
```

Add a new message to the notification wall.

## Parameters

<i>text</i>	The text of the message to be added.
-------------	--------------------------------------

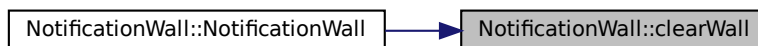
Here is the call graph for this function:



### 6.23.3.2 clearWall()

```
void NotificationWall::clearWall ( )
```

Clear all notifications from the wall. Here is the caller graph for this function:



### 6.23.3.3 getFont()

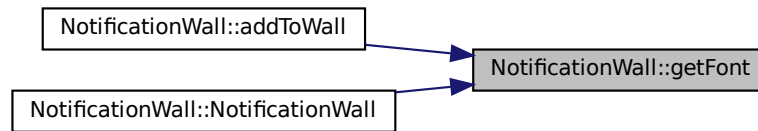
```
sf::Font & NotificationWall::getFont ( )
```

Get the font used for rendering notifications.

**Returns**

Reference to font of [Player](#) object.

Here is the caller graph for this function:

**6.23.3.4 getFontSize()**

```
unsigned int NotificationWall::getFontSize ( ) const
```

Get the font size used for rendering notifications.

**Returns**

Used font size for [Player](#) object.

Here is the caller graph for this function:

**6.23.3.5 getWall()**

```
std::vector< std::shared_ptr< sf::Text > > & NotificationWall::getWall ( )
```

Get the vector of shared pointers to `sf::Text` for notifications.

**Returns**

Reference for vector of pointer to text messages in the wall.

Here is the caller graph for this function:

**6.23.3.6 setFont()**

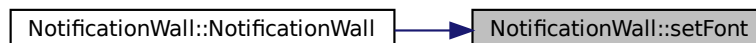
```
void NotificationWall::setFont (
    sf::Font font )
```

Set the font for rendering notifications.

**Parameters**

<i>font</i>	The font to be set.
-------------	---------------------

Here is the caller graph for this function:



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

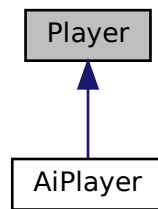
- [/home/kamil/zpr/Monopoly/NotificationWall.h](#)
- [/home/kamil/zpr/Monopoly/NotificationWall.cc](#)

**6.24 Player Class Reference**

Represents a player in a Monopoly game.

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

Inheritance diagram for Player:



## Public Member Functions

- [Player](#) ()
- [Player](#) (unsigned int money)
- unsigned int [getPosition](#) () const
- void [setPosition](#) (unsigned int new\_position)
- bool [hasFieldOwnedId](#) (unsigned int id) const
- std::vector< unsigned int > [getFieldOwnedId](#) () const
- void [addFieldOwnedId](#) (unsigned int id)
- void [removeFieldOwnedId](#) (unsigned int id)
- void [clearFieldOwnedId](#) ()
- void [setMoney](#) (unsigned int value)
- unsigned int [getMoney](#) () const
- void [addMoney](#) (unsigned int value)
- bool [subtractMoney](#) (unsigned int value)
- void [setJailStatus](#) (unsigned int new\_jail\_status)
- unsigned int [getJailStatus](#) () const
- void [setJailCards](#) (unsigned int new\_jail\_cards)
- unsigned int [getJailCards](#) () const
- void [reduceJailStatus](#) ()
- void [setId](#) (unsigned int new\_id)
- unsigned int [getId](#) () const
- void [setColor](#) (sf::Color new\_color)
- sf::Color [getColor](#) () const
- void [setIsAi](#) (bool new\_is\_ai)
- bool [getIsAi](#) () const
- void [setAiLevel](#) (unsigned int ai\_level)
- unsigned int [getAiLevel](#) () const
- void [setResultPlace](#) (unsigned int place)
- unsigned int [getResultPlace](#) () const
- void [createSprite](#) ()
- sf::Texture & [getTexture](#) ()
- sf::Sprite & [getSprite](#) ()
- float [getSpriteOffsetX](#) () const
- float [getSpriteOffsetY](#) () const
- void [setSpriteOffsetX](#) (const float offset\_x)
- void [setSpriteOffsetY](#) (const float offset\_y)



- void [setSpritePosition](#) (sf::Vector2f new\_pos)
- virtual [AiAdapter](#) & [getAdapter](#) ()
- virtual [ann::neuralnet](#) & [getNeuralNetwork](#) ()
- virtual [BuyDecision](#) [decideBuy](#) (unsigned int index)
- virtual [JailDecision](#) [decideJail](#) ()
- virtual [Decision](#) [decideMortgage](#) (unsigned int index)
- virtual [Decision](#) [decideUnmortgage](#) (unsigned int index)
- virtual unsigned int [decideAuctionBid](#) (unsigned int price)
- virtual unsigned int [decideBuildHouse](#) ()
- virtual unsigned int [decideSellHouse](#) ()
- virtual [Decision](#) [decideOfferTrade](#) ()
- virtual [Decision](#) [decideAcceptTrade](#) ()

### 6.24.1 Detailed Description

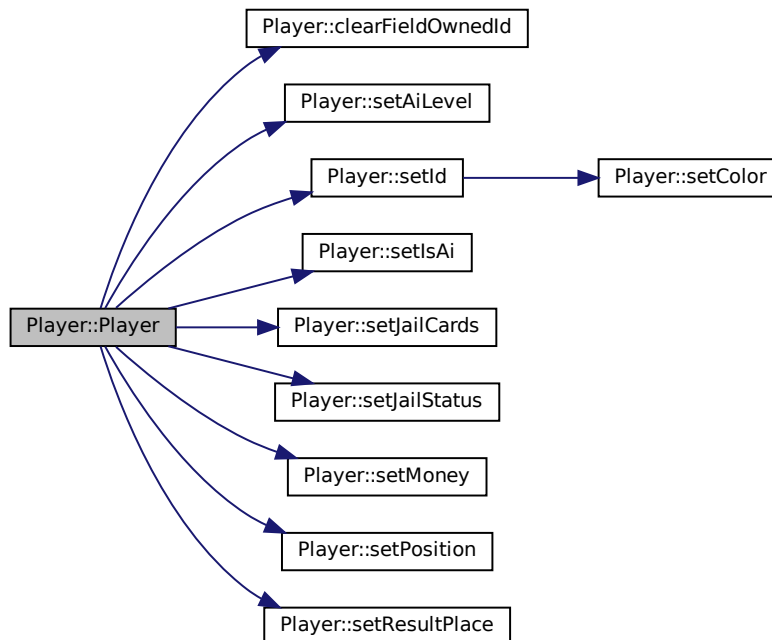
Represents a player in a Monopoly game.

### 6.24.2 Constructor & Destructor Documentation

#### 6.24.2.1 [Player\(\)](#) [1/2]

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

Default constructor for [Player](#) class. Here is the call graph for this function:



### 6.24.2.2 Player() [2/2]

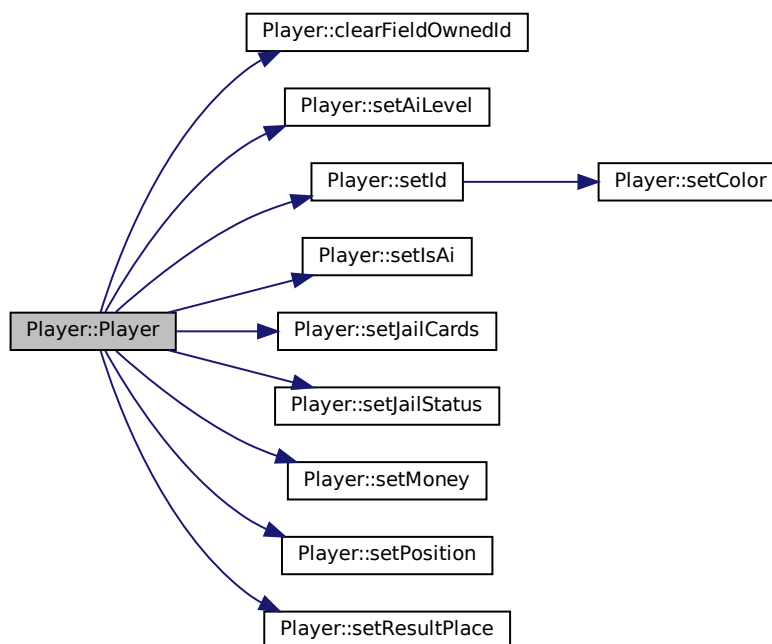
```
Player::Player (
    unsigned int money )
```

Constructor for [Player](#) class with initial money.

#### Parameters

<i>money</i>	Initial amount of money for the player.
--------------	---

Here is the call graph for this function:



## 6.24.3 Member Function Documentation

### 6.24.3.1 addFieldOwnedId()

```
void Player::addFieldOwnedId (
    unsigned int id )
```

Add a property with the specified ID to the list of properties owned by the player.

## Parameters

<i>id</i>	ID of the property to add.
-----------	----------------------------

Here is the call graph for this function:



### 6.24.3.2 addMoney()

```
void Player::addMoney (
    unsigned int value )
```

Add a specified amount of money to the player's balance.

## Parameters

<i>value</i>	Amount of money to add.
--------------	-------------------------

### 6.24.3.3 clearFieldOwnedId()

```
void Player::clearFieldOwnedId ( )
```

Clear the list of properties owned by the player. Here is the caller graph for this function:



#### 6.24.3.4 createSprite()

```
void Player::createSprite ( )
```

Create the sprite for the player.

#### 6.24.3.5 decideAcceptTrade()

```
Decision Player::decideAcceptTrade ( ) [virtual]
```

Make a decision for accepting a trade (virtual function, needs to be overridden by derived classes).

##### Returns

Decision object representing the acceptance of the trade.

Reimplemented in [AiPlayer](#).

#### 6.24.3.6 decideAuctionBid()

```
unsigned int Player::decideAuctionBid (
    unsigned int price ) [virtual]
```

Make a decision for auction bidding (virtual function, needs to be overridden by derived classes).

##### Parameters

<i>price</i>	Current price in the auction.
--------------	-------------------------------

##### Returns

The bid amount decided by the player.

Reimplemented in [AiPlayer](#).

#### 6.24.3.7 decideBuildHouse()

```
unsigned int Player::decideBuildHouse ( ) [virtual]
```

Make a decision for building a house (virtual function, needs to be overridden by derived classes).

##### Returns

The index of the property on which to build a house.

Reimplemented in [AiPlayer](#).

### 6.24.3.8 decideBuy()

```
BuyDecision Player::decideBuy (
    unsigned int index ) [virtual]
```

Make a buying decision (virtual function, needs to be overridden by derived classes).

#### Parameters

<i>index</i>	Index of the property to consider.
--------------	------------------------------------

#### Returns

BuyDecision object representing the decision.

Reimplemented in [AiPlayer](#).

### 6.24.3.9 decideJail()

```
JailDecision Player::decideJail ( ) [virtual]
```

Make a jail decision (virtual function, needs to be overridden by derived classes).

#### Returns

JailDecision object representing the decision.

Reimplemented in [AiPlayer](#).

### 6.24.3.10 decideMortgage()

```
Decision Player::decideMortgage (
    unsigned int index ) [virtual]
```

Make a mortgage decision (virtual function, needs to be overridden by derived classes).

#### Parameters

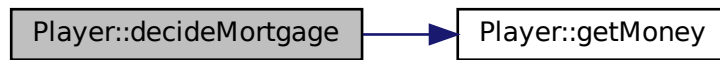
<i>index</i>	Index of the property to consider.
--------------	------------------------------------

#### Returns

Decision object representing the mortgage decision.

Reimplemented in [AiPlayer](#).

Here is the call graph for this function:



#### 6.24.3.11 decideOfferTrade()

`Decision` `Player::decideOfferTrade ( ) [virtual]`

Make a decision for offering a trade (virtual function, needs to be overridden by derived classes).

##### Returns

Decision object representing the trade offer.

Reimplemented in [AiPlayer](#).

#### 6.24.3.12 decideSellHouse()

`unsigned int` `Player::decideSellHouse ( ) [virtual]`

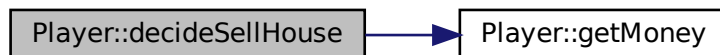
Make a decision for selling a house (virtual function, needs to be overridden by derived classes).

##### Returns

The index of the property from which to sell a house.

Reimplemented in [AiPlayer](#).

Here is the call graph for this function:



#### 6.24.3.13 decideUnmortgage()

`Decision` `Player::decideUnmortgage (`  
    `unsigned int index ) [virtual]`

Make an unmortgage decision (virtual function, needs to be overridden by derived classes).

**Parameters**

<i>index</i>	Index of the property to consider.
--------------	------------------------------------

**Returns**

Decision object representing the unmortgage decision.

Reimplemented in [AiPlayer](#).

**6.24.3.14 getAdapter()**

```
AiAdapter & Player::getAdapter ( ) [virtual]
```

Get the AI adapter (virtual function, needs to be overridden by derived classes). DO NOT USE THIS

**Returns**

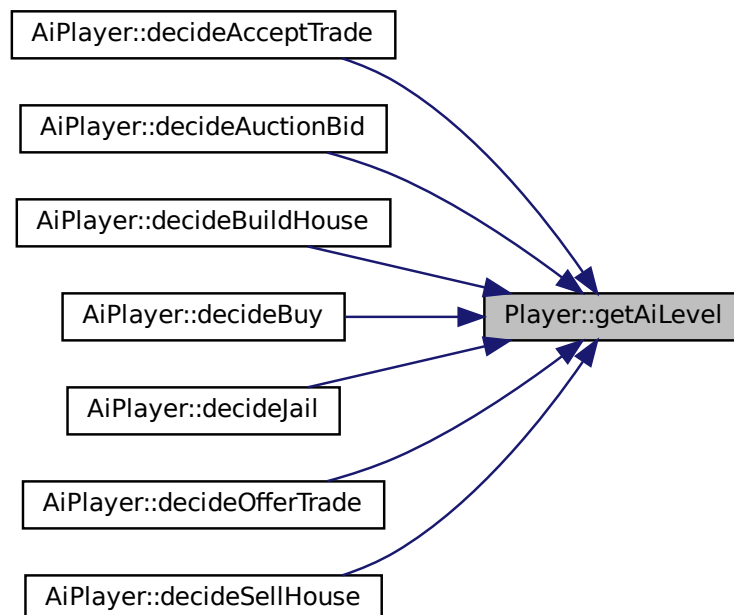
Reference to the AI adapter.

Reimplemented in [AiPlayer](#).

**6.24.3.15 getAiLevel()**

```
unsigned int Player::getAiLevel ( ) const
```

Get the AI level of the player. Here is the caller graph for this function:



#### 6.24.3.16 getColor()

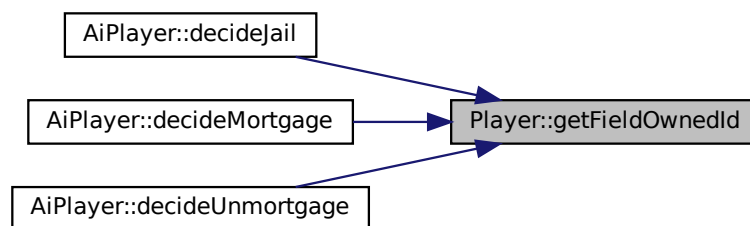
```
sf::Color Player::getColor ( ) const
```

Get the color associated with the player.

#### 6.24.3.17 getFieldOwnedId()

```
std::vector< unsigned int > Player::getFieldOwnedId ( ) const
```

Get a vector of property IDs owned by the player. Here is the caller graph for this function:



#### 6.24.3.18 getId()

```
unsigned int Player::getId ( ) const
```

Get the unique identifier of the player.

#### 6.24.3.19 getIsAi()

```
bool Player::getIsAi ( ) const
```

Get the AI status of the player.

#### 6.24.3.20 getJailCards()

```
unsigned int Player::getJailCards ( ) const
```

Get the current number of jail cards the player has.



### 6.24.3.21 `getJailStatus()`

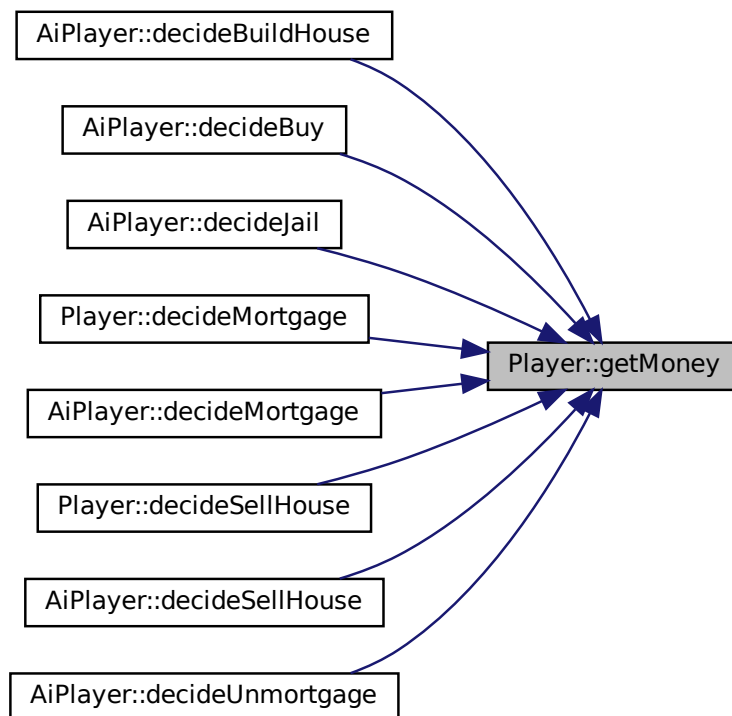
```
unsigned int Player::getJailStatus ( ) const
```

Get the current jail status of the player.

### 6.24.3.22 `getMoney()`

```
unsigned int Player::getMoney ( ) const
```

Get the current amount of money the player has. Here is the caller graph for this function:



### 6.24.3.23 `getNeuralNetwork()`

```
virtual ann::neuralnet& Player::getNeuralNetwork ( ) [inline], [virtual]
```

Get the Neural Network of a player (virtual function, needs to be overridden by derived classes).

#### Returns

Reference to the neuralnet class from a tinyai library.

Reimplemented in [AiPlayer](#).

#### 6.24.3.24 getPosition()

```
unsigned int Player::getPosition ( ) const
```

Get the current position of the player.

#### 6.24.3.25 getResultPlace()

```
unsigned int Player::getResultPlace ( ) const
```

Get the final result place of the player in the game.

#### 6.24.3.26 getSprite()

```
sf::Sprite & Player::getSprite ( )
```

Get the sprite representing the player on the game board.

#### 6.24.3.27 getSpriteOffsetX()

```
float Player::getSpriteOffsetX ( ) const
```

Get the offset of the player sprite along the X-axis.

#### 6.24.3.28 getSpriteOffsetY()

```
float Player::getSpriteOffsetY ( ) const
```

Get the offset of the player sprite along the Y-axis.

#### 6.24.3.29 getTexture()

```
sf::Texture & Player::getTexture ( )
```

Get the texture of the player's sprite.

#### 6.24.3.30 hasFieldOwnedId()

```
bool Player::hasFieldOwnedId (
    unsigned int id ) const
```

Check if the player owns a property with the specified ID.

##### Parameters

<i>id</i>	ID of the property to check.
-----------	------------------------------

**Returns**

True if the player owns the property, false otherwise.

Here is the caller graph for this function:

**6.24.3.31 reduceJailStatus()**

```
void Player::reduceJailStatus ( )
```

Reduce the jail status of the player by one.

**6.24.3.32 removeFieldOwnedId()**

```
void Player::removeFieldOwnedId (
    unsigned int id )
```

Remove a property with the specified ID from the list of properties owned by the player.

**Parameters**

<i>id</i>	ID of the property to remove.
-----------	-------------------------------

**6.24.3.33 setAiLevel()**

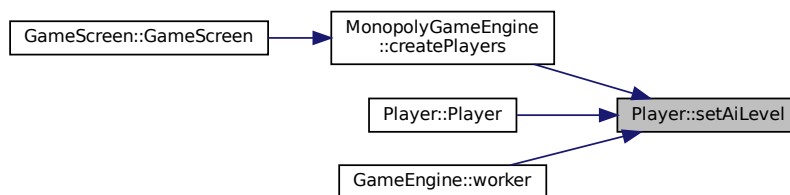
```
void Player::setAiLevel (
    unsigned int ai_level )
```

Set the AI level of the player.

**Parameters**

<i>ai_level</i>	New AI level for the player.
-----------------	------------------------------

Here is the caller graph for this function:



#### 6.24.3.34 setColor()

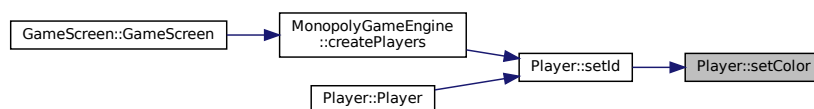
```
void Player::setColor (
    sf::Color new_color )
```

Set the color associated with the player.

##### Parameters

<i>new_color</i>	New color for the player.
------------------	---------------------------

Here is the caller graph for this function:



#### 6.24.3.35 setId()

```
void Player::setId (
    unsigned int new_id )
```

Set the unique identifier for the player.

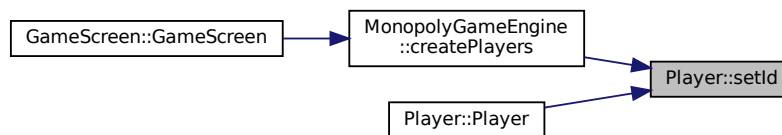
##### Parameters

<i>new_id</i>	New unique identifier for the player.
---------------	---------------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.24.3.36 setIsAi()

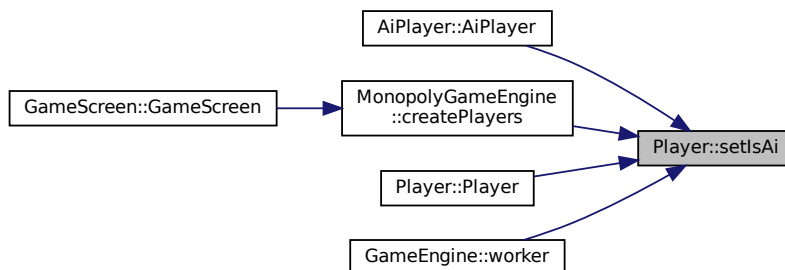
```
void Player::setIsAi (
    bool new_is_ai )
```

Set the AI status of the player.

##### Parameters

<i>new_is_ai</i>	New AI status for the player.
------------------	-------------------------------

Here is the caller graph for this function:



#### 6.24.3.37 setJailCards()

```
void Player::setJailCards (
    unsigned int new_jail_cards )
```

Set the number of jail cards the player has.

##### Parameters

<i>new_jail_cards</i>	New number of jail cards for the player.
-----------------------	--

Here is the caller graph for this function:



#### 6.24.3.38 setJailStatus()

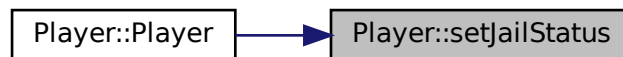
```
void Player::setJailStatus (
    unsigned int new_jail_status )
```

Set the jail status of the player.

## Parameters

<i>new_jail_status</i>	New jail status for the player.
------------------------	---------------------------------

Here is the caller graph for this function:



### 6.24.3.39 setMoney()

```
void Player::setMoney (
    unsigned int value )
```

Set the amount of money the player has.

## Parameters

<i>value</i>	New amount of money for the player.
--------------	-------------------------------------

Here is the caller graph for this function:



### 6.24.3.40 setPosition()

```
void Player::setPosition (
    unsigned int new_position )
```

Set the current position of the player.

## Parameters

<i>new_position</i>	New position for the player.
---------------------	------------------------------

Here is the caller graph for this function:



#### 6.24.3.41 setResultPlace()

```
void Player::setResultPlace (
    unsigned int place )
```

Set the final result place of the player in the game. Here is the caller graph for this function:



#### 6.24.3.42 setSpriteOffsetX()

```
void Player::setSpriteOffsetX (
    const float offset_x )
```

Set the offset of the player sprite along the X-axis.

## Parameters

<i>offset_x</i>	New offset value.
-----------------	-------------------



#### 6.24.3.43 setSpriteOffsetY()

```
void Player::setSpriteOffsetY (
    const float offset_y )
```

Set the offset of the player sprite along the Y-axis.

##### Parameters

<i>offset_y</i>	New offset value.
-----------------	-------------------

#### 6.24.3.44 setSpritePosition()

```
void Player::setSpritePosition (
    sf::Vector2f new_pos )
```

Set the position of the player sprite.

##### Parameters

<i>new_pos</i>	New position for the player sprite.
----------------	-------------------------------------

#### 6.24.3.45 subtractMoney()

```
bool Player::subtractMoney (
    unsigned int value )
```

Subtract a specified amount of money from the player's balance.

##### Parameters

<i>value</i>	Amount of money to subtract.
--------------	------------------------------

##### Returns

True if the player had enough money and the subtraction was successful, false otherwise.

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

- [/home/kamil/zpr/Monopoly/Player.h](#)
- [/home/kamil/zpr/Monopoly/Player.cc](#)

## 6.25 playerSettings Struct Reference

Struct describing player settings manipulated from game menu.

```
#include <main.h>
```

### Public Attributes

- bool [isNone](#)
- bool [isHuman](#)
- int [level](#)

### 6.25.1 Detailed Description

Struct describing player settings manipulated from game menu.

### 6.25.2 Member Data Documentation

#### 6.25.2.1 isHuman

```
bool playerSettings::isHuman
```

#### 6.25.2.2 isNone

```
bool playerSettings::isNone
```

#### 6.25.2.3 level

```
int playerSettings::level
```

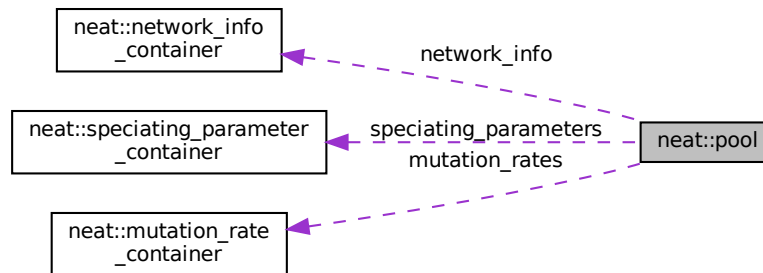
The documentation for this struct was generated from the following file:

- [/home/kamil/zpr/Monopoly/main.h](#)

## 6.26 neat::pool Class Reference

```
#include <Tinyneat.h>
```

Collaboration diagram for neat::pool:



### Public Member Functions

- [pool](#) (unsigned int input, unsigned int output, unsigned int bias=1, bool rec=false)
- void [new\\_generation](#) ()
- unsigned int [generation](#) ()
- std::vector< std::pair< [specie](#) \*, [genome](#) \* > > [get\\_genomes](#) ()
- void [import\\_fromfile](#) (std::string filename)
- void [export\\_tofile](#) (std::string filename)

### Public Attributes

- unsigned int [max\\_fitness](#) = 0
- [mutation\\_rate\\_container](#) [mutation\\_rates](#)
- [speciating\\_parameter\\_container](#) [speciating\\_parameters](#)
- [network\\_info\\_container](#) [network\\_info](#)
- std::random\_device [rd](#)
- std::mt19937 [generator](#)
- std::list< [specie](#) > [species](#)

### 6.26.1 Constructor & Destructor Documentation

#### 6.26.1.1 pool()

```

neat::pool::pool (
    unsigned int input,
    unsigned int output,
    unsigned int bias = 1,
    bool rec = false ) [inline]
  
```

## 6.26.2 Member Function Documentation

### 6.26.2.1 export\_tofile()

```
void neat::pool::export_tofile (
    std::string filename )
```

### 6.26.2.2 generation()

```
unsigned int neat::pool::generation ( ) [inline]
```

### 6.26.2.3 get\_genomes()

```
std::vector<std::pair<specie*, genome*> > neat::pool::get_genomes ( ) [inline]
```

### 6.26.2.4 import\_fromfile()

```
void neat::pool::import_fromfile (
    std::string filename )
```

Here is the call graph for this function:



### 6.26.2.5 new\_generation()

```
void neat::pool::new_generation ( )
```

## 6.26.3 Member Data Documentation

### 6.26.3.1 generator

`std::mt19937 neat::pool::generator`

### 6.26.3.2 max\_fitness

`unsigned int neat::pool::max_fitness = 0`

### 6.26.3.3 mutation\_rates

`mutation_rate_container neat::pool::mutation_rates`

### 6.26.3.4 network\_info

`network_info_container neat::pool::network_info`

### 6.26.3.5 rd

`std::random_device neat::pool::rd`

### 6.26.3.6 speciating\_parameters

`speciating_parameter_container neat::pool::speciating_parameters`

### 6.26.3.7 species

`std::list<specie> neat::pool::species`

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

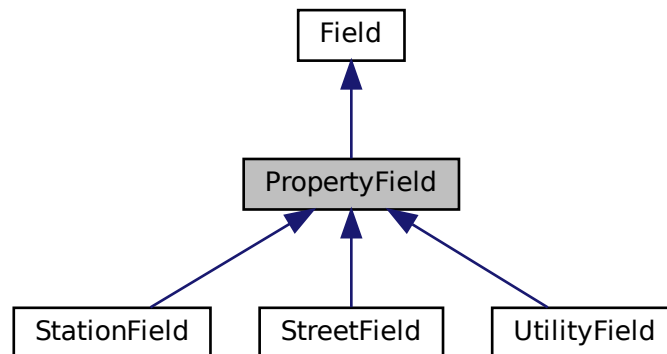
- `/home/kamil/zpr/Monopoly/Tinyneat.h`
- `/home/kamil/zpr/Monopoly/Tinyneat.cc`

## 6.27 PropertyField Class Reference

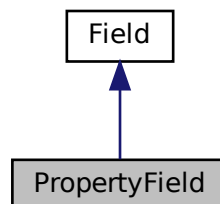
Derived class representing a property field on the game board.

```
#include <Field.h>
```

Inheritance diagram for PropertyField:



Collaboration diagram for PropertyField:



### Public Member Functions

- [PropertyField](#) (const unsigned int id, const [FieldType](#) type, const std::string name, const std::string graphic↵  
\_path, const unsigned int [width](#), const unsigned int [height](#), const float rotation, const sf::Vector2i position,  
const unsigned int price, const std::vector< unsigned int > group\_members, const unsigned int mortgage)  
*Constructor for the [PropertyField](#) class.*
- unsigned int [getPrice](#) ()  
*Gets the price of the property.*
- const std::vector< unsigned int > [getGroupMembers](#) ()  
*Gets the IDs of the group members (properties in the same group).*

- unsigned int [getMortgage](#) ()  
*Gets the mortgage value of the property.*
- bool [getIsMortgaged](#) ()  
*Checks if the property is mortgaged.*
- unsigned int [getUnmortgageValue](#) ()  
*Gets the value needed to unmortgage the property.*
- std::shared\_ptr< [Player](#) > [getOwner](#) ()  
*Gets the owner of the property.*
- sf::RectangleShape & [getOwnerFlag](#) ()  
*Gets the owner flag sprite.*
- void [setIsMortgaged](#) (bool new\_state)  
*Sets the mortgaged state of the property.*
- void [setOwner](#) (std::shared\_ptr< [Player](#) > new\_owner)  
*Sets the owner of the property.*
- void [resetOwner](#) ()  
*Resets the owner of the property.*
- void [resetDefault](#) ()  
*Resets the property to its default state.*
- void [createFlagSprite](#) ()  
*Creates the owner flag sprite.*

### 6.27.1 Detailed Description

Derived class representing a property field on the game board.

### 6.27.2 Constructor & Destructor Documentation

#### 6.27.2.1 PropertyField()

```
PropertyField::PropertyField (
    const unsigned int id,
    const FieldType type,
    const std::string name,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,
    const float rotation,
    const sf::Vector2i position,
    const unsigned int price,
    const std::vector< unsigned int > group_members,
    const unsigned int mortgage ) [inline]
```

Constructor for the [PropertyField](#) class.

#### Parameters

<i>id</i>	The ID of the field.
<i>type</i>	The type of the field.

## Parameters

<i>name</i>	The name of the field.
<i>graphic_path</i>	The file path to the field's graphic.
<i>width</i>	The width of the field.
<i>height</i>	The height of the field.
<i>rotation</i>	The rotation angle of the field.
<i>position</i>	The position of the field on the board.
<i>price</i>	The price of the property.
<i>group_members</i>	The IDs of the group members (properties in the same group).
<i>mortgage</i>	The mortgage value of the property.

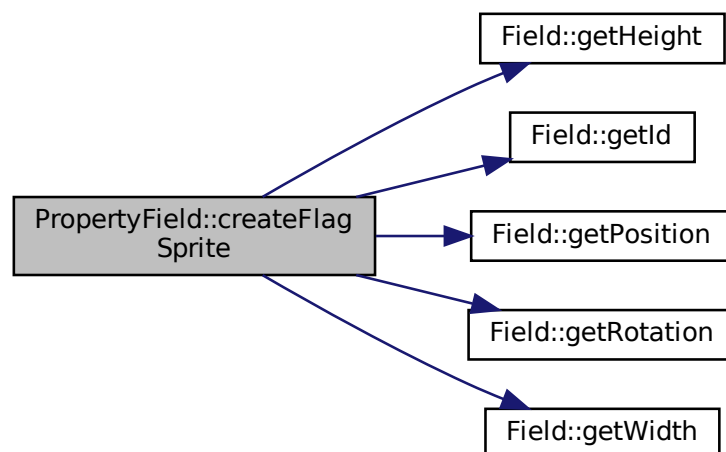
## 6.27.3 Member Function Documentation

### 6.27.3.1 createFlagSprite()

```
void PropertyField::createFlagSprite ( )
```

Creates the owner flag sprite.

Here is the call graph for this function:





Here is the caller graph for this function:



### 6.27.3.2 getGroupMembers()

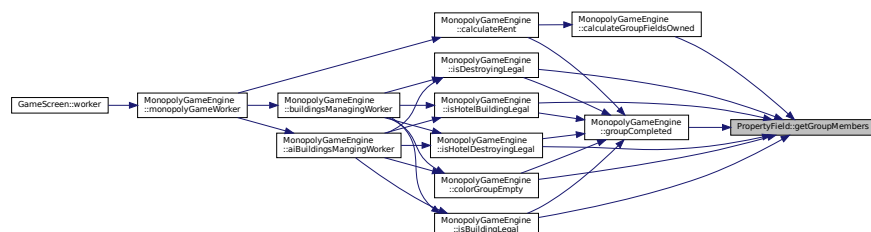
```
const std::vector< unsigned int > PropertyField::getGroupMembers ( )
```

Gets the IDs of the group members (properties in the same group).

#### Returns

The IDs of the group members.

Here is the caller graph for this function:



### 6.27.3.3 getIsMortgaged()

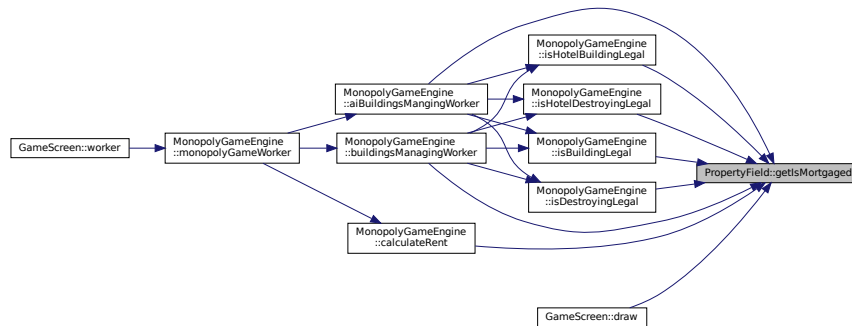
```
bool PropertyField::getIsMortgaged ( )
```

Checks if the property is mortgaged.

**Returns**

True if the property is mortgaged, false otherwise.

Here is the caller graph for this function:

**6.27.3.4 getMortgage()**

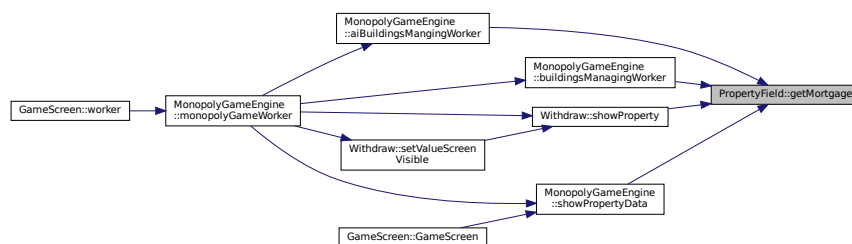
```
unsigned int PropertyField::getMortgage ( )
```

Gets the mortgage value of the property.

**Returns**

The mortgage value of the property.

Here is the caller graph for this function:



### 6.27.3.5 getOwner()

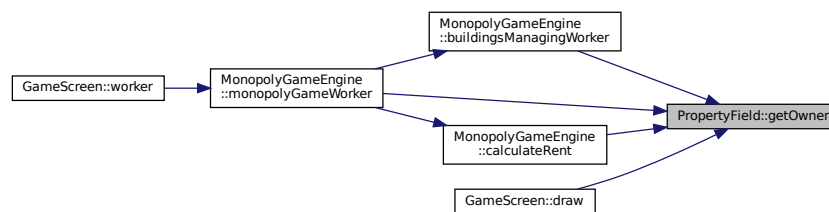
```
std::shared_ptr< Player > PropertyField::getOwner ( )
```

Gets the owner of the property.

#### Returns

A shared pointer to the owner of the property.

Here is the caller graph for this function:



### 6.27.3.6 getOwnerFlag()

```
sf::RectangleShape & PropertyField::getOwnerFlag ( )
```

Gets the owner flag sprite.

#### Returns

The owner flag sprite.

Here is the caller graph for this function:



### 6.27.3.7 getPrice()

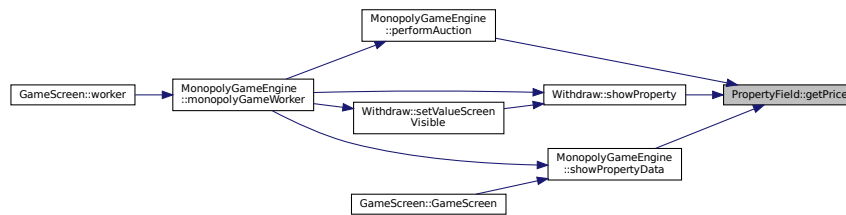
```
unsigned int PropertyField::getPrice ( )
```

Gets the price of the property.

#### Returns

The price of the property.

Here is the caller graph for this function:



### 6.27.3.8 getUnmortgageValue()

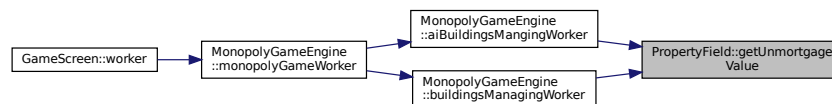
```
unsigned int PropertyField::getUnmortgageValue ( )
```

Gets the value needed to unmortgage the property.

#### Returns

The value needed to unmortgage the property.

Here is the caller graph for this function:

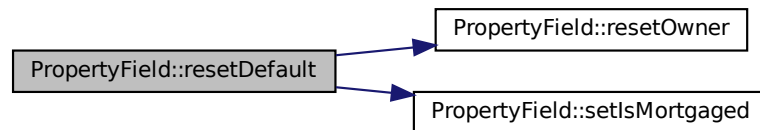


### 6.27.3.9 resetDefault()

```
void PropertyField::resetDefault ( )
```

Resets the property to its default state.

Here is the call graph for this function:

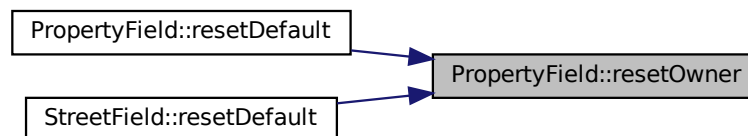


### 6.27.3.10 resetOwner()

```
void PropertyField::resetOwner ( )
```

Resets the owner of the property.

Here is the caller graph for this function:



### 6.27.3.11 setIsMortgaged()

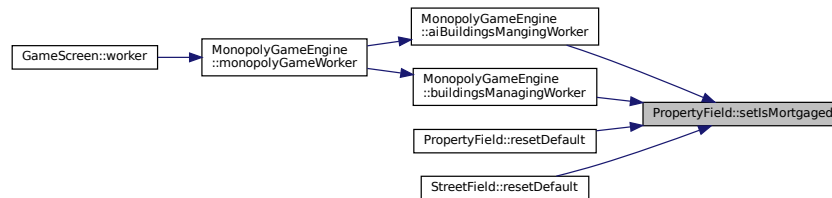
```
void PropertyField::setIsMortgaged (
    bool new_state )
```

Sets the mortgaged state of the property.

**Parameters**

<code>new_state</code>	The new mortgaged state.
------------------------	--------------------------

Here is the caller graph for this function:

**6.27.3.12 setOwner()**

```
void PropertyField::setOwner (
    std::shared_ptr< Player > new_owner )
```

Sets the owner of the property.

**Parameters**

<code>new_owner</code>	A shared pointer to the new owner.
------------------------	------------------------------------

Here is the caller graph for this function:



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

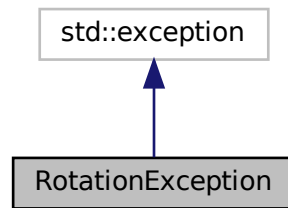
- [/home/kamil/zpr/Monopoly/Field.h](#)
- [/home/kamil/zpr/Monopoly/Field.cc](#)

**6.28 RotationException Class Reference**

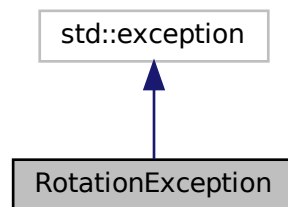
Exception for handling passing wrong rotation to any displayed object.

```
#include <main.h>
```

Inheritance diagram for RotationException:



Collaboration diagram for RotationException:



## Public Member Functions

- [RotationException](#) (float rotation)
- [RotationException](#) (const [RotationException](#) &e) throw ()
- float [getBadRotation](#) ()

### 6.28.1 Detailed Description

Exception for handling passing wrong rotation to any displayed object.

### 6.28.2 Constructor & Destructor Documentation

#### 6.28.2.1 RotationException() [1/2]

```
RotationException::RotationException (  
    float rotation ) [inline]
```

### 6.28.2.2 RotationException() [2/2]

```
RotationException::RotationException (
    const RotationException & e ) throw ( )    [inline]
```

## 6.28.3 Member Function Documentation

### 6.28.3.1 getBadRotation()

```
float RotationException::getBadRotation ( )    [inline]
```

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

- [/home/kamil/zpr/Monopoly/main.h](#)

## 6.29 neat::speciating\_parameter\_container Struct Reference

```
#include <Tinyneat.h>
```

### Public Member Functions

- void [read](#) (std::ifstream &o)
- void [write](#) (std::ofstream &o, std::string prefix)

### Public Attributes

- unsigned int [population](#) = 240
- double [delta\\_disjoint](#) = 2.0
- double [delta\\_weights](#) = 0.4
- double [delta\\_threshold](#) = 1.3
- unsigned int [stale\\_species](#) = 15

## 6.29.1 Member Function Documentation

### 6.29.1.1 read()

```
void neat::speciating_parameter_container::read (
    std::ifstream & o )
```



### 6.29.1.2 write()

```
void neat::speciating_parameter_container::write (
    std::ostream & o,
    std::string prefix )
```

## 6.29.2 Member Data Documentation

### 6.29.2.1 delta\_disjoint

```
double neat::speciating_parameter_container::delta_disjoint = 2.0
```

### 6.29.2.2 delta\_threshold

```
double neat::speciating_parameter_container::delta_threshold = 1.3
```

### 6.29.2.3 delta\_weights

```
double neat::speciating_parameter_container::delta_weights = 0.4
```

### 6.29.2.4 population

```
unsigned int neat::speciating_parameter_container::population = 240
```

### 6.29.2.5 stale\_species

```
unsigned int neat::speciating_parameter_container::stale_species = 15
```

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

- /home/kamil/zpr/Monopoly/[Tinyneat.h](#)
- /home/kamil/zpr/Monopoly/[Tinyneat.cc](#)

## 6.30 neat::specie Struct Reference

```
#include <Tinyneat.h>
```

### Public Attributes

- unsigned int [top\\_fitness](#) = 0
- unsigned int [average\\_fitness](#) = 0
- unsigned int [staleness](#) = 0
- std::vector<[genome](#)> [genomes](#)

### 6.30.1 Member Data Documentation

#### 6.30.1.1 average\_fitness

```
unsigned int neat::specie::average_fitness = 0
```

#### 6.30.1.2 genomes

```
std::vector<genome> neat::specie::genomes
```

#### 6.30.1.3 staleness

```
unsigned int neat::specie::staleness = 0
```

#### 6.30.1.4 top\_fitness

```
unsigned int neat::specie::top_fitness = 0
```

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

- [/home/kamil/zpr/Monopoly/Tinyneat.h](#)

## 6.31 SpriteOffsetException Class Reference

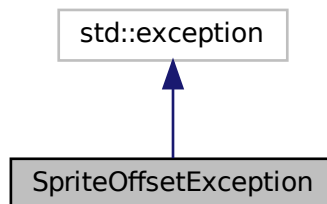
Exception for handling wrong offset to any displayed object.

```
#include <main.h>
```

Inheritance diagram for SpriteOffsetException:



Collaboration diagram for SpriteOffsetException:



### Public Member Functions

- [SpriteOffsetException](#) (float offset)
- [SpriteOffsetException](#) (const [SpriteOffsetException](#) &e) throw ()
- float [getBadOffset](#) ()

#### 6.31.1 Detailed Description

Exception for handling wrong offset to any displayed object.

#### 6.31.2 Constructor & Destructor Documentation

### 6.31.2.1 SpriteOffsetException() [1/2]

```
SpriteOffsetException::SpriteOffsetException (
    float offset ) [inline]
```

### 6.31.2.2 SpriteOffsetException() [2/2]

```
SpriteOffsetException::SpriteOffsetException (
    const SpriteOffsetException & e ) throw ( ) [inline]
```

## 6.31.3 Member Function Documentation

### 6.31.3.1 getBadOffset()

```
float SpriteOffsetException::getBadOffset ( ) [inline]
```

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

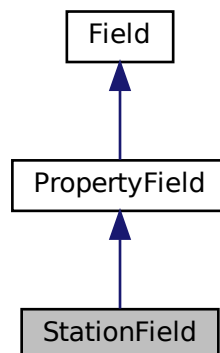
- /home/kamil/zpr/Monopoly/[main.h](#)

## 6.32 StationField Class Reference

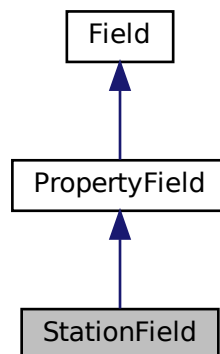
Derived class representing a station field on the game board.

```
#include <Field.h>
```

Inheritance diagram for StationField:



Collaboration diagram for StationField:



## Public Member Functions

- [StationField](#) (const unsigned int id, const [FieldType](#) type, const std::string name, const std::string graphic\_path, const unsigned int width, const unsigned int height, const float rotation, const sf::Vector2i position, const unsigned int price, const std::map< [StationTiers](#), unsigned int > rent\_values, const std::vector< unsigned int > group\_members, const unsigned int Mortgage)  
*Constructor for the [StationField](#) class.*
- const std::map< [StationTiers](#), unsigned int > [getRentValues](#) ()  
*Gets the rent values for different tiers.*
- unsigned int [calculateRent](#) ()  
*Calculates the rent for the property.*

### 6.32.1 Detailed Description

Derived class representing a station field on the game board.

### 6.32.2 Constructor & Destructor Documentation

#### 6.32.2.1 StationField()

```

StationField::StationField (
    const unsigned int id,
    const FieldType type,
    const std::string name,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,

```

```

const float rotation,
const sf::Vector2i position,
const unsigned int price,
const std::map< StationTiers, unsigned int > rent_values,
const std::vector< unsigned int > group_members,
const unsigned int Mortgage ) [inline]

```

Constructor for the [StationField](#) class.

#### Parameters

<i>id</i>	The ID of the field.
<i>type</i>	The type of the field.
<i>name</i>	The name of the field.
<i>graphic_path</i>	The file path to the field's graphic.
<i>width</i>	The width of the field.
<i>height</i>	The height of the field.
<i>rotation</i>	The rotation angle of the field.
<i>position</i>	The position of the field on the board.
<i>price</i>	The price of the property.
<i>rent_values</i>	The rent values for different tiers.
<i>group_members</i>	The IDs of the group members (properties in the same group).
<i>mortgage</i>	The mortgage value of the property.

## 6.32.3 Member Function Documentation

### 6.32.3.1 calculateRent()

```
unsigned int StationField::calculateRent ( )
```

Calculates the rent for the property.

#### Returns

The calculated rent value.

### 6.32.3.2 getRentValues()

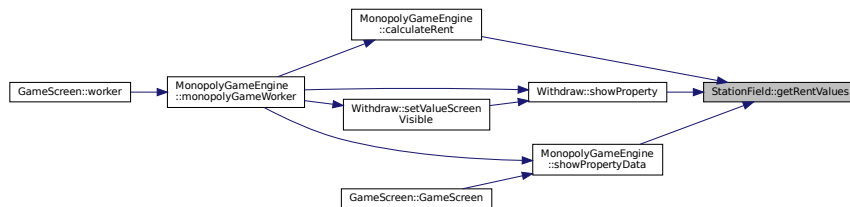
```
const std::map< StationTiers, unsigned int > StationField::getRentValues ( )
```

Gets the rent values for different tiers.

**Returns**

The rent values for different tiers.

Here is the caller graph for this function:



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

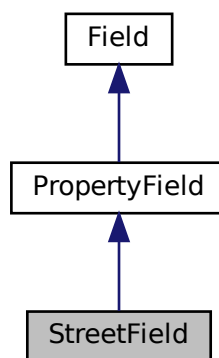
- `/home/kamil/zpr/Monopoly/Field.h`
- `/home/kamil/zpr/Monopoly/Field.cc`

## 6.33 StreetField Class Reference

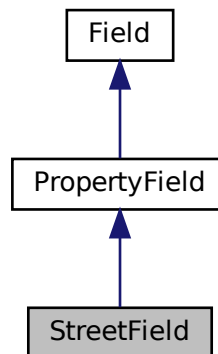
Derived class representing a street field on the game board.

```
#include <Field.h>
```

Inheritance diagram for StreetField:



Collaboration diagram for StreetField:



## Public Member Functions

- [StreetField](#) (const unsigned int id, const [FieldType](#) type, const std::string name, const std::string graphic↵  
\_path, const unsigned int [width](#), const unsigned int [height](#), const float rotation, const sf::Vector2i position,  
const unsigned int price, const unsigned int house\_price, const unsigned int hotel\_price, const std::map<  
[StreetTiers](#), unsigned int > rent\_values, const std::vector< unsigned int > group\_members, const unsigned  
int Mortgage)  
*Constructor for the [StreetField](#) class.*
- const std::map< [StreetTiers](#), unsigned int > [getRentValues](#) ()  
*Gets the rent values for different tiers.*
- unsigned int [getHousePrice](#) ()  
*Gets the price of a house.*
- unsigned int [getHotelPrice](#) ()  
*Gets the price of a hotel.*
- unsigned int [getHouseNumber](#) ()  
*Gets the number of houses on the property.*
- bool [getIsHotel](#) ()  
*Checks if there is a hotel on the property.*
- void [setHouseNumber](#) (unsigned int new\_house\_number)  
*Sets the number of houses on the property.*
- void [setIsHotel](#) (bool new\_state)  
*Sets the hotel state on the property.*
- void [resetDefault](#) ()  
*Resets the property to its default state.*
- unsigned int [calculateRent](#) ()  
*Calculates the rent for the property.*

### 6.33.1 Detailed Description

Derived class representing a street field on the game board.



## 6.33.2 Constructor & Destructor Documentation

### 6.33.2.1 StreetField()

```
StreetField::StreetField (
    const unsigned int id,
    const FieldType type,
    const std::string name,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,
    const float rotation,
    const sf::Vector2i position,
    const unsigned int price,
    const unsigned int house_price,
    const unsigned int hotel_price,
    const std::map< StreetTiers, unsigned int > rent_values,
    const std::vector< unsigned int > group_members,
    const unsigned int Mortgage ) [inline]
```

Constructor for the [StreetField](#) class.

#### Parameters

<i>id</i>	The ID of the field.
<i>type</i>	The type of the field.
<i>name</i>	The name of the field.
<i>graphic_path</i>	The file path to the field's graphic.
<i>width</i>	The width of the field.
<i>height</i>	The height of the field.
<i>rotation</i>	The rotation angle of the field.
<i>position</i>	The position of the field on the board.
<i>price</i>	The price of the property.
<i>house_price</i>	The price of a house.
<i>hotel_price</i>	The price of a hotel.
<i>rent_values</i>	The rent values for different tiers.
<i>group_members</i>	The IDs of the group members (properties in the same group).
<i>mortgage</i>	The mortgage value of the property.

## 6.33.3 Member Function Documentation

### 6.33.3.1 calculateRent()

```
unsigned int StreetField::calculateRent ( )
```

Calculates the rent for the property.





### 6.33.3.6 getRentValues()

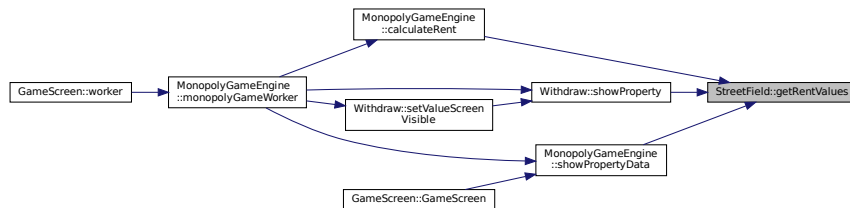
```
const std::map< StreetTiers, unsigned int > StreetField::getRentValues ( )
```

Gets the rent values for different tiers.

Returns

The rent values for different tiers.

Here is the caller graph for this function:

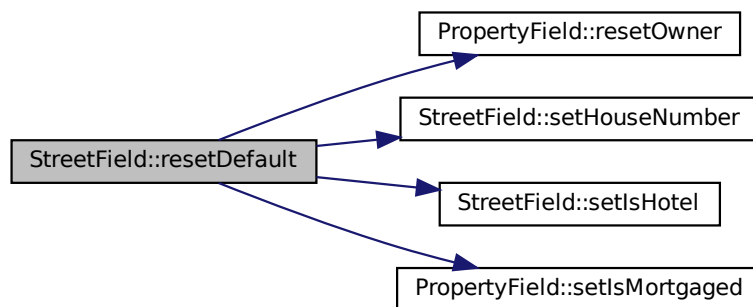


### 6.33.3.7 resetDefault()

```
void StreetField::resetDefault ( )
```

Resets the property to its default state.

Here is the call graph for this function:



### 6.33.3.8 setHouseNumber()

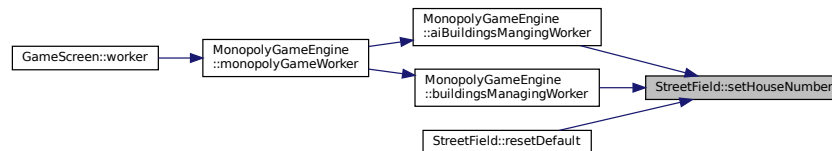
```
void StreetField::setHouseNumber (
    unsigned int new_house_number )
```

Sets the number of houses on the property.

## Parameters

<code>new_house_number</code>	The new number of houses on the property.
-------------------------------	---

Here is the caller graph for this function:



## 6.33.3.9 setIsHotel()

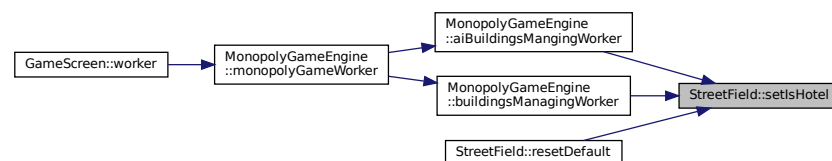
```
void StreetField::setIsHotel (
    bool new_state )
```

Sets the hotel state on the property.

## Parameters

<code>new_state</code>	The new hotel state.
------------------------	----------------------

Here is the caller graph for this function:



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

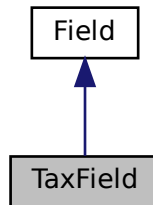
- `/home/kamil/zpr/Monopoly/Field.h`
- `/home/kamil/zpr/Monopoly/Field.cc`

## 6.34 TaxField Class Reference

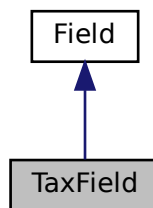
Derived class representing a tax field on the monopoly board.

```
#include <Field.h>
```

Inheritance diagram for TaxField:



Collaboration diagram for TaxField:



## Public Member Functions

- [TaxField](#) (const unsigned int id, const [FieldType](#) type, const std::string name, const std::string graphic\_path, const unsigned int [width](#), const unsigned int [height](#), const float rotation, const sf::Vector2i position, const unsigned int tax\_value)  
*Constructor for the [TaxField](#) class.*
- unsigned int [getTaxValue](#) ()  
*Calculates the tax for Tax property.*

### 6.34.1 Detailed Description

Derived class representing a tax field on the monopoly board.

### 6.34.2 Constructor & Destructor Documentation

### 6.34.2.1 TaxField()

```
TaxField::TaxField (
    const unsigned int id,
    const FieldType type,
    const std::string name,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,
    const float rotation,
    const sf::Vector2i position,
    const unsigned int tax_value ) [inline]
```

Constructor for the [TaxField](#) class.

#### Parameters

<i>id</i>	The unique identifier of the field.
<i>type</i>	The type of the field.
<i>name</i>	The name of the field.
<i>graphic_path</i>	The file path to the graphic representation of the field.
<i>width</i>	The width of the field.
<i>height</i>	The height of the field.
<i>rotation</i>	The rotation angle of the field.
<i>position</i>	The position of the field on the game board.
<i>tax_value</i>	The value of the tax.

## 6.34.3 Member Function Documentation

### 6.34.3.1 getTaxValue()

```
unsigned int TaxField::getTaxValue ( )
```

Calculates the tax for Tax property.

#### Returns

The calculated tax value.

Here is the caller graph for this function:



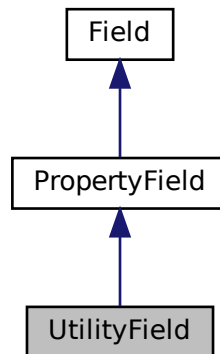
The documentation for this class was generated from the following files:

- [/home/kamil/zpr/Monopoly/Field.h](#)
- [/home/kamil/zpr/Monopoly/Field.cc](#)

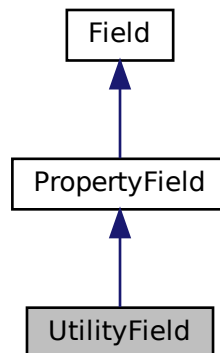
## 6.35 UtilityField Class Reference

```
#include <Field.h>
```

Inheritance diagram for UtilityField:



Collaboration diagram for UtilityField:



### Public Member Functions

- **UtilityField** (const unsigned int id, const [FieldType](#) type, const std::string name, const std::string graphic\_↵ path, const unsigned int [width](#), const unsigned int [height](#), const float rotation, const sf::Vector2i position, const unsigned int price, const std::map< [UtilityTiers](#), unsigned int > rent\_multipliers, const std::vector< unsigned int > group\_members, const unsigned int Mortgage)  
*Constructor for the [UtilityField](#) class.*
- const std::map< [UtilityTiers](#), unsigned int > [getRentMultipliers](#) ()  
*Calculates the rent for the utility property.*
- unsigned int [calculateRent](#) (unsigned int dice\_roll)  
*Calculates the rent for the utility property.*



## 6.35.1 Constructor & Destructor Documentation

### 6.35.1.1 UtilityField()

```
UtilityField::UtilityField (
    const unsigned int id,
    const FieldType type,
    const std::string name,
    const std::string graphic_path,
    const unsigned int width,
    const unsigned int height,
    const float rotation,
    const sf::Vector2i position,
    const unsigned int price,
    const std::map< UtilityTiers, unsigned int > rent_multipliers,
    const std::vector< unsigned int > group_members,
    const unsigned int Mortgage ) [inline]
```

Constructor for the [UtilityField](#) class.

#### Parameters

<i>id</i>	The ID of the field.
<i>type</i>	The type of the field.
<i>name</i>	The name of the field.
<i>graphic_path</i>	The file path to the field's graphic.
<i>width</i>	The width of the field.
<i>height</i>	The height of the field.
<i>rotation</i>	The rotation angle of the field.
<i>position</i>	The position of the field on the board.
<i>price</i>	The price of the property.
<i>rent_multipliers</i>	The rent multipliers for different tiers.
<i>group_members</i>	The IDs of the group members (properties in the same group).
<i>mortgage</i>	The mortgage value of the property.

## 6.35.2 Member Function Documentation

### 6.35.2.1 calculateRent()

```
unsigned int UtilityField::calculateRent (
    unsigned int dice_roll )
```

Calculates the rent for the utility property.

## Parameters

<code>dice_roll</code>	The roll of the dice.
------------------------	-----------------------

## Returns

The calculated rent value.

### 6.35.2.2 getRentMultipliers()

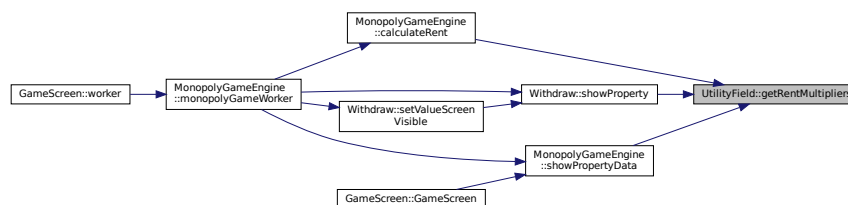
```
const std::map< UtilityTiers, unsigned int > UtilityField::getRentMultipliers ( )
```

Calculates the rent for the utility property.

## Returns

Map for Utility tiers mapped to mulitpliers values

Here is the caller graph for this function:



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

- [/home/kamil/zpr/Monopoly/Field.h](#)
- [/home/kamil/zpr/Monopoly/Field.cc](#)

## 6.36 Withdraw Class Reference

Represents the trade and withdraw mechanism in a monopoly game.

```
#include <Withdraw.h>
```

## Public Member Functions

- [Withdraw](#) ()  
*Constructor for the [Withdraw](#) class.*
- `std::vector< unsigned int > getPlayer1IndexProperties ()`  
*Getter for the properties of [Player 1](#) offer.*
- `std::vector< unsigned int > getPlayer2IndexProperties ()`  
*Getter for the properties of [Player 2](#) offer.*
- `void setPlayer1IndexProperties (std::vector< unsigned int > new_index_properties)`  
*Setter for the properties of [Player 1](#) offer.*
- `void setPlayer2IndexProperties (std::vector< unsigned int > new_index_properties)`  
*Setter for the properties of [Player 2](#) offer.*
- `void setBoard (std::shared_ptr< Board > board_ptr)`  
*Setter for the gameboard.*
- `void setTurnState (TurnState state)`  
*Setter for the turn state of monopolygame when withdraw was started.*
- `TurnState getTurnState ()`  
*Getter for the turn state.*
- `sf::Font & getFont ()`  
*Getter for the font used in rendering text.*
- `unsigned int getFontSize () const`  
*Getter for the font size used in rendering text.*
- `void setFont (sf::Font font)`  
*Setter for the font used in rendering text.*
- `std::vector< std::shared_ptr< Button > > & getButtons ()`  
*Getter for the vector of buttons used in the withdrawal process.*
- `std::vector< std::shared_ptr< sf::Text > > & getTexts ()`  
*Getter for the vector of text objects used in the withdrawal process.*
- `std::shared_ptr< Button > getPlayer1Button ()`  
*Getter for button for choosing [Player 1](#) as passive in withdraw.*
- `std::shared_ptr< Button > getPlayer2Button ()`  
*Getter for button for choosing [Player 2](#) as passive in withdraw.*
- `std::shared_ptr< Button > getPlayer3Button ()`  
*Getter for button for choosing [Player 3](#) as passive in withdraw.*
- `std::shared_ptr< Button > getPlayer4Button ()`  
*Getter for button for choosing [Player 4](#) as passive in withdraw.*
- `std::shared_ptr< Button > getResignButton ()`  
*Getter for the resign button.*
- `void createChoosePlayerScreen ()`  
*Creates the screen for choosing players during withdrawal.*
- `void createValuePlayerScreen ()`  
*Creates the screen for specifying values during withdrawal.*
- `void createDecisionPlayerScreen ()`  
*Creates the screen for making decisions during withdrawal.*
- `void setChooseScreenVisible (bool is_visible)`  
*Sets the visibility of the choose player screen.*
- `void setValueScreenVisible (bool is_visible)`  
*Sets the visibility of the value player screen.*
- `void setDecisionScreenVisible (bool is_visible)`  
*Sets the visibility of the decision player screen.*
- `void setPlayer1ToWithdraw (std::shared_ptr< Player > player_ptr)`

- Sets Active [Player](#) for withdrawal.*

  - void [setPlayer2ToWithdraw](#) (std::shared\_ptr< [Player](#) > player\_ptr)
- Sets Passive [Player](#) for withdrawal.*

  - std::shared\_ptr< [Player](#) > [getPlayer1ToWithdraw](#) ()

*Getter for Active [Player](#) object.*

  - std::shared\_ptr< [Player](#) > [getPlayer2ToWithdraw](#) ()

*Getter for Passive [Player](#) object.*

  - std::shared\_ptr< [Button](#) > [getResignValueButton](#) ()

*Getter for the resign button from value screen.*

  - std::shared\_ptr< [Button](#) > [getSubmitValueButton](#) ()

*Getter for the submit value button from value screen.*

  - std::shared\_ptr< [Button](#) > [getPlayer1minus1](#) ()

*Getter for Active [Player](#) minus 1 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1minus10](#) ()

*Getter for Active [Player](#) minus 10 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1minus100](#) ()

*Getter for Active [Player](#) minus 100 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1plus1](#) ()

*Getter for Active [Player](#) plus 1 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1plus10](#) ()

*Getter for Active [Player](#) plus 10 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1plus100](#) ()

*Getter for Active [Player](#) plus 100 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2minus1](#) ()

*Getter for Passive [Player](#) minus 1 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2minus10](#) ()

*Getter for Passive [Player](#) minus 10 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2minus100](#) ()

*Getter for Passive [Player](#) minus 100 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2plus1](#) ()

*Getter for Passive [Player](#) plus 1 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2plus10](#) ()

*Getter for Passive [Player](#) plus 10 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2plus100](#) ()

*Getter for Passive [Player](#) plus 100 money button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1NextButton](#) ()

*Getter for [Player](#) 1 next property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1PreviousButton](#) ()

*Getter for [Player](#) 1 previous property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2NextButton](#) ()

*Getter for [Player](#) 2 next property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2PreviousButton](#) ()

*Getter for [Player](#) 2 previous property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1IndexNextButton](#) ()

*Getter for [Player](#) 1 offer next property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer1IndexPreviousButton](#) ()

*Getter for [Player](#) 1 index previous property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2IndexNextButton](#) ()

*Getter for [Player](#) 2 index next property button.*

  - std::shared\_ptr< [Button](#) > [getPlayer2IndexPreviousButton](#) ()

*Getter for [Player](#) 2 index previous property button.*

- `std::shared_ptr< Button > getPlayer1AddButton ()`  
*Getter for [Player 1](#) add to offer property button.*
- `std::shared_ptr< Button > getPlayer1RemoveButton ()`  
*Getter for [Player 1](#) remove from offer property button.*
- `std::shared_ptr< Button > getPlayer2AddButton ()`  
*Getter for [Player 2](#) add to offer property button.*
- `std::shared_ptr< Button > getPlayer2RemoveButton ()`  
*Getter for [Player 2](#) remove from offer property button.*
- `std::shared_ptr< Button > getResignDecisionButton ()`  
*Getter for the resign decision button.*
- `std::shared_ptr< Button > getAcceptDecisionButton ()`  
*Getter for the accept decision button.*
- `void makeWithdraw ()`  
*Initiates the withdrawal process (exchange of money and properties).*
- `void moneyTransferIndex (unsigned int player_num, int money)`  
*Transfers money during the withdrawal process.*
- `void moneyTextUpdate ()`  
*Updates the displayed money text during the withdrawal process.*
- `void showProperty (int column)`  
*Shows the property details for the specified column during the withdrawal process.*
- `sf::Sprite & getSpritePropertyPlayer1 ()`  
*Getter for [Player 1](#)'s property sprite.*
- `std::vector< std::shared_ptr< sf::Text > > & getTexturesPropertyPlayer1 ()`  
*Getter for the text objects representing [Player 1](#)'s property details.*
- `sf::Sprite & getSpritePropertyPlayer1Index ()`  
*Getter for [Player 1](#)'s offered property sprite.*
- `std::vector< std::shared_ptr< sf::Text > > & getTexturesPropertyPlayer1Index ()`  
*Getter for the text objects representing [Player 1](#)'s offered property details.*
- `sf::Sprite & getSpritePropertyPlayer2 ()`  
*Getter for [Player 2](#)'s property sprite.*
- `std::vector< std::shared_ptr< sf::Text > > & getTexturesPropertyPlayer2 ()`  
*Getter for the text objects representing [Player 2](#)'s property details.*
- `sf::Sprite & getSpritePropertyPlayer2Index ()`  
*Getter for [Player 2](#)'s offered property sprite.*
- `std::vector< std::shared_ptr< sf::Text > > & getTexturesPropertyPlayer2Index ()`  
*Getter for the text objects representing [Player 2](#)'s offered property details.*
- `std::shared_ptr< sf::Texture > getTexturePropertyPlayer1 ()`  
*Getter for [Player 1](#)'s property texture.*
- `std::shared_ptr< sf::Texture > getTexturePropertyPlayer1Index ()`  
*Getter for [Player 1](#)'s offer property texture.*
- `std::shared_ptr< sf::Texture > getTexturePropertyPlayer2 ()`  
*Getter for [Player 2](#)'s property texture.*
- `std::shared_ptr< sf::Texture > getTexturePropertyPlayer2Index ()`  
*Getter for [Player 2](#)'s offer property texture.*
- `void addPropertyPlayerShowed (int i, unsigned int col)`  
*Changes showed property in certain column.*
- `void propertyPlayerMoveIndex (int dir, unsigned int plr_num)`  
*Moves properties between players ownership and withdraw offer.*
- `bool isNonZeroValue ()`  
*Check whenever withdraw is legal to be done.*

### 6.36.1 Detailed Description

Represents the trade and withdraw mechanism in a monopoly game.

The [Withdraw](#) class handles the trade and withdraw functionality between players in a monopoly game. It includes features such as choosing players, specifying values, and making decisions during the withdrawal process.

### 6.36.2 Constructor & Destructor Documentation

#### 6.36.2.1 Withdraw()

```
Withdraw::Withdraw ( )
```

Constructor for the [Withdraw](#) class.

### 6.36.3 Member Function Documentation

#### 6.36.3.1 addPropertyPlayerShowed()

```
void Withdraw::addPropertyPlayerShowed (
    int i,
    unsigned int col )
```

Changes showed property in certain column.

##### Parameters

<i>i</i>	if $i > 0$ moves right in properties vector, if $i < 0$ moves left
<i>col</i>	(1-4) defines properties column

Here is the caller graph for this function:

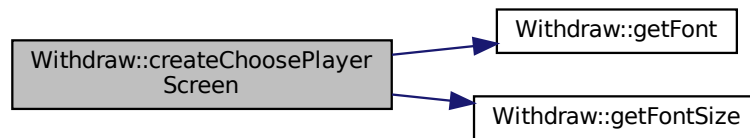


### 6.36.3.2 createChoosePlayerScreen()

```
void Withdraw::createChoosePlayerScreen ( )
```

Creates the screen for choosing players during withdrawal.

Here is the call graph for this function:



Here is the caller graph for this function:

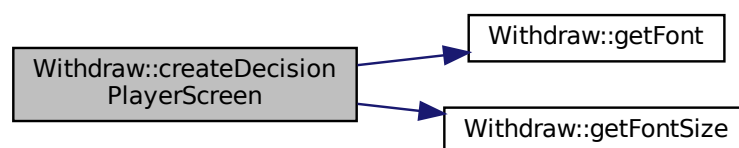


### 6.36.3.3 createDecisionPlayerScreen()

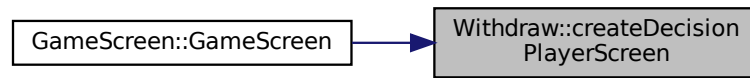
```
void Withdraw::createDecisionPlayerScreen ( )
```

Creates the screen for making decisions during withdrawal.

Here is the call graph for this function:



Here is the caller graph for this function:

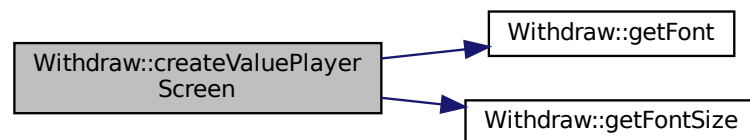


#### 6.36.3.4 createValuePlayerScreen()

```
void Withdraw::createValuePlayerScreen ( )
```

Creates the screen for specifying values during withdrawal.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.36.3.5 getAcceptDecisionButton()

```
std::shared_ptr< Button > Withdraw::getAcceptDecisionButton ( )
```

Getter for the accept decision button.

##### Returns

A shared pointer to the accept decision `Button` object.



### 6.36.3.6 getButtons()

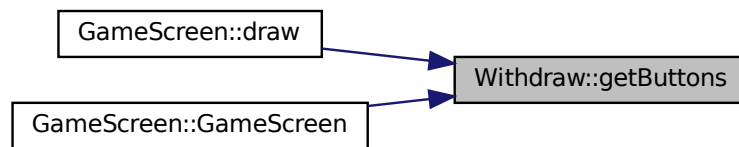
```
std::vector< std::shared_ptr< Button > > & Withdraw::getButtons ( )
```

Getter for the vector of buttons used in the withdrawal process.

#### Returns

A vector of shared pointers to [Button](#) objects.

Here is the caller graph for this function:



### 6.36.3.7 getFont()

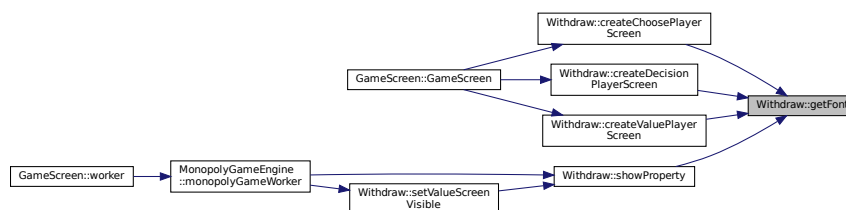
```
sf::Font & Withdraw::getFont ( )
```

Getter for the font used in rendering text.

#### Returns

A reference to the SFML Font object.

Here is the caller graph for this function:



### 6.36.3.8 getFontSize()

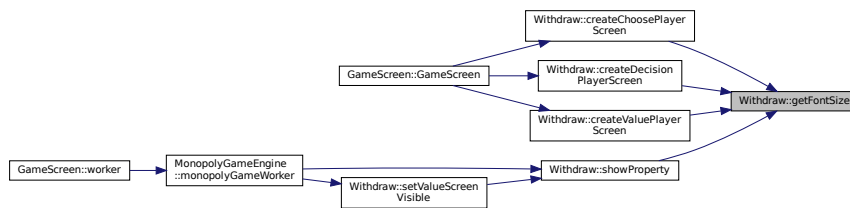
```
unsigned int Withdraw::getFontSize ( ) const
```

Getter for the font size used in rendering text.

#### Returns

An unsigned integer representing the font size.

Here is the caller graph for this function:



### 6.36.3.9 getPlayer1AddButton()

```
std::shared_ptr< Button > Withdraw::getPlayer1AddButton ( )
```

Getter for [Player](#) 1 add to offer property button.

#### Returns

A shared pointer to the add property [Button](#) object for [Player](#) 1 offer.

### 6.36.3.10 getPlayer1Button()

```
std::shared_ptr< Button > Withdraw::getPlayer1Button ( )
```

Getter for button for choosing [Player](#) 1 as passive in withdraw.

#### Returns

A shared pointer to choose [Player](#) 1 [Button](#) object.

#### 6.36.3.11 getPlayer1IndexNextButton()

```
std::shared_ptr< Button > Withdraw::getPlayer1IndexNextButton ( )
```

Getter for [Player](#) 1 offer next property button.

##### Returns

A shared pointer to the next property [Button](#) object for [Player](#) 1 offer.

#### 6.36.3.12 getPlayer1IndexPreviousButton()

```
std::shared_ptr< Button > Withdraw::getPlayer1IndexPreviousButton ( )
```

Getter for [Player](#) 1 index previous property button.

##### Returns

A shared pointer to the previous property [Button](#) object for [Player](#) 1 offer.

#### 6.36.3.13 getPlayer1IndexProperties()

```
std::vector< unsigned int > Withdraw::getPlayer1IndexProperties ( )
```

Getter for the properties of [Player](#) 1 offer.

##### Returns

A vector of unsigned integers representing the indexes of [Player](#) 1 offer properties.

#### 6.36.3.14 getPlayer1minus1()

```
std::shared_ptr< Button > Withdraw::getPlayer1minus1 ( )
```

Getter for Active [Player](#) minus 1 money button.

##### Returns

A shared pointer to the minus 1 [Button](#) object for Active [Player](#).

#### 6.36.3.15 getPlayer1minus10()

```
std::shared_ptr< Button > Withdraw::getPlayer1minus10 ( )
```

Getter for Active [Player](#) minus 10 money button.

##### Returns

A shared pointer to the minus 10 [Button](#) object for Active [Player](#).

#### 6.36.3.16 getPlayer1minus100()

```
std::shared_ptr< Button > Withdraw::getPlayer1minus100 ( )
```

Getter for Active [Player](#) minus 100 money button.

##### Returns

A shared pointer to the minus 100 [Button](#) object for Active [Player](#).

#### 6.36.3.17 getPlayer1NextButton()

```
std::shared_ptr< Button > Withdraw::getPlayer1NextButton ( )
```

Getter for [Player](#) 1 next property button.

##### Returns

A shared pointer to the next property [Button](#) object for [Player](#) 1.

#### 6.36.3.18 getPlayer1plus1()

```
std::shared_ptr< Button > Withdraw::getPlayer1plus1 ( )
```

Getter for Active [Player](#) plus 1 money button.

##### Returns

A shared pointer to the minus 1 [Button](#) object for Active [Player](#).

### 6.36.3.19 getPlayer1plus10()

```
std::shared_ptr< Button > Withdraw::getPlayer1plus10 ( )
```

Getter for Active [Player](#) plus 10 money button.

#### Returns

A shared pointer to the minus 10 [Button](#) object for Active [Player](#).

### 6.36.3.20 getPlayer1plus100()

```
std::shared_ptr< Button > Withdraw::getPlayer1plus100 ( )
```

Getter for Active [Player](#) plus 100 money button.

#### Returns

A shared pointer to the minus 100 [Button](#) object for Active [Player](#).

### 6.36.3.21 getPlayer1PreviousButton()

```
std::shared_ptr< Button > Withdraw::getPlayer1PreviousButton ( )
```

Getter for [Player](#) 1 previous property button.

#### Returns

A shared pointer to the previous property [Button](#) object for [Player](#) 1.

### 6.36.3.22 getPlayer1RemoveButton()

```
std::shared_ptr< Button > Withdraw::getPlayer1RemoveButton ( )
```

Getter for [Player](#) 1 remove from offer property button.

#### Returns

A shared pointer to the remove property [Button](#) object from [Player](#) 1 offer.

### 6.36.3.23 getPlayer1ToWithdraw()

```
std::shared_ptr< Player > Withdraw::getPlayer1ToWithdraw ( )
```

Getter for Active [Player](#) object.

#### Returns

A shared pointer to the [Player](#) object.

### 6.36.3.24 getPlayer2AddButton()

```
std::shared_ptr< Button > Withdraw::getPlayer2AddButton ( )
```

Getter for [Player](#) 2 add to offer property button.

#### Returns

A shared pointer to the add property [Button](#) object for [Player](#) 2 offer.

### 6.36.3.25 getPlayer2Button()

```
std::shared_ptr< Button > Withdraw::getPlayer2Button ( )
```

Getter for button for choosing [Player](#) 2 as passive in withdraw.

#### Returns

A shared pointer to choose [Player](#) 2 [Button](#) object.

### 6.36.3.26 getPlayer2IndexNextButton()

```
std::shared_ptr< Button > Withdraw::getPlayer2IndexNextButton ( )
```

Getter for [Player](#) 2 index next property button.

#### Returns

A shared pointer to the next property [Button](#) object for [Player](#) 2 offer.

### 6.36.3.27 getPlayer2IndexPreviousButton()

```
std::shared_ptr< Button > Withdraw::getPlayer2IndexPreviousButton ( )
```

Getter for [Player](#) 2 index previous property button.

#### Returns

A shared pointer to the previous property [Button](#) object for [Player](#) 2 offer.

### 6.36.3.28 getPlayer2IndexProperties()

```
std::vector< unsigned int > Withdraw::getPlayer2IndexProperties ( )
```

Getter for the properties of [Player](#) 2 offer.

#### Returns

A vector of unsigned integers representing the indexes of [Player](#) 2 offer properties.

### 6.36.3.29 getPlayer2minus1()

```
std::shared_ptr< Button > Withdraw::getPlayer2minus1 ( )
```

Getter for Passive [Player](#) minus 1 money button.

#### Returns

A shared pointer to the minus 1 [Button](#) object for Passive [Player](#).

### 6.36.3.30 getPlayer2minus10()

```
std::shared_ptr< Button > Withdraw::getPlayer2minus10 ( )
```

Getter for Passive [Player](#) minus 10 money button.

#### Returns

A shared pointer to the minus 10 [Button](#) object for Passive [Player](#).

#### 6.36.3.31 getPlayer2minus100()

```
std::shared_ptr< Button > Withdraw::getPlayer2minus100 ( )
```

Getter for Passive [Player](#) minus 100 money button.

##### Returns

A shared pointer to the minus 100 [Button](#) object for Passive [Player](#).

#### 6.36.3.32 getPlayer2NextButton()

```
std::shared_ptr< Button > Withdraw::getPlayer2NextButton ( )
```

Getter for [Player](#) 2 next property button.

##### Returns

A shared pointer to the next property [Button](#) object for [Player](#) 2.

#### 6.36.3.33 getPlayer2plus1()

```
std::shared_ptr< Button > Withdraw::getPlayer2plus1 ( )
```

Getter for Passive [Player](#) plus 1 money button.

##### Returns

A shared pointer to the plus 1 [Button](#) object for Passive [Player](#).

#### 6.36.3.34 getPlayer2plus10()

```
std::shared_ptr< Button > Withdraw::getPlayer2plus10 ( )
```

Getter for Passive [Player](#) plus 10 money button.

##### Returns

A shared pointer to the plus 10 [Button](#) object for Passive [Player](#).



#### 6.36.3.35 getPlayer2plus100()

```
std::shared_ptr< Button > Withdraw::getPlayer2plus100 ( )
```

Getter for Passive [Player](#) plus 100 money button.

##### Returns

A shared pointer to the plus 100 [Button](#) object for Passive [Player](#).

#### 6.36.3.36 getPlayer2PreviousButton()

```
std::shared_ptr< Button > Withdraw::getPlayer2PreviousButton ( )
```

Getter for [Player](#) 2 previous property button.

##### Returns

A shared pointer to the previous property [Button](#) object for [Player](#) 2 .

#### 6.36.3.37 getPlayer2RemoveButton()

```
std::shared_ptr< Button > Withdraw::getPlayer2RemoveButton ( )
```

Getter for [Player](#) 2 remove from offer property button.

##### Returns

A shared pointer to the remove property [Button](#) object from [Player](#) 2 offer.

#### 6.36.3.38 getPlayer2ToWithdraw()

```
std::shared_ptr< Player > Withdraw::getPlayer2ToWithdraw ( )
```

Getter for Passive [Player](#) object.

##### Returns

A shared pointer to the [Player](#) object.

#### 6.36.3.39 getPlayer3Button()

```
std::shared_ptr< Button > Withdraw::getPlayer3Button ( )
```

Getter for button for choosing [Player 3](#) as passive in withdraw.

##### Returns

A shared pointer to choose [Player 3 Button](#) object.

#### 6.36.3.40 getPlayer4Button()

```
std::shared_ptr< Button > Withdraw::getPlayer4Button ( )
```

Getter for button for choosing [Player 4](#) as passive in withdraw.

##### Returns

A shared pointer to choose [Player 4 Button](#) object.

#### 6.36.3.41 getResignButton()

```
std::shared_ptr< Button > Withdraw::getResignButton ( )
```

Getter for the resign button.

##### Returns

A shared pointer to the resign [Button](#) object.

#### 6.36.3.42 getResignDecisionButton()

```
std::shared_ptr< Button > Withdraw::getResignDecisionButton ( )
```

Getter for the resign decision button.

##### Returns

A shared pointer to the resign decision [Button](#) object.

#### 6.36.3.43 getResignValueButton()

```
std::shared_ptr< Button > Withdraw::getResignValueButton ( )
```

Getter for the resign button from value screen.

##### Returns

A shared pointer to the resign value [Button](#) object.

#### 6.36.3.44 getSpritePropertyPlayer1()

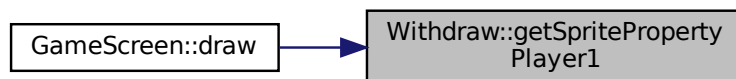
```
sf::Sprite & Withdraw::getSpritePropertyPlayer1 ( )
```

Getter for [Player](#) 1's property sprite.

##### Returns

A reference to the SFML Sprite object representing [Player](#) 1's properties.

Here is the caller graph for this function:



#### 6.36.3.45 getSpritePropertyPlayer1Index()

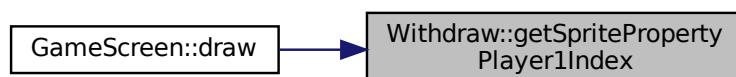
```
sf::Sprite & Withdraw::getSpritePropertyPlayer1Index ( )
```

Getter for [Player](#) 1's offered property sprite.

##### Returns

A reference to the SFML Sprite object representing [Player](#) 1's offered properties.

Here is the caller graph for this function:



#### 6.36.3.46 `getSpritePropertyPlayer2()`

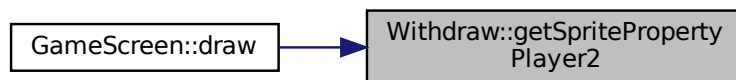
```
sf::Sprite & Withdraw::getSpritePropertyPlayer2 ( )
```

Getter for [Player 2](#)'s property sprite.

##### Returns

A reference to the SFML Sprite object representing [Player 2](#)'s properties.

Here is the caller graph for this function:



#### 6.36.3.47 `getSpritePropertyPlayer2Index()`

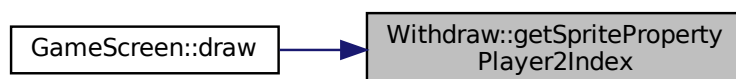
```
sf::Sprite & Withdraw::getSpritePropertyPlayer2Index ( )
```

Getter for [Player 2](#)'s offered property sprite.

##### Returns

A reference to the SFML Sprite object representing [Player 2](#)'s offered properties.

Here is the caller graph for this function:



**6.36.3.48 getSubmitValueButton()**

```
std::shared_ptr< Button > Withdraw::getSubmitValueButton ( )
```

Getter for the submit value button from value screen.

**Returns**

A shared pointer to the submit value [Button](#) object.

**6.36.3.49 getTextures()**

```
std::vector< std::shared_ptr< sf::Text > > & Withdraw::getTexts ( )
```

Getter for the vector of text objects used in the withdrawal process.

**Returns**

A vector of shared pointers to SFML Text objects.

Here is the caller graph for this function:

**6.36.3.50 getTextsPropertyPlayer1()**

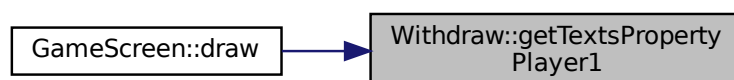
```
std::vector< std::shared_ptr< sf::Text > > & Withdraw::getTextsPropertyPlayer1 ( )
```

Getter for the text objects representing [Player](#) 1's property details.

**Returns**

A vector of shared pointers to SFML Text objects.

Here is the caller graph for this function:



### 6.36.3.51 `getTextsPropertyPlayer1Index()`

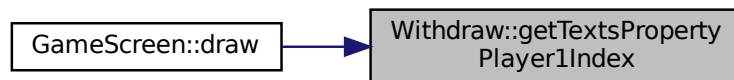
```
std::vector< std::shared_ptr< sf::Text > > & Withdraw::getTextsPropertyPlayer1Index ( )
```

Getter for the text objects representing [Player 1](#)'s offered property details.

#### Returns

A vector of shared pointers to SFML Text objects.

Here is the caller graph for this function:



### 6.36.3.52 `getTextsPropertyPlayer2()`

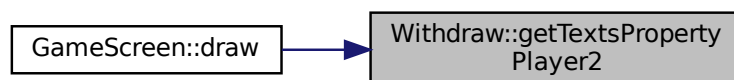
```
std::vector< std::shared_ptr< sf::Text > > & Withdraw::getTextsPropertyPlayer2 ( )
```

Getter for the text objects representing [Player 2](#)'s property details.

#### Returns

A vector of shared pointers to SFML Text objects.

Here is the caller graph for this function:



### 6.36.3.53 `getTextsPropertyPlayer2Index()`

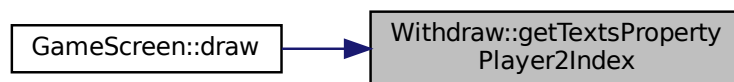
```
std::vector< std::shared_ptr< sf::Text > > & Withdraw::getTextsPropertyPlayer2Index ( )
```

Getter for the text objects representing [Player](#) 2's offered property details.

#### Returns

A vector of shared pointers to SFML Text objects.

Here is the caller graph for this function:



### 6.36.3.54 `getTexturePropertyPlayer1()`

```
std::shared_ptr< sf::Texture > Withdraw::getTexturePropertyPlayer1 ( )
```

Getter for [Player](#) 1's property texture.

#### Returns

SFML Texrute object representing [Player](#) 1's properties.

Here is the caller graph for this function:



### 6.36.3.55 `getTexturePropertyPlayer1Index()`

```
std::shared_ptr< sf::Texture > Withdraw::getTexturePropertyPlayer1Index ( )
```

Getter for [Player](#) 1's offer property texture.

#### Returns

SFML Texrute object representing [Player](#) 1's properties offer.

Here is the caller graph for this function:



### 6.36.3.56 `getTexturePropertyPlayer2()`

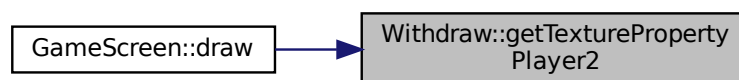
```
std::shared_ptr< sf::Texture > Withdraw::getTexturePropertyPlayer2 ( )
```

Getter for [Player](#) 2's property texture.

#### Returns

SFML Texrute object representing [Player](#) 2's properties.

Here is the caller graph for this function:





### 6.36.3.57 `getTexturePropertyPlayer2Index()`

```
std::shared_ptr< sf::Texture > Withdraw::getTexturePropertyPlayer2Index ( )
```

Getter for [Player](#) 2's offer property texture.

#### Returns

SFML Textute object representing [Player](#) 2's properties offer.

Here is the caller graph for this function:



### 6.36.3.58 `getTurnState()`

```
TurnState Withdraw::getTurnState ( )
```

Getter for the turn state.

#### Returns

The current TurnState enumeration representing the turn state of monopolygame when withdraw was started.

### 6.36.3.59 `isNonZeroValue()`

```
bool Withdraw::isNonZeroValue ( )
```

Check whenever withdraw is legal to be done.

#### Returns

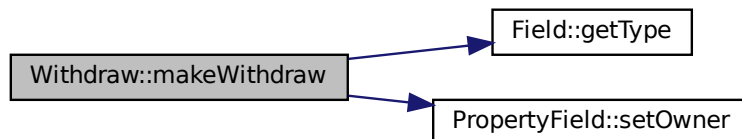
true if withdraw is legal, otherwise false

### 6.36.3.60 makeWithdraw()

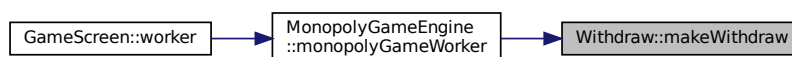
```
void Withdraw::makeWithdraw ( )
```

Initiates the withdrawal process (exchange of money and properties).

Here is the call graph for this function:



Here is the caller graph for this function:

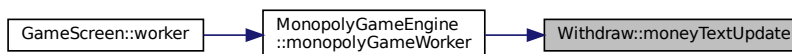


### 6.36.3.61 moneyTextUpdate()

```
void Withdraw::moneyTextUpdate ( )
```

Updates the displayed money text during the withdrawal process.

Here is the caller graph for this function:



### 6.36.3.62 moneyTransferIndex()

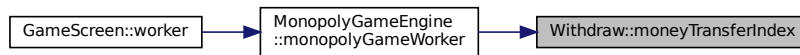
```
void Withdraw::moneyTransferIndex (
    unsigned int player_num,
    int money )
```

Transfers money during the withdrawal process.

## Parameters

<i>player_num</i>	An unsigned integer representing the player number (1-2).
<i>money</i>	An integer representing the amount of money to transfer (can be negative).

Here is the caller graph for this function:



## 6.36.3.63 propertyPlayerMoveIndex()

```

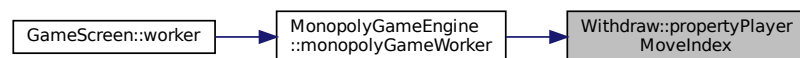
void Withdraw::propertyPlayerMoveIndex (
    int dir,
    unsigned int plr_num )
  
```

Moves properties between players ownership and withdraw offer.

## Parameters

<i>dir</i>	if > 0 moves to offer vector, if < 0 moves from offer vectpr
<i>plr_num</i>	if 1 - active player, if 2 - passive player, in withdraw

Here is the caller graph for this function:



## 6.36.3.64 setBoard()

```

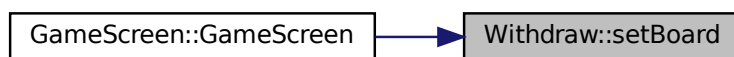
void Withdraw::setBoard (
    std::shared_ptr< Board > board_ptr )
  
```

Setter for the gameboard.

## Parameters

<i>board_ptr</i>	A shared pointer to the <a href="#">Board</a> object.
------------------	---

Here is the caller graph for this function:



### 6.36.3.65 setChooseScreenVisible()

```
void Withdraw::setChooseScreenVisible (
    bool is_visible )
```

Sets the visibility of the choose player screen.

## Parameters

<i>is_visible</i>	A boolean indicating whether the screen should be visible.
-------------------	--

Here is the caller graph for this function:



### 6.36.3.66 setDecisionScreenVisible()

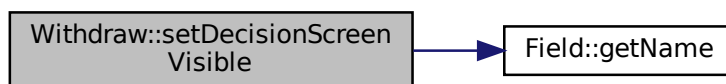
```
void Withdraw::setDecisionScreenVisible (
    bool is_visible )
```

Sets the visibility of the decision player screen.

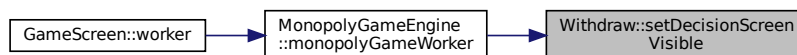
## Parameters

<i>is_visible</i>	A boolean indicating whether the screen should be visible.
-------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.36.3.67 setFont()

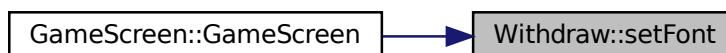
```
void Withdraw::setFont (
    sf::Font font )
```

Setter for the font used in rendering text.

## Parameters

<i>font</i>	A reference to the SFML Font object.
-------------	--------------------------------------

Here is the caller graph for this function:



### 6.36.3.68 setPlayer1IndexProperties()

```
void Withdraw::setPlayer1IndexProperties (
    std::vector< unsigned int > new_index_properties )
```

Setter for the properties of [Player](#) 1 offer.

#### Parameters

<i>new_index_properties</i>	A vector of unsigned integers representing the new index for <a href="#">Player</a> 1 offer properties.
-----------------------------	---

### 6.36.3.69 setPlayer1ToWithdraw()

```
void Withdraw::setPlayer1ToWithdraw (
    std::shared_ptr< Player > player_ptr )
```

Sets Active [Player](#) for withdrawal.

#### Parameters

<i>player_ptr</i>	A shared pointer to the <a href="#">Player</a> object.
-------------------	--

### 6.36.3.70 setPlayer2IndexProperties()

```
void Withdraw::setPlayer2IndexProperties (
    std::vector< unsigned int > new_index_properties )
```

Setter for the properties of [Player](#) 2 offer.

#### Parameters

<i>new_index_properties</i>	A vector of unsigned integers representing the new index for <a href="#">Player</a> 2 offer properties.
-----------------------------	---

### 6.36.3.71 setPlayer2ToWithdraw()

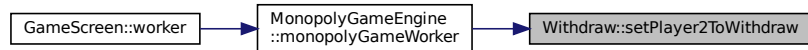
```
void Withdraw::setPlayer2ToWithdraw (
    std::shared_ptr< Player > player_ptr )
```

Sets Passive [Player](#) for withdrawal.

## Parameters

<i>player_ptr</i>	A shared pointer to the <a href="#">Player</a> object.
-------------------	--

Here is the caller graph for this function:



### 6.36.3.72 setTurnState()

```
void Withdraw::setTurnState (
    TurnState state )
```

Setter for the turn state of monopolygame when withdraw was started.

## Parameters

<i>state</i>	A TurnState enumeration representing the turn state.
--------------	--

### 6.36.3.73 setValueScreenVisible()

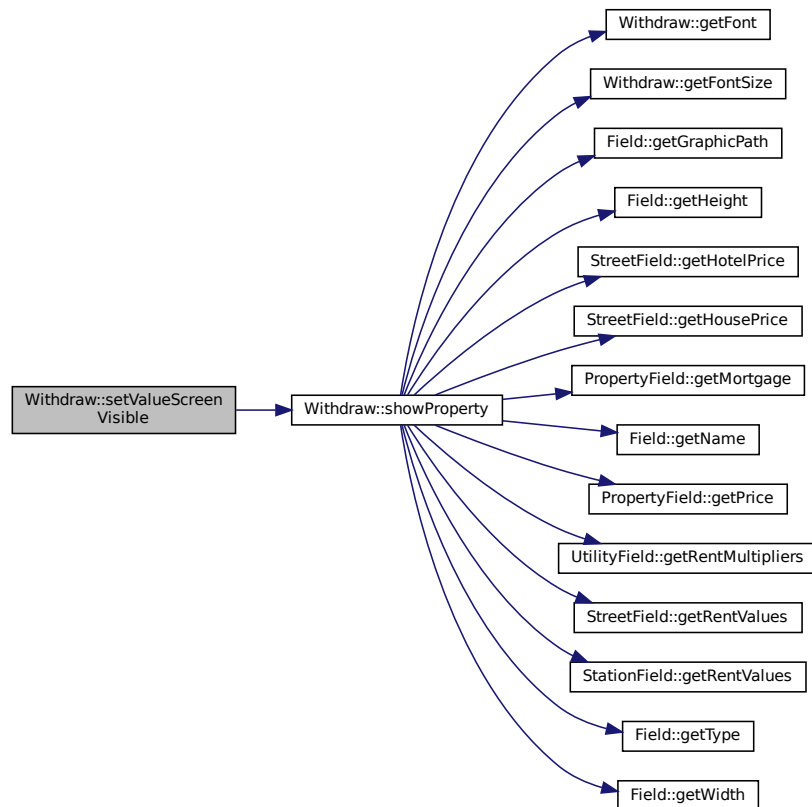
```
void Withdraw::setValueScreenVisible (
    bool is_visible )
```

Sets the visibility of the value player screen.

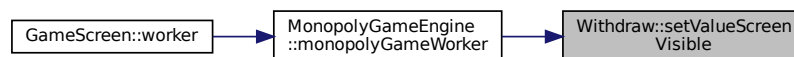
## Parameters

<i>is_visible</i>	A boolean indicating whether the screen should be visible.
-------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.36.3.74 showProperty()

```
void Withdraw::showProperty (
    int column )
```

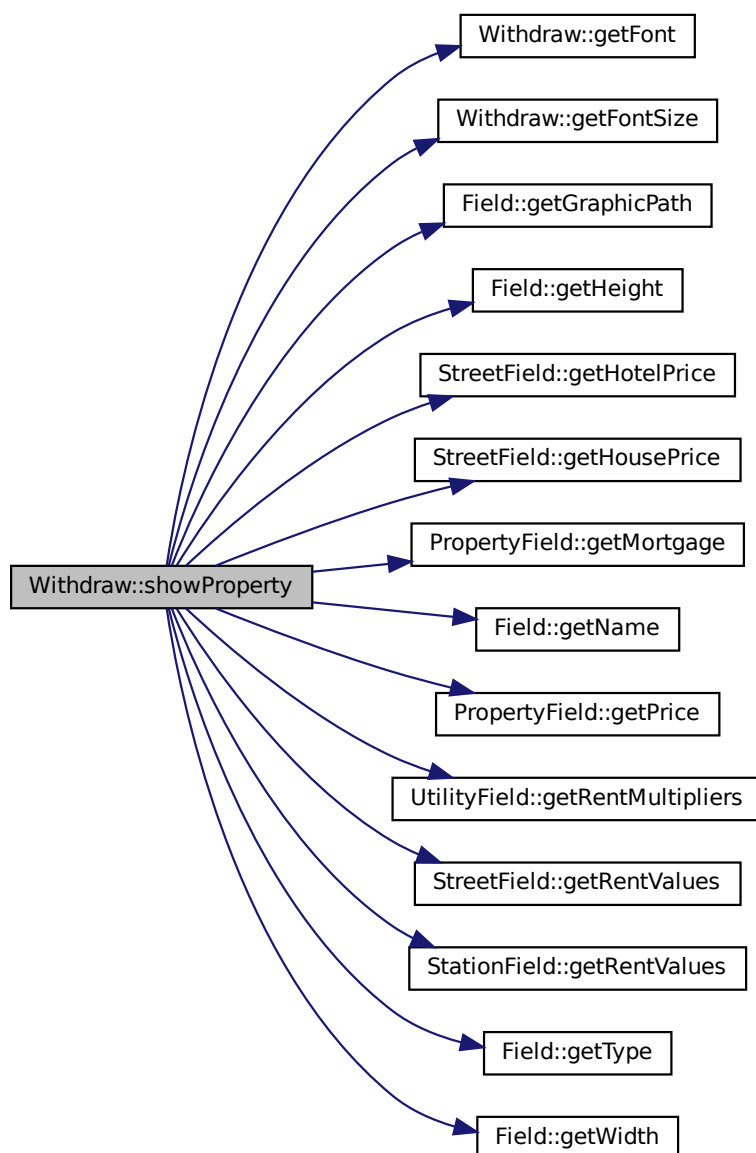
Shows the property details for the specified column during the withdrawal process.

##### Parameters

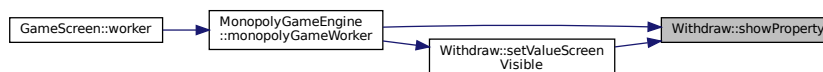
<i>column</i>	An integer representing the column index (1-4).
---------------	---



Here is the call graph for this function:



Here is the caller graph for this function:



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

- </home/kamil/zpr/Monopoly/Withdraw.h>
- </home/kamil/zpr/Monopoly/Withdraw.cc>

## Chapter 7

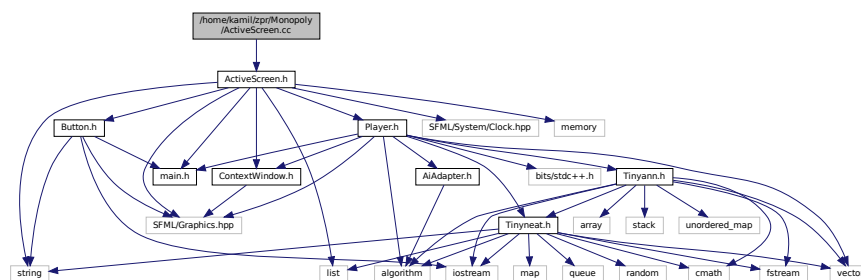
# File Documentation

### 7.1 /home/kamil/zpr/Monopoly/ActiveScreen.cc File Reference

Source file handling displayed screens of project Base class is [ActiveScreen](#), then derived class are used to work with specific screen shown.

```
#include "ActiveScreen.h"
```

Include dependency graph for ActiveScreen.cc:



#### 7.1.1 Detailed Description

Source file handling displayed screens of project Base class is [ActiveScreen](#), then derived class are used to work with specific screen shown.

#### Author

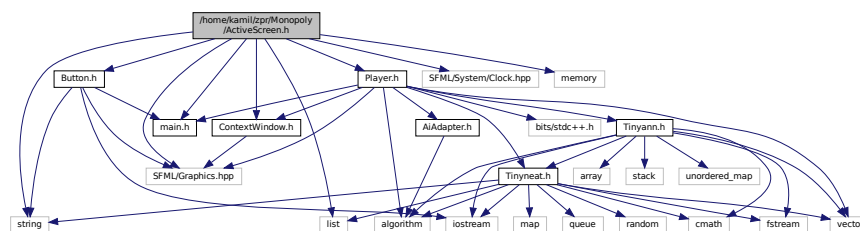
Kamil Kosnik, Kacper Radzikowski

## 7.2 /home/kamil/zpr/Monopoly/ActiveScreen.h File Reference

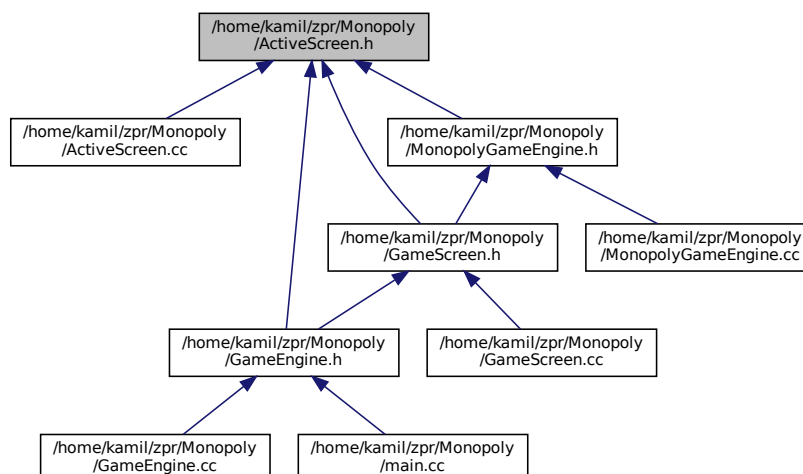
Header file handling displayed screens of project Base class is [ActiveScreen](#), then derived class are used to work with specific screen shown.

```
#include <SFML/Graphics.hpp>
#include <SFML/System/Clock.hpp>
#include <list>
#include <memory>
#include <string>
#include "Button.h"
#include "ContextWindow.h"
#include "Player.h"
#include "main.h"
```

Include dependency graph for ActiveScreen.h:



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



### Classes

- class [ActiveScreen](#)  
Represents the base class for handling displayed screens in the project.
- class [GameMenuScreen](#)  
Represents the screen for the game menu.
- class [MainMenuScreen](#)  
Represents the screen for the main menu.

### 7.2.1 Detailed Description

Header file handling displayed screens of project Base class is [ActiveScreen](#), then derived class are used to work with specific screen shown.

#### Author

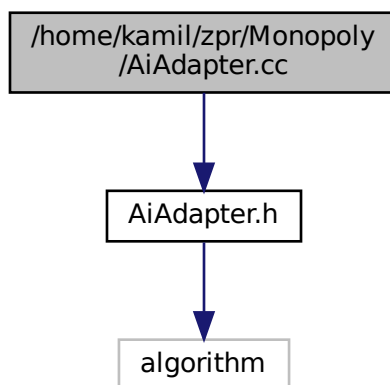
Kamil Kosnik, Kacper Radzikowski

## 7.3 /home/kamil/zpr/Monopoly/AiAdapter.cc File Reference

TODO.

```
#include "AiAdapter.h"
```

Include dependency graph for AiAdapter.cc:



### 7.3.1 Detailed Description

TODO.

#### Author

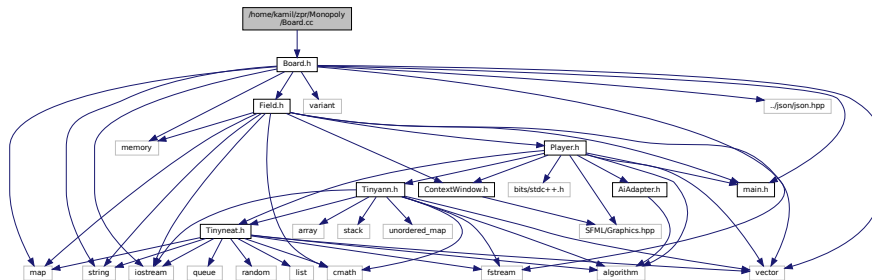
Kamil Kosnik, Kacper Radzikowski



## 7.5 /home/kamil/zpr/Monopoly/Board.cc File Reference

Source file for monopoly game board, creation is based on json file 'board.json'. Mainly handles all types of fields and their usage.

```
#include "Board.h"
Include dependency graph for Board.cc:
```



### Typedefs

- using `json` = `nlohmann::json`

### Functions

- `std::map< StreetTiers, unsigned int > jsonToStreetRent` (const `json` &element)  
*Converts JSON data to a map of rent values for street properties.*
- `std::map< StationTiers, unsigned int > jsonToStationRent` (const `json` &element)  
*Converts JSON data to a map of rent values for station properties.*
- `std::map< UtilityTiers, unsigned int > jsonToUtilityRent` (const `json` &element)  
*Converts JSON data to a map of rent multipliers for utility properties.*

#### 7.5.1 Detailed Description

Source file for monopoly game board, creation is based on json file 'board.json'. Mainly handles all types of fields and their usage.

#### Author

Kamil Kosnik, Kacper Radzikowski

#### 7.5.2 Typedef Documentation

##### 7.5.2.1 json

```
using json = nlohmann::json
```

### 7.5.3 Function Documentation

#### 7.5.3.1 jsonToStationRent()

```
std::map<StationTiers, unsigned int> jsonToStationRent (
    const json & element )
```

Converts JSON data to a map of rent values for station properties.

##### Parameters

<i>element</i>	The JSON data containing rent values.
----------------	---------------------------------------

##### Returns

A map of rent values for station properties.

Here is the caller graph for this function:



#### 7.5.3.2 jsonToStreetRent()

```
std::map<StreetTiers, unsigned int> jsonToStreetRent (
    const json & element )
```

Converts JSON data to a map of rent values for street properties.

##### Parameters

<i>element</i>	The JSON data containing rent values.
----------------	---------------------------------------



#### Returns

A map of rent values for street properties.

Here is the caller graph for this function:



#### 7.5.3.3 jsonToUtilityRent()

```
std::map<UtilityTiers, unsigned int> jsonToUtilityRent (  
    const json & element )
```

Converts JSON data to a map of rent multipliers for utility properties.

#### Parameters

<i>element</i>	The JSON data containing rent multipliers.
----------------	--

#### Returns

A map of rent multipliers for utility properties.

Here is the caller graph for this function:



## 7.6 /home/kamil/zpr/Monopoly/Board.h File Reference

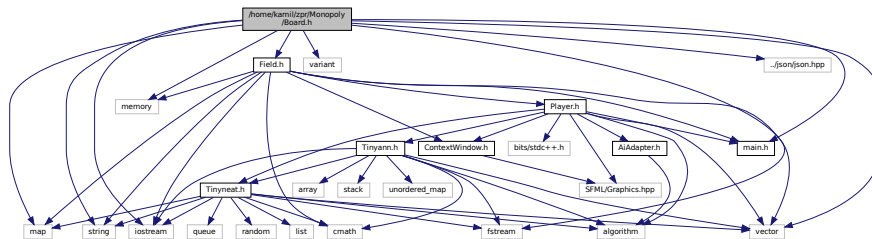
Header file for monopoly game board, creation is based on json file 'board.json'. Mainly handles all types of fields and their usage.

```

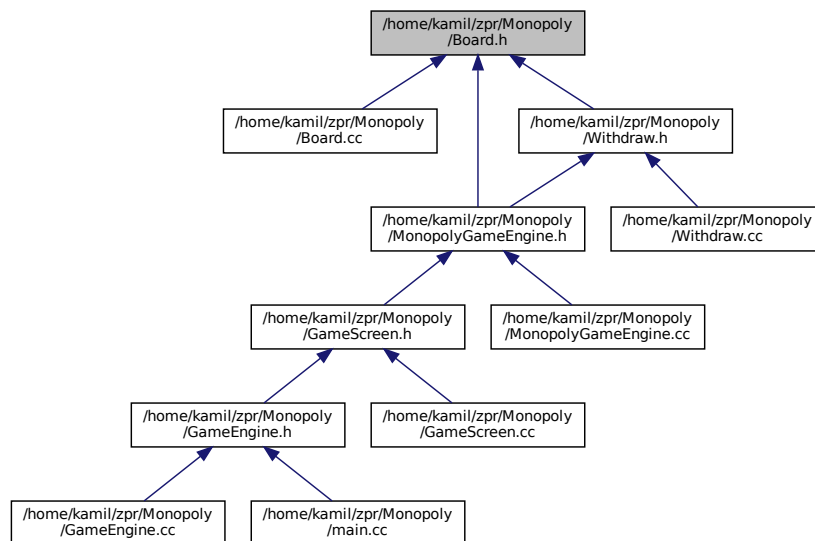
#include <fstream>
#include <iostream>
#include <map>
#include <memory>
#include <string>
#include <variant>
#include <vector>
#include "../json/json.hpp"
#include "Field.h"
#include "main.h"

```

Include dependency graph for Board.h:



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



## Classes

- class [Board](#)

*Class representing the monopoly game board.*

## Typedefs

- using [json](#) = nlohmann::json
- using [PossibleFields](#) = std::variant< [Field](#), [PropertyField](#), [StreetField](#), [StationField](#), [UtilityField](#), [TaxField](#) >

*A variant type representing different types of fields on the monopoly board.*

## Functions

- `std::map< StreetTiers, unsigned int > jsonToStreetRent (const json &element)`
- Converts JSON data to a map of rent values for street properties.*
- `std::map< StationTiers, unsigned int > jsonToStationRent (const json &element)`
- Converts JSON data to a map of rent values for station properties.*
- `std::map< UtilityTiers, unsigned int > jsonToUtilityRent (const json &element)`
- Converts JSON data to a map of rent multipliers for utility properties.*

### 7.6.1 Detailed Description

Header file for monopoly game board, creation is based on json file 'board.json'. Mainly handles all types of fields and their usage.

#### Author

Kamil Kosnik, Kacper Radzikowski

### 7.6.2 Typedef Documentation

#### 7.6.2.1 `json`

```
using json = nlohmann::json
```

#### 7.6.2.2 `PossibleFields`

[PossibleFields](#)

A variant type representing different types of fields on the monopoly board.

### 7.6.3 Function Documentation

#### 7.6.3.1 `jsonToStationRent()`

```
std::map<StationTiers, unsigned int> jsonToStationRent (  
    const json & element )
```

Converts JSON data to a map of rent values for station properties.

**Parameters**

<i>element</i>	The JSON data containing rent values.
----------------	---------------------------------------

**Returns**

A map of rent values for station properties.

Here is the caller graph for this function:

**7.6.3.2 jsonToStreetRent()**

```
std::map<StreetTiers, unsigned int> jsonToStreetRent (  
    const json & element )
```

Converts JSON data to a map of rent values for street properties.

**Parameters**

<i>element</i>	The JSON data containing rent values.
----------------	---------------------------------------

**Returns**

A map of rent values for street properties.

Here is the caller graph for this function:



### 7.6.3.3 jsonToUtilityRent()

```
std::map<UtilityTiers, unsigned int> jsonToUtilityRent (
    const json & element )
```

Converts JSON data to a map of rent multipliers for utility properties.

#### Parameters

<i>element</i>	The JSON data containing rent multipliers.
----------------	--

#### Returns

A map of rent multipliers for utility properties.

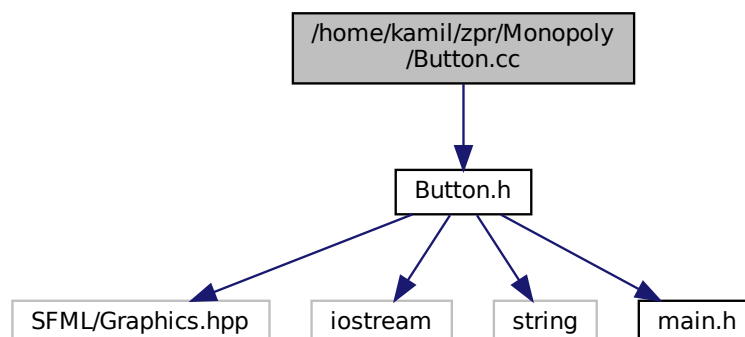
Here is the caller graph for this function:



## 7.7 /home/kamil/zpr/Monopoly/Button.cc File Reference

```
#include "Button.h"
```

Include dependency graph for Button.cc:

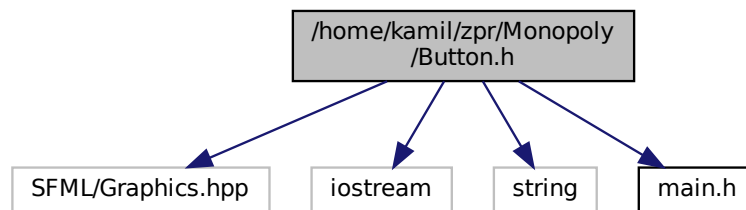


## 7.8 /home/kamil/zpr/Monopoly/Button.h File Reference

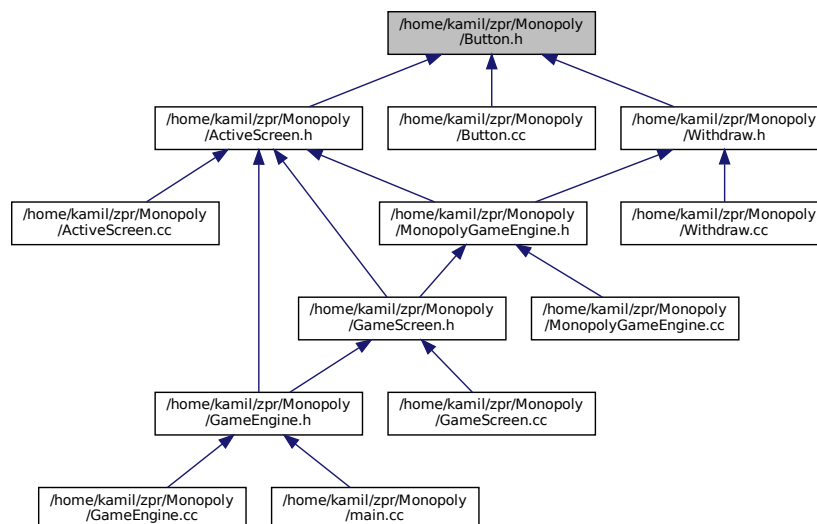
Source file for handling button objects actions used to communicate with user.

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>
#include "main.h"
```

Include dependency graph for Button.h:



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



## Classes

- class [Button](#)

*Represents a button for handling user interactions.*

### 7.8.1 Detailed Description

Source file for handling button objects actions used to communicate with user.

Header file for handling button objects actions used to communicate with user.

#### Author

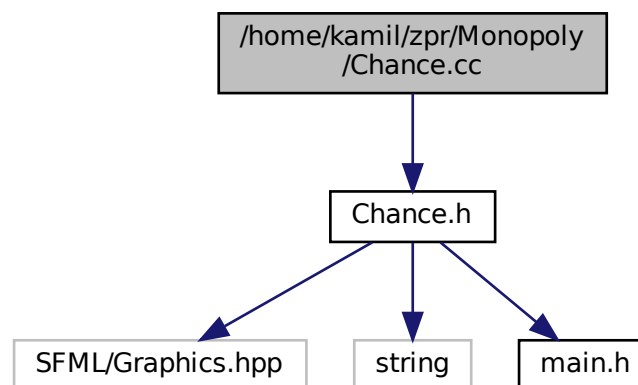
Kamil Kosnik, Kacper Radzikowski

## 7.9 /home/kamil/zpr/Monopoly/Chance.cc File Reference

Source file for monopoly game chance cards, their types, actions...

```
#include "Chance.h"
```

Include dependency graph for Chance.cc:



### 7.9.1 Detailed Description

Source file for monopoly game chance cards, their types, actions...

#### Author

Kamil Kosnik, Kacper Radzikowski





## Enumerations

- enum [ChanceType](#) {  
[MOVEMENT\\_TO\\_PROPERTY](#) , [MOVEMENT\\_WITH\\_BUY\\_OR\\_PAY](#) , [BANK\\_PAYS\\_YOU](#) , [GET\\_OUT\\_OF\\_JAIL\\_CARD](#)  
 ,  
[MOVEMENT\\_SPACES](#) , [GO\\_TO\\_JAIL\\_CARD](#) , [PAY\\_FOR\\_HOUSE\\_HOTEL](#) , [TAX\\_CARD](#) ,  
[PAY\\_PLAYERS](#) }

*Enumeration representing different types of chance cards.*

### 7.10.1 Detailed Description

Header file for monopoly game chance cards, their types, actions...

#### Author

Kamil Kosnik, Kacper Radzikowski

### 7.10.2 Enumeration Type Documentation

#### 7.10.2.1 ChanceType

enum [ChanceType](#)

Enumeration representing different types of chance cards.

#### Enumerator

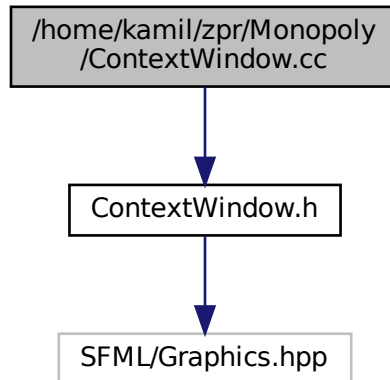
<a href="#">MOVEMENT_TO_PROPERTY</a>	Move to a specific property.
<a href="#">MOVEMENT_WITH_BUY_OR_PAY</a>	Move with the option to buy or pay.
<a href="#">BANK_PAYS_YOU</a>	Receive money from the bank.
<a href="#">GET_OUT_OF_JAIL_CARD</a>	Receive a "Get Out of Jail Free" card.
<a href="#">MOVEMENT_SPACES</a>	Move a certain number of spaces.
<a href="#">GO_TO_JAIL_CARD</a>	Go directly to jail.
<a href="#">PAY_FOR_HOUSE_HOTEL</a>	Pay for each house and hotel owned.
<a href="#">TAX_CARD</a>	Pay a tax.
<a href="#">PAY_PLAYERS</a>	Pay other players.

## 7.11 /home/kamil/zpr/Monopoly/ContextWindow.cc File Reference

Source file for context Window class It is Singleton class type used mainly for handling SFML window operations between other classes.

```
#include "ContextWindow.h"
```

Include dependency graph for ContextWindow.cc:



### 7.11.1 Detailed Description

Source file for context Window class It is Singleton class type used mainly for handling SFML window operations between other classes.

#### Author

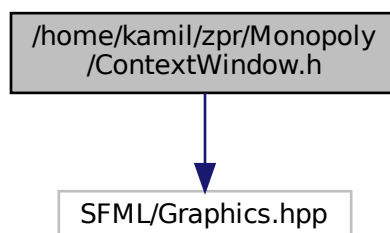
Kamil Kosnik, Kacper Radzikowski

## 7.12 `/home/kamil/zpr/Monopoly/ContextWindow.h` File Reference

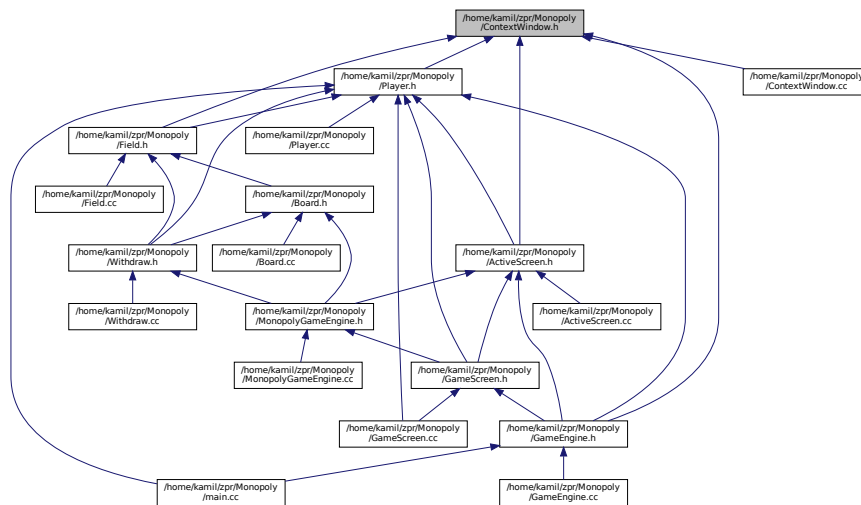
Source file for context Window class It is Singleton class type used mainly for handling SFML window operations between other classes.

```
#include <SFML/Graphics.hpp>
```

Include dependency graph for ContextWindow.h:



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



## Classes

- class [ContextWindow](#)

*Represents a Singleton class for handling SFML window operations.*

### 7.12.1 Detailed Description

Source file for context Window class It is Singleton class type used mainly for handling SFML window operations between other classes.

#### Author

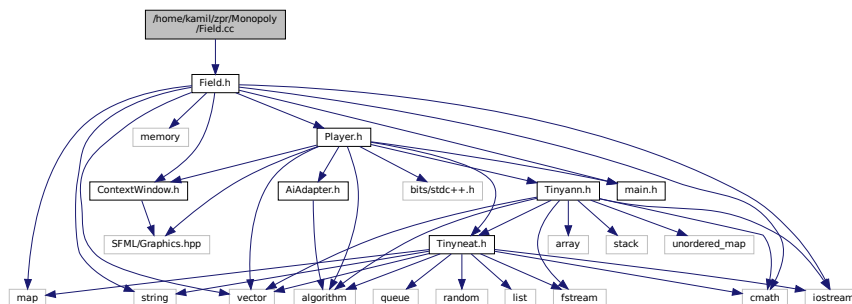
Kamil Kosnik, Kacper Radzikowski

## 7.13 /home/kamil/zpr/Monopoly/Field.cc File Reference

Source file containing staff for single monopoly board game field. Separated to many deriving classes types each for specific field type.

```
#include "Field.h"
```

Include dependency graph for Field.cc:



### 7.13.1 Detailed Description

Source file containing staff for single monopoly board game field. Separated to many deriving classes types each for specific field type.

Author

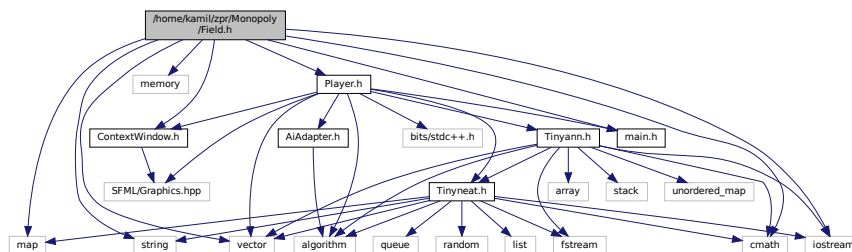
Kamil Kosnik, Kacper Radzikowski

## 7.14 /home/kamil/zpr/Monopoly/Field.h File Reference

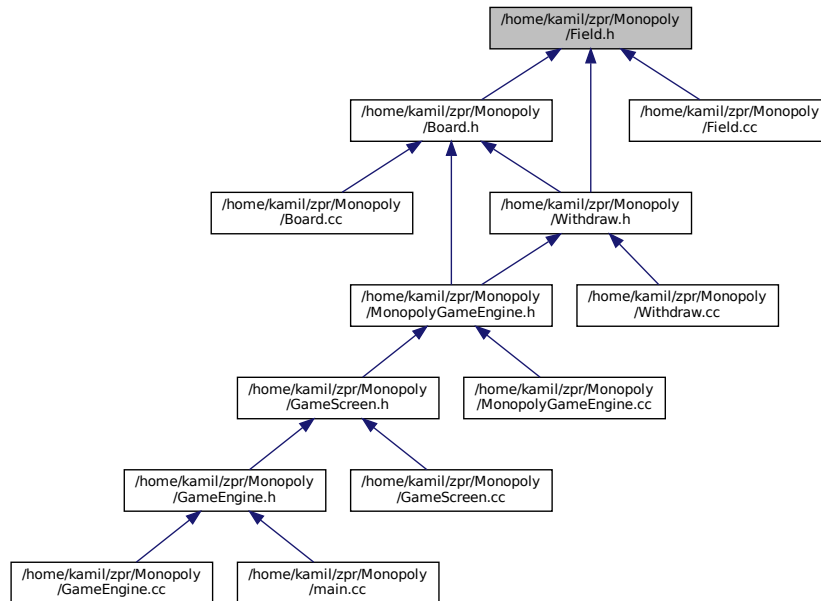
Header file containing staff for single monopoly board game field. Separated to many deriving classes types each for specific field type.

```
#include <cmath>
#include <iostream>
#include <map>
#include <memory>
#include <string>
#include <vector>
#include "ContextWindow.h"
#include "Player.h"
#include "main.h"
```

Include dependency graph for Field.h:



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



## Classes

- class [HouseException](#)  
*Custom exception class for handling invalid house numbers.*
- class [Field](#)  
*Base class representing a generic game field.*
- class [PropertyField](#)  
*Derived class representing a property field on the game board.*
- class [StreetField](#)  
*Derived class representing a street field on the game board.*
- class [StationField](#)  
*Derived class representing a station field on the game board.*
- class [UtilityField](#)
- class [TaxField](#)  
*Derived class representing a tax field on the monopoly board.*

### 7.14.1 Detailed Description

Header file containing staff for single monopoly board game field. Separated to many deriving classes types each for specific field type.

#### Author

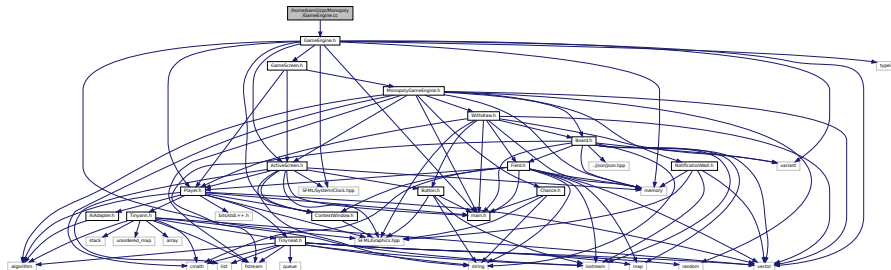
Kamil Kosnik, Kacper Radzikowski

## 7.15 /home/kamil/zpr/Monopoly/GameEngine.cc File Reference

Source file for [GameEngine](#) class, used to handle low level program operations as input interactions (mouse, keyboard) or display window.

```
#include "GameEngine.h"
```

Include dependency graph for GameEngine.cc:



### 7.15.1 Detailed Description

Source file for [GameEngine](#) class, used to handle low level program operations as input interactions (mouse, keyboard) or display window.

Author

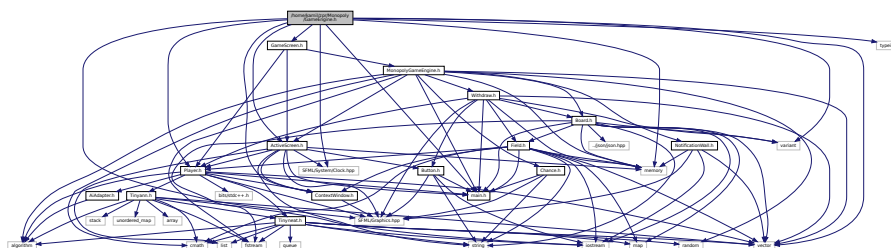
Kamil Kosnik, Kacper Radzikowski

## 7.16 /home/kamil/zpr/Monopoly/GameEngine.h File Reference

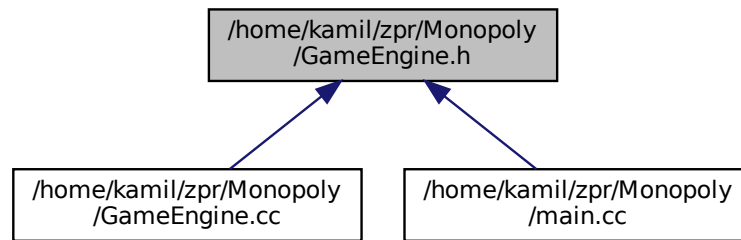
Header file for [GameEngine](#) class, used to handle low level program operations as input interactions (mouse, keyboard) or display window.

```
#include <SFML/Graphics.hpp>
#include <SFML/System/Clock.hpp>
#include <memory>
#include <typeinfo>
#include <variant>
#include <vector>
#include "ActiveScreen.h"
#include "ContextWindow.h"
#include "GameScreen.h"
#include "Player.h"
#include "main.h"
```

Include dependency graph for GameEngine.h:



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



## Classes

- class [GameEngine](#)

*Handles low-level program operations, including input interactions and window display.*

### 7.16.1 Detailed Description

Header file for [GameEngine](#) class, used to handle lowest level program operations as input interactions (mouse, keyboard) or display window.

#### Author

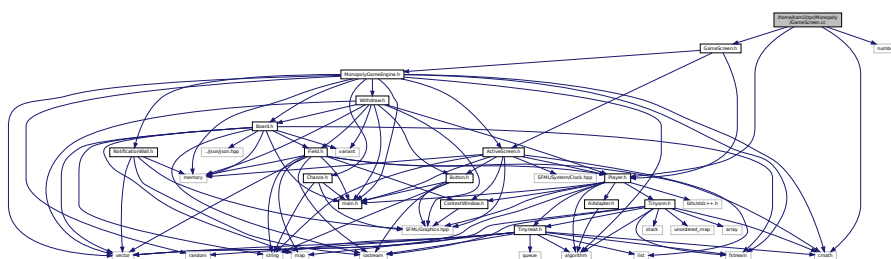
Kamil Kosnik, Kacper Radzikowski

## 7.17 /home/kamil/zpr/Monopoly/GameScreen.cc File Reference

Source file for game screen class deriving from [ActiveScreen](#) class. Used to handle monopoly game activities and drawing.

```

#include "GameScreen.h"
#include <cmath>
#include <numbers>
#include "Player.h"
Include dependency graph for GameScreen.cc:
  
```



### 7.17.1 Detailed Description

Source file for game screen class deriving from [ActiveScreen](#) class. Used to handle monopoly game activities and drawing.

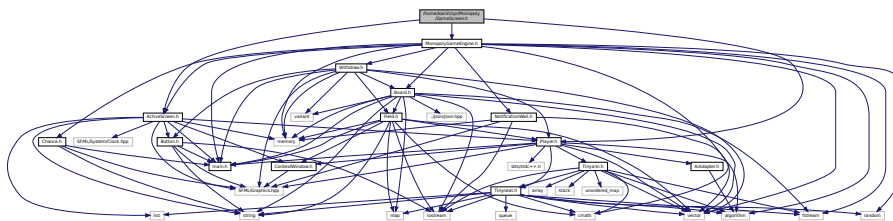
Author

Kamil Kosnik, Kacper Radzikowski

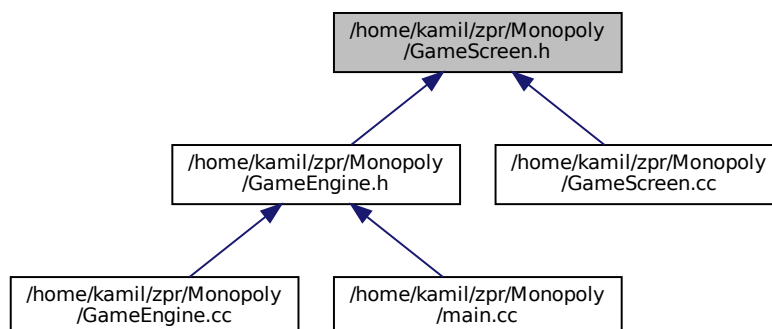
## 7.18 /home/kamil/zpr/Monopoly/GameScreen.h File Reference

Header file for game screen class deriving from [ActiveScreen](#) class. Used to handle monopoly game activities and drawing.

```
#include "ActiveScreen.h"
#include "MonopolyGameEngine.h"
#include "Player.h"
Include dependency graph for GameScreen.h:
```



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



## Classes

- class [GameScreen](#)



### 7.18.1 Detailed Description

Header file for game screen class deriving from [ActiveScreen](#) class. Used to handle monopoly game activities and drawing.

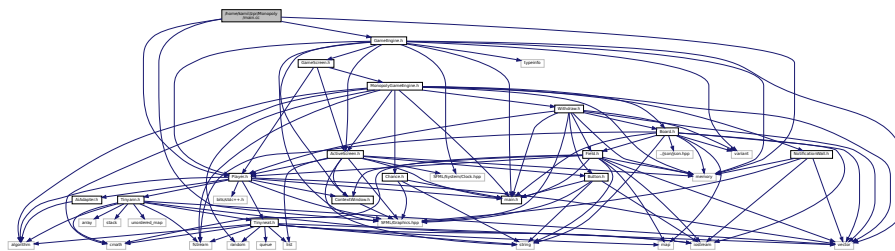
#### Author

Kamil Kosnik, Kacper Radzikowski

## 7.19 /home/kamil/zpr/Monopoly/main.cc File Reference

Source file launching monopoly game vs AI project.

```
#include <memory>
#include "GameEngine.h"
#include "Player.h"
#include "SFML/Graphics.hpp"
Include dependency graph for main.cc:
```



### Functions

- int [main](#) ()

### Variables

- unsigned int [WIDTH\\_MAX](#) = 1920
- unsigned int [HEIGHT\\_MAX](#) = 1080
- unsigned int [width](#)
- unsigned int [height](#)
- unsigned int [FRAMES\\_PER\\_SEC\\_MAX](#) = 30
- bool [TRAIN](#) = false
- int [GAMES\\_IN\\_ROUND](#) = 5

### 7.19.1 Detailed Description

Source file launching monopoly game vs AI project.

#### Author

Kamil Kosnik, Kacper Radzikowski

## 7.19.2 Function Documentation

### 7.19.2.1 main()

```
int main ( )
```

## 7.19.3 Variable Documentation

### 7.19.3.1 FRAMES\_PER\_SEC\_MAX

```
unsigned int FRAMES_PER_SEC_MAX = 30
```

### 7.19.3.2 GAMES\_IN\_ROUND

```
int GAMES_IN_ROUND = 5
```

### 7.19.3.3 height

```
unsigned int height
```

### 7.19.3.4 HEIGHT\_MAX

```
unsigned int HEIGHT_MAX = 1080
```

### 7.19.3.5 TRAIN

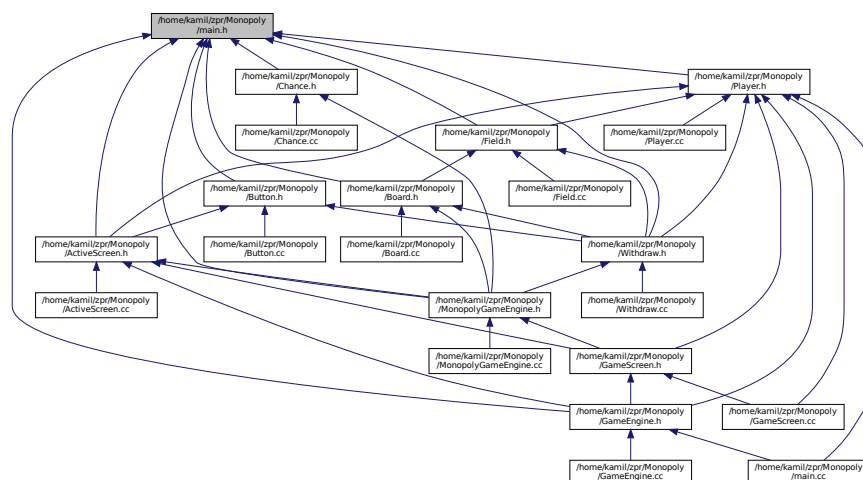
```
bool TRAIN = false
```

```
unsigned int width
```

```
unsigned int WIDTH_MAX = 1920
```

Header file containing structures shared between project files.

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



- class `DimensionException`  
*Exception for handling passing wrong dimensions to any displayed object.*
- class `RotationException`  
*Exception for handling passing wrong rotation to any displayed object.*
- class `SpriteOffsetException`  
*Exception for handling wrong offset to any displayed object.*
- struct `playerSettings`  
*Struct describing player settings manipulated from game menu.*

## Enumerations

- enum `TurnState` {  
`ROLL_DICE` , `FIELD_ACTION` , `BUY_ACTION` , `PAY_RENT` ,  
`TURN_END` , `WITHDRAW_ONGOING` , `RESULTS` , `NO_TURN` }  
*Enum describing monopoly game states.*
- enum `GameScreenType` {  
`BOARDGAME` , `WITHDRAW_CHOOSE_PLAYER` , `WITHDRAW_ADD_VALUE` , `WITHDRAW_DECISION` ,  
`AUCTION` , `RESULT` }  
*Enum describing possible showed screens in game screen object.*
- enum `ActiveScreenType` { `NONE` , `MAIN_MENU` , `GAME_MENU` , `MONOPOLY_GAME` }  
*Enum describing possible screen to be shown.*
- enum `ScreenEventType` {  
`IDLE` , `EXIT` , `PLAY` , `RETURN_TO_MAIN_MENU` ,  
`PLAYER_1_SET_NONE` , `PLAYER_2_SET_NONE` , `PLAYER_3_SET_NONE` , `PLAYER_4_SET_NONE` ,  
`PLAYER_1_SET_HUMAN` , `PLAYER_2_SET_HUMAN` , `PLAYER_3_SET_HUMAN` , `PLAYER_4_SET_HUMAN` ,  
  
`PLAYER_1_SET_AI` , `PLAYER_2_SET_AI` , `PLAYER_3_SET_AI` , `PLAYER_4_SET_AI` ,  
`PLAYER_1_SET_AI_LEVEL_1` , `PLAYER_2_SET_AI_LEVEL_1` , `PLAYER_3_SET_AI_LEVEL_1` ,  
`PLAYER_4_SET_AI_LEVEL_1` ,  
`PLAYER_1_SET_AI_LEVEL_2` , `PLAYER_2_SET_AI_LEVEL_2` , `PLAYER_3_SET_AI_LEVEL_2` ,  
`PLAYER_4_SET_AI_LEVEL_2` ,  
`PLAYER_1_SET_AI_LEVEL_3` , `PLAYER_2_SET_AI_LEVEL_3` , `PLAYER_3_SET_AI_LEVEL_3` ,  
`PLAYER_4_SET_AI_LEVEL_3` ,  
`START_GAME` , `GAME_ENDED` }  
*Enum describing events that can be returned from showed screens of ActiveScreenType type.*
- enum `FieldType` {  
`STREET` , `STATION` , `UTILITY` , `GO` ,  
`CHANCE` , `COMMUNITY_CHEST` , `TAX` , `JAIL` ,  
`FREE_PARKING` , `GO_TO_JAIL` }  
*Enum describing possible fields types in monopoly board.*
- enum `StreetTiers` {  
`NO_HOUSES` , `ONE_HOUSE` , `TWO_HOUSES` , `THREE_HOUSES` ,  
`FOUR_HOUSES` , `HOTEL` }  
*Enum describing possible street tiers (houses and hotel possessions)*
- enum `StationTiers` { `ONE_STATION` , `TWO_STATIONS` , `THREE_STATIONS` , `FOUR_STATIONS` }  
*Enum describing possible station tiers (how many is owned)*
- enum `UtilityTiers` { `ONE_UTILITY` , `TWO_UTILITIES` }  
*Enum describing possible utility tiers (how many is owned)*
- enum `Decision` { `YES` , `NO` }  
*Enums describing ai adapter decisions.*
- enum `JailDecision` { `ROLL` , `PAY` , `CARD` }
- enum `BuyDecision` { `BUY` , `RESIGN` }

### 7.20.1 Detailed Description

Header file containing structures shared between project files.

#### Author

Kamil Kosnik, Kacper Radzikowski

## 7.20.2 Enumeration Type Documentation

### 7.20.2.1 ActiveScreenType

enum [ActiveScreenType](#)

Enum describing possible screen to be shown.

Enumerator

NONE	
MAIN_MENU	
GAME_MENU	
MONOPOLY_GAME	

### 7.20.2.2 BuyDecision

enum [BuyDecision](#)

Enumerator

BUY	
RESIGN	

### 7.20.2.3 Decision

enum [Decision](#)

Enums describing ai adapter decisions.

Enumerator

YES	
NO	

### 7.20.2.4 FieldType

enum [FieldType](#)

Enum describing possible fields types in monopoly board.

**Enumerator**

STREET	
STATION	
UTILITY	
GO	
CHANCE	
COMMUNITY_CHEST	
TAX	
JAIL	
FREE_PARKING	
GO_TO_JAIL	

**7.20.2.5 GameScreenType**

enum [GameScreenType](#)

Enum describing possible showed screens in game screen object.

**Enumerator**

BOARDGAME	
WITHDRAW_CHOOSE_PLAYER	
WITHDRAW_ADD_VALUE	
WITHDRAW_DECISION	
AUCTION	
RESULT	

**7.20.2.6 JailDecision**

enum [JailDecision](#)

**Enumerator**

ROLL	
PAY	
CARD	

**7.20.2.7 ScreenEventType**

enum [ScreenEventType](#)

Enum describing events that can be returned from showed screens of ActiveScreenType type.

## Enumerator

IDLE	
EXIT	
PLAY	
RETURN_TO_MAIN_MENU	
PLAYER_1_SET_NONE	
PLAYER_2_SET_NONE	
PLAYER_3_SET_NONE	
PLAYER_4_SET_NONE	
PLAYER_1_SET_HUMAN	
PLAYER_2_SET_HUMAN	
PLAYER_3_SET_HUMAN	
PLAYER_4_SET_HUMAN	
PLAYER_1_SET_AI	
PLAYER_2_SET_AI	
PLAYER_3_SET_AI	
PLAYER_4_SET_AI	
PLAYER_1_SET_AI_LEVEL↵ _1	
PLAYER_2_SET_AI_LEVEL↵ _1	
PLAYER_3_SET_AI_LEVEL↵ _1	
PLAYER_4_SET_AI_LEVEL↵ _1	
PLAYER_1_SET_AI_LEVEL↵ _2	
PLAYER_2_SET_AI_LEVEL↵ _2	
PLAYER_3_SET_AI_LEVEL↵ _2	
PLAYER_4_SET_AI_LEVEL↵ _2	
PLAYER_1_SET_AI_LEVEL↵ _3	
PLAYER_2_SET_AI_LEVEL↵ _3	
PLAYER_3_SET_AI_LEVEL↵ _3	
PLAYER_4_SET_AI_LEVEL↵ _3	
START_GAME	
GAME_ENDED	

## 7.20.2.8 StationTiers

```
enum StationTiers
```

Enum describing possible station tiers (how many is owned)



**Enumerator**

ONE_STATION	
TWO_STATIONS	
THREE_STATIONS	
FOUR_STATIONS	

**7.20.2.9 StreetTiers**

enum `StreetTiers`

Enum describing possible street tiers (houses and hotel possessions)

**Enumerator**

NO_HOUSES	
ONE_HOUSE	
TWO_HOUSES	
THREE_HOUSES	
FOUR_HOUSES	
HOTEL	

**7.20.2.10 TurnState**

enum `TurnState`

Enum describing monopoly game states.

**Enumerator**

ROLL_DICE	
FIELD_ACTION	
BUY_ACTION	
PAY_RENT	
TURN_END	
WITHDRAW_ONGOING	
RESULTS	
NO_TURN	

**7.20.2.11 UtilityTiers**

enum `UtilityTiers`

Enum describing possible utility tiers (how many is owned)

Enumerator

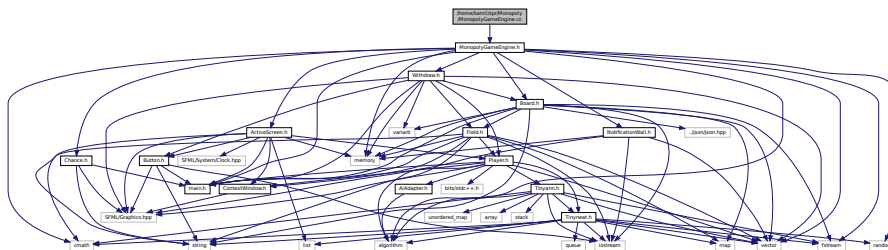
ONE_UTILITY	
TWO_UTILITIES	

## 7.21 /home/kamil/zpr/Monopoly/MonopolyGameEngine.cc File Reference

Source file of class used to handle whole monopoly game process, turns, actions with players, board etc.

```
#include "MonopolyGameEngine.h"
```

Include dependency graph for MonopolyGameEngine.cc:



### 7.21.1 Detailed Description

Source file of class used to handle whole monopoly game process, turns, actions with players, board etc.

Author

Kamil Kosnik, Kacper Radzikowski

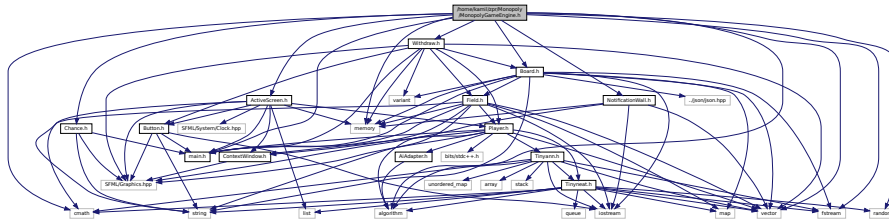
## 7.22 /home/kamil/zpr/Monopoly/MonopolyGameEngine.h File Reference

Header file of class used to handle whole monopoly game process, turns, actions with players, board etc.

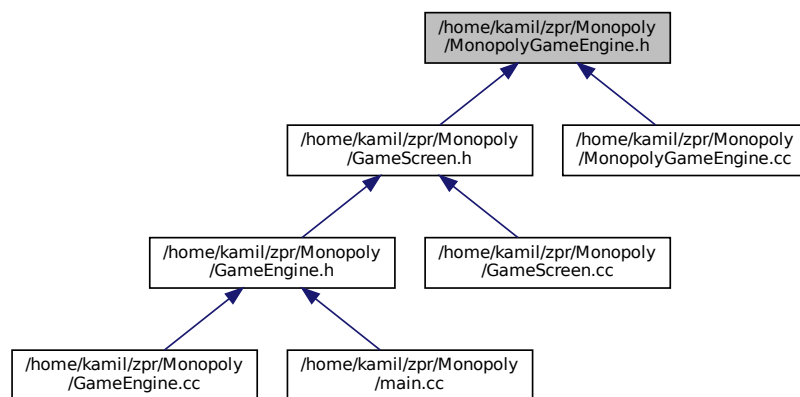
```
#include <algorithm>
#include <cmath>
#include <fstream>
#include <memory>
#include <random>
#include <vector>
#include "ActiveScreen.h"
#include "Board.h"
#include "Chance.h"
#include "NotificationWall.h"
#include "Withdraw.h"
```

```
#include "main.h"
```

Include dependency graph for MonopolyGameEngine.h:



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



## Classes

- class [MonopolyGameEngine](#)  
*Class representing the main game engine for the Monopoly game.*

## Enumerations

- enum [AuctionState](#) {  
  [NO\\_AUCTION](#) , [INITIALIZATION](#) , [PASS\\_BIDDING\\_TURN](#) , [BIDDING](#) ,  
  [ENDING](#) }  
*Enum representing the state of an auction.*

### 7.22.1 Detailed Description

Header file of class used to handle whole monopoly game process, turns, actions with players, board etc.

#### Author

Kamil Kosnik, Kacper Radzikowski

## 7.22.2 Enumeration Type Documentation

### 7.22.2.1 AuctionState

enum [AuctionState](#)

Enum representing the state of an auction.

Enumerator

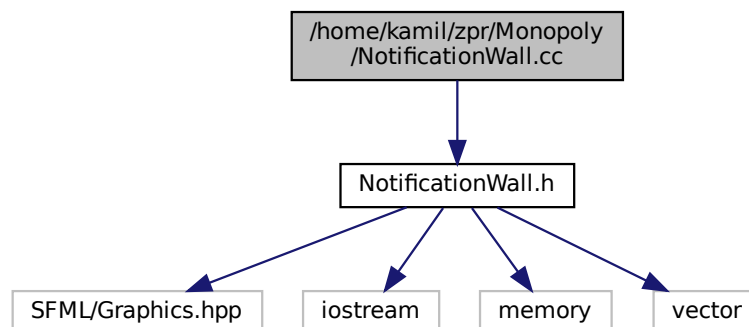
NO_AUCTION	
INITIALIZATION	
PASS_BIDDING_TURN	
BIDDING	
ENDING	

## 7.23 /home/kamil/zpr/Monopoly/NotificationWall.cc File Reference

Source file for the [NotificationWall](#) class.

```
#include "NotificationWall.h"
```

Include dependency graph for NotificationWall.cc:



### 7.23.1 Detailed Description

Source file for the [NotificationWall](#) class.

The [NotificationWall](#) class is used to display a list of messages/notifications as a finite list with a certain length. It automatically rolls over when it receives the next message.

Author

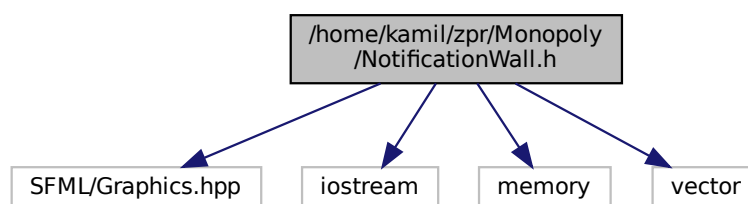
Kamil Kosnik, Kacper Radzikowski

## 7.24 /home/kamil/zpr/Monopoly/NotificationWall.h File Reference

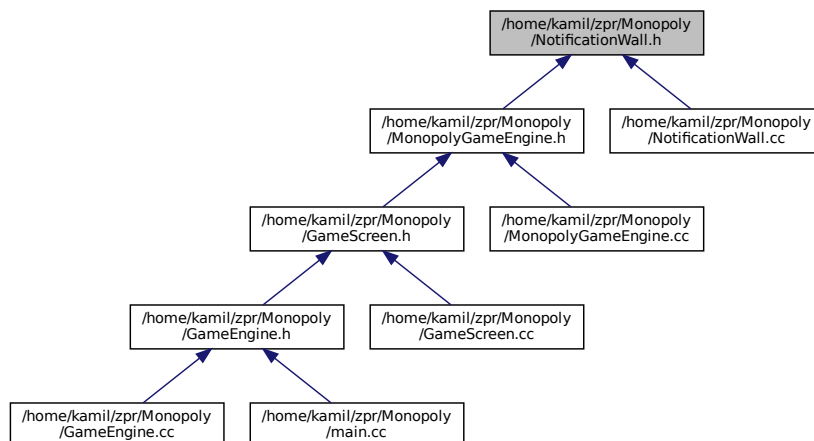
Header file for the [NotificationWall](#) class.

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <memory>
#include <vector>
```

Include dependency graph for NotificationWall.h:



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



### Classes

- class [NotificationWall](#)

*Represents a notification wall that displays messages.*

### 7.24.1 Detailed Description

Header file for the [NotificationWall](#) class.

The [NotificationWall](#) class is used to display a list of messages/notifications as a finite list with a certain length. It automatically rolls over when it receives the next message.

#### Author

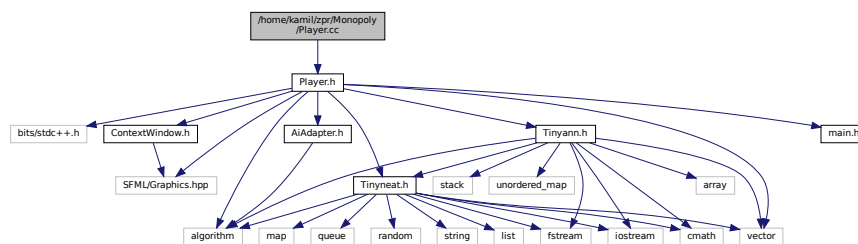
Kamil Kosnik, Kacper Radzikowski

## 7.25 /home/kamil/zpr/Monopoly/Player.cc File Reference

Implementation file for [Player](#) class and AI [Player](#) class, containing data and methods for a player in a Monopoly game.

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

Include dependency graph for Player.cc:



### 7.25.1 Detailed Description

Implementation file for [Player](#) class and AI [Player](#) class, containing data and methods for a player in a Monopoly game.

#### Author

Kamil Kosnik, Kacper Radzikowski

## 7.26 /home/kamil/zpr/Monopoly/Player.h File Reference

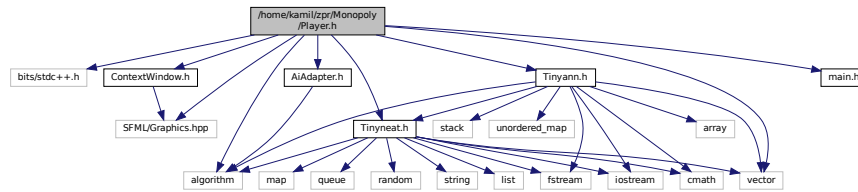
Implementation file for [Player](#) class and AI [Player](#) class, containing data and methods for a player in a Monopoly game.

```

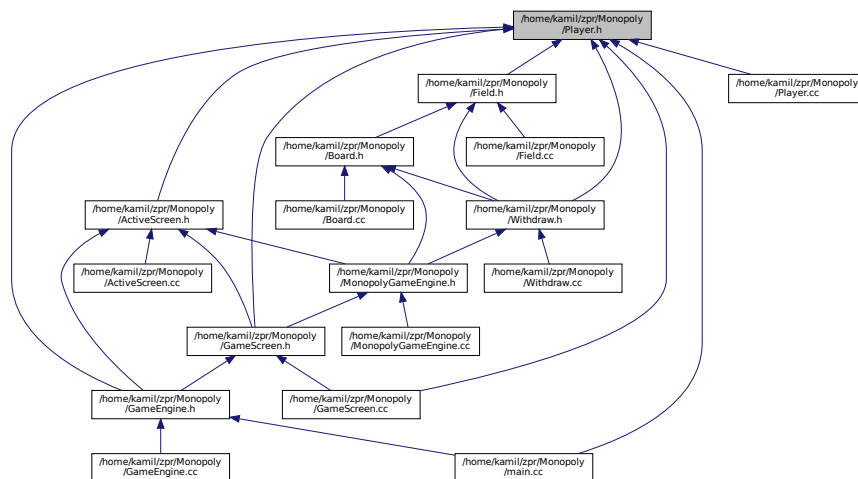
#include <bits/stdc++.h>
#include <SFML/Graphics.hpp>
#include <algorithm>
#include <vector>
#include "AiAdapter.h"
#include "ContextWindow.h"

```

```
#include "Tinyann.h"
#include "Tinyneat.h"
#include "main.h"
Include dependency graph for Player.h:
```



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



## Classes

- class [Player](#)  
*Represents a player in a Monopoly game.*
- class [AiPlayer](#)  
*Represents an AI player in a Monopoly game, inheriting from [Player](#).*

### 7.26.1 Detailed Description

Implementation file for [Player](#) class and AI [Player](#) class, containing data and methods for a player in a Monopoly game.

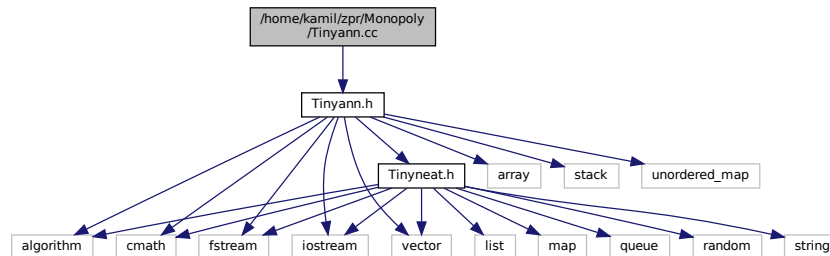
#### Author

Kamil Kosnik, Kacper Radzikowski

## 7.27 /home/kamil/zpr/Monopoly/Tinyann.cc File Reference

```
#include "Tinyann.h"
```

Include dependency graph for Tinyann.cc:

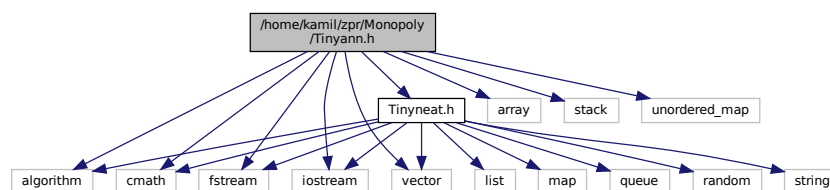


## 7.28 /home/kamil/zpr/Monopoly/Tinyann.h File Reference

Header file for AI implementation of tiny library from github user hav4ik, repository: <https://github.com/hav4ik/tinyai>.

```
#include <algorithm>
#include <array>
#include <cmath>
#include <fstream>
#include <iostream>
#include <stack>
#include <unordered_map>
#include <vector>
#include "Tinyneat.h"
```

Include dependency graph for Tinyann.h:





[illegible]

- class `ann::neuron`
- class `ann::neuralnet`

- ann

- enum `ann::type` { `ann::RECURRENT` , `ann::NON_RECURRENT` }

Header file for AI implementation of tiny library from github user hav4ik, repository: <https://github.com/hav4ik/tinyai>.

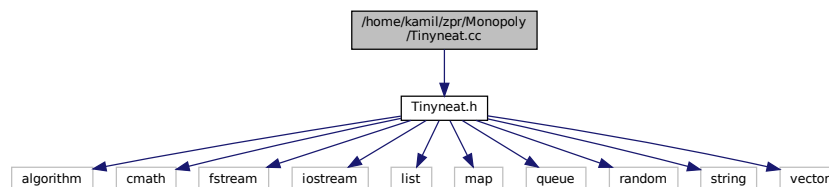
hav4ik

## 7.29 /home/kamil/zpr/Monopoly/Tinyneat.cc File Reference

Source file for AI implementation of tiny library from github user hav4ik, repository: <https://github.com/hav4ik/tinyai>.

```
#include "Tinyneat.h"
```

Include dependency graph for Tinyneat.cc:



### 7.29.1 Detailed Description

Source file for AI implementation of tiny library from github user hav4ik, repository: <https://github.com/hav4ik/tinyai>.

Author

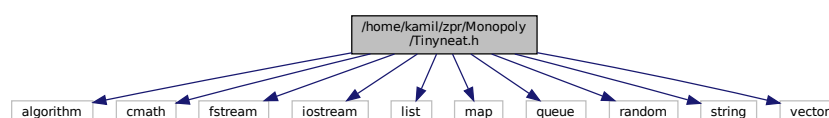
hav4ik

## 7.30 /home/kamil/zpr/Monopoly/Tinyneat.h File Reference

Header file for AI implementation of tiny library from github user hav4ik, repository: <https://github.com/hav4ik/tinyai>.

```
#include <algorithm>
#include <cmath>
#include <fstream>
#include <iostream>
#include <list>
#include <map>
#include <queue>
#include <random>
#include <string>
#include <vector>
```

Include dependency graph for Tinyneat.h:



[illegible]

- struct `neat::mutation_rate_container`
- struct `neat::speciating_parameter_container`
- struct `neat::network_info_container`
- struct `neat::gene`
- class `neat::genome`
- struct `neat::specie`
- class `neat::innovation_container`
- class `neat::pool`

- neat

Header file for AI implementation of tiny library from github user hav4ik, repository: <https://github.com/hav4ik/tinyai>.

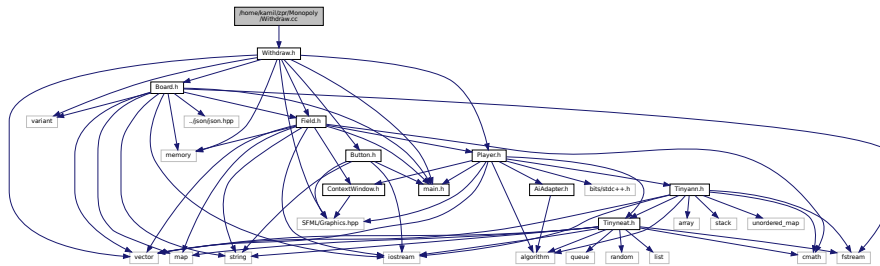
hav4ik

## 7.31 /home/kamil/zpr/Monopoly/Withdraw.cc File Reference

Source file for trade/withdraw mehanism in monopoly game between players.

```
#include "Withdraw.h"
```

Include dependency graph for Withdraw.cc:



### 7.31.1 Detailed Description

Source file for trade/withdraw mehanism in monopoly game between players.

Author

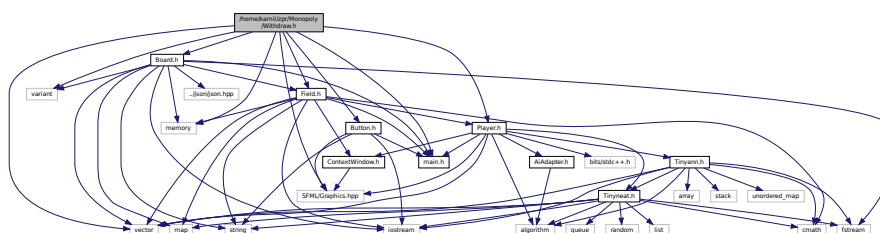
Kamil Kosnik, Kacper Radzikowski

## 7.32 /home/kamil/zpr/Monopoly/Withdraw.h File Reference

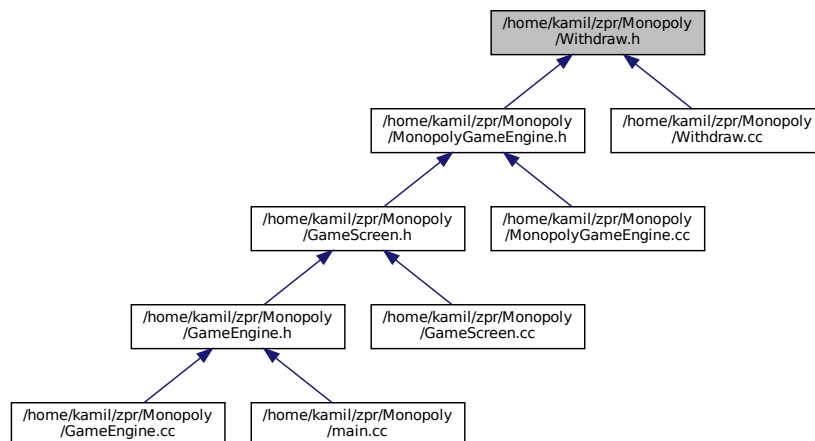
Header file for trade/withdraw mechanism in the monopoly game between players.

```
#include <SFML/Graphics.hpp>
#include <memory>
#include <variant>
#include <vector>
#include "Board.h"
#include "Button.h"
#include "Field.h"
#include "Player.h"
#include "main.h"
```

Include dependency graph for Withdraw.h:



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



## Classes

- class [Withdraw](#)

*Represents the trade and withdraw mechanism in a monopoly game.*

### 7.32.1 Detailed Description

Header file for trade/withdraw mechanism in the monopoly game between players.

This file contains the declaration of the [Withdraw](#) class, which represents the trade and withdraw mechanism in a monopoly game between players. [Player 1](#) - active in withdraw (offering one) [Player 2](#) - passive in withdraw (decision maker)

#### Author

Kamil Kosnik, Kacper Radzikowski



# Index

- /home/kamil/zpr/Monopoly/ActiveScreen.cc, [271](#)
- /home/kamil/zpr/Monopoly/ActiveScreen.h, [272](#)
- /home/kamil/zpr/Monopoly/AiAdapter.cc, [273](#)
- /home/kamil/zpr/Monopoly/AiAdapter.h, [274](#)
- /home/kamil/zpr/Monopoly/Board.cc, [275](#)
- /home/kamil/zpr/Monopoly/Board.h, [277](#)
- /home/kamil/zpr/Monopoly/Button.cc, [281](#)
- /home/kamil/zpr/Monopoly/Button.h, [282](#)
- /home/kamil/zpr/Monopoly/Chance.cc, [283](#)
- /home/kamil/zpr/Monopoly/Chance.h, [284](#)
- /home/kamil/zpr/Monopoly/ContextWindow.cc, [285](#)
- /home/kamil/zpr/Monopoly/ContextWindow.h, [286](#)
- /home/kamil/zpr/Monopoly/Field.cc, [287](#)
- /home/kamil/zpr/Monopoly/Field.h, [288](#)
- /home/kamil/zpr/Monopoly/GameEngine.cc, [290](#)
- /home/kamil/zpr/Monopoly/GameEngine.h, [290](#)
- /home/kamil/zpr/Monopoly/GameScreen.cc, [291](#)
- /home/kamil/zpr/Monopoly/GameScreen.h, [292](#)
- /home/kamil/zpr/Monopoly/MonopolyGameEngine.cc, [302](#)
- /home/kamil/zpr/Monopoly/MonopolyGameEngine.h, [302](#)
- /home/kamil/zpr/Monopoly/NotificationWall.cc, [304](#)
- /home/kamil/zpr/Monopoly/NotificationWall.h, [305](#)
- /home/kamil/zpr/Monopoly/Player.cc, [306](#)
- /home/kamil/zpr/Monopoly/Player.h, [306](#)
- /home/kamil/zpr/Monopoly/Tinyann.cc, [308](#)
- /home/kamil/zpr/Monopoly/Tinyann.h, [308](#)
- /home/kamil/zpr/Monopoly/Tinyneat.cc, [310](#)
- /home/kamil/zpr/Monopoly/Tinyneat.h, [310](#)
- /home/kamil/zpr/Monopoly/Withdraw.cc, [312](#)
- /home/kamil/zpr/Monopoly/Withdraw.h, [312](#)
- /home/kamil/zpr/Monopoly/main.cc, [293](#)
- /home/kamil/zpr/Monopoly/main.h, [295](#)
- ~neuron
  - ann::neuron, [182](#)
- ActiveScreen, [11](#)
  - ActiveScreen, [12](#)
  - addButton, [12](#)
  - addText, [13](#)
  - buttonSetColors, [13](#)
  - draw, [14](#)
  - getButtons, [14](#)
  - getContextWindow, [14](#)
  - getFont, [15](#)
  - getPlayersResult, [15](#)
  - getPlayersSettings, [16](#)
  - getScreenType, [16](#)
  - getTexts, [16](#)
  - setContextWindow, [17](#)
  - setFont, [17](#)
  - setScreenType, [18](#)
  - worker, [19](#)
- ActiveScreenType
  - main.h, [297](#)
- add\_gene
  - neat::innovation\_container, [104](#)
- addAuctionButton
  - MonopolyGameEngine, [112](#)
- addAuctionText
  - MonopolyGameEngine, [113](#)
- addButton
  - ActiveScreen, [12](#)
  - MonopolyGameEngine, [114](#)
- addFieldOwnedId
  - Player, [190](#)
- addHotels
  - MonopolyGameEngine, [115](#)
- addHouses
  - MonopolyGameEngine, [115](#)
- addMoney
  - Player, [191](#)
- addPropertyPlayerShown
  - Withdraw, [242](#)
- addText
  - ActiveScreen, [13](#)
  - MonopolyGameEngine, [116](#)
- addToWall
  - NotificationWall, [184](#)
- adjusted\_fitness
  - neat::genome, [101](#)
- AiAdapter, [19](#)
  - AiAdapter, [20](#)
  - clearSelectionState, [21](#)
  - convertCard, [21](#)
  - convertHouse, [21](#)
  - convertHouseValue, [22](#)
  - convertMoney, [23](#)
  - convertMoneyValue, [23](#)
  - convertPosition, [24](#)
  - getInputs, [24](#)
  - setCard, [25](#)
  - setHouse, [26](#)
  - setJail, [26](#)
  - setMoney, [27](#)
  - setMoneyContext, [27](#)
  - setMortgage, [28](#)
  - setOwner, [28](#)

- setPosition, 28
- setSelection, 29
- setSelectionState, 29
- setTurn, 29
- aiBuildingsManagingWorker
  - MonopolyGameEngine, 117
- AiPlayer, 30
  - AiPlayer, 31, 32
  - decideAcceptTrade, 32
  - decideAuctionBid, 33
  - decideBuildHouse, 34
  - decideBuy, 34
  - decideJail, 35
  - decideMortgage, 36
  - decideOfferTrade, 37
  - decideSellHouse, 37
  - decideUnmortgage, 38
  - getAdapter, 39
  - getNeuralNetwork, 39
  - setNeuralNetwork, 39
- ann, 9
  - NON\_RECURRENT, 9
  - RECURRENT, 9
  - type, 9
- ann::neuralnet, 180
  - evaluate, 180
  - export\_tofile, 181
  - from\_genome, 181
  - import\_fromfile, 181
  - neuralnet, 180
- ann::neuron, 182
  - ~neuron, 182
  - in\_nodes, 183
  - neuron, 182
  - type, 183
  - value, 183
  - visited, 183
- AUCTION
  - main.h, 299
- AuctionState
  - MonopolyGameEngine.h, 304
- average\_fitness
  - neat::specie, 222
- BANK\_PAYS\_YOU
  - Chance.h, 285
- bias\_mutation\_chance
  - neat::mutation\_rate\_container, 177
- bias\_size
  - neat::network\_info\_container, 179
- BIDDING
  - MonopolyGameEngine.h, 304
- Board, 40
  - Board, 40
  - clearBoard, 41
  - getBoard, 41
  - getBoardPosition, 41
  - getFieldById, 42
  - getFieldNumber, 42
  - getFieldPosition, 42
  - getFieldRotation, 43
- Board.cc
  - json, 275
  - jsonToStationRent, 276
  - jsonToStreetRent, 276
  - jsonToUtilityRent, 277
- Board.h
  - json, 279
  - jsonToStationRent, 279
  - jsonToStreetRent, 280
  - jsonToUtilityRent, 280
  - PossibleFields, 279
- BOARDGAME
  - main.h, 299
- buildingsManagingWorker
  - MonopolyGameEngine, 118
- Button, 44
  - Button, 45
  - draw, 47
  - getEventType, 47
  - getIsActive, 47
  - getIsClicked, 47
  - getIsFocus, 48
  - getIsVisible, 48
  - getSize, 48
  - getText, 48
  - getWasReleased, 49
  - isMouseOver, 49
  - mouseIsNotOver, 49
  - mouseIsOver, 50
  - setActiveBackColor, 50
  - setActiveTextColor, 50
  - setButtonFocus, 51
  - setButtonUnfocus, 51
  - setEventType, 52
  - setFocusBackColor, 52
  - setFocusTextColor, 52
  - setFont, 53
  - setInactiveBackColor, 53
  - setInactiveTextColor, 53
  - setIsActive, 53
  - setIsClicked, 54
  - setIsFocus, 54
  - setIsVisible, 54
  - setPosition, 55
  - setWasReleased, 55
  - updateColors, 55
- buttonClickHandle
  - GameMenuScreen, 85
- buttonSetColors
  - ActiveScreen, 13
- BUY
  - main.h, 297
- BUY\_ACTION
  - main.h, 301
- BuyDecision
  - main.h, 297



- calculateGroupFieldsOwned
  - MonopolyGameEngine, 119
- calculateRent
  - MonopolyGameEngine, 120
  - StationField, 226
  - StreetField, 229
  - UtilityField, 237
- can\_be\_recurrent
  - neat::genome, 101
- CARD
  - main.h, 299
- CHANGE
  - main.h, 299
- Chance.h
  - BANK\_PAYS\_YOU, 285
  - ChanceType, 285
  - GET\_OUT\_OF\_JAIL\_CARD, 285
  - GO\_TO\_JAIL\_CARD, 285
  - MOVEMENT\_SPACES, 285
  - MOVEMENT\_TO\_PROPERTY, 285
  - MOVEMENT\_WITH\_BUY\_OR\_PAY, 285
  - PAY\_FOR\_HOUSE\_HOTEL, 285
  - PAY\_PLAYERS, 285
  - TAX\_CARD, 285
- ChanceCard, 55
  - ChanceCard, 56
  - createSprite, 58
  - getGraphicPath, 58
  - getHeight, 58
  - getId, 58
  - getPosition, 59
  - getRotation, 59
  - getSprite, 59
  - getText, 59
  - getTexture, 60
  - getType, 60
  - getValue, 60
  - getWidth, 61
  - setPosition, 61
  - setRotation, 61
- ChanceType
  - Chance.h, 285
- clear
  - ContextWindow, 63
  - GameEngine, 79
- clearBoard
  - Board, 41
  - MonopolyGameEngine, 121
- clearFieldOwnedId
  - Player, 191
- clearPlayers
  - MonopolyGameEngine, 121
- clearSelectionState
  - AiAdapter, 21
- clearWall
  - NotificationWall, 185
- colorGroupEmpty
  - MonopolyGameEngine, 122
- COMMUNITY\_CHEST
  - main.h, 299
- connection\_mutate\_chance
  - neat::mutation\_rate\_container, 178
- ContextWindow, 62
  - clear, 63
  - ContextWindow, 63
  - display, 63
  - GetInstance, 64
  - getView, 64
  - getWindow, 65
  - isOpen, 65
  - operator=, 66
  - view\_, 66
  - window\_, 66
- convertCard
  - AiAdapter, 21
- convertHouse
  - AiAdapter, 21
- convertHouseValue
  - AiAdapter, 22
- convertMoney
  - AiAdapter, 23
- convertMoneyValue
  - AiAdapter, 23
- convertPosition
  - AiAdapter, 24
- createAuctionBidButton
  - MonopolyGameEngine, 123
- createAuctionOfferButtons
  - MonopolyGameEngine, 123
- createAuctionResignButton
  - MonopolyGameEngine, 124
- createAvailableHousesHotelText
  - MonopolyGameEngine, 125
- createBoard
  - MonopolyGameEngine, 126
- createButtonBuyResign
  - MonopolyGameEngine, 126
- createButtonNextProperty
  - MonopolyGameEngine, 127
- createButtonPerviousProperty
  - MonopolyGameEngine, 128
- createButtonRollDice
  - MonopolyGameEngine, 129
- createButtonsBankrupt
  - MonopolyGameEngine, 129
- createButtonsBuySellHouseHotel
  - MonopolyGameEngine, 130
- createButtonsJailPay
  - MonopolyGameEngine, 131
- createButtonsNextTurn
  - MonopolyGameEngine, 132
- createButtonWithdraw
  - MonopolyGameEngine, 133
- createChanceCards
  - MonopolyGameEngine, 134
- createChoosePlayerScreen

- Withdraw, [242](#)
- createCurrentOfferBidderInfo
  - MonopolyGameEngine, [135](#)
- createDecisionPlayerScreen
  - Withdraw, [243](#)
- createFlagSprite
  - PropertyField, [212](#)
- createMortgagingButton
  - MonopolyGameEngine, [135](#)
- createPlayers
  - MonopolyGameEngine, [136](#)
- createResultScreenStuff
  - MonopolyGameEngine, [137](#)
- createSprite
  - ChanceCard, [58](#)
  - Field, [70](#)
  - Player, [191](#)
- createTextBiddedProperty
  - MonopolyGameEngine, [138](#)
- createTextBidderInfo
  - MonopolyGameEngine, [138](#)
- createTextHighestBidInfo
  - MonopolyGameEngine, [139](#)
- createTextLeadingBidderInfo
  - MonopolyGameEngine, [140](#)
- createTextPlayersInfo
  - MonopolyGameEngine, [141](#)
- createTextrolled\_value
  - MonopolyGameEngine, [142](#)
- createTextTurnInfo
  - MonopolyGameEngine, [142](#)
- createValuePlayerScreen
  - Withdraw, [244](#)
- crossover\_chance
  - neat::mutation\_rate\_container, [178](#)
- decideAcceptTrade
  - AiPlayer, [32](#)
  - Player, [192](#)
- decideAuctionBid
  - AiPlayer, [33](#)
  - Player, [192](#)
- decideBuildHouse
  - AiPlayer, [34](#)
  - Player, [192](#)
- decideBuy
  - AiPlayer, [34](#)
  - Player, [192](#)
- decideJail
  - AiPlayer, [35](#)
  - Player, [193](#)
- decideMortgage
  - AiPlayer, [36](#)
  - Player, [193](#)
- decideOfferTrade
  - AiPlayer, [37](#)
  - Player, [194](#)
- decideSellHouse
  - AiPlayer, [37](#)
- Player, [194](#)
- decideUnmortgage
  - AiPlayer, [38](#)
  - Player, [194](#)
- Decision
  - main.h, [297](#)
- delta\_disjoint
  - neat::speciating\_parameter\_container, [221](#)
- delta\_threshold
  - neat::speciating\_parameter\_container, [221](#)
- delta\_weights
  - neat::speciating\_parameter\_container, [221](#)
- DimensionException, [67](#)
  - DimensionException, [67](#), [68](#)
  - getBadDimension, [68](#)
- disable\_mutation\_chance
  - neat::mutation\_rate\_container, [178](#)
- display
  - ContextWindow, [63](#)
  - GameEngine, [80](#)
- draw
  - ActiveScreen, [14](#)
  - Button, [47](#)
  - GameMenuScreen, [86](#)
  - GameScreen, [95](#)
  - MainMenuScreen, [106](#)
- enable\_mutation\_chance
  - neat::mutation\_rate\_container, [178](#)
- enabled
  - neat::gene, [99](#)
- ENDING
  - MonopolyGameEngine.h, [304](#)
- evaluate
  - ann::neuralnet, [180](#)
- EXIT
  - main.h, [300](#)
- export\_tofile
  - ann::neuralnet, [181](#)
  - neat::pool, [208](#)
- Field, [68](#)
  - createSprite, [70](#)
  - Field, [69](#)
  - getContextWindow, [70](#)
  - getGraphicPath, [71](#)
  - getHeight, [71](#)
  - getId, [72](#)
  - getName, [72](#)
  - getNameText, [73](#)
  - getPosition, [73](#)
  - getRotation, [74](#)
  - getSprite, [74](#)
  - getTexture, [75](#)
  - getType, [75](#)
  - getWidth, [76](#)
  - setHeight, [76](#)
  - setPosition, [76](#)
  - setRotation, [77](#)

- setWidth, 77
- FIELD\_ACTION
  - main.h, 301
- FieldType
  - main.h, 297
- fitness
  - neat::genome, 101
- FOUR\_HOUSES
  - main.h, 301
- FOUR\_STATIONS
  - main.h, 301
- FRAMES\_PER\_SEC\_MAX
  - main.cc, 294
- FREE\_PARKING
  - main.h, 299
- from\_genome
  - ann::neuralnet, 181
- from\_node
  - neat::gene, 99
- functional\_nodes
  - neat::network\_info\_container, 179
- GAME\_ENDED
  - main.h, 300
- GAME\_MENU
  - main.h, 297
- GameEngine, 77
  - clear, 79
  - display, 80
  - GameEngine, 78, 79
  - getContextWindow, 81
  - getWindow\_height, 81
  - getWindow\_width, 81
  - pollForEvents, 82
  - worker, 82
- gameMenuCreate
  - GameMenuScreen, 87
- GameMenuScreen, 83
  - buttonClickHandle, 85
  - draw, 86
  - gameMenuCreate, 87
  - GameMenuScreen, 85
  - getPlayerNumFromEventType, 87
  - getPlayersSettings, 88
  - isEventTypeAiLevel, 88
  - isEventTypeSetAi, 89
  - setAiLevelColumnVisibility, 89
  - setDefaultAiLevelButtonsFocus, 90
  - setOtherButtonsInactive, 91
  - setPlayerSettings, 92
  - worker, 92
- GAMES\_IN\_ROUND
  - main.cc, 294
- GameScreen, 93
  - draw, 95
  - GameScreen, 94
  - getPlayersResult, 96
  - worker, 97
- GameScreenType
  - main.h, 299
- generation
  - neat::pool, 208
- generator
  - neat::pool, 209
- genes
  - neat::genome, 101
- genome
  - neat::genome, 100
- genomes
  - neat::specie, 222
- get\_genomes
  - neat::pool, 208
- GET\_OUT\_OF\_JAIL\_CARD
  - Chance.h, 285
- getAcceptDecisionButton
  - Withdraw, 244
- getAdapter
  - AiPlayer, 39
  - Player, 195
- getAiLevel
  - Player, 195
- getAllPropertyDataSprite
  - MonopolyGameEngine, 143
- getAllPropertyDataTexts
  - MonopolyGameEngine, 143
- getAuctionButtons
  - MonopolyGameEngine, 144
- getAuctionState
  - MonopolyGameEngine, 144
- getAuctionTexts
  - MonopolyGameEngine, 145
- getBadDimension
  - DimensionException, 68
- getBadOffset
  - SpriteOffsetException, 224
- getBadRotation
  - RotationException, 220
- getBoard
  - Board, 41
  - MonopolyGameEngine, 145
- getBoardPosition
  - Board, 41
- getButtons
  - ActiveScreen, 14
  - MonopolyGameEngine, 146
  - Withdraw, 244
- getColor
  - Player, 195
- getContextWindow
  - ActiveScreen, 14
  - Field, 70
  - GameEngine, 81
- getEventType
  - Button, 47
- getFieldById
  - Board, 42
- getFieldNumber

- Board, [42](#)
- getFieldOwnedId
  - Player, [196](#)
- getFieldPositon
  - Board, [42](#)
- getFieldRotation
  - Board, [43](#)
- getFont
  - ActiveScreen, [15](#)
  - MonopolyGameEngine, [146](#)
  - NotificationWall, [185](#)
  - Withdraw, [245](#)
- getFontSize
  - MonopolyGameEngine, [148](#)
  - NotificationWall, [186](#)
  - Withdraw, [245](#)
- getGraphicPath
  - ChanceCard, [58](#)
  - Field, [71](#)
- getGroupMembers
  - PropertyField, [213](#)
- getHeight
  - ChanceCard, [58](#)
  - Field, [71](#)
- getHotelCount
  - MonopolyGameEngine, [149](#)
- getHotelPrice
  - StreetField, [230](#)
- getHotelSprite
  - MonopolyGameEngine, [150](#)
- getHotelTexture
  - MonopolyGameEngine, [151](#)
- getHouseCount
  - MonopolyGameEngine, [151](#)
- getHouseNumber
  - StreetField, [230](#)
- getHousePrice
  - StreetField, [230](#)
- getHouseSize
  - MonopolyGameEngine, [152](#)
- getHouseSprite
  - MonopolyGameEngine, [152](#)
- getHouseTexture
  - MonopolyGameEngine, [153](#)
- getId
  - ChanceCard, [58](#)
  - Field, [72](#)
  - Player, [196](#)
- getInputs
  - AiAdapter, [24](#)
- GetInstance
  - ContextWindow, [64](#)
- getInvalidNumber
  - HouseException, [103](#)
- getIsActive
  - Button, [47](#)
- getIsAi
  - Player, [196](#)
- getIsClicked
  - Button, [47](#)
- getIsFocus
  - Button, [48](#)
- getIsHotel
  - StreetField, [231](#)
- getIsMortgaged
  - PropertyField, [213](#)
- getIsVisible
  - Button, [48](#)
- getJailCards
  - Player, [196](#)
- getJailStatus
  - Player, [196](#)
- getMoney
  - Player, [197](#)
- getMortgage
  - PropertyField, [214](#)
- getName
  - Field, [72](#)
- getNameText
  - Field, [73](#)
- getNeuralNetwork
  - AiPlayer, [39](#)
  - Player, [197](#)
- getNotificationsWall
  - MonopolyGameEngine, [154](#)
- getOwner
  - PropertyField, [214](#)
- getOwnerFlag
  - PropertyField, [215](#)
- getPlayer1AddButton
  - Withdraw, [246](#)
- getPlayer1Button
  - Withdraw, [246](#)
- getPlayer1IndexNextButton
  - Withdraw, [246](#)
- getPlayer1IndexPreviousButton
  - Withdraw, [247](#)
- getPlayer1IndexProperties
  - Withdraw, [247](#)
- getPlayer1minus1
  - Withdraw, [247](#)
- getPlayer1minus10
  - Withdraw, [247](#)
- getPlayer1minus100
  - Withdraw, [248](#)
- getPlayer1NextButton
  - Withdraw, [248](#)
- getPlayer1plus1
  - Withdraw, [248](#)
- getPlayer1plus10
  - Withdraw, [248](#)
- getPlayer1plus100
  - Withdraw, [249](#)
- getPlayer1PreviousButton
  - Withdraw, [249](#)
- getPlayer1RemoveButton

- Withdraw, [249](#)
- getPlayer1ToWithdraw
  - Withdraw, [249](#)
- getPlayer2AddButton
  - Withdraw, [250](#)
- getPlayer2Button
  - Withdraw, [250](#)
- getPlayer2IndexNextButton
  - Withdraw, [250](#)
- getPlayer2IndexPreviousButton
  - Withdraw, [250](#)
- getPlayer2IndexProperties
  - Withdraw, [251](#)
- getPlayer2minus1
  - Withdraw, [251](#)
- getPlayer2minus10
  - Withdraw, [251](#)
- getPlayer2minus100
  - Withdraw, [251](#)
- getPlayer2NextButton
  - Withdraw, [252](#)
- getPlayer2plus1
  - Withdraw, [252](#)
- getPlayer2plus10
  - Withdraw, [252](#)
- getPlayer2plus100
  - Withdraw, [252](#)
- getPlayer2PreviousButton
  - Withdraw, [253](#)
- getPlayer2RemoveButton
  - Withdraw, [253](#)
- getPlayer2ToWithdraw
  - Withdraw, [253](#)
- getPlayer3Button
  - Withdraw, [253](#)
- getPlayer4Button
  - Withdraw, [254](#)
- getPlayerNumFromEventType
  - GameMenuScreen, [87](#)
- getPlayers
  - MonopolyGameEngine, [154](#)
- getPlayersResult
  - ActiveScreen, [15](#)
  - GameScreen, [96](#)
  - MonopolyGameEngine, [155](#)
- getPlayersSettings
  - ActiveScreen, [16](#)
  - GameMenuScreen, [88](#)
- getPosition
  - ChanceCard, [59](#)
  - Field, [73](#)
  - Player, [197](#)
- getPrice
  - PropertyField, [215](#)
- getPropertyDataSprite
  - MonopolyGameEngine, [155](#)
- getPropertyDataTexts
  - MonopolyGameEngine, [156](#)
- getPropertyNameToDraw
  - MonopolyGameEngine, [156](#)
- getRentMultipliers
  - UtilityField, [238](#)
- getRentValues
  - StationField, [226](#)
  - StreetField, [231](#)
- getResignButton
  - Withdraw, [254](#)
- getResignDecisionButton
  - Withdraw, [254](#)
- getResignValueButton
  - Withdraw, [254](#)
- getResultPlace
  - Player, [198](#)
- getResultTexts
  - MonopolyGameEngine, [157](#)
- getRotation
  - ChanceCard, [59](#)
  - Field, [74](#)
- getScreenType
  - ActiveScreen, [16](#)
  - MonopolyGameEngine, [158](#)
- getSize
  - Button, [48](#)
- getSprite
  - ChanceCard, [59](#)
  - Field, [74](#)
  - Player, [198](#)
- getSpriteOffsetX
  - Player, [198](#)
- getSpriteOffsetY
  - Player, [198](#)
- getSpritePropertyPlayer1
  - Withdraw, [255](#)
- getSpritePropertyPlayer1Index
  - Withdraw, [255](#)
- getSpritePropertyPlayer2
  - Withdraw, [255](#)
- getSpritePropertyPlayer2Index
  - Withdraw, [256](#)
- getSubmitValueButton
  - Withdraw, [256](#)
- getTaxValue
  - TaxField, [235](#)
- getText
  - Button, [48](#)
  - ChanceCard, [59](#)
- getTexts
  - ActiveScreen, [16](#)
  - MonopolyGameEngine, [158](#)
  - Withdraw, [257](#)
- getTextsPropertyPlayer1
  - Withdraw, [257](#)
- getTextsPropertyPlayer1Index
  - Withdraw, [257](#)
- getTextsPropertyPlayer2
  - Withdraw, [258](#)

- getTextsPropertyPlayer2Index
  - Withdraw, 258
- getTexture
  - ChanceCard, 60
  - Field, 75
  - Player, 198
- getTexturePropertyPlayer1
  - Withdraw, 259
- getTexturePropertyPlayer1Index
  - Withdraw, 259
- getTexturePropertyPlayer2
  - Withdraw, 260
- getTexturePropertyPlayer2Index
  - Withdraw, 260
- getTurnState
  - MonopolyGameEngine, 159
  - Withdraw, 261
- getType
  - ChanceCard, 60
  - Field, 75
- getUnmortgageValue
  - PropertyField, 216
- getValue
  - ChanceCard, 60
- getView
  - ContextWindow, 64
- getWall
  - NotificationWall, 186
- getWasReleased
  - Button, 49
- getWidth
  - ChanceCard, 61
  - Field, 76
- getWindow
  - ContextWindow, 65
- getwindow\_height
  - GameEngine, 81
- getwindow\_width
  - GameEngine, 81
- getWithdraw
  - MonopolyGameEngine, 159
- global\_rank
  - neat::genome, 101
- GO
  - main.h, 299
- GO\_TO\_JAIL
  - main.h, 299
- GO\_TO\_JAIL\_CARD
  - Chance.h, 285
- groupCompleted
  - MonopolyGameEngine, 160
- hasFieldOwnedId
  - Player, 198
- height
  - main.cc, 294
- HEIGHT\_MAX
  - main.cc, 294
- HOTEL
  - main.h, 301
- HouseException, 102
  - getInvalidNumber, 103
  - HouseException, 103
- IDLE
  - main.h, 300
- import\_fromfile
  - ann::neuralnet, 181
  - neat::pool, 208
- in\_nodes
  - ann::neuron, 183
- INITIALIZATION
  - MonopolyGameEngine.h, 304
- innovation\_container
  - neat::innovation\_container, 104
- innovation\_num
  - neat::gene, 99
- input\_size
  - neat::network\_info\_container, 179
- isBuildingLegal
  - MonopolyGameEngine, 161
- isDestroyingLegal
  - MonopolyGameEngine, 162
- isEventTypeAllLevel
  - GameMenuScreen, 88
- isEventTypeSetAI
  - GameMenuScreen, 89
- isHotelBuildingLegal
  - MonopolyGameEngine, 163
- isHotelDestroyingLegal
  - MonopolyGameEngine, 164
- isHuman
  - playerSettings, 206
- isMouseOver
  - Button, 49
- isNone
  - playerSettings, 206
- isNonZeroValue
  - Withdraw, 261
- isOpen
  - ContextWindow, 65
- JAIL
  - main.h, 299
- JailDecision
  - main.h, 299
- json
  - Board.cc, 275
  - Board.h, 279
- jsonToStationRent
  - Board.cc, 276
  - Board.h, 279
- jsonToStreetRent
  - Board.cc, 276
  - Board.h, 280
- jsonToUtilityRent
  - Board.cc, 277
  - Board.h, 280

- level
  - playerSettings, 206
- link\_mutation\_chance
  - neat::mutation\_rate\_container, 178
- main
  - main.cc, 294
- main.cc
  - FRAMES\_PER\_SEC\_MAX, 294
  - GAMES\_IN\_ROUND, 294
  - height, 294
  - HEIGHT\_MAX, 294
  - main, 294
  - TRAIN, 294
  - width, 294
  - WIDTH\_MAX, 295
- main.h
  - ActiveScreenType, 297
  - AUCTION, 299
  - BOARDGAME, 299
  - BUY, 297
  - BUY\_ACTION, 301
  - BuyDecision, 297
  - CARD, 299
  - CHANCE, 299
  - COMMUNITY\_CHEST, 299
  - Decision, 297
  - EXIT, 300
  - FIELD\_ACTION, 301
  - FieldType, 297
  - FOUR\_HOUSES, 301
  - FOUR\_STATIONS, 301
  - FREE\_PARKING, 299
  - GAME\_ENDED, 300
  - GAME\_MENU, 297
  - GameScreenType, 299
  - GO, 299
  - GO\_TO\_JAIL, 299
  - HOTEL, 301
  - IDLE, 300
  - JAIL, 299
  - JailDecision, 299
  - MAIN\_MENU, 297
  - MONOPOLY\_GAME, 297
  - NO, 297
  - NO\_HOUSES, 301
  - NO\_TURN, 301
  - NONE, 297
  - ONE\_HOUSE, 301
  - ONE\_STATION, 301
  - ONE\_UTILITY, 302
  - PAY, 299
  - PAY\_RENT, 301
  - PLAY, 300
  - PLAYER\_1\_SET\_AI, 300
  - PLAYER\_1\_SET\_AI\_LEVEL\_1, 300
  - PLAYER\_1\_SET\_AI\_LEVEL\_2, 300
  - PLAYER\_1\_SET\_AI\_LEVEL\_3, 300
  - PLAYER\_1\_SET\_HUMAN, 300
  - PLAYER\_1\_SET\_NONE, 300
  - PLAYER\_2\_SET\_AI, 300
  - PLAYER\_2\_SET\_AI\_LEVEL\_1, 300
  - PLAYER\_2\_SET\_AI\_LEVEL\_2, 300
  - PLAYER\_2\_SET\_AI\_LEVEL\_3, 300
  - PLAYER\_2\_SET\_HUMAN, 300
  - PLAYER\_2\_SET\_NONE, 300
  - PLAYER\_3\_SET\_AI, 300
  - PLAYER\_3\_SET\_AI\_LEVEL\_1, 300
  - PLAYER\_3\_SET\_AI\_LEVEL\_2, 300
  - PLAYER\_3\_SET\_AI\_LEVEL\_3, 300
  - PLAYER\_3\_SET\_HUMAN, 300
  - PLAYER\_3\_SET\_NONE, 300
  - PLAYER\_4\_SET\_AI, 300
  - PLAYER\_4\_SET\_AI\_LEVEL\_1, 300
  - PLAYER\_4\_SET\_AI\_LEVEL\_2, 300
  - PLAYER\_4\_SET\_AI\_LEVEL\_3, 300
  - PLAYER\_4\_SET\_HUMAN, 300
  - PLAYER\_4\_SET\_NONE, 300
  - RESIGN, 297
  - RESULT, 299
  - RESULTS, 301
  - RETURN\_TO\_MAIN\_MENU, 300
  - ROLL, 299
  - ROLL\_DICE, 301
  - ScreenEventType, 299
  - START\_GAME, 300
  - STATION, 299
  - StationTiers, 300
  - STREET, 299
  - StreetTiers, 301
  - TAX, 299
  - THREE\_HOUSES, 301
  - THREE\_STATIONS, 301
  - TURN\_END, 301
  - TurnState, 301
  - TWO\_HOUSES, 301
  - TWO\_STATIONS, 301
  - TWO\_UTILITIES, 302
  - UTILITY, 299
  - UtilityTiers, 301
  - WITHDRAW\_ADD\_VALUE, 299
  - WITHDRAW\_CHOOSE\_PLAYER, 299
  - WITHDRAW\_DECISION, 299
  - WITHDRAW\_ONGOING, 301
  - YES, 297
- MAIN\_MENU
  - main.h, 297
- mainMenuCreate
  - MainMenuScreen, 107
- MainMenuScreen, 105
  - draw, 106
  - mainMenuCreate, 107
  - MainMenuScreen, 106
  - worker, 108
- makeWithdraw
  - Withdraw, 261
- max\_fitness

- neat::pool, 209
- max\_neuron
  - neat::genome, 101
- moneyTextUpdate
  - Withdraw, 262
- moneyTransferIndex
  - Withdraw, 262
- MONOPOLY\_GAME
  - main.h, 297
- MonopolyGameEngine, 108
  - addAuctionButton, 112
  - addAuctionText, 113
  - addButton, 114
  - addHotels, 115
  - addHouses, 115
  - addText, 116
  - aiBuildingsMangingWorker, 117
  - buildingsManagingWorker, 118
  - calculateGroupFieldsOwned, 119
  - calculateRent, 120
  - clearBoard, 121
  - clearPlayers, 121
  - colorGroupEmpty, 122
  - createAuctionBidButton, 123
  - createAuctionOfferButtons, 123
  - createAuctionResignButton, 124
  - createAvailableHousesHotelText, 125
  - createBoard, 126
  - createButtonBuyResign, 126
  - createButtonNextProperty, 127
  - createButtonPerviousProperty, 128
  - createButtonRollDice, 129
  - createButtonsBankrupt, 129
  - createButtonsBuySellHouseHotel, 130
  - createButtonsJailPay, 131
  - createButtonsNextTurn, 132
  - createButtonWithdraw, 133
  - createChanceCards, 134
  - createCurrentOfferBidderInfo, 135
  - createMortagingButton, 135
  - createPlayers, 136
  - createResultScreenStuff, 137
  - createTextBiddedProperty, 138
  - createTextBidderInfo, 138
  - createTextHighestBidInfo, 139
  - createTextLeadingBidderInfo, 140
  - createTextPlayersInfo, 141
  - createTextrolled\_value, 142
  - createTextTurnInfo, 142
  - getAllPropertyDataSprite, 143
  - getAllPropertyDataTexts, 143
  - getAuctionButtons, 144
  - getAuctionState, 144
  - getAuctionTexts, 145
  - getBoard, 145
  - getButtons, 146
  - getFont, 146
  - getFontSize, 148
  - getHotelCount, 149
  - getHotelSprite, 150
  - getHotelTexture, 151
  - getHouseCount, 151
  - getHouseSize, 152
  - getHouseSprite, 152
  - getHouseTexture, 153
  - getNotificationsWall, 154
  - getPlayers, 154
  - getPlayersResult, 155
  - getPropertyDataSprite, 155
  - getPropertyDataTexts, 156
  - getPropertyNameToDraw, 156
  - getResultTexts, 157
  - getScreenType, 158
  - getTexts, 158
  - getTurnState, 159
  - getWithdraw, 159
  - groupCompleted, 160
  - isBuildingLegal, 161
  - isDestroyingLegal, 162
  - isHotelBuildingLegal, 163
  - isHotelDestroyingLegal, 164
  - MonopolyGameEngine, 112
  - monopolyGameWorker, 165
  - performAuction, 167
  - setAuctionState, 167
  - setFont, 168
  - setHotelCount, 168
  - setHouseCount, 169
  - setplayer\_index\_turn, 169
  - setScreenType, 170
  - showPropertyData, 170
  - shuffleChanceCards, 172
  - subtractHotels, 172
  - subtractHouses, 173
  - turnInfoTextShow, 174
  - updateAvailableHousesHotelText, 174
  - updateResultScreenStuff, 175
  - updateTextPlayersInfo, 175
- MonopolyGameEngine.h
  - AuctionState, 304
  - BIDDING, 304
  - ENDING, 304
  - INITIALIZATION, 304
  - NO\_AUCTION, 304
  - PASS\_BIDDING\_TURN, 304
- monopolyGameWorker
  - MonopolyGameEngine, 165
- mouelsNotOver
  - Button, 49
- mouelsOver
  - Button, 50
- MOVEMENT\_SPACES
  - Chance.h, 285
- MOVEMENT\_TO\_PROPERTY
  - Chance.h, 285
- MOVEMENT\_WITH\_BUY\_OR\_PAY



- Chance.h, 285
- mutation\_rates
  - neat::genome, 101
  - neat::pool, 209
- neat, 9
- neat::gene, 98
  - enabled, 99
  - from\_node, 99
  - innovation\_num, 99
  - to\_node, 99
  - weight, 99
- neat::genome, 100
  - adjusted\_fitness, 101
  - can\_be\_recurrent, 101
  - fitness, 101
  - genes, 101
  - genome, 100
  - global\_rank, 101
  - max\_neuron, 101
  - mutation\_rates, 101
  - network\_info, 102
- neat::innovation\_container, 103
  - add\_gene, 104
  - innovation\_container, 104
  - number, 104
  - pool, 104
  - reset, 104
- neat::mutation\_rate\_container, 176
  - bias\_mutation\_chance, 177
  - connection\_mutate\_chance, 178
  - crossover\_chance, 178
  - disable\_mutation\_chance, 178
  - enable\_mutation\_chance, 178
  - link\_mutation\_chance, 178
  - node\_mutation\_chance, 178
  - perturb\_chance, 178
  - read, 177
  - step\_size, 178
  - write, 177
- neat::network\_info\_container, 179
  - bias\_size, 179
  - functional\_nodes, 179
  - input\_size, 179
  - output\_size, 179
  - recurrent, 180
- neat::pool, 207
  - export\_to\_file, 208
  - generation, 208
  - generator, 209
  - get\_genomes, 208
  - import\_from\_file, 208
  - max\_fitness, 209
  - mutation\_rates, 209
  - network\_info, 209
  - new\_generation, 208
  - pool, 207
  - rd, 209
  - speciating\_parameters, 209
  - species, 209
- neat::speciating\_parameter\_container, 220
  - delta\_disjoint, 221
  - delta\_threshold, 221
  - delta\_weights, 221
  - population, 221
  - read, 220
  - stale\_species, 221
  - write, 220
- neat::specie, 222
  - average\_fitness, 222
  - genomes, 222
  - staleness, 222
  - top\_fitness, 222
- network\_info
  - neat::genome, 102
  - neat::pool, 209
- neuralnet
  - ann::neuralnet, 180
- neuron
  - ann::neuron, 182
- new\_generation
  - neat::pool, 208
- NO
  - main.h, 297
- NO\_AUCTION
  - MonopolyGameEngine.h, 304
- NO\_HOUSES
  - main.h, 301
- NO\_TURN
  - main.h, 301
- node\_mutation\_chance
  - neat::mutation\_rate\_container, 178
- NON\_RECURRENT
  - ann, 9
- NONE
  - main.h, 297
- NotificationWall, 183
  - addToWall, 184
  - clearWall, 185
  - getFont, 185
  - getFontSize, 186
  - getWall, 186
  - NotificationWall, 184
  - setFont, 187
- number
  - neat::innovation\_container, 104
- ONE\_HOUSE
  - main.h, 301
- ONE\_STATION
  - main.h, 301
- ONE\_UTILITY
  - main.h, 302
- operator=
  - ContextWindow, 66
- output\_size
  - neat::network\_info\_container, 179

PASS\_BIDDING\_TURN  
     MonopolyGameEngine.h, 304  
 PAY  
     main.h, 299  
 PAY\_FOR\_HOUSE\_HOTEL  
     Chance.h, 285  
 PAY\_PLAYERS  
     Chance.h, 285  
 PAY\_RENT  
     main.h, 301  
 performAuction  
     MonopolyGameEngine, 167  
 perturb\_chance  
     neat::mutation\_rate\_container, 178  
 PLAY  
     main.h, 300  
 Player, 187  
     addFieldOwnedId, 190  
     addMoney, 191  
     clearFieldOwnedId, 191  
     createSprite, 191  
     decideAcceptTrade, 192  
     decideAuctionBid, 192  
     decideBuildHouse, 192  
     decideBuy, 192  
     decideJail, 193  
     decideMortgage, 193  
     decideOfferTrade, 194  
     decideSellHouse, 194  
     decideUnmortgage, 194  
     getAdapter, 195  
     getAiLevel, 195  
     getColor, 195  
     getFieldOwnedId, 196  
     getId, 196  
     getIsAi, 196  
     getJailCards, 196  
     getJailStatus, 196  
     getMoney, 197  
     getNeuralNetwork, 197  
     getPosition, 197  
     getResultPlace, 198  
     getSprite, 198  
     getSpriteOffsetX, 198  
     getSpriteOffsetY, 198  
     getTexture, 198  
     hasFieldOwnedId, 198  
     Player, 189  
     reduceJailStatus, 199  
     removeFieldOwnedId, 199  
     setAiLevel, 199  
     setColor, 200  
     setId, 200  
     setIsAi, 201  
     setJailCards, 202  
     setJailStatus, 202  
     setMoney, 203  
     setPosition, 203  
     setResultPlace, 204  
     setSpriteOffsetX, 204  
     setSpriteOffsetY, 205  
     setSpritePosition, 205  
     subtractMoney, 205  
 PLAYER\_1\_SET\_AI  
     main.h, 300  
 PLAYER\_1\_SET\_AI\_LEVEL\_1  
     main.h, 300  
 PLAYER\_1\_SET\_AI\_LEVEL\_2  
     main.h, 300  
 PLAYER\_1\_SET\_AI\_LEVEL\_3  
     main.h, 300  
 PLAYER\_1\_SET\_HUMAN  
     main.h, 300  
 PLAYER\_1\_SET\_NONE  
     main.h, 300  
 PLAYER\_2\_SET\_AI  
     main.h, 300  
 PLAYER\_2\_SET\_AI\_LEVEL\_1  
     main.h, 300  
 PLAYER\_2\_SET\_AI\_LEVEL\_2  
     main.h, 300  
 PLAYER\_2\_SET\_AI\_LEVEL\_3  
     main.h, 300  
 PLAYER\_2\_SET\_HUMAN  
     main.h, 300  
 PLAYER\_2\_SET\_NONE  
     main.h, 300  
 PLAYER\_3\_SET\_AI  
     main.h, 300  
 PLAYER\_3\_SET\_AI\_LEVEL\_1  
     main.h, 300  
 PLAYER\_3\_SET\_AI\_LEVEL\_2  
     main.h, 300  
 PLAYER\_3\_SET\_AI\_LEVEL\_3  
     main.h, 300  
 PLAYER\_3\_SET\_HUMAN  
     main.h, 300  
 PLAYER\_3\_SET\_NONE  
     main.h, 300  
 PLAYER\_4\_SET\_AI  
     main.h, 300  
 PLAYER\_4\_SET\_AI\_LEVEL\_1  
     main.h, 300  
 PLAYER\_4\_SET\_AI\_LEVEL\_2  
     main.h, 300  
 PLAYER\_4\_SET\_AI\_LEVEL\_3  
     main.h, 300  
 PLAYER\_4\_SET\_HUMAN  
     main.h, 300  
 PLAYER\_4\_SET\_NONE  
     main.h, 300  
 playerSettings, 206  
     isHuman, 206  
     isNone, 206  
     level, 206  
 pollForEvents

- GameEngine, 82
- pool
  - neat::innovation\_container, 104
  - neat::pool, 207
- population
  - neat::speciating\_parameter\_container, 221
- PossibleFields
  - Board.h, 279
- PropertyField, 210
  - createFlagSprite, 212
  - getGroupMembers, 213
  - getIsMortgaged, 213
  - getMortgage, 214
  - getOwner, 214
  - getOwnerFlag, 215
  - getPrice, 215
  - getUnmortgageValue, 216
  - PropertyField, 211
  - resetDefault, 216
  - resetOwner, 217
  - setIsMortgaged, 217
  - setOwner, 218
- propertyPlayerMoveIndex
  - Withdraw, 263
- rd
  - neat::pool, 209
- read
  - neat::mutation\_rate\_container, 177
  - neat::speciating\_parameter\_container, 220
- RECURRENT
  - ann, 9
- recurrent
  - neat::network\_info\_container, 180
- reduceJailStatus
  - Player, 199
- removeFieldOwnedId
  - Player, 199
- reset
  - neat::innovation\_container, 104
- resetDefault
  - PropertyField, 216
  - StreetField, 232
- resetOwner
  - PropertyField, 217
- RESIGN
  - main.h, 297
- RESULT
  - main.h, 299
- RESULTS
  - main.h, 301
- RETURN\_TO\_MAIN\_MENU
  - main.h, 300
- ROLL
  - main.h, 299
- ROLL\_DICE
  - main.h, 301
- RotationException, 218
  - getBadRotation, 220
- RotationException, 219
- ScreenEventType
  - main.h, 299
- setActiveBackColor
  - Button, 50
- setActiveTextColor
  - Button, 50
- setAiLevel
  - Player, 199
- setAllLevelColumnVisibility
  - GameMenuScreen, 89
- setAuctionState
  - MonopolyGameEngine, 167
- setBoard
  - Withdraw, 263
- setButtonFocus
  - Button, 51
- setButtonUnfocus
  - Button, 51
- setCard
  - AiAdapter, 25
- setChooseScreenVisible
  - Withdraw, 264
- setColor
  - Player, 200
- setContextWindow
  - ActiveScreen, 17
- setDecisionScreenVisible
  - Withdraw, 264
- setDefaultAllLevelButtonsFocus
  - GameMenuScreen, 90
- setEventType
  - Button, 52
- setFocusBackColor
  - Button, 52
- setFocusTextColor
  - Button, 52
- setFont
  - ActiveScreen, 17
  - Button, 53
  - MonopolyGameEngine, 168
  - NotificationWall, 187
  - Withdraw, 265
- setHeight
  - Field, 76
- setHotelCount
  - MonopolyGameEngine, 168
- setHouse
  - AiAdapter, 26
- setHouseCount
  - MonopolyGameEngine, 169
- setHouseNumber
  - StreetField, 232
- setId
  - Player, 200
- setInactiveBackColor
  - Button, 53
- setInactiveTextColor

- Button, [53](#)
- setIsActive
  - Button, [53](#)
- setIsAi
  - Player, [201](#)
- setIsClicked
  - Button, [54](#)
- setIsFocus
  - Button, [54](#)
- setIsHotel
  - StreetField, [233](#)
- setIsMortgaged
  - PropertyField, [217](#)
- setIsVisible
  - Button, [54](#)
- setJail
  - AiAdapter, [26](#)
- setJailCards
  - Player, [202](#)
- setJailStatus
  - Player, [202](#)
- setMoney
  - AiAdapter, [27](#)
  - Player, [203](#)
- setMoneyContext
  - AiAdapter, [27](#)
- setMortgage
  - AiAdapter, [28](#)
- setNeuralNetwork
  - AiPlayer, [39](#)
- setOtherButtonsInactive
  - GameMenuScreen, [91](#)
- setOwner
  - AiAdapter, [28](#)
  - PropertyField, [218](#)
- setPlayer1IndexProperties
  - Withdraw, [266](#)
- setPlayer1ToWithdraw
  - Withdraw, [266](#)
- setPlayer2IndexProperties
  - Withdraw, [266](#)
- setPlayer2ToWithdraw
  - Withdraw, [266](#)
- setplayer\_index\_turn
  - MonopolyGameEngine, [169](#)
- setPlayerSettings
  - GameMenuScreen, [92](#)
- setPosition
  - AiAdapter, [28](#)
  - Button, [55](#)
  - ChanceCard, [61](#)
  - Field, [76](#)
  - Player, [203](#)
- setResultPlace
  - Player, [204](#)
- setRotation
  - ChanceCard, [61](#)
  - Field, [77](#)
- setScreenType
  - ActiveScreen, [18](#)
  - MonopolyGameEngine, [170](#)
- setSelection
  - AiAdapter, [29](#)
- setSelectionState
  - AiAdapter, [29](#)
- setSpriteOffsetX
  - Player, [204](#)
- setSpriteOffsetY
  - Player, [205](#)
- setSpritePosition
  - Player, [205](#)
- setTurn
  - AiAdapter, [29](#)
- setTurnState
  - Withdraw, [267](#)
- setValueScreenVisible
  - Withdraw, [267](#)
- setWasReleased
  - Button, [55](#)
- setWidth
  - Field, [77](#)
- showProperty
  - Withdraw, [268](#)
- showPropertyData
  - MonopolyGameEngine, [170](#)
- shuffleChanceCards
  - MonopolyGameEngine, [172](#)
- speciating\_parameters
  - neat::pool, [209](#)
- species
  - neat::pool, [209](#)
- SpriteOffsetException, [223](#)
  - getBadOffset, [224](#)
  - SpriteOffsetException, [223](#), [224](#)
- stale\_species
  - neat::speciating\_parameter\_container, [221](#)
- staleness
  - neat::specie, [222](#)
- START\_GAME
  - main.h, [300](#)
- STATION
  - main.h, [299](#)
- StationField, [224](#)
  - calculateRent, [226](#)
  - getRentValues, [226](#)
  - StationField, [225](#)
- StationTiers
  - main.h, [300](#)
- step\_size
  - neat::mutation\_rate\_container, [178](#)
- STREET
  - main.h, [299](#)
- StreetField, [227](#)
  - calculateRent, [229](#)
  - getHotelPrice, [230](#)
  - getHouseNumber, [230](#)

- getHousePrice, [230](#)
- getIsHotel, [231](#)
- getRentValues, [231](#)
- resetDefault, [232](#)
- setHouseNumber, [232](#)
- setIsHotel, [233](#)
- StreetField, [229](#)
- StreetTiers
  - main.h, [301](#)
- subtractHotels
  - MonopolyGameEngine, [172](#)
- subtractHouses
  - MonopolyGameEngine, [173](#)
- subtractMoney
  - Player, [205](#)
- TAX
  - main.h, [299](#)
- TAX\_CARD
  - Chance.h, [285](#)
- TaxField, [233](#)
  - getTaxValue, [235](#)
  - TaxField, [234](#)
- THREE\_HOUSES
  - main.h, [301](#)
- THREE\_STATIONS
  - main.h, [301](#)
- to\_node
  - neat::gene, [99](#)
- top\_fitness
  - neat::specie, [222](#)
- TRAIN
  - main.cc, [294](#)
- TURN\_END
  - main.h, [301](#)
- turnInfoTextShow
  - MonopolyGameEngine, [174](#)
- TurnState
  - main.h, [301](#)
- TWO\_HOUSES
  - main.h, [301](#)
- TWO\_STATIONS
  - main.h, [301](#)
- TWO\_UTILITIES
  - main.h, [302](#)
- type
  - ann, [9](#)
  - ann::neuron, [183](#)
- updateAvailableHousesHotelText
  - MonopolyGameEngine, [174](#)
- updateColors
  - Button, [55](#)
- updateResultScreenStuff
  - MonopolyGameEngine, [175](#)
- updateTextPlayersInfo
  - MonopolyGameEngine, [175](#)
- UTILITY
  - main.h, [299](#)
- UtilityField, [236](#)
  - calculateRent, [237](#)
  - getRentMultipliers, [238](#)
  - UtilityField, [237](#)
- UtilityTiers
  - main.h, [301](#)
- value
  - ann::neuron, [183](#)
- view\_
  - ContextWindow, [66](#)
- visited
  - ann::neuron, [183](#)
- weight
  - neat::gene, [99](#)
- width
  - main.cc, [294](#)
- WIDTH\_MAX
  - main.cc, [295](#)
- window\_
  - ContextWindow, [66](#)
- Withdraw, [238](#)
  - addPropertyPlayerShowed, [242](#)
  - createChoosePlayerScreen, [242](#)
  - createDecisionPlayerScreen, [243](#)
  - createValuePlayerScreen, [244](#)
  - getAcceptDecisionButton, [244](#)
  - getButtons, [244](#)
  - getFont, [245](#)
  - getFontSize, [245](#)
  - getPlayer1AddButton, [246](#)
  - getPlayer1Button, [246](#)
  - getPlayer1IndexNextButton, [246](#)
  - getPlayer1IndexPreviousButton, [247](#)
  - getPlayer1IndexProperties, [247](#)
  - getPlayer1minus1, [247](#)
  - getPlayer1minus10, [247](#)
  - getPlayer1minus100, [248](#)
  - getPlayer1NextButton, [248](#)
  - getPlayer1plus1, [248](#)
  - getPlayer1plus10, [248](#)
  - getPlayer1plus100, [249](#)
  - getPlayer1PreviousButton, [249](#)
  - getPlayer1RemoveButton, [249](#)
  - getPlayer1ToWithdraw, [249](#)
  - getPlayer2AddButton, [250](#)
  - getPlayer2Button, [250](#)
  - getPlayer2IndexNextButton, [250](#)
  - getPlayer2IndexPreviousButton, [250](#)
  - getPlayer2IndexProperties, [251](#)
  - getPlayer2minus1, [251](#)
  - getPlayer2minus10, [251](#)
  - getPlayer2minus100, [251](#)
  - getPlayer2NextButton, [252](#)
  - getPlayer2plus1, [252](#)
  - getPlayer2plus10, [252](#)
  - getPlayer2plus100, [252](#)
  - getPlayer2PreviousButton, [253](#)

- getPlayer2RemoveButton, [253](#)
- getPlayer2ToWithdraw, [253](#)
- getPlayer3Button, [253](#)
- getPlayer4Button, [254](#)
- getResignButton, [254](#)
- getResignDecisionButton, [254](#)
- getResignValueButton, [254](#)
- getSpritePropertyPlayer1, [255](#)
- getSpritePropertyPlayer1Index, [255](#)
- getSpritePropertyPlayer2, [255](#)
- getSpritePropertyPlayer2Index, [256](#)
- getSubmitValueButton, [256](#)
- getTexts, [257](#)
- getTextsPropertyPlayer1, [257](#)
- getTextsPropertyPlayer1Index, [257](#)
- getTextsPropertyPlayer2, [258](#)
- getTextsPropertyPlayer2Index, [258](#)
- getTexturePropertyPlayer1, [259](#)
- getTexturePropertyPlayer1Index, [259](#)
- getTexturePropertyPlayer2, [260](#)
- getTexturePropertyPlayer2Index, [260](#)
- getTurnState, [261](#)
- isNonZeroValue, [261](#)
- makeWithdraw, [261](#)
- moneyTextUpdate, [262](#)
- moneyTransferIndex, [262](#)
- propertyPlayerMoveIndex, [263](#)
- setBoard, [263](#)
- setChooseScreenVisible, [264](#)
- setDecisionScreenVisible, [264](#)
- setFont, [265](#)
- setPlayer1IndexProperties, [266](#)
- setPlayer1ToWithdraw, [266](#)
- setPlayer2IndexProperties, [266](#)
- setPlayer2ToWithdraw, [266](#)
- setTurnState, [267](#)
- setValueScreenVisible, [267](#)
- showProperty, [268](#)
- Withdraw, [242](#)
- WITHDRAW\_ADD\_VALUE
  - main.h, [299](#)
- WITHDRAW\_CHOOSE\_PLAYER
  - main.h, [299](#)
- WITHDRAW\_DECISION
  - main.h, [299](#)
- WITHDRAW\_ONGOING
  - main.h, [301](#)
- worker
  - ActiveScreen, [19](#)
  - GameEngine, [82](#)
  - GameMenuScreen, [92](#)
  - GameScreen, [97](#)
  - MainMenuScreen, [108](#)
- write
  - neat::mutation\_rate\_container, [177](#)
  - neat::speciating\_parameter\_container, [220](#)
- YES
  - main.h, [297](#)