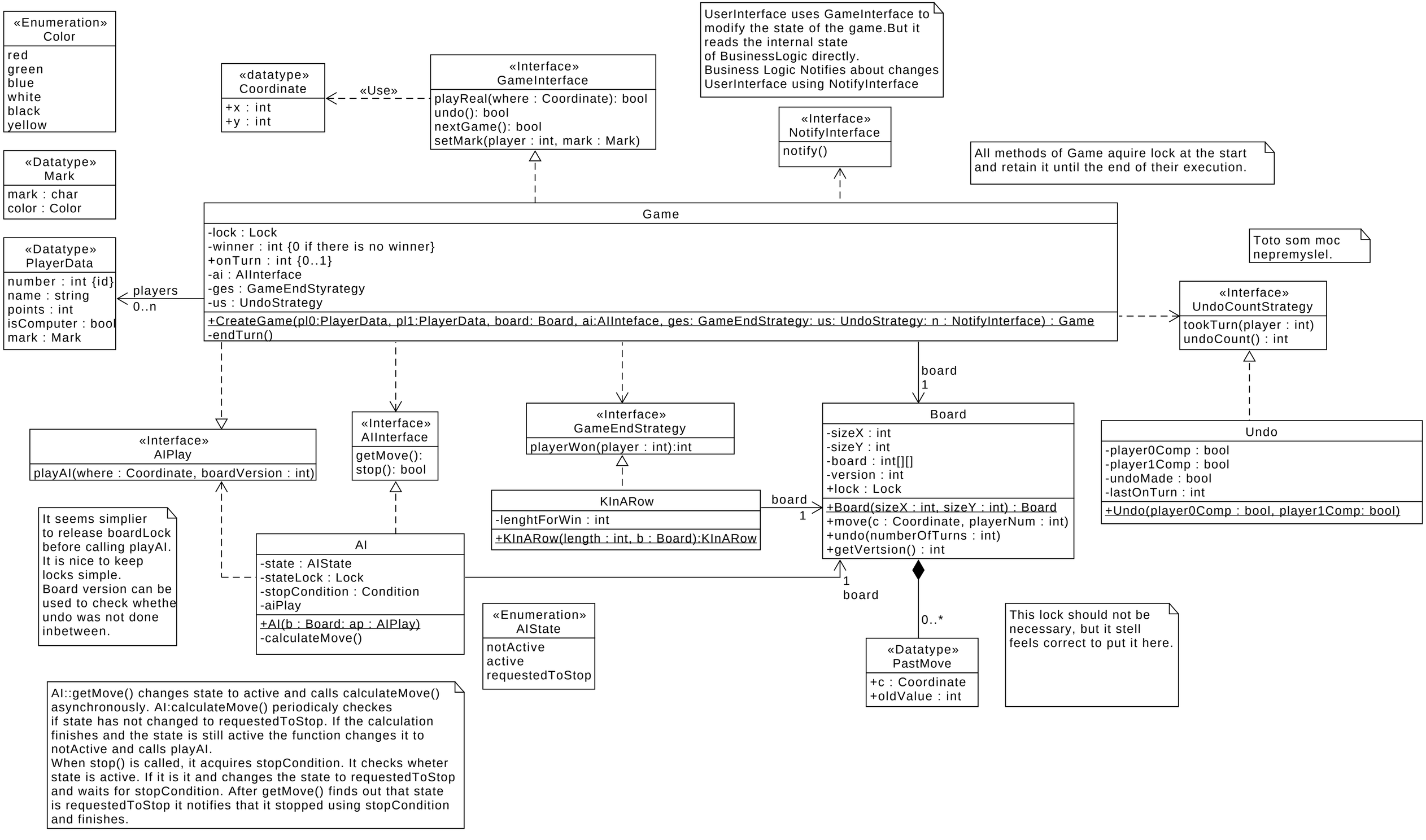
«Enumeration»
Color
red
green
blue
white
black
yellow
«Datatype»
Mark
mark : char
color : Color
«Datatype»
PlayerData
number : int {id}
name : string
points : int
isComputer : bool
mark : Mark
«datatype»
Coordinate
+x : int
+y : int
«Use»
«Interface»
GameInterface
playReal(where : Coordinate): bool
undo(): bool
nextGame(): bool
setMark(player : int, mark : Mark)
UserInterface uses GameInterface to modify the state of the game.But it reads the internal state of BusinessLogic directly.
Business Logic Notifies about changes UserInterface using NotifyInterface
«Interface»
NotifyInterface
notify()
All methods of Game aquire lock at the start and retain it until the end of their execution.
Game
-lock : Lock
-winner : int {0 if there is no winner}
+onTurn : int {0..1}
-ai : AIInterface
-ges : GameEndStyrategy
-us : UndoStrategy
+CreateGame(pl0:PlayerData, pl1:PlayerData, board: Board, ai:AIInteface, ges: GameEndStrategy: us: UndoStrategy: n : NotifyInterface) : Game
-endTurn()
players
0..n
Toto som moc nepremyslel.
«Interface»
UndoCountStrategy
tookTurn(player : int)
undoCount() : int
board
1
«Interface»
AIPlay
playAI(where : Coordinate, boardVersion : int)
«Interface»
AIInterface
getMove():
stop(): bool
«Interface»
GameEndStrategy
playerWon(player : int):int
Board
-sizeX : int
-sizeY : int
-board : int[][]
-version : int
+lock : Lock
+Board(sizeX : int, sizeY : int) : Board
+move(c : Coordinate, playerNum : int)
+undo(numberOfTurns : int)
+getVertsion() : int
Undo
-player0Comp : bool
-player1Comp : bool
-undoMade : bool
-lastOnTurn : int
+Undo(player0Comp : bool, player1Comp: bool)
KInARow
-lenghtForWin : int
+KInARow(length : int, b : Board):KInARow
board
1
It seems simplier to release boardLock before calling playAI. It is nice to keep locks simple.
Board version can be used to check whethe undo was not done inbetween.
AI
-state : AIState
-stateLock : Lock
-stopCondition : Condition
-aiPlay
+AI(b : Board: ap : AIPlay)
-calculateMove()
1
board
«Enumeration»
AIState
notActive
active
requestedToStop
0..*
«Datatype»
PastMove
+c : Coordinate
+oldValue : int
This lock should not be necessary, but it stell feels correct to put it here.
AI::getMove() changes state to active and calls calculateMove() asynchronously. AI:calculateMove() periodicaly checkes if state has not changed to requestedToStop. If the calculation finishes and the state is still active the function changes it to notActive and calls playAI.
When stop() is called, it acquires stopCondition. It checks wheter state is active. If it is it and changes the state to requestedToStop and waits for stopCondition. After getMove() finds out that state is requestedToStop it notifies that it stopped using stopCondition and finishes.