Team Panda - Tests Description

Aseel Butto	318395175
Maryam Wahbi	207186016
Saleh Ghraiyib	315259192
Firas Dakwar	207416850

Test ID	Description	Expected Results	Actual Results
JUNIT TESTS			
1	The test checks if the movement of the soldier is Legal or not. There are 3 soldiers on the board in locations:(6,'F'),(4,'H'),(7,'G') and the soldier wants to move to the location(8,'H').	return true,the soldier can move to the location(8,'H').	return true (the soldier can move to the location(8,'H')) .
2	the test checks in case the current playing player is stuck,is game finished	true	true
3	the test checks in case the opponent playing player is stuck,is game finished	false	false
4	the test checks if playing player is out of pieces, does the game finish	true	true
5	the test checks if opponent player is out of pieces, does the game finish	true	true
6	Test checks score after answering an **Intermediate** question both correctly and incorrectly	Correct : Score + 200 Incorrect : Score - 100	Score + 200 Score - 100
7	Test checks score after answering an **Easy** question both correctly and incorrectly	Correct : Score + 100 Incorrect : Score - 250	Score + 100 Score - 250
8	Test checks score after answering an **Hard** question both correctly and incorrectly	Correct : Score + 500 Incorrect : Score - 50	Score + 500 Score - 50
9	Test checks yellow tile	Incorrect : false	Incorrect : false

	functionality from drawing random question until player answering both correct and incorrect answers and checking the yellow tile methods	Correct : True	Correct : True
10	Checks if the queen is blocked according to the given board. Queen move: from : A1 To: C7 Direction: UP Left	false	false
11	Returns a edible piece location	E5	E5
12	Checks whether the queen can move to specific location Queen move: from: A1 To: C7 Direction UP Left	true	true
White box tests			
13	**Whitebox** Checks if the piece can move, using CanPieceMove method Expected result will be given according to soldier that tried to move forward by 1 tile, the destination is empty and available	true	
14	**Whitebox**: checks if the movement of the soldier is Legal or not.	true	
15	**Whitebox**: **FinishGame** **Go Down For Description**	(Not Running Game With Player 1 having more score than player 2)	
16	**WhitBox**-isGame finished: checks if game is finished depending on if current player is stuck (no more possible moves for any of his pieces) or one of the players ran out of pieces without. input takes current and opponent colors and current and opponent pieces on board	true if current player is stuck or current Player is out of pieces or opponent is out of pieces on board,otherwise false	
Black Box Tests			
17	**BlackBox**: Checks if the soldier can eat according to the given scenarios in requirement document	true	

	Expected result will be given according to the soldier trying to eat a piece that is located in close tile to him.		
18	**BlackBox**:calculate the number of new scores according to the time of the turn.	true	
19	**BlackBox**:checks if blue tile can be added to board	true if board contains exactly one queen and 2 soldiers,false otherwise	
20	**BlackBox**: **Yellow Tile** **Description Down Below**	(Hard Question, Chosen Correct Answer)	

WhiteBox:

id: 13

זהו בדיקת קופסא לבנה על בדיקת תנועה לכלי משחק כלשהו (מלכה / חייל) כלל התנאים נבדקים וניתן חריגות בהתאם.

קלט:

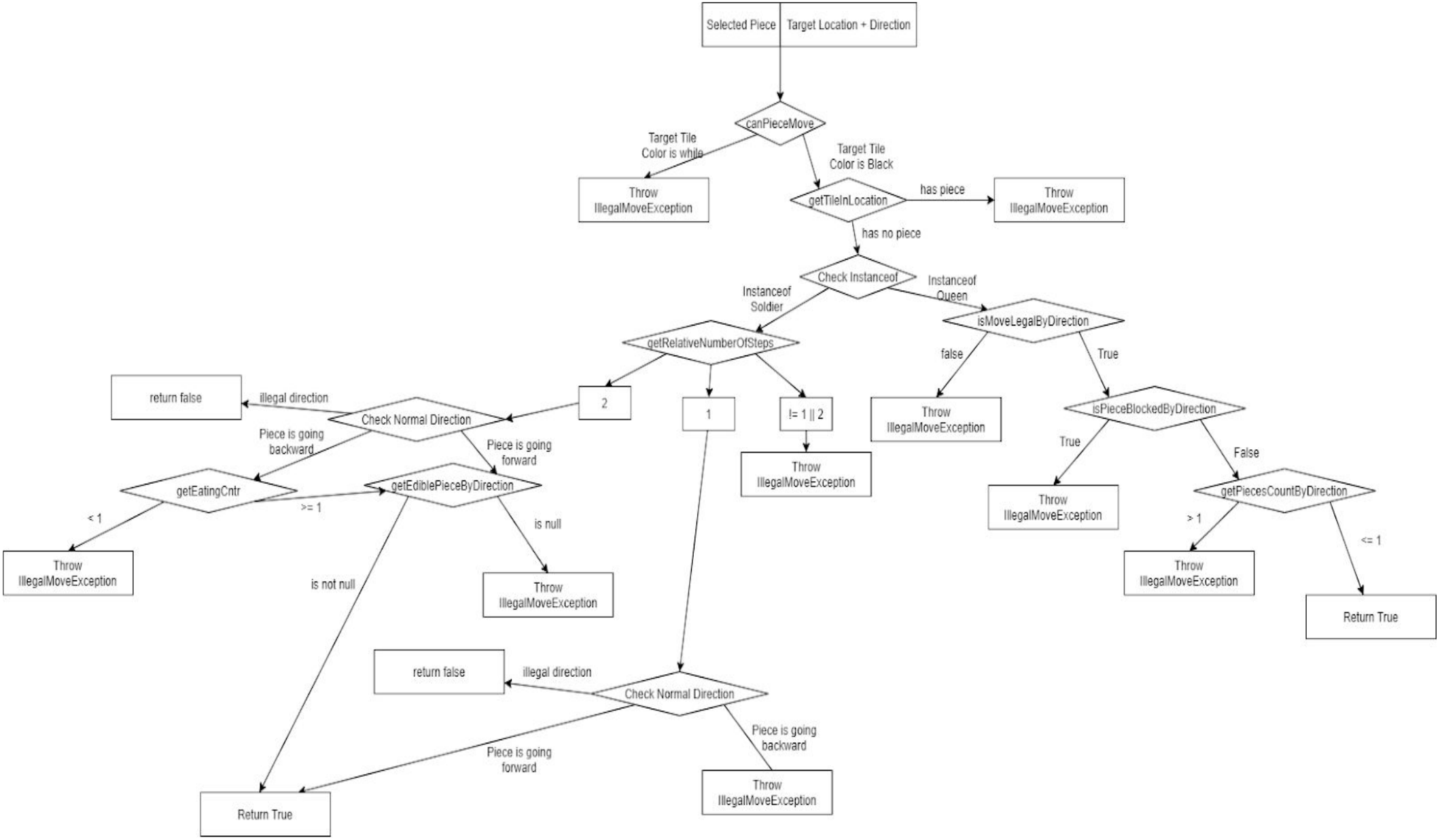
- **Piece – כלי משחק שרוצים לנוע**
- **Target Location – היעד שנבחר מתוך לוח המשחק**
- **Direction – את הכיוון בה השחקן בחר לנוע את הכלי**

פלט:

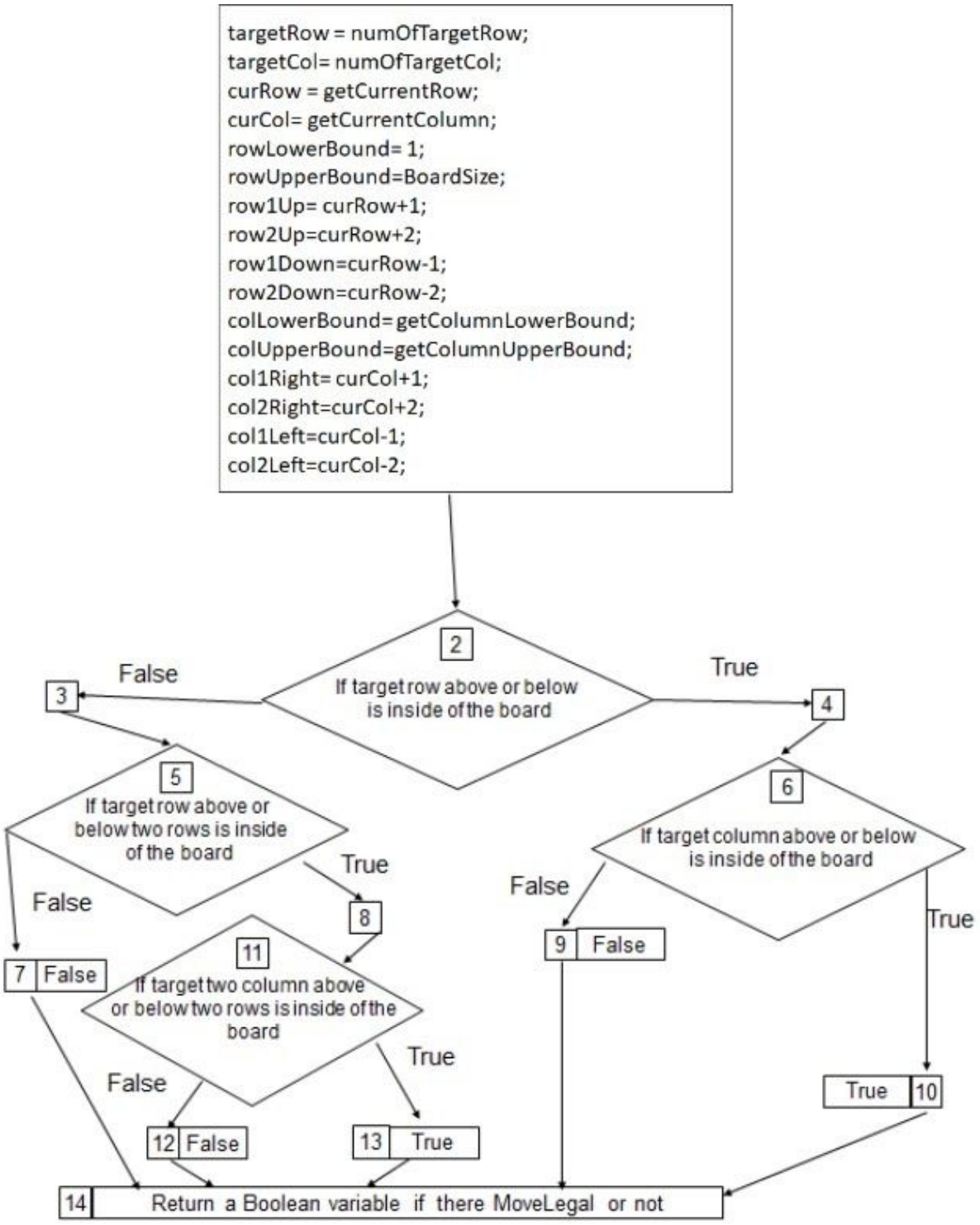
- **False – תנועה אינה אפשרית**
  - **יכול להיות סיום הבדיקה תהיה ע"י זריקת חריגה כך זאת מתארת הפלט זה.**
- **True – תנועה אפשרית**

גרף:

מצורף קובץ תמונה המכיל את הגרף במקרה ואיננו ברור


Selected Piece
Target Location + Direction
canPieceMove
Target Tile Color is while
Throw IllegalMoveException
Target Tile Color is Black
getTileInLocation
has piece
Throw IllegalMoveException
has no piece
Check Instanceof
Instanceof Soldier
Instanceof Queen
isMoveLegalByDirection
false
Throw IllegalMoveException
True
isPieceBlockedByDirection
True
Throw IllegalMoveException
False
getPiecesCountByDirection
> 1
Throw IllegalMoveException
<= 1
Return True
getRelativeNumberOfSteps
2
return false
illegal direction
Check Normal Direction
Piece is going backward
getEatingCntr
< 1
Throw IllegalMoveException
>= 1
getEdiblePieceByDirection
is null
Throw IllegalMoveException
Piece is going forward
is not null
Return True
1
return false
illegal direction
Check Normal Direction
Piece is going backward
Throw IllegalMoveException
Piece is going forward
Return True
!= 1 || 2
Throw IllegalMoveException

id: 14
description:checks if the movement of the soldier is Legal or not.
input: target location is (8,'H').
output:true.

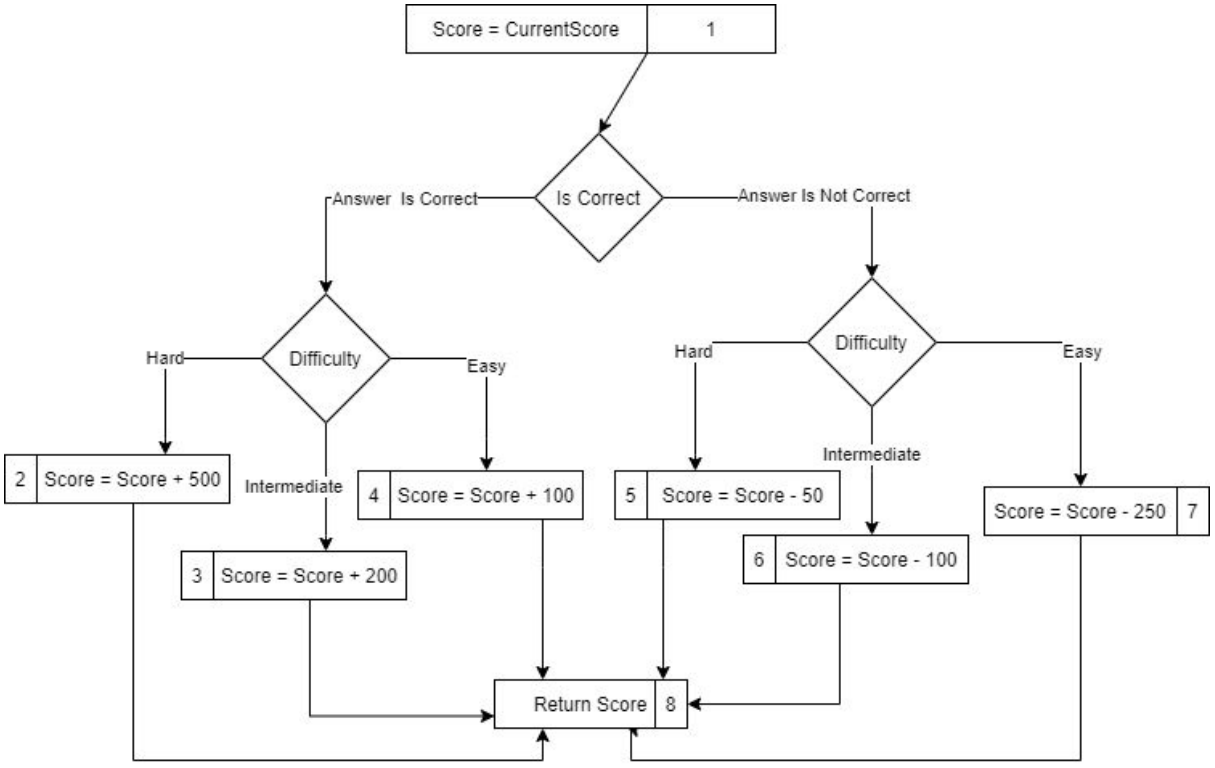


id: 15

Description: Yellow Tile Question

- Draw Random Question
- Player Answers
- If Question is Hard:
    - Correct answer +500
    - Incorrect Answer -50
- If Question is Intermediate
    - Correct Answer + 200
    - Incorrect Answer -100
- If Question is Easy
    - Correct Answer + 100
    - Incorrect Answer -250

Assume That Random Question is Provided to The Answer Process



Input: (Hard Question, Chosen Correct Answer)

Output: (1 -> 2 -> 8)

id 16:

Description: check if isGame finished

input:

currentPlaye=”WHITE”

opponentColor=”Black”

board=Board.getInstance()=>{

soldierLocations:

Soldier s2bb= new Soldier(1, PrimaryColor.BLACK, new Location(2, 'B'));

Soldier s3cb=new Soldier(2, PrimaryColor.BLACK, new Location(3, 'C'));

Soldier s1aw= new Soldier(3, PrimaryColor.WHITE, new Location(1, 'A'));

Soldier s1cw=new Soldier(4, PrimaryColor.WHITE, new Location(1, 'C'));

Soldier s2dw=new Soldier(5, PrimaryColor.WHITE, new Location(2, 'D'));

Soldier s1ew=new Soldier(6, PrimaryColor.WHITE, new Location(1, 'E'));

}

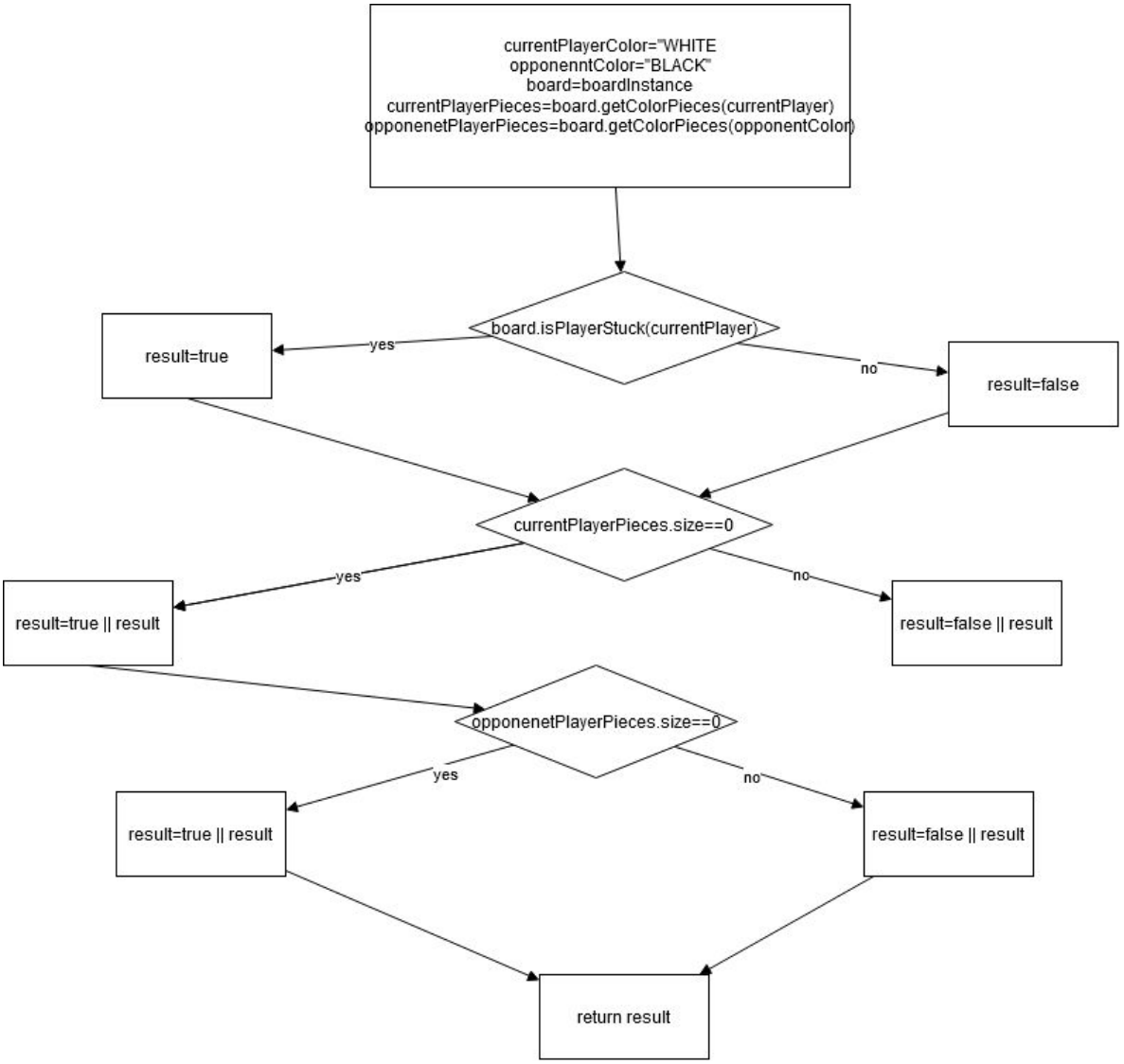
currentPlayerPieces=board.getColorPieces(currentPlayer)

opponenetPlayerPieces=board.getColorPieces(opponentColor)

output:

result= (true->false->false)=> true

Graph:


currentPlayerColor="WHITE
opponenntColor="BLACK"
board=boardInstance
currentPlayerPieces=board.getColorPieces(currentPlayer)
opponenetPlayerPieces=board.getColorPieces(opponentColor)
board.isPlayerStuck(currentPlayer)
yes
result=true
no
result=false
currentPlayerPieces.size==0
yes
result=true || result
no
result=false || result
opponenetPlayerPieces.size==0
yes
result=true || result
no
result=false || result
return result

BlackBox:

id: 17

זוהי בדיקת קופסה שחורה המהווה את האפשרות לאכילה ע"י חייל.

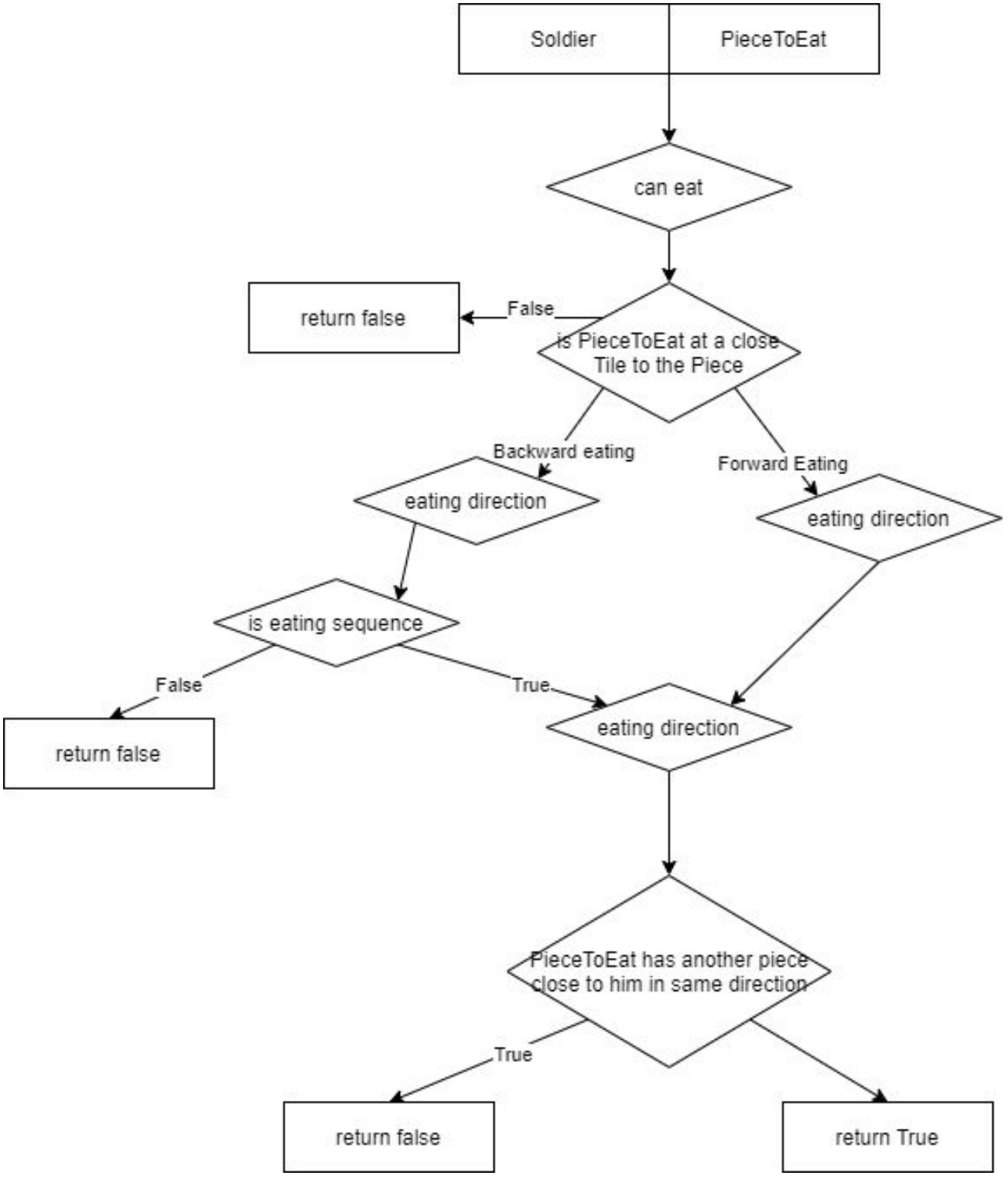
קלט:

- **Soldier – חייל כלשהו שרוצים לבדוק אם הוא יכול לאכול כלי משחק אחר כלשהו**
- **PieceToEat – כלי משחק כלשהו שהחייל רוצה לאכול**

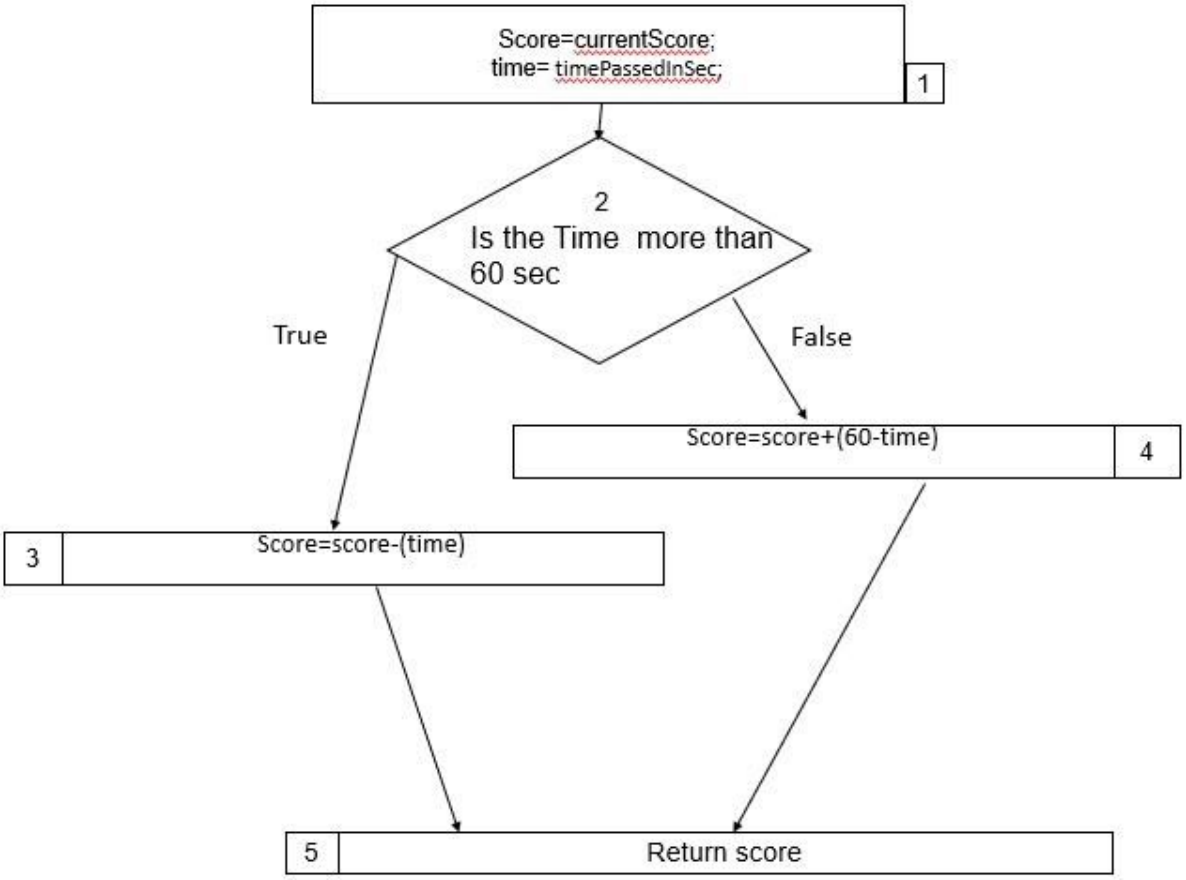
פלט:

- **false – במקרה ואי אפשר לאכול**
  - **למשל: אם הכלי שרוצה לאכול איננו נתמצא במשבצת צמודה לחייל**
- **true – אפשר לאכול**
  - **במקרה וכל התנאים מתקיימים.**

גרף:


Soldier
PieceToEat
can eat
return false
False
is PieceToEat at a close Tile to the Piece
Backward eating
Forward Eating
eating direction
eating direction
is eating sequence
False
True
return false
eating direction
PieceToEat has another piece close to him in same direction
True
return false
return True

id: 18
description::calculate the number of new scores according to the time of the turn.
example:
input :64 sec,100 current score.
output:current score-4=100-4=96



id: 19:

Description:checks if blue tile can be added to board

input:

currentPlayer color=black
currentPlayerPieces:{

Soldier s2db= new Soldier(1, PrimaryColor.BLACK, new Location(2, 'D'))

Soldier s2fb=new Soldier(2, PrimaryColor.BLACK, new Location(2, 'F'))

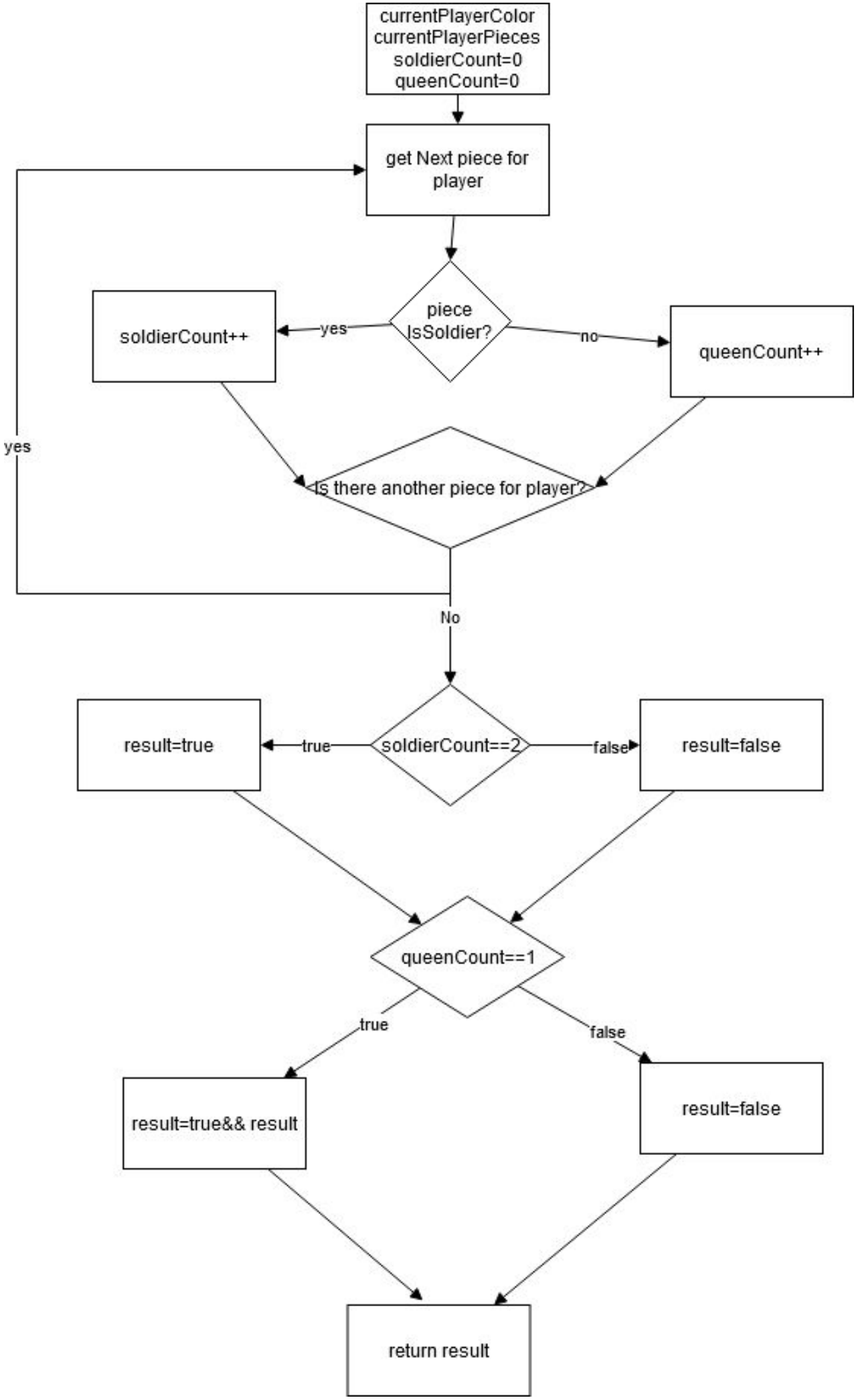
Queen q2hb=new Queen(7,PrimaryColor.BLACK, new Location(2, 'H'))

}

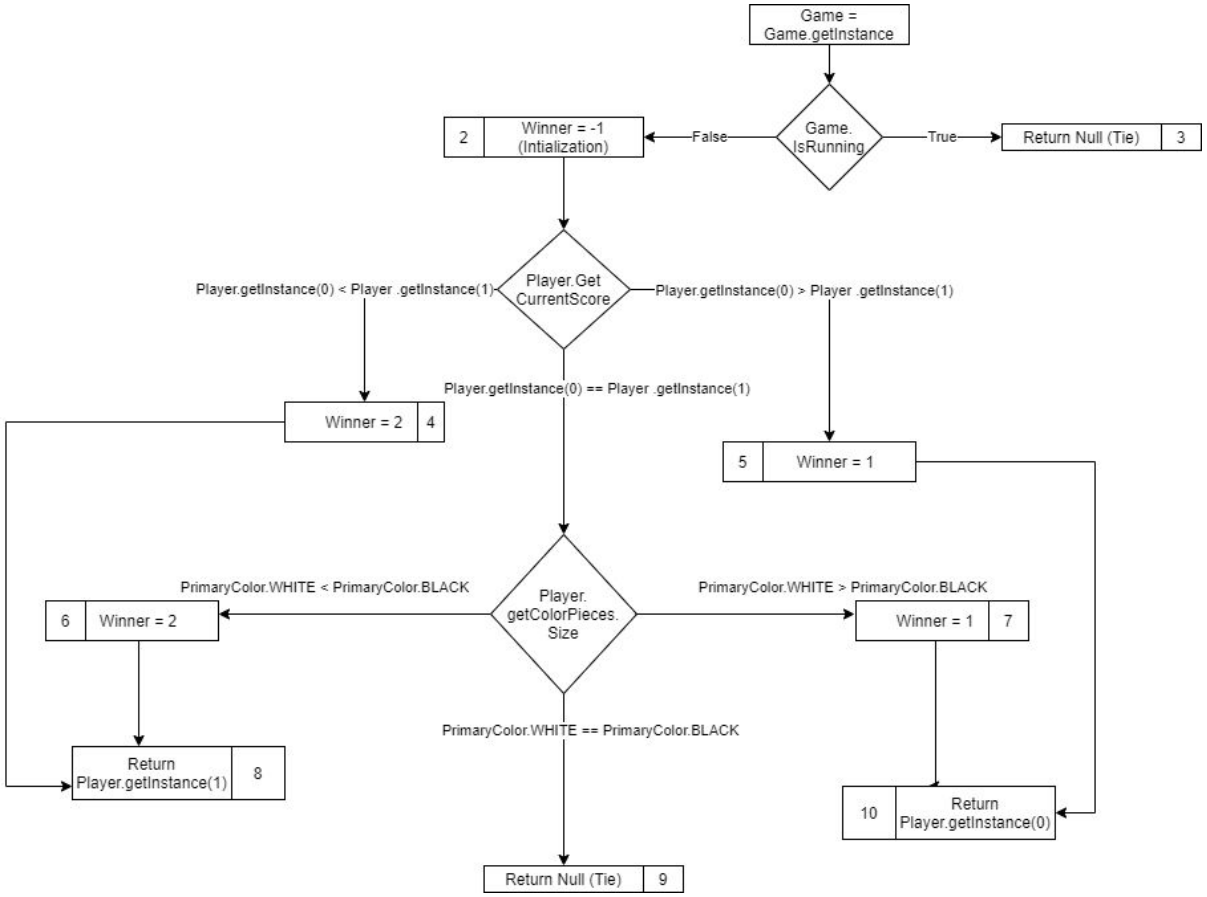
output:

(yes->yes->no->true->true)=>true

Graph:


currentPlayerColor
currentPlayerPieces
soldierCount=0
queenCount=0
get Next piece for player
piece IsSoldier?
yes
soldierCount++
no
queenCount++
yes
Is there another piece for player?
No
result=true
true
soldierCount==2
false
result=false
queenCount==1
true
false
result=true&& result
result=false
return result

id: 20



1. Get Winner Method : works when there are no more moves to do or when somebody exits
2. Input : Full Game Status (contains players,timer's ,scores ….)

    1) Exiting in mid of game gives tie when both players have same score

    2) If player 1 score bigger than player 2 score – player 1 wins

    3) If play 2 score bigger than player 1 score – player 2 wins

    4) If both have same score –

        a. if(player 1 's pieces are more than player 2's pieces) -> player 1 wins

        b. if(player 2 's pieces are more than player 1's pieces) - > player 2 wins

3. return winner if its not -1 else it's a tie

Possible Input : (Not Running Game With Player 1 Having More Score Than Player 2)

Expected Output : (1 -> 2 -> 5 -> 10)