**Solver source code:**

**from ten\_to\_zero\_game import ten\_to\_zero**

def Solve(game, position):

GenerateMoves = game.GenerateMoves

DoMove = game.DoMove

PrimitiveValue = game.PrimitiveValue

result = PrimitiveValue(position)

if result == "undecided":

has\_tie = False

for move in GenerateMoves(position):

after\_move\_result = Solve(game, DoMove(position, move))

if after\_move\_result == "lose":

return "win"

elif after\_move\_result == "tie":

has\_tie = True

if has\_tie:

return "tie"

else:

return "lose"

else:

return result

**Generic game source code:**

class Game:

primitive\_position = 0

possible\_moves = []

def DoMove(self, position, move):

if not move in self.possible\_moves or position - move < 0:

return "illegal move"

if position == self.primitive\_position:

return "lose"

return position - move

def GenerateMoves(self, position):

ret\_moves = []

for move in self.possible\_moves:

if self.DoMove(position, move) != "illegal move" and type(self.DoMove(position, move)) != "String":

ret\_moves.append(move)

return ret\_moves

def PrimitiveValue(self, position):

if position in self.primitive\_position:

return "lose"

else:

return "undecided"

**10-to-0-by-1-or-2 source code:**

from generic\_game import Game

class twentyfive\_to\_zero(Game):

primitive\_position = [0]

possible\_moves = [1,2]

**Tested output of 10-to-0-by-1-or-2:**

10: win

9: lose

8: win

7: win

6: lose

5: win

4: win

3: lose

2: win

1: win

0: lose

**Source-code for 25-to-0-by-1-3-4**

from generic\_game import Game

class twentyfive\_to\_zero(Game):

primitive\_position = [0]

possible\_moves = [1,3,4]

**Tested output of 25-to-0-by-1-3-4**

25: win

24: win

23: lose

22: win

21: lose

20: win

19: win

18: win

17: win

16: lose

15: win

14: lose

13: win

12: win

11: win

10: win

9: lose

8: win

7: lose

6: win

5: win

4: win

3: win

2: lose

1: win

0: lose