1.

class Game a

get\_available\_moves :: a -> [Move]

next\_state :: a -> Move -> a

is\_gameover :: a -> Bool

----------------------------------------------------------

2.

def minimax(game\_state):

moves = game\_state.get\_available\_moves()

best\_move = moves[0]

best\_score = float('-inf')

for move in moves:

clone = game\_state.next\_state(move)

score = min\_play(clone)

if score > best\_score:

best\_move = move

best\_score = score

return best\_move

----------------------------------------------------------

Simulate game play and the game tree traversals are made

recursively

----------------------------------------------------------

3.

def min\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state) //calculate game scores

moves = game\_state.get\_available\_moves()

best\_score = float('inf') //inf says that player wins

for move in moves:

clone = game\_state.next\_state(move)

score = max\_play(clone)

if score < best\_score:

best\_move = move

best\_score = score

return best\_score

def max\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state)

moves = game\_state.get\_available\_moves()

best\_score = float('-inf') //-inf says that player loses

for move in moves:

clone = game\_state.next\_state(move)

score = min\_play(clone)

if score > best\_score:

best\_move = move

best\_score = score

return best\_score

def min\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state)

----------------------------------------------------------

def min\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state)

return min(scores) # Incomplete

----------------------------------------------------------

def min\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state)

return min(

map(lambda move: max\_play(game\_state.next\_state(move)),

game\_state.get\_available\_moves())

----------------------------------------------------------

def minimax(game\_state):

return max(

map(lambda move: (move, min\_play(game\_state.next\_state(move))),

game\_state.get\_available\_moves()),

key = lambda x: x[1])

----------------------------------------------------------

def minimax(game\_state):

return max(

map(lambda move: (move, min\_play(game\_state.next\_state(move))),

game\_state.get\_available\_moves()),

key = lambda x: x[1])

def min\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state)

return min(

map(lambda move: max\_play(game\_state.next\_state(move)),

game\_state.get\_available\_moves()))

def max\_play(game\_state):

if game\_state.is\_gameover():

return evaluate(game\_state)

return max(

map(lambda move: min\_play(game\_state.next\_state(move)),

game\_state.get\_available\_moves()))