```
# Skeleton Program for the AQA A1 Summer 2019 examination
# this code should be used in conjunction with the Preliminary Material
# written by the AQA AS1 Programmer Team
# developed in a Python 3 environment

# Version number: 0.1.3


SPACE = '     '
UNUSED = 'XXXXX'

BOARD_SIZE = 8
NUMBER_OF_PIECES = 12
MAX_MOVES = 50
ROW = 0
COLUMN = 1
DAME = 2

class MoveRecord:
  def __init__(self):
    self.Piece = ''
    self.NewRow = -1
    self.NewColumn = -1
    self.CanJump = False

def LoadPieces(FileHandle, PlayersPieces):
  for Index in range(NUMBER_OF_PIECES + 1):
    PlayersPieces[Index][ROW] = int(FileHandle.readline())
    PlayersPieces[Index][COLUMN] = int(FileHandle.readline())
    PlayersPieces[Index][DAME] = int(FileHandle.readline())
  return PlayersPieces

def CreateNewBoard(Board):
  for ThisRow in range(BOARD_SIZE):
    for ThisColumn in range(BOARD_SIZE):
      if (ThisRow + ThisColumn) % 2 == 0:
        Board[ThisRow][ThisColumn] = UNUSED
      else:
        Board[ThisRow][ThisColumn] = SPACE
  return Board

def AddPlayerA(Board, A):
  for Index in range(1, NUMBER_OF_PIECES + 1):
    PieceRow = A[Index][ROW]
    PieceColumn = A[Index][COLUMN]
    PieceDame = A[Index][DAME]
    if PieceRow > -1:
      if PieceDame == 1:
        Board[PieceRow][PieceColumn] = 'A' + str(Index)
      else:
        Board[PieceRow][PieceColumn] = 'a' + str(Index)
  return Board
```

```
def AddPlayerB(Board, B):
  for Index in range(1, NUMBER_OF_PIECES + 1):
    PieceRow = B[Index][ROW]
    PieceColumn = B[Index][COLUMN]
    PieceDame = B[Index][DAME]
    if PieceRow > -1:
      if PieceDame == 1:
        Board[PieceRow][PieceColumn] = 'B' + str(Index)
      else:
        Board[PieceRow][PieceColumn] = 'b' + str(Index)
  return Board

def DisplayErrorCode(ErrorNumber):
  print('Error ', ErrorNumber)

def SetUpBoard(Board, A, B, FileFound):
  FileName = 'game1.txt'
  Answer = input('Do you want to load a saved game? (Y/N): ')
  if Answer == 'Y' or Answer == 'y':
    FileName = input('Enter the filename: ')
  try:
    FileHandle = open(FileName, 'r')
    FileFound = True
    A = LoadPieces(FileHandle, A)
    B = LoadPieces(FileHandle, B)
    FileHandle.close()
    Board = CreateNewBoard(Board)
    Board = AddPlayerA(Board, A)
    Board = AddPlayerB(Board, B)
  except:
    DisplayErrorCode(4)
  return Board, A, B, FileFound

def PrintHeading():
  print('    ', end='')
  for BoardColumn in range(BOARD_SIZE):
    print('{0:3}'.format(BoardColumn), end='   ')
  print()

def PrintRow(Board, ThisRow):
  print('   |', end='')
  for BoardColumn in range(BOARD_SIZE):
    if Board[ThisRow][BoardColumn] == UNUSED:
      print(Board[ThisRow][BoardColumn], end='|')
    else:
      print(SPACE, end='|')
  print()

def PrintMiddleRow(Board, ThisRow):
  print('{0:>2}'.format(ThisRow), end=' |')
  for BoardColumn in range(BOARD_SIZE):
```

```
    if Board[ThisRow][BoardColumn] == UNUSED or Board[ThisRow][BoardColumn] ==
SPACE:
      print(Board[ThisRow][BoardColumn], end='|')
    else:
     print('{0:>4}'.format(Board[ThisRow][BoardColumn]), end=' |')
  print()

def PrintLine():
  print('   ', end='')
  for BoardColumn in range(BOARD_SIZE):
    print('------', end='')
  print('-')

def DisplayBoard(Board):
  PrintHeading()
  PrintLine()
  for ThisRow in range(BOARD_SIZE):
    PrintRow(Board, ThisRow)
    PrintMiddleRow(Board, ThisRow)
    PrintRow(Board, ThisRow)
    PrintLine()

def PrintPlayerPieces(A, B):
  print()
  print('Player A:')
  print(A)
  print('Player B:')
  print(B)
  print()

def ClearList(ListOfMoves):
  for Index in range(MAX_MOVES):
    ListOfMoves[Index].Piece = ''
    ListOfMoves[Index].NewRow = -1
    ListOfMoves[Index].NewColumn = -1
    ListOfMoves[Index].CanJump = False
  return ListOfMoves

def ValidMove(Board, NewRow, NewColumn):
  Valid = False
  if NewRow in range(BOARD_SIZE) and NewColumn in range(BOARD_SIZE):
    if Board[NewRow][NewColumn] == SPACE:
      Valid = True
  return Valid

def ValidJump(Board, PlayersPieces, Piece, NewRow, NewColumn):
  Valid = False
  MiddlePiece = ''
  Player = Piece[0].lower()
  Index = int(Piece[1:])
  if Player == 'a':
    OppositePiecePlayer = 'b'
```

```
  else:
    OppositePiecePlayer = 'a'
  if NewRow in range(BOARD_SIZE) and NewColumn in range(BOARD_SIZE):
    if Board[NewRow][NewColumn] == SPACE:
      CurrentRow = PlayersPieces[Index][ROW]
      CurrentColumn = PlayersPieces[Index][COLUMN]
      MiddlePieceRow = (CurrentRow + NewRow) // 2
      MiddlePieceColumn = (CurrentColumn + NewColumn) // 2
      MiddlePiece = Board[MiddlePieceRow][MiddlePieceColumn]
      MiddlePiecePlayer = MiddlePiece[0].lower()
      if MiddlePiecePlayer != OppositePiecePlayer and MiddlePiecePlayer != ' ':
        Valid = True
  return Valid

def ListPossibleMoves(Board, PlayersPieces, NextPlayer, ListOfMoves):
  if NextPlayer == 'a':
    Direction = 1
  else:
    Direction = -1
  NumberOfMoves = 0
  for i in range(1, NUMBER_OF_PIECES + 1):
    Piece = NextPlayer + str(i)
    CurrentRow = PlayersPieces[i][ROW]
    CurrentColumn = PlayersPieces[i][COLUMN]
    if PlayersPieces[i][DAME] == 1:
      Piece = Piece.upper()
    NewRow = CurrentRow + Direction
    LeftColumn = CurrentColumn - 1
    RightColumn = CurrentColumn + 1
    if ValidMove(Board, NewRow, LeftColumn):
      print(Piece, ' can move to ', NewRow, ' , ', LeftColumn)
      NumberOfMoves += 1
      ListOfMoves[NumberOfMoves].Piece = Piece
      ListOfMoves[NumberOfMoves].NewRow = NewRow
      ListOfMoves[NumberOfMoves].NewColumn = LeftColumn
      ListOfMoves[NumberOfMoves].CanJump = False
    if ValidMove(Board, NewRow, RightColumn):
      print(Piece, ' can move to ', NewRow, ' , ', RightColumn)
      NumberOfMoves += 1
      ListOfMoves[NumberOfMoves].Piece = Piece
      ListOfMoves[NumberOfMoves].NewRow = NewRow
      ListOfMoves[NumberOfMoves].NewColumn = RightColumn
      ListOfMoves[NumberOfMoves].CanJump = False
    JumpRow = CurrentRow + Direction + Direction
    JumpLeftColumn = CurrentColumn - 2
    JumpRightColumn = CurrentColumn + 2
    if ValidJump(Board, PlayersPieces, Piece, JumpRow, JumpLeftColumn):
      print(Piece, ' can jump to ', JumpRow, ' , ', JumpLeftColumn)
      NumberOfMoves += 1
      ListOfMoves[NumberOfMoves].Piece = Piece
      ListOfMoves[NumberOfMoves].NewRow = JumpRow
      ListOfMoves[NumberOfMoves].NewColumn = JumpLeftColumn
```

```
      ListOfMoves[NumberOfMoves].CanJump = True
    if ValidJump(Board, PlayersPieces, Piece, JumpRow, JumpRightColumn):
      print(Piece, ' can jump to ', JumpRow, ' , ', JumpRightColumn)
      NumberOfMoves += 1
      ListOfMoves[NumberOfMoves].Piece = Piece
      ListOfMoves[NumberOfMoves].NewRow = JumpRow
      ListOfMoves[NumberOfMoves].NewColumn = JumpRightColumn
      ListOfMoves[NumberOfMoves].CanJump = True
  print('There are ', NumberOfMoves, ' possible moves')
  return ListOfMoves

def ListEmpty(ListOfMoves):
  if ListOfMoves[1].Piece == '':
    return True
  else:
    return False

def SelectMove(ListOfMoves):
  ValidPiece = False
  while not ValidPiece:
    Found = False
    EndOfList = False
    Piece = input('Which piece do you want to move? ')
    Index = 0
    if Piece == '':
      EndOfList = True
    while not Found and not EndOfList:
      Index += 1
      if ListOfMoves[Index].Piece == Piece:
        Found = True
      elif ListOfMoves[Index].Piece == '':
        EndOfList = True
        DisplayErrorCode(1)
    if Found:
        ValidPiece = True
  ChosenPieceIndex = Index
  ValidMove = False
  while not ValidMove:
    RowString = input('Which row do you want to move to? ')
    ColumnString = input('Which column do you want to move to? ')
    try:
      NewRow = int(RowString)
      NewColumn = int(ColumnString)
      Found = False
      EndOfList = False
      Index = ChosenPieceIndex - 1
      while not Found and not EndOfList:
        Index += 1
        if ListOfMoves[Index].Piece != Piece:
          EndOfList = True
          DisplayErrorCode(2)
        elif ListOfMoves[Index].NewRow == NewRow and
```

```
ListOfMoves[Index].NewColumn == NewColumn:
          Found = True
      ValidMove = Found
    except:
      DisplayErrorCode(3)
  return Index

def MoveDame(Board, Player, NewRow, NewColumn):
  if Player == 'a':
    for i in [1, 3, 5, 7]:
      if Board[0][i] == SPACE:
        NewColumn = i
        NewRow = 0
        break
  else:
    for i in [0, 2, 4, 6]:
      if Board[BOARD_SIZE - 1][i] == SPACE:
        NewColumn = i
        NewRow = BOARD_SIZE - 1
        break
  return NewRow, NewColumn

def MovePiece(Board, PlayersPieces, ChosenPiece, NewRow, NewColumn):
  Index = int(ChosenPiece[1:])
  CurrentRow = PlayersPieces[Index][ROW]
  CurrentColumn = PlayersPieces[Index][COLUMN]
  Board[CurrentRow][CurrentColumn] = SPACE

  if NewRow == BOARD_SIZE - 1 and PlayersPieces[Index][DAME] == 0:
    Player = 'a'
    PlayersPieces[0][1] += 1
    PlayersPieces[Index][DAME] = 1
    ChosenPiece = ChosenPiece.upper()
    NewRow, NewColumn = MoveDame(Board, Player, NewRow, NewColumn)
  elif NewRow == 0 and PlayersPieces[Index][DAME] == 0:
    Player = 'b'
    PlayersPieces[0][1] += 1
    PlayersPieces[Index][DAME] = 1
    ChosenPiece = ChosenPiece.upper()
    NewRow, NewColumn = MoveDame(Board, Player, NewRow, NewColumn)
  PlayersPieces[Index][ROW] = NewRow
  PlayersPieces[Index][COLUMN] = NewColumn
  Board[NewRow][NewColumn] = ChosenPiece
  return Board, PlayersPieces

def MakeMove(Board, PlayersPieces, OpponentsPieces, ListOfMoves, PieceIndex):
  PlayersPieces[0][0] += 1
  if PieceIndex > 0:
    Piece = ListOfMoves[PieceIndex].Piece
    NewRow = ListOfMoves[PieceIndex].NewRow
    NewColumn = ListOfMoves[PieceIndex].NewColumn
    PlayersPieceIndex = int(Piece[1:])
```

```
    CurrentRow = PlayersPieces[PlayersPieceIndex][ROW]
    CurrentColumn = PlayersPieces[PlayersPieceIndex][COLUMN]
    Jumping = ListOfMoves[PieceIndex].CanJump
    Board, PlayersPieces = MovePiece(Board, PlayersPieces, Piece, NewRow,
NewColumn)
    if Jumping:
      MiddlePieceRow = (CurrentRow + NewRow) // 2
      MiddlePieceColumn = (CurrentColumn + NewColumn) // 2
      MiddlePiece = Board[MiddlePieceRow][MiddlePieceColumn]
      print('jumped over ', MiddlePiece)
  return Board, PlayersPieces, OpponentsPieces

def SwapPlayer(NextPlayer):
  if NextPlayer == 'a':
    return 'b'
  else:
    return 'a'

def PrintResult(A, B, NextPlayer):
  print('Game ended')
  print(NextPlayer, ' lost this game as they cannot make a move')
  PrintPlayerPieces(A, B)

def Game():
  A = [[0, 0, 0] for Piece in range(NUMBER_OF_PIECES + 1)]
  B = [[0, 0, 0] for Piece in range(NUMBER_OF_PIECES + 1)]
  Board = [['' for Column in range(BOARD_SIZE)] for Row in range(BOARD_SIZE)]
  ListOfMoves = [MoveRecord() for Move in range(MAX_MOVES)]
  GameEnd = False
  FileFound = False
  NextPlayer = 'a'
  Board, A, B,  FileFound = SetUpBoard(Board, A, B, FileFound)
  if not FileFound:
    GameEnd = True
  while not GameEnd:
    PrintPlayerPieces(A, B)
    DisplayBoard(Board)
    print('Next Player: ', NextPlayer)
    ListOfMoves = ClearList(ListOfMoves)
    if NextPlayer == 'a':
      ListOfMoves = ListPossibleMoves(Board, A, NextPlayer, ListOfMoves)
      if not ListEmpty(ListOfMoves):
        PieceIndex = SelectMove(ListOfMoves)
        Board, A, B = MakeMove(Board, A, B, ListOfMoves, PieceIndex)
        NextPlayer = SwapPlayer(NextPlayer)
      else:
        GameEnd = True
    else:
      ListOfMoves = ListPossibleMoves(Board, B, NextPlayer, ListOfMoves)
      if not ListEmpty(ListOfMoves):
        PieceIndex = SelectMove(ListOfMoves)
        Board, B, A = MakeMove(Board, B, A, ListOfMoves, PieceIndex)
```

```
        NextPlayer = SwapPlayer(NextPlayer)
      else:
        GameEnd = True
  if FileFound:
    PrintResult(A, B , NextPlayer)

if __name__ == "__main__":
  Game()
```