The demo for this project can be viewed at:

https://youtu.be/9EVGJtWL_jA

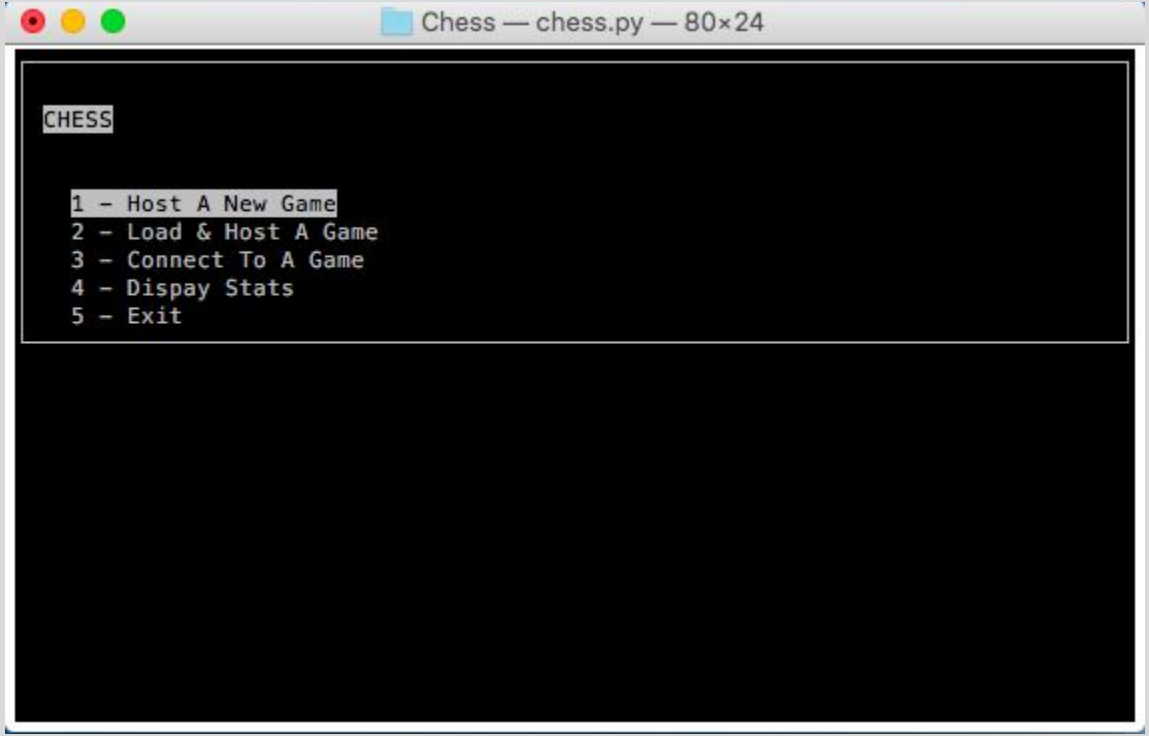
Chess

Eric Fossas * Ryan O'Connell * Chase Springer
Kevin Rau * Ryan Riley

About

- A chess game developed with Python.
- Runs in a shell.
- Sockets connect over local network.
- Features include:
  - Saving games
  - Saving stats

Use Case 1: Interact With Menu



Use Case 2: Move A Chess Piece

```
Terminal — chess.py — 80×24

  A   B   C   D   E   F   G   H
-------------------------------
| R | N | B | Q | K | B | N |   | 1
-------------------------------
| P | P | P | P | P | P | P |   | 2
-------------------------------
|   |   |   |   |   |   |   | R | 3
-------------------------------
|   |   |   |   |   |   |   | P | 4
-------------------------------
|   |   |   |   | p |   |   |   | 5
-------------------------------
|   |   | N | B |   |   |   |   | 6
-------------------------------
| p | p | p | p |   | p | p | p | 7
-------------------------------
| R |   | B | Q | K |   | N | R | 8
-------------------------------

BLACK ->
Origin:
```

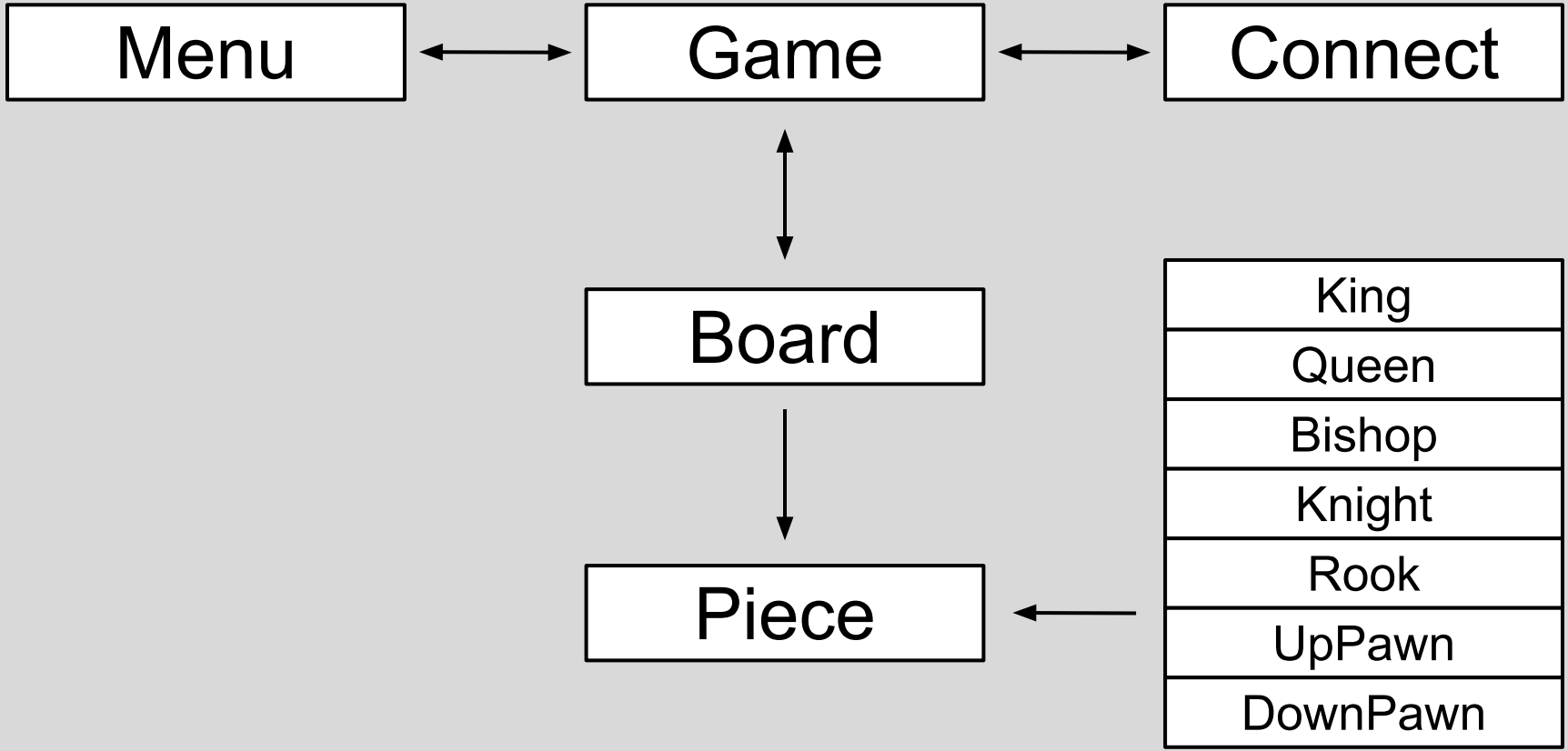
Use Case 3: Save A Game

Terminal — chess.py — 80×24

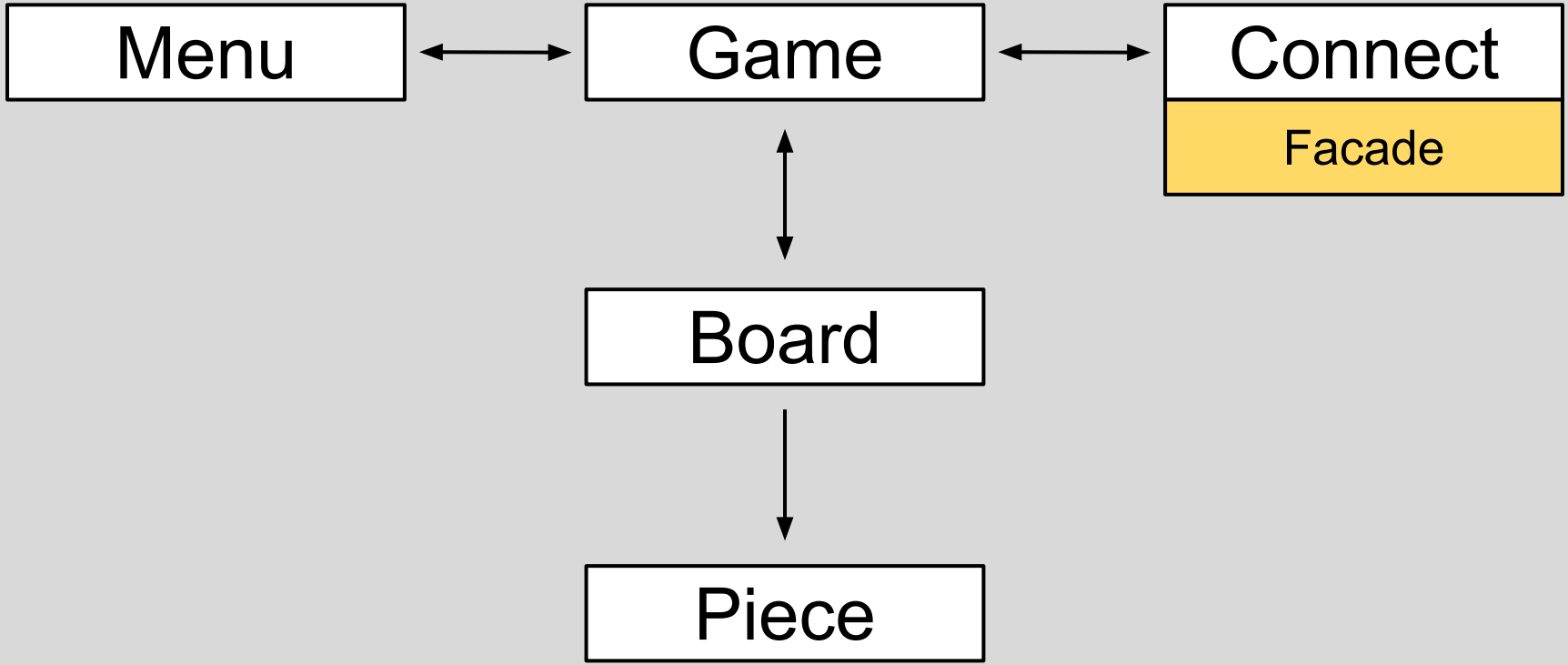
```
Origin: save
Game Saved


  A   B   C   D   E   F   G   H
 ---------------------------------
| R | N | B | Q | K | B |   |   | 1
 ---------------------------------
| P | P | P | P | P | P | P |   | 2
 ---------------------------------
|   |   | R |   |   | N |   |   | 3
 ---------------------------------
|   |   |   |   |   |   |   | Q | 4
 ---------------------------------
|   |   |   |   | p |   |   |   | 5
 ---------------------------------
|   |   | N | B |   |   |   |   | 6
 ---------------------------------
| p | p | p | p |   | p | p | p | 7
 ---------------------------------
| R |   | B |   | K |   | N | R | 8
 ---------------------------------
```


```
True,R,False,N,False,B,False,Q,False,K,False,B,False,empty,empty,empty,empty,P,False,
P,False,P,False,P,False,P,False,P,False,P,False,empty,empty,empty,empty,empty,empty,R
,False,empty,empty,empty,empty,N,False,empty,empty,empty,empty,empty,empty,empty,empt
y,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,Q,True,empty,empty,empt
y,empty,empty,empty,empty,empty,p,True,empty,empty,empty,empty,empty,empty,empty,empt
y,empty,empty,N,True,B,True,empty,empty,empty,empty,empty,empty,empty,empty,p,True,p,
True,p,True,p,True,empty,empty,p,True,p,True,p,True,R,True,empty,empty,B,True,empty,e
mpty,K,True,empty,empty,N,True,R,True,
```

Quick Overview of Classes

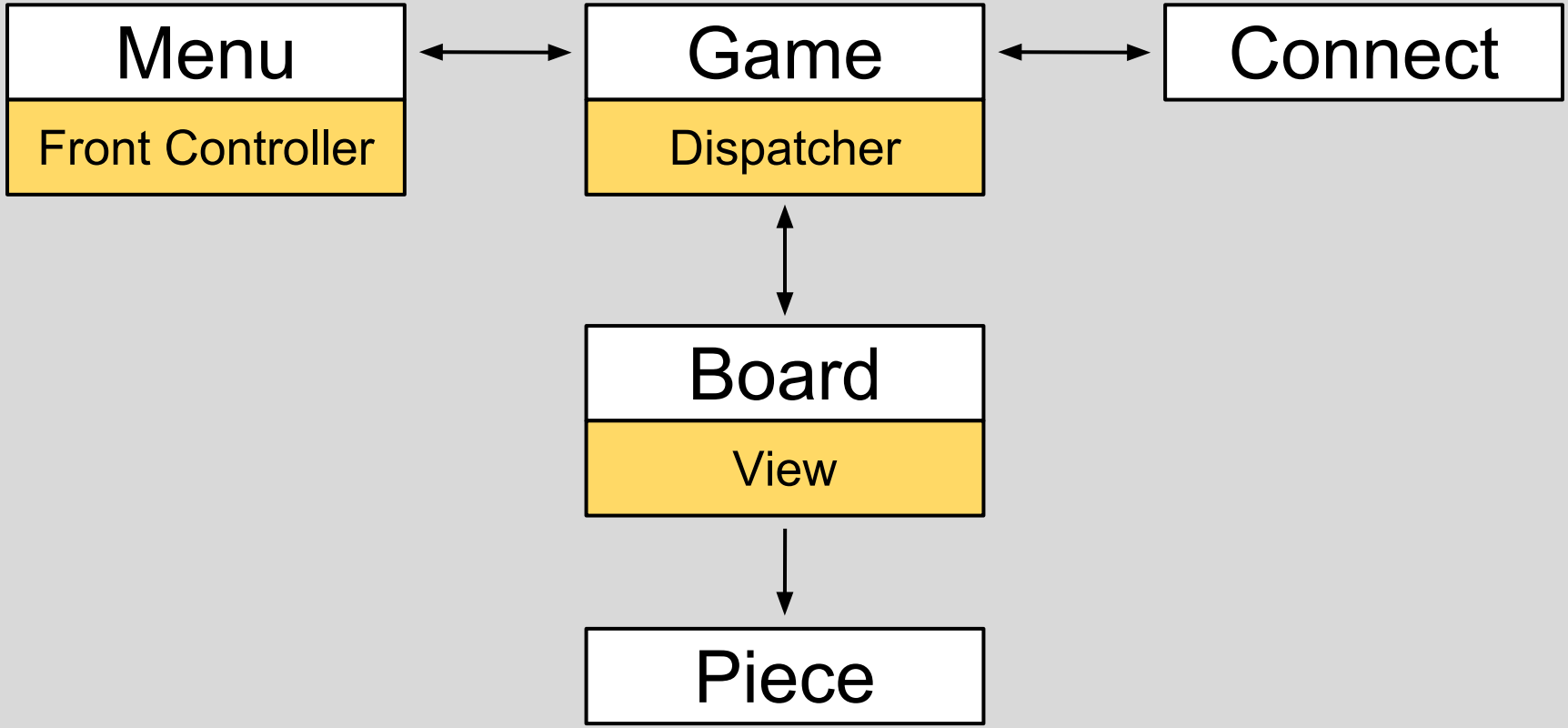


Facade Pattern

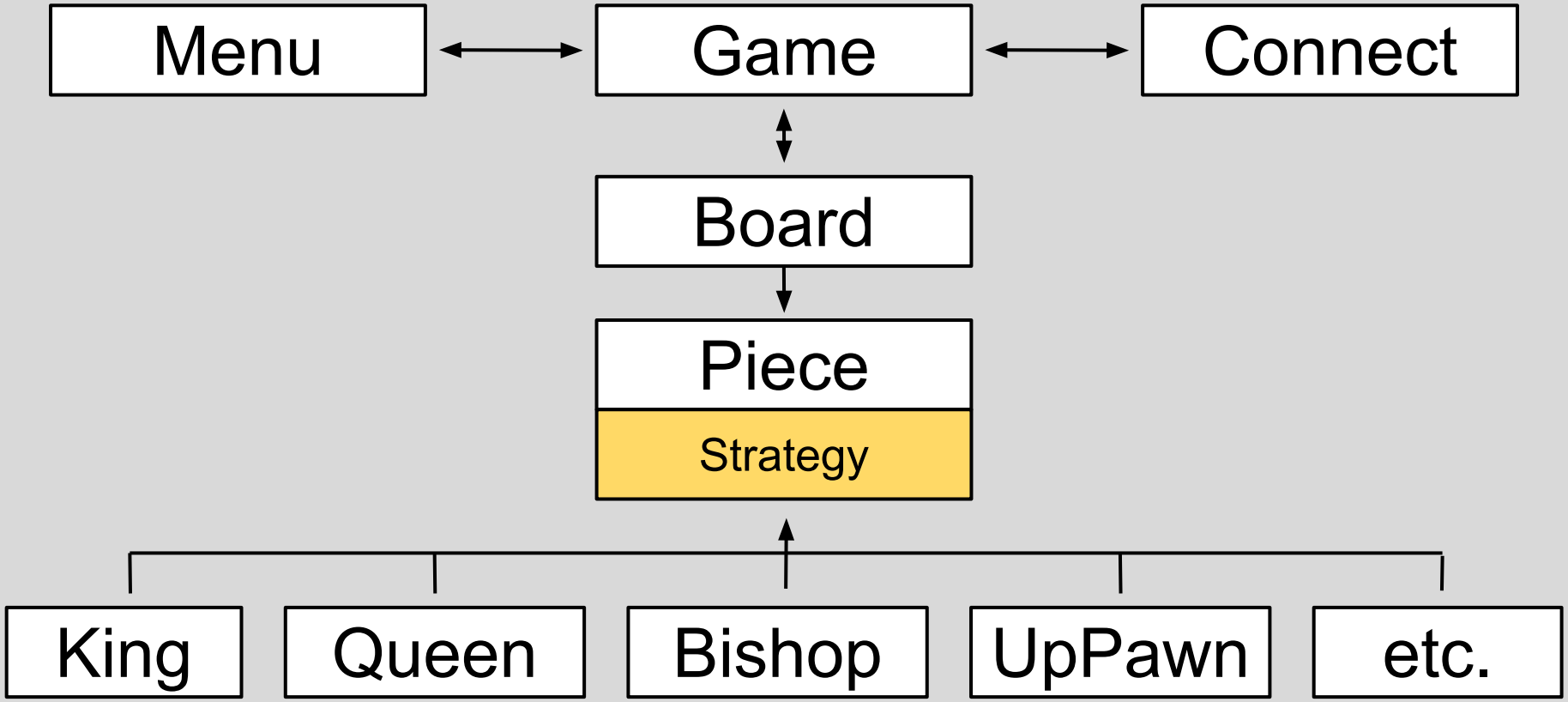


```python
# client calls this
def receiveFromHost(self):
    message = self._socket.recv(1024) # blocking, waits for connection
    return message.decode("utf-8")

# host calls this
def receiveFromClient(self):
    message = self._connection.recv(1024) # blocking, waits for connection
    return message.decode("utf-8")

# client calls this
def sendToHost(self,output):
    self._socket.send(str.encode(output))

# host calls this
def sendToClient(self,output):
    self._connection.send(str.encode(output))

# client closes their connection
def clientCloseConnection(self):
    self._socket.close()

# host closes client connection
def hostCloseConnection(self):
    self._connection.close()
    self._socket.close()
    self._socket = None

# reset the connection after a game ends
def resetConnection(self):
    self._socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self._host = socket.gethostname()
    self._port = 8080
```

Front Controller Pattern



Strategy Design Pattern



```python
class Piece:

    # id = P pawn, R rook, N knight, B bishop
    # color = False black, True white
    # state = False dead, True alive
    _id = ''
    _color = False
    _state = False
    _move = None

    def __init__(self,id,color,move):
        self.setID(id)
        self.setColor(color)
        self.setState(1)
        self.moveType = move

    def setID(self,id):
        self._id = id

    def getID(self):
        return self._id

    def setColor(self,color):
        self._color = color

    def getColor(self):
        return self._color

    def setState(self,state):
        self._state = state

    def getState(self):
        return self._state

    # this will be needed for upgrading pawn,
    def setMove(self,move):
        self.moveType = move
```

```python
def King(orig,dest,board):
    if dest[0] > orig[0] + 1 or dest[0] < orig[0] - 1:
        return False
    elif dest[1] > orig[1] + 1 or dest[1] < orig[1] - 1:
        return False
    else:
        return True
```

```python
def Knight(orig,dest,board):
    #two vertical spaces, one horizontal
    if abs(dest[0] - orig[0]) == 2 and abs(dest[1] - orig[1]) == 1:
        return True
    #one verical space, two horizontal
    elif  abs(dest[0] - orig[0]) == 1 and abs(dest[1] - orig[1]) == 2:
        return True
    else:
        return False
```

```python
def UpPawn(orig,dest,board):
    squares = []

    # first move double space move
    if orig[0] == 6 and orig[1] == dest[1] and orig[0] - 2 == dest[0]:
        squares.append((orig[0] - 1,dest[1]))
        squares.append((dest[0],dest[1]))
        valid = checkEmpty(squares,board)
        if valid:
            return True
        else:
            return False
```

Chess

Eric Fossas * Ryan O'Connell * Chase Springer
Kevin Rau * Ryan Riley