Django signals run in the same database transaction as the caller. This means that if you trigger a signal during a database operation, the signal's receiver will be part of the same transaction. If the transaction is rolled back, the changes made by the signal will also be rolled back.

```
# models.py
from django.db import models, transaction
from django.db.models.signals import post_save
from django.dispatch import receiver

class MyModel(models.Model):
    name = models.CharField(max_length=100)

class AnotherModel(models.Model):
    related_name = models.CharField(max_length=100)

@receiver(post_save, sender=MyModel)
def create_another_model(sender, instance, created, **kwargs):
    if created:
        # This will create a new instance of AnotherModel
        AnotherModel.objects.create(related_name=instance.name)

# In Django shell or a view
from myapp.models import MyModel, AnotherModel

# Start a transaction
with transaction.atomic():
    new_instance = MyModel(name='Test Instance')
    new_instance.save()  # This will trigger the post_save signal
    # Now let's raise an exception to roll back the transaction
    raise Exception("Rolling back the transaction")
```

In this code, we have two models: **MyModel** and **AnotherModel**. The signal receiver **create_another_model** creates an instance of **AnotherModel** whenever a **MyModel** instance is created.

When we run the code in the Django shell or a view, we start a transaction using **transaction.atomic()**. We create and save a new **MyModel** instance, which triggers the signal and tries to create an **AnotherModel** instance. However, we then raise an exception to roll back the transaction.