```
mainmodel Component 1 {
    aux ActualWeather {
        type Integer
        unit DISPLAYAS(Weather)
        def IF(RandomSkill<ForecastSkill,
                {Hist_ActualWthr,Drier_ActualWthr,Wetter_ActualWthr}[INDEX(ClimateSwitch)]
                ,
                RndmHist
                )
            //if forecast skill is greater than randomskill, then actual weather from the forecast is used,
            //else random historical weather variable is used
    }
    aux AdaptiveCrop {
        autotype Integer
        unit DISPLAYAS(Crops)
        dim Farmer_Perception
        def IF(WetWeather>=7,1,
                IF(WetWeather<3 AND DryWeather>=6,
                DryCrop,
                NormalCrop))
            //if %wet is more than/= 70%, then farmer will plant paddy, otherwise farmer will look at probability of dry
            weather to normal weather breakdown
            //if %dry is more than/= 60%, then farmer will plant onion (assuming trust in mrkt isn't low)
            //else farmer will plant soybean (assuming trust in mrkt isn't low)
            //crops: 1-paddy, 2-soybean, 3-onion, 4-fallow
    }
    aux AllOptions {
        type Integer
        dim 1..10
        def {1,1,1,1,1,1,1,2,3,4}
    }
    level Bank Acct {
        autotype Real
        dim FarmerPlantingApproach
        init InitialIncome
        inflow { autodef  Net_Return }
    }
    aux ChangeinMrktTrust {
        autotype Real
        autodim FarmerPlantingApproach
        def Seasonal_Mrkt_Exp
            //overall trust in market is adjusted by that season's market experience
        zeroorder
    }
    aux ChangeinWthrTrust {
        autotype Real
        autounit %
        autodim FarmerPlantingApproach
        def 'Seasonal WthrExperience'
            //farmer's weather trust for the future changes based on that season's weather experience
        zeroorder
    }
    const ClimateSwitch {
        type Integer
        init 1
    }
    aux Crop Yield {
        autotype Integer
        unit DISPLAYAS(Yields)
        autodim FarmerPlantingApproach
        def FOR (i = FarmerPlantingApproach|
                OutcomeYield[INDEX(CropPlanted[i])][INDEX(ActualWeather)]
```

```
            )
        //crop yields are a function of the crop planted and the actual weather
}
aux CropCosts {
    autotype Real
    dim Crops
    def {1,2,5,0}
        //crop costs of rice, soybean, onions, and fallow
}
aux CropOptions {
    type Integer
    unit DISPLAYAS(Crops)
    dim 1..2, LE_Farmer
    def {{TrustCrop[LE_Climate],TrustCrop[LE_Forecast]},
        {RandomCrop,RandomCrop}}
        //note: rational planting for less educ farmer is still based off their previous
        //experiences to some degree but is also a bit more variable/biased towards rice
}
aux CropPlanted {
    type Integer
    unit DISPLAYAS(Crops)
    dim FarmerPlantingApproach
    def {RiceCrop,RandomCrop,
          Variable_LE_Crops[Climate],AdaptiveCrop[ME_Climate],
          Variable_LE_Crops[Forecast],AdaptiveCrop[ME_Forecast]
          }
        //array of crops planted by the farmers, "LE" less educated are more variable,
        //"ME" are relatively more rational
}
aux d1_mrkt {
    autotype Real
    autounit %
    autodim FarmerPlantingApproach
    def IF(TrustinMarket>=0%,
          100%-TrustinMarket,
          0%)
}
aux d1_wthr {
    autotype Real
    unit %
    autodim FarmerPlantingApproach
    def IF(TrustinForecast>=0%,
          100%-TrustinForecast,
          0%)
}
aux Drier_ActualWthr {
    autotype Integer
    unit DISPLAYAS(Weather)
    def //samples weather from associated seasonal forecast
        Drier_Ssnl_Frcst[INDEX(INTEGER(RANDOM(1,11)))]
}
aux Drier_Ssnl_Frcst {
    type Integer
    autodim 1..10
    def //samples associated climate probability 10 times to generate seasonal forecast
        {DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
```

```
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))],
        DrierFuture[INDEX(INTEGER(RANDOM(1,11)))]}
}
const DrierFuture {
    type Integer
    autodim 1..10
    init {1,1,1,1,1,2,2,2,2,3}
        //slightly drier relative to historical with 50% dry-40%normal-10% wet instead of 40-40-20
}
aux DryCrop {
    autotype Integer
    unit DISPLAYAS(Crops)
    dim Farmer_Perception
    def //if farmer's trust in market is poor, then will leave fallow over planting onion
        //else follow graded trust model of random probability of planting onion/fallow
        IF(TrstMrkt_Adpt_Frmr<MrktStd,
          4,
          IF(RndmMrktTrust<TrstMrkt_Adpt_Frmr,
            RndDryCrop,
            4))
}
aux DryWeather {
    autotype Real
    autodim Farmer_Perception
    def FOR (i=Farmer_Perception|
          COUNTEQ(PerceivedWeather[i],1)
        )
        //what is the probability of dry weather for the 4 rationally guided farmers: Climate (LE, ME), Frcst (LE, ME)
}
const ForecastSkill {
    autotype Real
    unit %
    init 70
}
aux Hist_ActualWthr {
    autotype Integer
    unit DISPLAYAS(Weather)
    def //samples weather from associated seasonal forecast
        Hist_Ssnl_Frcst[INDEX(INTEGER(RANDOM(1,11)))]
}
aux Hist_Ssnl_Frcst {
    type Integer
    autodim 1..10
    def //samples associated climate probability 10 times to generate seasonal forecast
        {Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))],
        Historical[INDEX(INTEGER(RANDOM(1,11)))]}
}
const Historical {
    type Integer
    dim 1..10
    init {1,1,1,1,2,2,2,2,3,3}
        //defined as 40% dry, 40%normal, and 20% wet
}
```

```
aux IncrementArray {
    autotype Real
    dim Crops, Weather
    def {{-0.1,0,0.06},{-0.1,0.06,0},{0.06,0,-0.1},{0.06,0,-0.1}}
        //rows = crops/intended weather, cols = actual weather
        //these values represent whether a farmer experiences an increase or loss in trust
        //for the combination of weather and crops
        //view the autoreport to understand the exact relationships used
}
aux IncrementFrctTrust {
    autotype Real
    autounit %
    dim FarmerPlantingApproach
    def MIN(d1_wthr,100%-d1_wthr)
        //increment used for adjusting trust
}
aux IncrementMrktTrust {
    autotype Real
    autounit %
    autodim FarmerPlantingApproach
    def MIN(d1_mrkt,100%-d1_mrkt)
        //increment used for adjusting trust
}
const InitialFrcstTrust {
    autotype Real
    unit %
    init 80
        //farmers begin with full trust in forecast
}
const InitialIncome {
    autotype Real
    init 0 //since initial income in 0, bank account is essentially net income over time
}
const InitialMrktTrust {
    autotype Real
    unit %
    init 70
}
aux MaximumReturns {
    autotype Real
    dim Crops,Yields
    def {{3,2,1,0},{4,3,2,0},{12,9,6,0},{0,0,0,0}}
        //maximum returns possible
}
aux MaxReturn_Season {
    autotype Real
    dim FarmerPlantingApproach
    def FOR (i = FarmerPlantingApproach|
          MaximumReturns[INDEX(CropPlanted[i])][INDEX('Crop Yield'[i])]
        )
        //Maximum returns for the crop planted given the yield
}
const MrktStd {
    autotype Real
    unit %
    init 50
}
const MrktTrustRatioStd {
    autotype Real
    init 0.8
        //ratio below which farmers start to lose trust in the market
}
```

```
aux Net_Return {
    autotype Real
    autodim FarmerPlantingApproach
    def 'Seasonal Return'-'Seasonal Cost'
    zeroorder
}
aux NormalCrop {
    type Integer
    dim Farmer_Perception
    def //Graded trust model
        IF(TrstMrkt_Adpt_Frmr<MrktStd,
          1,
          IF(RndmMrktTrust<TrstMrkt_Adpt_Frmr,
            2,
            1))
}
aux OnionFallow {
    type Integer
    dim 1..4
    def {3,3,3,4}
        //based on survey data, 3/4 of the time, farmers are likely to plant even when facing water shortage
        //3 = onion, 4 = fallow
}
aux OutcomeReturn {
    autotype Real
    dim Crops,Yields
    def {{3,2,1,0},{RS_W,RS_N,RS_D,0},{RO_D,RO_N,RO_W,0},{0,0,0,0}}
        //recall that order is with respect to success of crop yield rather than weather
        //so rice and soybean so best is wet weather so their success is ordered from wet to dry
        //onions do better in dry weather so its success is ordered from dry to wet
}
aux OutcomeYield {
    autotype Integer
    unit DISPLAYAS(Yields)
    dim Crops,Weather
    def {{3,2,1},{3,2,1},{1,2,3},{4,4,4}}
        //rows = crops/intended weather, cols = actual weather
}
aux PerceivedWeather {
    autotype Real
    dim Farmer_Perception, 1..10
    def {{Historical,Historical,Hist_Ssnl_Frcst,Hist_Ssnl_Frcst},
        {DrierFuture,DrierFuture,Drier_Ssnl_Frcst,Drier_Ssnl_Frcst},
        {WetterFuture,WetterFuture,Wetter_Ssnl_Frcst,Wetter_Ssnl_Frcst}}[INDEX(ClimateSwitch)]
}
aux RandomCrop {
    type Integer
    unit DISPLAYAS(Crops)
    def AllOptions[INDEX(INTEGER(RANDOM(1,11)))]
}
aux RandomSkill {
    autotype Real
    unit %
    def RANDOM(0,1)
        //RANDOM(0,1)
}
aux Ratio_Actual_Expctd {
    autotype Real
    autodim FarmerPlantingApproach
    def IF(MaxReturn_Season=0,1,'Seasonal Return' DIVZ0 MaxReturn_Season)
        //if max return is 0, then ratio is 1 to ensure that farmers don't experience any risk averseness
        //else, calculate ratio of seasonal return to max return (divz0 just in case)
```

```
}
aux RiceCrop {
    autotype Integer
    unit DISPLAYAS(Crops)
    def 1
}
aux RndDryCrop {
    type Integer
    def OnionFallow[INDEX(INTEGER(RANDOM(1,5)))]
}
aux RndmHist {
    type Integer
    unit DISPLAYAS(Weather)
    def //samples weather from associated seasonal forecast
        Historical[INDEX(INTEGER(RANDOM(1,11)))]
}
aux RndmIndex {
    type Integer
    def INTEGER(RANDOM(1,3))
}
aux RndmMrktTrust {
    autotype Real
    unit %
    def RANDOM(0,1)
}
aux RndmWthrTrust {
    autotype Real
    unit %
    def RANDOM(0,1)
}
aux RO_D {
    autotype Real
    def //INTEGER(RANDOM(1,13))
        INTEGER(RANDOM(5,16))
        //based on dry weather return from game return sheet
}
aux RO_N {
    autotype Real
    def //INTEGER(RANDOM(1,10))
        INTEGER(RANDOM(4,13))
        //based on normal weather return from game return sheet
}
aux RO_W {
    autotype Real
    def //INTEGER(RANDOM(0,7))
        INTEGER(RANDOM(3,8))
        //based on wet weather return from game return sheet
        //updated to be 3-7 based on group convo
}
aux RS_D {
    autotype Real
    def INTEGER(RANDOM(1,4))
}
aux RS_N {
    autotype Real
    def INTEGER(RANDOM(2,5))
}
aux RS_W {
    autotype Real
    def INTEGER(RANDOM(3,6))
}
aux Seasonal Cost {
```

```
    autotype Real
    autodim FarmerPlantingApproach
    def FOR (i=FarmerPlantingApproach|
          CropCosts[INDEX(CropPlanted[i])]
        )
        //crop costs of actual planted crops for the different farming approaches
}
aux Seasonal Return {
    autotype Real
    autodim FarmerPlantingApproach
    def FOR (i=FarmerPlantingApproach|
          OutcomeReturn[INDEX(CropPlanted[i])][INDEX('Crop Yield'[i])]
        )
        //crop returns of actual planted crops for the different farming approaches
}
aux Seasonal WthrExperience {
    autotype Real
    autounit %
    dim FarmerPlantingApproach
    def FOR (i=FarmerPlantingApproach|
          IncrementArray[INDEX(CropPlanted[i])][INDEX(ActualWeather)]*IncrementFrctTrust[i]
        )
        //trust in weather is adjusted based on the crop planted and actual weather realized
}
aux Seasonal_Mrkt_Exp {
    autotype Real
    autodim FarmerPlantingApproach
    def IF(CropPlanted=1 OR CropPlanted=4,0,
          IF(Ratio_Actual_Expctd<=MrktTrustRatioStd,IncrementMrktTrust*-0.1,IncrementMrktTrust*0.06))
        //trust in market is adjusted only for soybean and onion crops
        //lossess in trust are weighted more than gains
}
aux SeasonIncrease {
    autotype Real
    autounit qtr^-1
    def 1/1<<qtr>>
}
level SeasonNo {
    autotype Real
    init 1
    inflow { autodef  SeasonIncrease }
}
aux TrstFrcst_Adpt_Frmr {
    autotype Real
    autounit %
    dim Farmer_Perception
    def {TrustinForecast[Adaptive_LessEduc_Climate],TrustinForecast[Adaptive_MoreEduc_Climate],
        TrustinForecast[Adaptive_LessEduc_Forecast],TrustinForecast[Adaptive_MoreEduc_Forecast]
        }
}
aux TrstMrkt_Adpt_Frmr {
    autotype Real
    autounit %
    dim Farmer_Perception
    def {TrustinMarket[Adaptive_LessEduc_Climate],TrustinMarket[Adaptive_MoreEduc_Climate],
        TrustinMarket[Adaptive_LessEduc_Forecast],TrustinMarket[Adaptive_MoreEduc_Forecast]
        }
}
aux TrustCrop {
    type Integer
    unit DISPLAYAS(Crops)
    autodim Farmer_Perception
```

```
    def //if trust in weather is low, risk averse behavior is rice farming
        //else follow graded trust of using weather info based on randomly generated weather trust
        IF(TrstFrcst_Adpt_Frmr<WthrTrustStd,
          {RiceCrop,RiceCrop,RiceCrop,RiceCrop},
          IF(RndmWthrTrust<TrstFrcst_Adpt_Frmr,
            AdaptiveCrop,
            {RiceCrop,RiceCrop,RiceCrop,RiceCrop}))
}
level TrustinForecast {
    autotype Real
    unit %
    dim FarmerPlantingApproach
    init InitialFrcstTrust
    inflow { autodef  ChangeinWthrTrust }
}
level TrustinMarket {
    autotype Real
    unit %
    dim FarmerPlantingApproach
    init InitialMrktTrust
    inflow { autodef  ChangeinMrktTrust }
}
aux Variable_LE_Crops {
    type Integer
    unit DISPLAYAS(Crops)
    dim LE_Farmer
    def CropOptions[INDEX(RndmIndex)]
        //note: less educ farmers are consistent with behavior across perceptions
        //(i.e., all plant random or all plant rationally)
}
aux Wetter_ActualWthr {
    type Integer
    unit DISPLAYAS(Weather)
    def //samples weather from associated seasonal forecast
        Wetter_Ssnl_Frcst[INDEX(INTEGER(RANDOM(1,11)))]
}
aux Wetter_Ssnl_Frcst {
    type Integer
    dim 1..10
    def //samples associated climate probability 10 times to generate seasonal forecast
        {WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))],
        WetterFuture[INDEX(INTEGER(RANDOM(1,4)))]}
}
const WetterFuture {
    type Integer
    autodim 1..10
    init {1,1,1,2,2,2,2,3,3,3}
        //slightly wetter relative to historical with 30% dry-40%normal-30% wet instead of 40-40-20
}
aux WetWeather {
    autotype Real
    dim Farmer_Perception
    def FOR (i=Farmer_Perception|
          COUNTEQ(PerceivedWeather[i],3)
```

```
            )
            //what is the probability of wet weather for the 4 rationally guided farmers: Climate (LE, ME), Frcst (LE, ME)
    }
    const WthrTrustStd {
        autotype Real
        unit %
        init 30
    }
}
range Crops {
    def Rice, Soybean, Onions, Fallow
}
range Farmer_Perception {
    def LE_Climate, ME_Climate, LE_Forecast, ME_Forecast
}
range FarmerPlantingApproach {
    def Rice, Random, Adaptive_LessEduc_Climate, Adaptive_MoreEduc_Climate, Adaptive_LessEduc_Forecast, Adaptive_MoreEduc_Forecast
    doc 1: rice farmer
        2: random farmer
        3: adaptive_less educ farmer
        4: adaptive_more educ farmer
        HP vs DP vs WP for the farmer types 3 and 4 indicate type of perception at play:
        historical perception vs drying perception vs wetter forecast perception
}
range LE_Farmer {
    def Climate, Forecast
}
range Qtr {
    def Qtr1, Qtr2, Qtr3, Qtr4
}
range Weather {
    def Dry, Normal, Wet
}
range Yields {
    def Successful, Normal, Poor, None
}
unit Chip {
    def ATOMIC
}
unit dimensionless {
    def 1
}
unit LKR {
    def __CURRENCY("LKR")
}
```