Week 7 - Object Oriented Programming

```
library(ggplot2)
library(scatterplot3d)
```

No discussion of R would be complete without an examination of the object oriented capabilities. The R programming community has embraced many of the same constructs found in other object oriented languages, so a good understanding of these topics and the creation of classes is crucial to unlocking more advanced parts of the language. This would include advanced simulations and the creation of new R packages.

This week's assignment is about creating a farming simulation.

Weekly Learning Objectives

Create an R script with the following components:

1. Create a S3 class named crops that will track soybeans, wheat, and corn harvests. Include a function to evaluate the mean of all three crops, and include a simple error handling routine.
2. Create a S4 class named workers that will track farm workers harvesting those same crops. Assume that each worker has a skillset that is unique to each crop, plus another set of workers that can harvest multiple types. Include a function that will provide summary information about those workers.
3. Create an inherited class based on one of the classes you have created.
4. Build a farming simulation based on the combination of both workers and crops. Use the classes that were created in the previous steps and have it set to calculate yearly projections. Create input variables for each of the components, and run the simulation for a five year cycle.

1. Create a S3 class for Crops

S3 classes are one of the most tricky things to master in R. For those who may be familiar with object oriented programming language, the S4 class may be slightly easier to implement. In addition to the text, I would strongly encourage you to read the R documentation sections:

https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/class https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/UseMethod

Full Points

For full points you need to write a similar class.

- Create a constructor function
- Create a print function for the class
- Create the class

Given the difficulty of initial understanding of classes, there will be some flexibility in grading this section.

In this example, the class is dependent on a constructor function called crops. In that, we will setup a data frame that is used through the rest of the class. When you see the .crops functions, it will be referring to elements within this data frame.

```r
# A crops constructor.  Create one of these for your class.  This is executed when the first instance o
# is created.  In this case
crops <- function(aCropType,
                  aCropYield,
                  aYear,
                  aFieldNumber) {
  aLen <- length(aCropType)                                          # store the length of aCropTy
  aDataFrame <- NULL                                                 # initialize aDataFrame
  if ((aLen != length(aCropYield) |                                  # Multiple OR statements (use
       (aLen != length(aYear)) | (aLen != length(aFieldNumber)))) {
    cat("Error: Parameters need to be of the same length!")          # Send out an error message
                                                                     #   This is your error handle
  } else {
    aDataFrame <-                                                    # initialize your data frame
      data.frame(
        cropType = aCropType,
        cropYield = aCropYield,
        year = aYear,
        fieldNumber = aFieldNumber
      )
    class(aDataFrame) <- "crops"                                     # initialize your class using
  }                                                                  # and call it crops
  return(aDataFrame)
}

# Implement a print function for the class
print.crops <- function(obj) {            #  This is extremely basic.
  aDF <- data.frame(unclass(obj))         #  Unclass the object, turn it into a dataframe
  print(aDF)                              #  Print the dataframe
}
```

New Class Functions

New Functions

- cropMean() calculates the crop mean
- cropYearlyMean() calculates the crop yearly mean
- addCrops() adds or merges crops

Default Functions

- cropMean.default()
- cropYearlMean.default()
- addCrops.default()

Extending Functions

- cropMean.crops()
- cropYearlyMean.crops()

- addCrops.crops()

```
# Create new class function
cropMean <- function(anObj) {
  UseMethod("cropMean")
}
# The default could have handled the mean, but built this to show a generic class and an extension
cropMean.default <- function(anObj) {
  cat("This is a generic function\n")
}
cropMean.crops <- function(anObj) {
  anObj <- data.frame(unclass(anObj))
  aResult <- mean(anObj$cropYield)
  return(aResult)
}


cropYearlyMean <- function(anObj) {
  UseMethod("cropYearlyMean")
}
cropYearlyMean.default <- function(anObj) {
  cat("This is a generic function\n")
}
cropYearlyMean.crops <- function(anObj) {
  anObj <- data.frame(unclass(anObj))
  aResult <- aggregate(anObj[, "cropYield"], list(year = anObj$year), mean)
  colnames(aResult) <- c("Year", "Mean")
  return(aResult)
}


# Create new addCrops class function.  This will add or merge crops into a single data structure.
addCrops <- function(anObj, aNewObj) {
  UseMethod("addCrops")
}
addCrops.default <- function(anObj, aNewObj) {
  cat("This is a generic function\n")
}

addCrops.crops <- function(anObj, aNewObj) {
  anObj <- data.frame(unclass(anObj))
  aNewObj <- data.frame(unclass(aNewObj))
  aResult <- rbind(anObj, aNewObj)
  class(aResult) <- "crops"
  return(aResult)
}
```

Testing the Crops S3 class

Now let's test it out. We will create a new instance called aTestCrop of class crops. We will initially populate it with soybeans, but then add wheat and corn to it. *VERY IMPORTANT* notice that there are five elements in each of these function calls to ensure we do not have a parameter mismatch.

For the test of the error handling, we want to cause that problem. Instead of using five elements for the second parameter, we will use six.

```r
# Test new class
aTestCrop <- crops(             # NOTICE each of these have FIVE elements.  That is important!
    rep("soybeans", 5),         # Remember the use of the repeat function
    aCropYield = 6:10,          # Setup crop yields as numbers between 6 and 10
    aYear = c(1, 1, 1, 1, 1),   # Setup years as a concatenation of year 1
    aFieldNumber = 1:5          # Setup field number as 1 - 5
)
class(aTestCrop)
```

```
## [1] "crops"
```

```r
cropMean(aTestCrop)
```

```
## [1] 8
```

```r
aTestCrop
```

```
##   cropType cropYield year fieldNumber
## 1 soybeans         6    1           1
## 2 soybeans         7    1           2
## 3 soybeans         8    1           3
## 4 soybeans         9    1           4
## 5 soybeans        10    1           5
```

```r
aNewTestCrop <- crops(rep("wheat", 5), 6:10, 1:5, 1:5)
aTestCrop <- addCrops(aTestCrop, aNewTestCrop)

aNewTestCrop <- crops(rep("corn", 10), 1:10, 1:10, 1:10)
aTestCrop <- addCrops(aTestCrop, aNewTestCrop)
cropMean(aTestCrop)
```

```
## [1] 6.75
```

```r
cropYearlyMean(aTestCrop)
```

```
##    Year      Mean
## 1     1  6.714286
## 2     2  4.500000
## 3     3  5.500000
## 4     4  6.500000
## 5     5  7.500000
## 6     6  6.000000
## 7     7  7.000000
## 8     8  8.000000
## 9     9  9.000000
## 10   10 10.000000
```

```r
aTestCrop
```

```
##    cropType cropYield year fieldNumber
## 1  soybeans         6    1           1
## 2  soybeans         7    1           2
## 3  soybeans         8    1           3
## 4  soybeans         9    1           4
## 5  soybeans        10    1           5
## 6     wheat         6    1           1
## 7     wheat         7    2           2
## 8     wheat         8    3           3
## 9     wheat         9    4           4
## 10    wheat        10    5           5
## 11     corn         1    1           1
## 12     corn         2    2           2
## 13     corn         3    3           3
## 14     corn         4    4           4
## 15     corn         5    5           5
## 16     corn         6    6           6
## 17     corn         7    7           7
## 18     corn         8    8           8
## 19     corn         9    9           9
## 20     corn        10   10          10
```

```
# Test Error Handling
aNewTestCrop <- crops(rep("soybeans", 5), 5:10, 1:5, 1:5)
```

```
## Error: Parameters need to be of the same length!
```

2. Create a S4 class for Workers

Creating a S4 class is much easier. Again, I would strongly recommend you review the R Documentation talking about S4 methods and classes.

https://www.rdocumentation.org/packages/methods/versions/3.3.2/topics/Introduction

Full Points

For full points you need to write a similar class.

- Create a Farm Worker class. It should have an ID, Crop Skillset, and indicator for multiple types
- Create a generic function for worker summary
- Create an overridden worker summary for multiple types
- Create an add worker function
- Create an overridden add worker function

Given the difficulty of initial understanding of classes, there will be some flexibility in grading this section.

```
setClass(
  "Farm_Worker",
  representation(
    ID = "numeric",
    Crop_Skillset = "character",
    Multiple_Types = "logical"
  )
)
Farm_Worker <- function(anId, aCrop_Skillset, aMultiple_Types) {
  aResult <-
    new(
      "Farm_Worker",
      ID = anId,
      Crop_Skillset = aCrop_Skillset,
      Multiple_Types = aMultiple_Types
    )
  return(aResult)
}

# Create new generic function
setGeneric("Worker_Summary",
           function(anObj) {
             standardGeneric("Worker_Summary")
           })
```

```
## [1] "Worker_Summary"
```

```
setMethod("Worker_Summary", "Farm_Worker",
          function(anObj) {
            ids <- slot(anObj, "ID")
            #    cat("Total number of workers: ",length(ids),"\n")
            multiSkilled <- sum(anObj@Multiple_Types)

            skillsetTable <-
              table(crop = anObj@Crop_Skillset,
                    multiSkill = anObj@Multiple_Types)
            #    print(skillsetTable)
            aResult <- data.frame(skillsetTable)
            return(aResult)
          })
setGeneric("Add_Worker",
           function(anObj, aNewObj) {
             standardGeneric("Add_Worker")
           })
```

```
## [1] "Add_Worker"
```

```
setMethod("Add_Worker", "Farm_Worker",
          function(anObj, aNewObj) {
            anIndex <- length(anObj@ID)
            anObj@ID <- append(anObj@ID, aNewObj@ID, anIndex)
            anObj@Crop_Skillset <-
```

```
            append(anObj@Crop_Skillset, aNewObj@Crop_Skillset, anIndex)
          anObj@Multiple_Types <-
            append(anObj@Multiple_Types, aNewObj@Multiple_Types, anIndex)
          return(anObj)
        })
```

Testing the S4 Worker Class

```
cornWorkers <- Farm_Worker(1:10, rep("corn", 10), rep(FALSE, 10))
wheatWorkers <- Farm_Worker(11:15, rep("wheat", 5), rep(FALSE, 5))
soybeanWorkers <- Farm_Worker(16:20, rep("soybeans", 5), rep(FALSE, 5))
superWorkers <- Farm_Worker(21:25, rep("corn", 5), rep(TRUE, 5))
workers <- cornWorkers
workers <- Add_Worker(workers, wheatWorkers)
workers <- Add_Worker(workers, soybeanWorkers)
workers <- Add_Worker(workers, superWorkers)
aSummary <- Worker_Summary(workers)
aSummary
```

```
##       crop multiSkill Freq
## 1     corn      FALSE   10
## 2 soybeans      FALSE    5
## 3    wheat      FALSE    5
## 4     corn       TRUE    5
## 5 soybeans       TRUE    0
## 6    wheat       TRUE    0
```

3. Create an inherited class

Creating an inherited class is relatively simple once you have mastered the creation of a class. Here we are going to create a FT Employee class that is inherited from the class Farm_Worker that we just created. We will then create a test to ensure it is working properly.

Full Points

For full points you need to write a similar function.

```
setClass("FT Employee", contains = "Farm_Worker", representation(name = "character"))
aWorker2 <-
  new(
    "FT Employee",
    ID = 1,
    Crop_Skillset = "corn",
    Multiple_Types = FALSE,
    name = "Joe"
  )

# Test
aWorker2
```

```
## An object of class "FT Employee"
## Slot "name":
## [1] "Joe"
##
## Slot "ID":
## [1] 1
##
## Slot "Crop_Skillset":
## [1] "corn"
##
## Slot "Multiple_Types":
## [1] FALSE
```

```
class(aWorker2)
```

```
## [1] "FT Employee"
## attr(,"package")
## [1] ".GlobalEnv"
```

```
Worker_Summary(aWorker2)
```

```
##   crop multiSkill Freq
## 1 corn      FALSE    1
```

```
str(aWorker2)
```

```
## Formal class 'FT Employee' [package ".GlobalEnv"] with 4 slots
##   ..@ name          : chr "Joe"
##   ..@ ID            : num 1
##   ..@ Crop_Skillset : chr "corn"
##   ..@ Multiple_Types: logi FALSE
```

4. Build a farming simulation based on the combination of workers and crops

FarmSim assumes it will take one worker per field to product 1 unit of yield. A value can be entered into either workerSD and/or fieldSD to introduce standard deviation into the simulation

Full Points

For full points you need to write a similar function.

First let's run some cleanup:

```
rm(cornWorkers)
rm(wheatWorkers)
rm(soybeanWorkers)
rm(superWorkers)
rm(workers)
rm(aSummary)
rm(aTestCrop)
rm(aNewTestCrop)
rm(aWorker2)
```

Now let's build the simulator:

```
farmSim <- function(cornWorkers = 0,
                    wheatWorkers = 0,
                    soybeanWorkers = 0,
                    multiSkilledWorkers = 0,
                    workerSD = 0,
                    cornFields = 0,
                    wheatFields = 0,
                    soybeanFields = 0,
                    fieldSD = 0,
                    yearsToSim = 1) {
  # Internal function to generate a work pool, with a normal distribution, for a year.  The work pool
  # changes are meant to simulate work force fluctuations.  Any negatives work force distributions are
  # voided out and set to zero.
  workPool <-
    function(cornWorkers = 0,
             wheatWorkers = 0,
             soybeanWorkers = 0,
             multiSkilledWorkers = 0,
             workerSD = 0) {
      nbrWorkers <- round(rnorm(1, cornWorkers, workerSD))
      if (nbrWorkers < 0)
        nbrWorkers = 0
      workers <-
        Farm_Worker(1:nbrWorkers,
                    rep("corn", nbrWorkers),
                    rep(FALSE, nbrWorkers))

      anIndex <- nbrWorkers + 1
      nbrWorkers <- round(rnorm(1, wheatWorkers, workerSD))
      if (nbrWorkers < 0)
        nbrWorkers = 0
      anEndIndex <- anIndex + nbrWorkers - 1
      newWorkers <-
        Farm_Worker(anIndex:anEndIndex,
                    rep("wheat", nbrWorkers),
                    rep(FALSE, nbrWorkers))
      workers <- Add_Worker(workers, newWorkers)

      anIndex <- anIndex + nbrWorkers + 1
      nbrWorkers <- round(rnorm(1, soybeanWorkers, workerSD))
      if (nbrWorkers < 0)
        nbrWorkers = 0
      anEndIndex <- anIndex + nbrWorkers - 1
      newWorkers <-
        Farm_Worker(anIndex:anEndIndex,
                    rep("soybeans", nbrWorkers),
                    rep(FALSE, nbrWorkers))
      workers <- Add_Worker(workers, newWorkers)

      anIndex <- anIndex + nbrWorkers + 1
      nbrWorkers <- round(rnorm(1, multiSkilledWorkers, workerSD))
      if (nbrWorkers < 0)
```

```
      nbrWorkers = 0
    anEndIndex <- anIndex + nbrWorkers - 1
    newWorkers <-
      Farm_Worker(anIndex:anEndIndex,
                  rep("multi", nbrWorkers),
                  rep(TRUE, nbrWorkers))
    workers <- Add_Worker(workers, newWorkers)

    return(workers)
  }

# This function is intended to dispense the multi-skilled workers when needed
requestMultiSkill <-
  function(nbrRequested,
           setInitialNumber = 0,
           resetNumber = FALSE) {
    if (resetNumber == TRUE) {
      aMSWC <<- setInitialNumber
    }

    aResult <- 0
    if (nbrRequested <= aMSWC) {
      aResult <- nbrRequested
      aMSWC <<- aMSWC - nbrRequested
    } else {
      aResult <- aMSWC
      aMSWC <<- 0
    }

    return(aResult)
  }

# This is the core loop for the simulation
for (i in 1:yearsToSim) {
  aWorkPool <-
    workPool(cornWorkers,
             wheatWorkers,
             soybeanWorkers,
             multiSkilledWorkers,
             workerSD)
  aSummary <- Worker_Summary(aWorkPool)

  # Number of multi-skilled crop workers.  These workers will be used after the crop specific workers
  # have been depleted.
  nbrMultSkilledWorkers <-
    sum(aSummary[which(aSummary$multiSkill == TRUE), "Freq"])
  mSW <- requestMultiSkill(0, nbrMultSkilledWorkers, TRUE)


  # Start with Corn Sim
  nbrFields <- round(rnorm(1, cornFields, fieldSD))
  nbrWorkers <-
    aSummary[which((aSummary$crop == "corn") &
```

```
                    (aSummary$multiSkill == FALSE)), "Freq"]
  if (nbrFields < nbrWorkers) {
    if (exists("aCrop") == TRUE) {
      aNewCrop <-
        crops(
          "corn",
          aCropYield = nbrFields,
          aYear = i,
          aFieldNumber = nbrFields
        )
      aCrop <- addCrops(aCrop, aNewCrop)
    } else {
      aCrop <-
        crops(
          "corn",
          aCropYield = nbrFields,
          aYear = i,
          aFieldNumber = nbrFields
        )
    }
  } else {
    if (exists("aCrop") == TRUE) {
      mSW <- requestMultiSkill(nbrFields - nbrWorkers)
      aNewCrop <-
        crops(
          "corn",
          aCropYield = nbrWorkers + mSW,
          aYear = i,
          aFieldNumber = nbrFields
        )
      aCrop <- addCrops(aCrop, aNewCrop)
    } else {
      mSW <- requestMultiSkill(nbrFields - nbrWorkers)
      aCrop <-
        crops(
          "corn",
          aCropYield = nbrWorkers + mSW,
          aYear = i,
          aFieldNumber = nbrFields
        )
    }
  }

  # Soybean Sim
  nbrFields <- round(rnorm(1, soybeanFields, fieldSD))
  nbrWorkers <-
    aSummary[which((aSummary$crop == "soybeans") &
                    (aSummary$multiSkill == FALSE)), "Freq"]
  if (nbrFields < nbrWorkers) {
    aNewCrop <-
      crops(
        "soybeans",
        aCropYield = nbrFields,
```

```
        aYear = i,
        aFieldNumber = nbrFields
      )
    aCrop <- addCrops(aCrop, aNewCrop)
  } else {
    mSW <- requestMultiSkill(nbrFields - nbrWorkers)
    aNewCrop <-
      crops(
        "soybeans",
        aCropYield = nbrWorkers + mSW,
        aYear = i,
        aFieldNumber = nbrFields
      )
    aCrop <- addCrops(aCrop, aNewCrop)
  }

  # Wheat Sim
  nbrFields <- round(rnorm(1, wheatFields, fieldSD))
  nbrWorkers <-
    aSummary[which((aSummary$crop == "wheat") &
                     (aSummary$multiSkill == FALSE)), "Freq"]
  if (nbrFields < nbrWorkers) {
    aNewCrop <-
      crops(
        "wheat",
        aCropYield = nbrFields,
        aYear = i,
        aFieldNumber = nbrFields
      )
    aCrop <- addCrops(aCrop, aNewCrop)
  } else {
    mSW <- requestMultiSkill(nbrFields - nbrWorkers)
    aNewCrop <-
      crops(
        "wheat",
        aCropYield = nbrWorkers + mSW,
        aYear = i,
        aFieldNumber = nbrFields
      )
    aCrop <- addCrops(aCrop, aNewCrop)
  }

}

cat("\nCrop Yield Mean: ", cropMean(aCrop), "\n\n")

cat("Yearly Projected Simulation Means: \n")
print(cropYearlyMean(aCrop))

cat("\n\nYearly Projected Simulation Details\n")
print(aCrop)

return(aCrop)
```

```
}
```

Testing Farm Simulation

Now let's test it out with:

- cornWorkers = 100
- wheatWorkers = 100
- soybeanWorkers = 100
- multiSkilledWorkers = 20
- workerSD = 5
- cornFields = 100
- wheatFields = 120
- soybeanFields = 125
- fieldSD = 5
- yearsToSim = 5

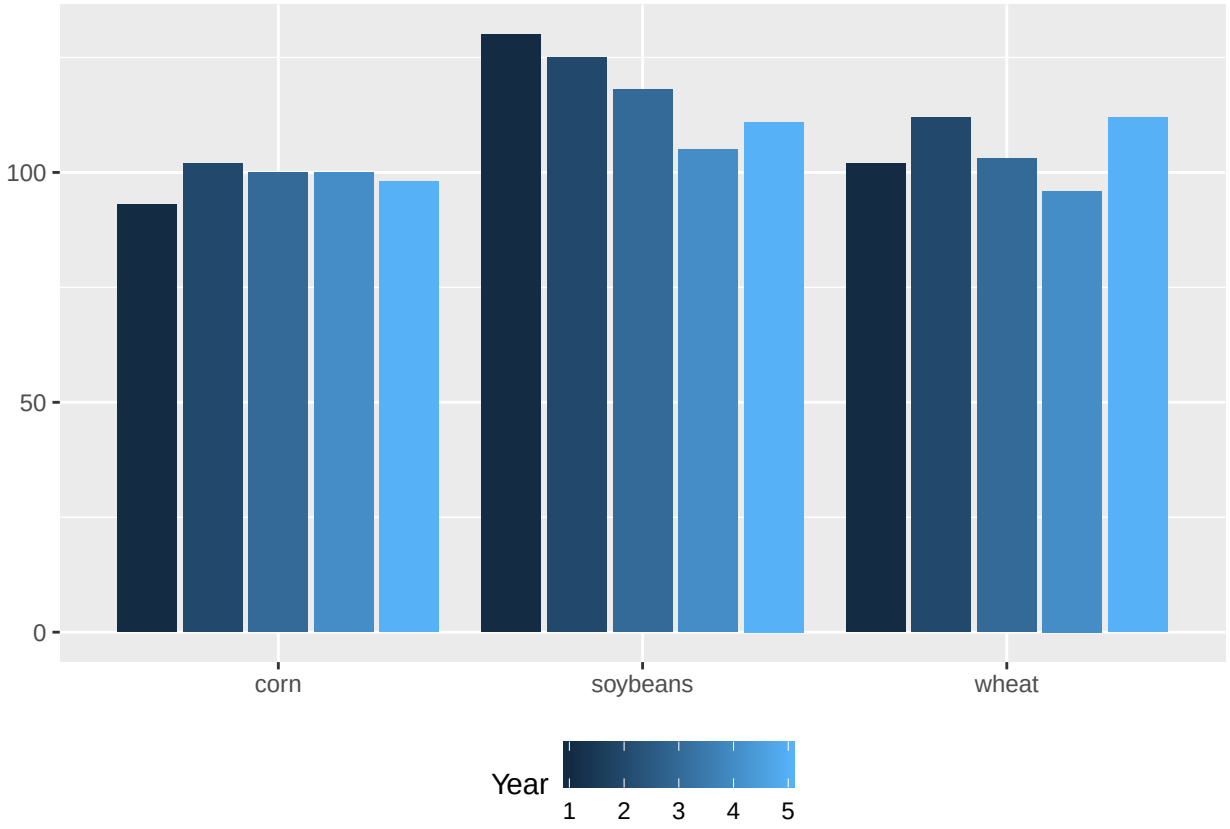
```
a <- farmSim(cornWorkers = 100,
             wheatWorkers = 100,
             soybeanWorkers = 100,
             multiSkilledWorkers = 20,
             workerSD = 5,
             cornFields = 100,
             wheatFields = 120,
             soybeanFields = 125,
             fieldSD = 5,
             yearsToSim = 5
)
```

```
##
## Crop Yield Mean:  107.1333
##
## Yearly Projected Simulation Means:
##   Year     Mean
## 1    1 108.3333
## 2    2 113.0000
## 3    3 107.0000
## 4    4 100.3333
## 5    5 107.0000
##
##
## Yearly Projected Simulation Details
##    cropType cropYield year fieldNumber
## 1      corn        93    1          93
## 2  soybeans       130    1         133
## 3     wheat       102    1         115
```

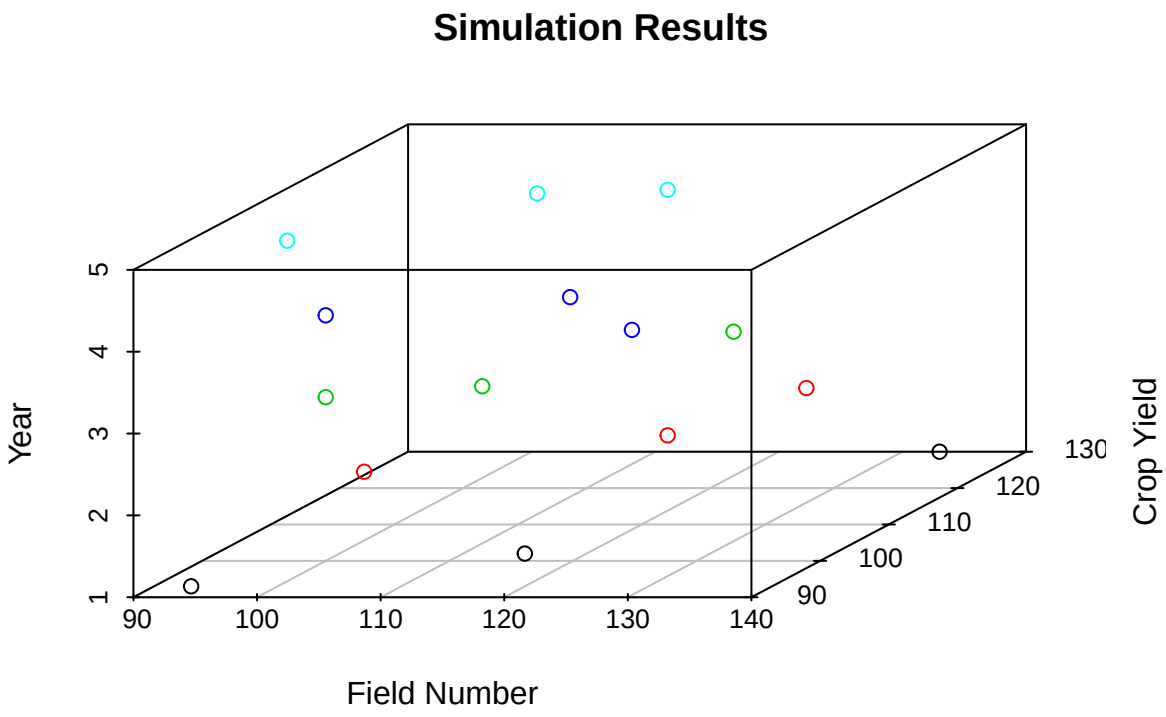
```
## 4       corn       102    2         102
## 5   soybeans       125    2         125
## 6      wheat       112    2         121
## 7       corn       100    3         100
## 8   soybeans       118    3         123
## 9      wheat       103    3         111
## 10      corn       100    4         100
## 11  soybeans       105    4         117
## 12     wheat        96    4         127
## 13      corn        98    5          98
## 14  soybeans       111    5         111
## 15     wheat       112    5         121
```

```
test_plot <- as.data.frame(unclass(a))

ggplot(data=test_plot,
       aes(x=test_plot$cropType,
           y=test_plot$cropYield,
           fill = test_plot$year
        )) +
    geom_bar(stat="identity", position = "dodge2") +
    theme(
        plot.title = element_text("Simulation Results"),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        legend.position="bottom"
        ) +
    labs(fill = "Year")
```




```
scatterplot3d(test_plot$fieldNumber,
              test_plot$cropYield,
              test_plot$year,
              main = "Simulation Results",
              xlab = "Field Number",
              ylab = "Crop Yield",
              zlab = "Year",
              color = test_plot$year)
```


Simulation Results
Year
Field Number
Crop Yield
1
2
3
4
5
90
100
110
120
130
140
90
100
110
120
130