SYSC-5104

Methodologies for Discrete

Event Modeling and Simulation

Assignment 1

A Discrete Event System Specification Model for

An Agricultural Farming System

Bruno St-Aubin – 101089879

Carleton University

Chika Chelvis Obioha Emetochukwu – 8896689

University of Ottawa

October 16th, 2017

Contents

[Conceptual Model of the Agricultural Farming System 3](#_Toc497504661)

[Description of the problem 3](#_Toc497504662)

[Conceptual Model 3](#_Toc497504663)

[Component Behavioral Description 3](#_Toc497504664)

[Formal Specification of the Agricultural Farming Model 6](#_Toc497504665)

[Types 6](#_Toc497504666)

[Atomic Models 6](#_Toc497504667)

[Coupled Models 11](#_Toc497504668)

[Experimentation strategy 13](#_Toc497504669)

[Implementation of the Agricultural Farming Model 16](#_Toc497504670)

[Implementation code 16](#_Toc497504671)

[Sample results and analysis 16](#_Toc497504672)

[Model corrections following experimentation 21](#_Toc497504673)

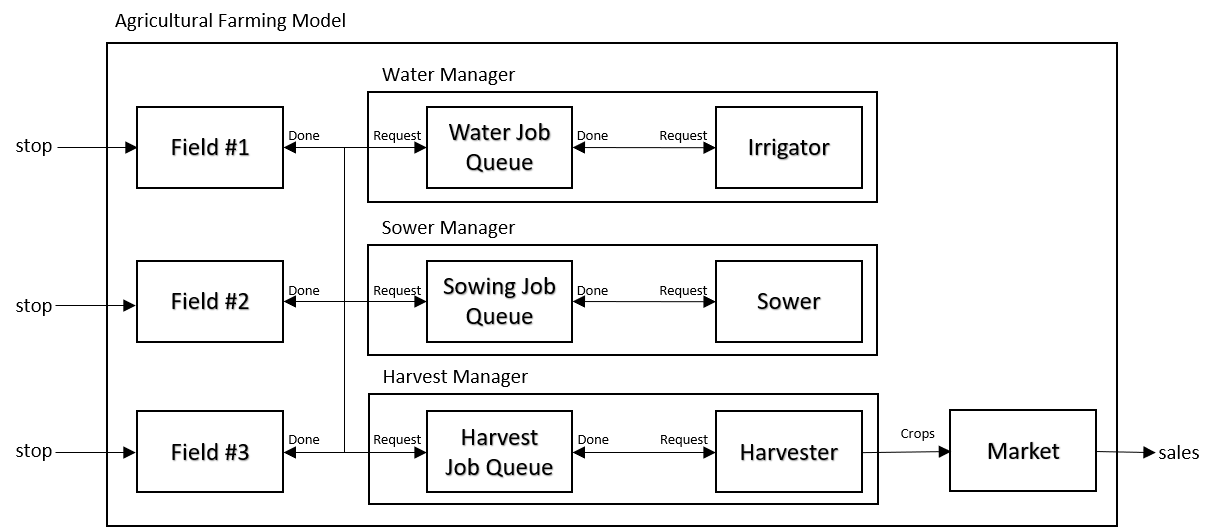
# Conceptual Model of the Agricultural Farming System

## Description of the problem

Agriculture is the root of human nutrition. Sustainable and optimized agricultural production on industrial farms is a topic that generates world-wide academic and industrial research. Industrial farms are complex, large-scale and expensive systems that involve many sub-systems in interaction with one another. For the industrial farmer, a failure in any one of the components may result in irrecoverable costs, forecasting an eventual business failure. In this context, it is understandable that testing configurations of fields, farming equipment or delivery patterns is a risky undertaking for a farmer.

In this first assignment, we propose a conceptual model of a large-scale industrial farming system. Our objective is to be able to test and optimize the configuration of the different sub-systems, the growth cycles, the crop harvesting and the patterns of produce delivery. A successful model would allow industrial farmers to optimize their farming practice while mitigating, possibly eliminating, financial risks to their business.

## Conceptual Model



## Component Behavioral Description

|  |
| --- |
| After going through multiple iterations of our models, we realized that it presented similarities with the Queue-Processor-Transducer model used as an example in the textbook. Our fields act as the generators and generate three types of jobs. Our Job Manager is the queue, it splits, holds and distributes the requests according to their types. The Irrigator, Sower and Harvester act as the processors of our model. Finally, the market acts as the transducer, converting an input into another output, in this case, crops into sales. |

Lesson Learned #1

### 1. Fields

The field component represents the cultivated area where seeds are sown and crops are grown. They generate requests that are sent to the Job Manager. These requests can be of 3 types, water requests when the field is dry, seeds request when the field is empty and harvest requests when the field is fully grown. The field receives a done signal that will allow it to proceed to the next step in its seed-water-harvest cycle. Since the field automatically generates requests forever, we have added a stop input to end the generation of jobs when required.

To make our model more interesting, we have decided to implement a crop rotation pattern. Crop rotation is a method that allows fields to maintain their productivity and soil quality over subsequent growth cycles. It involves cyclically cultivating a variety of crops that are carefully selected to avoid draining the soil of any of its nutrients. Modern crop rotation systems are usually based on the number of crops involved. In our case, we have decided to limit the complexity of our model by including only three types of crops.

Our field model has a surface parameter (acres) that will impact the amount of crops that can be harvested. They also have a growth period (days) that depends on the type of crops that are seeded. And finally, they have an evaporation rate (days) that will determine at which frequency it will output a water request.

|  |
| --- |
| The field is the model that was most difficult for us to model. When deciding how to model the field, we quickly realized that we could spend an unreasonable amount of time trying to represent a “real” cultivatable field. There are likely an infinite number of factors to consider, evaporation rate, the rain, the temperature, the type of crops, the soil quality, humidity in the air, etc. It was obvious that our model for the field would be lacking in several ways. Considering the time constraints, we settled on a very limited number of parameters to model the growth of our fields, evaporation rate and growth time.  Also note that we added a stop signal input to control when the model stop generating new inputs. The reason we did it this way is because we wanted to simulate a period spanning up to a year and the simulator did not seem to support this when creating a .bat file. |

Lesson Learned #2

### 2. Job Managers

A job manager is a coupled model that contains a queue and a processor. Our model has one job manager for each type of request that can be sent by the fields (i.e water, seeds, harvest). When the Job Manager’s queue receives a request, it holds it until the processor is available to execute it, then it sends the request to the processor. The queue can also receive a done signal, at which point the top request is dequeued and output back to the field so it can continue its growing process. Admittedly, the processors could have sent the done signals back to the 3 fields. We decided to let the queue signal the field to keep our processor models field agnostic.

Note: The queue in the job manager operates like a standard queue and will not be detailed further. The processors (irrigator, sower, harvester) are detailed below.

|  |
| --- |
| While discussing our model, we had difficulty deciding what our coupled model would contain. We could have assembled the 3 fields into a Generator coupled model, our 3 queues into a job manager model or we could have assembled a queue with a processor to create an enhanced processor. We settled on assembling each queue with its processor for two reasons. First is that conceptually, it is what made the most sense for us. Second is that it allowed us to reuse our coupled models multiple time and consequently, experiment with another feature of CD++.  In the end we realized that the choice was somewhat inconsequential because of the closure property of coupled models. Any coupled model can be substituted by its sub-models, and vice-versa, without having an impact on the global model. |

Lesson Learned #3

### 3. Irrigator

The Irrigator provides water to the fields. This component receives a request from the Water Job Queue and outputs a done signal back to the queue once finished. The irrigator has a speed parameter (acres/h) that will impact its operation time. Its operation time is determined by its speed and the area to be covered (acres).

|  |
| --- |
| We understood the complexity of modeling a system when we realized that all our models had to provide feedback to the previous models to queue and launch the next steps (i.e, Irrigator feedbacks into the Water Job Queue, Water Job Queue feedbacks into the field). Multiplying the generators, the queues and the processors increased the complexity in a non-linear manner. For this reason, we decided to remove the rain, a natural phenomenon that is difficult to model. We also removed the storage and delivery models because we deemed them trivial and less interesting than the rest of the model. |

Lesson Learned #4

### 4. Sower

The Sower provides seeds to the fields. This component receives a request from the Sowing Job Queue and outputs a done signal back to the queue once finished. The sower has a speed parameter (acres/h) that will impact its operation time. Its operation time is determined by its speed and the area to be covered (acres).

### 5. Harvester

The crop harvester is responsible of harvesting a fully-grown field. This component receives a request from the Harvest Job Queue. Once done harvesting, it outputs a done signal back to the queue and an amount of crops to be sold. The harvester has a speed parameter (Acres/h) that will impact its operation time. Its operation time is determined by its speed and the area to be covered (acres). The crops it generates is determined by the surface harvested and the yield of the type of crop being harvested (bushels/acre).

### 6. Market

The market component is responsible for selling the bushels of crops generated by the harvester. This component receives bushels of produce from the harvester. The market model has a value parameter that determines the value of each bushel of produce according to its type. For simplicity, this model does not require any duration to transduce the crops into sales. Its only output is sales.

# Formal Specification of the Agricultural Farming Model

## Types

|  |  |  |  |
| --- | --- | --- | --- |
| Queue Request field\_id Є {1, 2, 3}  surface Є R+  type Є {1, 2, 3} | | Task Request task id Є N+  surface Є R+  type Є {1, 2, 3} | |
| In CD++, it seems that it is impossible for a model to output an object. To work around this issue, we made it so our models output multiple values, representing the properties of the object we would have output otherwise. The receiving model then waits for all the outputs to have arrived before actually proceeding with the external transition function. Obviously, this has multiplied the number of ports required by most of our models by a factor of 3 (type, surface, field id). | |

Lesson Learned #5

## Atomic Models

### Field

S = phase Є {passive, active}, state Є {empty, dry, watered, ready},

growth\_times IS {type: Time}, evaporation\_times IS {type:Time},

time\_left IS {type:Time}, seed\_type Є {1, 2, 3}

X = {in\_stop, in\_done}

Y = { out\_water\_request IS {type: Queue Request},

out\_seed\_request IS {type: Queue Request},

out\_harvest\_request IS {type: Queue Request}}

δext (s, e, x)

{

if (x.port == in\_stop)

{

state = “stopped”

phase = passive

}

else if (state == “empty”) // Received seeds

{

state = “dry”

seed\_type = nextSeedType()

time\_left = getGrowthTime(seed\_type)

s.phase = active

}

else if (state == “dry”) state = “watered” // Received water

else if ( x.port == “ready” ) state = “empty” // Received harvest

}

δint (s)

{

If (state != “watered”) phase = passive

if (time\_left == 0) state = “ready” // Field becomes ready for harvest

Else

{

time\_left = time\_left – evaporation

state = “dry” // Field needs water

}

}

λ(s)

{

If (state == “dry”) Output WaterRequest Into out\_water\_request

If (state == “ready”) Output HarvestRequest Into out\_harvest\_request

If (state == “empty”) Output SeedRequest Into out\_seed\_request

}

ta(state == “watered”)

{

evaporation = getEvaporationTime()

return (evaporation < time\_left) ? evaporation : time\_left

}

ta(state == “ready” || state == “empty” || state == “dry”) return 0

ta(phase == passive) return INFINITY

|  |
| --- |
| As said before, we find our field model to be lacking in several aspects. Notably, it completely stops growing when it becomes dry. We would have liked to consider a buffer period where it continues growing for a while if it doesn’t receive water immediately. Furthermore, we have implemented a mechanism that emulates time elapsed (time left -= evaporation time), ideally, we would have used the elapsed time. Both these options would have made the code much more complex and we wouldn’t have been able to complete the assignment in the allowed time.  Additionally, it would have been safer to distinguish the input type (water done, seed done or harvest done) by using dedicated input ports instead of the previous state. In our case we assume that we receive each input sequentially (i.e Seed -> Water -> Harvest). This could lead to some inconsistencies in cases where the inputs are not received in the correct order. (ex: field is dry -> seeds received -> field is watered). We did not implement this because of the amount of ports we would’ve had to add to the model. It already has 9 output ports (3 X 3 request types) and two input ports. We would’ve needed 3 more input ports.  In this case, being able to output objects would’ve reduced the number of ports required significantly. |

Notes about the field model

### Irrigator

S = phase Є {active, passive}, speed Є R+

X = {in\_water\_request IS {type: Task Request}}

Y = {out\_done}

δint (s, e)

{

If (s.phase == active) s.phase = passive

}

δext (s, e, x {type: water request})

{

If (s.phase == passive) s.phase = active

}

λ(s)

{

Output 1 Into out\_done // 1 is done signal value

}

ta(phase == active)

{

return in\_water\_request.surface / speed

}

ta(s Є {passive}) = INFINITY

### Sower

S = phase Є {active, passive}, speed Є R+

X = {in\_seed\_request IS {type: Task Request}}

Y = {out\_done}

δint (s, e)

{

If (s.phase == active) s.phase = passive

}

δext (s, e, x)

{

If (s.phase == passive) s.phase = active

}

λ(s)

{

Output 1 Into out\_done // 1 is done signal value

}

ta(s Є {active})

{

return in\_seed\_request.surface / speed

}

ta(s Є {passive}) = INFINITY

|  |
| --- |
| In retrospective, we can see that the sower and irrigator models are identical. They should have both used the same model, for example, a generic processor. The harvester model is a little different because it outputs bushels externally. Ideally, we would’ve inherited from the generic processor model and made a subclassed model for the harvester. Because of time constraints, this was impossible. |

Lesson Learned #6

### Harvester

S = phase Є {active, passive}, speed Є R+, growth\_yields IS {type : Time}

X = {in\_harvest\_request IS {type: Task Request}}

Y = {out\_done, out\_bushels Є R+, out\_type Є {1,2,3}}

δint (s, e)

{

If (s.phase == active) s.phase = passive

}

δext (s, e, x)

{

If (s.phase == passive) s.phase = active

}

λ(s)

{

bushels = in\_harvest\_request.surface \* growth\_yields[in\_harvest\_request.type]

Output bushels Into out\_bushels

Output type Into out\_type

Output 1 Into out\_done // 1 is done signal value

}

ta(s Є {active})

{

return in\_harvest\_request.surface / speed

}

ta(s Є {passive}) = INFINITY

### Queue

S = phase Є {active, passive}, state Є {done, default},

Requests IS {type: List OF Request IS {type: Queue Request}}

X = {in \_request IS {type: Queue Request}, in\_done}

Y = {out\_task\_request IS {type: Task Request}, out\_done\_field\_1, out\_done \_field\_2, out\_done \_field\_3}

δint (s, e)

{

If (s.phase == active) s.phase = passive

}

δext (s, e, x)

{

If (x.Port == in\_done && Requests.length > 0)

{

state = done

s.phase = active

}

If (x.Port == in\_request)

{

Requests.AddToBack(in\_request)

if (Requests.length == 1) s.phase = active

}

}

λ(s)

{

If (state == done)

{

If (requests.first().field == 1) Output 1 Into out\_done\_field\_1

If (requests.first().field == 2) Output 1 Into out\_done\_field\_2

If (requests.first().field == 3) Output 1 Into out\_done\_field\_3

Requests.RemoveFirst();

State = default

}

If (Requests.length > 0)

{

TaskRequest task = ConvertToTask(Requests.First())

Output task Into out\_task\_request

}

}

ta(s Є {active}) return 0

ta(s Є {passive}) = INFINITY

### Market

S = phase Є {active, passive}, Values IS {type: List OF R+}, current\_type Є {1,2,3}, current\_bushels Є R+

X = {in\_type Є {1,2,3}, in\_bushels Є R+ }

Y = {out\_type Є {1,2,3}, out\_bushels Є R+, out\_sales Є R+}

δint (s, e)

{

s.phase = passive

}

δext (s, e, x)

{

s.current\_type = in\_type

s.current\_bushels = in\_bushels

s.phase = active

}

λ(s)

{

bushel\_value = GetValueByType(current\_type)

sales = current\_bushels \* bushel\_value

Output current\_type Into out\_type

Output current\_bushels Into out\_bushels

Output sales Into out\_sales

}

ta(s Є {active}) return 0

ta(s Є {passive}) = INFINITY

## Coupled Models

### Job Manager (Harvester Manager)

X = {in\_request IS {type: Queue Request}}

Y = {out\_field\_1, out\_field\_2, out\_field\_3, out\_bushels\*, out\_type\*}

D = {Queue, Harvester};

Md = { Mqueue, MHarvester};

EIC = {(self, in\_request), (Queue, in\_request)};

EOC = {(Harvester, out\_bushels), (self, out\_bushels); (Harvester, out\_type), (self, out\_type) ;(Queue, out\_done\_field\_1), (self, out\_field\_1) ;(Queue, out\_ done\_field\_2),(self, out\_field\_2) ;(Queue, out\_ done\_field\_3),(self, out\_field\_3) }

IC = {(Queue, out\_task\_request), (Harvester, in\_harvest\_request);(Harvester, out\_done),(Queue, in\_done) }.

\* The three processor managers (Water Manager, Sower Manager, Harvester Manager) are identical except for the output parameters out\_bushels and out\_type. These are only present in the Harvester Manager.

### Agricultural Farm

X = {in\_stop\_f1, in\_stop\_f2, in\_stop\_f3}

Y = {out\_bushels, out\_type, out\_sales}

D = {F1 IS {Type: Field}, F2 IS {Type: Field}, F3 IS {Type: Field}, SM IS {Type: Job\_Manager}, WM IS {Type: Job Manager}, HM IS {Type: Job Manager}, Market};

Md = { Mfield, MJob Manager, MMarket};

EIC = {(self, in\_stop\_f1), (F1, in\_stop); {(self, in\_stop\_f2), (F2, in\_stop); {(self, in\_stop\_f3), (F3, in\_stop)};

EOC = {(Market, out\_bushels), (self, out\_bushels) ; (Market, out\_type), (self, out\_type) ; (Market, out\_sales), (self, out\_sales)}

IC = {(F1, out\_water\_request), (WM, in \_request); (F1, out\_seed\_request), (SM, in \_request); (F1, out\_harvest\_request), (HM, in \_request); (F2, out\_water\_request), (WM, in \_request); (F2, out\_seed\_request), (SM, in \_request); (F2, out\_harvest\_request), (HM, in \_request); (F3, out\_water\_request), (WM, in \_request); (F3, out\_seed\_request), (SM, in \_request); (F3, out\_harvest\_request), (HM, in \_request); (WM, out\_field\_1), (F1, in\_done); (SM, out\_field\_1), (F1, in\_done); (HM, out\_field\_1), (F1, in\_done); (WM, out\_field\_2), (F2, in\_done); (SM, out\_field\_2), (F2, in\_done); (HM, out\_field\_2), (F2, in\_done); (WM, out\_field\_3), (F3, in\_done); (SM, out\_field\_3), (F3, in\_done); (HM, out\_field\_3), (F3, in\_done); (HM, out\_bushels),(Market, in\_bushels); (HM, out\_type),(Market, in\_type)}

|  |
| --- |
| As mentioned in the behavioral description of the job manager model, we opted to output the done signal through the queue so that our processor could remain field agnostic. Ideally, we would have made it so the queues were field agnostic as well. To achieve this, we could’ve integrated a splitter/sorter model that would’ve taken in all requests, separated them by type and sent them to the appropriate job manager. Then, the job managers could have output the done requests back to the splitter/sorter so that it could send the done signal back to the appropriate field.  The advantage of this is that the fields wouldn’t have to know how many queues there are and the queues wouldn’t have to know how many fields there are. This information would be encapsulated in the splitter/sorter model. It would have allowed us to add or remove fields or job managers without reconfiguring the coupled models. Because of time constraints, we were unable to do this. In addition, the fact that it is difficult (impossible?) to output objects made this more time consuming. |

Lesson Learned #7

## Experimentation strategy

To experiment and test our agricultural farm model, we will adopt in a bottom down, incremental strategy. We will first start by testing the processor models (irrigation, sower and harvester) and the transducer model (market). Once the processors tested, we will test the queue and their integration with the processors as coupled models (water manager, sower manager, harvest manager). The next step will be testing the field model. Once all the sub-models have been tested, we will proceed with testing the complete, integrated, agricultural farm model.

For each of our experimentation steps, we will vary the inputs and confirm that the model behaves accordingly. We will test possible edge cases such as multiple inputs coming simultaneously or inputs arriving in the wrong order.

### Sample Tests for processors (irrigation)

|  |  |
| --- | --- |
| 12:00:00:00 | in\_water\_request (surface == 100) |
| 24:00:00:00 | in\_water\_request (surface == 200) |
| 24:00:00:00 | in\_water\_request (surface == 200) |
| 30:00:00:00 | in\_water\_request (surface == 5000) |
| 50:00:00:00 | in\_water\_request (surface == 200) |
| 70:00:00:00 | in\_water\_request (surface == 400) |

### Sample Tests for transducer (market)

|  |  |
| --- | --- |
| 10:00:00:00 | in\_bushels 16000 |
| 10:00:00:00 | in\_type 1 |
| 15:00:00:00 | in\_bushels 30000 |
| 15:00:00:00 | in\_type 2 |
| 15:00:00:00 | in\_bushels 20000 |
| 15:00:00:00 | in\_type 1 |
| 20:00:00:00 | in\_type 3 |
| 20:00:00:00 | in\_bushels 35000 |
| 25:00:00:00 | in\_type 3 |
| 25:00:00:00 | in\_type 3 |
| 25:00:00:00 | in\_bushels 45000 |

### Sample Tests for queue

|  |  |
| --- | --- |
| 00:00:00:00 | in\_water\_request (surface == 100) |
| 02:00:00:00 | in\_done |
| 10:00:00:00 | in\_water\_request (surface == 200) |
| 12:00:00:00 | in\_done |
| 13:00:00:00 | in\_done |
| 20:00:00:00 | in\_water\_request (surface == 200) |
| 30:00:00:00 | in\_water\_request (surface == 5000) |
| 30:00:00:00 | in\_water\_request (surface == 200) |
| 30:00:00:00 | in\_done |
| 30:00:00:00 | in\_done |
| 30:00:00:00 | in\_done |
| 50:00:00:00 | in\_water\_request (surface == 400) |

### Sample Tests for Job Managers (Water Manager)

|  |  |
| --- | --- |
| 12:00:00:00 | in\_water\_request (surface == 100) |
| 24:00:00:00 | in\_water\_request (surface == 200) |
| 28:00:00:00 | in\_water\_request (surface == 1000) |
| 40:00:00:00 | in\_water\_request (surface == 100) |
| 50:00:00:00 | in\_water\_request (surface == 500) |
| 50:00:00:00 | in\_water\_request (surface == 400) |
| 70:00:00:00 | in\_water\_request (surface == 100) |
| 80:00:00:00 | in\_water\_request (surface == 200) |

### Sample Tests for Fields

|  |  |
| --- | --- |
| 5:00:00:00 | in\_done |
| 10:00:00:00 | in\_done |
| 15:00:00:00 | in\_done |
| 15:00:00:00 | in\_done |
| 20:00:00:00 | in\_done |
| 25:00:00:00 | in\_stop |
| 30:00:00:00 | in\_done |

### Sample Tests for Agricultural Farm

|  |  |
| --- | --- |
| 12760:00:00:00 | in\_stop\_f2 |
| 18760:00:00:00 | in\_stop\_f1 |
| 14760:00:00:00 | in\_stop\_f2 |
| 37520:00:00:00 | in\_stop\_f3 |

Note : The times are high because we wanted to simulate yearly operation. 18760 hours is equal to one year.

# Implementation of the Agricultural Farming Model

## Implementation code

The full code for the agricultural farm model and simulation can be found in the compressed (.zip) archive included with this document delivery. Note that the results presented in this section correspond to an experimental framework that varies from the one proposed in the previous section. While developing our model, our experimentation framework kept evolving according to aspects we wanted to test. The experimental framework in the following section is the one we used to verify our model.

## Sample results and analysis

### Sample Results for processors (harvester)

|  |  |
| --- | --- |
| **Events** | **Results** |
| 00:00:00:00 in\_id 1  00:00:00:00 in\_type 1  00:00:00:00 in\_surface 111  35:02:00:00 in\_id 3  35:02:00:00 in\_type 2  35:02:00:00 in\_surface 111  35:02:00:00 in\_id 3  35:02:00:00 in\_type 2  35:02:00:00 in\_surface 111  70:00:00:00 in\_id 5  70:00:00:00 in\_type 2  70:00:00:00 in\_surface 222  100:00:00:00 in\_id 6  100:00:00:00 in\_type 3  100:00:00:00 in\_surface 333  150:00:00:00 in\_id 7  150:00:00:00 in\_type 2  150:00:00:00 in\_surface 111  170:00:00:00 in\_id 8  170:00:00:00 in\_type 3  170:00:00:00 in\_surface 11111  300:00:00:00 in\_id 8  300:00:00:00 in\_type 3  300:00:00:00 in\_surface 111  1300:00:00:00 in\_id 9  1300:00:00:00 in\_type 1  1300:00:00:00 in\_surface 333  1350:00:00:00 in\_id 10  1350:00:00:00 in\_type 3  1350:00:00:00 in\_surface 111 | 11:06:00:001 out\_id 1  11:06:00:001 out\_bushels 16650  11:06:00:001 out\_type 1  46:08:00:001 out\_id 3  46:08:00:001 out\_bushels 13320  46:08:00:001 out\_type 2  92:12:00:002 out\_id 5  92:12:00:002 out\_bushels 26640  92:12:00:002 out\_type 2  133:17:59:997 out\_id 6  133:17:59:997 out\_bushels 56610  133:17:59:997 out\_type 3  161:06:00:001 out\_id 7  161:06:00:001 out\_bushels 13320  161:06:00:001 out\_type 2  1281:05:59:912 out\_id 8  1281:05:59:912 out\_bushels 1.88887e+06  1281:05:59:912 out\_type 3  1333:17:59:997 out\_id 9  1333:17:59:997 out\_bushels 49950  1333:17:59:997 out\_type 1  1361:06:00:001 out\_id 10  1361:06:00:001 out\_bushels 18870  1361:06:00:001 out\_type 3 |

From the harvester results, we can see it works as intended. It receives requests, executes them and, when a duration that is determined by the surface expires, it outputs bushels. When it receives multiple requests simultaneously (35:02:00:00) it executes only one the first one. The results also confirm that when it receives a request while busy, it ignores the request. This occurs at 300:00:00:00, the harvester receives a request but is busy executing a previous request with a very large surface (170:00:00:00). Finally, by comparing specific requests (00:00:00:00 with 1300:00:00, 00:00:00:00 with 1350:00:00:00) we can confirm that the amount of bushels output depends on the surface and the culture type associated to the request.

### Sample Results for transducer (market)

|  |  |
| --- | --- |
| **Events** | **Results** |
| 10:00:00:00 in\_bushels 16000  10:00:00:00 in\_bushels 16000  10:00:00:00 in\_type 1  15:00:00:00 in\_bushels 30000  15:00:00:00 in\_type 2  15:00:00:00 in\_bushels 20000  15:00:00:00 in\_type 1  20:00:00:00 in\_type 3  20:00:00:00 in\_bushels 35000  25:00:00:00 in\_type 3  25:00:00:00 in\_type 3  27:00:00:00 in\_bushels 45000 | 10:00:00:000 out\_type 1  10:00:00:000 out\_bushels 16000  10:00:00:000 out\_sales 40000  15:00:00:000 out\_type 1  15:00:00:000 out\_bushels 20000  15:00:00:000 out\_sales 50000  20:00:00:000 out\_type 3  20:00:00:000 out\_bushels 35000  20:00:00:000 out\_sales 143500  27:00:00:000 out\_type 3  27:00:00:000 out\_bushels 45000  27:00:00:000 out\_sales 184500 |

From the results, we can immediately observe one issue. If the market receives two inputs at the same time, it will only process the last one received (Time 15:00:00:00). In the case of our global agricultural farm, this is a situation that shouldn’t happen. Indeed, the harvester requires time before outputting bushels to the market, it will never be able to execute multiple bushels output simultaneously. However, the issue could be fixed by integrating a queue to the market.

We can also see that the model only outputs when it has received both a type and an amount of bushels (Time 25:00:00:00 & 27:00:00:00). Furthermore, receiving the same type twice before receiving the second part of the output has no effect (Time 10:00:00:00 & Time 25:00:00:00).

### Sample Results for queue

|  |  |
| --- | --- |
| **Events** | **Results** |
| 00:00:00:00 in\_surface 111  00:00:00:00 in\_field 1  00:00:00:00 in\_type 1  12:00:00:00 in\_surface 222  12:00:00:00 in\_field 2  12:00:00:00 in\_type 3  35:00:00:00 in\_surface 111  35:00:00:00 in\_field 3  35:00:00:00 in\_type 2  50:00:00:00 in\_surface 111  50:00:00:00 in\_field 3  50:00:00:00 in\_type 3  55:00:00:00 in\_done 1  55:00:01:00 in\_done 1  55:00:02:00 in\_done 1  60:00:00:00 in\_done 1  60:00:01:00 in\_done 1  70:00:00:00 in\_surface 333  70:00:00:00 in\_field 1  70:00:00:00 in\_type 3  75:00:00:00 in\_surface 111  75:00:00:00 in\_field 2  75:00:00:00 in\_type 2  100:00:00:00 in\_done 1 | 00:00:00:000 out\_id 1  00:00:00:000 out\_surface 111  00:00:00:000 out\_type 1  55:00:00:000 out\_field\_1 1  55:00:00:000 out\_id 2  55:00:00:000 out\_surface 222  55:00:00:000 out\_type 3  55:00:01:000 out\_field\_2 2  55:00:01:000 out\_id 3  55:00:01:000 out\_surface 111  55:00:01:000 out\_type 2  55:00:02:000 out\_field\_3 3  55:00:02:000 out\_id 4  55:00:02:000 out\_surface 111  55:00:02:000 out\_type 3  60:00:00:000 out\_field\_3 4  70:00:00:000 out\_id 5  70:00:00:000 out\_surface 333  70:00:00:000 out\_type 3  100:00:00:000 out\_field\_1 5  100:00:00:000 out\_id 6  100:00:00:000 out\_surface 111  100:00:00:000 out\_type 2 |

From these results, we can see that the queue works properly. It receives a first request, immediately outputs it then holds further requests in a list until it receives its first done signal (Time 55:00:00:00). At this point, it outputs a done signal back to a field and outputs the next request to the processor. At Time 60:00:00:01, the queue receives an extra done signal but appropriately ignores it. It then restarts the queuing process with new requests.

### Sample Results for Job Managers (Harvest Manager)

|  |  |
| --- | --- |
| **Events** | **Results** |
| 00:00:00:00 in\_surface 111  00:00:00:00 in\_field 1  00:00:00:00 in\_type 1  12:00:00:00 in\_surface 222  12:00:00:00 in\_field 2  12:00:00:00 in\_type 3  12:00:00:00 in\_surface 333  12:00:00:00 in\_field 1  12:00:00:00 in\_type 1  35:00:00:00 in\_surface 111  35:00:00:00 in\_field 3  35:00:00:00 in\_type 2  50:00:00:00 in\_surface 111  50:00:00:00 in\_field 3  50:00:00:00 in\_type 3  70:00:00:00 in\_surface 333  70:00:00:00 in\_field 1  70:00:00:00 in\_type 3  75:00:00:00 in\_surface 111  75:00:00:00 in\_field 2  75:00:00:00 in\_type 2  150:00:00:00 in\_surface 222  150:00:00:00 in\_field 2  150:00:00:00 in\_type 2 | 11:06:00:001 out\_field\_1 1  34:12:00:002 out\_field\_2 2  67:29:59:999 out\_field\_1 3  78:36:00:000 out\_field\_3 4  89:42:00:001 out\_field\_3 5  122:59:59:998 out\_field\_1 6  134:05:59:999 out\_field\_2 7  172:12:00:002 out\_field\_2 8 |

These results are straightforward. The sower manager receives 8 requests and eventually outputs 8 requests. We can see that the outputs are deferred by the time the sower requires to process them. Indeed, the sower’s speed is 10 acres per hour so a field with a surface of 111 acres requires 11.1 hours to process (Time 11:06:00:00). We can also see that concurrent requests are all held in the queue and eventually processed. Finally, we can observe what happens when a request is delayed because the processor is busy. For example, the request received at 70:00:00:00 requires 33.3 hours to complete, it should be done at around 103 hours. However, the results show that it is complete only at 122:59:59:99 hours. Time is wasted by the model.

### Sample Results for Fields

|  |  |
| --- | --- |
| **Events** | **Results** |
| 6:00:00:00 in\_done 1  30:00:00:00 in\_done 1  90:00:00:00 in\_done 1  90:00:00:00 in\_done 1  150:00:00:00 in\_done 1  180:00:00:00 in\_done 1  210:00:00:00 in\_done 1  240:00:00:00 in\_done 1  270:00:00:00 in\_done 1  300:00:00:00 in\_done 1  330:00:00:00 in\_done 1  360:00:00:00 in\_done 1  390:00:00:00 in\_done 1  420:00:00:00 in\_done 1  450:00:00:00 in\_done 1  480:00:00:00 in\_done 1  510:00:00:00 in\_done 1  520:00:00:00 in\_stop 1 | 00:00:00:000 out\_sower\_field 1  00:00:00:000 out\_sower\_surface 100  00:00:00:000 out\_sower\_type 1  06:00:00:000 out\_water\_field 1  06:00:00:000 out\_water\_surface 100  06:00:00:000 out\_water\_type 2  78:00:00:000 out\_water\_field 1  78:00:00:000 out\_water\_surface 100  78:00:00:000 out\_water\_type 2  138:00:00:000 out\_water\_field 1  138:00:00:000 out\_water\_surface 100  138:00:00:000 out\_water\_type 2  198:00:00:000 out\_harvest\_field 1  198:00:00:000 out\_harvest\_surface 100  198:00:00:000 out\_harvest\_type 2  210:00:00:000 out\_sower\_field 1  210:00:00:000 out\_sower\_surface 100  210:00:00:000 out\_sower\_type 2  240:00:00:000 out\_water\_field 1  240:00:00:000 out\_water\_surface 100  240:00:00:000 out\_water\_type 3  342:00:00:000 out\_water\_field 1  342:00:00:000 out\_water\_surface 100  342:00:00:000 out\_water\_type 3  432:00:00:000 out\_water\_field 1  432:00:00:000 out\_water\_surface 100  432:00:00:000 out\_water\_type 3 |

As we can see from the results that the field behaves in the intended way. The first request it outputs, at 00:00:00:00 is a request for seeds, it receives a done signal at 06:00:00:00 which triggers a request for water. When there is a long delay between inputs, we notice that the evaporation process works well. For example, between 30:00:00:00 and 90:00:00:00 a water request is fired at 78:00:00:00. This corresponds to the 48 hours evaporation period for culture type 2. At 198:00:00:00, the growth cycle is complete (The time is wrong in these results, see note below.) so the field requests harvest and once harvest is done, it requests seeds. The cycle then begins again.

Additionally, at 90:00:00:00, we can also see that simultaneous requests do not cause any problems for the fields.

|  |
| --- |
| From the field model results above, we can see that the time for the first harvest (198:00:00:00) is nowhere near the expected growth time of the model (6 days or 144 hours). This is due to the way we calculated the time left to achieve full growth. In our model, every time a field receives water and is in a dry state, it means that it has evaporated. Therefore, we subtract the evaporation time from the time left for full growth. The consequence of this is that when the model receives water at arbitrary times such as when we use an event file, the growth process is not respected. For example, if the growth period is 4 days, the evaporation is 2 days, and the event file sends two in\_done inputs within 4 hours, the field, will be considered grown. This does not happen in the full Farm because the field waits for the evaporation time to be elapsed before asking for water. The in\_done input from the water manager will not arrive unless evaporation time has elapsed.  The ideal way to evaluate growth and evaporation would have been to integrate the simulator’s elapsed time in the model. However, due to time constraints and our unfamiliarity with the CD++ Toolkit, we chose to implement our simplified method of calculating growth and evaporation times. |

Lesson Learned #8

### Sample Results for Agricultural Farm

|  |  |
| --- | --- |
| **Events** | **Results** |
| 5000:00:00:00 in\_stop\_f2 1  10000:00:00:00 in\_stop\_f1 1  10000:00:00:00 in\_stop\_f2 1  15000:00:00:00 in\_stop\_f3 1 | 1838:00:00:000 out\_type 2  1838:00:00:000 out\_bushels 12000  1838:00:00:000 out\_sales 38400  1948:00:00:000 out\_type 3  1948:00:00:000 out\_bushels 34000  1948:00:00:000 out\_sales 139400  2380:00:00:000 out\_type 1  2380:00:00:000 out\_bushels 45000  2380:00:00:000 out\_sales 112500  3684:00:00:000 out\_type 3  3684:00:00:000 out\_bushels 17000  3684:00:00:000 out\_sales 69700  3768:00:00:000 out\_type 1  3768:00:00:000 out\_bushels 30000  3768:00:00:000 out\_sales 75000  5042:00:00:000 out\_type 2  5042:00:00:000 out\_bushels 36000  5042:00:00:000 out\_sales 115200  5230:00:00:000 out\_type 1  5230:00:00:000 out\_bushels 15000  5230:00:00:000 out\_sales 37500  6722:00:00:000 out\_type 2  6722:00:00:000 out\_bushels 12000  6722:00:00:000 out\_sales 38400  7276:00:00:000 out\_type 3  7276:00:00:000 out\_bushels 51000  7276:00:00:000 out\_sales 209100  8476:00:00:000 out\_type 3  8476:00:00:000 out\_bushels 17000  8476:00:00:000 out\_sales 69700  9380:00:00:000 out\_type 1  9380:00:00:000 out\_bushels 45000  9380:00:00:000 out\_sales 112500  11682:00:00:000 out\_type 2  11682:00:00:000 out\_bushels 36000  11682:00:00:000 out\_sales 115200  13842:00:00:000 out\_type 3  13842:00:00:000 out\_bushels 51000  13842:00:00:000 out\_sales 209100 |

We can see that the farm behaves as expected. It outputs bushels, sales and types at intervals that roughly follow the expected times based on the various factors that impact it. For example, the first output can be easily traced. According to the model parameters, the field’s surface is 100, the growth rate is 40 days and it evaporates every day. At this state, it requires 960 hours to grow, during which time it will output 40 water requests. Each water request requires 10 hours to execute (100 acres divided by the speed of the irrigator, 10 acres/hour). The sowing and harvest times are each 20 hours. This brings the total time required to 1400 hours, the missing 438 hours can likely be explained by time wasted waiting for processors to be ready. The conclusion is that there is a lot of room for optimization.

In this case, it is obvious to us, having developed the model, that the irrigator is the bottleneck. This can easily be confirmed by running another simulation after doubling the irrigator’s speed in the model parameters (see farm.ma).

|  |  |
| --- | --- |
| **Events** | **Results** |
| 5000:00:00:00 in\_stop\_f2 1  10000:00:00:00 in\_stop\_f1 1  10000:00:00:00 in\_stop\_f2 1  15000:00:00:00 in\_stop\_f3 1 | 1296:00:00:000 out\_type 2  1296:00:00:000 out\_bushels 12000  1296:00:00:000 out\_sales 38400  1772:00:00:000 out\_type 1  1772:00:00:000 out\_bushels 45000  1772:00:00:000 out\_sales 112500  1812:00:00:000 out\_type 3  1812:00:00:000 out\_bushels 34000  1812:00:00:000 out\_sales 139400  2970:00:00:000 out\_type 3  2970:00:00:000 out\_bushels 17000  2970:00:00:000 out\_sales 69700  ... |

## Model corrections following experimentation

Many small changes were brought to the models while we were developing and testing. We did not document each of them while we were developing. However, we can still discuss some of the bigger, more memorable changes.

### Field would run past the stop time

The stopped state was not being set when the field model received a stop signal, the model was only passivating. This resulted in the model answering to inputs arriving after the stop signal. It also caused the field model to run indefinitely. Fortunately, the “kill simulation” button allowed us to stop simulating.

### Problem with days to hours conversion

Testing allowed us to identify a problem with our conversion from days to hours. Our field model is configured in days (i.e growth time and evaporation time are in days). These values had to be converted to hours for the simulator. There was an issue with our conversion and we could easily see the incorrect time values in the results files.

### Problem with queue

While testing with the queue, we discovered that the queue crashed if it received more done signal than it had requests. This was not apparent when testing with carefully crafted test events but, by testing edge cases, we could see that we were missing a validation on the queue size before processing a done signal.