

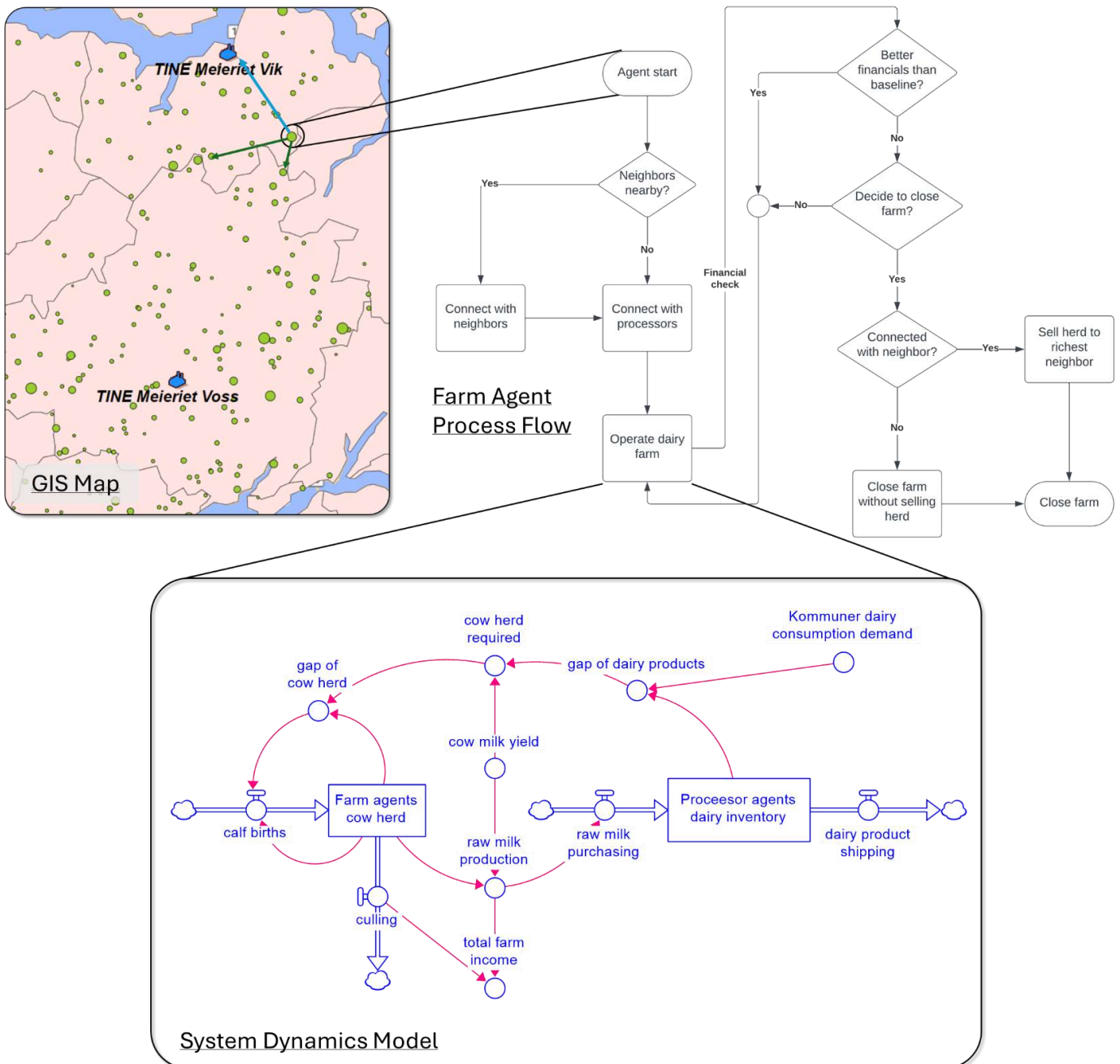
Supplemental - Multi-Method Approach of System Dynamics And Agent-Based Modeling: A Practical Case Study of the Norwegian Dairy Supply Chain

Takuma Ono, Master of Philosophy in System Dynamics, University of Bergen

Hugo Herrera, Postdoctoral Fellow, System Dynamics Group, University of Bergen

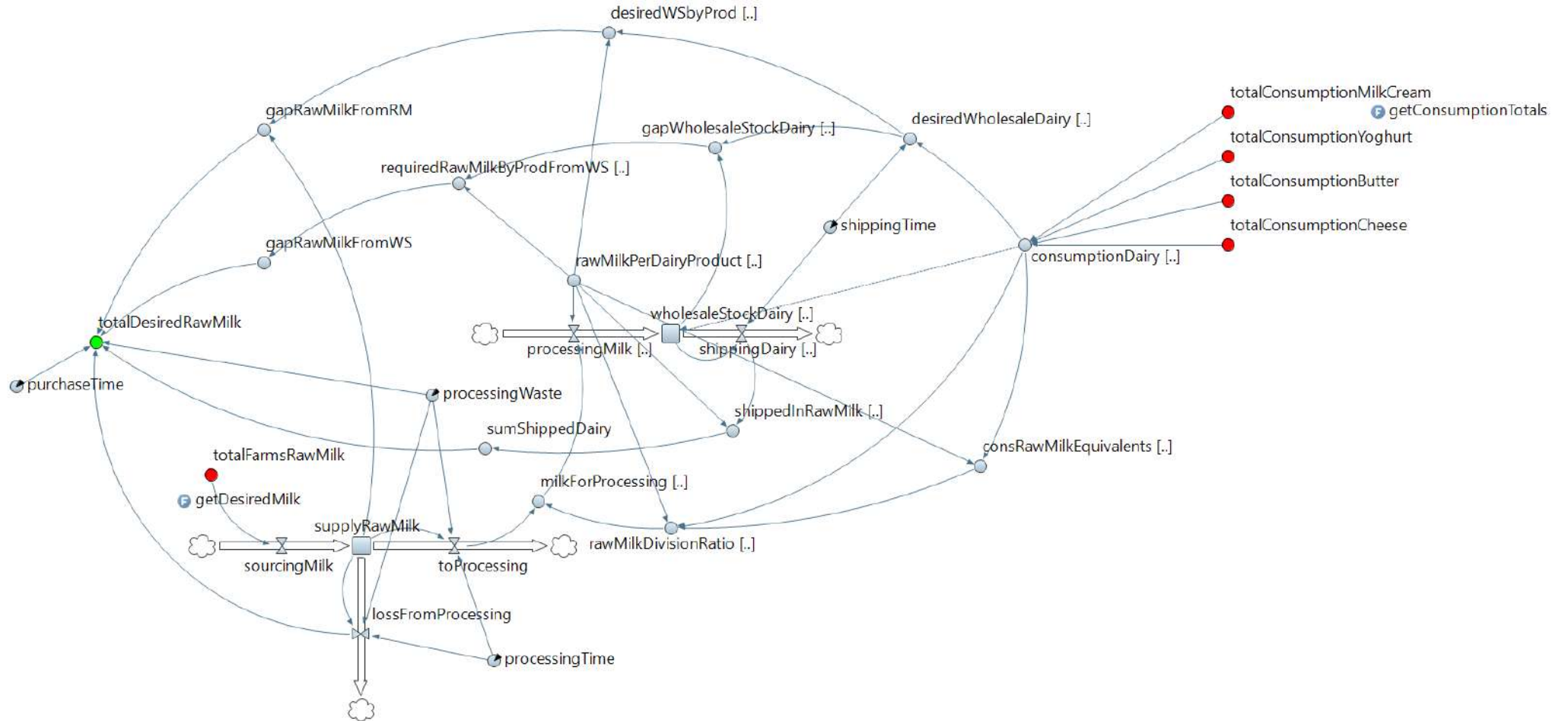
2024 International System Dynamis Conference, August 4-8, Bergen, Norway

Appendix A: Expanded Model Conceptualization Diagram

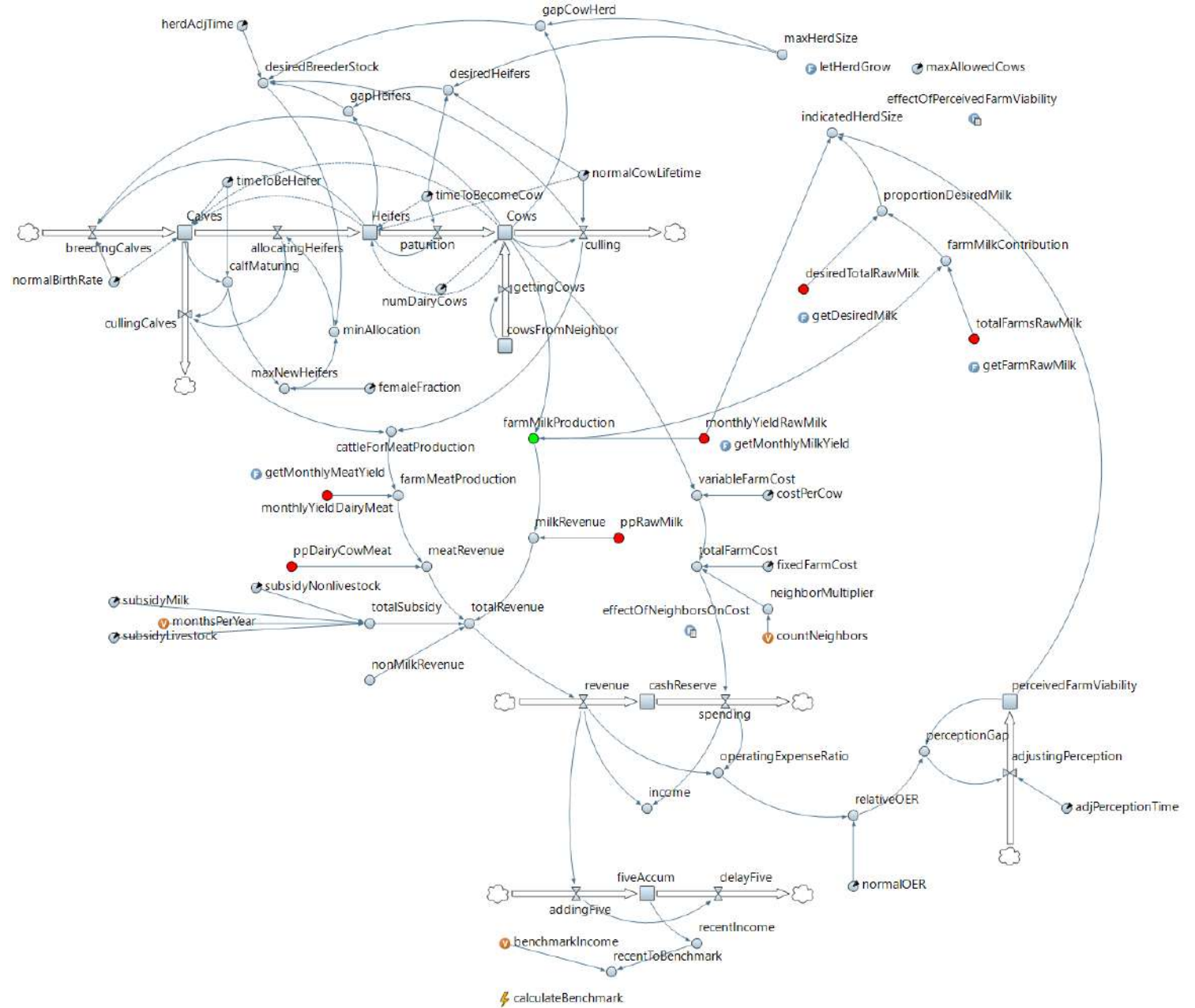


Appendix B: AnyLogic Stock-and-Flow Diagrams

Stock-and-Flow Diagram of Dairy Processors (5 Processor Agents)



Stock-and-Flow Diagram of Dairy Farms (1,527 Farm Agents)



Appendix C: AnyLogic System Dynamics Model

Component Description

The System Dynamic component itself is comprised of fairly standard structures. Annual dairy product consumption from kommuner calculated exogenously from per capita consumption of each dairy product, and these production targets will be relayed to their corresponding dairy processors as individually arrayed goals in a goal-gap mechanism.

Dairy Processor Stock-and-Flow Structure

The consumption demands of dairy products received by the five Vestland county dairy processors will depend on the nearby kommuner which identified them as the closest partner processor to form a “handshake” with. The product demands from the customer kommuner are summed into an arrayed variable as the goal of a goal-gap mechanism and identified as *consumptionDairy* [...] (Figure 21). There are two purposes for the *consumptionDairy* variable, one of which is the production target goal—renamed to *desiredWholesaleDairy* [...]—as discussed previously, the other being the basis for the *rawMilkDivisionRaio* variable.

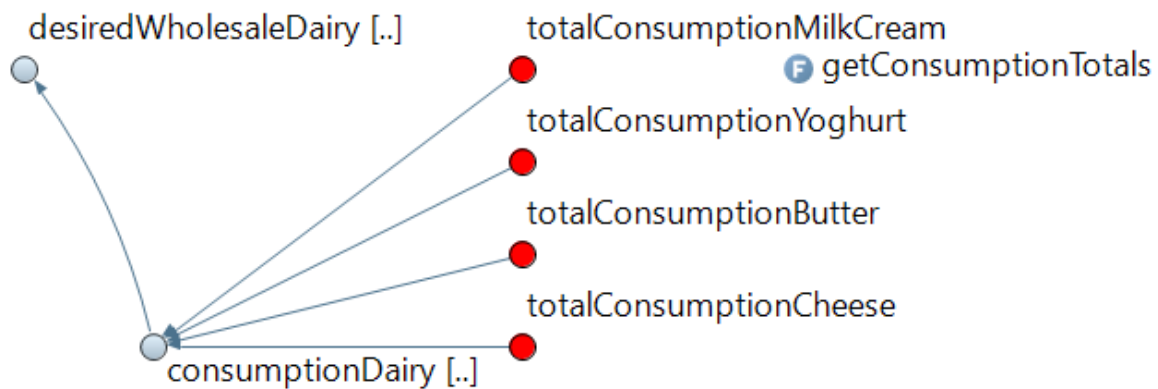


Figure 21: Dairy product demands from partner kommuner summed into an arrayed variable in a dairy processor agent

The *rawMilkDivisionRatio* determines how the total raw milk received from all of its partner dairy farms should be apportioned among the four dairy products (Figure 21). As such, *rawMilkDivisionRatio* is an arrayed variable which if summed to one value it must equal 1.0. This division ratio is calculated by first translating the total demands of each dairy product to amount of raw milk equivalent needed to fulfill them. This conversion factor is provided by the parameter *rawMilkPerDairyProduct*, as adapted from the SYNAGRI NetLogo model (synagri.no, Accessed 2024).

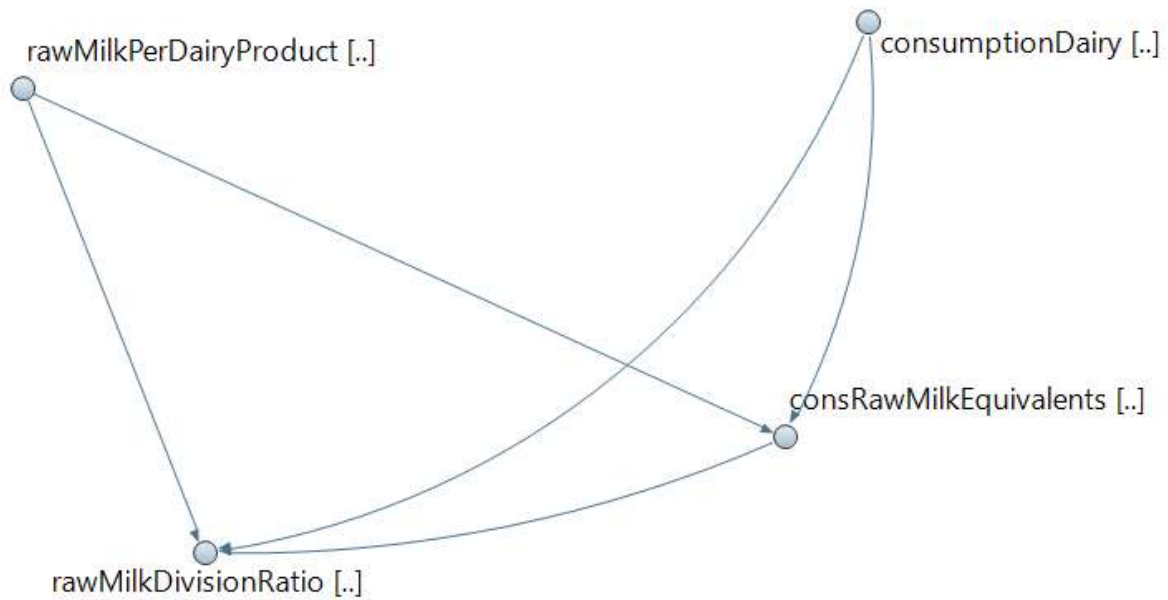


Figure 22: Calculating of rawMilkDivisionRatio, which determines the split of raw milk received.

Once the amount of raw milk needed for each dairy product is calculated, then milk requirement for each product is then divided by the total raw milk requirement to determine the “split” for each product. To demonstrate, the *rawMilkDivisionRatio* for the processor TINE Meieriet Bergen calculated in Table 5.

Table 5: Calculation of rawMilkDivisionRatio for dairy processor TINE Meieriet Bergen

Product	Kommuner Dairy Product Demand (tonnes / yr)	Raw milk per dairy product (kg / kg)	Raw milk required per product (tonnes / yr)	Raw milk division ratio (dimensionless)
Milk	9,291	1	9,291	0.25
Yoghurt	711	1.13	804	0.02
Butter	1,274	20	25,490	0.69
Cheese	230	6.91	1,591	0.04
		TOTAL	37,175	

As noted previously, the kommuner consumption demand is also used as the desired dairy product goal. The gap in the goal-gap structure is calculated by subtracting the arrayed wholesale stock value from the desired consumption value (Figure 23).

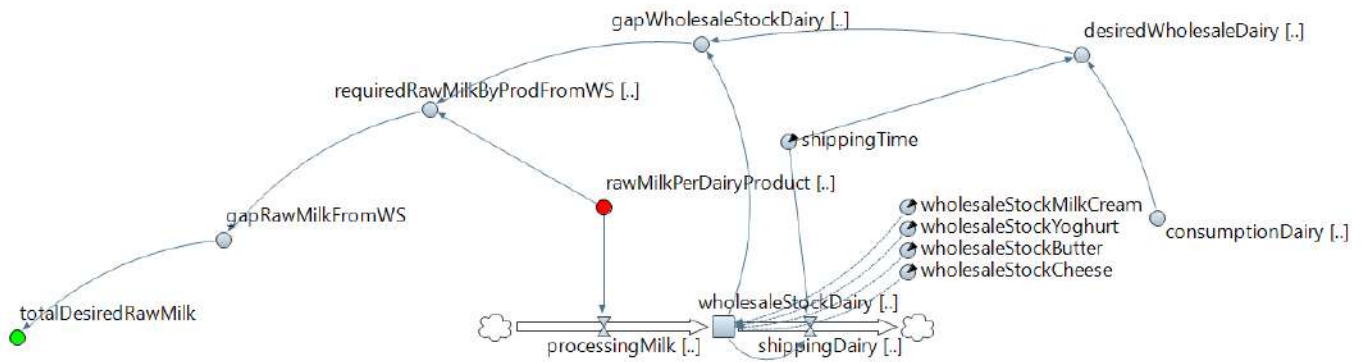


Figure 23: Goal-gap mechanism of dairy products and wholesale stock at dairy processor

The calculated gap is then multiplied by the aforementioned *rawMilkPerDairyProduct* to determine the raw milk equivalent requirement for each dairy product (*requiredRawMilkByProdFromWS*). This variable is still arrayed by product type, so it is summed into a single value and added to the *totalDesiredRawMilk* variable, which is passed onto the partner dairy farm agents.

The *totalDesiredRawMilk* variable must also correct for three other variables: stock of raw milk waiting to be processed, raw milk lost from processing, and finished dairy product shipped out from wholesale stock.

The wholesale stock of dairy products is preceded by purchased raw milk stock inventory, which must also be corrected with a similar goal-gap mechanism (Figure 24). The *desiredWholesaleDairy* is similarly multiplied by *rawMilkPerDairyProduct* then summed by array to determine the total raw milk required. Finally, a gap is calculated by subtracting by the current stock of *supplyRawMilk*. This value is then added to the *totalDesiredRawMilk* variable.

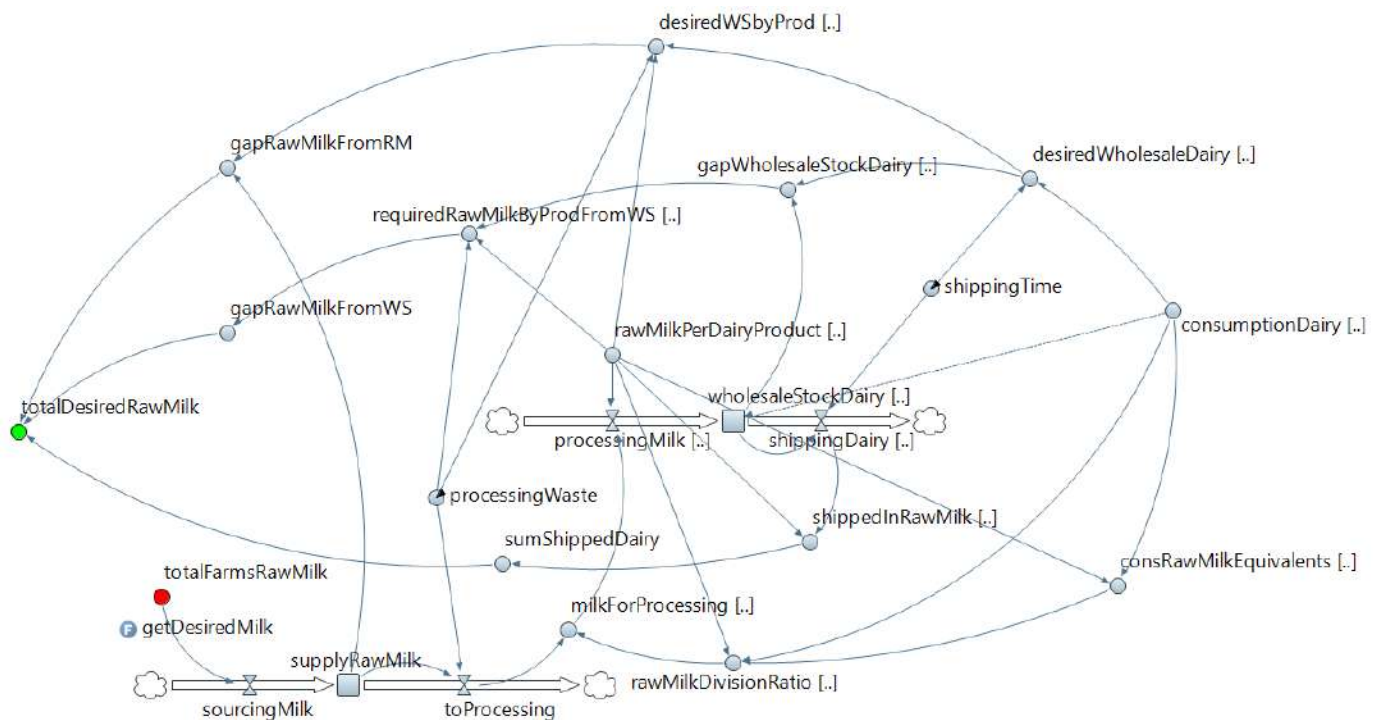


Figure 24: Raw milk inventory stock gap calculation also using the kommuner dairy consumption demand variable

A small amount of processing loss occurs when converting the raw milk into dairy products. This outflow is compensator for by simply adding the loss back to the *totalDesiredRawMilk* so that dairy processors know to purchase extra milk to make up for the processing loss (Figure 25).

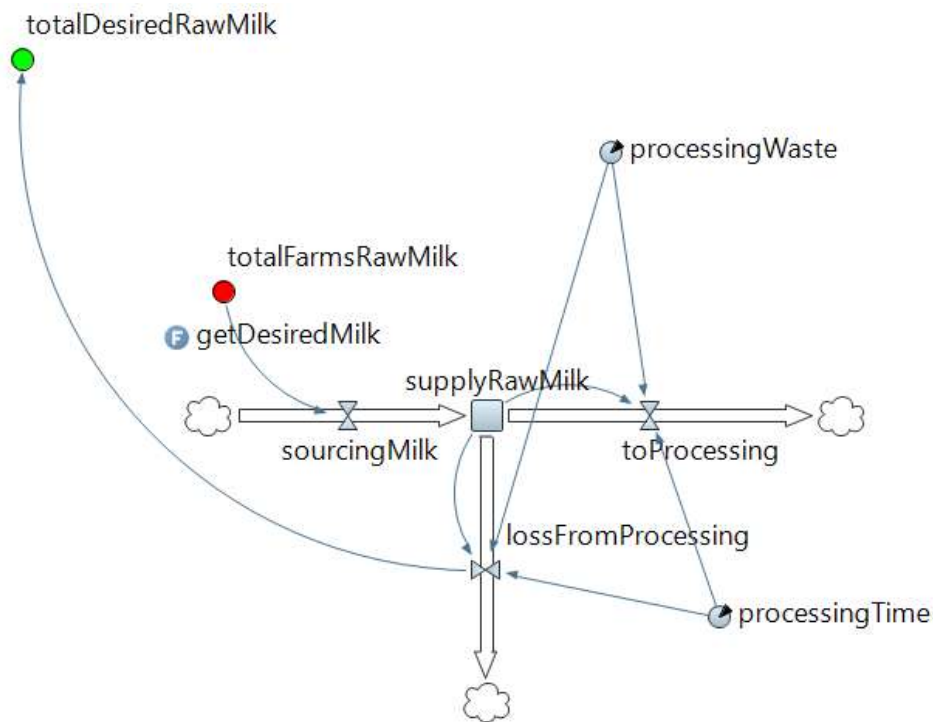


Figure 25: Raw milk processing loss outflow added back to *totalDesiredRawMilk* to make up the loss

Finally, another outflow that should be compensated for is the final dairy products shipped out from the wholesale stock. This outflow is treated similarly to the processing loss outflow, but the variable must first be converted into raw milk equivalent by again multiplying by *rawMilkPerDairyProduct* factor, named *shippedInRawMilk* [..]. Then, because this outflow is arrayed by dairy product type, it must then be summed into a single variable (*sumShippedDairy*) before adding to *totalDesiredRawMilk* (Figure 26).

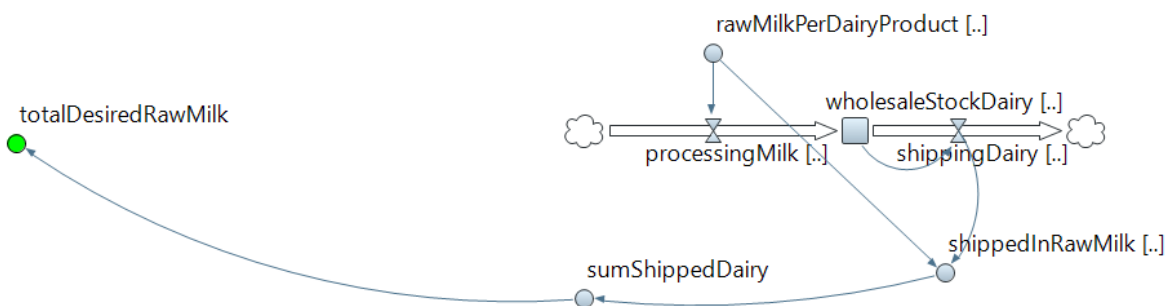


Figure 26: Outflow *shippingDairy*, arrayed by dairy product is converted into raw milk equivalent, then summed into single value before adding to *totalDesiredRawMilk*

Dairy Farm Stock-and-Flow Structure

As seen in *Appendix B: AnyLogic Stock-and-Flow Diagrams*, because the dairy farm structure is relatively large, it will be discussed using these four parts called sectors.

Similar to kommuner consumption demand being the target goal for the dairy processor structure, the *totalDesiredRawMilk* variable in turn becomes the target production goal for the partner dairy farms (Figure 27).

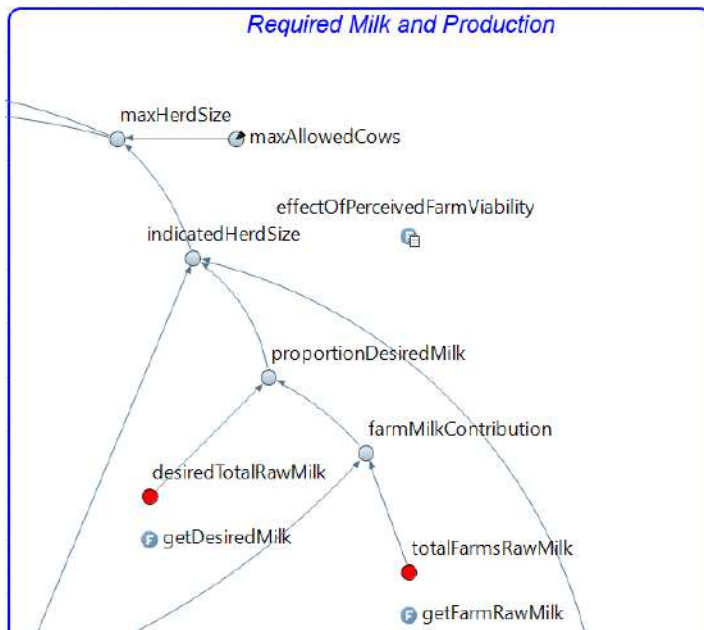


Figure 27: Dairy farm sector for receiving production demand and selling produced milk

The total desired raw milk production from their partner processor is received, but the total required milk is not helpful for a single farm agent on how much it should produce. Therefore, a farm's production volume is dividing by the total production volume of all farms connected to the same dairy processor. This calculates the portion of the total milk required of the individual farm and scales it proportionally to individual dairy farms (*proportionDesiredMilk*).

The proportional milk target is divided by the expected milk yield per cow over the same time period, which determines the cow herd needed to fulfill the production needs of the partner dairy processor (*indicatedHerdSize*). There is also an effect from the perceived financial viability (to be discussed subsequently) of the farm which may increase or decrease the indicated herd size. The model has a ceiling cow herd size of 500 (*maxAllowedHerd*), so no matter how much the processor milk demand increases, the indicated herd size will not be greater than 500.

Once the *indicatedHerdSize* is determined (and capped at 500 if needed), this variable is used to determine the cow herd stocks. The herd population is split into three: *Cows*, *Heifers*, and *Calves* (Figure 28). The *Cows* stock is the only milk-producing population, so the indicated herd size relate to this stock value. Heifers are cows that have not yet become pregnant, and is not productive yet. Calves are produced from the Heifer and Cow population at the farm. The only exogenous population input is transferred from nearby closing farms (to be discussed subsequently).

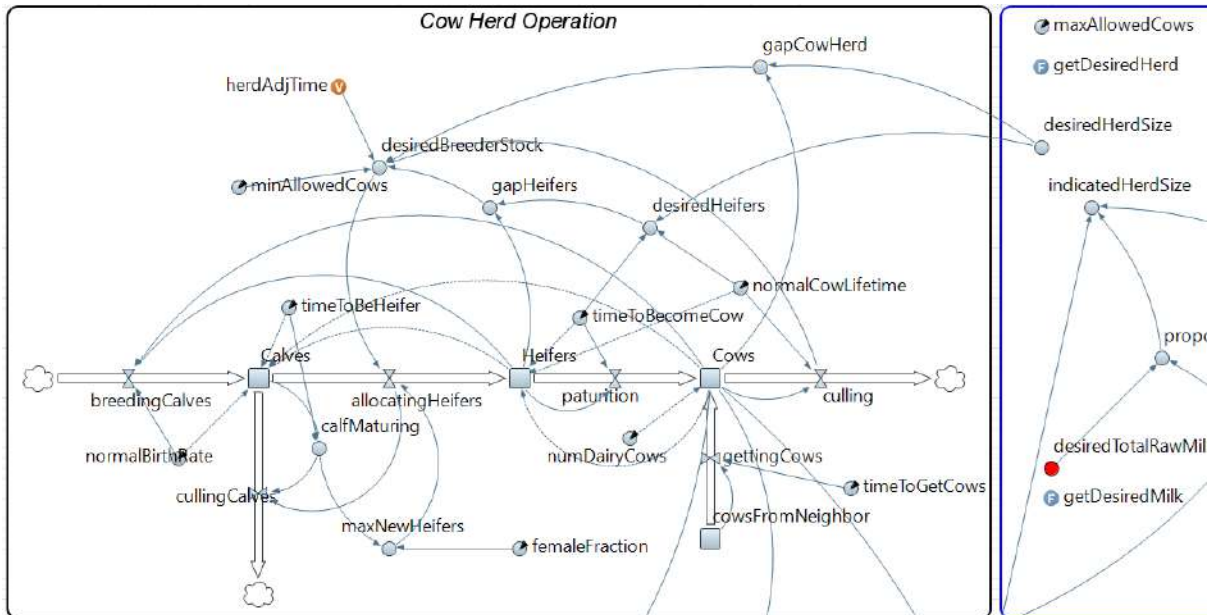


Figure 28: Cow herd operation sector of farm agent

Similar to the kommuner consumption demand gap, the gap is calculated between the indicated herd size and the current value in the *Cows* stock. The calculated gap (*gapCowHerd*) is added to the variable *desiredBreederStock*, which determines how many calves should be recruited as Heifers instead of culled. The formula for *desiredHeifers* is documented in *Appendix D: System Dynamics Model Equation Documentation* which is derived from an equilibrium state, where inflow and outflows are equal, which allows for algebraic calculation of the desired heifer value. The current Heifers value is subtracted from *desiredHeifers* to calculate the *gapHeifers*, which is also added to the *desiredBreederStock*. Because the desired number of heifer recruits cannot be less than 0 (*minAllowedCows*), the model puts a hard floor on the allowed minimum value of the variable at 0 in the equation for *desiredBreederStock*.

The *desiredBreederStock* variable must be compared against the number of female calves available to the farm. It is assumed that every Heifer and Cows produce 1 calf per year, which inflows to the *Calves* stock. Then, the calves spend two years (24 months) before they are ready to potentially become heifers. Since only females can be recruited as heifers, half of the calf population is by default diverted to *cullingCalves* outflow. The other half is the maximum allowed number of calves to be heifers (*maxNewHeifers*). Between *desiredBreederStock* and *maxNewHeifers*, the lesser of the two is recruited as heifers, and the remainder diverted to *cullingCalves*. The recruited Heifer stock population spends 1 year (12 months) before they inflow to *Cows* stock.

There are three avenues of income for farms: raw milk from *Cows*, meat from culled cattle, and subsidy (Figure 29). The number of *Cows* is multiplied by the *monthlyYieldRawMilk* to determine the total raw milk production for the period. Then, the milk production is multiplied by the producer price per tonne of raw milk (*ppRawMilk*) to determine the revenue from milk in NOK (Norwegian Krone).

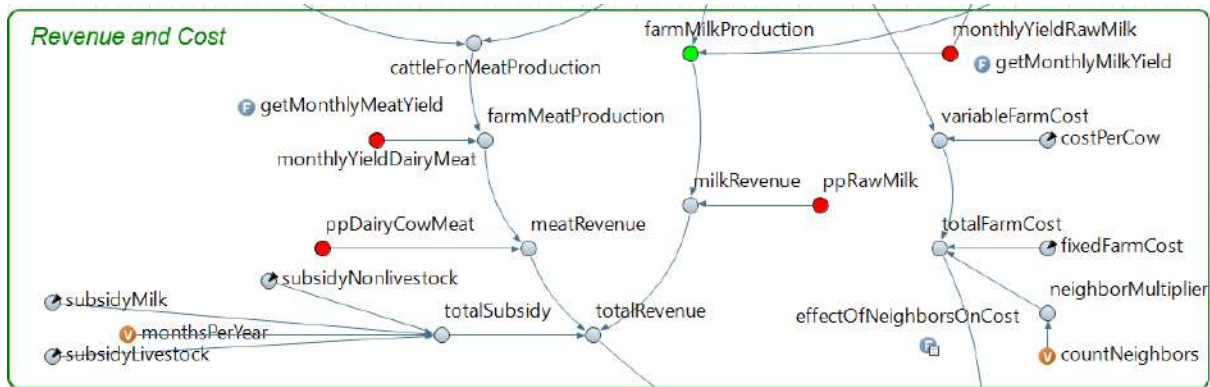


Figure 29: Revenue and cost sector of dairy farm agent

A farm may sell a portion of the population as meat through two outflows – culling calves and culling cows. A cow’s average productive time is assumed to be 4 years, and are culled after that period. Additionally, calves not recruited to be heifers are similarly culled, and the combined production of meat is similarly multiplied by the corresponding producer price to obtain meat revenue in NOK.

According to the SYNAGRI model data, dairy farms receive government subsidy in three ways: from producing milk, managing livestock, and general farm subsidy (non-livestock). The total subsidy is added to the revenues from milk and meat to determine the total revenue for the period.

The cost side comes from two simple factors: variable cost and fixed cost. Variable cost is per cow, which is 27,000 NOK per cow per year, according to Asheim (2014). The fixed cost comes from all other costs, such as building maintenance and business overhead, and is a flat 225,000 NOK per farm (Asheim, 2014). The sum of the two factors become the *totalFarmCost* variable.

In the Financial Evaluation sector, the total revenue acts as the inflow into a *cashReserve* stock, while the total cost plus *personalSpending* named *spending*, acts as the outflow from the stock (Figure 30). A business performance indicator called *operatingExpenseRatio* is calculated by dividing the *spending* by the *revenue*. Operating Expense Ratio, or OER, gives indication of the business’s operating efficiency (zebra bi, 2023). This fraction is then normalized by a ratio of 0.70, considered to be a good OER for dairy farms (Nolan, 2023). This *relativeOER* fraction is 1.0 if the farm is performing normally. It will be greater than 1.0 if performing well, and less than 1.0 if underperforming. The *relativeOER* is used as a simple goal-gap mechanism, where the farm holder’s perception of the farm’s viability is constantly being updated with an adjustment time delay of 5 years. The *perceivedFarmViability* stock value is used as an effect to the aforementioned *indicatedHerdSize*.

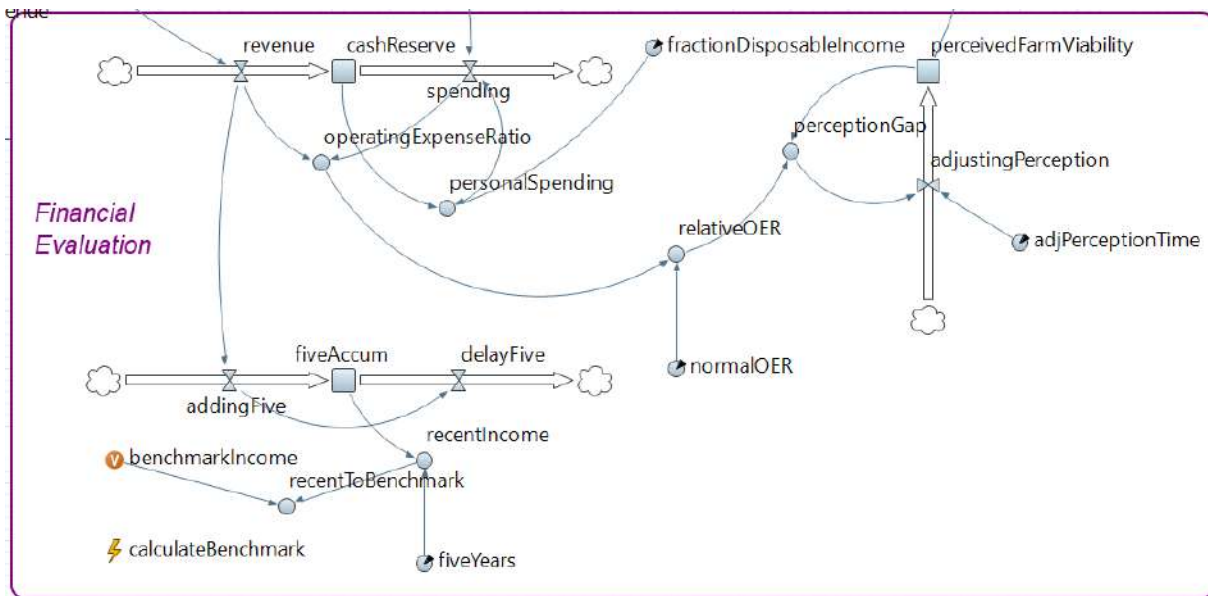


Figure 30: Financial evaluation sector of the dairy farm agent model

Also, net value, or *income*, is calculated by subtracting cost from revenue. The SYNAGRI NetLogo model uses accumulation of *income* over time to determine if a farm should continue operating or close. Another inflow, *addingFive* accumulates the income in a new stock, *fiveAccum*. At year 5 of the simulation, the total accumulation is divided by 5 to obtain the variable constant *benchmarkIncome*, which is the average annual income in the first 5 years of simulation.

At year 5, the *delayFive* outflow is then activated. This is because the value of *fiveAccum* is used to annually calculate the 5 year average as a new variable *recentIncome*. In the 6th year, the income from the 1st year is “discarded”, or outflow from *fiveAccum* stock; at 7th year, income from 2nd year flows out (Figure 31). This is in effect a “conveyor” stock with a 5 year incubation period, which keeps the *fiveAccum* stock to represent a running 5-year accumulation which can be used to calculate the annual average *recentIncome*.

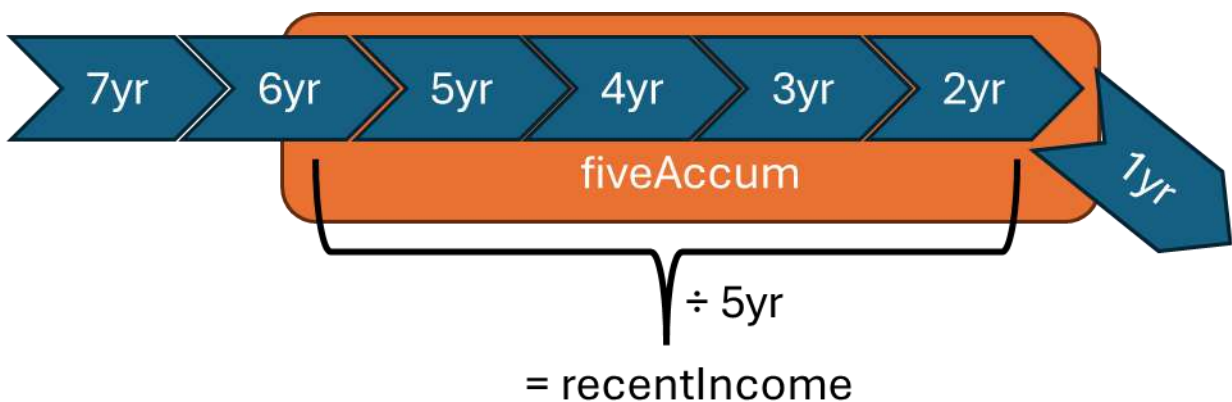


Figure 31: “Conveyor” system which continually calculates the 5-year running annual average

The decision-making process for farm closure is adopted from the SYNAGRI model. Each farm evaluates its own financial performance annually by comparing their own ratio of *recentIncome* to *benchmarkIncome*. Per the SYNAGRI model assumption, if the fraction is less than 0.85, then

the farm agent undergoes a stochastic decision making process on whether to keep operating or to close the farm and sell their cow herd to one of its neighbors.

When a farm has a ratio of less than 0.85 and has not received cows from a neighbor in the last year, then the farm agent simply has a 25% chance of closing. If the farm does decide to close, then it transfers its cow herd to a neighbor with the best financial viability, if any. If the closing farm does not have any neighbors, then the farm agent and its cow herd is removed from the simulation. If the farm “clears” the 25% chance of closing, it will keep operating until the next annual financial self-evaluation.

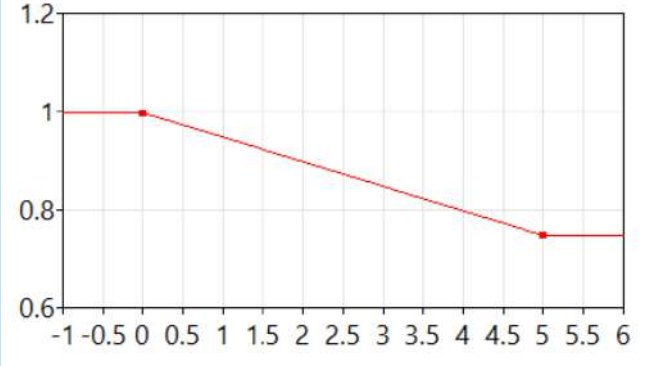
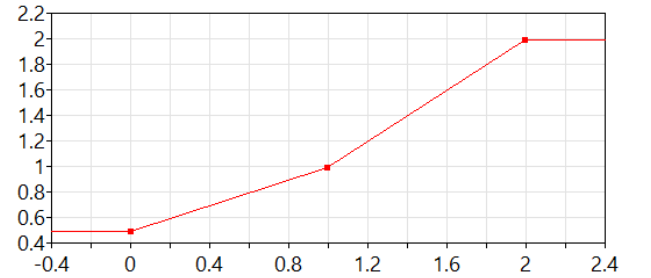
One modification made to the SYNAGRI algorithm is that if a farm has received cows from a neighbor who is closing, then the receiving farm should wait another year to re-evaluate its financial viability, even if its recent-to-benchmark income is currently less than 0.85, so that it has a chance to see whether the additional cows help with its finances.

Appendix D: System Dynamics Model Equation Documentation

Dairy Farm Agent

Farm Variable [unit]	Equation	Documentation
Calves(t) [cows]	INTEG (+ breedingCalves - allocatingHeifers - cullingCalves, (Cows + Heifers) * normalBirthRate * timeToBeHeifer)	Stock: The initial value is determined analytically. Because in an equilibrium inflows must be equal to outflows. calves_born = calves_allocated_to_heifers + calves_to_culling (Cow_Herd + Heifers) * normal_birth_rate = calves_allocated_to_heifers + calf_maturation_rate - calves_allocated_to_heifers (Cow_Herd + Heifers) * normal_birth_rate = calf_maturation_rate (Cow_Herd + Heifers) * normal_birth_rate = Calves / time_to_be_heifer Calves = (Cow_Herd + Heifers) * normal_birth_rate * time_to_be_heifer
cashReserve (t) [NOK]	INTEG (+ revenue - spending, 900000)	Stock: This is the accumulation of revenue minus spending over the simulation.
Cows (t) [cows]	INTEG (+ paturation + gettingCows - culling, numDairyCows)	Stock: Cows are the only cattle stock that is actively producing milk on a farm. The initial value is given by SYNAGRI data.
cowsFromNeighbor [cows]	INTEG (- gettingCows, 0)	Stock: This structure is activated only when another farm agent who is closing sells its herd.
fiveAccum [NOK]	INTEG (+ addingFive - delayFive, 0)	Stock: This structure accumulates income for five years, in which the 5-year benchmark annual income is calculated, then continues to be used to calculate the running 5 year annual average.
Heifers(t) [cows]	INTEG (+ allocatingHeifers - paturation, Cows * timeToBecomeCow / normalCowLifetime)	Stock: Heifers are female cows allocated for milk production but have not given birth yet. The number of initial heifers is calculated analytically, since at equilibrium all flows should be equal. Therefore the flow increasing herd should equal culling. Increasing herd = culling Heifers / time_to_become_cow = Cows / normal_cow_lifetime and since at equilibrium Cows should equal the desired cow herd size: Heifers = Cows * time_to_become_cow/normal_cow_lifetime
Perceived_Farm_Viability(t) [dmnl]	INTEG (+/- adjustingPerception, 1)	Stock: This is a soft value of how the farm holder feels about the viability of the farm, with 1 being neutral. > 1 is good, < 1 is bad.
addingFive [NOK/month]	revenue	Flow: This is a duplicate of the flow <i>revenue</i> , but will be used for the farm closure decision process
adjPerceptionTime [months]	60	Parameter: Default is 60 months (5 years), which is cited by dairy economics studies as a medium-term outlook of dairy farms (Réquillart et al., 2008).

Farm Variable [unit]	Equation	Documentation
adjustingPerception [1/months]	$\text{perceptionGap} / \text{adjPerceptionTime}$	Flow: This bidirectional flow adjusts the farmers' perception of farm viability up/down according to the relative expense ratio.
allocatingHeifers [cows/Months]	$\min(\text{desiredBreederStock}, \text{maxNewHeifers})$	Flow: This flow takes the lesser of the two - the maximum available female calves ready to become heifer, or the desired number of new heifers
breedingCalves [cows/Months]	$(\text{Heifers} + \text{Cows}) * \text{normalBirthRate}$	Flow: Both heifers and cows are assumed to have the same birth rate, 1 calf per year per cow.
calfMaturing [cows/Months]	$\text{Calves} / \text{timeToBeHeifer}$	Variable: This is the total number of calves that become cattle ready to either be allocated to be heifers or sent to culling. This variable also includes males.
cattleForMeatProduction [cows/month]	$\text{Cows} * \text{monthlyYieldDairyMeat}$	Variable: A portion of the cow herd is assumed to be culled for meat production per SYNAGRI data
costPerCow [NOK/(cows*month)]	$27000 / 12$	Parameter: variable costs for dairy cows include forages, concentrates, and miscellaneous needs in livestock care (Asheim et al., 2014)
culling [cows/Months]	$\text{Cows} / \text{normalCowLifetime}$	Flow: After the productive years of a dairy cow, it is assumed that they are sent for meat processing.
cullingCalves [cows/Months]	$\text{calfMaturing} - \text{allocatingHeifers}$	Flow: Out of all of the calves not allocated to heifer population is assumed to be sent out for culling and meat production
delayFive [NOK/month]	$\text{delayMaterial}(\text{addingFive}, 5, 0, 0)$	Material delay: This material delay aids in calculating the running annual income average starting from the 5 th year by starting to subtract from the stock <i>fiveAccum</i> .
desired_herd_size [cows]	$\min(\text{maxAllowedCows}, \text{indicatedHerdSize})$	Min: Desired cow herd can be increased up to the ceiling of maximum allowed herd size of 500 for a given farm size.
desiredBreederStock [cows/Months]	$\max(0, (\text{gapCowHerd} + \text{gapHeifers}) / \text{herdAdjTime} + \text{culling})$	Max: The desired number of new calves allocated for milk production should equal the number of desired number of cows and heifers PLUS replacement of culled cows. However, due to the dynamic this value may be negative, which is not possible in a cow herd, so a floor of 0 desired new cow is implemented.
desiredHeifers [cows]	$\text{maxHerdSize} * \text{timeToBecomeCow} / \text{normalCowLifetime}$	Variable: This value is calculated the same as the initial Heifer stock value. The equation shows the analytical relationship between the current desired cow herd and the desired heifer cow herd
desiredTotalRawMilk [tonnes/month]	From partner processor	Java Variable: The total volume of milk requested by municipalities and thus relayed to farm agents as production demand

Farm Variable [unit]	Equation	Documentation
effectOfNeighborsOnCost	 <p>Points: (0.000, 1.000), (5.000, 0.750)</p>	Table function: Number of neighbors within a distance (5km default) allows for cost reduction down to 0.75.
effectOfPerceivedFarmViability [dmn1]	 <p>Points: (0.000, 0.500), (1.000, 1.000), (2.000, 2.000)</p>	Table function: The more a farmer perceives that their farm is financially successful, they will want to continue their success by expanding further. If they find that their farm is not successful, they may look to reducing their farm for cost savings. "Success to successful" archetype.
ceaseFarmingProb	0.25	Parameter: If the farm finds that if it is underperforming, this is the probability at which it will close
farmIncomeViability	0.15	Parameter: Per SYNAGRI model data, this is the fraction of benchmarkIncome which recentIncome must not be lower than when compared to benchmarkIncome. If recentIncome is lower in the annual financial check, then the agent will check if it should close
farmMilkContribution [dmn1]	$\text{farmMilkProduction} / \text{totalFarmsRawMilk}$	Variable: This fraction is calculated to determine the portion of the total desired milk from processors the agent is responsible for
farmMilkProduction [Liters/Months]	$\text{Cows} * \text{monthlyYieldRawMilk}$	Variable: Only cows that have given birth can produce milk, and milk yield is averaged out for all cows.
femaleFraction [dmn1]	0.5	Parameter: While sexed breeding is possible with cows, for simplicity the biological 50-50 chance is used (geno.no, accessed 2023).
fiveYears [year]	5	Parameter: Allows for calculating the running 5-year annual average income, <i>recentIncome</i>

Farm Variable [unit]	Equation	Documentation
fixedFarmCost [NOK/month]	225000 / 12 {Asheim}	Parameter: The fixed costs include farm structures, machinery, maintenance, as well as administration and management. Costs also include hired labour (Asheim et al., 2014).
fractionDisposableIncome [dmnl]	0.3	This portion of accumulated cash reserve is assumed to go to pay the farmer's own salary.
gapCowHerd [cows]	desiredHerdSize - Cows	Variable: The difference between the current cow herd size and the desired cow herd size is the basis for driving herd growth behavior.
gapHeifers [cows]	desiredHeifers - Heifers	Variable: Once the number of desired heifers is determined then the difference between the current number of heifers can be calculated
gettingCows [cows/month]	cowsFromNeighbor * timeToGetCows	Flow: inflow into farm agent's mature Cow stock from closing farm that sold its herd
herdAdjTime [months]	60	Parameter: The long adjustment time is rooted in the long lifetime of cattle livestock buildings and related infrastructure, which limit the flexibility with which farmers enter and exit the cattle sector.
indicatedHerdSize [cows]	proportionDesiredMilk / monthlyYieldRawMilk * effectOfPerceivedFarmViability	Variable: The current herd is influenced by the effect of perceived farm viability, and the desired herd is adjusted up or down.
maxAllowedCows [cows]	500	Parameter: Assumption that a dairy farm cannot sustain more than a certain population (500 default) of cows
maxNewHeifers [cows/Months]	calfMaturing * femaleFraction	Max: The maximum number of new heifers is the number of female weaned calves each year.
minAllowedCows [cows/month]	0	Parameter: Physically impossible for recruitment rate to be less than 0 cows/month
meatRevenue [NOK/month]	cattleForMeatProduction * ppDairyCowMeat	Variable: Number of culled cows are multiplied by producer price (pp) to get the total revenue from culling activities.
milkRevenue [NOK/month]	farmMilkProduction * ppRawMilk	Variable: Revenue from raw milk shipment is the amount of shipped raw milk multiplied by its price.
monthlyYieldDairyMeat [cows/(cows*month)]	SYNAGRI data	Parameter: The portion of the cow herd culled for meat production is sourced from SYNAGRI data
monthlyYieldRawMilk [tonnes/(cows*month)]	From SYNAGRI data	Parameter: Average dairy cow yield data from SYNAGRI is different for each county, but this study only uses yield from Vestland county
normalBirthRate [cows/(cows*month)]	1/12	Parameter: Dairy cows typically produce a new calf once every year, which is required to maintain milk production (geno.no, accessed 2023).
normalCowLifetime [months]	48	Parameter: Determines the average productive life in years that mature cows are milked. After the milking period they are assumed to be processed for meat. Due to variability by region, practice, and breed, the productive life can range from 3 up to 6 years. A conservative estimate of 4 years is used (Dallago et al., 2021).
normalOER [dmnl]	0.7	Parameter: operating expense ratio of 0.7 is considered healthy for a dairy farm (dairyherd.com, accessed 2023).
numDairyCows [cows]	SYNAGRI data	Parameter: This initializing cow herd value is imported from SYNAGRI data.

Farm Variable [unit]	Equation	Documentation
operatingExpenseRatio [dmnl]	spending / revenue	Variable: Operating expense ratio can help gauge the efficiency of a company. It is the fraction of revenue that is spent on operation (zebrabi.com, accessed 2023).
parturition [cows/Months]	Heifers / timeToBecomeCow	Flow: As heifers give their first birth, they are considered cows, and go to the main milk producing population.
perceptionGap [dmnl]	relativeOER - perceivedFarmViability	Variable: difference between the current relative OER and perceivedFarmViability is calculated to determine how much the perception should be adjusted, up or down.
personalSpending [NOK/month]	cashReserve * fractionDisposableIncome	Variable: A fraction of the available cash reserve is spent on discretionary and disposable consumption.
ppDairyCowMeat [NOK/cows]	SYNAGRI data	Parameters: The producer price (pp) data for culled cows is imported from the SYNAGRI data.
ppRawMilk [NOK/tonne]	SYNAGRI data	Parameter: Selling price of farm raw milk per tonne is sourced from SYNAGRI model data
proportionDesiredMilk [tonnes/month]	desiredTotalRawMilk * farmMilkContribution	Variable: This calculation determines the production target for the farm agent for this time step.
recentIncome [NOK/year]	fiveAccum / fiveYears	Variable: Starting from year 5 of simulation, this variable continuously calculates the annual average income over a 5 year period
recentToBenchmark [dmnl]	recentIncome / benchmarkIncome	Variable: The fraction of recentIncome compared to benchmarkIncome calculates a value which if below a threshold (0.85 default) will trigger a decision process to close the farm.
relativeOER [dmnl]	normalOER / operatingExpenseRatio	Variable: The normalizing value is in the numerator because a smaller OER is more desirable, thus a smaller OER is perceived as more viable
revenue [NOK/month]	totalRevenue	Flow: Total revenue is the only contributor to the inflow to Cash Reserve.
spending [NOK/month]	totalFarmCost	Flow: Total spending outflow comes from farm operation cost, cheese production cost, and fraction from personal consumption
subsidies [NOK/month]	(subsidyNonlivestock + subsidyMilk + subsidyLivestock) / 12	Parameter: Subsidies from SYNAGRI data are summed into one variable
subsidyLivestock [NOK/year]	SYNAGRI data	Parameter: Per SYNAGRI data each farm agent has unique amounts of subsidy for livestock husbandry
subsidyMilk [NOK/year]	SYNAGRI data	Parameter: Per SYNAGRI data each farm agent has unique amounts of subsidy for operating a dairy farm
subsidyNonlivestock [NOK/year]	SYNAGRI data	Parameter: Per SYNAGRI data each farm agent has unique amounts of subsidy for operating a farm in general
timeToBecomeCow [months]	12	Parameter: Once a heifer births her first calf, it is considered a cow (swandairy.com, accessed 2023).
timeToBeHeifer [months]	24	Parameter: Cows typically give birth for the first time (i.e. allocated to be heifers) 2 to 3 years of age (ciwf.org.uk, accessed 2023).
timeToGetCows [months]	1	Parameter: Assumption that a herd sold to the agent will be fully incorporated within a year

Farm Variable [unit]	Equation	Documentation
totalFarmCost [NOK/month]	$(\text{fixedFarmCost} + \text{variableFarmCost}) * \text{neighborMultiplier}$	Variable: total cost per time is simply the combination of the variable per cow cost and fixed farm cost. A multiplier is included for scenario testing of cost reduction via resource pooling with neighbors
totalFarmsRawMilk [tonnes/month]	From partner processor	Java Variable: The total volume of milk purchased by the partnering dairy processor is used to calculate the agent's proportional contribution to that total
totalRevenue [NOK/Months]	$\text{milkRevenue} + (\text{meatRevenue} + \text{totalSubsidy}) * \text{nonMilkRevenue}$	Variable: Sum of revenue from selling milk, meat, and receiving subsidies
variableFarmCost [NOK/month]	$\text{Cows} * \text{costPerCow}$	Variable: The per cow cost is multiplied by the cow herd

Dairy Processor Agent

Processor Variable [unit]	Equation	Documentation
wholesaleStockDairy [tonnes]	$\text{INTEG} (+ \text{processingMilk} - \text{shippingDairy} , [\text{wholesaleStockMilkCream}, \text{wholesaleStockYoghurt}, \text{wholesaleStockButter}, \text{wholesaleStockCheese}])$	Stock [Arrayed]: This stock is represents the full amount that will be shipped out to nearby municipalities this timestep. Arrayed by the four dairy product categories. The full amount is expected to be shipped out, but is consistently replenished through the inflow processingMilk. Initial values for each dairy product category are parameter inputs.
supplyRawMilk [tonnes]	$\text{INTEG} (+ \text{sourcingMilk} - \text{toProcessing}, \text{supplyRawmilk})$	Stock: This stock acts equivalent to the receiving warehouse for the processor. Raw milk produced by partnered farms inflow into this stock. Initial values are calculated parameter inputs
totalConsumptionMilkCream [tonnes/month]	SYNAGRI data	Java variable: Milk consumption demand from all municipalities partnered with this processor agent are summed and relayed to calculate the goal in a goal-gap structure
totalConsumptionYoghurt [tonnes/month]	SYNAGRI data	Java variable: Yoghurt consumption demand from all municipalities partnered with this processor agent are summed and relayed to calculate the goal in a goal-gap structure
totalConsumptionButter [tonnes/month]	SYNAGRI data	Java variable: Butter consumption demand from all municipalities partnered with this processor agent are summed and relayed to calculate the goal in a goal-gap structure
totalConsumptionCheese [tonnes/month]	SYNAGRI data	Java variable: Cheese consumption demand from all municipalities partnered with this processor agent are summed and relayed to calculate the goal in a goal-gap structure
consumptionDairy [tonnes/month]	$[\text{totalConsumptionMilkCream}, \text{totalConsumptionYoghurt}, \text{totalConsumptionButter}, \text{totalConsumptionCheese}]$	Variable [Arrayed]: This variable simply collects the total consumption demand into one arrayed variable for further calculations
desiredWholesaleDairy [tonnes]	$\text{consumptionDairy}[\text{Dairy}] * \text{shippingTime}$	Variable [Arrayed]: The consumption demand rate is multiplied by the expected shipping time to determine the tonnes of dairy products that need to be processed

Processor Variable [unit]	Equation	Documentation
desiredWSbyProd [tonnes]	$\text{desiredWholesaleDairy}[\text{Dairy}] * \text{rawMilkPerDairyProduct}[\text{Dairy}] * (\text{processingTime} / \text{shippingTime})$	Variable [Arrayed]: The amount of raw milk needed per demanded dairy product is calculated
gapRawMilkFromRM [tonnes]	$\text{desiredWSbyProd.sum}() - \text{supplyRawMilk}$	Variable: The arrayed raw milk demand is summed and subtracted by the current raw milk supply stock to determine the gap of raw milk
totalDesiredRawMilk [tonnes/month]	$\text{sumShippedDairy} + \text{lossFromProcessing} + (\text{gapRawMilkFromWS} + \text{gapRawMilkFromRM}) / \text{purchaseTime}$	Variable: Gap of raw milk from the wholesale stock and raw milk stock, as well as processing loss and loss through shipping outflow are summed together to determine how much raw milk should be requested of partnered farm agents
purchaseTime [month]	1	Parameter: It is assumed required raw milk will be ordered and purchased within the month
gapWholesaleStockDairy [tonnes]	$\text{desiredWholesaleDairy}[\text{Dairy}] - \text{wholesaleStockDairy}[\text{Dairy}]$	Variable [Arrayed]: The desired consumption of dairy products by the municipalities are subtracted by the current inventory of dairy products to determine the gap in inventory
requiredRawMilkByProdFromWS [tonnes]	$\text{gapWholesaleStockDairy}[\text{Dairy}] * \text{rawMilkPerDairyProduct}[\text{Dairy}]$	Variable [Arrayed]: The gap in dairy products is converted to their raw milk equivalents, but still arrayed by the four product categories
gapRawMilkFromWS [tonnes]	$\text{requiredRawMilkByProdFromWS.sum}()$	Variable: The arrayed variable of dairy product gap in raw milk equivalent is summed into one value
wholesaleStockMilkCream [tonnes]	SYNAGRI data	Parameter: Initial milk dairy product stock is calculated using SYNAGRI model data
wholesaleStockYoghurt [tonnes]	SYNAGRI data	Parameter: Initial yoghurt dairy product stock is calculated using SYNAGRI model data
wholesaleStockButter [tonnes]	SYNAGRI data	Parameter: Initial butter dairy product stock is calculated using SYNAGRI model data
wholesaleStockCheese [tonnes]	SYNAGRI data	Parameter: Initial cheese dairy product stock is calculated using SYNAGRI model data
shippingTime [month]	1	Parameter: It is assumed that any product in the processor warehouse stock is shipped out within the month.
processingMilk [tonnes/month]	$\text{milkForProcessing}[\text{Dairy}] / \text{rawMilkPerDairyProduct}[\text{Dairy}]$	Flow [Arrayed]: The raw milk allocated to each dairy product category is converted into the expected amount of final dairy products and flows into the warehouse stock
shippingDairy [tonnes/month]	$\text{wholesaleStockDairy}[\text{Dairy}] / \text{shippingTime}$	Flow [Arrayed]: It is assumed that dairy products do not remain in warehouse and are shipped out within the month
shippedInRawMilk [tonnes/month]	$\text{shippingDairy}[\text{Dairy}] * \text{rawMilkPerDairyProduct}[\text{Dairy}]$	Variable [Arrayed]: The amount of just shipped dairy products are converted into their raw milk equivalents
sumShippedDairy [tonnes/month]	$\text{shippedInRawMilk.sum}()$	Variable: The arrayed shipped dairy products converted into raw milk equivalents are summed into one value to be added to the total raw milk requirement
rawMilkPerDairyProduct [dmn1]	SYNAGRI data	Parameter [Arrayed]: Tonnes of raw milk needed to produce 1 tonne of dairy product varies by category, so this arrayed conversion factor is used throughout the structure

Processor Variable [unit]	Equation	Documentation
consRawMilkEquivalents [tonnes/month]	consumptionDairy[Dairy] * rawMilkPerDairyProduct[Dairy]	Variable [Arrayed]: Consumption demand rate arrayed by dairy products from municipalities are converted into their raw milk equivalent
sumMilkConsumption [tonnes/month]	consRawMilkEquivalents.sum()	Variable: The arrayed consumption demand by dairy product categories are summed into one raw milk value
rawMilkDivisionRatio [dmn1]	(consumptionDairy[Dairy] * rawMilkPerDairyProduct[Dairy]) / sumMilkConsumption	Variable [Arrayed]: Raw milk requirements per arrayed dairy product categories are each divided by the total raw milk requirement to determine how the procured raw milk should be apportioned during processing
relativeFulfillment [dmn1]	sumShippedDairy / sumMilkConsumption	Variable: Ratio between consumption demand and the actual amount of dairy product delivered
fulfillmentGap [tonnes/month]	sumMilkConsumption - sumShippedDairy	Variable: The consumption rate is subtracted by the shipping rate to determine deficit or excess
indicatedFarmsNeeded [farms]	fulfillmentGap / avgFarmsRawMilk	Variable: Number of additional farm partners is calculated by considering the gap in fulfillment
dairyMinCapacity [dmn1]	0.85	Parameter: Per SYNAGRI model data this is the minimum relativeFulfillment value that is acceptable for dairy processors
avgFarmsRawMilk [tonnes/[farm*month]	totalFarmsRawMilk / partnerFarms	Variable: Raw milk delivery rate for all currently partners farms are divided by the number of farms to determine the average raw milk delivery per farm monthly
partnerFarms [farms]	partnerFarms.size()	Java Variable: Number of elements in the parterFarms list array is counted
totalFarmsRawMilk [tonnes/month]	From partnered dairy farms	Java Variable: Raw milk production from all partnered dairy farms are summed into one variable
sourcingMilk [tonnes/month]	totalFarmsRawMilk	Flow: The summed total farm raw milk production variable is simply used in this inflow
lossFromProcessing [tonnes/month]	supplyRawMilk * processingWaste / processingTime	Flow: All raw milk received from farms are expected to be processed within the month with a portion of the volume lost to processing waste
toProcessing [tonnes/month]	supplyRawMilk * (1 - processingWaste) / processingTime	Flow: Received raw milk not lost to processing outflows to the final dairy product processing
milkForProcessing [tonnes/month]	toProcessing * rawMilkDivisionRatio[Dairy]	Variable [Arrayed]: The milk left over after processing loss is apportioned their respective dairy product categories.
processingWaste [dmn1]	0.1105857	Parameter: SYNAGRI model assumption - for 1 tonne of raw milk procured, 0.11 tonne is assumed to be lost in the processing stage
processingTime [momnth]	1	Parameter: All raw milk procured is assumed to undergo processing within the month

Main, Kommuner, and Fylke Agents

Processor Variable [unit]	Agent	Equation	Documentation
policyYear [year]	Main	5	Parameter: The year from which farm agents will begin self-evaluating financial performance. If they find recentIncome is too low compared to benchmarkIncome they will check if they need to close.
farmsFindNeighbors	Main	true	Boolean: If set to true the farm agents will seek out other farm agents to form dynamic network relationships
farmsSearchDistance	Main	true	Boolean: If farmFindNeighbors AND this parameter are both set to true, the farms will find their neighbors within a strict radius (5km default). If this parameter is false, but farmFindNeighbors is still true, the farms will instead search an incrementally increasing radius until it finds 5 (by default) neighbors.
sellCowsWhenClosing	Main	false	Boolean: If farmsFindNeighbors AND this parameter are set to true, then a closing farm agent will transfer its cow herd population to a neighbor with the greatest perceivedFarmViability.
costSharing	Main	false	Boolean: If farmsFindNeighbors AND this parameter are set to true, depending on the number of neighbors the farm agents find, they can reduce their total operation cost by up to 25%.
createNewFarms	Main	false	Boolean: If set to true, if a dairy processor underdelivers dairy products by a certain threshold compared to the consumption demand, it can begin creating new farm agents to try to boost production.
herdGrowth	Main	true	Boolean: If set to true, this will allow the farm cow herd to grow instead of staying static like the agent-based model
nonMilkRevenue	Main	1	Switch parameter: If 1, farm agents will obtain income from non-milk sources such as meat and subsidies.
monthsPerYear [month / year]	Main	12	Parameter: Conversion from months to year
BeefKgPerCapita [kg/(people*year)]	Main	13.4001	Parameter: Country-wide beef consumption per capita rate
MilkCreamKgPerCapita [kg/(people*year)]	Main	128.3	Parameter: Country-wide milk consumption per capita rate
YoghurtKgPerCapita [kg/(people*year)]	Main	9.82	Parameter: Country-wide yoghurt consumption per capita rate
ButterKgPerCapita [kg/(people*year)]	Main	3.18	Parameter: Country-wide butter consumption per capita rate
CheeseKgPerCapita [kg/(people*year)]	Main	17.6	Parameter: Country-wide cheese consumption per capita rate
rmPerMilkCream [tonne/tonne]	Main	1	Parameter: Amount of raw milk needed to produce milk
rmPerYoghurt [tonne/tonne]	Main	1.13	Parameter: Amount of raw milk needed to produce yoghurt
rmPerButter [tonne/tonne]	Main	20	Parameter: Amount of raw milk needed to produce butter
rmPerCheese [tonne/tonne]	Main	6.91	Parameter: Amount of raw milk needed to produce cheese
annualMilkPrice [NOK/tonne]	Main	SYNAGRI data	Parameters: The producer price (pp) data for culled cows is imported from the SYNAGRI data.
annualMeatPrice [NOK/tonne]	Main	SYNAGRI data	Parameter: Selling price of meat per tonne is sourced from SYNAGRI model data
yieldRawMilk [tonnes/(cows*year)]	Fylke	SYNAGRI data	Parameter: Average dairy cow yield data from SYNAGRI is different for each county, but this study only uses yield from Vestland county

Processor Variable [unit]	Agent	Equation	Documentation
yieldDairyCowCarcass [tonnes/(cows*year)]	Fylke	SYNAGRI data	Parameter: The portion of the cow herd culled for meat production is sourced from SYNAGRI data
monthlyYieldRawMilk [tonnes/(cows*month)]	Fylke	yieldRawMilk / monthsPerYear	Variable: The annual raw milk yield data is converted to per monthly
monthlyYieldDairyCowCarcass [tonnes/(cows*month)]	Fylke	yieldDairyCowCarcass / monthsPerYear	Variable: The annual cow carcass yield data is converted to per monthly
Population [people]	Kommune	SYNAGRI data	Parameter: Parameter: Population projection data is sourced from SYNAGRI data and unique to each Kommune agent
kgPerTonne [kg/tonne]	Kommune	1000	Parameter: Conversion factor from kg to metric tonne
consumptionBeef [tonne/month]	Kommune	PCBeef * population / kgPerTonne / monthsPerYear	Variable: Per capita consumption is multiplied by kommune population to get the demand for beef
consumptionMilkCream [tonne/month]	Kommune	(PCMilkCream * population / kgPerTonne) / monthsPerYear	Variable: Per capita consumption is multiplied by kommune population to get the demand for milk
consumptionYoghurt [tonne/month]	Kommune	PCYoghurt * population / kgPerTonne / monthsPerYear	Variable: Per capita consumption is multiplied by kommune population to get the demand for yoghurt
consumptionButter [tonne/month]	Kommune	PCButter * population / kgPerTonne / monthsPerYear	Variable: Per capita consumption is multiplied by kommune population to get the demand for butter
consumptionCheese [tonne/month]	Kommune	PCCheese * population / kgPerTonne / monthsPerYear	Variable: Per capita consumption is multiplied by kommune population to get the demand for cheese

Appendix E: AnyLogic Java Code Documentation

Function name [Agent]	Java code	Documentation
On startup [Farms]	<pre>setFarmPartners(); if(main.farmsFindNeighbors == true) { if(main.farmsSearchDistance == true) { neighborsByDistance(); } else { neighborsNearestFive(); } } if(main.costSharing == true) { countNeighbors = neighborFarms.size(); } }</pre>	<p>When farm agent is created, first call for function <i>setFarmPartners()</i>. Then if the switch <i>farmsFindNeighbors</i> is true then if the <i>farmsSearchDistance</i> switch is true then call for function <i>neighborsByDistance()</i>. Else call for call for function <i>neighborsNearestFive()</i>. Finally if the <i>costSharing</i> scenario switch is true then count the number of neighbors identified</p>
setFarmPartners [Farms]	<pre>dairy partnerDairy = this.getNearestAgentByRoute(main.dairies); this.set_partnerDairy(partnerDairy); partnerDairy.partnerFarms.add(this); for (Kommune k : main.kommuner) { if (this.kommuneId == k.kommuneId) { this.set_partnerKommune(k); } for (Fylke y : main.fylker) { if (k.fylkeId == y.fylkeId) { k.set_partnerFylke(y); this.set_partnerFylke(y); } } }</pre>	<p>Find the farm agent's closes processor, and make it the partnered processor.</p> <p>Also, take the farm's unique municipality ID and match it with the list of municipalities with their IDs and match. Take the farm's unique county ID and also match with list of counties and their IDs.</p>
neighborsByDistance [Farms]	<pre>neighborFarms.addAll(agentsInRange(main.farms, 5, KILOMETER)); if(neighborFarms.size() != 0) for(farm f : neighborFarms) { f.farmsConsiderMeNeighbor.add(this); }</pre>	<p>Find all other farms within a 5 km range and add to array list <i>neighborFarms</i>. If there are any neighbors found, then add own self to the neighbors' array list <i>farmsConsiderMeNeighbor</i></p>
neighborsNearestFive [Farms]	<pre>ArrayList<farm> farmCollection = new ArrayList<farm>(); farmCollection.addAll(agentsInRange(main.farms, 50, KILOMETER)); while(neighborFarms.size() < 5 && farmCollection.size() > 0) {</pre>	<p>Create an empty array list <i>farmCollection</i>. Search for number of neighbors within a 5 km radius. If there are less than 5 neighbors, expand the search by 5km increments until at least 5</p>

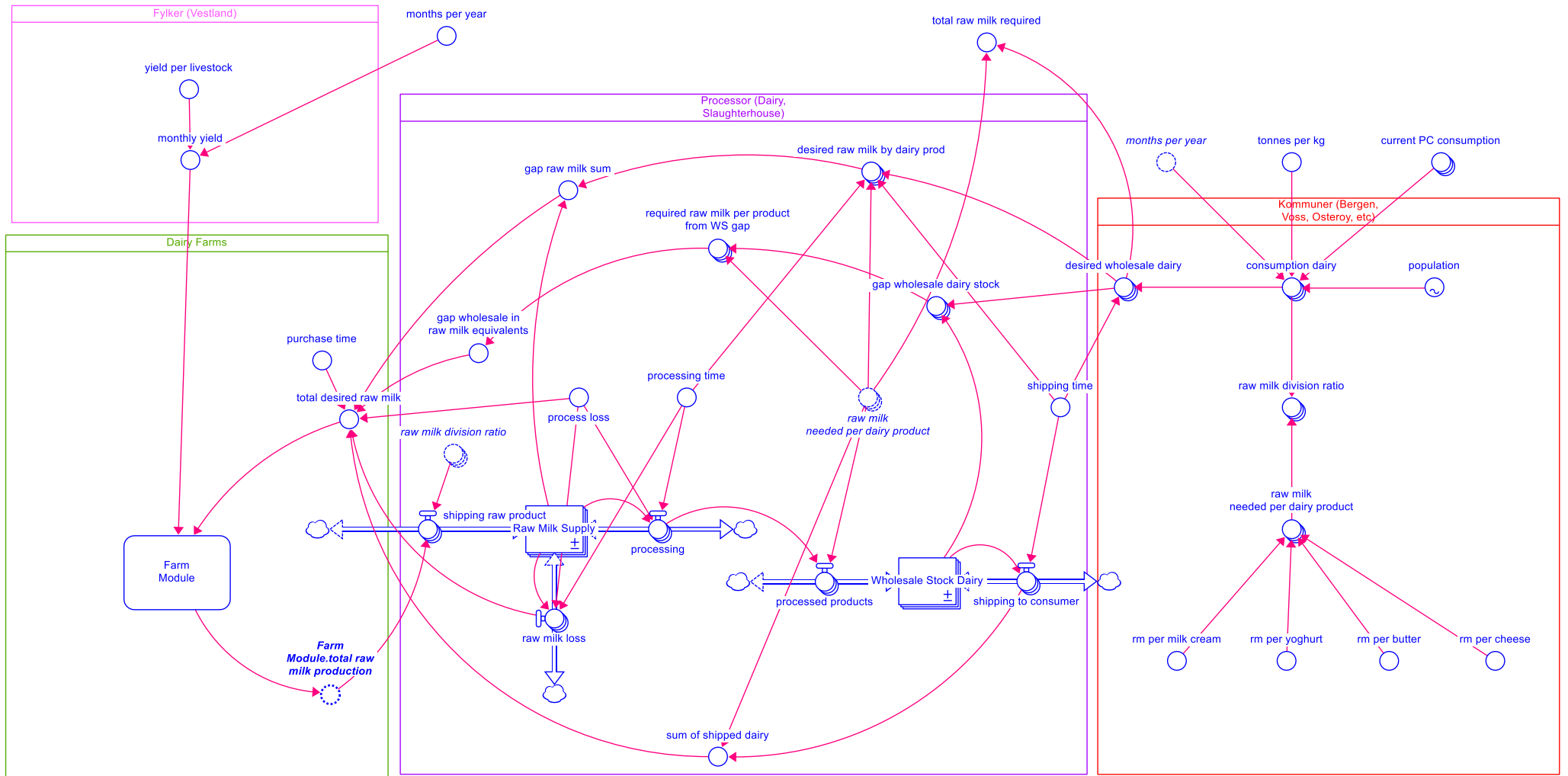
Function name [Agent]	Java code	Documentation
	<pre> farm newNeighbor = this.getNearestAgentByRoute(farmCollection); neighborFarms.add(newNeighbor); farmCollection.remove(newNeighbor); } if(neighborFarms.size() != 0) for(farm f : neighborFarms) { f.farmsConsiderMeNeighbor.add(this); } </pre>	<p>are found. Once at least 5 neighbors are found add to the <i>farmCollection</i> array list.</p> <p>From the <i>farmCollection</i> array find the 5 nearest neighbor farms and add to array <i>neighborFarms</i>.</p> <p>Add own self to the neighbors' array list <i>farmsConsiderMeNeighbor</i></p>
totalFarmsRawMilk [Farms]	<pre> double totalMilk = 0; if(partnerDairy != null) { totalMilk = partnerDairy.totalFarmsRawMilk; } return totalMilk; </pre>	Obtain the total amount of raw milk the partner dairy processor is currently procuring from all dairy farms
desiredTotalRawMilk [Farms]	<pre> double totalMilk = 0; if(partnerDairy != null) { totalMilk = partnerDairy.totalDesiredRawMilk; } return totalMilk; </pre>	Obtain the total amount of raw milk required by the partner dairy processor
monthlyYieldRawMilk [Farms]	<pre> double monthlyMilkYield = 0; if(partnerFylke != null) { monthlyMilkYield = partnerFylke.monthlyYieldRawMilk; } return monthlyMilkYield; </pre>	Obtain the current milk yield value from the partnered county
monthlyYieldDairyMeat [Farms]	<pre> double monthlyMeatYield = 0; if(partnerFylke != null) { monthlyMeatYield = partnerFylke.monthlyYieldDairyCowCarcass; } return monthlyMeatYield; </pre>	Obtain the current dairy cattle meat yield from the partnered county
closeFarmDecision [Farms]	<pre> if (recentToBenchmark < (1 - farmIncomeViability) && gotCows.isActive() != true) { if (Math.random() < ceaseFarmingProb) { if(main.sellCowsWhenClosing == true) { sellCowsToNeighbor(); } createBrownfield(); closeFarm(); } } </pre>	<p>If the <i>recentToBenchmark</i> ratio is less than the $(1 - farmIncomeViability)$, which is 0.85 by default AND have not received cow herd from a neighbor farm in the recent year, then enter decision process to close farm.</p> <p>With a probability of <i>ceaseFarmingProb</i> (0.25 by default), if the <i>sellCowsWhenClosing</i> switch is true, then call the function <i>sellCowsToNeighbor()</i>. Then create a new <i>ClosedFarm</i> agent in place of oneself for record keeping. Then finally call the <i>closeFarm()</i> function.</p>

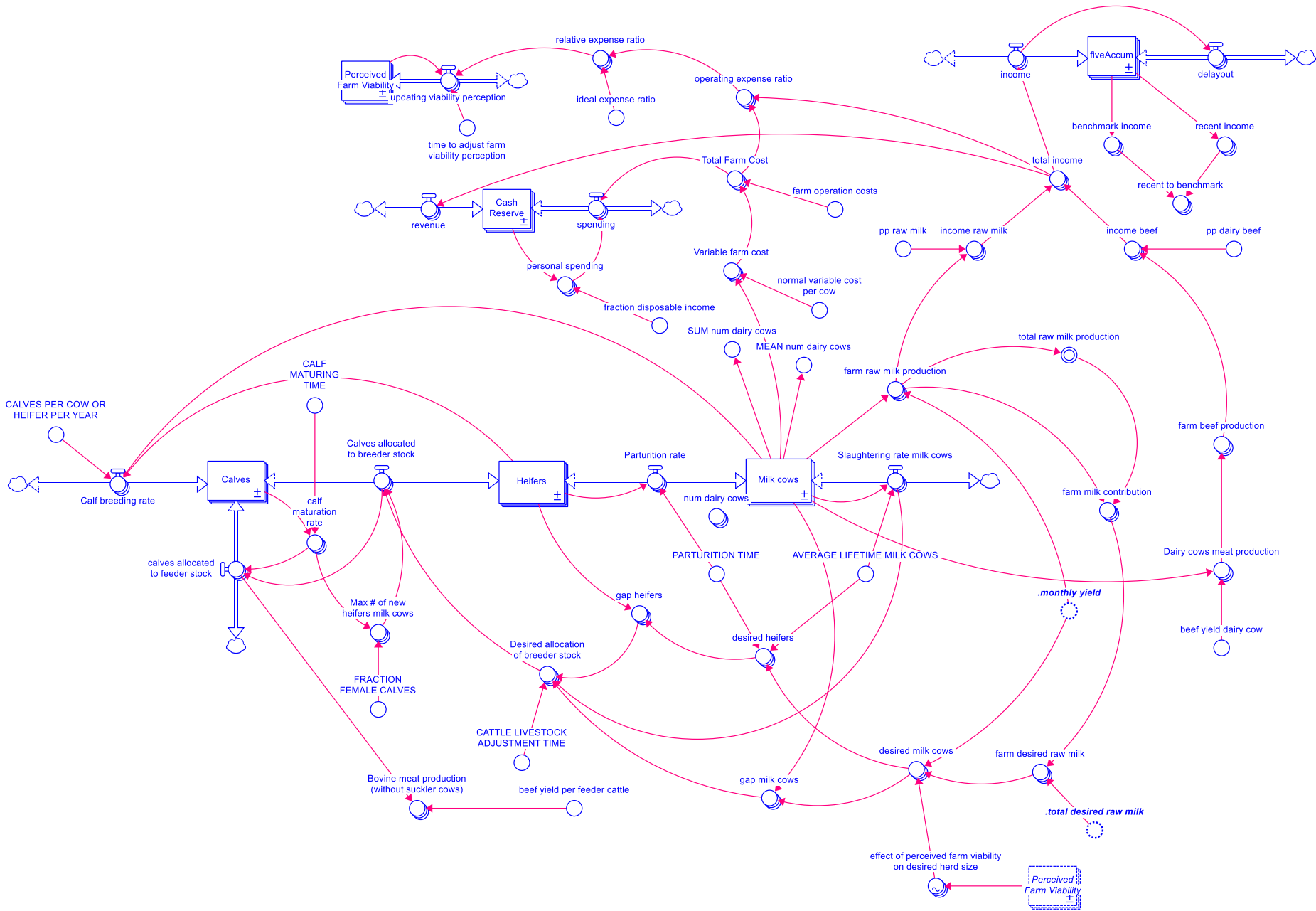
Function name [Agent]	Java code	Documentation
sellCowsToNeighbor [Farms]	<pre>if(neighborFarms.size() != 0) { richNeighbor = top(neighborFarms, n -> n.perceivedFarmViability); richNeighbor.cowsFromNeighbor = Cows + richNeighbor.cowsFromNeighbor; richNeighbor.gotCows.restart(); }</pre>	<p>If the farm has any neighbors, then find the neighbor farm with the best perceived farm viability and give own population of cows to that neighbor.</p> <p>The receiving neighbor is also safe from itself closing for 1 year to see if its financial situation improves</p>
closeFarm [Farms]	<pre>if(farmsConsiderMeNeighbor.size() != 0) { for(farm f : farmsConsiderMeNeighbor) f.neighborFarms.remove(this); } partnerDairy.partnerFarms.remove(this); main.remove_farms(this);</pre>	<p>If other farms consider me neighbor, then remove oneself from each of their <i>farmsConsiderMeNeighbor</i> array list.</p> <p>Remove oneself from the dairy processor's list of partnered dairy farms</p> <p>Finally, remove oneself from agent population of dairy farms in the simulation</p>
createBrownfield	<pre>ClosedFarm justClosedFarm = main.add_closedFarms(); justClosedFarm.setLocation(this); justClosedFarm.Cows = this.Cows; justClosedFarm.farmId = this.farmId; justClosedFarm.richNeighbor = this.richNeighbor; justClosedFarm.recentToBenchmark = this.recentToBenchmark; justClosedFarm.createdDate = this.createdDate; justClosedFarm.countNeighbors = this.neighborFarms.size(); justClosedFarm.closedDate = date(); if(richNeighbor != null) richNeighbor.farmsThatSoldToMe.add(justClosedFarm);</pre>	<p>Create one agent in the <i>closedFarms</i> population. Transfer to this new agent the following information about oneself: location, farmID, which neighbor received my cows, the income at which the farm closed, when it was created, how many neighbors, when was it closed.</p> <p>Add oneself to the neighbor agent's list <i>farmsThatSoldToMe</i> of farms that has given cows to it.</p>
totalFarmsRawMilk [Processor]	<pre>double totalFarmsRawMilk = 0; if(partnerFarms != null) { totalFarmsRawMilk = sum(partnerFarms, f -> f.farmMilkProduction); } return totalFarmsRawMilk;</pre>	<p>Take the raw milk production from all the partnered dairy farms and sum them</p>
totalConsumptionMilkCream totalConsumptionYoghurt totalConsumptionButter totalConsumptionCheese [Processor]	<pre>double sumMilkCream = 0.0; double sumYoghurt = 0.0; double sumButter = 0.0; double sumCheese = 0.0; ArrayList<Double> dairyTotals = new ArrayList<Double>(); if(kommunerToSellTo != null) {</pre>	<p>Take the dairy product consumption demand from all partnered municipalities and sum them according to each dairy product array.</p> <p>Take the total consumption demand for each dairy product and add to a newly created, empty array list <i>dairyTotals</i>.</p>

Function name [Agent]	Java code	Documentation
	<pre> for(Kommune k : kommunerToSellTo) { sumMilkCream += k.consumptionMilkCream; sumYoghurt += k.consumptionYoghurt; sumButter += k.consumptionButter; sumCheese += k.consumptionCheese; } } dairyTotals.add(sumMilkCream); dairyTotals.add(sumYoghurt); dairyTotals.add(sumButter); dairyTotals.add(sumCheese); return dairyTotals; </pre>	
fulfillmentCheck [Processor]	<pre> double check = -1; if(time(YEAR) >= main.policyYear) { if(relativeFulfillment < dairyMinCapacity) { check = 1; } else { check = 0; } } return check; </pre>	<p>If the current relative fulfillment is less than the dairy minimum capacity (0.85 default) then the switch will be set to 1 (true) and dairy will begin requesting more farm agents, else the switch will be 0 and no new farms will be created</p>
needFarms [Processor]	<pre> if(fulfillmentCheck == 1) { farmDeficitWait.restart(); } </pre>	<p>If fulfillment is found to be deficient, then a 3-year waiting period is triggered where the processor agent decides that it needs to start signaling that more farm agents are needed</p>
enoughFarms [Processor]	<pre> if(fulfillmentCheck == 0) { farmDeficitWait.reset(); requestMoreFarms = false; } </pre>	<p>If fulfillment is found to be sufficient then the processor will cancel the default 3-year waiting period and also changes the requestMoreFarms switch to false.</p>
farmDeficitWait [Processor]	<pre> requestMoreFarms = true; minMaxDairy(); </pre>	<p>After a 3-year delay, and if enoughFarms function isn't called in the meantime, then the requestMoreFarms switch is set to true.</p> <p>Also, a max and minimum latitude and longitude is assessed as a boundary to create new farm agents</p>

Function name [Agent]	Java code	Documentation
minMaxDairy [Processor]	<pre> for(farm f : partnerFarms) { if(f.getLatitude() < minLatDairy) { minLatDairy = f.getLatitude(); } else if(f.getLatitude() > maxLatDairy) { maxLatDairy = f.getLatitude(); } if(f.getLongitude() < minLongDairy) { minLongDairy = f.getLongitude(); } else if(f.getLongitude() > maxLongDairy) { maxLongDairy = f.getLongitude(); } } </pre>	<p>For all farm agents partners with this processor agent, go through all the farms and find the minimum and maximum latitude and longitude values.</p> <p>Establish this as the “box” in which new farm agents will be created.</p>
creatingMoreFarms [Processor]	<pre> double openedFarmCows = 0; if(requestMoreFarms == true) { farm justOpenedFarm = main.add_farms(); justOpenedFarm.fylkeId = "46"; justOpenedFarm.partnerDairy = this; partnerFarms.add(justOpenedFarm); justOpenedFarm.jumpTo(uniform(minLatDairy, maxLatDairy), uniform(minLongDairy, maxLongDairy)); openedFarmCows = zidz(avgFarmsRawMilk, justOpenedFarm.partnerFylke.monthlyYieldRawMilk); justOpenedFarm.Cows = openedFarmCows; } </pre>	<p>If requestMoreFarms switch is activated, then create a new farm agent. Assign the county ID to be Vestland, then add the new farm agent to own array list of partnered farms.</p> <p>Randomly relocate the new farm within the bounds established by function minMaxDairy().</p> <p>To get the farm start operating, give the current average cow herd size to the Cows stock of the new farm. This allows the new agent to start contributing raw milk production, and also the cow herd can start growing.</p>
setProcessorPartners [Kommune]	<pre> dairy dairyToBuyFrom = this.getNearestAgentByRoute(main.dairies); this.set_dairyToBuyFrom(dairyToBuyFrom); dairyToBuyFrom.kommunerToSellTo.add(this); </pre>	<p>Have each municipality agent look for the closest dairy farm agent. Make the closest processor the partner dairy processor.</p> <p>Add oneself to the processor’s list of <i>kommunerToSellTo</i>.</p>

Appendix F: System Dynamics Model Equivalent in Stella





Appendix G: How to Navigate and Run the AnyLogic Model

To open and view the AnyLogic model, the modeling software first needs to be downloaded and installed. Upon clicking the “Download” link from their website, there will be an option for three versions—Personal Learning Edition (PLE), University Researcher, and Professional (anylogic.com, Accessed May 2024). The PLE version is free for non-commercial application and the hybrid model was built upon this edition.



Figure 32: Three AnyLogic versions available on the website, PLE is recommended and free to use

The AnyLogic software itself is based on the Java programming language, and upon first opening the software, there may be a prompt to install separately Java Runtime Environment 11 (JRE) or later. The hybrid model depends on numerous exogenous data which is sourced from Excel tables; therefore, the *alp* AnyLogic model file must remain in its original directory to run correctly.

The default setting opens a welcome page where the user can explore sample models. This page can however be closed to view the main model window (Figure 33). The model should show the *Main* model canvas, which is the top model structure which contains all the agents and model elements.

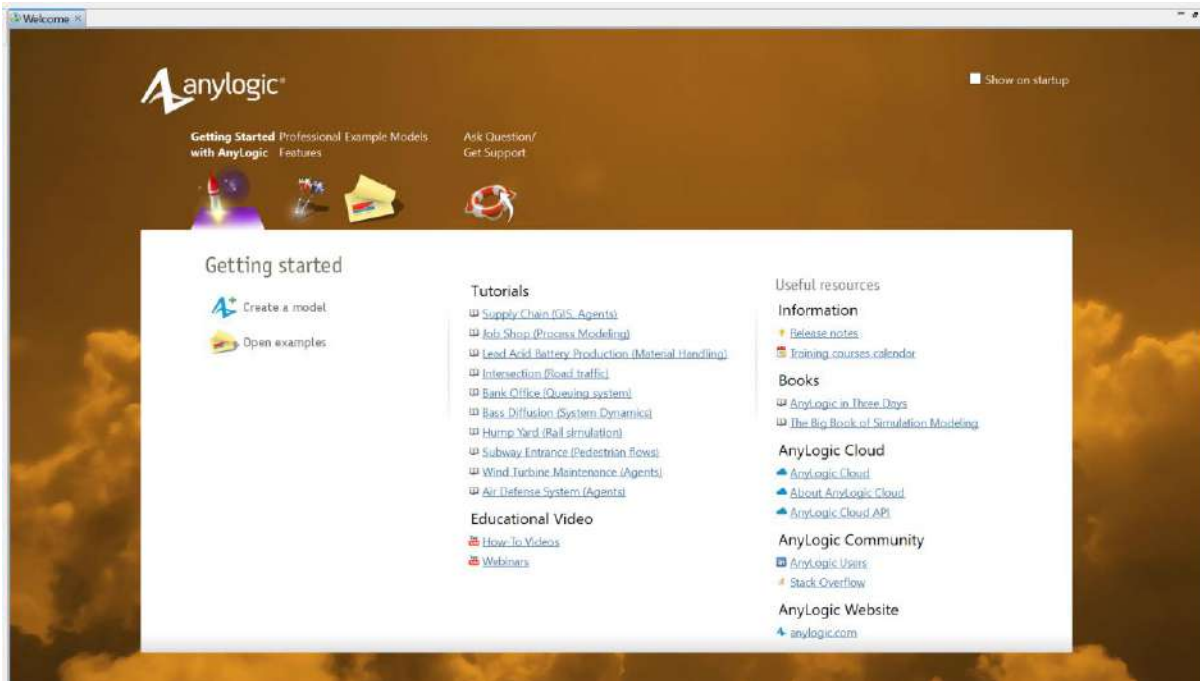


Figure 33: AnyLogic default welcome window

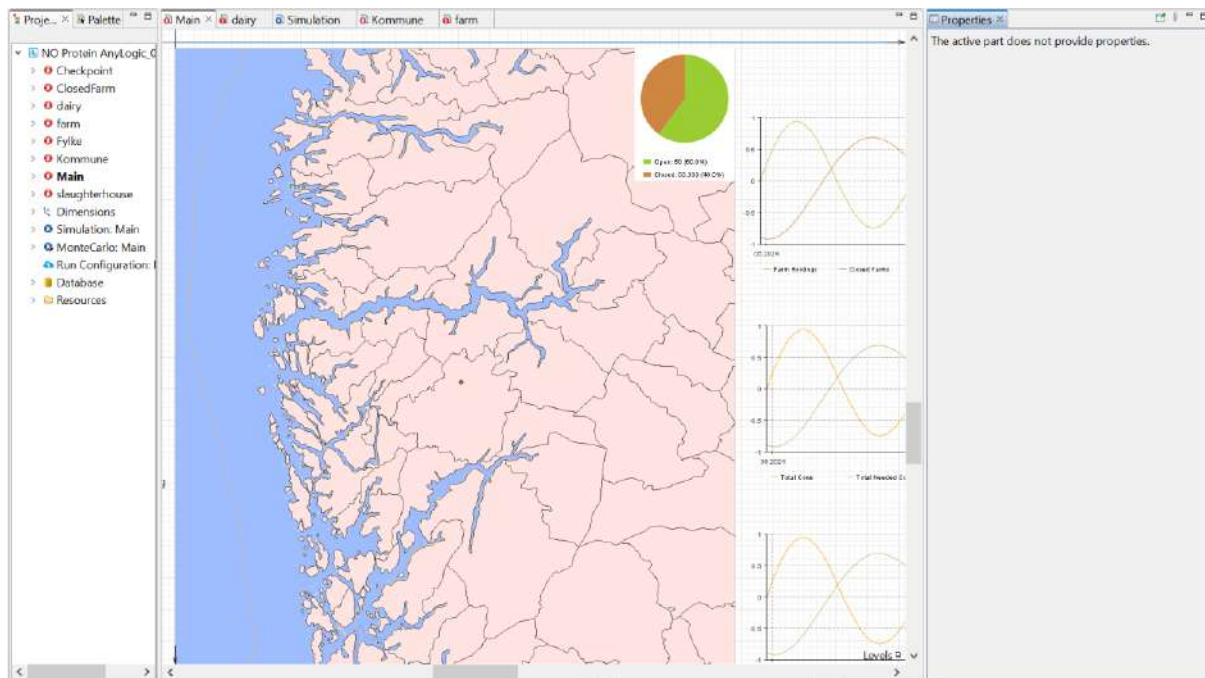


Figure 34: Main model canvas and Projects navigation panel on left side of the window and Properties on the right side

On the left side of the window there should be a side panel with two tabs, *Projects* and *Palette* (Figure 34). The *Palette* tab can be disregarded as that is used only for model construction. The *Projects* tab is a way to navigate all model elements like a folder structure. The red icons with the stick figure in the center represent agents placed in the model. The blue icons with X in the center represent the type of analyses configured in the model. *Simulation* is simply running the model, while *MonteCarlo* is the 1000-run Monte Carlo analysis described in section 4. The cylinder icon labelled *Database* is a repository of tables imported from Excel which the model uses to populate the agents during the simulation. Right side of the window is the *Properties*

frame, which will be used to view the documentation of model components such as the stock-and-flow and Java objects. For example, scrolling above the map will show several “Top-level” main *Parameter* objects, represented by a circle with a black wedge. Clicking on *rmPerMilkCream* will reveal data properties for this object (Figure 35).

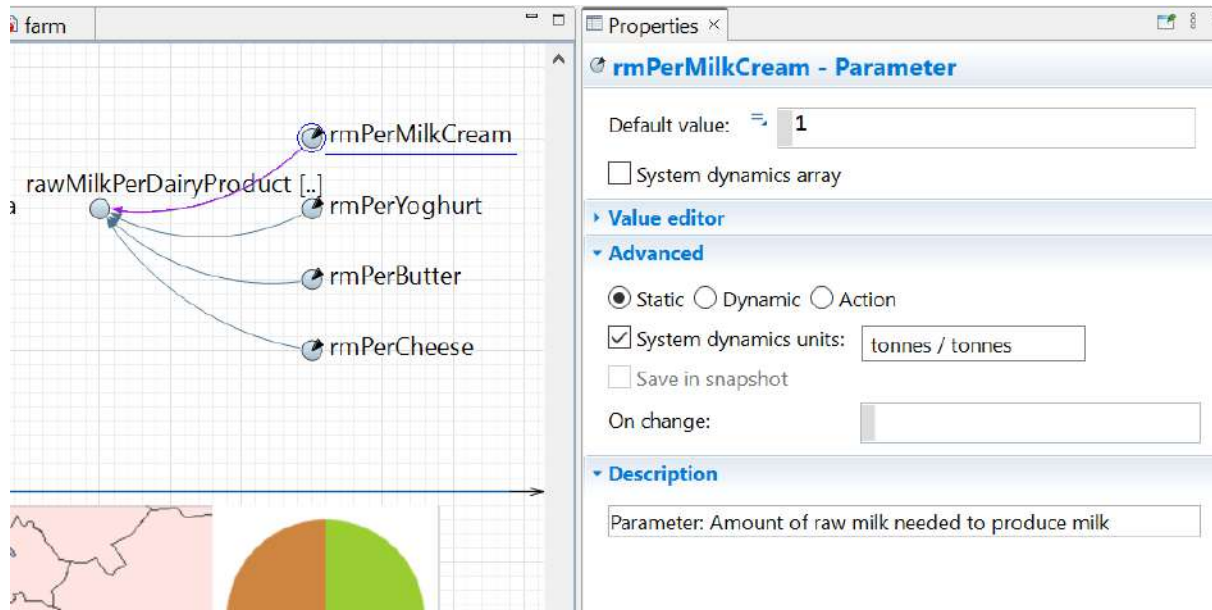


Figure 35: Object properties of parameter *rmPerMilkCream* and arrayed variable *rawMilkPerDairyProduct [..]*

The top section of the properties is the object’s basic information—name, data type (integer, double, string, Boolean, etc), and default value. For a parameter, the value will appear here. For a variable with a formula—called *Dynamic Variable* represented by a simple circle—the equation will in the *default value instead*. Under the *Advanced* section, most objects will have *System dynamics units* box checked and the appropriate unit populated. Most objects will have the *Static* radio button selected, but exogenous parameters that change over time, such as milk yield, will have *Dynamic* button selected instead. If the model object is arrayed by the four dairy products, the *System dynamics array* box on the top will be checked, and on the canvas the object name will have an additional suffix *[..]*, such as *rawMilkPerDairyProduct [..]* as seen in Figure 35. Regardless of the model object under review, the *Description* section should be populated documentation information which will be the same as the ones documented in *Appendix D: System Dynamics Model Equation Documentation* and *Appendix E: AnyLogic Java Code Documentation*.

The agents, either in the *Projects* or the left side of the *Main* model canvas, can be double clicked to investigate their inner System Dynamics components. Model agents are *farms*, *processors*, *kommuner*, and *fylker*. There are two other agents—slaughterhouses and *checkpoints*. They can be disregarded as they are copied over from the SYNAGRI agent-based model but not in active use with the hybrid implementation. The agent *closedFarms* is used only for accounting purposes and can be disregarded.

Double clicking on the *farms* agent, for example, reveals its inner System Dynamics component simulating the cow herd growth structure (Figure 36). There are a few model objects that do not appear on the *Main* canvas—variables, Java functions, and events. Variables are represented by

an orange circle icon with a V in the middle. They effectively act as ghost converters in Stella or shadows in Vensim. In the hybrid model, values must be passed from agent to agent, this can only be achieved by using *variables* as the recipient of the data.

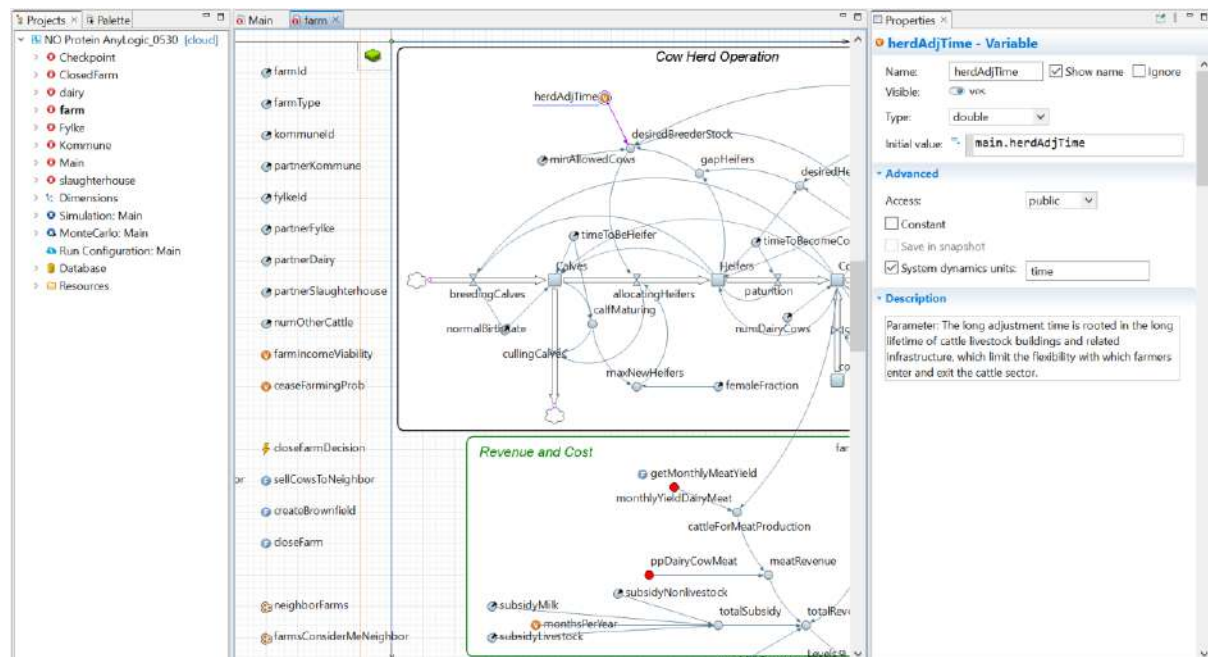






Figure 36: Stock-and-flow structure view of the dairy farm agents upon double clicking

Java functions are represented by a light blue circle with an *F* in the center of the icon. Java functions can essentially serve any purpose as long as the Java syntax is correct. In the hybrid implementation it serves numerous key purposes: retrieving values from other agents, neighbor-searching algorithm on the map, checking own financial viability, farm closure decision procedure, and adding/deleting farm agents. Events are essentially Java functions, but can be triggered at specific intervals or under certain conditions. They are represented by orange lightning bolts. Events are critical because the farm agents' financial self-evaluation must be set to begin after a certain period (5 years default), then annually afterwards, which can all be readily configured. The final object is *Collections*, which is essentially a list of agents that are beneficial to be organized in one place for Java variables to access. Collections allow dairy processor agents to retrieve raw milk production values from all of the farm agents connected to them. Moreover, each farm agents have a small collection of neighbors, one of which may receive its cow herd upon closure.

All of the AnyLogic model objects described are summarized in Table 6.

Table 6: Key AnyLogic model object icons, name, and modeling purpose

Object Icon	Object Name	Model Purpose
	Parameter	Contains exogenous data, which can be static or dynamic, such as inputs of historical data from a table
	Dynamic Variable	Calculates equations on a continual basis every model DT
	Agent	The "actors" in the model. Contains System Dynamic component as well as lives on the map canvas in the Main model object.

	Variable	Mainly acts as “ghost” values between agents, but can be used for any modeling purpose by Java functions
	Function	Java code which can be executed for any purpose necessary throughout the simulation
	Event	Java code which can be deployed at any specific simulation time, frequency, and condition.
	Collection	List of agents which can be modified dynamically.

To run the model simulation, click the down arrow next to the green play button under the menu bar of the window (Figure 37). There will be two options available, *Simulation* and *MonteCarlo*. *Simulation* should be selected to simply view the model results.

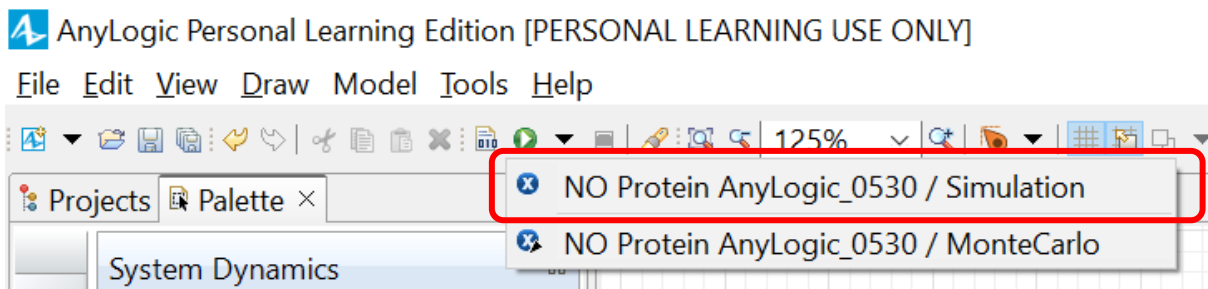


Figure 37: Simulation options available in the model

Once the simulation window opens, it is recommended to click on two objects to run the model effectively (Figure 38). First, the fast forward icon on the bottom of the window allows the simulation to run fast as the computer processor allows. Then, the gear icon on the bottom right can be clicked to open the Developer Panel, which shows informative simulation dialogue such as when a farm must close and when a dairy processor has created a new farm agent.

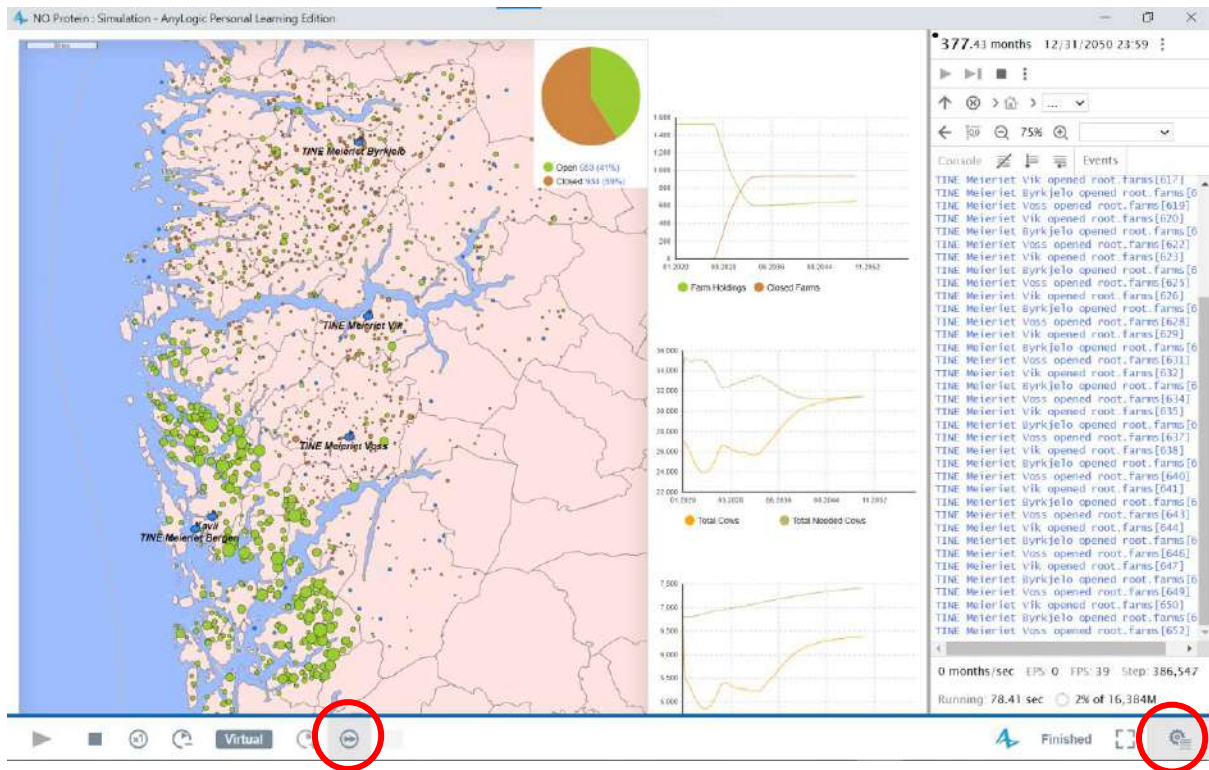


Figure 38: AnyLogic simulation window with virtual run speed and developer panel open (circled red)

The map shows many green dots, representing farm agents, and blue buildings, representing the dairy processors. The size of the green circles represent the relative size of the cow herd of the farm agent. However, it should be noted that the size of the circle does not represent the physical land area of each farm. Toward the latter half of the simulation, many green circles will turn brown, indicating that farm has shut down. There will also be blue circles that appear randomly. These are farm agents which are created by dairy processors that need to boost raw milk procurement. Each icon can be clicked to reveal the System Dynamic structure of the specific agent. Moreover, each stock-and-flow object can be clicked to reveal the running value of that object (Figure 39). To return to the Main canvas, the Home icon above the Developer Panel can be clicked.

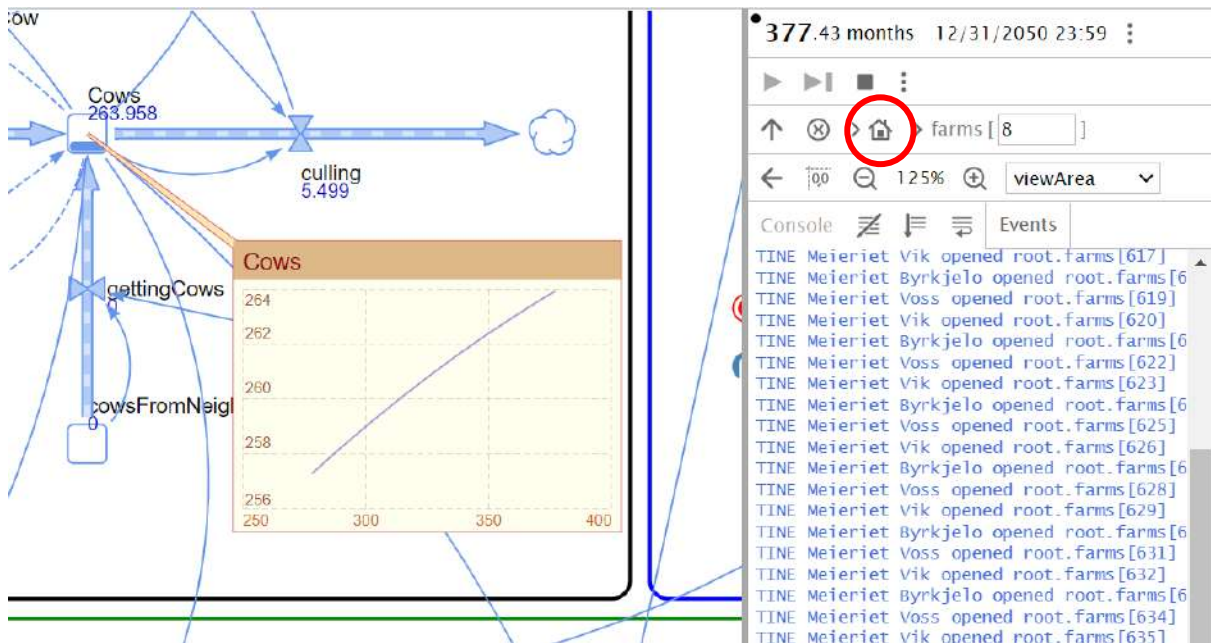


Figure 39: Running simulation value of the cow herd population for farm agent 8, go Main canvas by clicking Home (circled red)

Finally, to the right of the map in the Main canvas, there are several key indicators which can be seen continually updating as the simulation progresses (Figure 40).

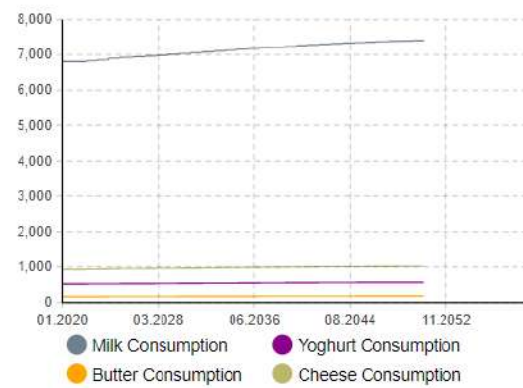
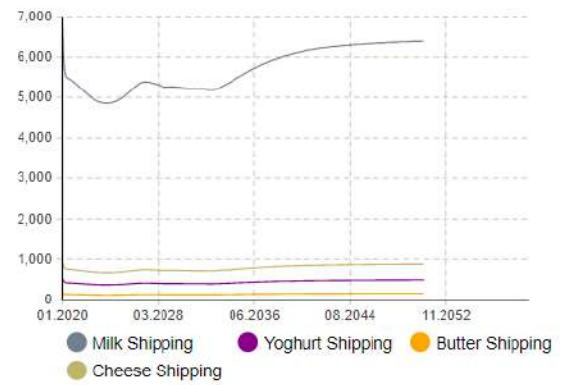
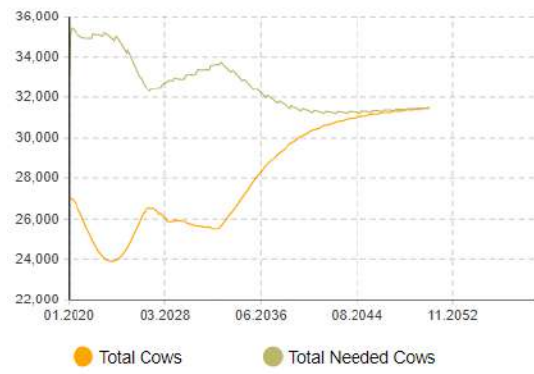
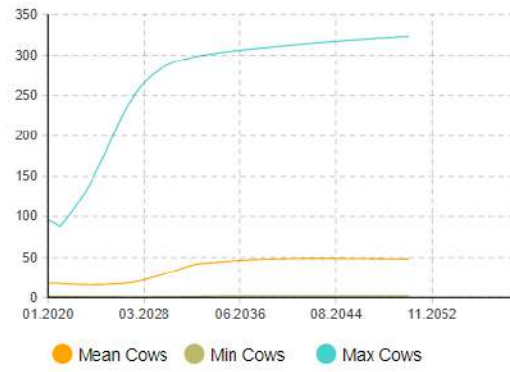


Figure 40: Selected KPIs for the model simulation