ACS6132 Agent-Based Modelling and Multi-Agent Systems

Group 1: Niruthan Puvanenthiran, William Maddex, Joel Nunes, Akinola Dada, Xingyu Chen

January 2019

Executive summary

Presented is the full explanation of the work undertaken by team members in producing the deliverable materials for ACS6132, Agent-Based Modelling and Multi-Agent Systems.

The report firstly provides a background, on the course topics and an analysis which contains descriptions and critiques of the original Artificial Anasazi model proposed by Axtell and Janssen which were emulated to produce material from which the tasks undertaken are derived.

The report goes on to give a written account of the model development, including the various interactions within the group and incites gained from the course material as well as other sources that made it possible to produce the implementation, testing and calibration procedures during the module. The report expands upon how agile project management was implemented by the team throughout the various activities of the module, showing details such as how a Jiri board was used to run sprints.

Conceptual overviews and UML design diagrams of proposed enhancements to the ABM modelling procedures are presented. These could have been performed during the overall project development processes, to improve performance, ease of operation and implementation. Proposals are presented by each individual team member.

The report then proceeds to present a critical analysis of the individually proposed enhancement options. Producing a synthesis which culminates in a single enhancement proposal. Taking the form of a conceptual overview and UML design, based off the current state of the developed model, for a future further enhanced Artificial Anasazi model.

A critical appraisal and reflection of the entire project process is undertaken, taking into consideration the actions and decisions with explained reasoning. An in-depth look into the various challenges faced by the team, both technical and otherwise, on how these challenges affected the various outcomes on this project

Finally, references to any external source material used to validate the rationale behind decisions as well as gleam further understanding of the course content matter are presented.

Table of Contents

List of Figures

List of Tables

1 Benchmark Model

1.1 Background

Identification of the behaviour of complex systems can be difficult to model through pure mathematics or verbal reasoning, especially when it comes to modelling human society, where behaviours are constantly changing. This also makes it difficult to conduct experiments for prehistoric human societies. Archaeologists find it difficult to explain and describe how these societies change through conditional changes. To predict the future of these behaviours, a well-known approach can be used to solve this known as agent-based modelling (ABM) [1]. ABM is a system which is designed through a set of decision-making entities known as agents. The agents are an indivisible social unit that lives on spatial landscapes that can either be an imaginary or a capture of a real-world situation. These agents individually determine their tasks based on their situation and complete the tasks based on a set of predefined rules. Each agent has a set of attributes such as movement capabilities, food requirements, consumptions, storage etc. By manipulating artificial behaviours of the agents, it allows us to look at the socio-cultural history and evaluate the factors to the sociocultural evolution. Simple ABM, which consist of agents and complex patterns, can provide useful information regarding the dynamics of the real-world system. Also, agents can evolve over time and exhibit different/unexpected behavioural changes. More complex ABM modelling include; neural networks, evolutionary algorithms and other techniques which use learning and adaptation methodology[2]. There are many benefits to agent-based modelling, the three main benefits are:

- It enables us to explain the nature of the system
- It is flexible
- It captures developing phenomena

ABM is an unverified source until they have something to compare it against. By plotting the real data to the agent-based model designed, it allows the model to be validated. Having a close fit between the real model and the designed model shows the accuracy of the developed model. If the degree of fit is not adequate, it shows that there is a lack of information about the model or it may require changes within the hyperparameters.

1.2 Artificial Anasazi Model

A prime example of the ABM is the Artificial Anasazi model. This model is based on the population dynamics in the 'Long House Valley, 96km^2 landform on the Navajo Indian Reservation in north-eastern Arizona'. The valley was inhabited by the karyenda Anasazi from circa 1800 B.C to circa A.D 1300. Between the years 7000 and 1000 B.C., the valley was first occupied by the Paleoindian and

Archaic. Maize came in at around 1800 B.C. which started a food-producing economy and the commencing start-up of the Anasazi cultural tradition. The data from the year 200 to 1500 A.D. is rich, consisting of palynology, alluvial geomorphology and dendroclimatology data which enables the reconstruction of the Long House Valley population behaviour. The relationship of Palmer Drought Severity Indices (PDSI) and annual crop yields are records used to create a dynamic landscape of potential maize production. The valley is made up of 7 independent environmental zones. Each zone has its own suitability of inhabitant and the yield of maize production such as Uplands zone which contains bedrock and it is steep which is not suitable for farming. Other zones have different soils quality and water quantity, hence affecting the farmlands. The other zones are General Valley Floor, Uplands Arable, North Valley Floor and Canyon, Midvalley Floor, and Sand Dune zones. The variant topography has given an indication on how the Anasazi lived and farmed. Through paleoenvironmental reconstruction, it can show an accurate representation of how they lived. There are 4 main reasons why the Long House Value is a perfect example of modelling an ABM. The first reason is that it is topographically bounded, meaning that it is easier to replicate on a system. The second reason is due to its rich palynology, paleoenvironmental, alluvial geomorphology, and dendroclimatology data, enabling the reconstruction of the Long House Valley model. Also due to rich data, it enables development of an accurate model. The combined method allows a good benchmark in testing ABM and recreating the dynamics of the Long House Valley. The third reason is because of its detailed regional ethnographies. This allows the agents to have plausible behavioural rules. The fourth reason is because of the intensive archaeological research conducted covering 100% of the area[3].

1.3 Benchmark Model

A study by Robert L Axtell and his colleagues, did an analysis on 'Population growth and collapse in a multiagent model of the Kayenta Anasazi in Long House Valley'. The model tries to reconstruct the Long House Valley between the years 800 and 1350 A.D. using the paleoenvironmental variables and populates the artificial agents. Households are the agents, which is the smallest social unit within the archaeological data set. The demographic characteristics and nutritional requirements of the households are firstly initialised depending on the ethnographic studies from the historic groups and other subsistence agriculturists. Each agent has a set of attributes which include age, size, composition and quantity of maize storage. In addition, each agent has specific rules for behavioural changes depending on the condition set from the data source. Depending on these rules, it determines the planting and dwelling location of each household.

Each household has a set of rules which are stated below:

Each household;

- Fissions at the age of 15.
- Can be defined as one pit house or five surface rooms.
- Consist of 5 individuals.
- Consumes 160Kg per individual.
- Can only store a maximum of 1,600Kg of maize and shall be held for no longer than two years.
- Only use 64% of the total maize yield, the rest is lost to other factors.
- Moves when the maizefield is below a set value of yield
    - Depending on the agricultural location
        - Within the map, the location should be empty
        - The location should have the capability of producing a minimum of 160kg maize per year per individual. The production of the maize depends on the neighbouring maize farm.
        - If multiple locations can meet this specification, then the closest to the current residence location is chosen.
        - If no requirements are met, the agent is removed from the map
    - Depending on the residence location
        - Shall be located within 1 km from the agricultural plot
        - The location should be unfarmed
        - The residence shall not be in a location where yield production is high.
        - If more than one condition is satisfied, the residence shall move to the closest water source
        - If none meets criteria, then the third followed by the first points are relaxed

For each year, the model is designed so that all agents are involved with the agricultural work and move depending on the nutritional requirements. If nutritional requirements are met, they will remain in the same location.

For each year, the household and field location are updated depending on the set of rules. Each agent is randomly assigned with an age and fission rate depending on the sampling from a uniform distribution. Six of the hyperparameters had a uniform distribution. An additional two hyperparameters were included for the maize field production. One of the hyperparameters was the harvest per hectare and the other was the harvest variance. Totalling 8 hyperparameters, making it an eight-dimensional problem. These hyperparameters were calibrated to get a close fit to the historical data. This was done by comparing the number of households to the historical record for

each iteration. These differences between the tested data and historical data are cumulated according to the L^p norm. By changing the 8 variables, the best configuration model was identified. It was done by looking at the best realisation and the best average of each run.

Critiques

Figure 1 shows a plot of the best run of the model and table 1 and 2 shows the optimisation parameter for a single run and an average over 15 runs. The data showed that the overall trend followed the actual historical model. However, in the first 200 years, the model underestimated the population. Around the year 1100, the population peaked which is higher than the actual value. The model also showed that the agents were still alive after the year 1300 even though in the actual model all agents die. This could be due to the model not assuming the sociocultural "push" or "pull" factor. This would have helped to predict the model accurately. Figure 2 shows the clustering of settlements of the Long House Valley. It shows the valley zonal boundaries were reproduced very accurately.

Table Optimized parameter settings based on single "runs" of the model

Parameter/norm	L^1	L^2	L^∞
Minimum death age	26	30	25
Maximum death age	32	39	34
Minimum age, end of fertility	30	28	30
Maximum age, end of fertility	32	30	30
Minimum fission probability	0.125	0.120	0.125
Maximum fission probability	0.129	0.125	0.125
Average harvest	0.60	0.62	0.60
Harvest variance	0.41	0.40	0.40

Table 1 - hyperparameter values of a single run

Table Optimized parameter settings based on the average over 15 runs of the model

Parameter/norm	L^1, L	L^2
Minimum death age	30	25
Maximum death age	36	38
Minimum age, end of fertility	30	30
Maximum age, end of fertility	32	38
Minimum fission probability	0.125	0.125
Maximum fission probability	0.125	0.125
Average harvest	0.6	0.6
Harvest variance	0.4	0.4

Table 2 - hyperparameter values of 15 runs

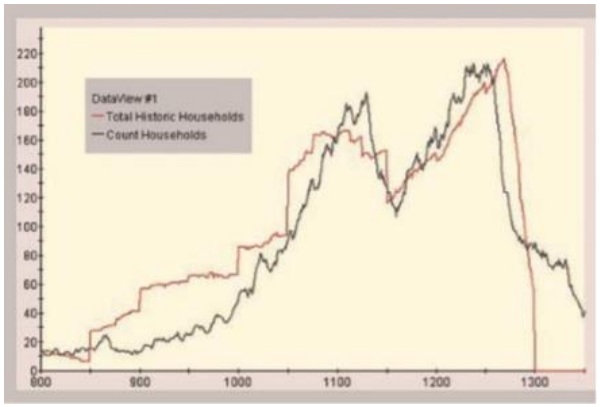

Figure 2 - Representation of the best model vs the real data set

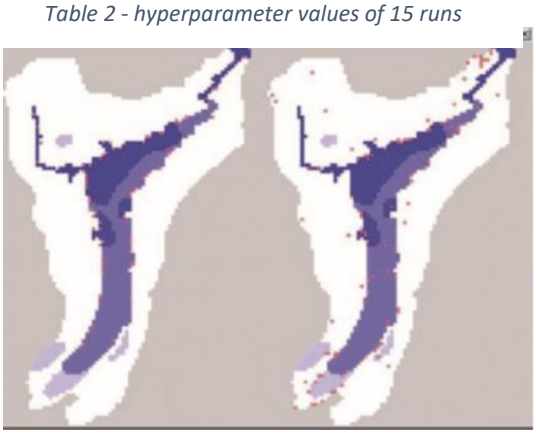

Figure 1- Representation of the simulated and historical settlements patterns in the Long House Valley

1.4 Aims and Objectives

The project aims to replicate the Artificial Anasazi model population dynamics between year 800 to 1300 in the Long House Valley in Arizona using the Repast HPC Libraries in C++. The objectives are:

- Replicate the Artificial Anasazi model using C++ and Repast Libraries.
- Analyse the historical data and calibrate the model designed to meet the real model.
- To propose adaptations and enhancements to the behaviours and networks included in the simulation.

2 Simulation Software Development

In this section, we will discuss the software implementation done by our group to reconstruct the Anasazi Model.

2.1 Model Development

Before any programming took place there was a process of design. It will provide a clear foundation on how to program a large complex system. By carrying out the techniques shown in this section, the collaborative programming could be carried out with higher efficiency.

With the expected output simulation provided by the stakeholders; a set of requirements were produced. The requirements give a platform upon which to build the simulation and to quantify its completion after the implementation phase.
The requirements were as follows:

2.1.1 Top Level Requirements:

- As far as possible, reproduce the structure of Axtell's benchmark model.
- Account for the archaeological data as a model input
- Enable model parameters to be varied without recompilation
- Generate the annual number of households over time as an output, in a way that can be compared to historical data
- Build on the existing functionality of Repast HPC
- (optional) Exploit available parallel computing resources to facilitate execution

2.1.2 Functional Requirements:

1. The Agent shall be able to store and track relevant attributes
    1.1 Shall store the current age for each agent
    1.2 Shall store the preordained age of death for each agent.
    1.3 Shall store the preordained age at which an agent becomes infertile
    1.4 Shall store the agent's maize quantity
2. The simulation shall be able to remove agents when specific conditions are met
    2.1 An agent shall be removed when the death age is reached
    2.2 If their current food production is less than 800kg and a new suitable one cannot be found
3. An agent shall be capable of autonomous decision-making depending parameters

3.1 Shall be capable of deciding whether an agent has enough maize to survive for that year

3.2 Shall be able to determine whether an agent should fission or not

3.3 Shall be able to decide to change location depending on specific criteria

4. The simulation shall work with uniquely identified cells

4.1 Each cell shall have an x and y coordinate

4.2 Each grid cell shall be a size of 100m by 100m

5. There shall be defined environmental conditions within the simulation

5.1 The annual maize yield of the field shall be based on a normal distribution

5.2 The dependency of environmental condition shall depend on the water location in the cells.

6. The maize yield will depend on several factors

6.1 The expected yield shall depend on the existing stocks and requirements for the following year

6.2 It will depend on archaeological data

6.3 Will depend on the zone that it's located in

7. The agent shall change its field location if it has insufficient maize

7.1 The new location shall not consist of an existing field or residence

7.2 The new location shall have the capacity to yield a minimum 800Kg of maize

7.3 If multiple cells meet requirements, then the closest to the resident shall be selected

7.4 If no cells meet condition 7.1 or 7.2 then agents get deleted

8. The agent location for the residence shall depend on the location of the field

8.1 The residence shall be within 1km of the field

8.2 The location shall not be the same location as the field

8.3 The selected location shall not be greater potential yield than the field

8.4 If more than one condition is met, then the cell closest to water shall be chosen

8.5 If no cells meet these conditions, then 8.3 is waived followed by requirement 8.1

9. The Simulation must initialise with the same specified parameters every time it is run

9.1 The simulation shall run from 850 A.D to 1300 A.D

9.2 The simulation shall initialise with 14 agents

10. An agent shall have an association location for each network

10.1 The network shall consist of the household residence location

10.2 The network shall consist of the household field of maize location

10.3 Agents may share their residence with other agents but shall only be permitted to one field per network

11. The simulation will output the number of agents to a file at the end of each tick

2.1.3 Non-Functional Requirements

1. The model shall be developed in C++
2. The designed model shall use the Repast HPC Libraries
3. The model shall use Repast Symphony

Having requirements defined enabled the next part of the design, creating a UML Class diagram. The classes and all their attributes are chosen must show a simulation that is capable of meeting all the set requirements.

2.1.4 UML Diagram

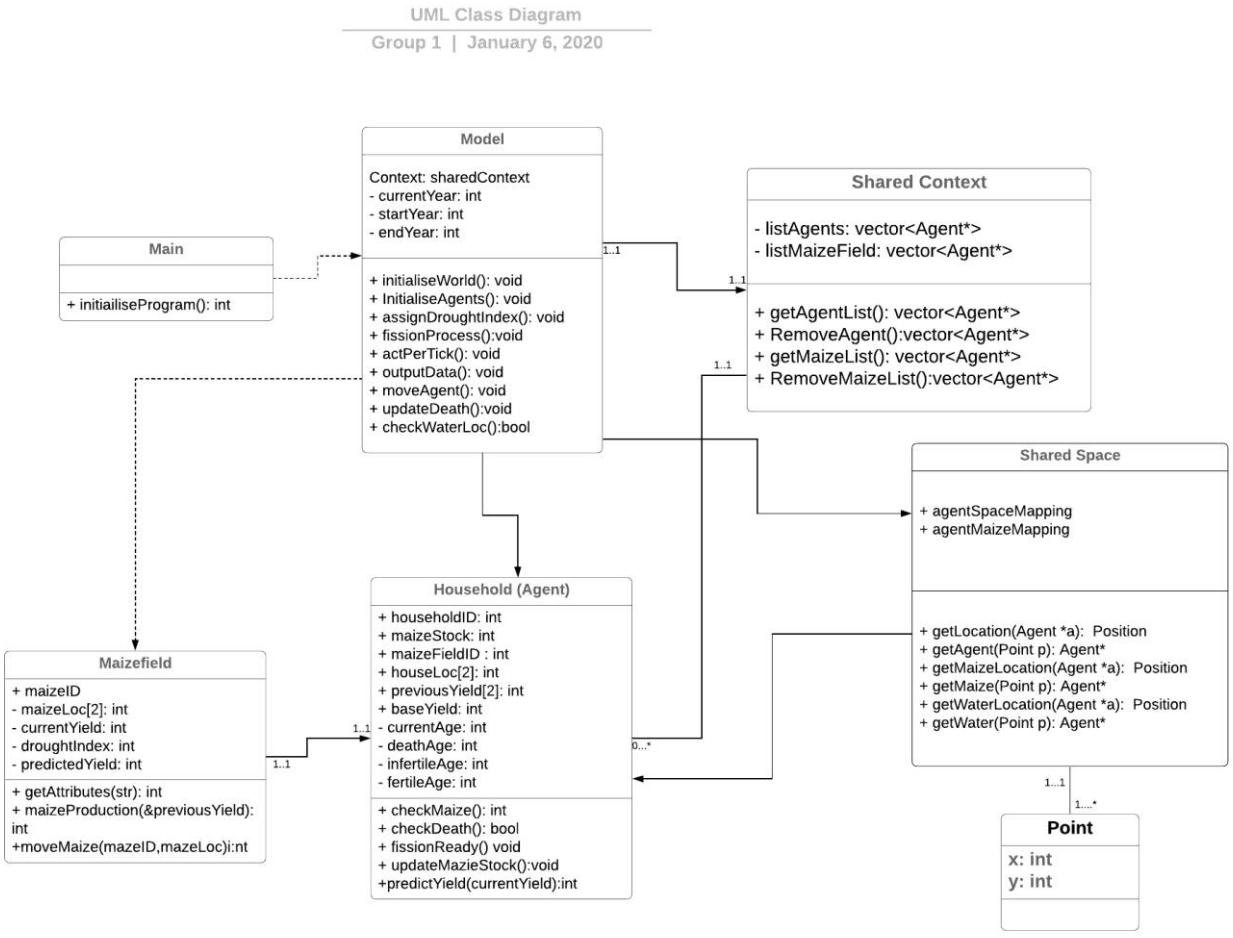

Figure 3 - UML class diagram for planning the simulation structure

Figure 3 shows the UML class diagram with all the important variables and functions. This is not a definitive list of all variables and functions used within the simulation but all that serve a significant purpose in meeting the requirements.

Class Name	Description
Household	This class will be the agents within the simulation. This class is responsible for all agents and how they behave within the simulation. Each object will be a different agent.
MaizeField	Each object represents a maizefield. For every agent, there will be one maizefield linked to it. This class will handle processes related to a maizefield including calculating predicted maize.
Model	Control the overall process of the simulation and manage what happens at run time. This will be the main class in the simulation with processes passing through here.
Shared Mapping	A Repast library that creates a grid on which to place the model on. Agents will be placed on this grid allowing us to use functionality that makes implementation easier.
Shared Context	A built-in Repast library to store agents in the simulation using a unique ID for each. The built-in functions give benefits that make using ABM much easier to implement.

Table 3 - Class description of UML Class Diagram

Each class's attributes and methods are explained in Table 3.

Household Class Methods (Agent class)	Definitions	Requirements Met
predictYield(): int	calculate Predicted yield	3.1, 6.1
checkDeath(): bool	Decide whether currentAge == deathAge, if so returns true, else false	2.1,
fissionReady() void	Query whether agent is ready to fission, if so returns true, else false	3.2
updateMazieStock():void	Updates the agent amount of maize each year to match the brief	1.4
Attributes		
householdID: int	Uniquely identifies a household, the ID will match its corresponding maize field	2, 3,
maizeStock[2]: int	The amount of maize stock saved by a household	2.3, 3.1
baseYield: int	How much maize a field needs to consume	3.1,
currentAge: int	The current age of the household	2.1,
deathAge: int	Preordained age the agents will die at. Given once initialised.	2.1,
infertileAge: int	The age an agent becomes infertile. Given once initialised.	1.3

Model Class Methods	Definitions	Requirements Met
initialiseWorld(): void	Get initial configuration data for the world with an initial set of 14 agents and start year of 800. Creates a grid made of cells.	4, 10
createAgents():void	Create agent and add it to a list using Repast functionality Shared context.	1
actPerTick(): void	Controls what happens each tick. Will call other main functions to run required processes. This function will also cycle through each agent.	9
outputDatafile(): void	Outputs the current year and number of households	11
fissionProcess(): void	Performs the process of fission if an agent is going to perform fission this year.	1.3
assignDroughtIndex(): void	Assigns drought index value to a maize field by using archaeological data of zones and PDSI.	5, 6,
moveAgent(): void	Change agent location if required. Follows the stated method defined within requirements.	3.3
updateDeath():void	Query whether an agent has reached its death age and remove from shared context if it has.	2.1, 1.1
Attributes		
Context: shared Context	Repast functionality, that stores all the agents	1, 10
currentYear: int	The year the simulation is currently on	9.1
startYear: int	The year to start the simulation on, will be 800	9.1
endYear: int	When the simulation should end, will be 1350	9.1

MaizeField Class Methods	Definitions	Requirements Met
maizeProduction(&previousYield): int	Update maize yield for each year based on the previous year's yield and drought Indtex derive predicted Yield	5, 6,
moveMaize(maizeID,maizeLoc):int	Change maizefield location to that specified	7, 8, 10
Attributes		
maizeID; int	Uniquely identifies the maize field and matches to its corresponding household.	10.2,
currentYield: int	Calculates the production of maize	2.2,
droughtIndex: int	The PDSI value for the maize field, used to calculate maize production	5, 6, 7

Table 4 - Description of method/attributes on UML diagram

The proposed design for this model has a specific method of how to implement maizefield, this is shown in table 4. The Repast functionality shared context is mainly designed for agents in the model, however, in addition, this will be used for maize fields. They will use the Repast functions just as the agents do but for the purpose of the model, they are not agents.

2.2 Model Implementation

Implementation Workflow:

Figure 4 - Shows the steps of implementation phase

During the implementation phase, a specific order of programming was used to develop the model. The steps shown in figure 4 above enabled the most efficient way to program as a group. The initialisation of the model and all other parts required to run a general program were written. Loading information from the model.props file was also implemented at this stage.

Each class was then written with a header file and a .cpp file. Only the important parts shown and discussed within the UML diagram section were written. This simulation was run to check for general syntax errors, but the actual processes were not tested at this stage. The methods themselves resembled more of a sudo-code standard.

Agent interactions were next to be implemented in a step process allowing us to test each process singularly. The interactions programmed were as follows:

- Fourteen agents initialised at the start of the simulation
- Agents getting deleted when their death age has been met
- Agents being created if they are to perform fission that tick
- Maize production being calculated
- Maize stock being updated
- Check for whether an agent should be moved
- Perform the moving sequence

The complexities of each of these interactions will be discussed later in this section.

Next to be implemented was the simulation flow. This means what happens in what order every tick. With the schedule runner implemented, it calls a function called “doPerTick” each tick of the simulation. This is part of the Repast functionality. Within every tick, the program will run through every agent alive on the map.

Steps:

1. Checks whether the death age of agent has been reached
2. Checks whether the agent will fission this turn or not
3. Updates maize stock
4. Checks whether agents need to move
5. Updates counter values such as the current year
6. Outputs to file the number of agents on the map that year

Simulation Complexities

At the start of the simulation, fourteen agents were initialised with various randomised parameters. Using different random variables shown in table 5. A model.props file is used to feed in values to use for the range the random number needs to be, discussed in the calibration section.

Parameter	Description	Distribution
DeathAge	Randomized between two values fed in from the model.props file. This decides when the agent will die.	Uniform
InfertileAge	Randomized between two values fed in from the model.props file. This decides when the agent will no longer be able to fission.	Uniform
MaizeStock	The randomised value between 1000-1800. The simulation has just begun so need a stock of maize.	Uniform
CurrentAge	The age of the agent has to be randomized to produce a realistic start to the simulation.	Uniform

Table 5 - Parameter given to agents when first created

Each simulation run; a function is called in the household class which returns true if that agent has reached its death age. The model class uses this as a flag to decide whether to delete the agent or not. To delete an agent the shared context (Repast functionality) is called to remove it from the list of stored agents.

```
context.removeAgent((*it) -> getId());
Mcontext.removeAgent((*Mit) -> getId());
```

Code snippet 1 – Code used to delete an agent

The process of fission is how the simulation creates new agents. Criteria must be met for fission. An agent must be between 16 and their own defined infertile age. If that is met there is a set probability, discussed within calibration, that fission will occur. If the probability is of fission is set as 0.125 the model class will call a method within the household class that checks the agents' ages then calculates a random number between 0->1/0.125 creating a one in 8 chance of fission occurring. A new agent is created by using the Repast functionality, assigned random parameters in the same way discussed before. The new agents, however, are given a current age of 0 and take 33% of the parent agents maize stock.

Each agent is stored and tracks an amount of maize stock. Maize will only last for a maximum of two years and will be deleted after that is done. This is implemented by using an array of size two to track each years' worth of maize and write over the expired maize. The maize is reduced by 800 each year to account for usage.

Each year/tick every maize fields creates an amount of maize. Quantity of maize created is reliant on several parameters with a small amount of random probability. Shown in the equations below is how this is calculated.

$$Base\_Yield = Y * Q * Ha$$
$$Harvest = Base\_Yield * (1 + sig)$$

Equation 1 - Calculate the harvest for that year

Where Y is the PDSI value from the archaeological data. Ha is the Harvest adjustment value which has a random uniform distribution between zero and the value passed in from model.props. Q is the quality of soil and is a random number, with uniform distribution, between zero and the number passed in from the file. Q will only change every year and not for each specific cell.

When loading archaeological data from a file to find the zone and PDSI values of a cell, they are saved in integer arrays at the start of the simulation. The arrays are stored within the model class and can be called whenever required. Water sources worked in the same saving the info within an array, then a function was created that passes in a cell location and returns true if there is a water source and false if there isn’t.

The moving process is the most complex of processes within the simulation. Firstly, to decide whether an agent should move from its current location there must be 800kg of maize to use. This is between the amount of maize that will be produced that year and the current stock the agent currently has. If a move process is required, the agent starts by finding a suitable place for the maize field then moves (household). The requirements for the process are outlined at http://jasss.soc.surrey.ac.uk/12/4/13.html and was implemented doing the following:

- To make sure the cell has no maize field/agent in its shared mapping function was used.
- Checking for a water source was done using the function previously described.
- The function that calculates the predicted yield was set up so that it can be called for any cell.
- By finding the observed cells PDSI value then passing this into the function the maize production for any cell can be calculated.
- A distance of 1km is ensured by using the Pythagoras of the number of cells across and up. If the value is less than 10 (cells) then this is within a 1km distance.
- When no condition can be met agents are taken from the valley using the Shared context built-in functions.

2.3 Model Testing

Three separate methods of testing were used throughout the implementation phase. With a scrum process being used this meant that low-level testing took place throughout the project. Component 3 of the module meant specific tests were performed to meet the criteria set. A full test plan was used to ensure full completion of the simulation before calibration began.

At each scrum stretch, testing was used to prove the completion of set tasks. An individual set a task, which is set to be completed by individuals within to group. This shows a low-level testing throughout, reducing the risk of large errors.

A small test plan was developed around the component 3. The marking criteria tested the main process of the simulation with each one broken down into individual tests. These tests were treated very much like black-box testing to prove the overall simulation achieves its task without any specific being checked. This is shown in table 6.

No.	Test	Method	Proof
1	**Instantiate an agent and update its properties**	Have agents created with relevant attributes given when initialised? Each year update the current age of each agent.	Each year print a list of agents with their ID and current age together. Only run the simulation for 5 years for this test.
2	**Instantiate the network and update its properties**	Create the grid with the required size. Place the agents and water source on the grid.	Print the grid to screen with agents in their designated location. With agents marked as an 'x' and maize fields as an 'o'.
3	**Read archaeological data as input and propagate this to the network**	Load the water sources from file saving in an array. Add the location of the water sources to the map.	Same as above, print map with water sources marked by a 'w'
4	**Reduce yields to below the minimum agent requirements and demonstrate the intended effect on population**	Maize fields needed to be added and linked to agents. Can use a forced value of making yields not meet requirements after year 10. Move process must be developed with the ability to remove agents.	Print to screen the number of agents at the end of each year and observe the response after year 10.
5	**Read a new 'death age' parameter at run-time and demonstrate the intended effect on agent life expectancy**	Change the death age parameter in the model.props file to a small number such as 20.	Run the simulation again with this new death age. Observe how the population differs from the first run. Use a new file name, for this run, so the two files can be compared.
6	**Generate an output file recording the annual number of households over time, which is accurate with respect to the simulation**	Must have written to a file completed. Use a counter to count the number of agents within the simulation. Output the year and number of agents to a file.	Print to screen the number of agents each year and compare this to the values on the file.

Table 6 - Tests to match the first deliverable

A full test plan was developed using the previous test plan as a base. The aim of this test plan was to make sure the simulation meets the requirements set out at the start of the project. Testing at this stage will make sure the simulation is ready for calibration and resources will not be wasted. This testing stage is styled for white box testing where every specific process in the simulation is checked. Table 7 shown below is an example of how the test plan was developed and used.

Requirement ID	Requirement	Test Method	Expected Result
1.1	Shall store the current age for each agent	Print every agent and its current age each year. Add a specific agent with a current age of 25, to show the correct value is stored.	Each agent should be printed with a corresponding current age to match that agent. Each agents age should increase by one each year.
2.1	An agent shall be removed when the death age is reached	Print the agents' current age next to its death age, each year. Add an extra print statement that declares when an agent has been deleted with its ID shown.	The ID of the agent will not show on the next lot of printed agents. Every agent where death age is equal to the current age should be removed
2.2	If their current food production is less than 800kg and a new suitable one cannot be found	After an agent's food storage has been updated print the current amount of maize stock. Print when an agent is trying to move and again if it can't find a suitable source.	When maize stock is less than 800kg a move should be attempted. Later in the simulation agents should be unable to find a new location and be removed from the simulation

Table 7 - Example of the final test plan used

2.4 Calibration Activities

Calibration phase of the project was all about research and deciding the best methods to undertake. There are several different methods of calibration that had to be discussed undertaking the best one.

The Approximate Bayesian Computation (ABC) method was implemented for the calibration phase. There are a few variations of ABC within ABM, but they all have an overriding common functionality. Each method will sample from a pre-defined range of model inputs known as

priors then run the simulation comparing the new model outputs with another desired data set. Three possible methods that can be used based on the ABC approach are rejection, sequential and MCMC. The advantages and disadvantages of each method are listed in table 8.

Methods	Advantage	Disadvantages
Rejection method	Simpler for implementation	• High computational complexity • very low estimated likelihood • Cannot run large iterations
MCMC	Does not require to sample from previous distribution; Efficient	Can get stuck in parameter space
Sequential	convergence to posterior distribution; Efficient low sampling size and less likely to be stuck in parameter space	multiple fine-tuning options making it complex

Table 8 - Advantages and disadvantages of each type of ABC method [4]

When looking at resources available at this stage of the project, the rejection was the best option given external circumstances such as time pressure. Rejection method gives us ease of implementation allowing more time for physically running the calibration, therefore giving us better final results.

The rejection method works by defining the number of samples to be taken and a range of values to be used within the simulation. The simulation runs with the sampled values, then by comparing the results to an expected data set will save the best results.

```
Cal <-ABC_rejection(model=binary_model("model.sh") , prior=anasazi_prior, nb_simul=100,
prior_test="(X2>X1) && (X4>X3)", summary_stat_target=c(target_data), tol=0.1,
progress_bar = TRUE)
```

Code snippet 2 – The ABC function used for the calibration rejection method

Code is shown above shows how the function used to implement the rejection method. Priors are the range of inputs to the simulation; tolerance is a ratio of how many of the best samples to save and the first part runs a bash file to allow the simulation to run. The bash file requires an input and an output file, the input file created will be the parameters written to the simulation and the output file will be the outputs from the simulation.

The priors used were narrowed down to make calibration as effective as possible. The first method to decrease the priors was by trailing the outermost values and discovering the values created undesirable outcomes. Some values were obvious such as death age and infertile age must be over 16 (the fertile age of agents), otherwise, the model wouldn't work. The priors are also given tests to ensure the values are usable such as minimum death age is not higher than maximum death age.

The final priors and the distribution used in calibration is shown in table 9:

Parameter	Range	Prior distribution
Min Death Age	(20,35)	Uniform distribution
Max Death Age	(30,50)	Uniform distribution
Min Infertile Age	(20,35)	Uniform distribution
Max Infertile Age	(25,40)	Uniform distribution
Fission Probability	(0.05,0.2)	Uniform distribution
Harvest variance	(0.5,1.2)	Uniform distribution
Harvest Adjustment Level	(0,0.5)	Uniform distribution

Table 9 - Priors used within the calibration

In order to improve our estimation procedure, instead of accepting the m runs that minimizes $\rho(X^i, D)$ with (θ^i, X^i) which forms a sample of the approximate posterior, where D is considered to be a realisation of the model run, θ being the vector of the model parameters drawn from the prior distribution, $\pi(\theta)$, and output X , by replacing it with a more probabilistic approach, where each (θ^i, X^i) pair is accepted with a probability $\pi_c(D - X^i)$, which is the probability density function of ε evaluated at $D - X^i$ and c is a constant as the maximum of $\pi_\epsilon(D - X^i)$. [8]

To do this, we first find $\hat{X}_j$, the model output X^i that minimizes $\rho(X^i, D)$. When all data are of the same type, $\rho(X^i, D)$ is the sum of all Euclidean distances between $\hat{X}_j$ and *D*.

To estimate the standard deviation λ of this normal distribution, we take the standard deviation $\hat{\lambda}$ of all the $\hat{X}_j - D_j$ values that are of the same type. Further, accept (θ^i, X^i) with a probability $\pi_{\chi^2}(s)s^{1-\frac{l}{2}}/c$ where $= \sum_{j=1}^{l}(X_j^i - D_j/\hat{\lambda}_{\tau(j)})^2$ i.e, the density of a chi-squared distribution with l degrees of freedom evaluated at s, the sum of squared errors multiplied by a Jacobian term $s^{1-\frac{l}{2}}$ and *c* is equal to the maximum acceptance probability across all runs[9].

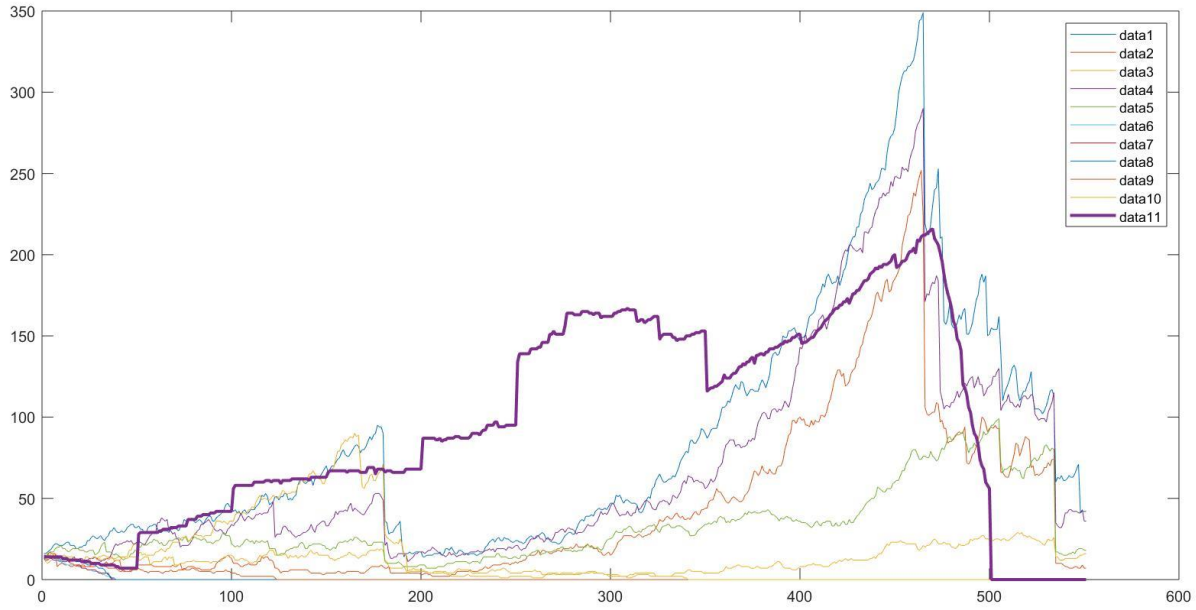

Figure 5 - Calibration with 1000 runs using a tolerance of 0.1

Figure 5, shown above, presents the results observed after calibration was performed. A value of 1000 runs and an acceptance tolerance of 0.1 gives the best 10 results. All show a very similar trend which means the model is repeatable and therefore can be seen as desirable. 1000 runs is a relatively small number when compared to the amount of combinations that could be used. Given additional time this would need to be run for a largest feasible number.

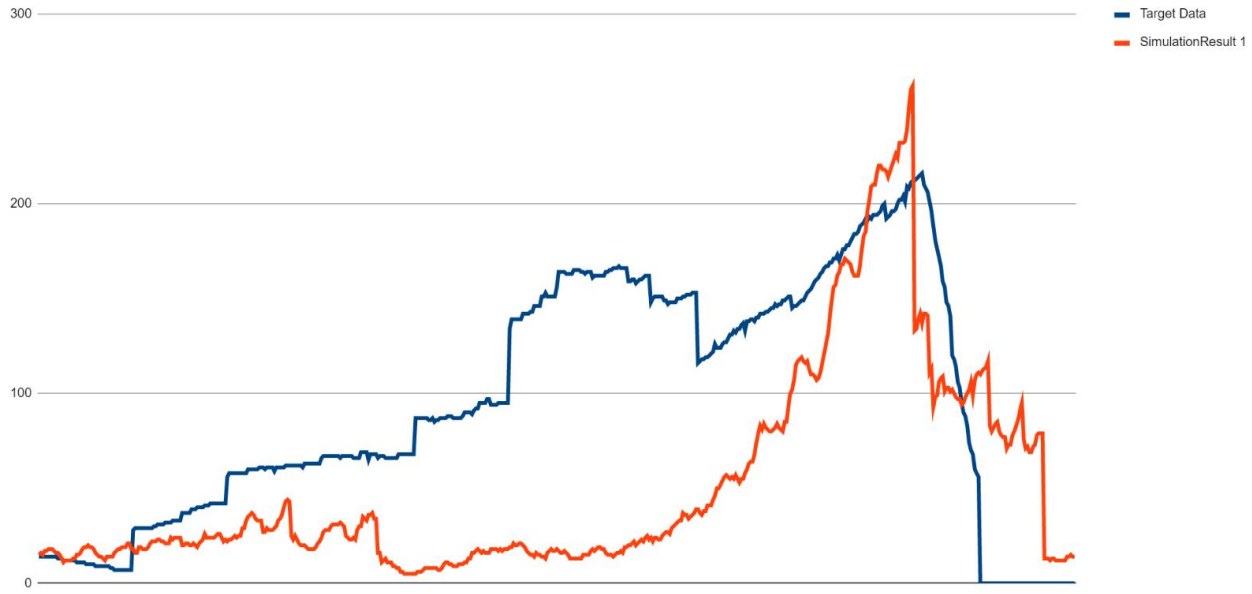

Figure 6 - Graph of the best result from calibration

Figure 6 shows the results of the best result during our calibration process. The results obtained matched the overall trend of the target data set. A peak of over 200 agents was achieved within a close time proximity. The sudden decrease in agents within the simulation matched that of the target data, however the decline did not reach zero as with the target data.

The result above was achieved with prior values of:

- Death age between [27,34]
- Infertility age [24,33]
- Fission probability 0.125
- Harvest Variance 0.952 (3 d.p.)
- Harvest Adjustment 0.211 (3 d.p.)

However, when the model is run again with these same values, a different result is achieved each time. All the random probability within the simulation creates a model that is not repeatable. One possible way to improve this would be to use different distribution such as normal instead of uniform. For priors such as death age the real-life relationships would be more of a normal distribution as people al tend to have a similar death age. This is evident even when looking at today's society, shown in a government survey [5] looking at figure 5 (of the referenced source) that trend is shown when looking at the number of deaths by age.

3 Agile Project Management

Agile Project Management was implemented during this project, which helped in planning and guiding the various phases of development.

The project was completed in several sections. Each section was reviewed and critiqued by every member of the team. The insights gained from the critique of every section were used to determine the next step that should be taken in the project.

The main benefit of using Agile Project Management in the project was its ability to respond to issues as they arise. Making the necessary changes to the project at the right time during the software development or the calibration process saved resources and helped to deliver a successful project on time. Agile methodology allows for a constant cycle of design, delivery and testing at a low-level scale. A very desirable trait when working on a simulated model such as this.

Jira, which was the software tool used for Agile Project Management, broke down the project into small objectives of work. Then completed in sessions that ran consistently throughout the project. These sessions are called sprints which ran for a duration of anywhere between one to four weeks.

The scrum approach was utilised in this project. It is a powerful framework for implementing agile processes in software development. It calls for a scrum master who helps set priorities and guide the project through to completion by creating a backlog. The team then selected items from the backlog that were to be completed in the sprint and created a sprint backlog that consisted of the features and tasks that were agreed on during the meeting. The Scrum master held brief meetings to get updates on the progress and sprint reviews were conducted at the end of each sprint. The task board shown in figure 7 was used by the team to track the progress of the tasks for each sprint.

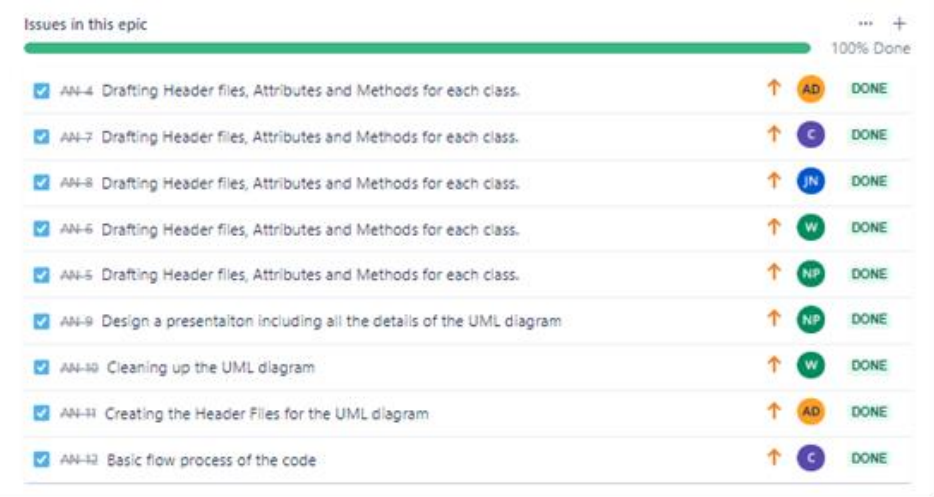

Figure 7- Jira board used to create the first sprint

The project utilised three sprints which ran over 8 weeks. The first sprint was concerned with tasks which were involved in the UML diagram design. The task of drafting header files, attributes and methods for each class was assigned to everyone on the team. It ran for 10 days and a meeting was arranged at the end of this period to draft the final UML diagram.

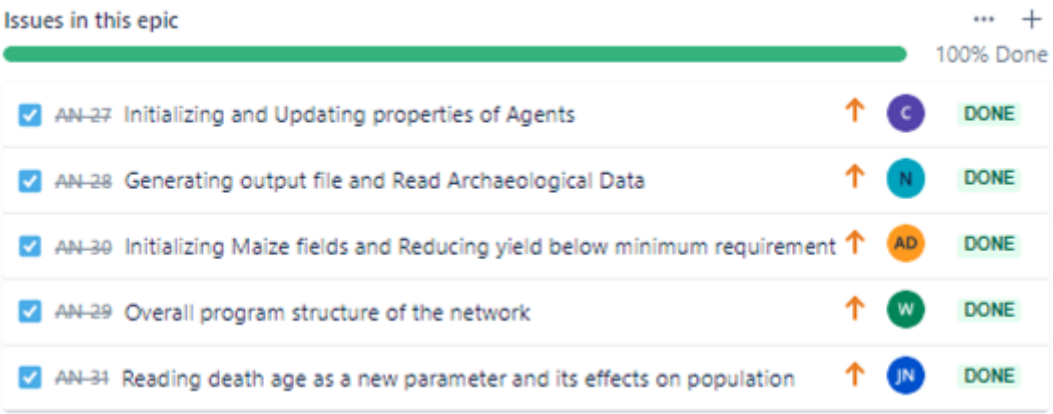

Figure 8- Jira board used to create the second sprint

The second sprint included tasks which were used for the software implementation required for simulation of the Anasazi Model. Each requirement for the simulation was assigned as a task to each team member. The tasks included initialising and updating the properties of agents, designing the overall program structure for the network, generating the output file, reading archaeological data, maize fields initialisation, reading death age as a parameter and demonstrating its effects on the population. The sprint ran for a period of two weeks.

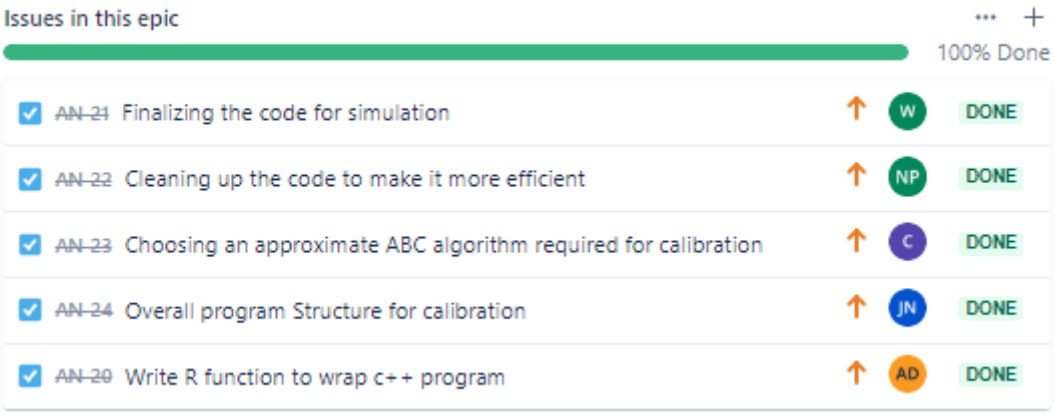

Figure 9 - Jira board used to create the third sprint

The third sprint consisted of tasks which were responsible for calibrating the model. The tasks involved were about deciding and finalising on an appropriate ABC algorithm. Also, choosing priors, a likelihood function and the overall program structure required for calibration. These tasks were common tasks assigned to three members of the team. Since our team was behind schedule in the software implementation phase, the third sprint also included tasks designated to two members of the team. Increasing the run time speed of the model. This sprint ran for 4 weeks and regular meetings were held during this period to check the progress of each component of the sprint.

4 Options for an Enhanced Model

4.1 Options by Xingyu Chen (190246633): Flexible Fission Process

In our model, we suppose each household has 5 people and they have the same age. When they reached min fertility age, there is a chance that the fission process will happen, and a new household with 5 people appeared in a random location of this village.

The main problems are:

(1) It is impossible that people have the same age.

(2) A woman cannot always have 5 babies in one time.

Because of these problems, sometimes the simulation will crash at the beginning or at the end when the number of households reduce to 0 in a short time or increase to a large number later. The reason is that when a household disappears, 5 people died; when a new household is created, 5 people born. The number of people changes sharply, so a slight change in parameters will lead to a huge difference.

It is not taken into consideration to change this at first because it will lead to a large amount of work, as well as slowing the simulation speed. But now these problems: could be solved:

(1) When initializing the households, each member of the household has his age. When people reach min fission age, there is a chance that offspring is born. The number of babies obeys exponential distribution. Max number of offspring is set to 3.

(2) The maximum number of people in a household is 5. If there are no people in a household, a household disappear. If there are more than 5 people in a household, it will split into 2 households. Each household will have 3 members.

(3) When a household splits, the storage of maize will also be divided into 2 parts according to the number of people in a household.

In this way, chances that a household dies in a short time at the beginning or number of households increases dramatically will reduce a lot.

No new function is needed in the UML, but several changes in functions should be applied in the model:

(1) Each people have his ID. The moving decision is still made by a household, but the fission process will be checked by each people.

(2) Each people have his age, so check death and fission should be done separately.

(3) The range of fission probability should be adjusted. It should be higher than the original model, as now only one offspring will be born at one time.

Part of UML for the enhanced model is shown in Figure 10.

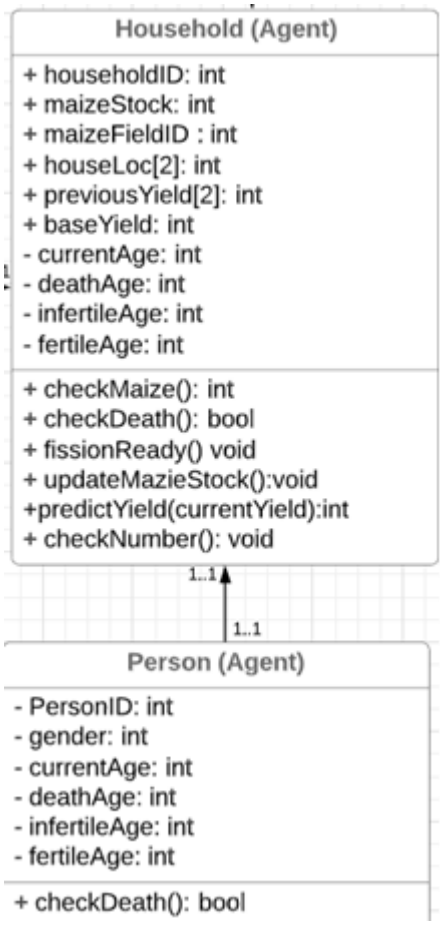

Figure 10- UML Class Diagram of the household and person

4.2 Options by Joel Nunes: Better farm plot selection (Reg: 190234179)

In contrast to the original model, where the farm plot selection was selected based on the closeness, selection of higher quality land and social presence are considered to be more important. In fact, the performance and accuracy could improve when agents select farm plots further away from their initial farm plot.

The original farm plot selection method is solely influenced by the minimization of distance over the entire set of available plots in the valley. This method is not entirely human-like and could have been influenced by a factor other than the distance. We include 4 different factors and four different social connectivity factors that may have driven the plot selection by the society: social presence near the potential farmland (F_{soc}), homophily by age (H_{age}), homophily by agricultural success (H_{agri}) and inter-zone migration with the social configurations: full information of the valley (S_{all}), information provided by family immediate family members (S_{Fam}), information provided by the most productive households or the nearest neighbours (S_{neigh}): where households only consider potential farm plots known to the best performing households.

Taking all these parameters into consideration, in particular, the selection of higher quality land that had a higher social presence could result in a behaviour similar to the Anasazi model rather than random initialization of the parameters. The social component is expressed through four mutually connected configurations through which the agent could receive information on a subset of potential farm plots s, out of the information on the entire set of potential farm plots in the valley (S_{all}). The information is then passed through a function, which considers both the initial state of the household and the conditions of the farm plot and its surroundings to determine the next farm plot.

The four social/economic factors included two types of homophily (the tendency for societies with similar traits to mix with one another), need of social presence and one for leaving/migrating. Each social parameter returns a score representing the desirability of each farm plot.

Homophily by age (H_{age}): represents the preference a household selects a farm plot near other households that are of a similar age, where the age is measured as the number of simulations steps a particular household has survived since splitting forms its parent.

Homophily by agricultural productivity (Hagri): where households select farm plots near households with a similar maize stock.

Social Presence (F_{soc}): where agents score potential farm plots with a higher number of households than those in isolation from other households.
Leaving/Migrating (F_{mig}): where agents score potential farm plots that are in a complete zone that the current one with a full score while plots in the same zone receive a zero score.

From the current model, it most likely that the households select the next potential farm with higher soil quality (PDSI value). This could be possible if the households had good knowledge of the potential land in the valley (S_{all}). Social presence and information on arable land known to neighbouring households are found to be one of the most important social configurations that are responsible for modelling the Anasazi model behaviour. Further, instead of choosing closer potential farm plots, choosing farm plots that were further or in a complete region was found to be the more desirable behaviour.

The above-defined 8 parameters will be defined in the Household Class and additional functions will be needed to be defined in the Household class to calculate: H_{age}, H_{agri}, F_{soc}, and F_{mig}.

4.3 Option by William Maddex (160141267): More Detailed Agents

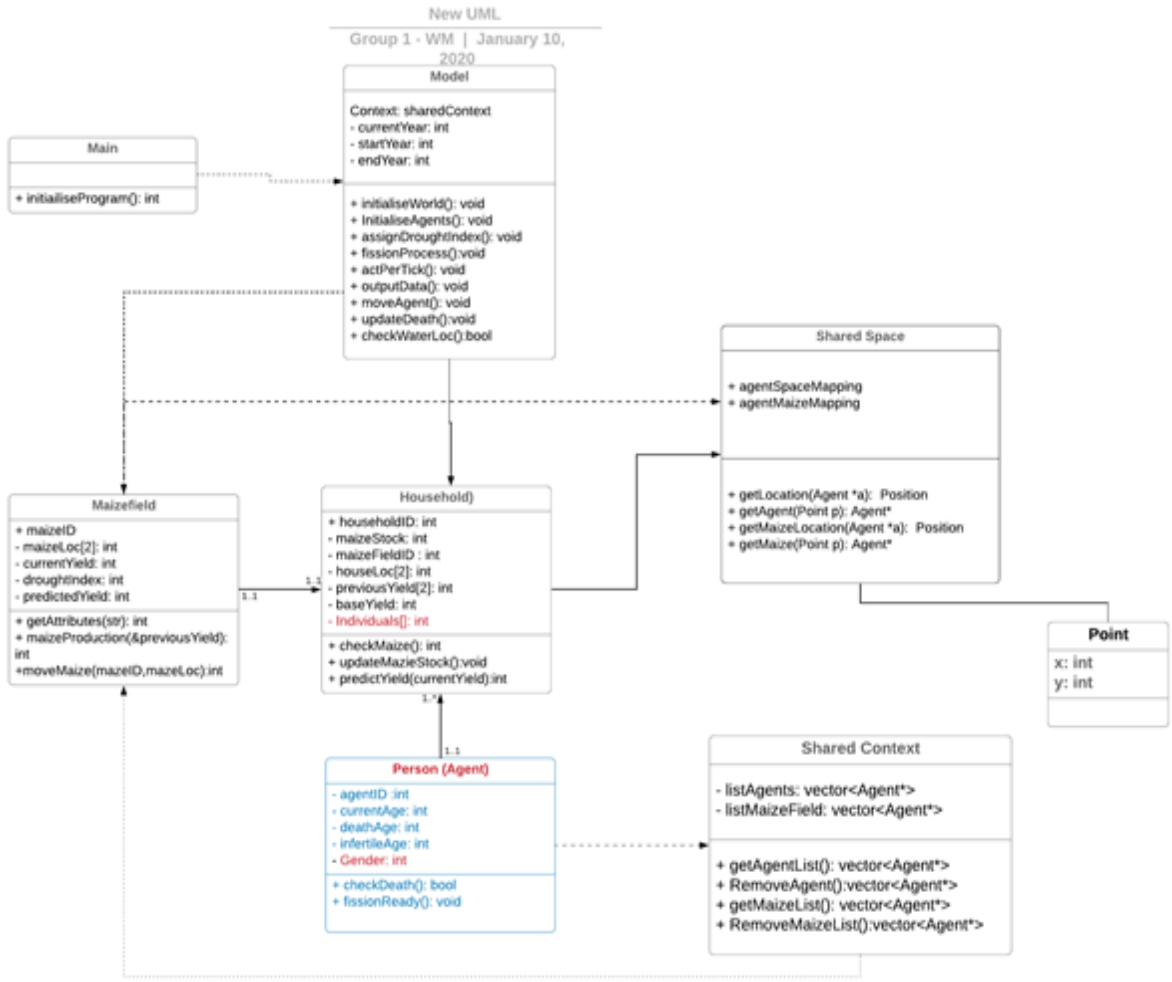

Figure 11 - New UML incorporating model improvements

Improvements can be made to the model by making the agents within the model more specific. Currently, the agents are set to a household, with the contents of a household creating no influence.

The proposal is to make agents specific people instead of a whole household. Households will still exist but will act to store a set amount of agents in. The UML for the new system would look like the following:

Highlighted in blue are attributes/methods that existed before and in red are ones new to the UML.
The Household class would still be responsible for controlling the maize stock and be linked in a 1…1 multiplicity with a maize field. The agents will now be individual people who will act as their own class within the simulation. All variables and functions that are related to a person moved from the Household class to the person (agent) class. Everything related to a household, such as maize

stock, has remained in the same class. The more specific use of agents brings extra functions and variables that enable a more realistic model to be developed. The shared context will be used to keep track of the agents, but shared mapping will be used to place households and maizefields on the map. With this improvement more complex processes can be implemented, developing a more accurate model that would replicate the real data with increased accuracy. With individuals having a death age the population curve would be more fluent.

Each household would use a different amount of maize depending on the residences. Leading to a dynamic where the maize field's yield matches the household, not just a network where every household needs 800kg, as within the current model.

Creating more specifically designed agents would now be possible such as using male and female genders. Male and female genders could show different life expectancies, they would also interact with the world differently.

The fission process would be more complex by using this method, and it would help replicate more realistic behaviour. The simplicity of the fission process is a possible drawback of the current model. When looking at the Axtel et al (2002) model it shows evidence of these extra processes. Within table 2 where household rules are, it gives more detail than used in this projects model. "1. Household fission when a daughter reaches the age of 15." Without knowing the gender of an individual, the distinction between son/daughter is not possible. Point 2 states "The location must have potential maize production sufficient for a minimum harvest of 160 kg per person per year" a per person measurement can only be achieved by knowing how many people live in each household [6].

4.4 Options by Niruthan Puvanenthiran (150148432): Trades between household and increasing disaggregating agents.

Problems

The trained model shows a large error compared to the real model even though it follows a similar time trend as the real model. This is due to the model using basic information including the basic environmental and demographic specification, hence giving only a few hyperparameters to adjust during calibration. By redesigning the structure of the model, it will reduce the error and hence increase the accuracy.

Solution

To improve the solution certain factors needs to be analysed. Currently, the agent dies when the maizefield quantity goes below 800Kg and cannot find a suitable space to produce enough maize for that year. This causes the agents to die off far quicker than before. By the agents being removed quicker, it causes the model to underestimate. Since the model has no food sharing mechanism between the household and the storage of food is also limit per household, it affects the overall results. By allowing trades of maizefield between household will mean that fewer agents will be removed from the simulation hence leading to a better fit of the real model. Since fewer agents are being removed, the model should automatically avoid underfitting, hence given an accurate result.

Another method to improve the solution is by increasing the number of agents and their heterogeneity. Currently, the lowest entity is the household (agent). Each agent contains five individuals, so when an agent is created, five new individuals are developed and when an agent dies, five individuals are removed from the household. By disaggregating the agents/household into the individual members means that each individual will have their own age, (in)fertility age, gender and death age, instead of a whole household. In addition, the current model assumes that all genders are females. By separating the individuals to males and females will improve the overall fit of the model. Also, by spreading these hyperparameters through a non-uniform distribution can increase the accuracy.

Implementation

New implementation requires a new function to be created for the model class. No new classes are required.

Figure 12 shows the proposed UML diagram changes. The new model class involves removing initialiseAgents(): void and replacing it with initialiseHouseholds(): void and initialiseIndividuals(): void function. initialiseHouseholds(): void will have similar features as initialiseAgents(): void.

Class	Description
initialiseIndividuals():	Used to initialise each individual regarding their age, gender, min and max fertility age, death age. All the removal of the agent will be as normal but linked to this function
initialiseHouseholds(): void	Holds the entire household and food quantity. When certain limits have occurred, it will send a signal to moveMaize(maizeID,m maizeLoc): int and see if trading can occur.

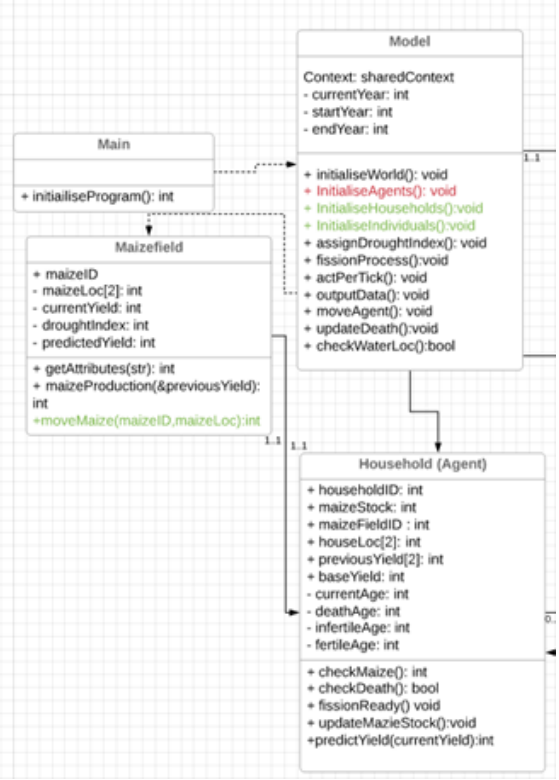

Figure 12 - UML Class Diagram of the updated model

By implementing those function, it will help with the overall fit with the function.

Other Potential implementation related to food production

By looking into increasing productivity of maizefield will help the overall model. By agriculture intensification - monocropping, water and soil control and use of fertilisers would have increased the productivity of maizefield, hence lowering the number of agents that have been removed as the threshold was below 800Kg. Also, farming not only in one location but farming in other location will increase the maizefield stock, so even in bad years, the humans will still have crops from the other farms to compensate for the loss.

Overall by considering other factors, it can increase the survivals of agents.

4.5 Option by Akinola Alexander Dada - Addition of Network Effects (160140802)

The Artificial Anasazi model currently implemented utilises a minimum realisation rule-based framework to chiefly using the state of the environment with respect to the state of any one agent to determine the behaviour and response of agents in the system. Environmental states such as:

- Priority distances to water sources
- Availability and proximity to fertile land
- Exact yield Amounts.

These factors are key to the behaviour of the agents however the model as it currently stands does not consider network effects of the decision machining processes of the agents, such as those derived from familial ties between preceding agents and the agents they fission, their children, or any sort of possible symbiosis between neighbouring agents due to proximity, when deciding the behaviour of the agents.

These network effects would have the ability to tend agent behaviour in such was as towards both mitigate the effects of and raising the threshold of the environmental factors necessary to lead to a change in the current agent's position or state, i.e. by enabling resource sharing from those networked agents who have set up their maize production on land more capable of yielding in excess of what is required for survival or those closer to a more abundant water sources, and exacerbate the difficulties in the ability of the agents to sustain themselves if its other networked agents have left the valley, increasing the probability of that agents would choose to leave the valley despite otherwise bountiful environmental factors, relative to those required for survival, and maize yields. This formulation more closely mirrors real-life interaction within communities.

The omission of accounting these effects, more specifically the exacerbating effect, from the considerations of the implementations initially presented by Axtel and later Janssen, could in part account for a phenomenon present in their results where their models were incapable if accounting from the sudden steep decline in population towards the end at around 1300AD.

In the Anasazi model implementation, the bonds created by these effects would take the form of weighted variables linking agents together which would change depending on the nature, the distance, time and generations between agents in the simulation. These values would also affect and modulate the required level of yield for the current agent location at any given time based on the yields of the linked agents.

For instance, a mother agent would have very strong ties to the agent from which it genialised. This would lead to a symbiotic relationship between the two agents, where, for a number of simulation cycles, they would effectively operate as a single agent with initially less, as the fission agent consumes more than it produces, but then subsequently more resources available to both. The value of this bond and the degree to which resources are shared then reduces asymptotically over time and distance. Similar operations are performed with neighbouring agents, but with weaker initial bonds and without the initial consumption function, but values that weaken based on functions of the number of agents within a set radius and the length of the radius.

In order to implement such a networking scheme, a network Manager class would be added to the model consisting of functions which check and set and update the value of the bonds between each agent at each time step by monitoring the ages and locations of each agent. Thus, this class would require access to the sharedSpace, sharedContext and points Repast classes, as well as access to agent and simulation data such as ages, maize crop yields and simulation time.

Figure 13 is a diagram showing how this implementation would look line in a UML class diagram.

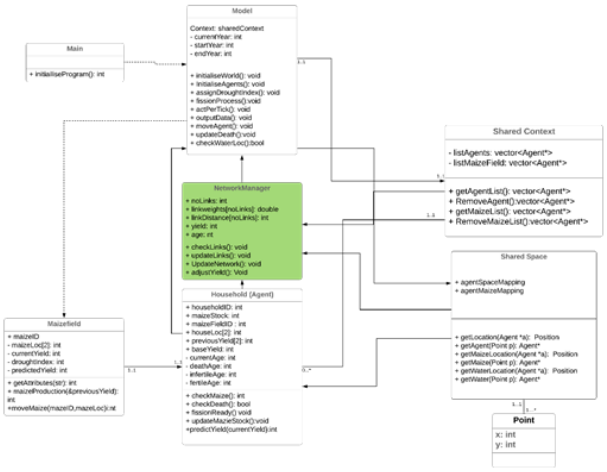

Figure 13 - Newly implemented UML diagram model

Updating the value of the bond between agents would rely on a multivariate function or several functions of both times, distance, number of agents with said distance, the values of the coefficients of which would be hyperparameters distributed within a certain range of values to be tuned during the calibration process.

5 Final Proposal for an Enhanced Model

All points described in the previous section, will improve the model developed in this project. A various selection of improvements were chosen, that can be synthesised, creating a design for an enhanced model. A few proposals of similar nature have been grouped together for simplicity of this section. It is important to give a critical appraisal for each point before combining them to ensure improvements will be beneficial not degrade the accuracy of the model.

Firstly, to improve the model it is possible to create a more detailed agent. Making agents people not just a generic household of 5 allows a more complex and therefore potentially accurate model. Secondly, an improvement to the process of moving and selecting a new farm plot was suggested. Thirdly, a proposal for a model that allows trading between households. Finally, the suggestion to improve network effects.

Each suggested improvement would undoubtedly increase complexity and therefore slow down the run time of the model. At this stage there is only an interest in improving the accuracy of results, therefore this factor can be ignored.

5.1 Critical Appraisal of Suggested Enhancements

5.1.1 Using More Detailed Agents

This concept follows the idea to create more specific agents, as an improvement to the model. With agents as individual people the model can be developed to create enhanced processes. These processes are explained in detail within section 4.1/4.3.

In making this improvement, the model would be more realistic in replicating the actual Anasazi behaviour that would have existed. When dealing with an ABM such as this it can be argued that using households is a needless complication. The complexities of the real-life Anasazi tribe behaviour are too advanced to replicate within a computer simulation. The model created will always be an abstract view that tries to replicate the original real behaviour. Therefore, by making the agents more specific would the model become less abstract, or due to relatively complexity, would there really be a change.

The extra processes that would now be available will undoubtedly increase the accuracy of the model. To match trends shown in the target data this improvement should still be implemented because it will create more accurate behaviour leading to more accurate results even if only by a small fraction.

5.1.2 Farm Plot selection

This proposal involves less changes from the original designed model compared to the last suggestion. The move and selection process make up the biggest section of the model and therefore will still make a significant impact if improved.

Giving the decision making more factors to create a significant farm plot selection a more accurate model can be developed. The factors presented within section 4.2 would all contribute to making the model process match real behaviour. The main reason agents leave the valley is the lack of suitable farm plot which is essential to the Anasazi tribe's population changes. When making this process more realistic it would help match the population changes to the target data.

Adding all these new behaviours would make it more accurate, but would it improve the observed results. The trends trying to be replicated only observe population changes. If this proposal was implemented it would be unlikely that significant changes would occur to the population trends shown. This is because there is no significant link between this change and why the agents would leave the valley, only make the process more complex.

5.1.3 Trades Between Households

This concept is designed to keep the current process and adding conditional statements to the move function to improve the overall model. Improvements in the overall distribution of maizefield can help to improve the model significantly.

By increasing the distribution of maizefield around the population can significantly improve the overall model. As explained in section 4.4, one of the main reasons why the agents are dying-off quickly is due to the lack of maizefield storage per household and due to the limitation of maizefield. By distributing the maizefield between household it leads to less agents to die, leading to a better fit of the overall model.

This design has a significant impact in the overall model, as there are less chances of underfitting. However, there may still be issues past year 1250, where the agents are not dying off quickly, since they will still have more food source to live on. This can reduce the accuracy overall.

5.1.4 Addition to Network Effects

This Proposal involves the addition of an extra class which superimposes an additional layer for monitoring and recording interactions between agents. This layer seeks to further mimic the

behaviour of people in real human communities by roughly including the effects of social links, connections and other community obligations have on the decisions made by the agents.

The proposal would create a numerical manifestation of the links between agents by introducing a mathematical framework describing the intensity of the various links. In order to implement such a networking scheme, a network monitoring class would be developed consisting of functions which check and set and update the value of these links at each time step by monitoring the ages and locations of each agent.

Though this proposition might be technically implementable in software, only requiring additional calls to the network manager class by the model class when deciding when and where to move agents. Determining the mechanics behind the linking mechanism, to the extent of being able to express a formulaic approach, would require great insight derived from extensive research into how these links manifest and sustain themselves. Such data would be extremely difficult to find for a model like this where archaeological data is the only source of real information.

Also, the implementation itself may lead to an unstable simulation. If the distances and link strength between agents and their offspring, or other linked agents becomes too important a condition. Agents may fail to find enough suitable places to relocate or farm, which will lead to the collapse of the simulation. That being said, it is a reasonable inclusion, but its modelled effects are as yet unproved.

5.2 UML for the Synthesised Enhanced model

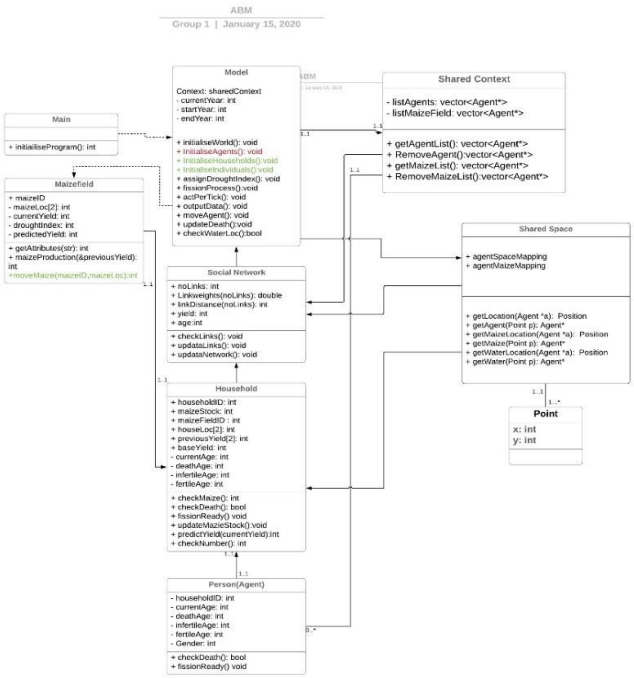

Figure 14 - the UML diagram of the enhanced synthesised model

Within this new conceptual model shown in figure 14, all the proposals discussed have been joined together creating a new design. Two new classes; Person and Social Network have been added. The new person class means agents are now individuals and the social network class means social connections will now also be incorporated. When moving maize fields there is now more reasoning behind choosing a field and trading of maize between households is now possible. By following design, implementation and calibration as before with the new additions it should yield more accurate results.

6 Reflection

6.1 Overview

The objectives of the project were to:

- Utilise a system's engineering approach to design, develop and test an agent-based simulation.
- Apply methods of calibrating the simulation to empirical data using advanced system identification methods.
- Propose adaptations and enhancements to the behaviours and networks of the simulation.

6.2 Brief Research Period

To perform these operations, a period of time was set aside at the very start of the course that was dedicated to understanding the nature of the tasks to be undertaken. The prevailing research material, most especially (Axtell, Janssen) was referenced heavily as the details of their implementations and understandings were used as the benchmark in emulating and corroborating the processes undertaken and results derived from our implementation. Time was also taken out to explore the Repast framework, however, as with learning and application of most new programming paradigms, most of the work in understanding the mechanics and lexicon of the framework came through on-the-fly iterative attempts at implementation. During the scheduled lab sessions requires significant usage and trial to become familiar with the processes of the framework. This issue was exacerbated especially by the scale of implementation required to achieve the objectives, within the time restrictions of a semester-long module.

6.3 Systems Engineering

An organisational and task structure was required to determine and map out the system characteristics as well as coordinate the activities of the team members and produce the series of events that make up the process. This structuring also served to assign these tasks to group members and create the degrees of accountability require to complete operations and meet deadlines effectively. As detailed in the systems engineering section, this structure was based on requirement derivation as well as the implementation of Agile techniques for task management. This was displayed when producing the first deliverable, which produced the UML class diagram and flowchart showing all the requisite parts of the model to be developed as well as a breakdown of their component functions, interactions and interdependencies.

The team was able to complete the majority of the tasks set by following the structures and methodologies highlighted in the Systems Engineering section and produced deliverables in time for their due dates, although many issues were faced in following this structure.

One of the issues faced was in sticking to the timing and planning structure generated for producing deliverables, which led to the mal-coordination of activities. This resulting in insufficient effort being put into fully grasping the interoperability and connections of Repast classes, such as those pertaining to the connections between the SharedContext, the SharedSpace and the Repast point class amongst others, while constructing solutions towards achieving the first deliverable. This created a situation of imbalanced task assignments to some team members as well as substantive re-working being needed for the design to be realisable with attention needing to be placed on how these different classes were to work together. This also led to insufficient time being allocated to the development of the UML diagram and its subsequent presentation, with certain elements of the design not being included in the diagram and the being inadequately prepared for the presentation.

6.4 Software Development

Software development was done primarily while moving towards achieving the targets required for the Second deliverable. In doing this the group assigned tasks based on individual classes as shown in the UML diagram. Each team member was assigned certain classes to develop, supposing that each class would be self-contained. The thesis behind the methodology was to modularise the code as far as possible based on the assumption of a decent degree of mutual exclusivity in the interoperability of each section. This meant that each section of code could be developed independently and upon completion be integrated and tested independently using fictitious values to initialise the classes, leading to easier troubleshooting and change-making.

Many elements often required access to data being manipulated by other classes such as; agent class calling the maizefield class's maize calculation method and then the model class deciding on the best course of action based on that result. Issues were increased as several members were unable to work with a C++ implementation due to the required skill level.

This contributed to uncertainty about the nature of the second deliverable where we had initially assumed full or substantially near-full working code needed to be presented rather than the more milestone centric approach of meeting and displaying certain performance capabilities.

By the time of the assessment, we had managed to:

- Instantiate an agent and update its properties
- Instantiate the network and update its properties
- Read archaeological data as input and propagate this to the network
- Generate an output file recording the annual number of households over time

However, we were unable to implement in time functions that would:

- Reduce yields to below the minimum agent requirements and demonstrate the effect
- Read a new 'death age' parameter at run-time and demonstrate the effect

These were later implemented afterwards along with completing the software development to the extent where we could have a calibratable program which could run un-aided.

The mix up also meant cutting into time that would have been used to organise our endeavours into a format that could be easily presented leading to problems with the smoothness and coherency of the demo.

Another set of issues faced were those concerned with version control. We initially started using Google Drive to store and share program files, as the was familiar to all the team members and made for an easy start, leading to multiple version from multiple people being available at any one time. We later moved to Github to host our codebase and handle version control. This still presented certain co-ordination problems where different people making changes to the same file would overwrite one another if the committed schedule was out of order. This was shown during the presentation as we were unclear which version of the codebase we were demonstrating.

6.5 Calibration

Due to not meeting the full requirements for the second deliverable on time as stated above, we had to complete those tasks while moving onto to the calibration of the program for the third deliverable. In doing this, the team was split into 2 groups with one group dealing with completing and making the required software changes, while the other investigating and implemented the various calibration schemes. As described in the calibration section, a rejection sampling method based on an RSME metric and elimination of model runs when the peak population exceeds 400 was selected and implemented in R using the EasyABC library.

We were able to successfully produce a workflow of a calibration process which provided an overview of ABC methods, including a discussion of the advantages and disadvantages of three different schemes, the chosen method inclusive. While deciding certain factors such as the Uninformative priors for a model, we choose to implement them using uniform distributions, with some logical conditioning during the sampling process, but also deliberated whether normal distributions would have been logical.

Due to the team being split and thus not having our resources fully dedicated to the calibration process, our rejection sampling implementation was done with an ad hoc sequential process. However, a methodical and more formulaic sequential method has been described in the Calibration section.

In preparing to demonstrate the calibration process, we ran our calibration program prior to the event in order to produce and present an off-line best fit result as the calibration time necessary to produce any useful results would run int to the hours if not days. We also managed to show a limited on-line run of the calibration process. It would have been better if we had incorporated a way to automatically display the results of the process rather than via manual data manipulation and entry. As well as display more data produced by the calibration process, such as the posterior densities for the off-line process and the output traces for all simulations corresponding to the posterior.

Our model ended up being able to closely represent the data the transition following the earlier stages of the historical data but trialled-off and become much less accurate and able to represent the historical data during the later sections of the data.

6.6 Model Enhancements

After the calibration process was concluded, team members were each tasked with devising ways the model and/or agent simulation could be improved to enable the model to track the historical data more closely, all of which are available in the individual model enhancement sections. Each of these suggestions was then utilised to form a singular proposed for model changes that could improve the model performance, which is presented in the Team proposal section.

6.7 Summary

Through the course of the module, we have been able to achieve all 3 top-level requirements of being able to utilise system's engineering approaches in the design, development and testing of an agent-based simulation while applying methods of calibrating such a simulation to empirical data that is to be reproduced, using advanced system identification methods and gaining insight enough to proposing adaptations and enhancements to the behaviours and networks of the simulation-based on knowledge derived from the module taught material and external research.

Many difficulties faced in pursuing these targets, both technical and organisational, but great insights into to workings of agent-based approaches and their capabilities were received as well as a possible set of potentially useful applications.

7 Reference

[1] D. Jared M., "Life with the artificial Anasazi," *Nature*, pp. 567–569, 2002.

[2] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 99, no. SUPPL. 3, pp. 7280–7287, 2002.

[3] J. S. Dean *et al.*, "Understanding anasazi culture change through agent-based modeling," *Gener. Soc. Sci. Stud. Agent-Based Comput. Model.*, no. May 2014, pp. 90–116, 2012.

[4] M. A. Janssen, "Understanding Artificial Anasazi," Oct. 2009.

[5] "Mortality in England and Wales - Office for National Statistics." [Online]. Available: https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/deaths/articles/mortalityinenglandandwales/2012-12-17. [Accessed: 17-Jan-2020].

[6] R. L. Axtell *et al.*, "Population growth and collapse in a multiagent model of the Kayenta Anasazi in Long House Valley," *Proc. Natl. Acad. Sci.*, vol. 99, no. suppl 3, pp. 7275 LP – 7279, May 2002.

[7] Gunaratne, C., Garibay, I., & Dang, N. (2020). Evolutionary model discovery of causal factors behind the socio-agricultural behavior of the ancestral Pueblo.

[8] van der Vaart, E., Prangle, D., & Sibly, R. (2018). Taking error into account when fitting models using Approximate Bayesian Computation. Ecological Applications, 28(2), 267-274. doi: 10.1002/eap.1656

[9] Csilléry, K., François, O., & Blum, M. (2012). abc: an R package for approximate Bayesian computation (ABC). Methods In Ecology And Evolution, 3(3), 475-479. doi: 10.1111/j.2041-210x.2011.00179.x