ODD protocol
Game of Thrones model Step 8

OVERVIEW

Purpose

This model explores the dynamics between sheep, wolves, humans, and whitewalkers in Westeros. This is the eight step towards creating a model of Westeros. It is based on the Wolf Sheep Predation model (Wilenski 1997). The model includes cities and roads connecting them. The major cities that are connected by major roads form a network (which can be either directed or undirected) that can be analyzed using different centrality measures. Conflicts between houses can occur, as they play the prisoner's dilemma game with one another. The house with the highest cumulative score gets to increase its territory. The population size and distribution can be either based on a slider, or on the story (each territory gets a defined number of humans). Humans can move at random or move mostly within their own territory. The model starts in summer, but the patches beyond the wall already have snow, which reduces their productivity. As time goes, the season changes to fall, and then winter. When winter starts, 10 whitewalkers appear in the land beyond the wall and they start their march South. As they encounter living beings (be it humans, sheep, or wolves), whitewalkers kill them and make a new whitewalker. As whitewalkers walk south, they cover each patch they walk on with snow. In this step, we added caches of obsidian that can be picked up, exchanged, and used to kill whitewalkers.

Entities, state variables, and scales

This model uses the wolves and sheep agents of the Wolf Sheep Predation model. To those, it adds a breed of human agents who can feed on grass or sheep, and who domesticate some of the wolves they encounter. It also adds a breed of cities that are sworn to a certain house and have a certain size, and a breed of 'houses' that represent the houses that hold territory and that play the prisoner's dilemma. It has a breed of whitewalkers who want to move south and kill any living beings they encounter. Humans now have obsidian weapons to fight back.

Table 1. Global variables used in this model.

Global variables	Explanation
Max-sheep	To stop the simulation if there are too many sheep
Land-patches	Identifies that patches that are land (can be walked on)
Initial-number-sheep	Determines the number of sheep at setup
Initial-number-wolves	Determines the number of wolves at setup
Initial-number-humans	Determines the number of humans at setup
Grass-regrowth-time	After how many ticks grass regrows (resource)
Sheep-gain-from-food	How much energy sheep get from grass
Wolf-gain-from-food	How much energy wolves get from eating a sheep

Rate-sheep-eating	How often humans eat an encountered sheep
Sheep-reproduce	Each sheep's probabilities of reproducing if they have energy
Wolf-reproduce	Each wolf's probabilities of reproducing if they have energy
Humans-reproduce	Each human's probabilities of reproducing if they have energy
Show-energy?	Switch, determines if we see sheep's energy on the window
Link-type	Determines the type of edges created between cities (directed or not)
Centrality-type	Determines which centrality measure is calculated between cities
Conflict?	Switch to determine if houses will fight or not
Battle-strategy	Determines the cooperation probabilities for houses
Chance-of-cooperation	If Battle-strategy is "Random", this determines the threshold for cooperation
Display-house-ownership	Switch that allows seeing the territory changes when conflict occurs
Average-household-strength	The mean throne scores of all houses playing the prisoner's dilemma game
Population-size	Determines the population size at setup
Territorial-movement	Determines if humans move randomly or prefer to stay within their territory
Season-change-rate	Sets the rough interval at which seasons change
Season	Records the current season (summer, fall, or winter)
Obsidian-distribution	Sets which cities get obsidian caches

Table 2. Turtle variables

turtle variables	**Explanation**
Energy	Counter that gets updated when an agent moves and eats. At 0, the agent dies.
Wolves variables	
Domesticated?	True/False. Record if a wolf has been domesticated.
Humans variables	
Obsidian	Record the quantity of obsidian the human has
Cities variables	
City-name	The name of the city or castle (from GIS data)
City-size	The size of the city or castle (from GIS data)
Centrality	Takes on the centrality measure calculated (only if part of network)
Obsidian-cache?	Identifies if the city has obsidian or not
Houses variables	
Name	Name of the house
My-color	Color associated with the house
Throne-score	Score of the prisoner's dilemma game
Cooperate?	Identifies if the house cooperates or defects that round
Rival-list	Identifies the houses they will play against
Defeated?	Records if the house still holds land
Population	Records the number of humans each house should have (from GIS data)

Table 3. Patch variables

Patch variables	**Explanation**
countdown	Counter used to regrow grass.
Land?	Boolean that identifies if the patch is land
Wall?	Boolean that identifies if the patch overlaps with the wall (and thus becomes it)
House	Identifies which house owns the patch
House-color	Records the color linked to the house (for visibility purposes)

Road?	Identifies if the patch is part of a road
Border?	Identifies if the patch is at the border of the winner territory
Resources	Records the grass resources at the patch
Snow?	Records if the patch is covered in snow or not

Process overview and scheduling

At every tick, humans, sheep, and wolves move, which depletes their energy, and they look for food. Humans can pick up obsidian, exchange some, or both if applicable. If agents' energy is depleted, they die. If they still have energy, they consider reproducing. If the user decides to include conflict, all houses that hold land play the prisoner's dilemma against one another and calculate their cumulative score. If both cooperate, both gets a payoff. If both defect, both get a negative payoff. If one cooperates and the other defects, the defector gets a very big payoff. When all houses have played, the house with the highest score wins and gets to increase its territory. If the user decides to set territorial movement, humans will tend to prefer walking to patches that are owned by the same house as them (with a certain probability).

Ticks do not represent any specific unit of time. However, at a certain number of ticks, roughly defined by the season-change-rate slider, the season changes (from summer at setup to fall, and then winter). When winter is here, whitewalkers appear north of the wall and start marching south. They kill every living beings they encounter and replace them with new whitewalkers. However, if they encounter a human that has obsidian, the whitewalkers die.

DESIGN CONCEPT

Basic principles

This model added the presence of obsidian to the previous model, which continued modifying the Wolf Sheep predation model that aimed to show the predator-prey equilibrium that can arise between two species.

Emergence

None.

Adaptation

None.

Objectives

As in previous versions, humans, sheep, and wolves have the objective to eat and reproduce. Cities do not have objectives. Houses' objective is to get the highest score at the prisoner's dilemma game so they can increase their territory. Humans' objective can be to move within their own territory. Whitewalkers' objective is to move South and kill every living being they encounter to take over the world. In general, the objective of this version of the model is to give humans some chance at survival by allowing them to use obsidian tools to kill whitewalkers.

Learning

None.

Prediction

None.

Sensing

Agents can sense if they have reached water or the edge of the map, in which case, they turn around. Humans can sense the ownership of the patches towards which they walk, if *territorial-movement* is ON, as well as sense if other people on the same patch do or do not have obsidian. Cities can sense who they are connected to via links. Houses playing the prisoner's dilemma get to know what strategy their opponent chose. Whitewalkers can sense if a living being is on the same patch, at which point, it kills it (if it doesn't have obsidian).

Interaction

Wolves and humans interact with sheep by eating them. Humans interact with wolves by either domesticating them or getting eaten by them. They interact with cities to pick-up obsidian and interact with one another to give each other obsidian tools when applicable. Houses interact with one another when they play the prisoner dilemma's game. Whitewalkers interact with every other agents by killing them and creating a new whitewalker in their place. Whitewalkers also interact with patches as they cover them with snow as they move.

Stochasticity

The energy level of the agents is set randomly at the beginning of a simulation (with a maximum value set by a slider). Which patches are depleted at setup is also random, as well as the regrowth countdown of depleted patches.

If the user chooses to have conflict between houses and sets the *battle-strategy* to "random", the cooperation probabilities are random, but affected by *the chance-of-cooperation* slider.

The timing of the season change is random but controlled on the *season-change-rate* slider.

Collectives

Land-patches identifies the set of patches that can be walked through and that have resources to deplete. Houses represent the whole territory held by each house.

Observation

There is one plot that follows the population sizes of sheep, wolves, dogs, and humans. There is also a monitor that tells us which house has the highest throne score when 'battles' occur.

DETAILS

Initialization

At setup, the model imports a raster and resizes the World window to fit the new map's dimensions, if necessary. The raster's values represent the house-color of the seven kingdoms. All patches record the raster's value as their *house-color*. Patches with *house-color* above 0 are land patches, whereas others are water. Water patches set their colors to blue. The model then imports a line vector that covers the extent of the wall. All patches that intersect that line become the wall. They set their *land?* variable to *false* to make sure that they cannot get walked through. Wall patches are white. The model then imports a political map, which holds vector polygons of the different kingdoms with specific information. Patches that intersect with any polygon record that polygon's 'claimedBy' value, which is the identity of the house that owns the land (e.g. Stark, Tyrell,…). All water patches are claimed by 'no one'. As there is a bit of disconnect between the polygon vector and the Westeros raster, the model tells any patch who thinks it is water but overlaps with a house polygon to simply forget that house and instead be claimed by 'no one'. Each territory creates a house agent, which will play the prisoner's dilemma if conflict arises. Those agents are hidden but know the name and color of the house to which they belong. Each of those house agents records their population size from the political vector file. If population-size is set to "faithful to the story", the model uses those numbers to create a certain number of humans in each territory. If that setting is set to "random", it uses the *initial-number-humans* value instead and spread all those people randomly on the landscape. Humans are colored based on the territory they were 'born' into (e.g., Starks are grey). The model then imports a vector file that contains the locations of cities and castles and creates cities there. Cities' sizes are twice the value of the vector's 'size' variable, whereas their color is based on the *house-color* of the patch on which they are created. Finally, the model imports another vector file, which has lines representing the major roads. All patches intersecting with those lines set their 'road?' variable to *true*, set their color to black and do not get resources.

At setup, the model is set in summer. However, all patches north of the wall are covered in snow (white) and have reduced productivity (big countdown and low resources). About half the

land patches that are not wall, road, or covered in snow become green and are given as much resources as the *sheep-gain-from-food* slider says. The rest get a countdown value that determines when their grass will grow back. The number of sheep and wolves created are based on sliders. They are placed randomly on the land, with a random energy value with max based on a slider.

Cities connected by roads are connected using links into a network, which can be either directed or undirected. Then, the model distributes obsidian caches among cities. The identity of cities with obsidian varies based on the obsidian-distribution variable. It can be restricted to Dragonstone (location), restricted to Targaryen cities (territory), or restricted to cities connected to Dragonstone via the link network (network).

Input data

This model uses five GIS maps. These maps were taken from https://www.cartographersguild.com/showthread.php?t=30472 and modified to suit our modeling needs.

- Land_raster.asc: raster map with the values representing the color number (in NetLogo swatch) of the house that owns the land.
- Wall.shp: line vector that covers the extent of the wall.
- W_political_revised.shp: Polygon vector. Each polygon represents a house. It holds the following information for each polygon:
    - Name: Name of the kindgom (e.g., “The North”,”Crownsland”,…)
    - Population: Estimated millions of people living in each kingdom (based on https://atlasoficeandfireblog.wordpress.com/2016/03/06/the-population-of-the-seven-kingdoms/)
    - House-color: NetLogo number associated with each house’s colors (arbitrary).
    - claimedBy: The name of the house that claims each kingdom (at the beginning of A Song of Ice and Fire).
- Westeros_locations.shp: Point vector of important locations. It holds the following information for each point:
    - Type: The type of location (e.g., Castle, city, ruin, other, …).
    - Name: Name of the location (e.g., Castle Black, Winterfell, …).
    - Size: An assigned size (scale of 1-5) for each location. We assume it represent the population size of each.
- Westeros_roads.shp: Line vector of major roads. While it holds a few variables (name and size), we do not use them in our model.

In addition, the model uses an .nls file that focuses on procedures that are linked to whitewalkers.

Submodels

Move:

This is called by all turtles (except humans if *territorial-movement* is ON). The turtles wiggle a bit. If the patch ahead is either the edge of the world or water, they turn 180 degree. They then move forward by 1.

Move-humans:
This can be called by humans if territorial-movement is ON. The human wiggles a bit and identifies a set of patches ahead (in a cone of 180 degree vision and 1.5 patch depth). If all the patches identified are from the same house, the human chooses one of them as its next step. If the patches are of different houses, the human gives them a probability. The probability is higher for patches of the same house as the human's, whereas patches from different territories have lower ones. The human chooses one of the patches, based on those probabilities. After choosing the next step, the human walks to it.

Eat-grass:
This is called by sheep. The patch becomes brown and the sheep adds a certain value (based on slider) to its energy level.

Eat-sheep:
This is called by wolves. If there is a sheep on the same patch as the wolf, the sheep dies and the wolf adds a certain value (based on slider to its energy level).

Death:
Called by all agents. If energy reaches 0, the agent dies.

Grow-grass:
Called by all land patches. If the patch is brown, it checks its counter. If the counter is at 0, the patch becomes green. If not, the counter is updated to its value -1.

Add-humans:
Called during *setup*. Creates a certain number of humans. If the *population-size* variable is set to "faithful to the story", the model uses the population value of each houses to set the number of humans created in each kingdom. That population value is divided by 100 to avoid creating too many humans. Each kingdom creates that number of humans and spread them out on their territory. If the resulting number is below 1, 2 humans are created. If the *population-size* is set to "Random", *initial-number-humans* of humans are created over Westeros. All humans set their colors based on the house that owns the patch where they were born.

Reproduce:
Called by all agents. They roll a die. If it is successful, they create a new agent and give them half of their energy.

Humans-eat:

Called by humans. A certain percentage of the time, they will look for a sheep to eat. If there is one on the same patch, that sheep dies, and the human adds a certain value (based on the wolf-gain-from-food slider) to its energy level.

When the human does not look for a sheep, it looks for grass. This calls the *eat-grass* procedure for simplicity.

Domesticate-wolves:
Called by humans. If there is a wolf on the same patch, and that wolf is still wild, the human rolls the dice. About 50% of the time, it will domesticate the wolf. The remaining times, it will get eaten by the wolf (who gets 1.5 times energy as it would from a sheep, arbitrary decision).

Humans-feed-animals:
If there is a domesticated wolf on the same patch (aka, a dog), it will ask the dog to feed on a sheep if there is one around, but the human will take some of that energy (they share the sheep).

Calculate-centrality:
This is called by an Interface button. It asks all the cities that have a link to calculate their centrality and change their size to reflect said centrality. The type of centrality varies based on the ***centrality-type*** chooser. It can be either degree, closeness, betweenness, or eigenvector.

Prisoner's dilemma procedures

Identify-houses
This reporter creates a list of the houses that still hold land. The houses are represented here by their color. The Greyjoys, Wildlings, and Night's Watch are not part of the list as they cannot play the game.

Update-players
This sets the *defeated?* status of houses. Houses that no longer hold land are defeated and cannot play anymore.

Houses-fight-for-territory:
Houses reset their throne score and call the game strategy that is based on the *battle-strategy* chooser. Then, all undefeated houses *fight*, and call the *adjust-territory* procedure.

Set-random-strategy:
Each house rolls the dice (random value up to 99) and if the value they get is lower than the value of chance-of-cooperation, they set their *cooperate?* variable to *TRUE*. If not, they set it to *FALSE*.

Set-faithful-to-show-strategy:
Each house gets a chance-of-cooperation that is based on our perception of the book:

- Tyrell: 70

- Arryn: 25
- Targaryen: 25
- Tully: 75
- Martell: 35
- Stark: 100
- Baratheon: 50
- Lannister: 15

Each house rolls the dice (random value up to 99). If the value they get is below their chance-of-cooperation, they set their *cooperate?* variable to *TRUE*. If not, they set it to *FALSE*.

Fight:

Each house chooses one rival from the list of undefeated houses. Based on the two houses' strategy, they add a certain value to their throne-score.

- If both houses cooperate, they both get 25.
- If both houses defect, they both get -100.
- If one house cooperates, but the other defects, the defector gets 100 while the cooperator gets nothing.

Adjust-territory:

The model identifies the mean throne score of all undefeated houses. It also identifies the house with the highest throne score (winner), and the range of throne scores. The model also calculates the score-difference of the winner, which is the difference between the winner's score and the average and the *percent-to-adjust* value, which corresponds to the score-difference of the winner divided by the range-of scores. So basically, the bigger the win for the winner, the more land it will take over.

If there is one winner (sometimes, all houses can get the same score, so no winner), the model calculates the population density of the winner, and multiplies that value with the *percent-to-adjust* value (and then multiplies it all by 1000) to update the *percent-to-adjust*. This is because a more densely populated territory should have more soldiers to help take over other lands. The winner's territory then identifies patches that border its territory and that are from another house. It tells percent-to-adjust patches to change their allegiance to the winner (color and name). If this creates patches that are surrounded by enemy patches, those take on the color and name of their surrounding neighbors. Finally, if all the main land patches of a certain house have been taken over, but they still have some land on an island, that island changes over.

Update-borders:

This is called by the patches of the winner. If a patch has a neighboring patch from a different house, it tells that patch to set its *border?* variable to TRUE.

Change-islands [*bh-color bh-house-name*]:

bh-color and *bh-house-name* represent the color and name of the winner house. This is based on how many island patches is own by each house. When a house's territory has lost all main land patches, its island patches change to take the color and name of the current winner house.

Change-season:
This reporter reports "fall" if the current season is "summer", and "winter" if the current season is either "fall" or "winter." It is called in *go* at intervals based on the *season-change-rate* slider (with some randomness to it).

Winter procedures

Winter-is-here:
This is an observer procedure that gets called when winter comes. 10 whitewalkers are created and set randomly on the landscape north of the wall. They all face South.

Others-move:
This is a whitewalker procedure. Whitewalkers wiggle a bit and move towards their goal. As for all other agents, they turn around when they reach water or the edge of the World window. If the patch below their feet does not have snow, they cover it with snow.

Winter-advance:
This is an observer procedure called in go when winter is here. It asks all whitewalkers to move (calls other-move). If the whitewalkers encounter an animal (sheep, or wolf), they kill it and create a new whitewalker in its stead. If the whitewalkers encounter a human, they check if that human has an obsidian tool or not. If the human is harmed, it kills the whitewalker. If the human does not have a weapon, the whitewalker kills it and creates a new whitewalker.

Obsidian procedures

Pick-up-obsidian:
This procedure is called by humans within Go. Humans check if they are close to Dragonstone and/or at any city with a cache of obsidian. If that is the case, the humans pick up 10 obsidian tools and place them in their kit.

Exchange-obsidian:
This procedure is called by humans within Go. Humans who have at least one obsidian tool in their kit check if there is another human on the same patch without any obisidian. If that is the case, it gives that human one tool.

REFERENCES CITED

Wilensky, U. (1997). NetLogo Wolf Sheep Predation model. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Most of the ideas modeled here come from the A Song of Ice and Fire books, written by George R.R. Martin.

Some ideas and concepts (colors of houses, for example) are inspired by the HBO Game of Thrones show.