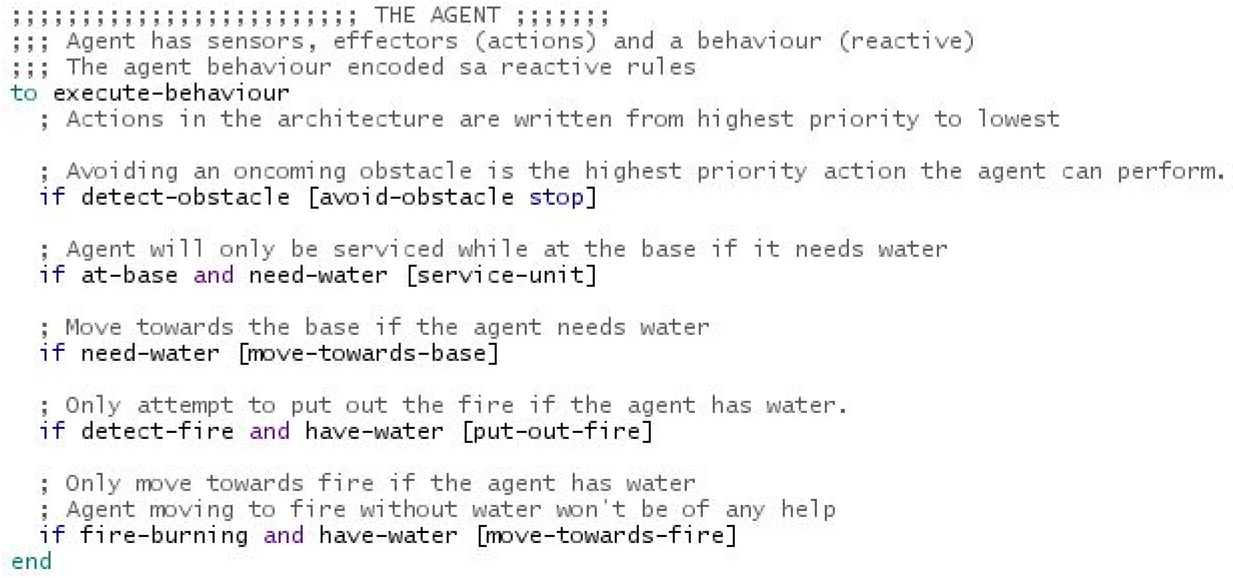
6CCS3AMS: CW1. Kyle Hodgetts

Task 1



```
;;;;;;;;;;;;;;;;;;;;;;;;; THE AGENT ;;;;;;;
;;; Agent has sensors, effectors (actions) and a behaviour (reactive)
;;; The agent behaviour encoded sa reactive rules
to execute-behaviour
  ; Actions in the architecture are written from highest priority to lowest

  ; Avoiding an oncoming obstacle is the highest priority action the agent can perform.
  if detect-obstacle [avoid-obstacle stop]

  ; Agent will only be serviced while at the base if it needs water
  if at-base and need-water [service-unit]

  ; Move towards the base if the agent needs water
  if need-water [move-towards-base]

  ; Only attempt to put out the fire if the agent has water.
  if detect-fire and have-water [put-out-fire]

  ; Only move towards fire if the agent has water
  ; Agent moving to fire without water won't be of any help
  if fire-burning and have-water [move-towards-fire]
end
```

The actions defined above have been written in such a way to take into account the fact that, as per the subsumption architecture, many of the actions could fire at the same time.[1] I have decided to list them in order of priority from highest to lowest. In essence, the code I have written for NetLogo is a flipped version of my proposed subsumption hierarchy.

From here, I will justify my design decisions with respect to application in a real world scenario.

The first action states that should the agent detect an obstacle, it should avoid it, ignoring (for now) any other percepts that may be received. The reason for this action being the highest priority is that, in the real world, if these agents were to be implemented into tangible agents, hitting an obstacle could render the unit completely useless.
If this were to happen, any subsequent percepts processed by the now, totalled, unit, any action returned by the action function may not be carried out.

On the next level, I state that, only if there isn't an obstacle that needs avoiding, if an agent is at the base where water refills take place, the unit should only be serviced when it needs water. A water refill station will have a finite amount of pumps and the worst case will occur when all pumps are in use and another unit then needs to also use the pumps. More formally, given N pumps, at least N + 1 units require refilling. By adding the condition that the unit needs water, before being eligible for refilling, can help mitigate this situation by preventing any units that happen to be 'passing through' from needlessly refilling. It's also worth noting that filling the tank of a fire engine unit will require a substantial amount of time to accomplish.

Further down, I have defined the action for moving towards the base. Naturally, the agent will only consider this action when it is in need of water. I have placed it below the previous action, at a lower priority, to prevent a water-starved agent from continuously evaluating this rule, attempting to move closer to the base when it is already at the appropriate patch / location. Hence, this rule will be checked if and only if it has already been determined that it is not at the base's location.

At this point in the execution of behaviour, we have established the following:

- There is no obstacle that needs immediately avoiding
- We do not need water

This now means that the unit is fit for its purpose of putting out fires and is in an adequate state to respond to the alert of a fire. The position in the hierarchy of responding to a fire is justifiable from the fact that without water, the unit would not be able to respond effectively. A fire that is close to the unit, as per the detect-fire method, is of a higher priority than a fire further away. By placing this action at this position in the hierarchy, an agent can deal with a local fire before moving onto one further away.

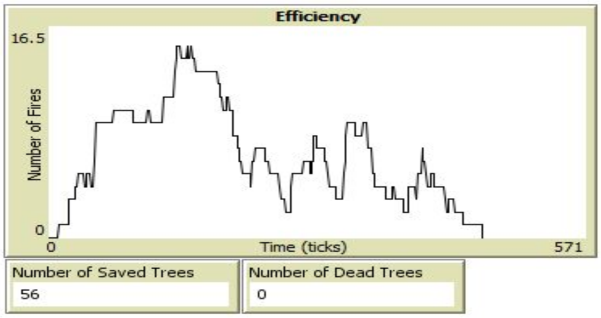
Model Evaluation

As per Michael VanBruaene's article on Useful Performance Measures & Metrics, it is important to choose metrics to measure against that link directly to our quantifiable objectives as an organisation [2]. In our case, this will be minimising the number of dead trees over time and attempting to extinguish all fires as quickly as possible to prevent further fire from spreading.

VanBruaene goes onto to mention efficiency measures that could apply to a range of business organisations. The ones I am going to focus on are: Cycle time for the first set of experiments and Cycle Time and Response Time for the second set of experiments, which involve the levels of water carried by the agent.

With respect to the NetLogo model, the three aspects I will be evaluating to measure the efficiency will be the ticks, which relate nicely to time, the number of saved and dead trees at the end of the simulation and finally the number of fires, and how they fluctuate, over the time span of the simulation.

To achieve a visual representation of agent performance, I added a simple line graph to aid with the monitoring of change of values over time. A sample result will look like the following example:



Number of Saved Trees
56

Number of Dead Trees
0

For this particular run, we can see clearly that no trees died as a result of the fire, meaning the agents were able to save all trees affected. We can also extract the amount of time it took to deal with the fires as a whole by taking the time when the first tree ignited and subtracting that from the time the last tree on fire was extinguished.

Using these figures we can determine the efficiency of assigning a particular number of units given a number of trees and work out how to save resources from there. From a business standpoint, the less units we can get away with on call at once, the better.

Experiment 1 Results

For each tree variable setting, I will run 5 simulations and take the average of all of the results. From there I will compare the averages and analyse the results. Values for initial water will be 25, number of fires will be 40 (max) and the number of fire units will be 10. I want to measure the efficiency of the units and adding more will obviously result in the fires being put out quicker. The values stated will be fixed for all runs of the first experiment. Instead I will increase the number of trees in the simulation and measure how efficient the fire units are at each stage.

Setting 1:
Number of trees = 100

Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	0	50	595
2	0	65	566
3	0	64	590
4	0	56	571
5	0	56	456

Average dead trees = 0.

Average saved trees = 58.2.
Average Time to put out fires = 555.6s

Setting 2:
Number of trees = 200

Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	0	89	477
2	1	92	558
3	0	85	496
4	0	89	584
5	5	83	487

Average dead trees = 1.2
Average saved trees = 87.6.
Average Time to put out fires = 520.4s

Setting 3:
Number of trees = 300

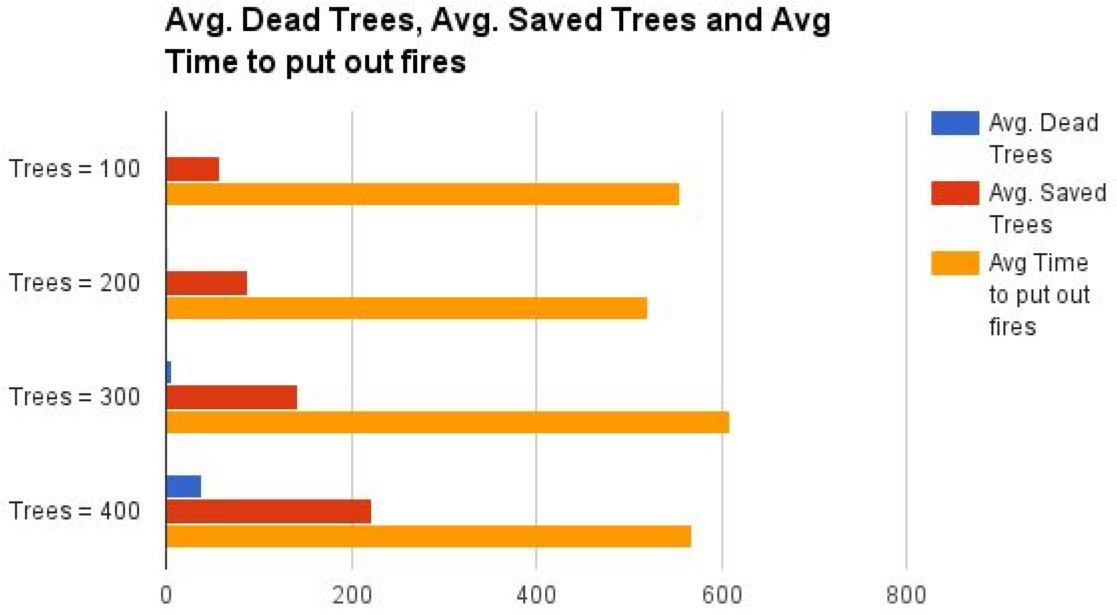
Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	1	147	620
2	5	136	643
3	0	149	641
4	3	135	520
5	21	148	622

Average dead trees = 6
Average saved trees = 143.
Average Time to put out fires = 609.2s

Setting 4:
Number of trees = 400

Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	58	219	645
2	65	190	558
3	19	244	605
4	28	224	770
5	29	234	569

Average dead trees = 39.8
Average saved trees = 222.2.
Average Time to put out fires = 569.4s



As the number of trees is increased, the number of saved trees increases. This was to be expected since more trees in the simulation will result in more fires being ignited and needing to

be extinguished, which will increase the total amount of time taken to extinguish them. The part I took particular notice of was the amount of average number of dead trees that were produced as a result of the increase in trees. I attribute this partly to the number of fire units being fixed and not being able to keep up with the rate of fire ignition, which spreads to nearby trees or, given that most of the units were on one side of the patches dealing with a large patch of tree fires, when a fire ignited on the other side of the patches, some units either didn't move as there was a closer fire to deal with, or if they did move, they didn't make to the fire in time.
I predict that with the steady increase in trees, the further that the number of dead trees will rise. I can therefore conclude that at a certain threshold of trees, more units will need to be deployed to handle the potential outbreak and spread.

To prove this point, I have simulated further runs with an increased amount of fire units to determine a new average of dead trees and whether or not there is an improvement, but only on the tree numbers where dead trees rose significantly[1].

Number of trees = 200

Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	0	92	500
2	0	82	518
3	0	93	424
4	0	94	470
5	0	95	478

Average dead trees = 0
Average saved trees = 91.2
Average Time to put out fires = 478s

[1] Where tree settings were 200, 300 and 400

Number of trees = 300

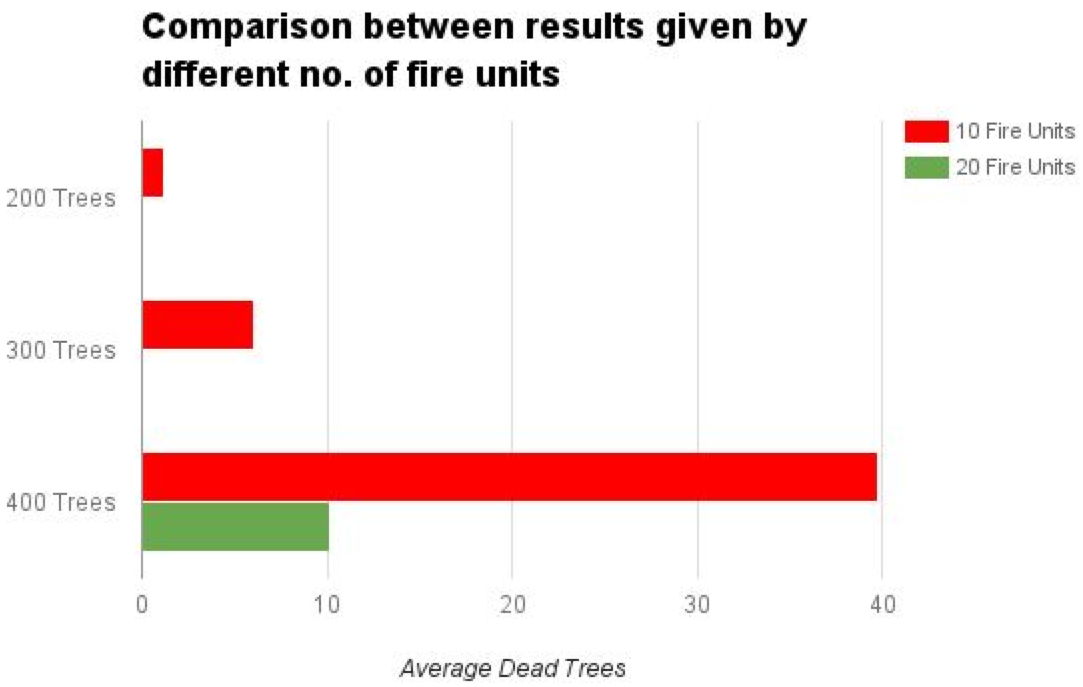
Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	0	99	480
2	0	102	448
3	0	94	389
4	0	93	508
5	0	87	438

Average dead trees = 0
Average saved trees = 95
Average Time to put out fires = 452.6s

Number of trees = 400

Run Number	Number of dead trees	Number of saved trees	Time to put out all fires.
1	0	240	495
2	9	229	516
3	24	214	599
4	8	237	576
5	10	244	539

Average dead trees = 10.2
Average saved trees = 232.8
Average Time to put out fires = 545s

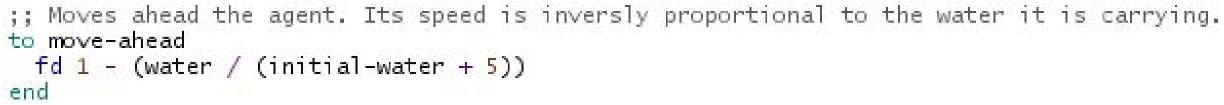


By simply adding 10 more fire units into each situation, I was able to reduce the amount of dead trees substantially and by looking at the other accompanying data, we can see that it also managed to extinguish all fires quicker on average, preventing further damage.

However, the reliability of all results considered can be called into question for various reasons. The most prevalent question to ask would be; How similar to the real life is the simulation? The time unit considered in NetLogo is counted in Ticks, which isn't a concrete unit of time but simply a counter used to aid the plotting of graphs and allow the viewer to see how many times the go procedure has been executed [3]. This doesn't give us a representative value for the amount of time a run takes but instead a relative view of how quickly a run takes to execute compared to another run, with different variable settings.

Task 2

By simply increasing the amount of initial water carried by the fire-units to eliminate the need to refill at the fire station won’t necessary make the model more optimal. More initial water means more weight resulting in slower agent movement.
As modelled in NetLogo by:



```
;; Moves ahead the agent. Its speed is inversly proportional to the water it is carrying.
to move-ahead
  fd 1 - (water / (initial-water + 5))
end
```

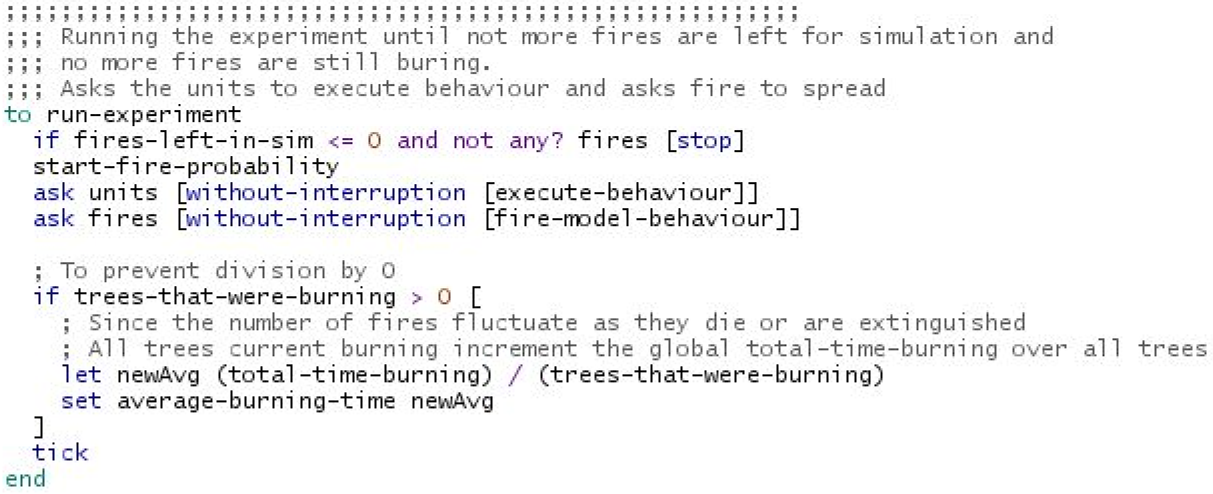
This introduces the question of how much initial water should the units be filled with as to ensure the following doesn’t happen:

- Not too little water so units need to consistently return to the fire station to refill
- Not too much water so units cannot arrive at a burning tree on time to extinguish the fire.
    - Here we can investigate the average waiting time of a burning tree before it is extinguished
    - Average number of dead trees for a given initial water amount.

Just like in task 1, I can calculate the average number of dead trees the same way I did before. However in addition to what I did in task 1, I added, to the NetLogo solution, the functionality to compute the ongoing average waiting time of burning trees.

I introduced the following new global-variables:

- Total-time-burning
    - As each tick passes, every currently burning tree will increment this value by 1
    - Until they die or are extinguished
- Trees-that-were-burning
    - To keep track of the number of trees that were, at some, burning.
    - When a tree dies or is extinguished, the number of turtles of fires breed changes, leading to erroneous results when computing the average.
- Average-burning-time
    - Updated after every program tick and used to update the plot.



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Running the experiment until not more fires are left for simulation and
;;; no more fires are still buring.
;;; Asks the units to execute behaviour and asks fire to spread
to run-experiment
  if fires-left-in-sim <= 0 and not any? fires [stop]
  start-fire-probability
  ask units [without-interruption [execute-behaviour]]
  ask fires [without-interruption [fire-model-behaviour]]

  ; To prevent division by 0
  if trees-that-were-burning > 0 [
    ; Since the number of fires fluctuate as they die or are extinguished
    ; All trees current burning increment the global total-time-burning over all trees
    let newAvg (total-time-burning) / (trees-that-were-burning)
    set average-burning-time newAvg
  ]
  tick
end
```

The new run-experiment procedure is as defined above.

Applying my conclusion from task1[2], I can adjust the initial water variable and compare the change in performance. As before, I will each setting 5 times and take the averages to compare.

Throughout all simulations the following values will be set:

- Fire-units-num = 20
- Tree-num = 400
- Number-of-fires = 40

Initial Hypothesis: Adding more water will improve performance due to lower water refills resulting in lower waiting times.
Goal: To minimise average waiting time and average dead trees

Setting 1:
Initial water = 25

Run Number	Number of dead trees	Number of saved trees	Average Waiting Time.
1	4	247	162.8
2	19	264	201.14
3	2	243	127.81
4	5	218	150.65

[2] 200 fire-units can tend to 400 trees with an initial water of 25

5	1	208	140.67

Average Dead Trees = 6.2
Average Saved Trees = 236
Average Avg Waiting TIme = 156.61

Setting 2:
Initial water = 35

Run Number	Number of dead trees	Number of saved trees	Average Waiting Time.
1	25	199	231.13
2	41	216	204.2
3	11	240	167.47
4	3	197	139.62
5	3	260	166.37

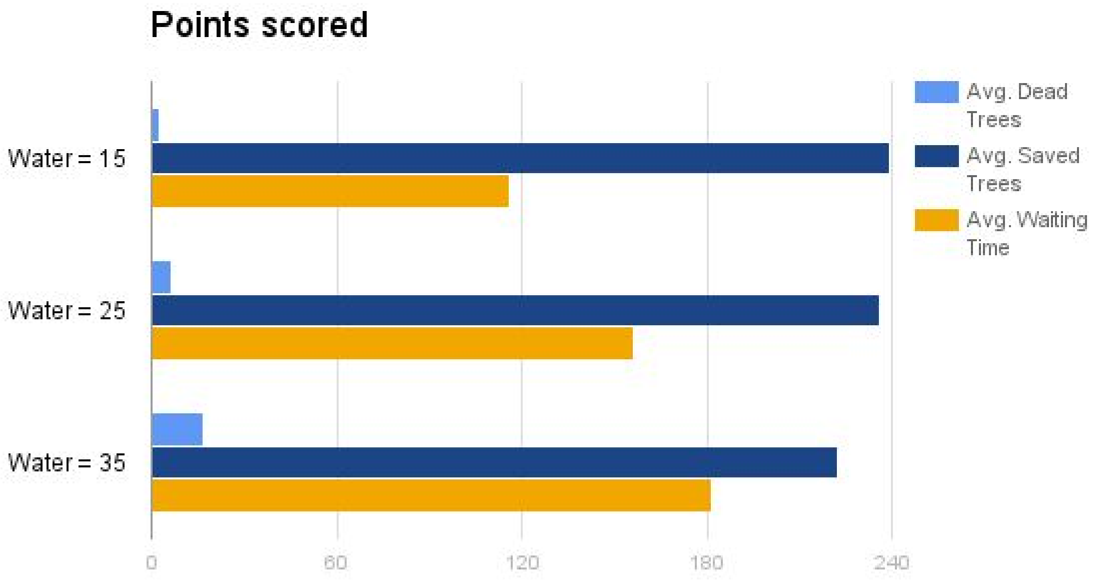
Average Dead Trees = 16.6
Average Saved Trees = 222.4
Average Avg Waiting TIme = 181.76

Setting 3:
Initial water = 15

Run Number	Number of dead trees	Number of saved trees	Average Waiting Time.
1	0	215	113.65
2	0	238	102.11
3	0	243	85.79
4	0	263	135.76
5	12	239	144.14

Average Dead Trees = 2.4

Average Saved Trees = 239.6
Average Avg Waiting TIme = 116.29



Immediately from the results we can see by reducing the water by 10 from its normal rate of 25, we have managed to reduce the waiting time of trees, attributed to the fire-units being able to move faster with less water. In addition, due to the now increased speed of the units, we have also managed to reduce the amount of dead trees on average caused by the fire.

By having less water on board, the units were in fact able to perform better on average, a complete contradiction on the initial hypothesis, despite the potential need for more return trips to the fire station to refill the on board water.

This is good news since in a real world situation, the actual act of refilling the unit will take less time as less water is required to perform better, as we see from the results.

Like in Task 1, the reliability of these results can be called into question in several areas.

Firstly, the simulation doesn't take into account any extra weight, and effect on speed, due to levels of fuel. Despite fuel being less dense than the water [4], it will still add weight and will decrease as it is consumed, just like the water. The simulation emits the use of fuel completely.

Secondly, also with respect to the average waiting times, the simulation doesn't take into account the amount of time it takes to refill a fire-unit with water. In the current state, as the unit goes onto the patch used to represent the fire station, the refill happens instantaneously. In a real life situation, this time will have an impact on how long a burning tree will be waiting for before its extinguished, if the unit being refilled is the only unit in a state to respond. I.e. All other units are busy / out of service / themselves are waiting to be refilled with water.

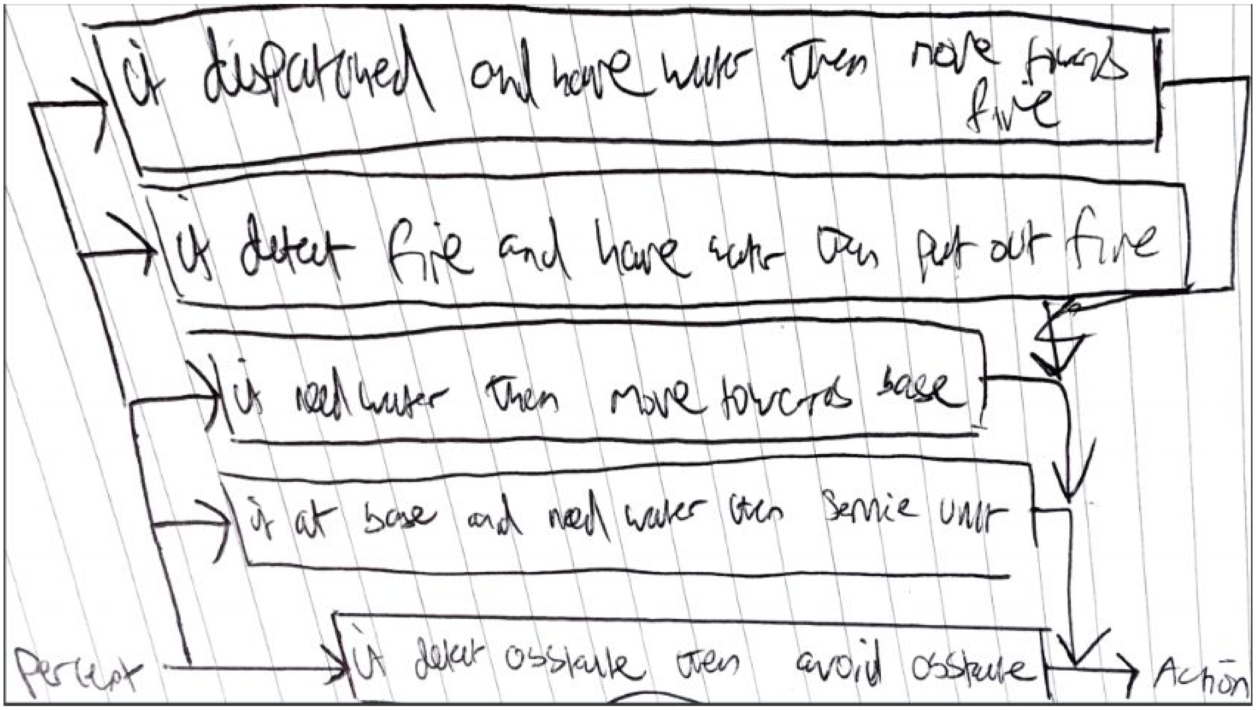
Task 3

In the current state, units simply attend to the closest fire that is currently burning. There is no indication between units that a fire is already being attended to and adding more units to help will not be useful. This also creates a potential issue of units possibly being too far away from another ignited fire to put it out on time, because they are already tending to the closest fire.

My first proposition to improve this current system would be to have that indication of when a fire is being attended to, or simply to prevent all units moving to a fire, purely because it is the closest.

A possible solution for this could be for the fire base to be the one who dispatches the units and not the units themselves. When the fire base receives notification of a fire burning, it will notify all units of how many should go and extinguish the fire, possibly through a broadcasted integer where a unit is dispatched one at a time until the integer is 0. The value of the integer would range between 1 and N, where N is the number of fire units on duty and the value of which would depend on the size of the fire. The larger the fire the larger the integer.

This fits in well with the already established subsumption architecture because essentially I am just updating the prerequisite required to move towards the fire. Rather than just a fire burning being enough to move, replace that condition with have received a signal and have water.


if dispatched and have water then move towards fire
if detect fire and have water then put out fire
if need water then move towards base
if at base and need water then service unit
Percept
if detect obstacle then avoid obstacle
Action

Furthermore, since fires can spread and more units may be required, units attending to fires can increase this broadcasted integer in order to request further assistance from other units.
While this may increase the average waiting time for trees who are burning to be put out, this does now, however, allow more units to remain available to handle other fires which could ignite.

Another possible solution, with respect to saving resources, could be to split the large area into smaller areas and simply have a number of fire units to be responsible for their own sub area and thus only have to be concerned with a smaller number of trees, with the intention of making them easier to manage. This does, however, still have the potential issue of not having enough units to tend to a larger fire, should a sub section have a marginally larger subset of the total trees. To mitigate this, units from sub sections can request aid from other sub sections in a similar way to my previously proposed solution, by emitting an SOS signal of sorts of which units in adjacent sections can respond to, if they are not receiving percepts that are of a higher priority in their subsumption architecture that require actions.

My final proposed, albeit aspirational and expensive, solution to increase efficiency of this system involves stripping much of the current implementation away and disregarding animate fire units completely. The reason trees die in this domain is because the fire units cannot extinguish them on time due to one or more of the following reasons:

- Situated too far away and cannot move fast enough to extinguish the fire before the tree dies.
- Possibly not having enough water and need to refill
- Units could essentially just be avoiding obstacles the whole time since that action is the highest priority in the current subsumption architecture.

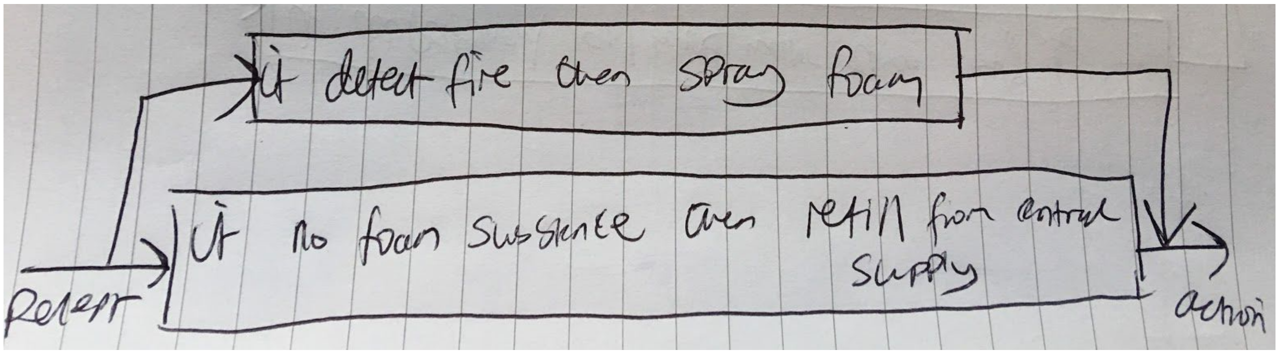
So I propose a new type of inanimate fire unit, such that these problems aren’t a factor in its performance. This would also allow me to greatly simplify the agent’s subsumption architecture into a few actions that focus solely on the problem that this system is trying to solve.

Firstly, this new agent would be situated at the top of every tree, removing the case where the old fire unit couldn’t move to the trees location in time and removing the need.

Since forest fires are catagorised as a class A fire [5], it is also essential that the material used to combat the fire is revised, to hopefully decrease the amount of time it takes to extinguish the fire. According to fire extinguishers 101, there are several ways a class A fire can be fought. One of which is to suffocate the fire, removing the presence of oxygen, by dousing the fire, and in this case the tree to, with foam. It is also mentioned that you can combat a fire through the use of air to remove as much heat as possible from the fire, but I feel that this could increase the risk of blowing the fire onto other trees, aiding the ignition of further fires.

Trees dowsed in foam can then subsequently be rinsed off using water once the fire has been

extinguished.
Finally, while there are no fires to be dealt with, the unit can ensure it has an adequate supply of the foam substance used to combat fires, being the highest priority action in its architecture.A possible subsumption architecture for this agent would look like the following illustration:


if detect fire then spray foam
if no foam substance then refill from central supply
Percept
action

By reducing the number of available actions, this makes the maintenance of the architecture much simpler. The agent can now be likened to a simple control system, much like the thermostat example [6], which checks to see whether the conditions are correct, and if not, applying an action to return the environment back to an “OK” state.

Finally, it is important to note that this proposition also eliminates any need for communication between itself and another agent, eliminating any conflicts. One noteworthy conflict that could occur, in regard to my previous propositions, would be; given a fire that cannot be contained by one agent, and that agent asks for help from another agent, the other agent can choose to not help because it is not in his best interest to leave the fire he is currently attending to, in order to help the agent in need. I have removed the possibility of this occurring by removing the need for communication and by constraining each agent’s immediate environment to a single tree.

Task 4

The current environment the agents inhibit is, in terms of the relevant information[3], is accessible. Based on this, all of the reactive agents decisions in this environment will be fully informed, without needing to update any internal representation of the world and derive information with uncertain data. Furthermore, the environment is also dynamic in the sense that other fire unit agents act within this world, as well as the trees themselves. However, since the environment is accessible, this won't have a negative impact on performance due to up to date relevant information being immediately available.

For example, when a tree is no longer on fire, our agents perceives that information and understands that it no longer needs to interact with the tree, regardless of whether the tree has died or been saved. In addition, the agents know it must return to the base to refill on water when its supply has run out. Therefore, based on the primitive requirements of the environment, the reactive approach is well suited.

On the other hand, there are some aspects of the world which some may argue that the reactive approach isn't the best approach to take. For example, if a particular set of trees were prone to catching fire at a certain time of year, for whatever reason, a reactive agent doesn't have the capability to remember that kind of information. They can only handle information in the here and now.

Moreover, due to this stateless nature, any new problems that the agent must face, needs to be hardcoded into the workings of the agent [7] since reactive agents are unable to infer new ways of solving current or new problems. If the environment were to change or develop in some way, our agent's reasoning would need to be updated to accommodate these changes.

Another environment where a reactive agent could be used is in card games, most notably in BlackJack. BlackJack has two main actions an agent could perform, hit and stick. Since all cards from all players are face up, the agent can fully observe the environment and make an educated decision on whether it should hit, receive another card and possibly go bust, or stick. Using a mixture of percepts, such as the hands of other players, the value of the agent's hand and the value of the dealer's hand, the agent could make a decision on how to play. For example, if the agent has a hand of 18 and there is a player with a value of 19, based on those percepts, the agent should stick because it is likely to bust and lose money.

In addition, this world is definitely deterministic and discrete, with respect to the number of players at the table, meaning a reactive agent could be built for this domain relatively easily. This is all under the assumption that card counting does not take place. If it were to take place, a stateful agent would be a better option as it would be able to infer what cards are yet to come based on what has already appeared, increasing the odds of the agent winning.

[3] Locations of fires, obstacles that may appear in front of the agent, amount of water remaining.

References

[1] Wooldridge, Michael J. *An Introduction to Multiagent Systems*. New York: J. Wiley, 2002.

[2] By Howard Schultz, with Joanne Gordon, 2011. "Useful Performance Measures & Metrics – How To Measure Efficiency & Effectiveness." *Michael VanBruaene*. 16 Mar. 2015. Web. 21 Oct. 2016.

[3] "NetLogo Help: Ticks." *NetLogo Help: Ticks*. Web. 21 Oct. 2016.

[4] Garber Metrology. (2015). *Gas and Water: In Equal Measure.*Available: http://www.garbermetrology.com/gas-water-which-is-heavier/. Last accessed 22nd Oct 2016.

[5] Fire Extinguisher 101. (Unknown). *Class A Fires: How to Fight Them.*Available: http://www.fire-extinguisher101.com/class-a-fires.html. Last accessed 23rd Oct 2016.

[6] Woolridge, M. (2002). Intelligent Agents. In: Woolridge, M *An Introduction to Multiagent Systems*. 2nd ed. West Sussex: John Wiley & Sons Ltd. p17.

[7] Alechina, N. and Logan, B. (2013) *Designing Intelligent Agents*. Available at: http://www.cs.nott.ac.uk/~psznza/G54DIA/lecture7-hybrid1.pdf (Accessed: 23 October 2016).