Compiling CDC/WHO/etc COVID-19 Recommendations into Interactive Behavior Trees Accessed Via Cell-Phone (DRAFT 1)

Andrew Dougherty
FRDCSA Project
adougher9@gmail.com

Abstract

CDC/WHO/etc have a last mile delivery problem in spreading efficacious knowledge and procedures to deal with COVID-19 to citizens, in order to save lives by preventing the spread of the pandemic and dealing with its consequences. Why not compile the recommendations into a *Behavior Tree* specification that can interactively walk people through tasks from their cell phones? For instance, the user enters into the interface (with autocompletion) some task, such as 'go to the grocery store'. The corresponding procedure is found, and it walks the user through this procedure, like an interactive checklist which can branch based on results of tasks. The sources of parts of rules can be included, in order to establish more trust, and users may import rules and edit their Behavior Trees to suit their unique circumstances. As rules become updated or changed, they can be automatically downloaded and reimported. We hope somebody better positioned either picks this Behavior Tree idea up and runs with it or helps us complete it to aid people to better cope with COVID-19.

Introduction

Getting the right information about COVID-19 to people at the right time is a life-critical problem.[1] What is clearly needed is some vehicle to deliver procedural information at the right time in order to deal with daily procedures altered by COVID-19[2]. We propose that the user be interactively walked through common procedures, such as going to the grocery store, receiving mail, etc, in such a way that the user is presented with the safest best practices for that procedure interactively from their cell phone.

If people used this program, the chance of them contracting the disease might lessen. Furthermore, people could better care for individuals in their house who have the disease, by following the CDC/WHO guidelines interactively. It would help to advise them regarding whether they should travel. It would help them to restock their pantries. It would help them follow proper cleaning and decontamination procedures.

Figure 1: Sample Interactive Plan Monitor

By compiling the recommendations into an executable format, we reduce the amount of work necessary to bring the relevant information to bear when applicable, increasing public safety and saving lives. Procedures may automatically updated to reflect the developments and changes to the best practices.

We propose using the well-known technique of *Behavior Trees* (from Robotics and Game AI) to represent the procedures. This representation offers imperative reactive "planning," "planning" with sensing, and interactive plan monitoring capabilities.

[1] A user might not be able to find the relevant information (in time to be applicable) even if they have web access. They may have forgotten the information. They may be under time pressure. They might be in the middle of some situation they hadn't anticipated. As an example, you might see how long it takes you to find out the recommended temperature at which point you may be considered symptomatic of COVID-19.

[2] such as helping to reduce transmission of COVID-19, quarantine and care for people in your house who contract it, resupply your house without transmitting it, and all other aspects of life affected by the disease.

Example Run-Through

Here we provide a sample run-through of a Behavior Tree system.[3]

Suppose the user enters into their cell phone (with auto-completion) the goal: "Go to the grocery." The system would have a behavior tree node for this. Figures 2 and 3 are two (here, incomplete) and alternate Behavior Tree specifications for the task of going to the grocery.

```
buy_groceries -->
    start_make_all_preparations,
    make_shopping_list,
    print_shopping_list,
    clear_staging_area,
    ensure_replete,
    end_make_all_preparations
    ensure_after_time_23_30_00,
    put_on_gloves,
    %% ...

put_on_gloves -->
    wash_hands,
    %% ...

wash_hands -->
    %% ...
    .
```

Figure 2: Example SimGen Behavior Tree

```
(   buy_groceries(Person)) ==>>
    begin_state(make_all_preparations),
    act(make_shopping_list(Person,List)),
    true_code(isa(List,shoppingList)),
    act(print(Person,List)),
    act(clear_staging_area(Person)),
    ensure_state(replete(Person)),
    end_state(make_all_preparations),
    true_code(currentTime(Time)),
    ensure_state(after(Time,[23:30:00])),
    act(put_on_gloves(Person)),
    %% ...

(   put_on_gloves(Person)) ==>>
    act(wash_hands(Person)),
    %% ...
    .

(   wash_hands(Person)) ==>>
    %% ...
    .
```

Figure 3: Example NomicMU Behavior Tree

The system would then proceed to follow the nodes of the behavior tree via an in-order-traversal. The system thus presents the current goal any constraints the user should be obeying (such as do not leave the house).

```
donny wants to> buy groceries
donny should begin making preparations
donny should make a shopping list
donny should print shopping list
donny should clear the staging area
donny should eat
donny should have finished making preparations
donny should wait until 11:30 PM
donny should wash hands
```

- Submitted from any mode to system:
  donny wants to>buy groceries
- Message sent to IEM to "donny begin making preparations."
- donny confirms the following prompt in the web-app:
  donny begins making preparations.
- The IEM system updates the world state, and proceeds with the "real-time simulation:"
- Message sent to IEM to "donny should make a shopping list."
- donny confirms the following prompt in the web-app:
  donny makes a shopping list.
- The IEM system updates the world state, and proceeds with the "real-time simulation:"

Node Types

Note that the examples here[4] do not implement any node type besides sequencers and leaf nodes, but this is easily possible and in fact the entire reason for not simply using checklists.

Strengths and Limitations of Behavior Trees

It is outside the scope of this paper to provide a thorough introduction to Behavior Trees. We refer the user to the following article: https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

We now compare Behavior Trees to other approaches such as BDI[5] agents and PDDL[6] planning. Behavior Trees appear to be a form of "imperative reactive planning-with-sensing, and execution." They are imperative in the sense that, rather than employing some form of declarative planning, where the rules are spelled out and the system infers a plan, you must essentially construct the plan yourself. For instance, in PDDL, you specify the operators and facts of the domain and the planning algorithm generates the plan. With Behavior Trees, you must edit them so that you get the desired behavior. They are therefore less deliberative than PDDL. In lieu of a generating the best possible plan via a

[3] https://frdcsa.org/~andrewdo/projects/flp-screencaps/

[4] due to time constraints

[5] https://tinyurl.com/v6hzdre

[6] https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language

deliberative planning system, they simply provide *a* plan. Having *a* plan is better than not having a plan at all. But Behavior Trees can lead you into a dead-end.

Conversely, PDDL has major limitations, such as:

- requiring perfect information
- inability to deal with uncertainty
- generally unable to sense or react outside of certain specialized planners
- inability to fail gracefully when no plan may be found
- inability to create new objects at run-time
- inability to modify the goal stack at run-time
- inability to have nonlinear/branching plans
- lack of inborn plan execution monitoring

BDI agents such as Spark provide both declarative and imperative features, but we have not been successful in implementing agents with a BDI agent development system such as Spark, Jason or GOAL. Peleus[7] implemented a planner combined with a BDI agent.

However, we have made more headway with Behavior Trees. NomicMU[8] implements planning and Behavior Trees using several planning systems such as the Event-Calculus Planner and Marty's Planner. Behaviac also implements HTN Planning, Finite State Machines and Behavior Trees.

Behavior Trees also implement execution monitoring as an automatic by-product. Whereas PDDL interactive execution monitoring is very difficult, involving checking and propagating complex preconditions and effects, respectively, as well as complex execution managers like PLEXIL. With Behavior Trees you get that basically for free.

Web Interface

We advocate for developing a web interface which uses WebSockets, so that triggers such as alarms and timers are able to push to the client. We already have one web interface under development using the Free-Life-Planner[9], but that project has a lot of technical debt and cannot be released in a timely fashion. Therefore we advocate for developing another web interface. Prolog affords a web interface with support for WebSockets.

Another feature desirable in the web interface would be individualized cell applications that wrap the interface, such as could be made with Apache Cordova.

Attempted Implementation

We implemented some rough pseudo-code[10] translating some portions of an earlier version of the CDC guidelines document called "Interim Guidance: Get Your Household Ready for Coronavirus Disease 2019 (COVID-19)"[11] for COVID-19 into Behavior Trees in a Prolog format.

We attempted to develop such a domain using the SimGen and NomicMU systems. We ran into issues while under time pressure that we have not yet solved using each approach.

SimGen Implementation

SimGen[12], developed by Simularity LLC, is a Behavior Tree implemention for Prolog. SimGen has the advantage that the user can simply read the human-readable English and interpret it. (see Figure 4). Originally designed to run Partial Differential Equation simulations, it is not perfectly suited to our application. While some of the issues we had are superficial[13] and may have been introduced by our code which attempted to harness SimGen, others are deeper such as the lack of variables for terms.

Human-readability may lead to issues like being unable to posit constraints, such as "Do not leave the house." At the very least, the web-based interface should show red flags with all such applicable constraints, but better would be for violation of constraints to be detected during invocation of goals such as "Go to the grocery.", ideally before the user has initiated the plan.

The lack of planning is one possible shortcoming of a pure Behavior Tree approach. Ideally this would be remedied. As an experiment, we tried to remedy it in the NomicMU implementation, which we address in the next subsection.

NomicMU Implementation

We attempted to implement the domain using NomicMU with variables and "modalities."

NomicMU is ideally a system to simulate the world using Natural Language Understanding (NLU). Users can posit facts and rules via English and the system interprets it. NomicMU is not yet at an alpha release.

This approach tended to bog down too much in details, such as having to create objects that denoted individual surfaces to be cleaned. Clearly this is too much detail. We attempted to calibrate the detail required to hit the sweet spot between adequately conveying and covering the domain vs practicality of implementation. Thus we developed an overly detailed PDDL domain, to try to relax the amount of detail in. The domain is an attempt to eliminate sickness from exposure to fomites when using the bathroom.[14] [15]

Other Existing Behavior Tree Implementations

There are a number of existing implementations of Behavior Trees in languages besides Prolog, such as Python, JavaScript, C++ and Java.

[7] http://www.meneguzzi.eu/felipe/software.shtml

[8] https://github.com/TeamSPoon/NomicMU

[9] https://github.com/aindilis/free-life-planner

[10] https://frdcsa.org/~andrewdo/projects/3/coronavirus.pl

[11] https://www.cdc.gov/coronavirus/2019-ncov/community/get-your-household-ready-for-COVID-19.html

[12] https://github.com/simularity/SimGen

[13] A good addition to the language would be a 'try in sequence til one succeeds' node type.

[14] https://frdcsa.org/~andrewdo/projects/3/bathroom.d.verb

[15] https://frdcsa.org/~andrewdo/projects/3/bathroom.p.verb

```
root ->
    go_to_bathroom
    .

go_to_bathroom ->
    grab_cell_phone,
    walk_to_door,
    attempt_to_open_door,
    walk_through_door,
    close_door
    .

grab_cell_phone ->
    confirm_execution
    .

walk_to_door ->
    confirm_execution
    .

attempt_to_open_door ->
    {try
    {->
      {not open_door},
      {not unlock_and_open_door},
      {not smash_door}
    }}
    .
```

Figure 4: Sample SimGen Behavior Tree

Behavior3[16] is a JavaScript implementation of Behavior Trees and a Rule Editor that outputs to an intermediate representation using JSON.

There is already a game which allows the user to edit their own behavior trees. The game called Gladiabots - AI Combat Arena[17].

Ideally, an organization such as a game development studio or a tech giant more experienced with all aspects of Behavior Trees and large-scale production deployments could take this further than the author has with his limited skills and resources. Or, they could help us to finish the SimGen and/or NomicMU implementations and develop the web interface.

Compilation of CDC Recommendations into Behavior Trees

NLU-MF

NLU-MF (standing for Natural Language Understanding - Manual Formalization) is a system we have developed for converting recommendations to Prolog (or in this case our Prolog representation of Behavior Trees). [18] The idea is to process the text into sentences in a machine-readable format and allow the user to construct the Behavior Trees in that format.

Rule Provenance

Another problem to be addressed is advice is often blended together to make rules. A future version of the software might include better provenance of rules, it has been suggested that semi-ring annotations[19] used in databases would be a state-of-the-art method to track how sources are combined into rules, to allow for some form of auditability (e.g. is this just hearsay) in order to increase public confidence in the rules.

Shared Priority System Editor v2 (SPSE2)

We also used a tool we developed called SPSE2[20](**?**) to begin generating plans for our own needs that we could later abstract into Behavior Trees.

```
MAKE_SHOPPING_LIST
PRINT_OUT_SHOPPING_AND_INSTRUCTION_LISTS
CLEAR_OFF_DINING_ROOM_TABLE
EAT_BEFOREHAND
MAKE_ALL_PREPARATIONS
WAIT_UNTIL_11_30_PM_TO_LEAVE
PUT_GLOVES_ON_BEFORE_LEAVING_HOUSE
BRING_WATER_BOTTLES_TO_REFILL
WEAR_MASKS
LEAVE_HOUSE
GET_IN_CAR
CHECK_GAS_LEVEL
DRIVE_TO_WALMART
ARE_LOTS_OF_PEOPLE_THERE
ARE_THEY_OUT_OF_INVENTORY
WALK_INTO_WALMART
START_SHOPPING
GET_EXTRA_5_GAL_JUGS
USE_SELF_CHECKOUT
FINISH_SHOPPING
DRIVE_HOME
PUT_FOOD_IN_STAGING_AREA
DECONTAMINATE_FOOD_WITH_BLEACH_SOLUTION
REMOVE_GLOVES_AND_MASKS
DISPOSE_PROPERLY_OF_GLOVES_AND_MASKS
```

Conclusion

Given the difficulty of containing COVID-19 and caring for people affected and the difficulty of following best practices, it seems reasonable that a real-time interactive system to help people adhere to the published guidelines, if possible, would be a major boost to containment and treatment efforts. Again, we hope that a better positioned organization can pick up the idea and run with it, or help us complete our work with SimGen and NomicMU.

[16] https://github.com/behavior3

[17] https://gladiabots.com/

[18] https://frdcsa.org/~andrewdo/projects/3/interim-guidance.txt.nlu.pl

[19] https://users.dcc.uchile.cl/~pbarcelo/KG.pdf

[20] https://frdcsa.org/visual-aid/pdf/SPSE2.pdf

```
sanitize_surface(Person,Surface) ==>>
  ensure_state(wearing_gloves(Person,_Gloves)),
  true_state(in(Surface,Room)),
  true_code(isa(Room,room)),
  ensure_state(well_ventilated(Room)),
  true_state(holding(Person,HouseholdCleaningSprayOrWipe)),
  true_code( isa(HouseholdCleaningSprayOrWipe,cleaningSpray)),
  act(read_label(Person,labelFn(HouseholdCleaningSprayOrWipe))),
  act(use_as_directed_on(Person,HouseholdCleaningSprayOrWipe,Surface)).
```

Figure 5: NomicMU Behavior Tree (with Variables and Modalities)

```
act(self_check_for_covid19_symptoms_is_positive(Person)) ==>>
  act(check_temperature_fahrenheit(Person,Temperature)),
  sources([s(d('https://www.cdc.gov/coronavirus/2019-ncov/downloads/COVID-19_CAREKit_ENG.pdf'
             s([56]))])),
  (   Temperature > 100.4 ->
      (
       declare(h(has_illness,Person,fever)),
       declare(h(is_symptomatic,Person,covid19))
      )).
```

Figure 6: Behavior Tree with Source Annotations, Conditional Rules and Knowledge-Base Interaction

Acknowledgements

I am grateful especially to Douglas Miles for his work on LogicMOO/NomicMU/PrologMUD and his mentorship with A.I.. I am also grateful to Simularity for making SimGen available, Anne Ogborn for helping advance FRDCSA projects such as the Free Life Planner, Jess Balint for his help with the Free Life Planner, and to my partner Meredith McGhan for her loving kindness.

References

Dougherty, A. J. 2011. Temporal planning and inferencing for personal task management with SPSE2.