You should write a one page README describing how you structured your model and what your model proved. You can assume that anyone reading it will be familiar with your project proposal. Here are some examples of points you might cover:

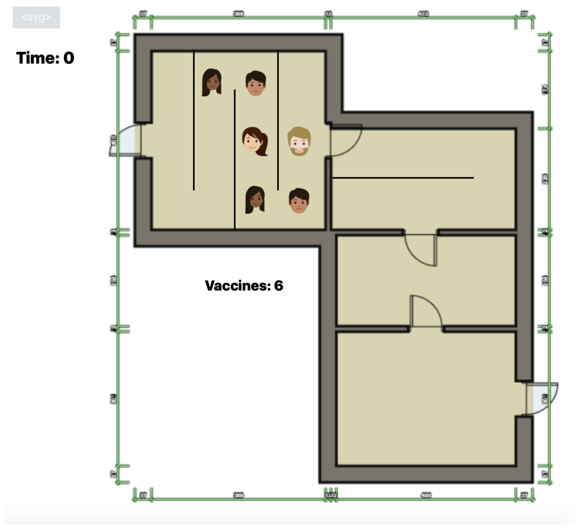
- **How should we understand an instance of your model and what your custom visualization shows?**

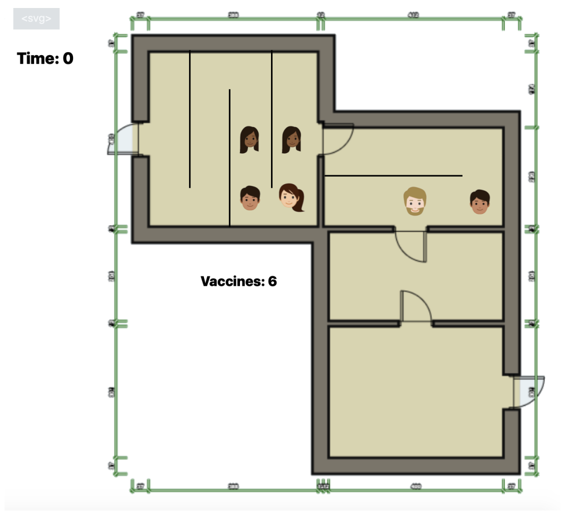
- A single instance of our model will display the locations of every person in all the rooms, as well as a Clock that indicates how much time has passed since the beginning. There are four rooms: the ballpark, the waiting room, the vaccination room, and the observation room. People flow through the vaccination site in that order. The people shown in the visualization are in queue order, and you will notice they preserve that order as they move through the vaccination site. Each room has a maximum capacity, so you will notice that with many people, the amount of people in each room will be limited.

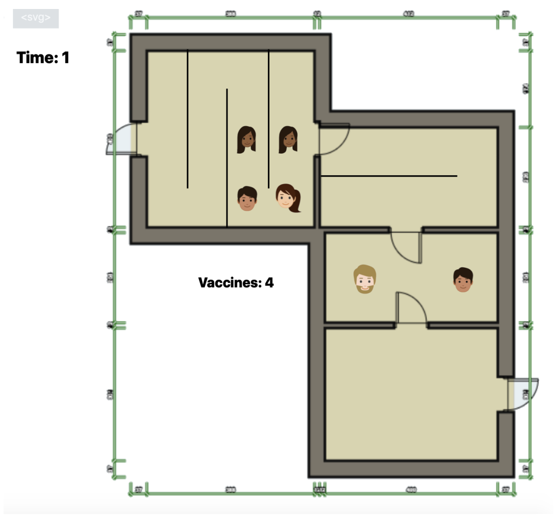
- As you step through the time-states, you will see people move through the rooms and the clock going up, subject to the constraints provided. Some movements, like moving from the ballpark to the vaccination room, will not increment the time; but getting a vaccination and staying in the observation room will. When the amount of vaccines is zero, the people in the room will have to wait 3 clock increments for 6 more vaccines to be produced.

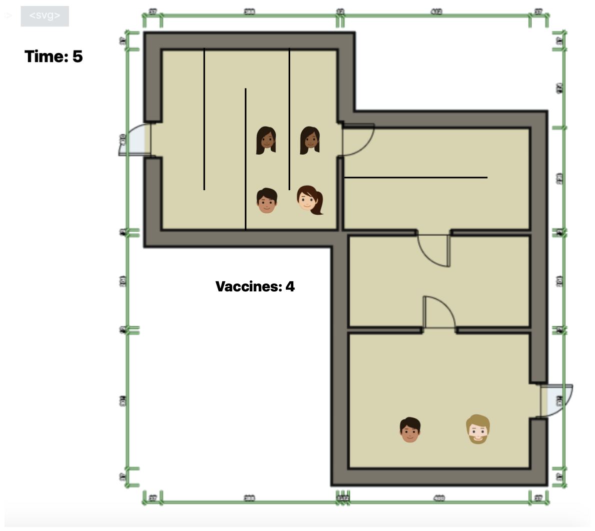
- As such, one example walkthrough with the custom visualization will go as follows: load up the first state, and any people shown will be in the ballpark. people will either join the line in the ballpark, or move into the waiting room.  From here, we will follow the journey of a single person at the head of the queue, Nim. When Nim is in the waiting room, he will continue into the vaccination room if there is space. Nim must wait in the vaccination room until the Clock timer increments by 1 and the amount of vaccines decreases (to represent receiving his vaccine), then he can continue into the observation room. For 4 states, Nim must stay in the observation room (to represent the observation period after receiving the vaccine). From there, Nim is free to exit and will leave the visualization.

- Some screenshots from a hardcoded trace of just two people moving. The last two images were selectively chosen directly after both had received their vaccine, and after they had completed their time in the observation room. You will notice that the timer went up and the amount of vaccines went down after they received their vaccine.









- **What tradeoffs did you make in choosing your representation? What else did you try that didn't work as well?**
    - Have multiple people move at the  same time vs. at most one person move in a time step (except from vac room to observation room, when there is no enforcement of a queue). We chose the latter. We thought the former might help us decrease trace length, but exactly how to allow multiple people to move at the same is not clear, as the movement of one person might affect the preconditions of the movements of other people (especially since we put people in a queue), and in electrum the solution has been to treat them as separate state transitions.
    - One think that might help to implement is to have all the state transition constraint take in a Person as a parameter; so, instead of saying "can we move the first person in waiting room to vac room", we say "can we move this person p from waiting room to vac room"; and then, at each step in the trace we ask all people who could move to move. However, there are several problems with this method:
        - Under this model, at each time step we have to check each person, which might take a large amount of time.

        - The allowed movements of a person might depend on the movement of the person before them, which means we might need recursive calls to predicates, and this does not seem to be doable in Forge/Electrum.

- **What assumptions did you make about scope? What are the limits of your model?**
    - Assumptions
        - We assumed that people have to be in a queue throughout the whole process. We assumed that the order of this queue is predetermined, even before people came in. But we do not lose any generality as people are symmetric objects in our model.
        - We assumed that if the ballpark was full, people would be able to wait outside. Theoretically, this means the line in the ballroom would be infinite— not contained by the ballpark capacity. However, to keep track of the "people outside", as they wouldn't be displayed, we have a NextPersonTracker to identify the head of the outdoor queue.
        - We assumed that people take no time to move to different rooms.
        - We picked some numbers for the waiting times in the observation room, room capacities, and the total number of people.
        - We assumed that the vaccine shot takes 1 clock cycle to be administered.
        - We assumed that the vaccine takes 3 clock cycles to be prepped, and the maximum number of available vaccines is 6. This assumption mimics the situation that the vaccines can expire, and we shouldn't waste any by having the vaccines sitting on the shelf.
    - Limitations
        - Clock cycle: we use doNothing to increase the clock cycle. So the time constraints for each room have to be multiples of the cycle, and can't just be arbitrary numbers. If we change the time constraints (e.g. waiting for 20 minutes in the observation room instead of waiting 10 minutes), we'll have to hard code to add several doNothing, instead of a simple change of updating a number.
        - Trace length: trace length builds up quickly, since most room transition predicates only allow one person to move at a time. We didn't know that Forge slows down significantly when the trace length reaches 20, which would only allow very few people to go through the whole process, and we couldn't experiment with larger examples.

- **Did your goals change at all from your proposal? Did you realize anything you planned was unrealistic, or that anything you thought was unrealistic was doable?**
    - Foundation goal: visualize the process of how a flow of people get vaccinated, and how much total time they spent waiting

        - Achieved. We are able to visualize the process of moving through the vaccination site by entering a hardcoded instance and using our visualization script to observe the process.
        - However, as explained in the roadblock section below, we are unable to use our model to produce instances of people moving through the site.
    - Target goal: model how many people can get vaccinated in a given time (e.g. if people keep coming to the site, how many vaccines can be given in 2 hours)
        - Unfortunately, due to our not being able to produce trace instances using our model, we were not able to achieve this goal.
    - Reach goal: if we change the capacity of waiting room or observation room, would it speed up the process, if so, can we optimize the capacity number
        - Unfortunately, due to our not being able to produce trace instances using our model, we were not able to achieve this goal.

- **Roadblock faced**

- Unfortunately, many of our goals were not achieved due to Forge not succeeding in finding traces for our model. Below is a description of what we think might be going on, and what we have tried to fix it.
- When we try a run to produce a solution trace, Sterling times out without finding a solution. This holds even when the number of people in the run is decreased from 10 people to only 1 person.
- We can verify that a hardcoded trace satisfies our mode and our general “always” trace specification, so we know that there are expected-looking traces that are consistent with our model.

```
pred traces{

   init

   ballToWaiting
   after waitingToVac
   after after (doNothing and doNothingGuard)
   after after after vacToObs
   after after after after (doNothing and doNothingGuard)
   after after after after after (doNothing and doNothingGuard)
   after after after after after after (doNothing and
doNothingGuard)
```

```
   after after after after after after after (doNothing and
doNothingGuard)
   after after after after after after after after obsToExit
   after after after after after after after after after always
(doAbosolutelyNothing and no people and no
NextPersonTracker.nextPerson)
}
```

We actually ran this hardcoded trace, and in Sterling evaluator put in our general trace specification

```
   always (ballToWaiting or waitingToVac or (doNothing and
doNothingGuard) or vacToObs or obsToExit or
(doAbosolutelyNothing and no people and no
NextPersonTracker.nextPerson))
```

Which evaluates to True.
We also tried  running the hardcoded trace with this general trace specification, and still got the same valid trace.

- The above attempts tells us that there is indeed a valid trace, and Forge might be able to find it quickly in some cases, but never gives us any trace in Sterling before timing out if we run the general "always" specification.

- One theory for what is going wrong is that the trace length is too long, and that we should try to stick to less than a ~15 trace length. Our trace length would be long because we have one state transition whenever one person moves from one room to the next.
    - However, with only 1 person, the expected trace length is less than 10. Therefore, this should not be the issue.
    - Decreasing our number of state transitions by letting multiple people at once would be complicated, as described under tradeoffs above. Because the run fails with even one person, we did not feel confident enough that decreasing traces would work to justify re-doing our model in a more complicated way through this approach.
- Another theory for what is going wrong is that Forge is not able to find a lasso trace, and because in Electrum these are the only kinds of trace that Forge can find, we are not getting a solution.
    - However, we have the doAbsolutelyNothing predicate as a stutter step to yield lasso traces and yet we still face the problem. Further, we also know that our hardcoded trace is indeed an infinite lasso trace (which also satisfies our general "always" trace specification).

- Another theory for what is going wrong is that the search space for transitions is too large.
    - However, we added a guard to the doNothing transition so that it only happens when no other transition is possible. This should limit the search space to close to linear in the one person case with respect to the number of time steps, because at each step there should only be one available transition. Thinking of the search space as a tree, only one full path is a valid trace in this case. Any other branch is invalid *as soon as it deviates from the correct path*, and so they do not have to be followed far before it becomes apparent that they are invalid paths. Therefore the search space should be very tractable in this case, so we are fairly confident that this should not be the problem.
- We are left thinking that there may be a problem with Forge or Sterling that is preventing finding traces. One piece of evidence in favor of this is that at one point we were able to get a quickly terminating test expect showing that a trace can be found, but we were unable to have a run statement terminate on exactly the same specification. This is documented below.
- When running a test-expect block, we could verify that Forge was able to find a solution trace for our model quickly. Therefore, we knew that finding solutions for our model is reasonably tractable.
    - A detailed description of what happened:
    - We ran the following code and it terminated within seconds, giving us SAT:

```
test expect {
   thm: {
     (init and always (ballToWaiting or waitingToVac or
(doNothing and doNothingGuard) or vacToObs or obsToExit or
(doAbosolutelyNothing and no people and no
NextPersonTracker.nextPerson)))
   } for exactly 1 Person, 5 Int is sat
}
```

    - We ran the exact same predicates, but in a run statement, then it timed out without terminating:

```
run{
   init and always (ballToWaiting or waitingToVac or
(doNothing and doNothingGuard) or vacToObs or obsToExit or
(doAbosolutelyNothing and no people and no
NextPersonTracker.nextPerson))
} for exactly 1 Person, 5 Int
```

- The test-expect does not terminate so fast any more after we made some modifications, but what happened earlier as we described here is so bizarre that we think it might tell us something interesting about Forge / Sterling itself

Notes for testing:
Because some of the test case take a long time to run, we suggest running one test expect block at a time (and comment all others out).