File structure:

* Run.R: run simulation and post-simulation calculation
* Inputs.R: default input parameters
* Main\_file.R: set initial attributes, event registry, counters, global functions, execution function
* Events\_simple.R: event functions
* Event\_secular\_death.R & age\_gompertz.R: secular death event
* Event\_main\_loop\_simple.R: setup functions
* Costs\_simple.R: functions for post-simulation computation

Simple scenario template:

* A population of 40-year old women are at risk for both secular death and an event (A) that happens at a **10%** rate over a 10-year period;
* All those who experience the event incur a cost of $10,000;
* Those who experience the event experience a **0.05** utility decrement for 1 year, and are placed on a drug for life that costs $0.5/day;
* There is a second event (B) that occurs downstream of Event A, with probability **0.02** over a 1-year period;
* Event B has a 5**% case fatality rate** **with a $15,000 cost among the decedents**and, among the survivors, incurs a $25,000 cost and a **0.1** disutility for life;
* There is a genetic test available that, if the person tests positive for the gene (**prevalence is 0.2**), results in an alternative drug (for life) that costs $5/day but reduces the rate of the second event by a relative risk of **0.7**;
* Event A and Event B are not recurrent;
* We are going to compare two strategies: “None” (no genetic testing) vs “Reactive Single” (genetic testing at the time of Event A indication).

Setup of a DES based on this template needs the following steps.

**Step 1: Blueprint the structure**

The first step is to plan what kind of events and parameters to include and how they interact with each other.

At the beginning of the simulation, each subject is assigned a set of initial attributes and time to each event is drawn based on risks parameters and user-defined functions. Over time, attributes can change and interact with events. Time to each event is also treated as an attribute and can be modified. The simulation will process events in the order of the time attribute. In this simple case, we need to define event A and event B. Because event B is downstream of A, we can create an attribute capturing whether a subject has experienced A or not acting as a condition to modify time to event B. Similarly, we need to prevent reoccurrence of event A and B. Another interaction is the treatment. Depending on the strategy and the gene attribute, a subject can be assigned standard or alternative treatment. Therefore, we need to specify in event A the algorithm for drug assignment and modify downstream risks.

To capture the process of the simulation, users need to define counters and then the simulation will generate one record every time when a counter is called. For example (Figure 1), to track how long a subject stays in the simulation, we can define a counter named “time\_in\_model”, “seize” it at the beginning, and “release” it at the end of the simulation. We have also defined a function called “mark” to seize and release at the same time to capture one-time events.

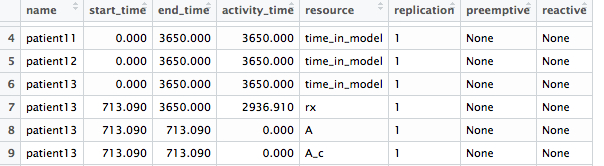


Figure 1

The simulation can return an event trajectory (Figure1) and attributes (Figure 2) for each subject. Computation of costs and QALYs is done externally afterwards. We primarily use the event data to count adverse events and sum up costs and QALYs. Our template differentiates between temporary and permanent effects in cost and utility, which requires different setup in input parameters (see Step 2).

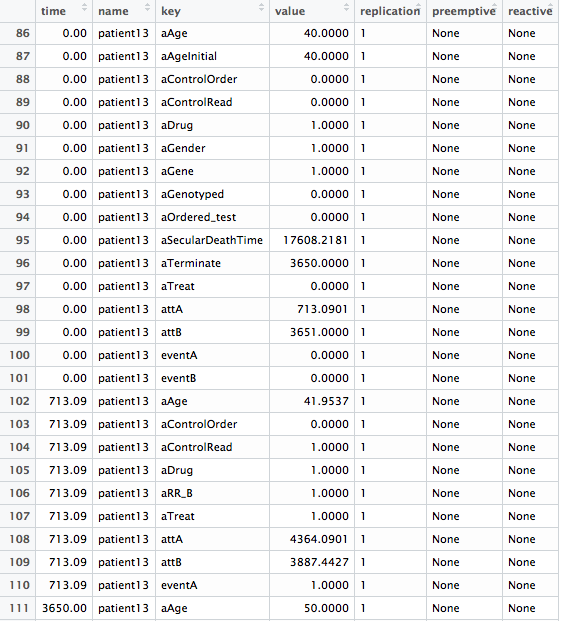


Figure 2

**Step 2: Set input parameters**

In the inputs.R file, we compile these default inputs into one list:

* Used in simulation:
  + Global control: time horizon, size
  + Demographics
  + Event risks
  + Other parameters that affect time-to-event (strategies, behavioral parameters, etc.)
* Used in post-simulation computation
  + Event disutility & cost
  + Event type (temporary vs. permanent)

As long as demographics affect risks/time-to-event, they should be included. In our example, age and gender affects secular death event and prevalence of a certain genetic variant interacts with treatments.

Event B permanently reduces utility, so we simply record the cost and disutility and specify its type as “permanent”. Then it incurs a one-time cost and disutility for the rest of time in simulation. In contrast, Event A temporarily reduces utility, so we specify its type and duration of disutility, and the post-simulation computation will only carve out the disutility for that duration. The cost aspect here is a bit tricky. There are two options: If we just record the cost, it will also count over the same duration, so we can apply a daily cost, which is how we handle drug cost; another way is to have two event counters, one as a temporary event that only affects utility, the other as a permanent event that only incur an one-time cost. For an adverse event, the second way is more accurate with discounting.

**Step 3: Define and register events**

We define event functions in the events\_simple.R file. Each event needs two functions: one decides time to event; the other specifies the effects of the event. Some commonly used building blocks include:

* “set\_attribute” function sets or modify attributes; [[1]](#footnote-1)
* subsetting code “attrs[[“*attribute name*”]]” can access value of attributes in the simulation;
* “branch” function assigns subjects to different sub-trajectories so that we can introduce heterogeneity into a single simulation; [[2]](#footnote-2)
* “seize” and “release” functions capture the start and end of a counter (usually an event);[[3]](#footnote-3)
* “mark” function is simply a wrap-up of “seize” and “release”, very handy for one-time events

Take Event A as an example. The “days\_till\_A” function returns time to event A based on risk parameters from the input list as well as an attribute that tells whether the subject has experienced A or not. If the value is 1, event time will be set over the simulation time horizon, which prevents reoccurrence of Event A.

The function “event\_A” modifies attributes, calls branching, and activates counters. The “prescribe\_drug” function shows a simple example of branching. We define the “aDrug” attribute as 1 representing standard treatment and 2 as alternative treatment, and subjects can take different values based on their genetic type and the strategy. The first argument of the branch returns the value of “aDrug” and then the function divides into two sub-trajectories. Arrivals with the value of 1 will seize the counter for the standard drug (“rx”), and those with 2 will seize the alternative drug correspondingly. Another important trick is the last “set\_attribute” call that modifies “attB”, which tells the simulation to redraw time to event B given the current conditions. In the simulation, event times are only automatically redrawn once the event is triggered. Before the presence of event A, event B is scheduled to trigger beyond the simulation time horizon and will never be re-evaluated unless specifically told so.

After setting up all events, we need to fill out the event registry and the counter list in main\_file.R. The “attr” is the name of the time-to-event attribute. The “time\_to\_event” and “func” take the name of the two functions we define for each event. The “reactive” element should be set as FALSE for a majority of cases. [[4]](#footnote-4)

For the “counters” list, we need to include all counters that are seized or marked in the event file.

**Step 4: Assign initial attributes**

As mentioned above, an attribute must be defined at the beginning either through events or initial setting. In main\_file.R, we assign those initial values in the “initialize\_patient” function.

**Step 5: Test and debug**

It is helpful to run the “exec.simulation” function to test the simulation, starting with a relatively small simulation size. Another function “get\_mon\_attributes” can obtain all attributes, especially helpful in validating the model. Use of random seeds is recommended to obtain reproducible results.

**Step 6: Post-simulation calculation**

The costs\_simple.R file includes a summary function for computing discounted costs and QALYs, a function for computing ICER, and a ggplot function for cost-effectiveness plane graph. The raw event trajectory and attributes data from the simulation have lots of potentials for more detailed subgroup analysis and other customized measures. The run.R file is a template for simulating multiple strategies and compiling results.

1. Please note that in order to use an attribute within the simulation, it must be either defined at the initial stage (“initialize\_patient”) or event time variable registered in the event registry. Defining a new attribute in the middle of the simulation would not work. [↑](#footnote-ref-1)
2. In the “continue” argument within a branch, TRUE means to continue, and FALSE means the subject exits the simulation (usually a “death” branch for fatal event). [↑](#footnote-ref-2)
3. Another easy mistake to make is to seize one counter and then forget to release it. The simulation only captures counters that are both seized and released within the simulation time horizon. [↑](#footnote-ref-3)
4. This element was originally designed for a special type of event. If sets TRUE, the event will automatically redraw its time whenever any other event is triggered, which might not be relevant to most cases. [↑](#footnote-ref-4)