Purpose

The purpose of this assessment is to create a fully functional simulation of a more complex system. Building simulations is one of the key objectives of this course. You will need to be able to do this in an industry or government job working in decision sciences. Building a simulation is also the best way to absorb and gain a deep understanding of the ideas and topics that are discussed in this course. Even if you are just managing simulations projects, you should have some experience in creating the simulation itself.

This is quite a large and difficult task, but the assessment will provide considerable structure.

Outcomes

This Task addresses the following Course Learning Outcomes:

1. Communicate how randomness and controlled variation can be used to model complex systems in a range of application domains such as industry, health, and transportation.
2. Create a model of a real-world problem specified in words and implement it as a discrete-event simulation.
3. Validate results from a discrete-event simulation.

Scenario

You will be simulating a factory production line that constructs lawnmowers to order.

The construction process involves a small number of main steps. Each is a series of tasks, but you will only model the main bottleneck task. Presume that the mower is built from a series of prefabricated parts (the engine, the blades, the frame and so on).

The bottleneck in this process is that, in this factory, there is only one machine that can properly fasten the blades to the motor as they have to be attached securely to make them safe.

Unfortunately, the blade-fitting machine is old and breaks down sometimes. If this happens, it is out of service while it is being repaired, and any orders will have to wait.

Assumptions

You should assume the following:

- Orders for new lawnmowers are processed in a FIFO manner and there can be an unlimited number waiting.
- Times: all times are independent and
    - the inter-arrival times between orders are independent and exponential with a mean of one hour.
    - the time to construct a lawnmower from parts is deterministic with a time of 45 minutes.
    - the time between breakdowns of the blade-fitting machine is exponential with a mean of two days as measured from the last time it was repaired.
    - the time to fix the machine when it breaks is exponential with a mean time of three hours.
- There is an unlimited number of parts available to construct new lawnmowers.
- If the blade-fitting machine breaks down while constructing a lawnmower, the work already completed on that lawnmower is interrupted but ‘saved’. That is, the total time to complete the lawnmower is the construction time, plus any repair times that interrupt construction (it is technically possible that more than one breakdown could occur during the same lawnmower build).

Questions

The factory owner wants to know how to improve their factory, the most obvious change would be to buy a replacement machine to attach the blades. The new machine would break down less often, so the mean time between breakdowns will be

longer. They would like to assess the business case for such a purchase, i.e., what would the reduction in waiting time for orders be, and what percentage of orders would be interrupted with the new machine. Therefore, the questions they would like to answer are:

- How much production time is lost to repairs?
- How many lawnmowers have their construction interrupted due to a breakdown and repair?
- How long do orders wait in this system before being completed, and how much would this be improved if the time between breakdowns was extended?

Your Task

This assessment is scaffolded into three parts:

- Part 1: Conceptual model (Module 2)
- Part 2: Functionality (Module 3)
- Part 3: Programming (Module 4)

The details of each part are outlined in Appendix 1-3 of this brief, and can also be found at the end of Module 2, Module 3 and Module 4 of this course.

Requirements

You will be assessed on three components of this work:

- Component 1: Conceptual model–your ability to formulate the model.
- Component 2: Functionality–your ability to write code to create a simulation, and ensure the sub-components of it work.
- Component 3: Programming style–your ability to write your program according to the specifications and general style guidelines for good code.

You are required to submit:

- a PDF document showing your schematic, state diagram, flow chart and any other documentation you created.
- two *.jl* files with your code for implementing the discrete-event simulation.
- a pair of data files (an entities and a state file) produced from your simulation with seed=1 and where the simulation was stopped at time T=1000.0.

Consult the assessment rubric when preparing your submission.

Questions can be posted to the relevant assessment Discussion Board.

Grading Criteria

This assessment is worth 40% of your overall grade. Refer to the attached rubric for detailed information on the grading criteria for this assessment.

Component 1: Conceptual Model				
Criteria	**Ratings**			**Points**
Task 1 Draw a schematic of the system	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	Your schematic correctly shows the standard queue components and the breakdown-repair cycle in some way.	You have correctly shown either the standard queue components or the breakdown-repair cycle.	Assessment requirements for this step have not been met.	
Task 2 Describe the state(s) of the system	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	You have correctly described the state(s) of the system.	You have partially described the state(s) of the system.	Assessment requirements for this step have not been met.	
Task 3 Determine the entities in the system in relation to the state	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	You have correctly determined the entities in the system.		Assessment requirements for this step have not been met.	

Task 4 List the types of events in your model • describe how each event changes the state of the system • describe the new events that may be created as a result of this event	**Points:** 1.0 **Name:** Full points You have correctly listed the types of events in your model.		**Points:** 0.0 **Name:** No points Assessment requirements for this step have not been met.	1 pt
Task 5 Draw a flow chart illustrating your simulation structure	**Points:** 3.0 **Name:** Full points Your flow chart: (i) is a valid flow chart for this model, (ii) does not have any dead ends (except for the intended termination step); and (iii) includes vacation (machine breakdowns).	**Points:** 2.0 **Name:** Partial points You have successfully completed two of the following: (i) a valid flow chart, (ii) there are no dead ends (except for the termination step); and (iii) included vacation (machine breakdowns).	**Points:** 1.0 **Name:** Partial points You have successfully completed one of the following: (i) correctly drawn a valid flow chart, (ii) there are no dead ends (except for the termination step); and (iii) included vacation (machine breakdowns).	**Points:** 0.0 **Name:** No points Assessment requirements for this step have not been met.	3 pts
			Section total:	**9 pts**

Component 2: Functionality

Criteria	Ratings	Points
Task 1 Refine your schematic of the system (see Component 1: Task 1 for marks)		

Task 2 Draw a state-diagram of the system	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your state diagram shows the correct state transitions.		Assessment requirements for this step have not been met.	
Task 3 Write code to implement a discrete-event simulation of the system.				
Output state file: Initialisation	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	Your initialisation creates a valid starting state with an arrival at time 0.0 and a breakdown at time 150.0.	You have done one of the following: (i) created a valid starting state with an arrival at time 0.0, and (ii) created a valid starting state with a breakdown at time 150.0.	Assessment requirements for this step have not been met.	
Output state file: run! Initialisation	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your run! function can be called with the prescribed inputs and does not crash.		Assessment requirements for this step have not been met.	

Output state file: Random number generators	**Points:** 4.0 **Name:** Full points	**Points:** 3.0 **Name:** Partial points	**Points:** 2.0 **Name:** Partial points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	4 pts
	Your random number generators create suitable random numbers for: (i) inter-arrival times, (ii) construction times, (iii) breakdown times, and (iv) repair times.	Your random number generators have created suitable random numbers for three of the following: (i) inter-arrival times, (ii) construction times, (iii) breakdown times, and (iv) repair times.	Your random number generators have created suitable random numbers for two of the following: (i) inter-arrival times, (ii) construction times, (iii) breakdown times, and (iv) repair times.	Your random number generators have created suitable random numbers for one of the following: (i) inter-arrival times, (ii) construction times, (iii) breakdown times, and (iv) repair times.	Assessment requirements for this step have not been met.	

Output state file: CSV format and metadata	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	Your output state file conforms to the correct CSV format and contains satisfactory metadata.	You have done one of the following: (i) created an output state file that conforms to the correct CSV format, (ii) created an output state file that contains satisfactory metadata.	Assessment requirements for this step have not been met.	
Output state file: States	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output state file contains states that are sequentially ordered in increasing time.		Assessment requirements for this step have not been met.	
Output state file: Event IDs	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output state file contains unique event IDs.		Assessment requirements for this step have not been met.	
Output state file: Arrival and breakdown	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	Your output state file contains an arrival at time 0.0 and breakdown time at 0.0.	You have done one of the following: (i) created an output state file that contains an arrival time at 0.0, and (ii) created an output state file that contains a breakdown time at 0.0.	Assessment requirements for this step have not been met.	
Output state file: Queue	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output state file shows that the queue grows during the breakdown.		Assessment requirements for this step have not been met.	

Output state file: Events	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output state file shows that the event list always has 1 or 2 events in it.		Assessment requirements for this step have not been met.	
Output state file: Machine status	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output state file shows that the machine status is 0 normally and 1 after the start of a breakdown up until a repair.		Assessment requirements for this step have not been met.	
Output entity file: CSV format and metadata	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	Your output entity file conforms to the correct CSV format and contains satisfactory metadata.	You have done one of the following: (i) created an output entity file that conforms to the correct CSV format, and (ii) created an output entity file that contains satisfactory metadata.	Assessment requirements for this step have not been met.	
Output entity file: Times	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output entity file contains arrival times, start service times and completion times that are in increasing order.		Assessment requirements for this step have not been met.	
Output state file: Event IDs	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	Your output entity file contains entities with unique IDs.		Assessment requirements for this step have not been met.	
			Section total:	**21 pts**

Component 3: Programming Style					
Criteria	**Ratings**				**Points**
Specification 1 File names, functions and data structures	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points		1 pt
	Your file names are correct, and the correct code is in the correct place (functions and data structures in particular).		Assessment requirements for this step have not been met.		
Specification 2 Packages	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points		1 pt
	Your file names are correct, and the correct code is in the correct place (functions and data structures in particular).		Assessment requirements for this step have not been met.		
Specification 3 Data structures	**Points:** 3.0 **Name:** Full points	**Points:** 2.0 **Name:** Partial points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	3 pts
	You have three data structures declared in a valid manner with appropriate constructors.	You have two of the three data structures declared in a valid manner with appropriate constructors.	You have one of the three data structures declared in a valid manner with appropriate constructors.	Assessment requirements for this step have not been met.	
Specification 3 Event types	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points		1 pt
	You have included concrete event types.		Assessment requirements for this step have not been met.		
Specification 4 Machine breakdowns	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points		1 pt
	You have included extra delays for machine breakdowns in the update function for breakdowns.		Assessment requirements for this step have not been met.		

Specification 5 Update functions	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	You have included update functions for four types of events.	Assessment requirements for this step have not been met.	
Specification 6 Parameters	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	Your code has a data structure for passing parameters.	Assessment requirements for this step have not been met.	
Specification 7 Constructor	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	You have included a random number generator constructor.	Assessment requirements for this step have not been met.	
Specification 7 Specification	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	You have included an initialise function matching specification.	Assessment requirements for this step have not been met.	
Specification 8 run! function	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	You have included a run! function that runs the simulation.	Assessment requirements for this step have not been met.	
Specification 9 Entities file	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	Your program writes an entities file.	Assessment requirements for this step have not been met.	
Specification 9 State file	**Points:** 1.0 **Name:** Full points	**Points:** 0.0 **Name:** No points	1 pt
	Your program writes a state file.	Assessment requirements for this step have not been met.	

Specification 10 Variables	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	You have not included any global variables.		Assessment requirements for this step have not been met.	
Specification 10 Variable names	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	You have used meaningful variable names, with the exception of local variables with small scope.		Assessment requirements for this step have not been met.	
Specification 10 Comments	**Points:** 2.0 **Name:** Full points	**Points:** 1.0 **Name:** Partial points	**Points:** 0.0 **Name:** No points	2 pts
	You have used comments efficiently and effectively.	You have partially used comments efficiently and effectively.	Assessment requirements for this step have not been met.	
Specification 10 White space	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	You have used correctly and consistently used indentation and space to separate important components and expressions.		Assessment requirements for this step have not been met.	
Specification 10 Inefficient approaches	**Points:** 1.0 **Name:** Full points		**Points:** 0.0 **Name:** No points	1 pt
	You have used correctly and consistently used indentation and space to separate important components and expressions.		Assessment requirements for this step have not been met.	
			Section total:	**20 pts**
			Assessment total:	**50 pts**

Appendix 1

Part 1: Conceptual Model

In this part, you will consider the system model that you will be implementing.

The system

You will be simulating a factory production line that constructs lawnmowers to order.

The construction process involves a small number of main steps. Each is a series of tasks, but you will only model the main bottleneck task. You can presume that the mower is built from a series of prefabricated parts (the engine, the blades, the frame and so on).

The bottleneck in this process is that, in this factory, there is only one machine that can properly fasten the blades to the motor as they have to be attached securely to make them safe, so only one lawnmower can be made at once.

Unfortunately, the blade-fitting machine is old and breaks down sometimes. If this happens, it is out of service while it is being repaired, and any orders have to wait.

Assumptions

You should assume the following:

- Orders for new lawnmowers are processed in a FIFO manner and there can be an unlimited number waiting.
- Times: all times are independent and
    - the inter-arrival times between orders are independent and exponential with a mean of one hour
    - the time to construct a lawnmower from parts is deterministic with a time of 45 minutes
    - the time between breakdowns of the blade-fitting machine is exponential with a mean of two days as measured from the last time it was repaired
    - the time to fix the machine when it breaks is exponential with a mean of time three hours.
- There is an unlimited number of parts available to construct new lawnmowers.
- If the blade-fitting machine breaks down while constructing a lawnmower, the work already completed on that lawnmower is interrupted but ‘saved’. That is, the total time to complete the lawnmower is the construction time, plus any repair times that interrupt construction (it is technically possible that more

than one breakdown could occur during the same lawnmower build).

Questions

The factory owner wants to know how to improve their factory, the most obvious change would be to buy a replacement machine to attach the blades. The new machine would break down less often, so the mean time between breakdowns will be longer. They would like to assess the business case for such a purchase, i.e., what would the reduction in waiting time for orders be, and what percentage of orders would be interrupted with the new machine. Therefore, the questions they would like to answer are:

- How much production time is lost to repairs?
- How many lawnmowers have their construction interrupted due to a breakdown and repair?
- How long do orders wait in this system before being completed, and how much would this be improved if the time between breakdowns was extended?

Tasks for Part 1

Your assessment will require you to write code to simulate the system. However, writing the code can wait until next week when you will have seen more examples and been taught more of the tools required.

This week, before you commence coding you should perform a series of modelling tasks. This week's tasks will prepare you for Part 2. Some of this week's tasks will be assigned marks but some will be part of a larger task assigned marks in Part 2 and 3.

1. Draw a schematic of the system. (2 marks)
2. Describe the state(s) of the system. (2 marks)
   Hint: What state details are needed to answer factory owner's questions?
3. Determine the entities in the system in relation to the state. (1 mark)
4. List the types of events in your model (1 mark):
   - describe how each event changes the state of the system.
   - describe the new events that may be created as a result of this event.
5. Draw a flow chart illustrating your simulation structure. (3 marks)

At the end of Week 4, you will be required to submit a PDF document showing your schematic, flow chart and responses to these tasks.

Hint: You can create flow charts with the free diagram drawing tool Inkscape, but there are many other tools available for drawing connected series of boxes. Find a

tool you like. Also, your boxes and links don't have to look exactly the same as mine, but they have to be: (i) clear and readable, and (ii) consistent.

Appendix 2

Part 2

In this part, you will start to program the model.

Reminder
The system to be modelled is described in Part 1.

Tasks for Part 2

1. Refine your schematic of the system (see Part 1).
2. Draw a state-diagram of the system. (see Part 1) (1 mark)
3. Write code to implement a discrete-event simulation of the system.
    - The code will follow the style of the code presented in this module, used to implement the car park model. Use that code as a starting point, but make sure to customise it to this problem or you will get zero marks. (20 marks)
    - The process you need to follow is outlined below under the heading 'Specification'. (20 marks)

By the end of this part, you should have a working simulation that can output results.

In Part 3 you will test it and use it to create some data with a simulation harness.

At the end of Week 4, you will be required to submit a PDF document showing your schematic and state diagram, along with the flow chart from Part 1 and responses to these tasks. You will also be required to submit two *.jl* files with your code for implementing the discrete-event simulation from Task 3 above.

Specification
This week you will start coding your simulation. I will provide a good deal of the structure for this code because:

- I want to help. There is a lot written below, but by following it carefully you will get a big start towards developing your simulation.
- I want to make sure that everyone has a common starting point.
- I want to make it easier to review and assess your progress by ensuring that everyone adopts the same basic structure for their implementation.

The last point is important. Part of your mark may be based on automated testing of your code. Hence you **must** set up your code in the manner given. Otherwise, you may lose marks for reasons that could be fixed with little effort.

Here are the required specification details:

1. Your code must be included in a two stand-alone *.jl* files. These should be named:
    - *factory_simulation.jl*
    - *factory_simulation_run.jl*

    The first file should contain all of your **data structures** and **functions**. The second file should initialise all variables to values you choose and run the main event loop.

    Note that the second file is a temporary file. The second is for your own benefit, for the moment to test your code. It will be replaced in Part 3 of this assessment with a simulation harness.

    (1 mark)

2. You should use the standard packages that you have been using in this course. These include:
    - *DataStructures*
    - *Distributions*
    - *StableRNGs*

    You may wish to use a small set of additional packages such as *Dates* or *Printf*. Do not use any packages other than these or those that have been discussed in the course.

    (1 mark)

3. Your code must specify three data structures:

```
abstract type Event end
mutable struct Entity ...
mutable struct State ...
```

    Each will contain fields as required. The state structure should contain any queues or lists required, for instance, the event list.

- The `Event` type is an abstract type, which will have more sub-types for each event in your simulation, for instance: `Arrival` and `Breakdown` events.
- The `Entity` event should contain fields to record important event times in the lifetime of the entity.
- The `State` event will, as in the car park simulations, contain a time, event list (queue) and queues for all resources in the system. It will contain other details as needed, such as the state of repair of the machine and the number of events so far.

You should create convenience constructor functions for each of these types, that allow you to create an object when not all of its fields are known. For instance, a function `State()` that returns an initial system `State` variable with any required queues or lists created, but empty, and the clock time set to `0.0`.

(4 marks)

4. Note that when the machine breaks down, the time of completion of the current lawnmower will be extended. That is, you need to change the time, i.e. the priority, of the corresponding departure event. You can modify the priority of an object in a priority queue in Julia as follows:


```
using DataStructures
pq = PriorityQueue()
pq["a"] = 10; pq["b"] = 5;
pq
```

```
PriorityQueue{Any, Any, Base.Order.ForwardOrdering} with 2 entries:
  "b" => 5
  "a" => 10
```

```
pq["a"] = 0 # change the priority of "a"
pq
```

```
PriorityQueue{Any, Any, Base.Order.ForwardOrdering} with 2
entries:
  "a" => 0
  "b" => 5
```

Note that the order of the two items in the queue has swapped. However, be careful if you also store the time of an event in the `Event` structure because that would not be updated in the code above.

(1 mark)

5. Your *factory_simulation.jl* code must have a set of `update!` functions with signatures:

   ```
   function update!( S::State, R::RandomNGs, E::SomeEvent )
   ```
   Each update function should process one of your event types so you will need one function per event type.

   Each update function should modify the state `S` appropriately, including

   - adding any new events created from this one
   - moving any entities from queues to servers and so on.

   These functions must not have side effects. That is, they should not write out any information to files, or interact with global variables. However, your functions may throw an error if the input is invalid.

   These functions need not return anything, but sometimes it is useful for them to return a customer/order entity to the main loop in order to write out information about that.

   (1 mark)

6. Your code should have a data structure for passing parameters:


```
struct Parameters
   seed::Int
   mean_interarrival::Float64
   mean_construction_time::Float64
   mean_interbreakdown_time::Float64
   mean_repair_time::Float64
end
```

   (1 mark)

7. Your *factory_simulation.jl* code should have an `initialise` function that takes as input the parameters of the system and returns an initial system state and creates the random number generators you are going to use. In order to make your code easy to extend or modify, you will encapsulate these random number generators into structures that are easy to pass around.

   - Your code will use four random number generators. Store these in another data structure:

     ```
     struct RandomNGs
        rng::StableRNGs.LehmerRNG
        interarrival_time::Function
        construction_time::Function
        interbreakdown_time::Function
     ```

```
    repair_time::Function
end
```

- Your `initialise` function should create a set of random number generators to populate the above structure. Create an initialisation constructor function for *RandomNGs* using code similar to that below (the parameters used here to create these should come from variable `P::Parameters`).

```
rng = StableRNG(P.seed)
interarrival_time() = rand(rng,
Exponential(P.mean_interarrival))
construction_time() = P.mean_construction_time
interbreakdown_time() = rand(rng,
   Exponential(P.mean_interbreakdown_time))
repair_time() = rand(rng, Exponential(P.mean_repair_time))
```

  Note that, although construction times are deterministic, you can use the same functional form to be consistent.

- The initialisation function should also create a new system state and inject an initial arrival at time 0.0 and initial breakdown at time 150.0 minutes. The function should return the system state and the random number structure.

  Therefore, your initialisation function should look like:


```
function initialise( P::Parameters )
    R = RandomNGs( P ) # create the RNGs
    system = State()   # create the initial state structure

    # add an arrival at time 0.0
    t0 = 0.0
    system.n_events += 1 # your system state should keep track of
# events
    enqueue!( system.event_queue, Arrival(0,t0), t0)

    # add a breakdown at time 150.0
    t1 = 150.0
    system.n_events += 1
    enqueue!( system.event_queue, Breakdown(system.n_events, t1
), t1 )

    return (system, R)
end
```

This code presumes you have created appropriate constructors for the random number generators and system state as described above.

(2 marks)

8. Your code should include a `run!(state::State, R::RandomNGs, T::Float64, fid_state::IO, fid_entities::IO)` function. The inputs are:
    - `state`: a structure containing the system state;
    - `R`: a structure of type *RandomNGs* that contains your random variable generators;
    - `T`: a floating-point number stating how long to run the simulation
    - `fid_state`: a file-ID (an `IO` variable) for the file to which to output your event-based output; and
    - `fid_entities`: a file-ID (an `IO` variable) for the file to which to output your entity-based output.

    The function should run the main simulation loop for time `T`. It should remove an event from the event list, call the appropriate function(s) to update the state and write any required output. This should be the function that writes any output when the code is performing correctly, but you may create some utility functions for writing data that are called by `run!` to make the `run!` function more readable.

    The `run!` function should return the system state when the simulation finishes.

    (1 mark)

9. Your code must output two CSV files. The files should both commence with metadata (you can include this using comments preceded with a `#`).
    - The first file should contain a time-ordered list of all events that are processed in the simulation. This should be written from the point of view **before** the event, for instance, you should report the system that an arriving customer sees immediately prior to their arrival. The CSV file should have columns titled:

      ```
      time,event_id,event_type,length_event_list,length_queue,in_service,machine_status
      ```

    - The second file should contain a list of all entities that have completed service. The CSV file should have columns titled:

      ```
      id,arrival_time,start_service_time,completion_time,interrupted
      ```

    You will need to write and construct these CSV files line by line, but you can do this using either the *CSV* or the *Printf* package or using raw `print` statements.

    (2 marks)

10. Your code must be written with good style. See Julia's style guidelines for information, but in particular:
    - avoid global variables wherever possible (1 mark)
    - choose good variable names (1 mark)
    - use comments efficiently and effectively (2 marks)
    - use white space well (1 mark)
    - avoid very inefficient approaches (1 mark).

You should use your *factory_simulation_run.jl* to run this code and construct some tests. More detail of running the code and testing it will follow in Part 3.

Appendix 3

Part 3

In this part, you will test your implementation and make sure it can output results.

Reminder
The system to be modelled is described in Part 1 and builds on the code written in Part 2.

At the end this week, you will be required to submit:

- a PDF document showing your schematic, state diagram, flow chart and responses to tasks in Part 1 and Part 2
- two *.jl* files with your code for implementing the discrete-event simulation
- a pair of data files (an entities and a state file) that you have produced from your simulation with seed=1 and where the simulation was stopped at time T=1000.0. Your tutor will check that these files came from your simulation.

When marking your assessment, your tutor will check the output of your code by running the code that you submit. This is the code you created in Part 2 and which you will now refine by completing the tasks outlined here.

Tasks for Part 3
Although you might not need to modify the code in *factory_simulation.jl*, you may need to modify it in response to bugs found in testing. The main tasks will be to verify that your code works correctly (as described in Module 3: Verification), and to construct a small simulation harness in which to run a set of comparison simulations (as described in Module 4: Tools to Automate Simulation). Following these steps will help ensure you get the maximum number of marks for the code you have written.

1. Test and verify your code.

   When I run my version of this code with seed=1 for T=1000.0 minutes, I get the following sort of output:

   - The start of the state file showing the event-based output is as follows:

```
# file created by code in factory_simulation.jl
# file created on 2021-07-28 at 14:54:36
# parameter seed = 1
# parameter mean_interarrival = 60.0
# parameter mean_construction_time = 25.0
# parameter mean_interbreakdown_time = 2880.0
# parameter mean_repair_time = 180.0
```

```
time,event_id,event_type,length_event_list,length_queue,in_servi
ce,machine_status
 0.000,     0,  Arrival,   1,   0,   0,   0
25.000,     4,Departure,   2,   0,   1,   0
40.499,     3,  Arrival,   1,   0,   0,   0
51.694,     6,  Arrival,   2,   0,   1,   0
65.499,     7,Departure,   2,   1,   1,   0
90.499,    10,Departure,   2,   0,   1,   0
117.235,     8,  Arrival,   1,   0,   0,   0
142.235,    13,Departure,   2,   0,   1,   0
150.000,     2,Breakdown,   1,   0,   0,   0
182.987,    12,  Arrival,   1,   0,   0,   1
```

- The start of the entity file showing a list of entities with information output on departure is as follows:

```
# file created by code in factory_simulation.jl
# file created on 2021-07-28 at 14:54:36
# parameter seed = 1
# parameter mean_interarrival = 60.0
# parameter mean_construction_time = 25.0
# parameter mean_interbreakdown_time = 2880.0
# parameter mean_repair_time = 180.0
id,arrival_time,start_service_time,completion_time,interrupted
1,0.0,0.0,25.0,0
2,40.49940643026974,40.49940643026974,65.49940643026974,0
3,51.69359364153775,65.49940643026974,90.49940643026974,0
4,117.23501827176 08,117.23501827176 08,142.23501827176 08,0
```


2. Create a test harness that will:
   - run your code for 100 different seed values ranging from 1-100
   - run your code for a set of parameters specified in the following CSV file:

```
# parameters
mean_interarrival,mean_construction_time,mean_interbreakdown_tim
e,mean_repair_time
60.0,25.0,2880.0,180.0
```

From these outputs, you would be able to inform the factory owner about their questions. Although you won't need to answer these questions specifically, this next stage that you would take in the process will be addressed in Assessment 3.

Test Data

In order to help you test your program, I have provided some example output files.

- *state.csv*
- *entities.csv*

1. If you have implemented your code exactly as I have specified, and used the same seed and random number generation, your output should look very, very similar (the only differences should be in details such as numbers of decimal points, white-spacing, or units such as hours or minutes).
2. However, your code may result in some differences. Some are important and others less so. You need to be quite analytical to understand which. That is, what differences occur because of a minor change in the order of actions, and what differences are caused by bugs. If there are differences, you should create some of your own tests to understand what is different from my code.

Note that you can see the metadata in the files, and from that determine any parameters used.