A Simple Simulation of the Magic Kingdom

Jameson D. Morgan
Wright State University
College of Engineering and Computer Science
Dayton, Ohio 45435
morgan.219@wright.edu

***Abstract*—We present a simulation capable of modeling the basic features of the Magic Kingdom theme park in Orlando, Florida. We build the simulation in a modular form and integrate the system to create a simulation capable of modeling the movement of guests at a theme park. Our system models guest arrivals, experience selection and departures. We use such a simulation to answer questions which would be difficult to answer without the use of simulation. The simulation is written in R, a language and environement for statistical computing, and documented using R Markdown.**

Introduction

The goal of this simulation is to build a sufficiently detailed simulation of the Magic Kingdom at Walt Disney World theme park in Orlando, Florida in order to answer a minimum of two *what-if* questions. We use this simulation to answer the following three (3) *what if* questions:

1. Examine how the number of open turnstiles affects the wait time required to enter the park and in so doing, find the optimum number of turnstiles to ensure that no agent waits for more than 10 minutes.
2. Every guest that enters the park must choose a number of rides. In our simulation, we generate this number from a distribution. Compare the generated values to the theoretic value and confirm that our simulation is producing results in agreement with the theory.
3. Based upon the number of turnstiles required to assure that no agent waits for more than 10 minutes, show the distribution of time spent in the park.

Approach

We begin by building the simulation and then proceed to answer the questions detailed above. We build the model in a modularized fashion, documenting and explaining the code as we proceed. The final product produced is a simulation which can be ran to produce output that can be analyzed to answer the three (3) *what if* questions described above. This package may be installed and accessed from https://github.com/morgan219/disney.world.sim. The produced output can be retrieved at https://data.world/morgan219/disneyworldsim.
An accompanying website is available at http://projects.jamesonmorgan.com/disneyworldsim which offers all the code used within this project as well as the unit testing performed.

Modeling: Theme Park

We define a *theme park* to be geographic region which consists of:

i. A Single Entrance and Exit,
ii. Attractions (Rides and/or Shows), and
iii. Dining Experiences

The individuals arriving at a theme park are called *agents*. Agents attend theme parks in order to enjoy the attractions and dining experiences (activities). Because many agents attend the same park and want to enjoy the same attractions and dining experiences, agents must typically wait in *lines* in order to enjoy these experiences. What experiences are chosen are typically a product of multiple factors, including time of day, desire, and current wait times. When a user leaves a theme park is dependent upon a number of factors, such as additional plans, tiredness of the traveling party, sickness, or injuries, and therefore is unique to each agent. An agent is defined as leaving a theme park when they are no longer within the confines of the geographic region defining the theme park.

Modeling: Agent Behavior

Based upon our definition of a theme park, we can model agent behavior as the following **stochastic mechanistic** model:

i. Arrive at Park
ii. Select Activity
iii. Navigate to Activity
iv. Wait to Perform Activity
v. Perform Activity
vi. Return to Step (ii) or Leave Park

The process is **mechanistic** as each agent follows the same process. The process is also **stochastic** as each agent performs these tasks in slightly different ways (as we will see below).

Note, that as mentioned above, because multiple agents want to enjoy the same experiences, agents typically wait in lines prior to enjoying the experience. Therefore, steps (ii), (iii) and (iv) may be a function of the current number of agents within the theme park desiring to have the same experience.

Modeling: Arrivals

As outlined in our theme park definition and agent behavior sections, modeling the arrival of agents to a theme park is the very first step in the simulation. Obviously, due to the proprietary nature of attendance data, theme parks do not

release their attendance data in any easy to use, granular form. Thus, for the purposes of this simulation, we do not have direct access to daily (or monthly) attendance totals as most theme parks publish annual approximations (Rubin 2017). In addition to yearly theme park totals, we do have access to state tourism information ("Visit Florida" 2017). Using both of these data sources, we can obtain the approximate number of agents arriving at a theme park.

In our case, the Magic Kingdom theme park at the Walt Disney World Resort in Orlando, Florida had a yearly attendance of approximately 20,450,000 agents in the fiscal year 2017 (Rubin 2017). Dividing this value by 365 days for year 2017, we get an average daily attendance of approximately 56,000 individuals. However, using tourism data from the state of Florida, USA ("Visit Florida" 2017), we can determine the number of tourists by quarter. Utilizing both of these data sources, we can determine the approximate number of agents per day. We do so using the following algorithm:

i. Determine the percentage of Florida tourists that visit the Magic Kingdom; $mk \approx \frac{total\_disney}{total\_florida}$
ii. Calculate quarterly totals for Magic Kingdom; $q_i \approx q_{i,tourism} \cdot mk$ where $q_{tourism}$ is the quarterly tourism data and i is the quarter
iii. Approximate the daily average for the Magic Kingdom; $\frac{q_i}{days(q_i)}$ where i is the quarter and $days(\cdot)$ is the number of days in $\cdot$.

In Code

- Source: R/modeling_arrivals.R
- Online: "Initial Simulation Attempt"

Continuing to Model Arrivals

Because each day is not exactly the same, particularly during off-season, we set the park attendance swing to be 10000 agents. In the end, we present a **stochastic emperical** model governing the number of arrivals during any particular day as $E[X|q_i] = \mathcal{N}(E[q_i], 10000)$ where X is a day's attendance and q_i is the quarter that X falls in. The process is **stochastic** because the arrivals are generated as a random deviate from a Gaussian distribution. The process is also **emperical** as we are basing this process on observations rather than on explanations which would explain why we are seeing such results. We believe that such a model is reasonable as the Walt Disney World Resort is a major tourist draw so it is not unreasonable to expect that a certain number of individuals are traveling to Florida because of the Magic Kingdom. Our model also spreads attendance across the year, something Disney has tried to do at Disney World ("When Are the Best (and Worst) Times of Year to Go to Walt Disney World?" n.d.).

We create a function called `get_attendance()` which performs this computation.

In Code

- Source: R/modeling_arrivals.R
- Online: "Initial Simulation Attempt"

Testing

No formal testing required as function simply performs division. Ad-hoc testing performed to ensure returned values were correct.

Modeling: Arrival Distribution

Now, we have to model the arrivals of guests throughout the day. Obviously, not all of the agents arrive at the same exact time, so a model must be developed which dictates when the agents arrive throughout the day. Again, theme parks do not publish the distribution of arrivals, but observation and experience provides powerful insights into a reasonable attendance model.

While agents may arrive any time during operating hours, certain hours tend to have more arrivals than others. For example, intuitively (and assuming no special event is occurring), we know that more agents will arrive earlier rather than an hour before closing. What is not quite as intuitive is that at the Magic Kingdom, the crowds tend to be the lowest during the morning hours ("Dealing with Magic Kingdom Crowds," n.d.). This makes sense logically since the agents are on vacation and would opt to arrive in a leisurely fashion rather than wake up early (something which is typically not associated with vacations). From field observations, such a period of low crowds typically exists for the first 3 hours after opening ("Dealing with Magic Kingdom Crowds," n.d.). After this, field observations indicate that crowds tend to build towards mid-afternoon ("Dealing with Magic Kingdom Crowds," n.d.). Rising crowd levels are an indicator that more agents are starting to arrive. For the purposes of simulation, we quantify mid-afternoon as 3:00pm. After this, arrival rates tend to be more flat until an evening parade or show, at which point, arrival rates decrease (i.e. the departure rate increases). For the purposes of simulation, we quantify the evening parade or show as occurring at 9:00pm. Based upon operator published hours of operation, we assume an opening of 9:00am. and a closing of 10:00pm (please note that opening and closing times very and these are just valid times that exist within the range of possible operating times) ("Month Calendar | Walt Disney World Resort," n.d.). Thus, visually speaking, the arrival rate grows until a peak in the mid-afternoon after which the arrival rate slows down to an almost steady-state up until about an hour before closing where the arrival rate become negative.

Setting 9am (park opening) to 0 and 10pm (park closing) to 13 hours (3 hours from 9am-12pm and 10 hours from 12pm-10pm), we define the percentage of arrivals parameter $\lambda(t)$ as follows:

$$\lambda(t) = \begin{cases} 0.15, & 0 \leq t < 3 \\ 0.70, & 3 \leq t < 6 \\ 0.14, & 6 \leq t < 12 \\ 0.01, & 12 \leq t < 13 \end{cases}$$

where t is the current time. Thus, $\lambda(t)$ defines a function that provides the percentage of arrivals expected across the simulation. The exact percentages are obviously unknown and

therefore $\lambda(t)$ is based upon experience and field observations. This is a **deterministic emperical** model. The model is **deterministic** as the output of the model is not subject to any randomness. The model is also **emperical** because the output is purely based upon observation. (Please note that we are not arguing that the arrival times generated are **deterministic**, rather, it is the model dictating the percentage of arrivals which is **deterministic**.)

Of course, simply knowing the percentage of agents which arrive between a given time window is not enough as this does not actually provide us with arrival times. Therefore, we define a **stochastic emperical** model that uses a *hybrid inverse transform technique* to generate the arrival times in accordance with the arrival distribution outlined above. Once again, this model is **emperical** as it is based upon observation and **stochastic** because the technique uses randomly generated numbers to produce the desired arrival times.

In Code

- Source: R/modeling_arrival_dist.R
- Online: "Initial Simulation Attempt"

Testing

No formal testing was done as functions are trivial. Eye-test used to check that arrival times generated matched the proposed model.

Modeling: Agent Entry

Agents may enter the park through any open turnstile. Entry into the park includes picking a turnstile line, waiting to get to the turnstile (i.e. time time required to reach the turnstile) and the service time at the turnstile. We define a **deterministic mechanistic** model of turnstile operation. Each agent takes ten (10) seconds to scan their ticket and proceed through the turnstile. The wait time to reach a turnstile is equal to the service times of all the agents currently waiting to be served $\mu \cdot n$ plus the time remaining until the agent currently being served is finished $t_f - t_c$, where t_f is the finish time of the agent currently being served and t_c is the current time within the simulation. Mathematically, $w(t) = (t_f - t_c) + (\mu \cdot n)$ where n is the number of agents waiting to be served. Agents will select the queue that currently has the lowest number of waiting agents. This model is **deterministic** as it is not subject to any stochastic process and is **mechanistic** as it is based upon how individuals actually enter a park.

In Code

- Source: R/modeling_agent_entry.R
- Online: "Initial Simulation Attempt"

Testing

Since the **Turnstile** class inherits from the **SimpleQueue** class, we perform all testing on the **SimpleQueue** class.

Simulation: SimpleQueue

We define a queue class called **SimpleQueue** which handles the queuing process. The queue allows for specifying an id and a service time. After instantiation, the queue allows adding agents, removing agents, and calculating the wait time if an agent were to enter the queue at the provided time.

In Code

- Source: r/simplequeue.R
- Online: "Initial Simulation Attempt"

Testing

Testing for the **SimpleQueue** class was performed via unit testing the results of which are included as part of the accompanying website.

Simulation: Master Event Queue

Before we can continue to build our model of a theme park, we must implement some mechanics to assist in the running of the simulation. We proceed to define an event record below.

The Event Record

An event record consists of the information necessary to process any of the events defined by the model. We define a class to handle the event record.

An event type is an integer specifying the kind of event that will be occurring at the trigger time. Going back to the model of a theme park, we define the following event types:

- event_arrive_park
- event_enter_park
- event_leave_park
- event_leave_dining
- event_process_attraction
- event_choose_experience
- event_choose_attraction
- event_choose_dining

Within the event record, the `trigger_time` field indicates when the event denoted by `event_type` is triggered. The `agent_id` field is a unique agent id provided to each agent on arrival and the `arrive_park` field denotes the arrival time of the agent at the park. The `remaining_attractions` and `remaining_dining` fields indicate the number of attractions and dining experiences that an agent still needs to visit, respectively. The `attractions_visited` and `dining_visited` fields provide a storage container for the ids of the attractions and dining experiences visited by the agent, respectively. `retired` indicates whether the event record has been retired (i.e. no longer active). The `turnstile_chosen` field indicates which turnstile an agent utilized to enter the park. The `attraction_index` field is used by attractions creating event records to indicate which attraction they are. The `data` field is currently not used and is provided for future use.

It is important to note that not every event record will use all of the fields. For example, an agent creating an event record will utilize the arrival time field, however, an attraction

scheduling its next departure will not utilize the arrival time field. The event record is simply a storage container for communicating events between time steps with the exact fields used, determined by the type of object creating the event.

In Code

- Source: R/event_record.R
- Online: "Initial Simulation Attempt"

Testing:

No formal testing performed as the **Event** class is simply a storage container with getter/setter methods.

The Master Event Queue

Now that we have defined the event record, we need a data structure that can hold the event records and allow for easy retrieval of the next event based upon the `trigger_time` field. We use a priority queue data structure implemented using a min heap (note that we implement the queue as a Fibonacci heap due to its better performance).

In Code

- Online: "Initial Simulation Attempt"; code chunk `load-agent-arrivals`

Testing

No formal testing performed on priority queue as we utilized the `datastructures` package within R to create the min heap. Ad-hoc testing performed to ensure that the trigger times of the items added to the queue matched the arrival times.

Modeling: Agent Departures

We now derive a model governing agent departures. We build a **stochastic mechanistic** model for modeling agent departures. Based upon data in (Foster 2014), we assume the average number of attractions visited is between 8 - 12 with 1 dining experience (this also assumes downtime for sleeping, eating, shopping and recreation at your local hotel). We define a Gaussian $\mathcal{N}(10, 2)$ to represent the number of attractions visited.

Using this process, when an agent arrives, we generate the number of attractions to be visited by that agent. Once the agent has visited the set number of attractions, the agent leaves the park with a constant time of 10 minutes. In addition to meeting the attractions requirement, the agent must also meet the dining experience requirement. We assume that all agents stop once at a dining experience. Therefore, the requirements for departure are:

i. The required number of attractions have been visited, and
ii. the agent has dined at a dining experience

Once these requirements are met, the agent leaves with a constant time of 10 minutes.

In Code

- Source: R/modeling_departures.R
- Online: "Complete Simulation"

Testing

No formal testing performed as the `generate_route_length()` function is rather trivial.

Modeling: Attractions

Obviously, at this point, without attractions and dining experiences to visit, we cannot correctly simulate departures. Therefore, we now turn our attention to modeling the attractions and dining experiences.

Attractions are visited by agents. Each attraction has a name, a duration (time it takes to experience the attraction), a batch size (number of agents serviced at one time) and a unique id. We utilize the TouringPlans (touringplans.com) website to collect the basic data needed to model the attractions. Due to limitations in obtaining accurate attraction durations and batch sizes, these numbers were invented.

Each attraction also has a line associated with it. We use the **Line** class to model this.

In Code

- Source: R/attraction.R
- Online: "Complete Simulation"

Testing

No formal testing performed on the **Attraction** class as it is simply a container for storing information that is accessible through getter/setter methods. Formal unit testing was performed, as noted below, on the **Line** class which the **Attraction** class makes use of.

Simulation: Line

We define a simulation line to handle the process of managing the flow of agents through an attraction.

In Code

- Source: R/line.R
- Online: "Complete Simulation"

Testing

Unit testing was performed with results available on the accompanying website.

Modeling: Dining Experiences

We now define a dining experience. A dining experience contains the experience's name and a category code defining the type of restaurant (either quick-service or table-service). For quick-service restaurants, $\mu_{qs} \in \{1500, 1800, 2100, 2400, 2700, 3000\}$ where each element of the set is measured in seconds. The mean of the quick-service set is 37.5 minutes (or 2250 seconds). For table-service restaurants, $\mu_{ts} \in \{2700, 3000, 3300, 3600, 3900, 4200, 4500\}$ where each element of the set is measured in seconds. The mean of the table-service set is 60 minutes (or 3600 seconds).

These values where created as **deterministic emperical** model based upon postulated average service times for each class of restaurant. The model is **deterministic** as the output is predetermined. The mode is also **emperical** as we based

it on postulated data. Quick-service restaurants are similar to fast-food and therefore we hypothesize that the average service time will be around 37 minutes. Table-service restaurants are more formal and therefore we believe the average service time for such restaurants will be closer to 60 minutes. Additionally, while quick-service restaurants have no restraint on the number of active users (at least within this simulation) table-service restaurants are limited by capacity. We take this into account by creating a counter that monitors the number of active diners in the restaurant. This counter is a field of the dining experience class defined above.

If an agent has remaining dining experiences and remaining attractions that must be visited, we use a binomial distribution (probability = 0.5) to determine whether an agent should choose a dining experience or an attraction next.

In Code

- Source: R/dining.R
- Online: "Complete Simulation"

Testing

Unit testing performed with the results available on the accompanying website.

Complete Simulation

We now define the complete simulation of our theme park.

Extracting the Required Data

We have compiled datasets that define a subset of the attractions and the dining experiences at the Magic Kingdom theme park. The datasets are stored at data.world.

Complete Simulation

The complete simulation code used for the simulation is available online and within the code project distributed.

In Code

- Source: R/theme_park_sim_final.R and inst/notebooks/simulation_trials.Rmd
- Online: "Complete Simulation" and "Simulation Trials"

Testing

Testing was performed with results available on the accompanying website under the "Testing" menu option.

Questions

As proposed originally, we would like to understand the following questions:

1. Examine how the number of open turnstiles affects the wait time required to enter the park and in so doing, find the optimum number of turnstiles to ensure that no agent waits for more than 10 minutes.
2. Every guest that enters the park must choose a number of rides. In our simulation, we generate this number from a distribution. Compare the generated values to the theoretic value and confirm that our simulation is producing results in agreement with the theory.
3. Based upon the number of turnstiles required to assure that no agent waits for more than 10 minutes, show the distribution of time spent in the park.

Question 1

We begin by examining the first question.

Experiment Setup

For this experiment, we are going vary the number of turnstiles from 1 to 50 and determine the first turnstile count where the maximum wait time is below the 10 minute threshold. Because of stochastic variation, we perform a Monte Carlo simulation with two (2) trials per turnstile count. (Of course, this is not nearly enough simulation trials, but due to time constraints it is the best that can be achieved.) The end result is a collection of 100 simulation files with each turnstile count (1 - 50) being simulated twice.

Once the simulation files have been generated, for each simulation file we extract the rows where an agent entered the park (event code: event_enter_park). Once these rows are extracted, we subtract the arrival times (a field in the row) from the time the agent entered the park (the current time parameter). We log whether the maximum wait times are less than the threshold value and proceed to the next simulation file.

Testing the Simulation Output

As we built the simulation, we tested the individual components of the simulation. The idea behind doing this is that if the individual components are working properly, the whole system, if properly configured, should also work properly. During this integration we noticed the following bug:

- Table-service dining experiences have a capacity limit in our simulation. Initially, as configured, the selected dining location was added to the `dining_visited` field prior to actually confirming if availability was present. Issue was corrected and simulations re-ran.

Results

The simulation was ran in parallel on an Intel i5 3470 (3.6GHz base) with 12GB of RAM. Running the simulation with the hardware specified took approximately 30 hours to complete.

With the simulation logs generated, we will now proceed to identify the first turnstile count where the wait time is less than 10 minutes. To do this we load each simulation log, extract the rows where agents entered the park (event code: event_enter_park) and subtract the associated arrival of each agent from the time they entered the park. We do this for all 100 simulations and analyze the output to determine which number of turnstiles produces wait times below 10 minutes. Please note that because we performed a Monte Carlo simulation, both simulation trials must have wait times below the threshold to be considered a success.

From Figure 1, we can see that having 28 turnstiles open assures that, for the agents generated, no agent waits more than 10 minutes to enter the park.

	Trial 1	Trial 2
Turnstile Count @ 21	false	false
Turnstile Count @ 22	false	false
Turnstile Count @ 23	false	false
Turnstile Count @ 24	false	false
Turnstile Count @ 25	false	false
Turnstile Count @ 26	false	false
Turnstile Count @ 27	false	false
Turnstile Count @ 28	true	true
Turnstile Count @ 29	true	true
Turnstile Count @ 30	true	true

Showing 21 to 30 of 50 entries Previous 1 2 3 4 5 Next

Fig. 1. Minimum Number of Turnstiles Required

```
## [1] 0.13
```

Fig. 2. Fail to Reject the Null Hypothesis

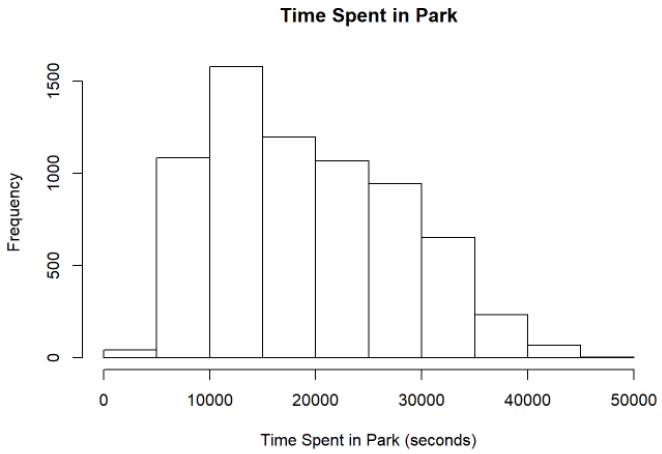

Fig. 3. Distribution of Durations

In Code

- Source: analysis_questions.Rmd
- Online: "Analysis Questions"

QUESTION 2

We now examine the second question.

Experiment Setup

We begin by computing a theoretic value for the number of rides to visit. The number of rides to visit is given by the following line of code `floor(rnorm(n, route_length_mean, route_length_std))` in the `R/modeling_departures.R` file. In the simulation, `route_length_mean = 10` and `route_length_std = 2`. Mathematically, we say

$$x = \lfloor \mathcal{N}(n, 10, 2) \rfloor$$

where x is the number of rides to visit and n is the number of random deviates to generate. It should be noted that this value is stochastic, however, because of the properties of the $\mathcal{N}$, a central limit should exist. For each simulation output file produced (same files as produced for Question 1) we will identify all the arrival rows (event code: event_arrive_park). Once the arrival rows have been identified, we will extract the number of rides for each agent. We will then perform a t-test on the number of rides where $H_0 = \mu_i = \mu_j$, $H_A = \mu_i \neq \mu_j$ and $\alpha = 0.05$. We process each simulation file independently. The results are detailed below.

Results

We process all of the simulation files and compute a t-test over the number of rides to visit and the theoretic mean. (Note, that because the `theo_mean` calculation is stochastic in nature, the exact value returned will vary between trials.)

From Figure 2, we can see that 13% of the simulation trials (13 trials) had a mean that were significantly different from the desired mean. We consider this percentage a success, considering the issues discussed regarding the stochastic nature of the theoretical mean computation, and therefore consider the simulation trials a success.

In Code

- Source: analysis_questions.Rmd
- Online: "Analysis Questions"

QUESTION 3

We now consider the last question.

Experiment Setup

Using the same simulation results produced for Question 1, we will identify the simulation results which prevent a wait time from exceeding 10 minutes and minimize the number of open turnstiles (using the answer from Question 1). After doing this, we will process the data by identifying all the rows in which an agent leaves the park (event code: event_leave_park) and subtracting the time of arrival (a field of the returned rows). We will then build a histogram - a non-parametric model - detailing the results. No formal statistical test is necessary as we are simply presenting results.

As can be seen in Question 1, having 28 turnstiles open will assure, given the arrivals simulated, that no agent waits more than 10 minutes.

Results

Figure 3 shows the distribution of agent durations.

In Code

- Source: analysis_questions.Rmd
- Online: "Analysis Questions"

References

"Dealing with Magic Kingdom Crowds." n.d. www.dadsguidetowdw.com; Online. https://doi.org/https://www.dadsguidetowdw.com/magic-kingdom-crowds.html.

Foster, Erin, ed. 2014. "How Much Can You Do During a Disney Vacation?" TouringPlans.com; Online. https://doi.org/https://blog.touringplans.com/2014/02/25/can-day-disney-world/.

"Month Calendar | Walt Disney World Resort." n.d. Walt Disney World Resort; Online. https://doi.org/https://disneyworld.disney.go.com/calendars/month/.

Rubin, Judith, ed. 2017. "Theme Index Museum Index 2017." Themed Entertainment Association (TEA); Online. https://doi.org/http://www.teaconnect.org/images/files/TEA_268_653730_180517.pdf.

"Visit Florida." 2017. Visit Florida; Online. https://doi.org/https://www.visitflorida.org/resources/research/.

"When Are the Best (and Worst) Times of Year to Go to Walt Disney World?" n.d. www.mousesavers.com; Online. https://doi.org/https://www.mousesavers.com/walt-disney-world-vacation-discounts-and-deals/frequently-asked-questions-about-walt-disney-world/#busy.