Monty Hall

Discovering the world of Monty Hall through Monty Carlo Simulation

Brayan Quevedo Ramos

Porterville College

Subject: Introduction to Engineering Methodologies II, 70206

Lab Report

Date: 9, October 2022

Contents

1 Introduction

The Monty Hall problem is an extremely famous radio and television host, most known for being the prominent host for "Let's Make a Deal". This problem is a probability puzzle where there's three doors, one with a large prize, and other ones with goats. The contestant is given the option to pick any of the doors, after which the host (or in this case the program) demonstrates what one of the three doors isn't the right answer. After which, the user is told if they wish to keep their original choice, or switch to the other. After all of this, it's revealed what door had the actual answer.

1.1 Purpose

The purpose of this program is not only to gain a deeper understanding of GUI in any programming language of choice, but also how to run a Monte Carlo simulation. This simulation is a model used to predict the probability of a variety of outcomes when random variables are present. In this instance, the variety of outcomes are 6 – these being choosing one of three doors, and having the option of keeping or switching doors. Below is a mathematical statement on the simulation with x attempts:

$$\lim_{x \to \infty} \frac{\sum wins}{\sum losses} \approx actualresult$$

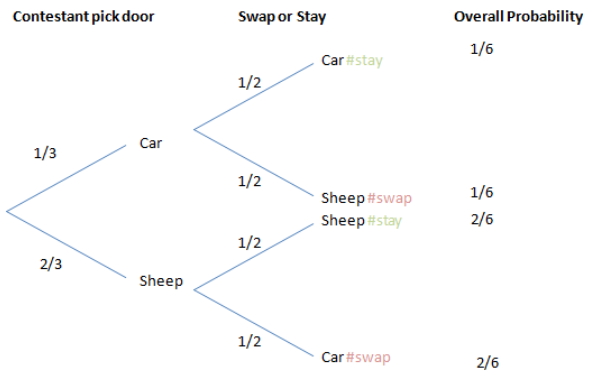
To verify the actual probability outcomes, one would use a probability tree to see and calculate the actual probability outcomes.

1.2 Requirements

1. Must give user choice of doors.
2. Give user choice to switch or stay.
3. Must report probability of winning with and without switching.
4. You must display a graphic of some kind.

1.3 Theory Used

Besides the theorems used above, there wasn't any additional 'theory crafting' needed solve this simulation. Because of this, the actual results are listed below (via a probability tree).



2 Discussion

2.1 Introduction to Web Apps

The way this program was developed was through a myriad of programming and markup languages in the WEB 2.0 and HTML 5 space. More specifically, the program was broken down into components, in which different languages had various responsibilities. The manner which components were broken down into followed the ES6 (ECMAScript 6) protocol. Because of this, a high level overview of program layering went as followed: Chromium for the web engine, JavaScript for the program logic, HTML for web interfacing, and CSS for styling. A more detailed look at the layers uncovers that each stack relied heavily on the other for operation. For example, the Chromium engine relied heavily on ES6 to follow the proper format to display the image – HTML rendering engine relied on Chromium and the CSS style assessment relied on HTML page layout. From CSS styles, the JavaScript framework is loaded, and then executed. For comparison, ES5 didn't load JavaScript, instead relying on Flash to run the logic. While this may seem inconsequential at a high level, many programs (including this one), are fundamentally web apps, which require acknowledgement from the web engine at a specific time, since content is often compiled and rendered during run-time.

2.2 Solution: Overview

The Monte Carlo simulation was run as a background task inside a web app. This web app is a recreation of the popular game-show, "*Let's make a deal*". The contestant (the user) will be the one choosing the door and making the appropriate selection in the hopes of winning the big prize! After each session, data from the game would be gathered, and shown to the user.

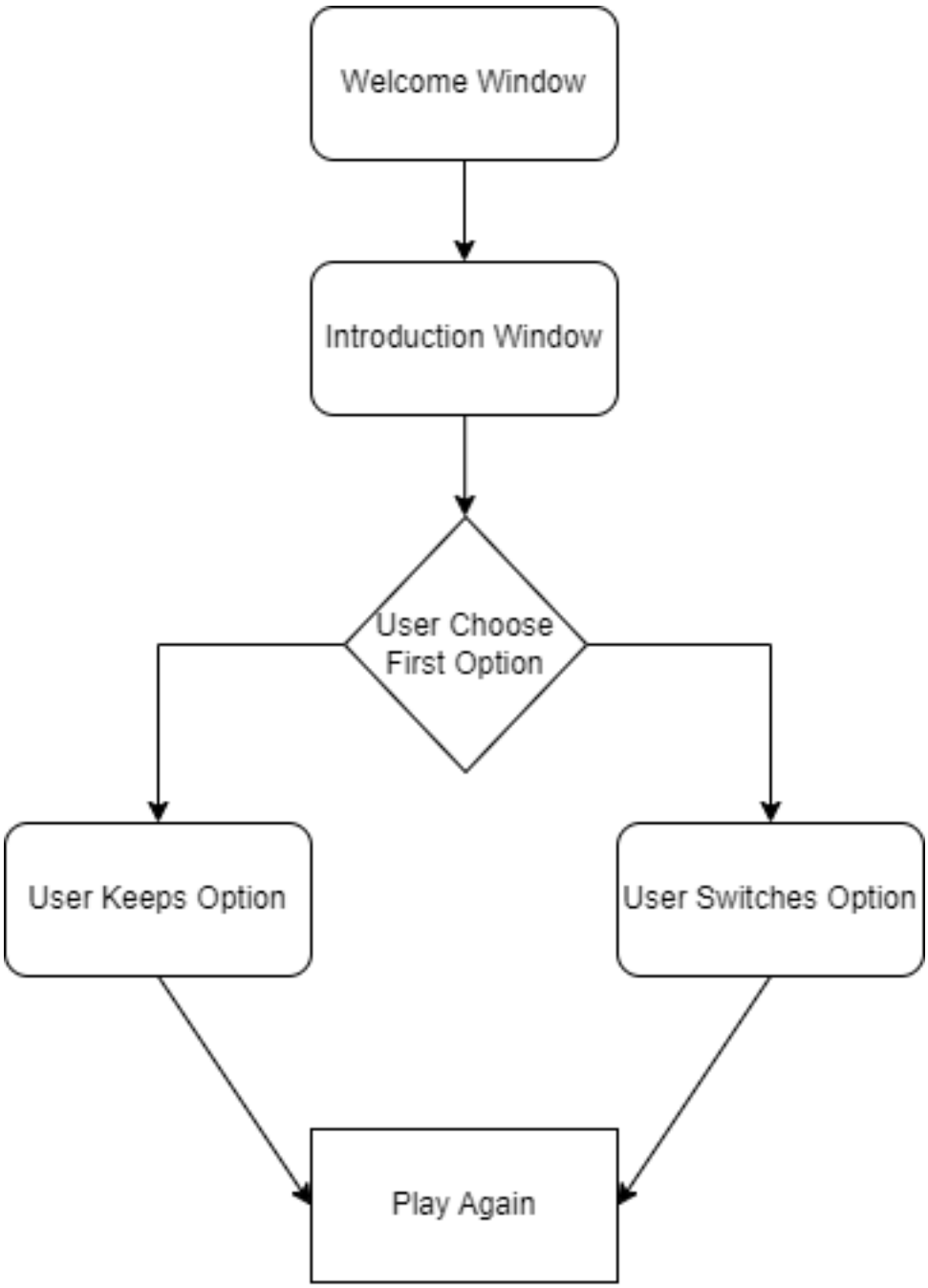
2.3 Solution: Technical

This web app was broken down into two layers to ensure optimal performance and code efficiency: Front-end and Back-end. The front end consisted of HTML and CSS. The frameworks used were Bootstrap 5.2.1, Popper 5.0.2, and Bundle 5.0. Bootstrap 5 was used as the HTML framework of choice due to its simplicity but fully modular control over the CSS Grid (the CSS Grid is essentially the canvas that the user sees but broken down into predefined chunks). Since the display resolution and specification is unknown to the programmer (the program might run on a 1920x1080 , a 720x480, 2160x1440, or some other aspect ratio screen), Bootstrap allows the development of a web app in one aspect ratio, to which the framework will automatically adjust and resize different content elements for all devices. Popper 5.0.2 is the JavaScript Framework that works in conjunction with Bootstrap. It's main focus in this project was for the programmer to be given the needed permissions to allow JavaScript to interact with various web elements. In vanilla JavaScript, JS can only interact with base HTML (such as retrieving or printing data to the user), but with Popper, the developer can now access different modules of the class, changing the properties and modifying the CSS as needed. This was critical in the development of the web app, since every time the user interacted with an object on the screen, I wanted JavaScript not only to get and process the data to make logical conclusions as to what the program should do next, but also what to display and in what order. For example, one of the first windows that the user is presented with is to choose a door. Once the user interacts with whatever door they chose, JavaScript has the ability to read whatever the door option was chosen, calculate which door can be opened to show the "goat", and display that information all in a new HTML window! Lastly, Bundle was hardly used in this

program, but it's used to package this program up and allow any developer to launch a live server to run the code.

The back end of this program was developed using AJAX and DOM. AJAX, or Asynchronous JavaScript and XML is a browser built-in HTTP request that allows JavaScript to load information into a web app without having to reload the entire page. While not needed for this web app since it's small in size, it's a nice to have since the user doesn't have to wait multiple seconds to continue playing the program. DOM stands for Document Object Manual, and allows JavaScript to have more access into the contents of the user's computer. Sound scary? Well, if you've ever clicked the button that states "Accept Cookies", you actually allowed DOM to run as a JavaScript process. At a high level, DOM can create and store little bits of information on a user computer to make certain actions load faster. For example, if a user has to log into a web-page, instead of having them type their information in every session, the website can simply create an authentication key and store it on the users computer. These website cashes that DOM makes can't be accessed by anyone else (user or website), since the registry is locked to a specific IP which only the website has rights to. For this program, there isn't any personal information being leaked to the government agencies, but rather to keep track of wins and losses of the game! At the end of the day, this program is meant to be a simulation, so while the results may not be most accurate since only a certain amount of cycles can happen a minute.

2.4 Flowchart



2.5 Implementation Issues

As I'm fairly familiar with JavaScript, this project didn't have anything difficult to implement. However there were annoyances that did slow this project significantly such as the initial environment setup, and limitations of Popper. Developing for Web is unique since there's typically many hurdles to get a web app done quickly. For example, since JavaScript is a run-time language, the only manner to know if the code works is to open it in a web browser and interact with it. While this seems inconsequential at first, imagine having to play through the entire game just to test one method of logic. Because of this, writing JS in a live server required Bundle, which is only officially supported on Visual Studio 2019/2022. Even if these basic requirements were met, the dependency's needed for the live server to launch were in a large "web development" package that Microsoft requires the developer to download.

The limitations of Popper were also annoying since Popper wasn't specifically designed for web apps (it just so happened it worked conveniently for this project). Instead, it simply gave developers more elevated access for issues such as translating HTML text to PHP or SQL queries to run. Because of this, I had to find different workarounds to display variable text/images depending on what the user chose. As a preface, the practices I'm going to share are awful but worked perfectly for this program scale. As shown in the code below, one of the goals of this assignment was to dynamically change the image of what door was the "goat" with the preconditions that the program knew what answer was the correct one, and what option the user chose.

```
/*CSS*/
#window_screen_2{
visibility: hidden;

/*JS*/
document.getElementById("#").style.visibility = "hidden";
document.getElementById("#").style.display = "none";

document.getElementById("#").innerHTML =
    '<img class="text-center" src="Assets/Images/lets-make-a-deal-door-1.jpg"
    alt="Door Number 1">
        <p class="text-center">Door 1 is fake!</p>
    </img>';
```

This code works extremely well! But it's also really bad to scale, since Chromium has to keep this image in memory, AND load it on the page without the user seeing it. This means that if the user had a really slow internet connection, some of the elements (such as the picture that they can't see yet) may not load in time, thus the program would hang until further notice.

2.6 Known Bugs

There are currently no known bugs to break this program, but two to make the make the experience less enjoyable. Since the user has limited input, and the logic is simple (the button the user clicks dictates what logic shall run), it's difficult to break the program. But just because there's limited input doesn't mean everything

works perfectly. To start, the buttons sometimes do and don't work. What I mean is that functionally, they do work, but the animations freak out on occasion. Either they won't show, or they'll show much after hovering on them. From initial debugging, I've seen that it might be because Chromium might be breaking the animation cycle since there's too much on the screen, or that it's a Chromium bug. I do believe the first issue might be it, since the way I have the program laid out, the HTML is preparing for the next command on hover, so the more information it has to load, the slower it is to animate. The second bug is that in some instances, the probability might be wrong. This isn't nearly as critical since it's literally one console log of code to fix (but finding out which line it is shall be the end of me). The logic works fine, since the console reports those values out (which are the real values), so nothing fundamental breaks!

2.7 Handing Bad Input

It's impossible to break this program since all the frameworks are made for different devices, and the user can only click a few buttons.

3 Lessons Learned

3.1 What Did I Learn?

Even though I'm already fairly comfortable with the web development process, it was still really fun to use what I learn in class and apply it to different programming languages! The last time I did a "proper" web development assignment, all the frameworks I was using were getting to be outdated (Like Bootstrap 3.1) and unsupported, so one of the defining features of this project was to read up on a ton of documentation to refresh my skills on how to develop with the updated versions of these frameworks! I also got a much deeper understanding on the technicalities of JavaScript. For example, I never really expected JavaScript to interact with HTML since my experience came from defining objects on an actual website. This is different than app development since the work I do is essentially a customized version of WordPress, as users can drag and drop media into a website, but built to have extremely specific features that no online host has.

3.2 Doing Things Differently

If I had the opportunity (and time) to redo this assignment, I'd love to redo the project in a different programming language in WEB 3.0. React is a newer JavaScript library that I've been dying to learn about!It's simple component structure would've made development so much easier since instead of creating a window class and manually managing them, since everything is built off different templates. Each component follows a specific format that makes debugging easy, but still gives the programmer the access they desire. Plus, the language incorporates really well into SQL, MongoDB, and even SalesForce! Better yet, they incorporate more modules into the base framework, so components like 'time', PHP, 'random' and more are all out-of-box features. If you wish to learn more about it, please visit *https://reactjs.org/*.

3.3 Revision of Exercise

This exercise was amazing – I had lots of fun doing it! If I had to make this better, I'd love an example of how you'd like each section of the requirements implemented. For example, I had the longest struggle wondering if I wanted the user to see the probabilities of each option, or simply write them in the console (which I ended up doing). Also, if we could have the set requirements from the start that would help immensely!!!!

4 Results