Hogwarts Hobo Project

Christian Wang - 500890567
Derek Lee - 500907276
Elijah Tungul - 500885285
Jason Zhu - 500893695
Miguel Nobre - 500807779

Table of Contents

The project is viewable online at (https://squishy123.github.io/HogwartsHobo/)
The project repo is available online as well(https://github.com/Squishy123/HogwartsHobo)

Product Backlog

Requirement	Story	Engineering Tasks	Initial Priority (1/2/3)	Effort Estimate /Risk Estimate	Sprint Details
Setup project dependencies and scripts	For this project, Node.js will be used, so various dependencies and scripts will be needed.	**Setup NPM with the dependencies:** ● npm ● nodemon ● babel ● jest ● lowdb ● tensorflowjs **Add the following scripts:** ● babel-node building and running ● test runner ● distribution building and export ● formatter using esLINT	3	2 hours ± 1 hour	Setting up the dependencies and scripts was the first story to be completed and allows the game to function in the Javascript (Node.js) language. The development team accomplished this task fairly quickly, and the creation of the hobo class was the next story to be complete.
Create the hobo class	Hobos are the main entities that interact with the game. They will	Create the **constructor method with the instance**	3	3 hours ± 2 hours	The creation of the hobo class was the second user story to be

	dodge trains by jumping from track to track. They must have **HP, position tracking and information input** “from the paper planes”. The hobo’s decisions come from the information of past trains 1 tick earlier.	**variables** of HP, position tracking, and information. Add a **class function to compute given information** and make an educated decision on where the safest track is. Additionally, add a **class function that allows the hobo to jump to any track** it thinks is free at that moment in time.			completed. The development team finished this story in a reasonable amount of time. The creation and implementation of the track class was the next story to complete.
Create the track class	This class generates information for the hobo (generates paper airplanes), as well as the number of trains on the tracks.	Create a separate Javascript file containing the class. Have the constructor method accept the following instance variables: time on the track, and time between each	3	2 hours ± 2 hours	The creation of the track class was the third story to be completed. The development team finished this fairly quickly, and the helper function module story was the next one to be completed.

		train spawn. The track class will also have a history and recent list, as well as a state instance variable. Create a function to generate trains depending on the number of trains needed.			
Create helper function module	This module includes various math functions needed for the game to run, including factorial, random integer generator, Poisson distribution probability mass function, and any other functions that may deem useful.	Create a separate **Javascript file** that includes these helper functions. Implement functions based on the user story given.	3	3 hours ± 2 hours	The creation of the helper function module was the fourth story to be completed. The development team finished this at a less than moderate pace, due to having to do research for Poisson distribution. The obstacle was overcome, and the next story was to implement the test cases for each helper function.

Implement test cases for each helper function	Testing the accuracy of each function will be needed.	Create a table of expected values for the helper functions and have expected pass and fail cases for each test.	2	2 hours ±2 hours	The implementation of test cases for each helper function was the fifth story to be completed. The development team took a fair amount of time researching Poisson distribution, but the obstacle was overcome once again. The next story to be completed is the simulator class.
Implement the simulator class	The simulator class will be used for the main game loop.	Create a separate **Javascript file** including this class with it's variables and functions. The simulator takes in parameters for the number of tracks, the average time on tracks, average time in between trains, and time for each step.	2	2 hours ± 2 hours	The implementation of the simulator class was the sixth story to be completed. The development team completed this in a reasonable amount of time. The next story to be completed is the implementation of an algorithm to maximize the hobo's HP.
Implement an algorithm to maximize the hobo's HP	This can be created as a new class: the smart hobo. The smart	Create a new class with new variables and functions:	2	3 hours ± 1 hour	The implementation of the algorithm was the seventh story

	hobo uses the expected/average time on tracks and expected/average time in between to calculate its chance of safety, and will act accordingly based on its safety, else use basic hobo act functionality	compute the safety of the hobo by using variables such as average time on tracks, average time in between tracks, as well as a new safety variable. The hobo should also have an improved act function using the safety variable.			to be completed. The development team completed this in a fair amount of time. The final story to be completed is the visualization of the game simulation.
Visualization of the game simulation	To simulate the game, visualizations (graphs, tables, etc.) will be used to track the results of the given inputs. The visualization will be given user inputs (i.e. number of tracks, average time for each train, the hobo's HP, and number of rounds).	Using HTML, CSS, and Javascript, create a web page displaying each graphical visualization. Use HTML and Javascript to send user inputs (created using textboxes) to a main Javascript file that generates the game simulation using graphs. If possible, add extra aesthetic flair to the web page to make it look nice.	2	2 hours ± 2 hours	The visualization of the game simulation was the final story to be completed. The extra aesthetic flair was added to make the page look cleaner. The development team completed this in a surprisingly fast amount of time. There were no more stories to be completed.

Implement a genetic algorithm for the hobo class	Using similar techniques to breeding and evolution found in nature, a genetic algorithm can be created to help the hobo evolve.	Create a new hobo class with new variables and functions, as well as implement the functionality of genetics.	1	4 hours ± 2 hours	
Allow user control (mapping) of the hobo	Users will want to play and interact with the full game, so movement and the control of the hobo may be necessary in the full game.	Create a class to detect specific keypresses, and implement the class in the hobo class, to allow for hobo movement.	1	2 hours ± 1 hour	
Create user interface and graphics for the game	In the full game, users will want a fully fleshed out interface for easier interaction between the game and the user,	Implement the user interface and graphics using HTML and CSS elements.	1	3 hours ± 1 hour	
Create visual sprites for the game	A visual representation for each class: a hobo could look old and dirty, the trains can have a locomotive carrying a payload of passengers, etc.	Create sprites using photo editing software of your choice (Photoshop, GIMP, Paint.NET, etc.)	1	2 hours ± 2 hours	

Test Report

Unit tests created for helper functions

Test	Pass?
Calculation of probability mass function over 100 combinations of 10 different lambda and 10 different k within 4 decimal places	Y
Simple factorial calculation	Y
Range of generated numbers in custom random function	Y
Correct creation of distribution	Y
Correct case of generated number outside of distribution	Y
Correct generation of random numbers according to pre made distribution	Y

Unit tests created for game state changes

Test	Pass?
Hobo initialization	Y
Hobo altering position	Y
Tracks initialization	Y
Short term game run without crash	Y
Hobo receiving damage	hobo health unobservable

Manual testing

Test	Process	Pass?
Hobo receiving damage	Initializing a game with one track, placing the hobo in it, setting probability of the train	Y

	arriving to 100%, ran for several cycles.	
Smart Hobo performing better than Basic Hobo	Run identical games with tracks and probabilities 10 times each for both types. Observe that average remaining life points for smart hobo is greater than basic hobo's	Y
Smart hobo correctly approximating average train time	Run a game with smart hobo, observe its approximation of average track time for trains over hundreds of cycles. Observe estimate is within +-1 of actual value.	Y
Correct sequence of hobo acting	Run a game with smart hobo, print all changes of state like health, adjacent track safety rating, movement. Observe logged behavior is as expected with a game with 2 tracks and small act count	Y

Code inspection

Name	Results
Full team inspection and walkthrough of poisson value calculation and usage to create probability distribution for use on train tracks. One person interpreting, original writer there to correct misunderstandings.	Found an issue with generating a number that lies outside the range of precalculated poisson values, came to a consensus to default out of bounds values to 0.

Results Report:

Simulation Constraints and Variables

Firstly, it is necessary to re-discuss the actual breakdown of the game simulation. This is important to explain constraints and variables kept constant throughout the environment.

The simulation is based around a virtual train tunnel, by which trains pass by on n, the number of tracks. A hobo automata stands at the end of the tunnel. This automaton gets the information about oncoming trains 1 game tick behind (since the information travels by paper plane and that takes time). The information basically shows the current track statuses, whether it is occupied(1) or not occupied(0). The automaton then has to react accordingly based on the information it gets. Every time the automaton gets hit, its HP is decremented by 1. The number of game rounds it survives is its final score.

Valid parameters that go into the simulation include the average time a train is on the track, the average time between trains, the number of tracks the tunnel has, the amount of HP the hobo automata starts with and the number of rounds that the simulation runs for. There is an additional parameter for timestep that is used to make generative distributions more accurate.

Track occupancy is generated using the poisson distribution based on the average time between trains and the average time a train is on the track parameters. Using those parameters, we generate 2 seperate poisson distributions for each and use a weighted randomized function to generate a unique train time on track and time between next train for each train passing on each track. This ensures that while the train generation is randomized, the probability for each independent event occurrence is still logical. This ensures that while there is an element of randomness for each individual simulation run, an average over multiple runs would give relatively accurate results/conclusions.

Discussion about the Algorithms

Next, we will discuss the different algorithms and their results.

Basic Hobo: This algorithm was used as a baseline for constant automaton reactions. The basic hobo gets the information about track status (recall this is 1 tick behind) and

jumps to a track that is empty. While this does not guarantee that the track is actually empty in the current tick, it is a reasonable assumption to make in certain situations. Additionally since the basic hobo has no learning ability, the rate at which it gets hit stays relatively constant.

Smart Hobo: This algorithm is fundamentally different from the basic hobo algorithm. The smart hobo automaton relies on a simple learning method in order to decide where to jump at each game tick. Operating on the assumption that train distribution is based on the poisson distribution, the smart hobo attempts to estimate the chance that a train will be on the track for the next tick(keep in mind that the information it gets is always 1 tick behind). This occurs by keeping a record of track occupancy, a running average of the time between trains and the time a train occupies a track. The hobo then calculates a track safeness score based on the current and historic information and jumps to the safest track.

Findings

Our findings are separated by 2 different main cases: number of tracks at 2 and the number of tracks at 5. For each main case there are 2 sub-cases: starting at low HP(10) and starting at high HP(50). The number of simulation rounds were kept constant at 500 for all cases. 5 simulations are played and averaged for each case and parameter.

Starting at Low HP

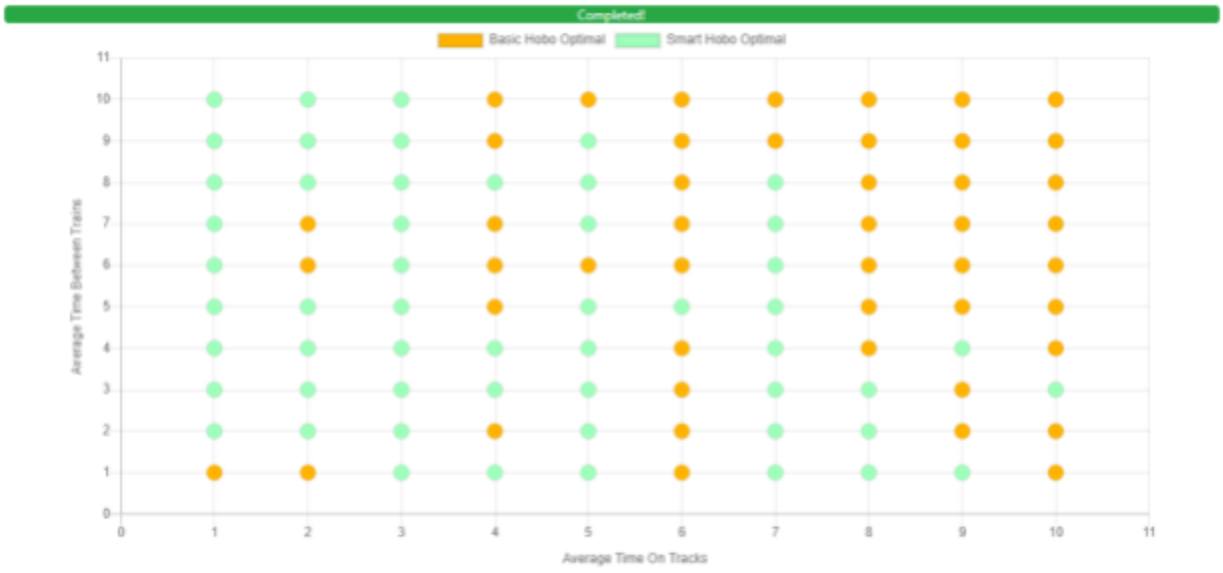

Figure 1: Optimal Matrix for Low Starting Automata HP and Number of Tracks 2

The results visible on the above figure show that on average, for low average time between trains and low average train time on tracks, the smart hobo outperforms the basic hobo. This is to be expected since in those cases, the basic hobo algorithm is expected to perform poorly. Its reactions to past information is likely to be inaccurate with the increase of the length and number of trains. However in general, the basic hobo still outperforms the smart hobo in many cases. We think this is due to the nature of the smart hobo's learning ability. Since it takes time to reach an accurate estimation, it could die before reaching a decent predictive capability.

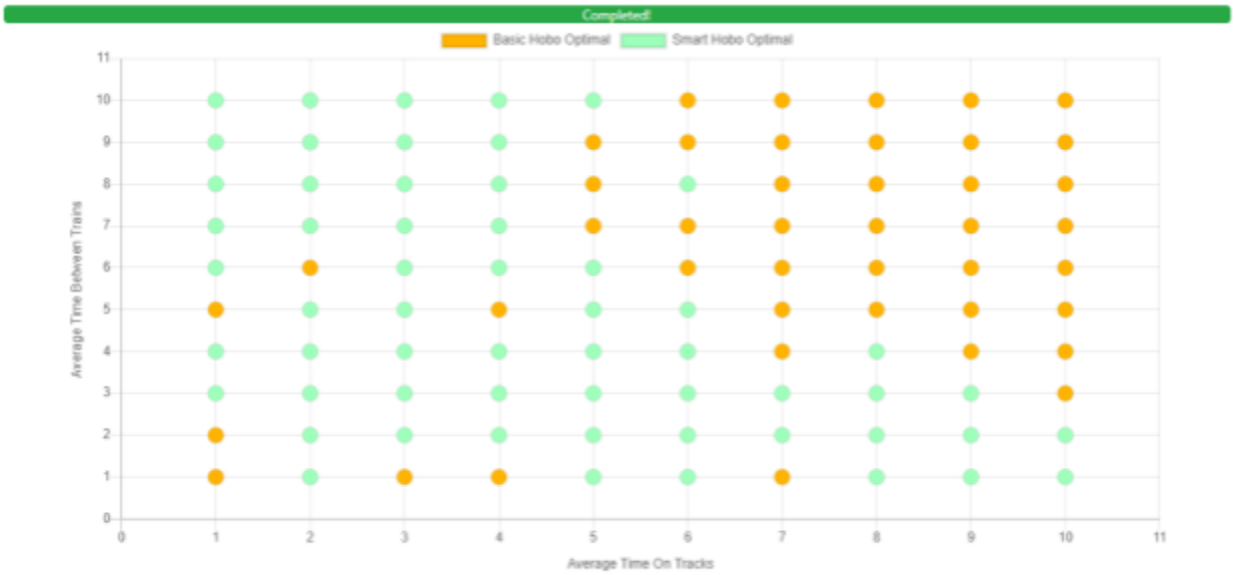

Figure 2: Optimal Matrix for Low Starting Automata HP and Number of Tracks 5

The results are more or less the same for the increased number of tracks.

Starting at High HP

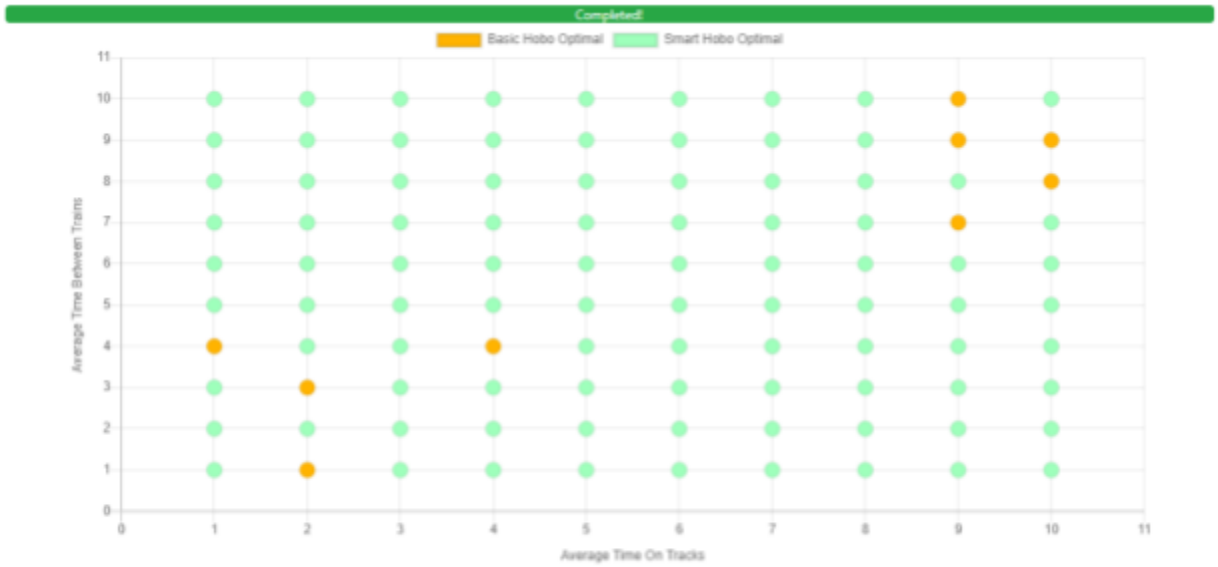

Figure 3: High Starting Automata HP and Number of Tracks 2

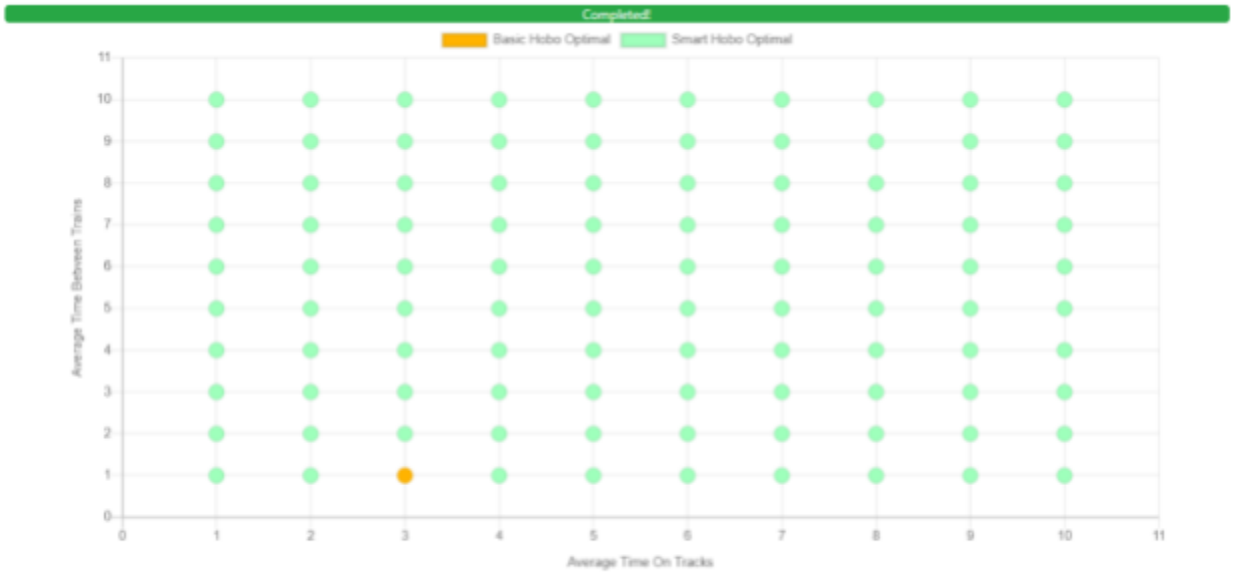

Figure 4: High Starting Automata HP and Number of Tracks 5

With the increase of starting HP, we see the smart hobo outperforming the basic hobo in many more cases (see figures 3 and 4). The increase of starting HP seems to have a huge impact on the learning capability of the smart hobo. While the basic hobo still outperforms the smart hobo in certain cases, in general the smart hobo is better than the basic hobo.

With the increase of the number of tracks (see figure 4), the smart hobo outperforms the basic hobo in almost all the cases. This is most likely due to the nature of the simulation. There are “lose-lose” cases where all the tracks are occupied in the next tick, meaning that both algorithms lose HP. Since this is due to the randomness layer of the simulation, these cases can have a large effect on the automata results.

Learning Capability and Hit Rate

Using the direct simulation tool, we can draw further conclusions about the smart hobo’s learning capability.

Poisson Prediction

Since the smart hobo automaton revolves around attempting to calculate the passed poisson parameters, we can see how close it gets to the actual passed values.

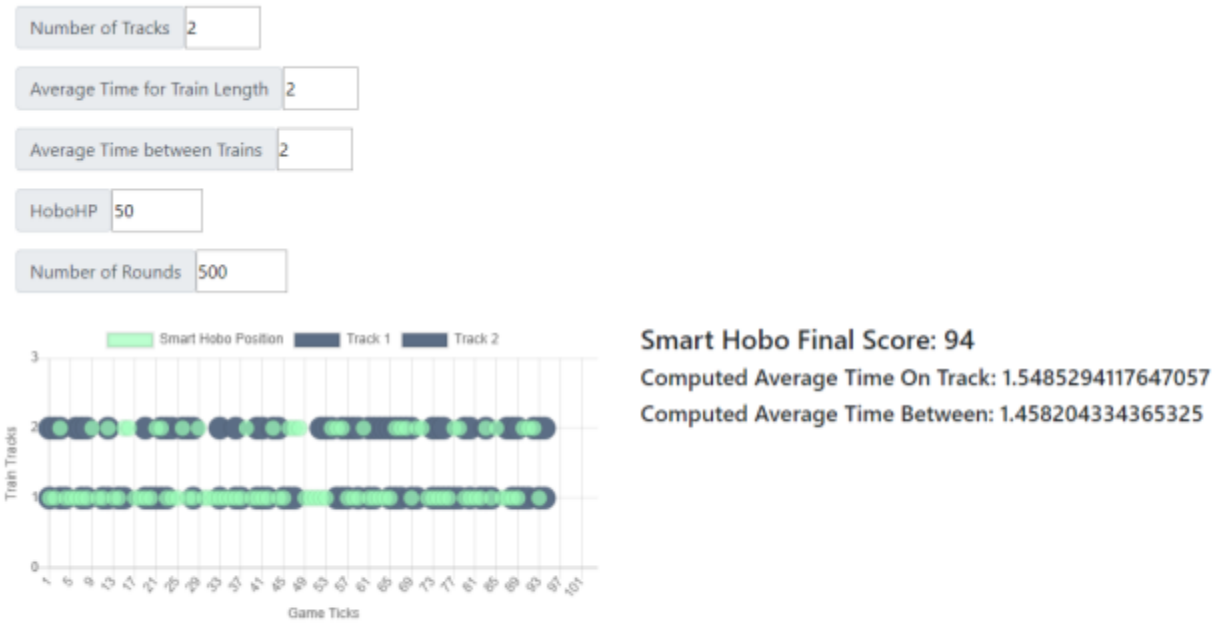

Figure 5: Computed Poisson Parameters

Clearly seen in figure 5, the smart hobo automaton does a pretty decent job at estimating the averages. Additionally, our team found that by increasing the number of rounds and the starting HP, the hobo came closer to the passed averages.

Hit Rate

We kept track of the hit rate of each automaton at every game tick. This gave us a clearer picture on the learning capability of the smart hobo compared to the more constant performance of the basic hobo.

Figure 6: Hit Rate Of Differing Automata

Based on the data found over several test cases, the hit rates for the smart hobo does decrease over time, in most cases plateauing. Commonly, the smart hobo starts off with a higher hit rate than the basic hobo and eventually reaches a point at which it starts to outperform the basic hobo, thus decreasing the overall hit rates. We can clearly see in figure 6 that while the basic hobo's hit rate seems to remain relatively linear, the smart hobo dips and plateaus.

Conclusion

Through the development and testing of the automaton, we found that the smart hobo automaton does outperform the basic hobo automaton in the majority of cases. This is due to the nature of the learning capabilities of the smart hobo, allowing him to predict future events more accurately. Additionally, our team discovered the need to do more research on potential optimizations and more analysis tools. Going forward, there is vast potential for being able to combine the techniques and concepts of each algorithm and improve the learning capability of the hobo automata.