Slim Ghodhbane

Professor Chun Wai Liew

CS 150: Data structures & Algorithms

7 March 2018

# Project #1 – Medieval Park Simulation

## 1-Background Story:

Centuries ago, in the land of the Scorched demons and leafy fairies, there was a big kingdom called “the clover Kingdom”. The people in this land were ruled by a smart yet dictator king. He loved power and the luxury of the throne and would do anything to keep it for him.

In the past few weeks, the king noticed the unrest of his advisers due to the agitation happening in the capital. The dissatisfaction of the citizens is growing at an exponential rate thus the king suspects a coup d’état happening in the next few months. To avoid this imminent danger, he starts thinking about a plan that could maintain the safety of his throne and extends his control even further. After thorough reflection, the king decided to build an amusement park. He thought this park would enable him to assuage his citizens’ anger but also keep them busy from plotting anything threatening to him.

However, it wasn’t a mere amusement park. The king wanted a long term guarantee to his power and control. Therefore, he decided that this park would contain the most dangerous of attraction: battlegrounds, dungeons, arenas, armories, and knife fights. He wanted to exploit the energy and vivacity that the people had to unable them revolt. Since these games were extremely dangerous, they could cause injuries to the people trying them.

Now, the King is concerned by how to increase the efficiency of his park. For that, he asked his master wizard to design a simulation of the behavior of the attraction park which answers the following question:

- How to maximize the injury ratio?
- How to maximize the wait time for every visitor of the park?

# 2-Introduction:

The purpose of this project is to create a simulation of a medieval attraction park. This simulation will exhibit the behavior of groups of people visiting the park with the purpose of analysis and optimization. Thus the project is divided into two main compartments: People and park.

The people visiting the park get assigned a ranking number relative to their social class: peasants (<5), servants (5-12), warriors (13-18), knights (19-22), and royals (22>). They also come in different sets: individuals, peer groups of same rank, or families (2 Royals + servants). Every combination of groups have a certain probability to show up to the park with a set a range size and type.

As for the park, it has a predefined space and is constituted of different rides. Each ride will have specific parameters stored in a configuration file which are its name, the space it takes in the park, its length to complete, its capacity, its queue capacity, and the social group it appeals to. The special thing about this park is that it can cause the people to get injured. Each ride has a specific chance of getting the riders injured.

Every person can try each ride 0 or 1 times. People with higher social ranking can go on rides for lower ranks but not vice versa. However, they cannot go on any rides designed for people that have less social ranking than members of the group. For example: if a family consists of 2 royals and 2 warriors they cannot go on a ride that is designed for servants groups leave the park when either they have exhausted the rides they want, or there is no space for them to join the queue for a ride *("CS150 Project 1 Description").*

The simulation runs for 1000 ticks and in every tick (each step of the simulation):

- A number of groups will come to the park.
- The rides will progress, will load, or will unload passengers.
- Passengers will go from one ride to another.
- Group leave when leaving condition met.

Our aim is to find out what makes the medieval park efficient. The questions we are going to answer with the help of the simulation are:

- Which configuration will generate the max number of injured people?
- Which configuration will generate the longest wait time.

For the first question, I speculate that the maximum number of injured visitors depends on the injury rate for each attraction. That means, if the park has more attraction that have higher injury rate, the more injured people we have at the end of the simulation. I also believes that it depends on the number of rides (many small rides). One downside of these hypotheses is that, the rides that have a certain space and rides with different injury risk has different space occupation. Another downside is the rides accessibility. Some of them are more accessible to visitors then others.

As for the second question, I believe that the wait time depends on the size of the queue. That also means that the more rides we have with a big queue the bigger is the average wait time for each group. It also depends on the length of each ride (the longer it takes for the ride to complete, the longer the people have to wait in queues). Thus I would say a combination of these two factors would maximize the wait time.

For this simulation we make the following assumptions:

- There will be a maximum space of 2650 slots ( 5 rides of each type of ride)
- The injury rate of the rides are:
    - Battleground: 28%
    - Dungeon: 33%
    - Arena 55%
    - Armory: 20%
    - Knife fight: 40%
- It takes one time step to load a ride
- The same distribution of visitors will arrive throughout the day
- Each person takes one space
- The space occupied in the park is the space of rides and their queues.

To answer the questions above with run the simulation with different parameters (number of each ride) while maintaining the same configuration file and calculate the average wait time, the number of people who visited the park, the number for people who got injured, and the percentage of injured visitors.

# 3-Approach:

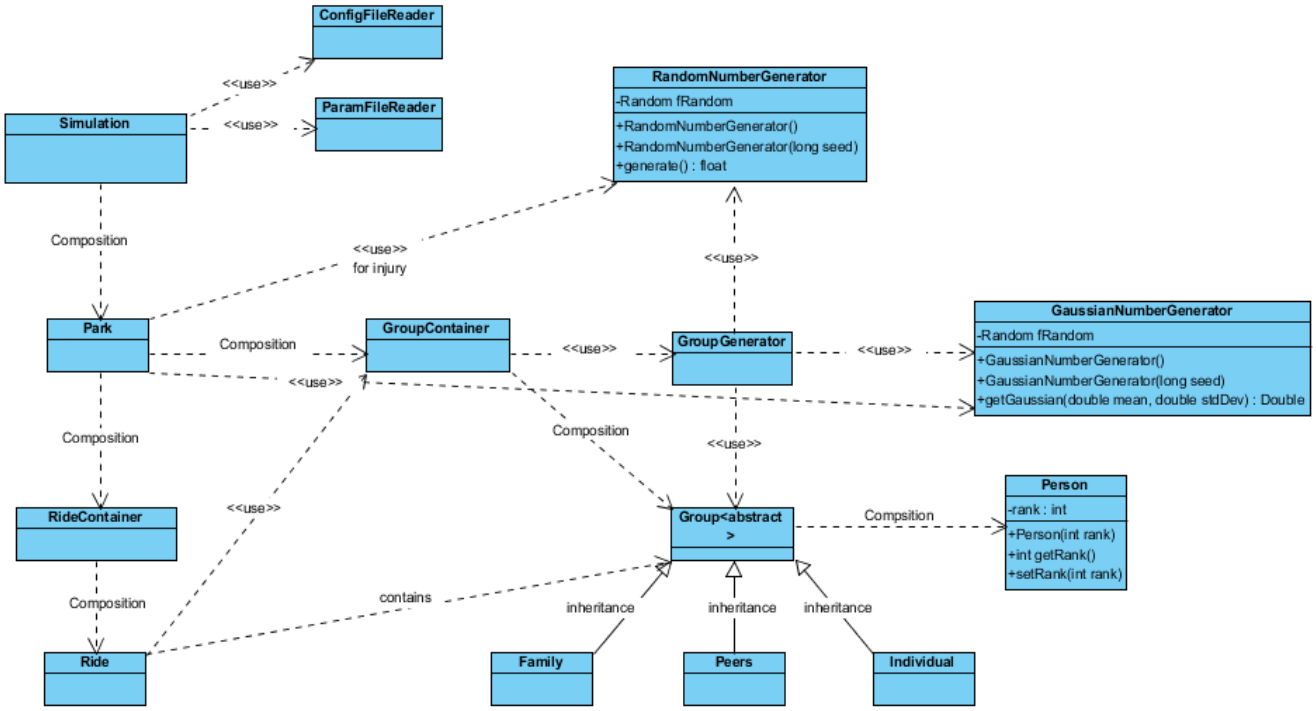


Figure 1 – Simplified class diagram of the project.

To design this project, we applied the concepts of object-oriented programming to follow the diagram shown above.

# Classes and Interactions:

- ***Simulation.java:*** This class is the controller / commander of the program. It reads the name of the Configuration and Parameter files (uses *ConfigFileReader.java* and *ParamFileReader.java* to deconstruct them and saves the data), it initializes the park, and runs the simulation while writing the output in 3 different output files. This class controls how the simulation will run and gets the inputs and outputs.

- ***ConfigFileReader.java:*** This class gets as parameters the name of the configuration file and then perform a keyword-based read of the data in it, analyses it, and saves the parameters in the correct variable for later use. These parameters controls the behavior of the park: arrival of groups, different possible rides and their parameters etc.… to Store the possible rides we use an *ArrayList<String>* which stores the groups that can access the ride for easier addition and access later.
- ***ParamFileReader.java:*** This class gets as parameter the name of Parameter file then it reads it, analyze it in a keyword-based manner to get the name of the rides in the park for the simulation and the number of each mentioned ride. To store which rides will be used for the simulation we use *ArrayList<String>* which contains the name of the rides for easier addition and access.
- ***Park.java:*** This class represents the actual park itself and where the simulation and data record happens.it takes as parameter the seed which controls the randomness and also two PrintWriters that will record the data. The park contains a *RideContainer* which stores the different rides in the park and *GroupContainer* which will hold the different groups that arrive to the park or unload from rides. Before the Park is run, it should initialized.
- ***RideContainer.java:*** This class as the name suggests will contain *Ride* elements. It allows the addition of new rides, accessing existing rides in the container, or getting the size of the container. It is has *ArrayList<Ride>* again for easier access and addition.
- ***Ride.java:*** this class represents the ride in the park and It stores all the data related it (size, length, queue size, etc.…) it also controls how the ride and their queue behave in the simulation by loading / unloading ride (which are *ArrayList<Group>*) and loading

the queues (which are *LinkedList<Group>:* we chose LinkedList structure for easier implementation of the queue).

- ***GroupContainer.java:*** Like RideContainer this class will be the storage for the different groups generated in the program. Since we're accessing elements in a linear method (first, second, third…) and we are deleting elements at random locations we decided that the container will be *LinkedList<Group>*.
- ***Group.java:*** This class is a generalization of the different groups of people visiting the park. It contains general methods applicable to all of them and behaves in a general manner to facilitate the manipulation of different group. Each group is made up of 1 or more persons thus making it a container for person thus we have *ArrayList<Person>* that makes a group. We also have *ArrayList<Integer>* which is used to track the rides that are already tried by the group (by saving the unique key of every ride in the park). This is also responsible for tracking the wait time for the group.
- ***Family.java/Peers.java/Individual.java:*** these three classes extends all the methods from the Group class with a minor modification in the constructor to make a distinction in the variation of groups.
- ***Person.java***: this class is the elementary constituent of the group class and represent the individual people in the park. Each person is assigned a rank upon creation and the class tracks if the person gets injured or not.
- ***GroupGenerator.java***: This class is responsible for generating the random groups (individual, family, or peer group) that arriving to the park every time step. It takes as parameters the different parameters written in the configuration file about the people to

generate a random group accordingly. It uses both GaussianNumberGenerator and RandomNumberGenerator to control the randomness of the groups.

- ***RandomNumberGenerator:*** responsible for generating random ranks for person, generating the different types of groups and distribution, and also used to decide whether a person is injured or not.
- ***GaussianNumberGenerator:*** Responsible for controlling the arrival rate, the number of companion in each family, and number of peers in every group of friends. To do that it generates a random integer in according to a normal distribution controlled by a mean and standard deviation passed as parameters.

## How Everything Comes Together:

After the configuration and parameter files are read using ConfigFileReader and ParamFileReader in the Simulation.java Class, the run method in the park.java class is run. This will use group generator to generate random groups (using arrival mean and standard deviation). These groups are stored in a group container. The rides then fill themselves and their queues from the Group Container and update their state every time step. At the end of every time step, the group container in the park is emptied (represents the people who completed all rides or can't find a free spot). The simulation goes for 1000 time steps.

# 4-Methods:

In order to answer all the questions mentioned above, we ran our simulation several times with different Parameters. For each Parameter set, we ran the program 5 times with different seeds and used the average results for the final report. Seeds used for this experiment are: 1, 25, 57, 77, and 101. After running the simulation, we gathered the data, analyzed and then changed the parameter file. (The total space occupied need to be maximized and <= 2650)

# 5-Data and Analysis:

## Parameter Set #1:

| Ride | Arena | Dungeon | Battleground | Armory | Knife Fight |
|---|---|---|---|---|---|
| Number | 5 | 5 | 5 | 5 | 5 |

For parameter set #1, we used what we call testimony Configuration, with 5 rides of each time to be the standard. We ran program for this parameter set in order to compare the results of new parameter files to the following ones:

**Space occupied:** 2650

**Average Wait Time:** 1.28691214 steps **Total percentage of injured People:** 24.9817964%

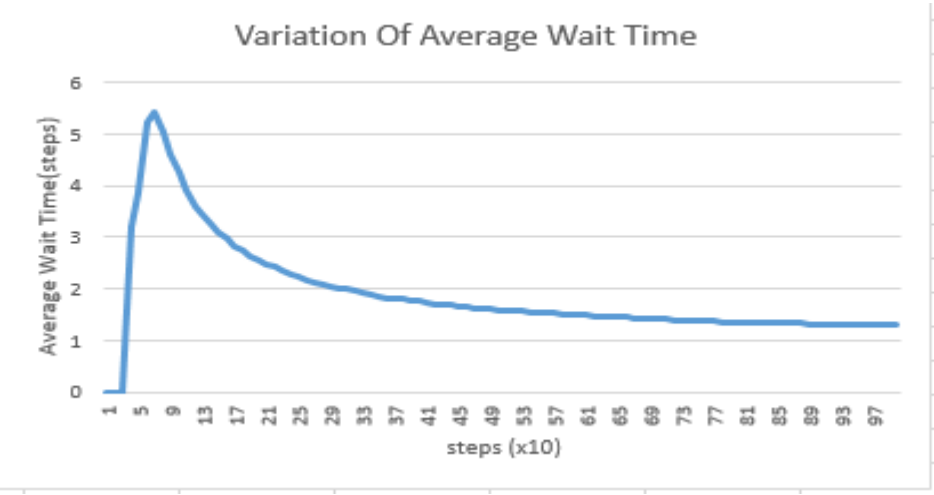


Fig 2 – Variation of Average Wait Time

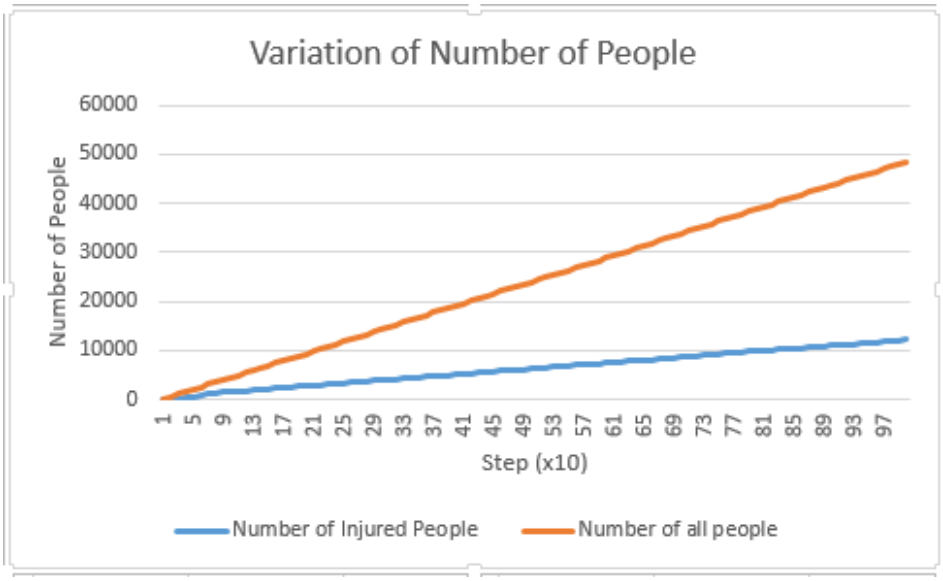


Fig 3 – Variation of Number of People

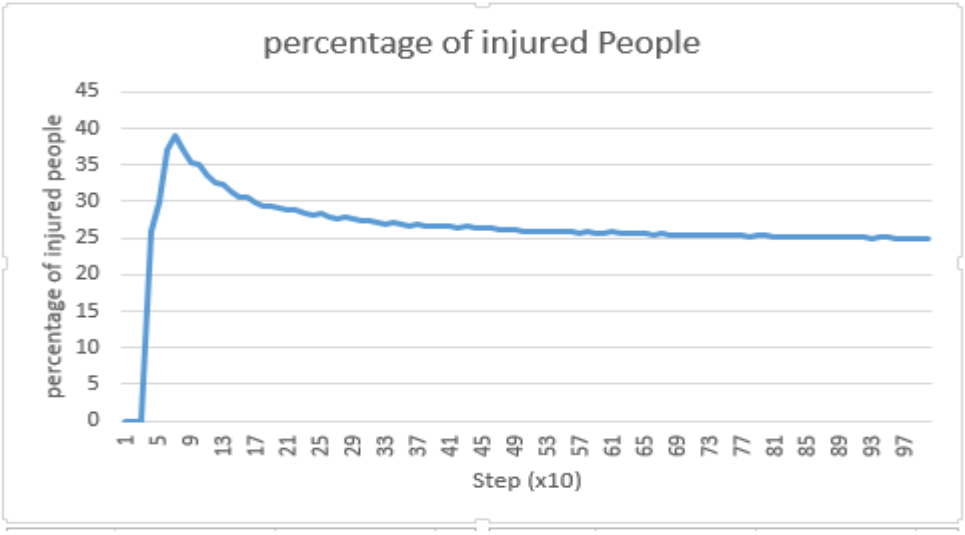


Fig 4 –Percentage of Injured People

## Parameter Set #2: increasing rides with higher injury rate

| **Ride** | Arena | Dungeon | Battleground | Armory | Knife Fight |
|---|---|---|---|---|---|
| **Number** | 13 | 1 | 1 | 2 | 1 |

For parameter set #2, we try to maximize the rides with the highest injury rate and see the outcome.

**Space occupied:** 2650

**Average Wait Time:** 0.928171554 steps **Total percentage of injured People:** 15.2160934%

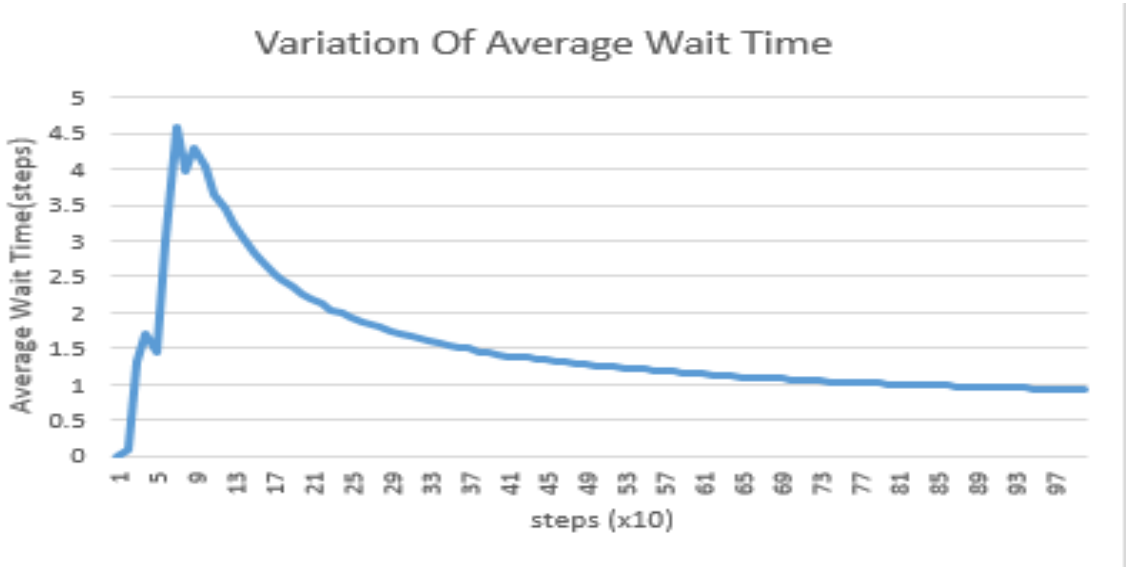


Fig 5 – Variation of Average Wait Time

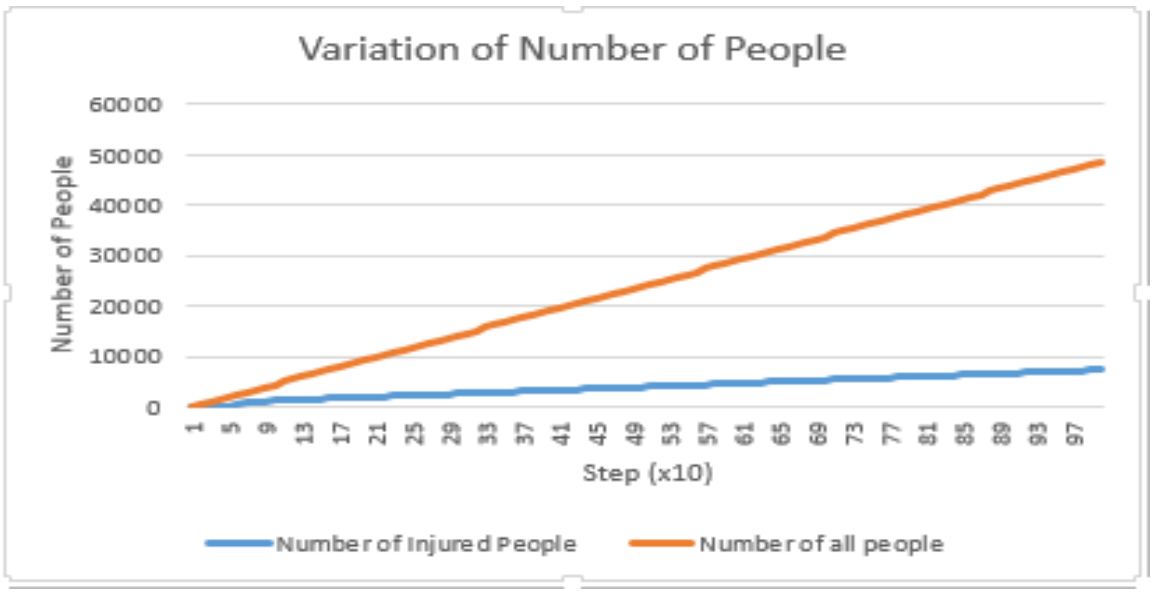


Fig 6 – Variation of Number of People

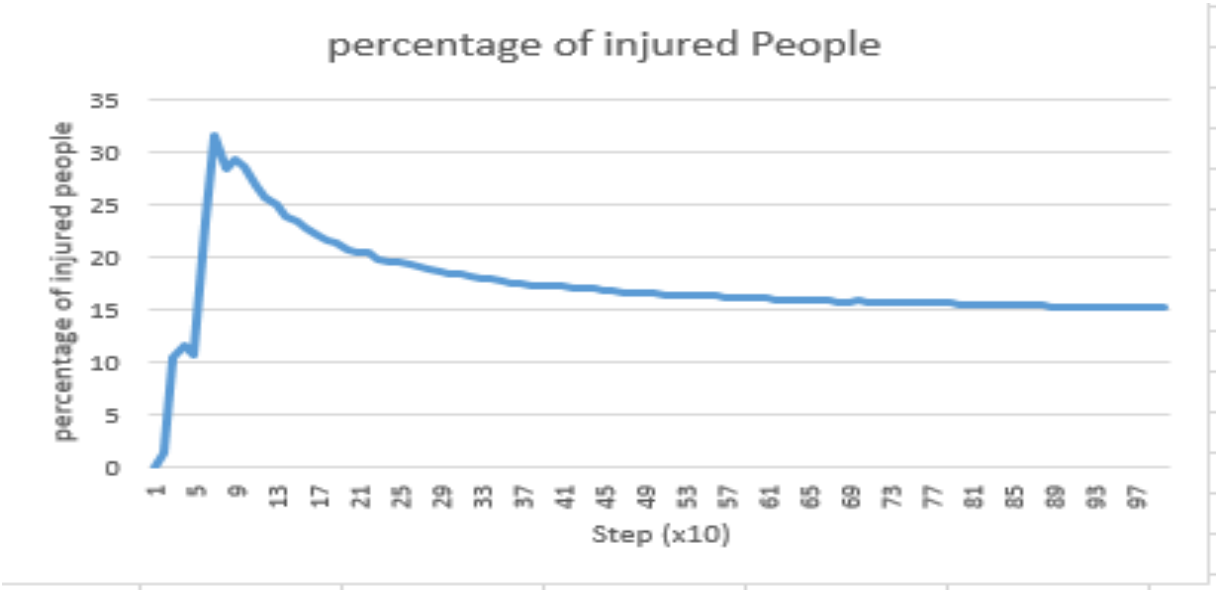


Fig 7 –Percentage of Injured People

When we tried to maximize the injury rate, both the wait time and percentage of injured people dropped drastically. This is in my opinion is due to the inaccessibility of the arena to all the types of groups. The arena only accept groups with royals and knights in it which only 2/5 of the group types. The rest would come to the park and leave instantly because they can’t find any compatible rides which lower the general wait time.

# Parameter Set #3: increasing number of rides

| **Ride** | Arena | Dungeon | Battleground | Armory | Knife Fight |
|---|---|---|---|---|---|
| **Number** | 0 | 2 | 34 | 9 | 0 |

For parameter set #3, try to maximize the number of rides (prioritize rides with the lowest)

**Space occupied:** 2640

**Average Wait Time:** 1.74502844 steps **Total percentage of injured People:** 25.239565%

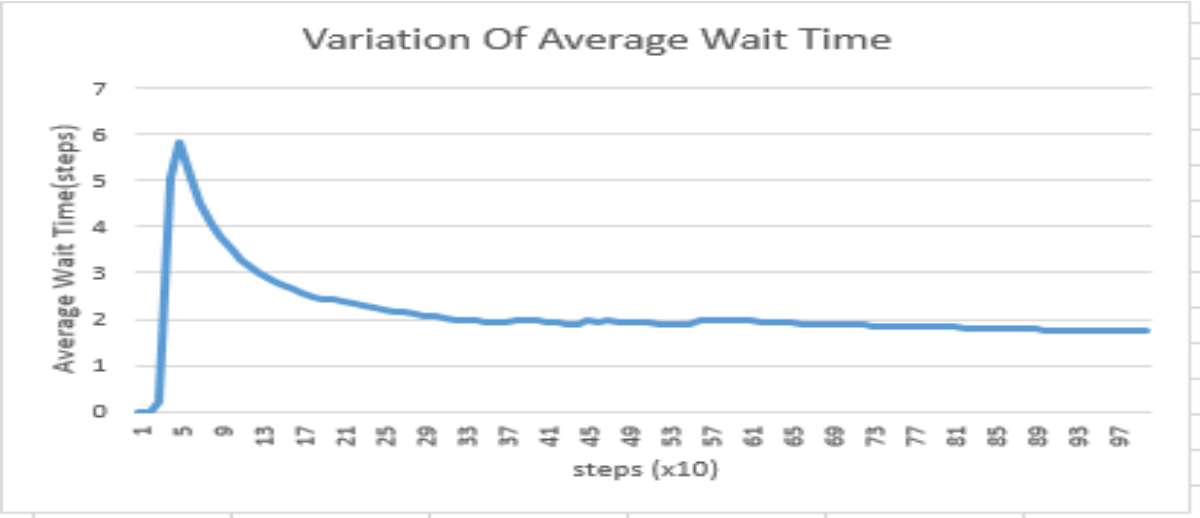


Fig 8 – Variation of Average Wait Time

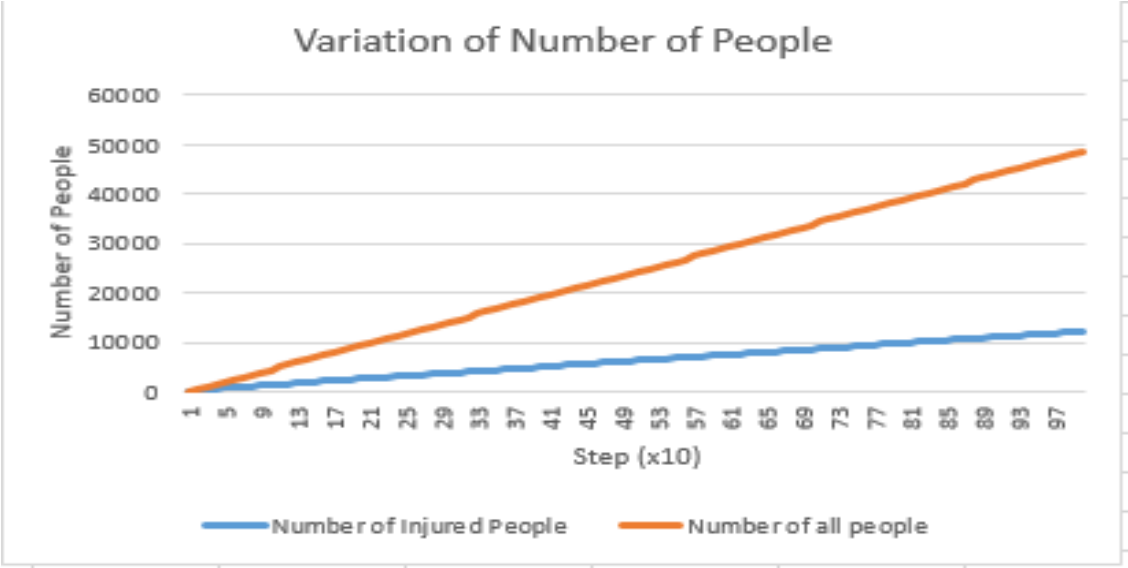


Fig 9 – Variation of Number of People

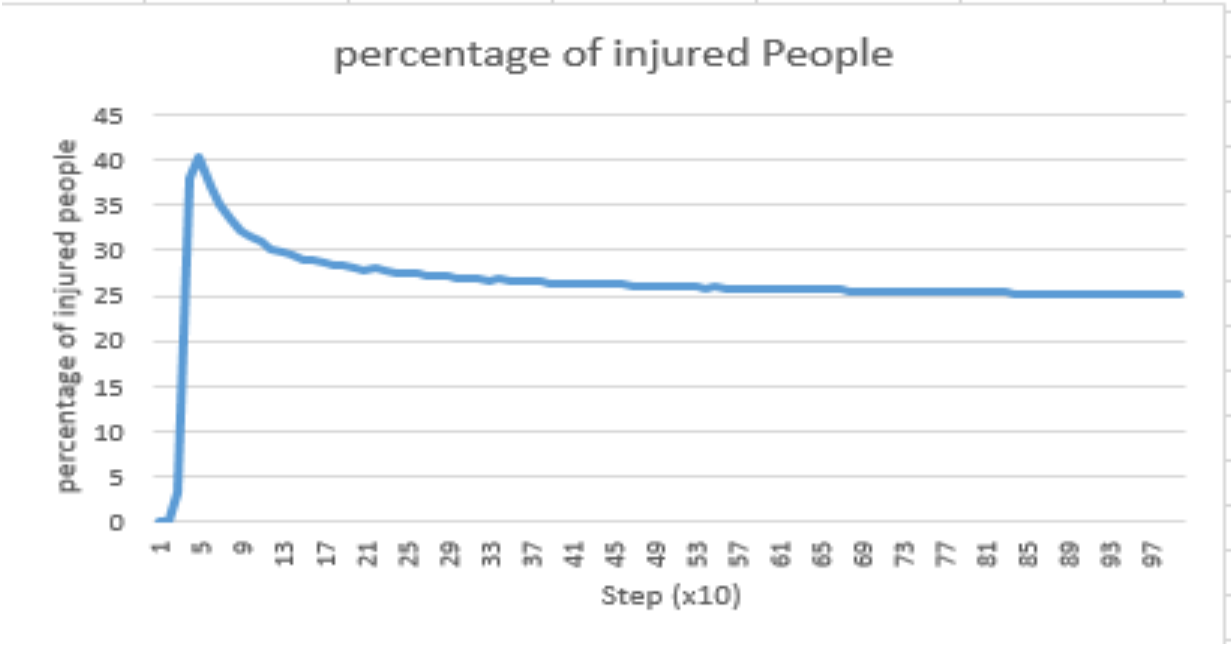


Fig 10 –Percentage of Injured People

When we tried to maximize number of rides in the park, we notice an increase in the average wait time. This might be explained by the increase in the number of queues in the park thus increasing the number of people who are waiting in line. However the number of injured people hasn't change noticeable and is close to the injury rate of the battleground which is the dominant ride in the park.

# Parameter Set #4: increasing number of rides with big queues

| **Ride** | Arena | Dungeon | Battleground | Armory | Knife Fight |
|---|---|---|---|---|---|
| **Number** | 0 | 14 | 1 | 1 | 8 |

For parameter set #4, try to maximize the number of rides with big queues so more people are in lines

**Space occupied:** 2630

**Average Wait Time:** 0.951633638 steps **Total percentage of injured People:** 17.3065902%

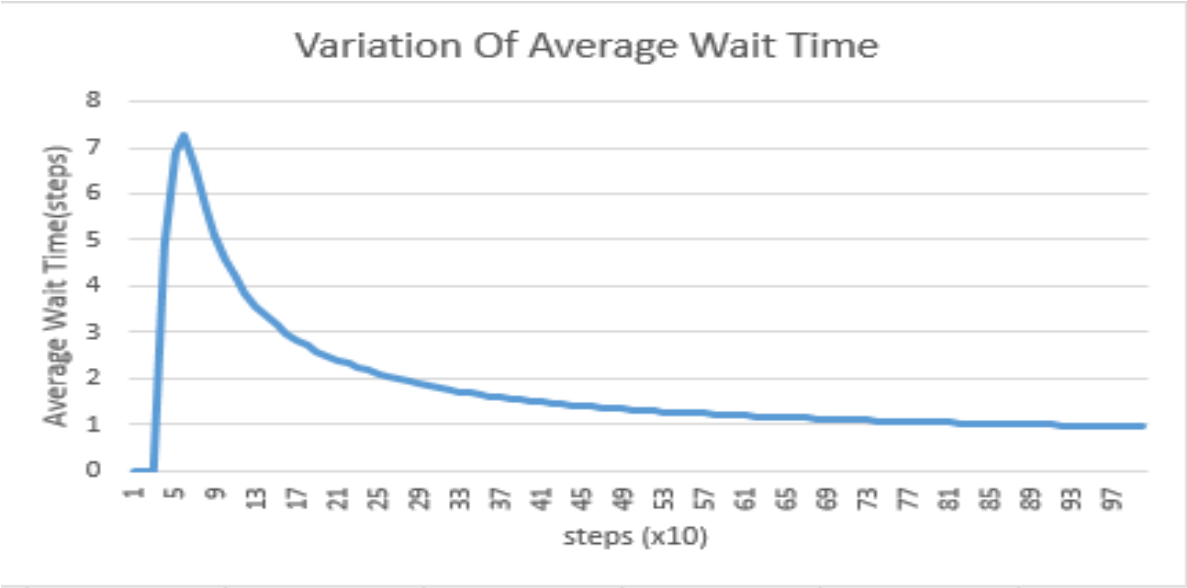


Fig 11 – Variation of Average Wait Time

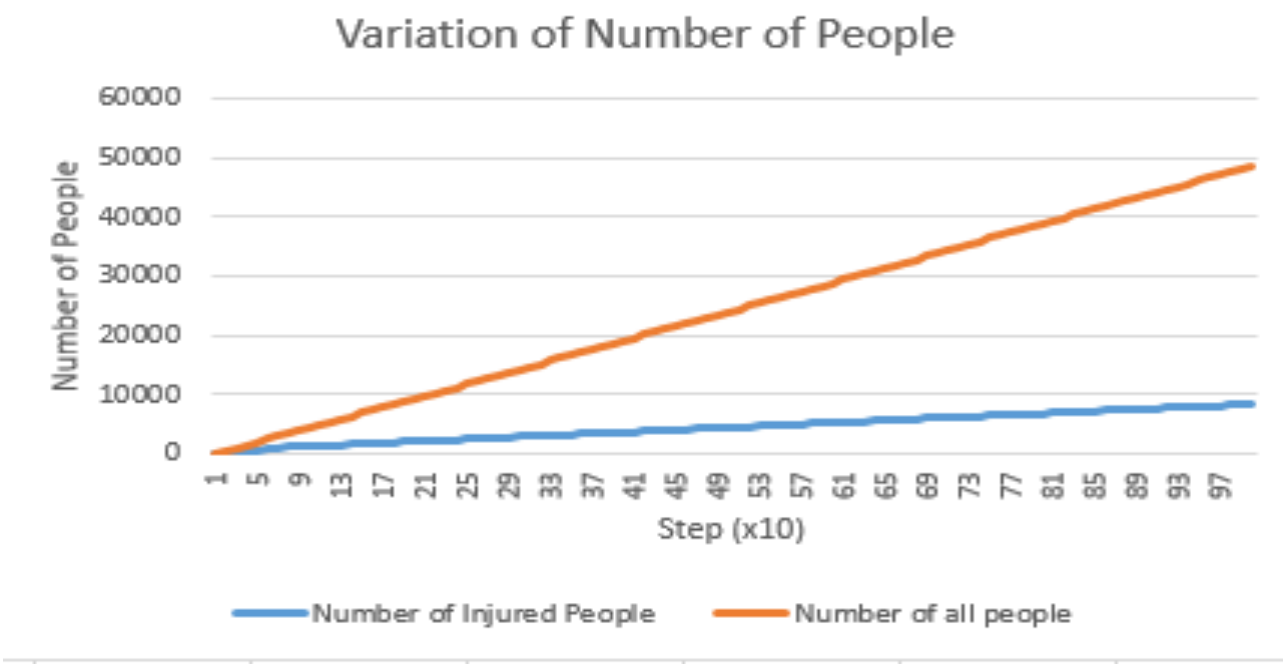


Fig 12 – Variation of Number of People

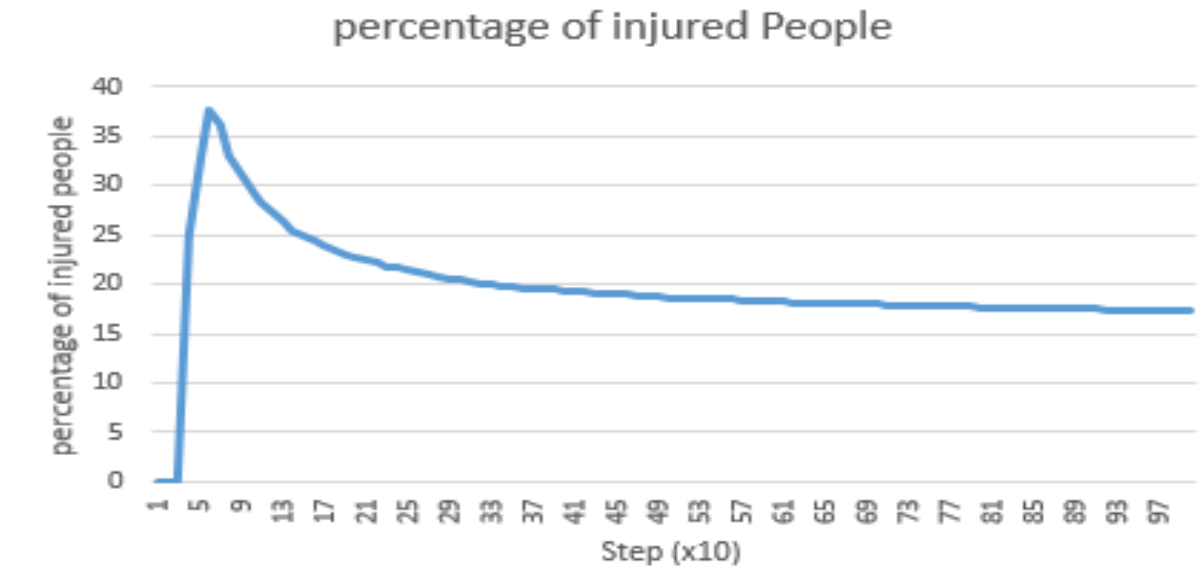


Fig 13 –Percentage of Injured People

When we tried to maximize number of rides in the park, we obtain results similar to Parameter set #2. The wait time has dropped due to the short length of the rides (3-5 ticks) so even though the queues are big, people no stay long enough in them. The decrease in percentage of injured people might be due to the inaccessibility of the Knife Fight rides which are only for servants and peasants.

# Parameter Set #5: increasing number of rides high lengths and rides that are accessible

| **Ride** | Arena | Dungeon | Battleground | Armory | Knife Fight |
|---|---|---|---|---|---|
| **Number** | 1 | 2 | 0 | 28 | 0 |

For parameter set #5, try to maximize the number of rides that take longer to complete, thus keeping the people more (with trying to minimize rides with high injury rate)

**Space occupied:** 2630

**Average Wait Time:** 8.2630486 steps **Total percentage of injured People:** 13.4244386%

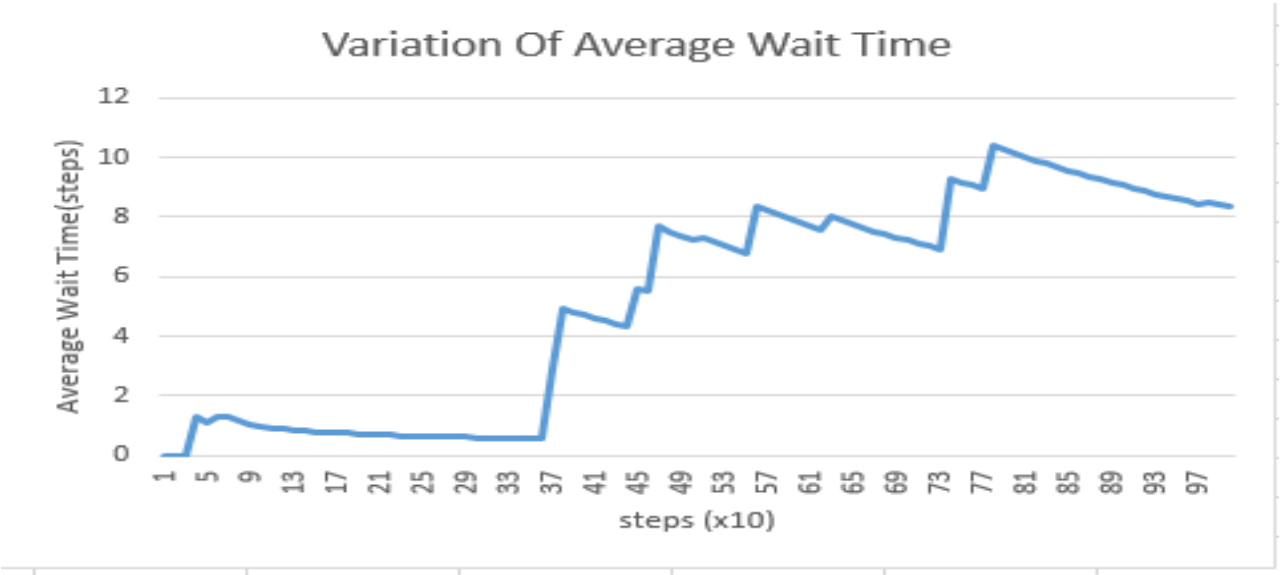


Fig 14 – Variation of Average Wait Time

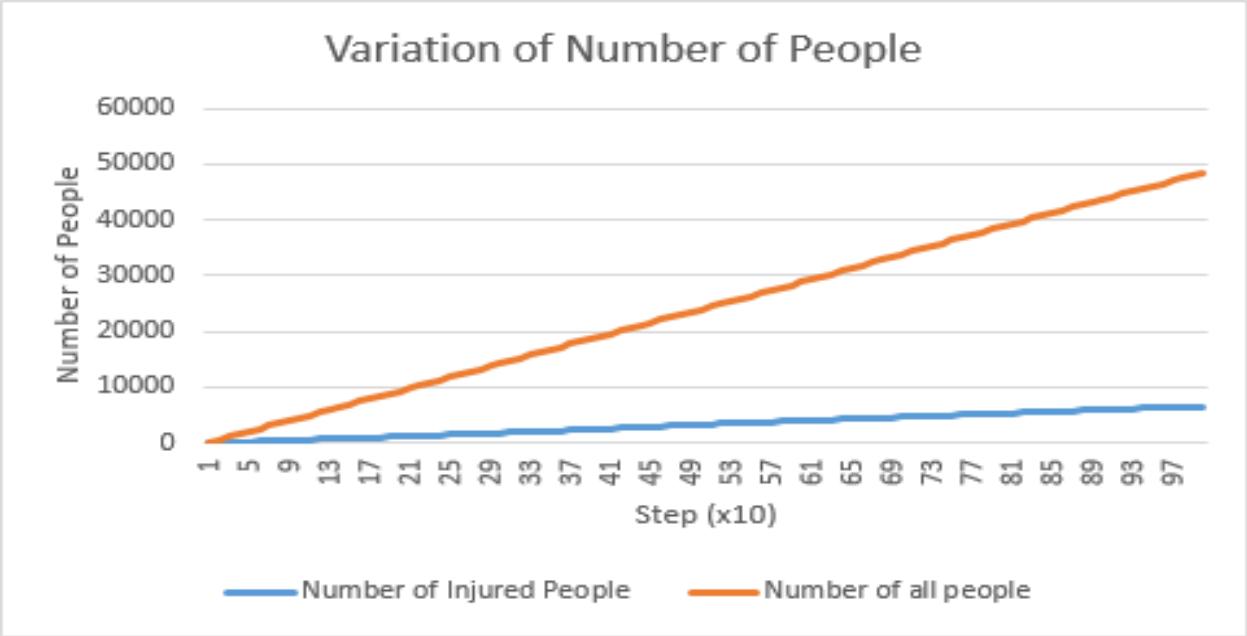


Fig 15 – Variation of Number of People

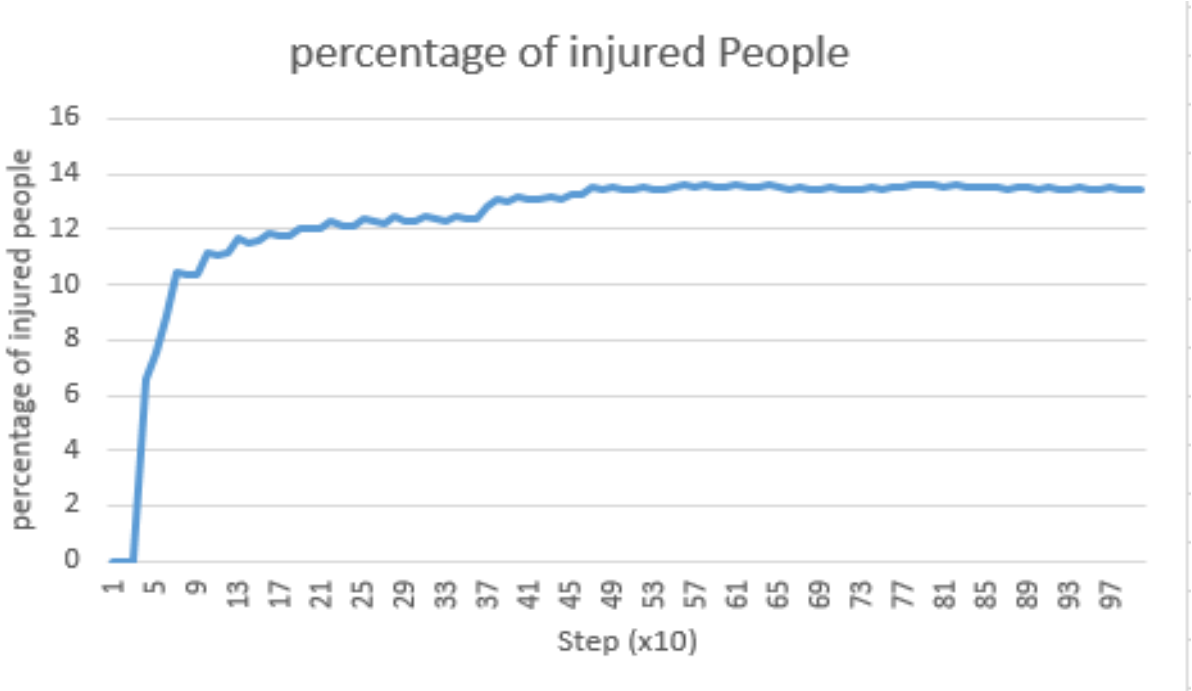


Fig 16 –Percentage of Injured People

When we try to increase number of rides high lengths and the rides that are accessible, we instantly notice an increase in the average wait time per group by 800% which as we suggested might be related to the length of the ride (people wait linger) and the accessibility of rides (people can find queues to stay in)

# Parameter Set #6: mixing #3 and #5: increase length accessibility and number with a relatively high injury rate

| **Ride** | Arena | Dungeon | Battleground | Armory | Knife Fight |
|---|---|---|---|---|---|
| **Number** | 0 | 0 | 21 | 20 | 0 |

For parameter set #6, try to combine both experiment 3 (high injury rate) and experiment 5 (high wait time). To see what the result would be.

**Space occupied:** 2650

**Average Wait Time:** 3.59507944 steps **Total percentage of injured People:** 18.6099938%

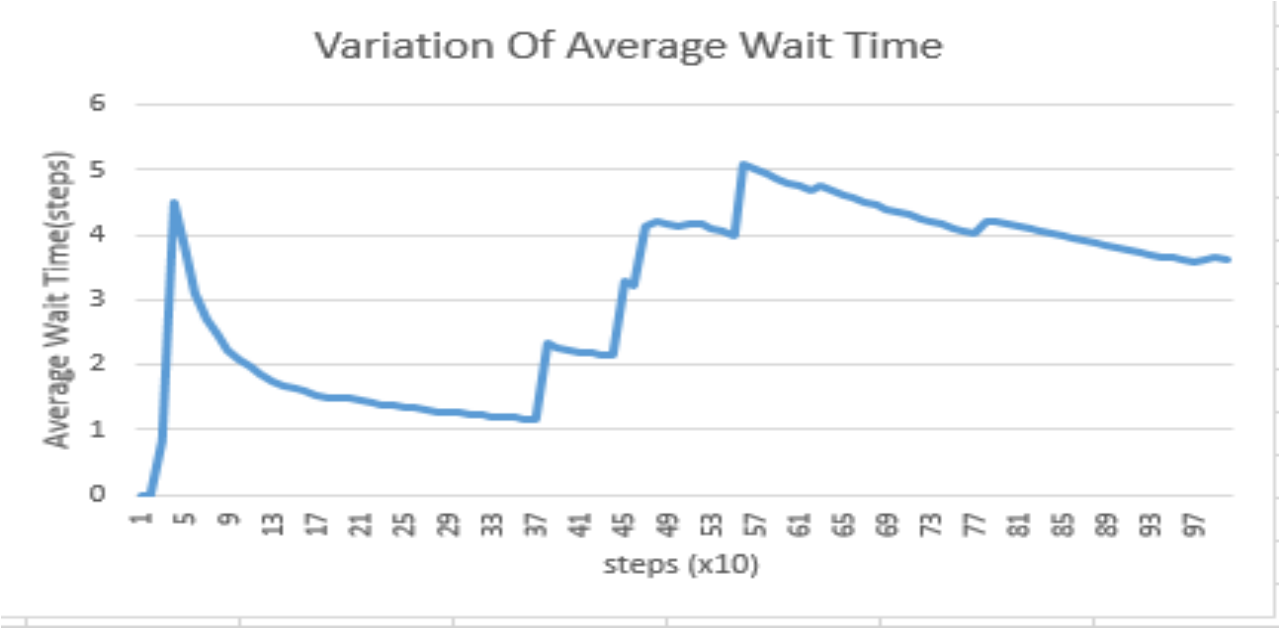


Fig 17 – Variation of Average Wait Time

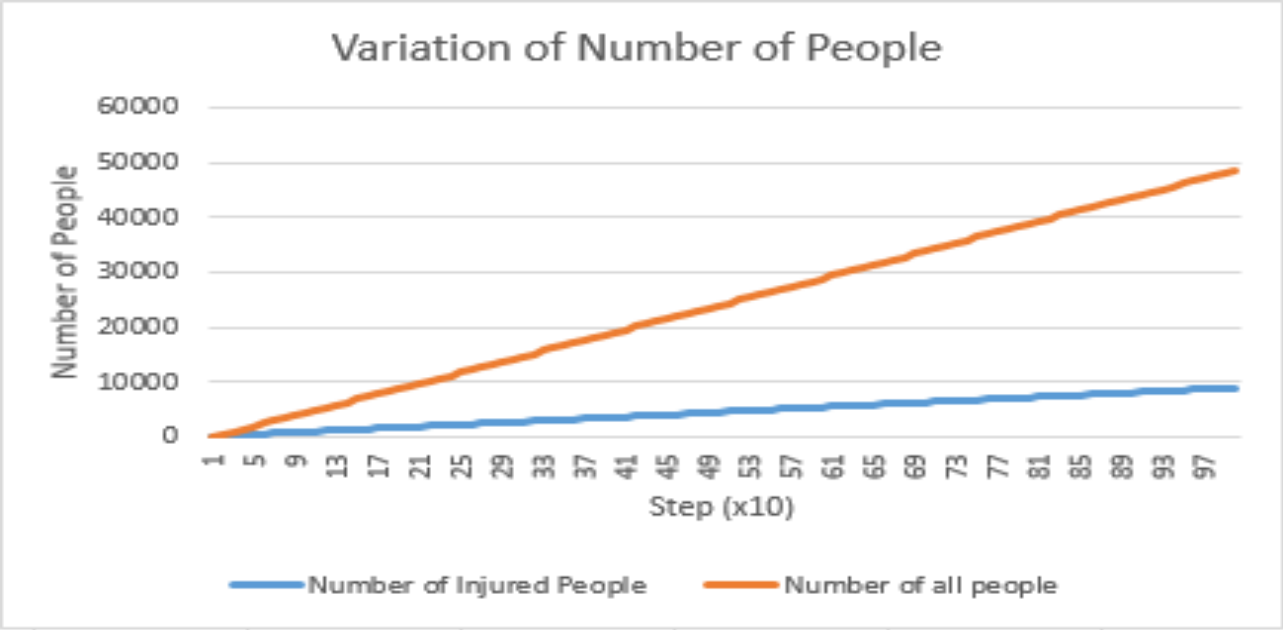


Fig 18 – Variation of Number of People

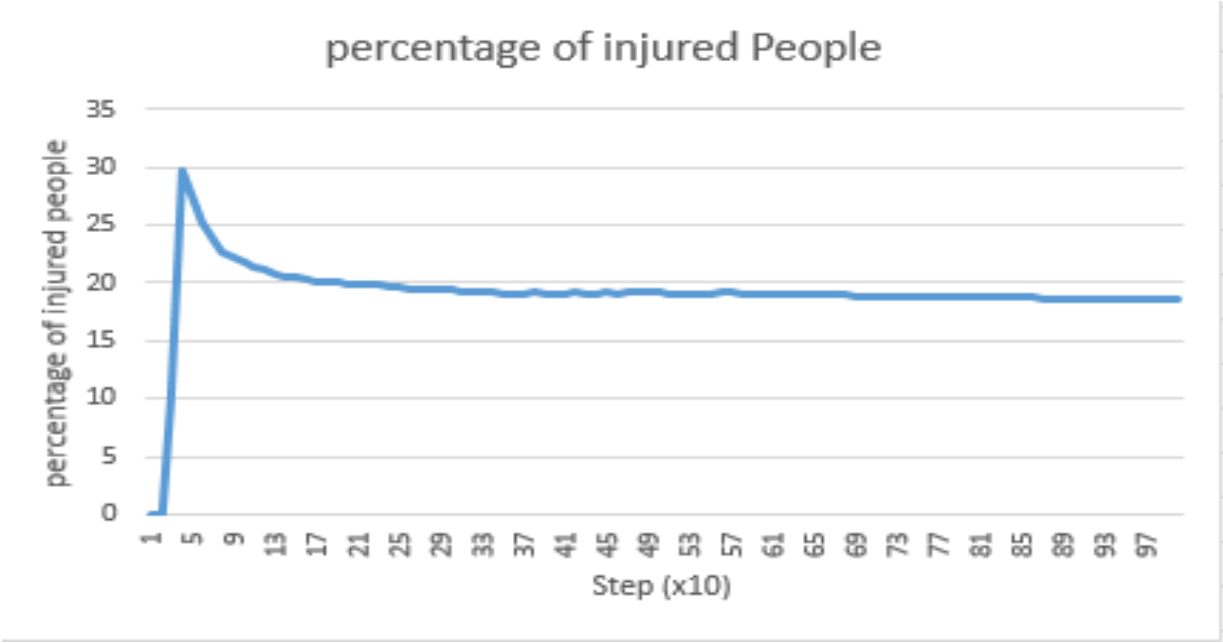


Fig 19 –Percentage of Injured People

When we try to increase number to combine these two approaches we end up with a mean result: the wait time increases by around 350% and the injury rate drops by around 28%.

# 6-Conclusion:

From the analysis we conducted above we get 3 ways of approaching the goal:

- If we want to maximize the injury (the king wants to completely disable a good portion of the population) the Park should be built in such a way that it maximizes the number of small rides (battleground) which will allow more people to participate in the games thus be more likely to be injured (specially injury rate is high for those rides compared to bigger ones)
- If we want to maximum wait time (the king want the citizens to be busy while he amass his fortune). We should maximize the ride which have a longer duration (armory) thus making the people wait longer in line (Especially that these rides are accessible to almost everyone).
- If we want an in-between result, we should go with a parameter set that combines the two approaches above.

# 7-References:


1. Liew, Chun W. *CS150 Project 1 Description*. Retrieved March 4th 2018 from https://moodle.lafayette.edu/pluginfile.php/380686/mod_resource/content/3/p1.pdf
2. "Java Platform SE 8." *Java Platform SE 8*. Retrieved March 6th 2018 from https://docs.oracle.com/javase/8/docs/api/
3. Gaussian Random Number Generator (normal distribution). *Generate Random Numbers*. Retrieved February 25th 2018 from http://www.javapractices.com/topic/TopicAction.do?Id=62 .