5-2020

Efficient Elevator Algorithm

Sean M. Toll
University of Tennessee, Knoxville, stoll@vols.utk.edu

Owen Barbour
University of Tennessee, Knoxville, obarbour2@gmail.com

Carl Edwards
University of Tennessee, Knoxville, cnedwards3@gmail.com

Daniel Nichols
University of Tennessee, Knoxville, danielnichols1998@gmail.com

Austin Day
University of Tennessee, Knoxville, aday12@vols.utk.edu

Recommended Citation

Toll, Sean M.; Barbour, Owen; Edwards, Carl; Nichols, Daniel; and Day, Austin, "Efficient Elevator Algorithm" (2020). *Chancellor’s Honors Program Projects.*
https://trace.tennessee.edu/utk_chanhonoproj/2338

Efficient Elevator Algorithm

ECE401/402: Final Detailed Design Report

Austin Day, Carl Edwards, Daniel Nichols, Sean Toll, Owen Barbour

24 April 2020

[illegible] [illegible]

[illegible] [illegible]

[illegible]

Executive Summary

Team 28's goal is to leverage modern deep reinforcement learning techniques to optimize elevator efficiency in tall, population-dense buildings. Such a feat will ultimately improve the lives of both elevator users and elevator owners. For this project, the top five Engineering Characteristics are: 1) integrating the reward functionality and simulations, 2) creating the reward function by optimizing the transportation of people, 3) creating a good training set, 4) simulating the physics and other variables (number of floors, distance between floors, etc.), and 5) developing a base model to determine the performance increase due to the implementation.

Contents

List of Figures

List of Tables

1 Problem Definition

“Why is this elevator taking so long?” This is a common thought that has popped into everyone’s head at some point in their lives, spurred on by elevators that leave much to be desired. Some are slow, taking ages to go from floor to floor. Others are baffling, going up when someone wants to go down or sending the most distant unit to pick up a user when there is clearly a closer one nearby. Granted, the speed of the elevator is limited by the physics of a massive metal box moving vertically through a building. However, the frequency of weird actions, the ones that are triggered by a programmable algorithm, can surely be reduced. While no elevator algorithm will ever reach perfection, they can certainly get closer. That is the objective of the team’s project.

Team 28 project’s success will chiefly benefit two customer groups: elevator users and elevator owners. For users, the benefit is clear: a better elevator experience. For owners, the benefit is more indirect, yet no less significant: a higher profit margin. To explain, regardless of the business sector, happier people means better business. These people can either be employees or customers, and, therefore, this increase can take the form of greater productivity, increased sales, or more positive reviews. No matter the channel, the end result remains the same. Improved elevators will make people happier, or, at the very least, will make people less unhappy. This effect will cascade until it results in more money for the business the elevator is supporting. Assuming the owner of the elevator has some stake in the business, if they are not already a part of it, they will reap the benefits of this increase. In this way, one elevator will have made a bigger difference than most would have anticipated. Granted, nothing ever works out as perfectly as the example above describes. However, it remains true that even small things like elevators can hit far above their weight.

2 Background

Although most elevator manufacturers offer their own proprietary algorithms which are mostly treated as trade secrets [1], this problem and its hypothetical optimizations have been tackled in several studies available to the public. In a 1996 study by Robert Crites and Andrew Barto [2], it was found that training an elevator algorithm using reinforcement learning offered notable improvements over older dynamic programming approaches to solving large scale optimization problems in various aspects, including performance and memory requirements. “These results demonstrate the utility of [Reinforcement Learning] on a very large scale dynamic optimization problem. [...] RL avoids the need of conventional [Dynamic Programming] algorithms to exhaustively sweep the state set. By storing information in artificial neural networks, it avoids the need to maintain large lookup tables” [2]. A later study by the same researchers in the year 1998 went deeper into this, comparing various reinforcement learning techniques to existing methods: “We used a team of RL agents, each of which was responsible for controlling one elevator car. Results obtained in simulation surpassed the best of the heuristic elevator control algorithms of which we are aware. Performance was also robust in the face of decreased levels of state information" [3]. Other more recent studies also exist, however most of these do not account for all of the factors which the group intends to include in the study. In particular, the monitoring of foot traffic and consideration of time of day and past foot traffic patterns will be used to add more context to optimize the algorithm, and hopefully achieve new improved results.

3 Requirements Specification

3.1 Customer Requirements

Table 1: Customer Requirements

Customer Requirement	**Description**
Accurate training set development	The use case is a high density buildings with a large number of floors
Normal implementation base model	A metric to determine the effectiveness of the improved algorithm
Deep reinforcement learning implementation	Takes into account the number of people waiting per floor, waiting time per person, and the number of elevators
Collection of data from UT campus elevator*	Base model for implementation on UT's campus
Implementation on UT campus*	Implementation on an existing building on UT's campus to measure the actual improvement

The customer requirements were developed by the initial request specifications and from the group's idea of what the best path forward is. Since the requirements are largely sequential, each of the steps need to be completed mostly in order. The (*) requirements are not necessarily going to be done: the collection of data on UT campus is an alternative design, and implementation on UT campus is time allowing as this is not the focus of the project. Thus, the simulation and reinforcement learning are the primary focuses.

3.2 Engineering Requirements

Table 2: Engineering Requirements (continued on next page)

Engineering Characteristic	**Target values**	**Units/Directions for improvement**	**DV/ DC**
Reinforcement learning implementation	Integrates the reward and simulations	Further research and use of modern, deep reinforcement learning techniques	DV
Reward function	Optimizes transporting people as effectively as possible	Best combination of variables including the number of people waiting per floor, waiting time per person, and number of elevators	DV
Training set requesting	Variable number of elevator requests	Take into account different densities in sections of floors to better model skyscraper distribution	DC
Training set physics simulation	Models people arriving and travel time. Accounts for multiple elevators, number of floors, and distance between floors	More complex variables such as elevator weight	DC
Normal implementation base model	Best replication of how elevators work in practice currently	Replication of "proprietary" algorithms used commercially	DC
Reinforcement learning floor priority	Certain companies necessitate/want priority for floors (ex. Hotel rooms vs staff), so have a floor range priority system	Possible development of ranking system	DV
Installation of sensors*	Sensors for collecting the number of people, waiting times, button presses, etc.	Expand to other elevators at the UT campus	DC

Table 3: Engineering Requirements Continued

Measure actual elevator use*	Specific UT elevator measurements	Expand to other elevators at the UT campus	DC
Implementation on physical elevator*	Integration of deep learning algorithm with elevator equipment, sensors, and model	Make easily expandable to other systems	DV

DV = design variable, DC = design constraint

Each of these Engineering Characteristics is a subset of the customer's requirements: for example, the deep reinforcement learning implementation is broken down into a reward function, a possible floor priority implementation, and an integration with the simulation. Similarly, the training set development is the requesting (elevator button presses) and the base model implementation encompasses the physical simulation and the implementation. As for the characteristics dealing with sensor and physical elevator implementation, this is again not the primary focus of the project. This is only time allowing and would be the appropriate followup going forward into a production environment.

For the target values, they were determined from the initial specifications sheet, the research, and intuition. For the non-starred values, these represent the base requirement for the deliverables. Further development can be seen in the directions for improvement.

4 Technical Approach

4.1 Simulation

The approach to the elevator simulation can be broken down into three main steps: research, programming, and testing (Figure 1).

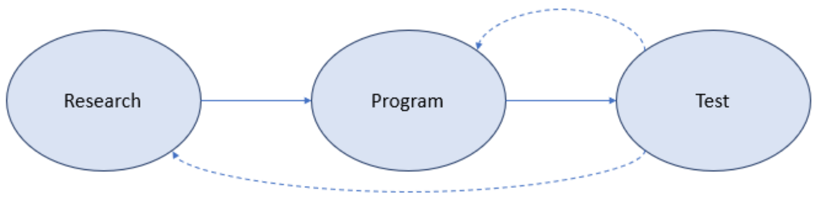


Figure 1: Engineering design pipeline for simulation creation.

In the research phase the focus will be on understanding how elevators work. While this will of course necessitate a high level investigation of elevator physics, the specific goal of this phase will be to discern how the various aspects of elevators (weight, speed, maximum load, etc.) interact with each other. For example, the trade-off between the weight of an elevator and its resulting speed will be a key area of focus, among many others. This research phase will mostly consist of online searches and physics calculations. However, it will also be necessary to verify the findings by observing a real-life elevator (such as one of the elevators in the Min Kao building). Through this additional analysis it will be possible to incorporate some elements of practical reality into the simulation, since theoretical calculations can only go so far.

In the programming phase the objective is simple: create the simulation. To do this python will be used, because it is easy and there is a lot of library support behind it. Further, the entire group is familiar with Python. As such, it is the most straightforward choice. Importantly, this simulation will by no means need to be perfect or complete by the end of the phase; however, the core functionality will need to be implemented. Additionally, it is critical that that simulation be designed with future expansion and modification in my mind. As the project goes on, it quite likely that new parameters will need to be added and old ones will need to be adjusted or even discarded. If the code is poorly structured, this revision process will be much more difficult. Therefore, it will be imperative that the simulation is coded well.

Once the simulation is fully implemented the testing phase will begin. The goal here is two-fold: ensure that the simulated elevator obeys the laws of physics and then improve its applicability to real-world elevators. The first part of the goal will require physics calculations and other mathematical checks. It is in some ways the most vital part to get down as an optimized elevator algorithm is worthless if its not physically possible. Once the physical constraints are successfully accounted for, the second part, real-world application, can begin in earnest. This will first necessitate that the parameters and performance of elevators around campus be recorded. Then these parameters will be inputted into the simulation and its performance will be compared to that of the campus elevators. The goal is to make simulation's performance as similar as possible. If this not the case after a round of testing, then it will be necessary to go back to the programming phase or even the research phase. This cyclical process will continue until the simulation is to the group's and more importantly the customer's satisfaction.

A summary of the major design decisions and the relevant engineering characteristics can be found in Table 3 below.

Table 4: Simulation Design Decisions

Design Decision	**Engineering Characteristic**
Programming a simulation	Training set requesting, Training set physics simulation, Normal implementation base model
Using python*	Training set physics simulation

(*) Minor Design Decision

4.2 Control Algorithm

In order to find the best elevator algorithm Team 28 will be trying to find the optimal greedy policy for the simulation's Markov model. As discussed above, previous work has used more traditional Q-learning techniques. The group will be investigating a recent innovation within reinforcement learning: the deep Q-network [6]. To turn the simulation into a Markov decision model, the team is going to use two two actions to transition between states. These actions are applying a force up and down and Team 28 will create a reward which incorporates travel time and waiting time for each person.

The objective of deep Q-learning is to train a deep learning model which approximates the estimated return of each action from some state. Since the state is not a specific type of data, the group will use a multilayer perceptron (MLP) with two hidden layers as an initial model. Next the team will use the deep Q-learning algorithm to train this model using mean-squared error as the loss function. The following is the loss function, which will be used from [6]:

$$L_i\left(\theta_i\right) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} \mathcal{Q}(s', a'; \theta_i^-) - \mathcal{Q}(s, a; \theta_i)\right)^2\right]$$

In this equation, s is the current state, a is the action taken,, r is the reward for taking these actions, s' is the next state, and a' is the action taken from the next state. γ is the discounting factor. The Adam optimizer will be used to minimize this loss function. $\mathcal{Q}$ is the action-value function that will be learned (the multilayer perceptron). $U(D)$ is a uniform distrubution of transitions in the replay buffer D. Θ is the model parameters.

In order to program this algorithm Team 28 will create a environment wrapper class for the simulation. This will include a reset function to create a new episode and a step function to take one timestep and get the reward for that. Since elevators are a nonepisodic task, it isn't important for us to reset the episode however. The group will collect state transitions into a replay buffer and use these to sample minibatches to train the network.

To implement this model the team will create this deep Q-learning algorithm in Python 3 using the deep learning library PyTorch. PyTorch is a good choice because it is a flexible, efficient library for deep learning calculations while still being relatively easy to use.

A summary of the major design decisions and the relevant engineering characteristics can be found in Table 4 below.

Table 5: Control Algorithm Design Decisions

Design Decision	**Engineering Characteristic**
Using deep Q-learning algorithm [6]	Reinforcement learning implementation
Using a Markov model with one state and up/down actions*	Training set requesting, Reward function, Reinforcement learning floor priority
Using Adam optimization [4] *	Reinforcement learning implementation, Reward function
Using multilayer perceptron*	Reinforcement learning implementation
Using PyTorch [7] for implementation*	Reinforcement learning implementation, Reward function
CPU Training*	Reinforcement learning implementation

(*) Minor Design Decision

5 Design Concepts, Evaluation & Selection

5.1 Concept Evaluation

5.1.1 Programming a simulation

Alternative Concepts:
One alternative to a simulation would be to use an existing elevator usage dataset. Another alternative would be to record the parameters and performance of elevators around campus and use these as the dataset.

Performance Predictions:

Table 6: Performance Prediction for Design Concept: Programming a Simulation with Custom Data Set via Simulation

Engineering Characteristic	**Performance Prediction**	**Explanation**
Training set requesting	5	With this option, the group will be able to completely customize what elevator requests are going into the simulation.
Training set physics simulation	4	The physics can be researched through product specifications to be fairly accurate.
Normal implementation base model	4	Custom implementation for accuracy.

Table 7: Performance Prediction for Design Concept: Programming a Simulation with Existing Elevator Data Set

Engineering Characteristic	**Performance Prediction**	**Explanation**
Training set requesting	2	With this option, there is no customization in what requests are going in. To change anything is cumbersome.
Training set physics simulation	5	The physics are measured so they are completely accurate.
Normal implementation base model	3	The base model will not be as accurate without custom data for the research focus.

Table 8: Performance Prediction for Design Concept: Programming a Simulation By Measuring Real World Data Manually

Engineering Characteristic	**Performance Prediction**	**Explanation**
Training set requesting	2	With this option, there is no customization in what requests are going in. To change anything is cumbersome.
Training set physics simulation	5	The physics are measured so they are completely accurate.
Normal implementation base model	2	Any elevator that can be measured is not the use case of high floor count.

Weighted Decision Matrix:

Table 9: Weighted Decision Matrix (Programming a simulation). Weight* Rating = Score

		Concepts					
		Simulation		**Existing Dataset**		**New Dataset**	
Criteria	**Weight**	**Rating**	**Score**	**Rating**	**Score**	**Rating**	**Score**
Applicability	0.3	1	0.3	3	0.9	2	0.6
Utility	0.3	3	0.9	0	0.0	0	0.0
Time Spent	0.3	1	0.3	3	0.9	-3	-0.9
Creativity	0.1	3	0.3	-1	-0.1	0	0.0
Total			1.8		1.7		-0.3
Rank			1		2		3
Continue?			**yes**		**no**		**NO**

Using an existing dataset would not be useful for later optimization algorithms if one even exists. The same can be said for creating a new dataset except doing this would require far too much work. Both of these concepts are also quite boring. Finally, while they would provide greater real-world applicability than a simulation, it is not worth the decreased usefulness in the later optimization algorithms. Therefore, programming a simulation is the superior design concept.

Selection Explanation:
A simulation is better for this project, because it will allow for directly manipulating the different parameters. This will allow the group to draw causal

relationships between the parameters and the simulated elevators performance. This will come in handy when developing the optimization algorithm.

5.1.2 Using python

Alternative Concepts:
One alternative is to use C++ to code up the simulation. Another alternative is to use Matlab.

Selection Explanation:
Python will be used, because it is easy and has a lot of hlepful library support. Additionally, the entire group is familiar with it.

5.1.3 Using deep Q-learning algorithm

Alternative Concepts:
Depending on the success of the deep Q-learning approach the group will compare against a more traditional tabular Q-learning approach. Additionally, the group will also consider a policy gradient model. Policy gradient is guaranteed to converge to a local optimum which is good. However, the policies are probabilistic and not deterministic and you need considerable knowledge of how exactly the system works to make your policy definitions. Also, policy gradient is able to handle continuous actions which is an option irrelevant to the current situation.
Performance Predictions:

Table 10: Performance Prediction for Design Concept: Optimization Algorithm with Deep Q-learning

Engineering Characteristic	**Performance Prediction**	**Explanation**
Reinforcement learning implementation	5	It leverages the power of deep learning while integrating it into q-learning.

Table 11: Performance Prediction for Design Concept: Optimization Algorithm with Policy Gradient

Engineering Characteristic	**Performance Prediction**	**Explanation**
Reinforcement learning implementation	4	It is probabilistic instead of deterministic and needs considerable knowledge of the system in order to create good policy definitions.

Table 12: Performance Prediction for Design Concept: Optimization Algorithm with Tabular Q-Learning

Engineering Characteristic	**Performance Prediction**	**Explanation**
Reinforcement learning implementation	3	This project uses a continuous state space.

Weighted Decision Matrix:

Table 13: Weighted Decision Matrix (Using deep Q-learning algorithm). Weight * Rating = Score

		Concepts					
		Deep Q-Learning		**Policy Gradient**		**Tabular Q-learning**	
Criteria	**Weight**	**Rating**	**Score**	**Rating**	**Score**	**Rating**	**Score**
Implementation Efficiency	0.1	0	0	0	0	3	.3
Implementation Cost	0.2	1	0.2	0	0.2	2	0.4
Implementation Difficulty	0.3	-1	-0.1	-2	-.6	2	0.6
Elevator Algorithm Quality	0.4	3	1.2	3	1.2	-2	-0.8
Total			1.3		0.8		0.5
Rank			1		2		3
Continue?			**Yes**		**Maybe**		**No**

A tabular approach would be more efficient. Unfortunately, it will be limited to a discrete state space and it will struggle to solve very large state spaces. The team want to use a continuous state space so the deep learning algorithms (Deep Q-learning and policy gradient) are important. They are less computationally efficient and trickier to implement. However, the group believes that the flexibility they provide will make up for the increase in computations and model complexity.

Selection Explanation:
Deep Q-learning will allow us to learn the best action to move in. It allows tabular Q-learning to be extended to continuous input domains, which is desirable for many tasks. It can also produce a deterministic action policy which policy gradient does not.

5.1.4 Using a Markov with one state and up/down actions

Alternative Concepts:
As another design option the group will also consider a simplified Markov decision model by modelling the simulation based on the current elevator floor. It will still use the same simulation, but the team will only consider a state once the elevator has reached each floor (there will be no states in between floors). This makes the state space discrete. Furthermore, state transitions will be changing floors or noop (no operation). The simulation will then calculate the time to the next floor, which the team will use as a reward. In order to better model the elevator's behavior the group will also consider using the last four states of the elevator as an input for the group's model as opposed to just one. This may improve its decision making by giving it temporal information as in [6].

Selection Explanation:
Team 28 choose a more complex Markov model with up/down actions first because it will give the algorithm more freedom to find a better solution to the problem. If a good solution is not found, then an alternative concept of a discrete state space will be used.

5.1.5 Using Adam optimization

Alternative Concepts:
Aside from optimizing with a technique like Adam the group could use RMSprop as an alternative. In addition to these gradient-based techniques, the team will also consider using a metaheuristic optimization approach, although this is not expected to be more efficient.

Selection Explanation:
Adam is used because it is commonly used in many reinforcement learning papers.

5.1.6 Using multilayer perceptron

Alternative Concepts:
After investigating the success of this initial algorithm, Team 28 will tune the hyperparameters of the model and consider other architecture modifications such as introducing a recurrent element into the neural network. The group will

also consider using spiking neural networks instead.

Selection Explanation:
The group will use a multilayer perceptron (MLP) because it is simple yet powerful. It can be used to approximate a lot of functions, and since the group does not have a great idea about the underlying statistical correlation structure of the Markov model transition data, then the group wants a potentially generalizable model with dense fully-connect hidden layers. MLP is also relatively simplistic and efficient so it is a great starting point.

5.1.7 Using PyTorch for implementation

Alternative Concepts:
As an alternative to PyTorch the group can use Tensorflow to run the computations for optimizing the parameters of the team's neural network model. Team 28 will also consider using Keras.

Selection Explanation:
PyTorch is a commonly used deep learning framework which is easy to use but allows for sufficient implementation detail complexity.

5.1.8 CPU Training

Alternative Concepts:
Since neural networks require processing a lot of data, team 28 will be also consider using a GPU to accelerate the team's training process.

Selection Explanation:
We don't anticipate our model to require the amount of computation a GPU will provide.

5.2 Selection Summary

For the major simulation design decision, the group chose to create a custom simulation instead of using an existing data set or measuring and collecting our own data manually. This is because we can customize the output of the simulation to match our use case of high floor counts. Additionally, we can customize the number of elevators in use. Overall, it allows the group to be less constrained.

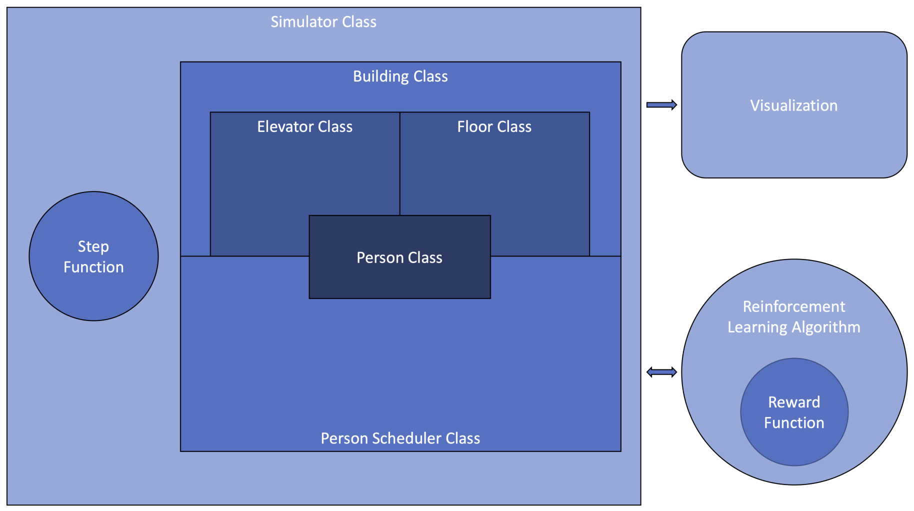
For the optimization algorithm, between the three proposed strategies the Q-Learning approach wins out. Table 13, containing the team's weighted decision matrix, highlights the different advantages of each concept and how they compare to each other. Ultimately Q-Learning, which is not necessarily cost or ease-of-implementation optimal, dominates in terms of optimality and speed. Thus team 28 have decided to explore it as the team's final approach.

6 Embodiment Design

6.1 Algorithm & Control Flow

The algorithm used in this project is split into two core functionalities as mentioned before: the simulation and the reinforcement learning algorithm.

Figure 2: Project Class Overview



6.1.1 Simulation

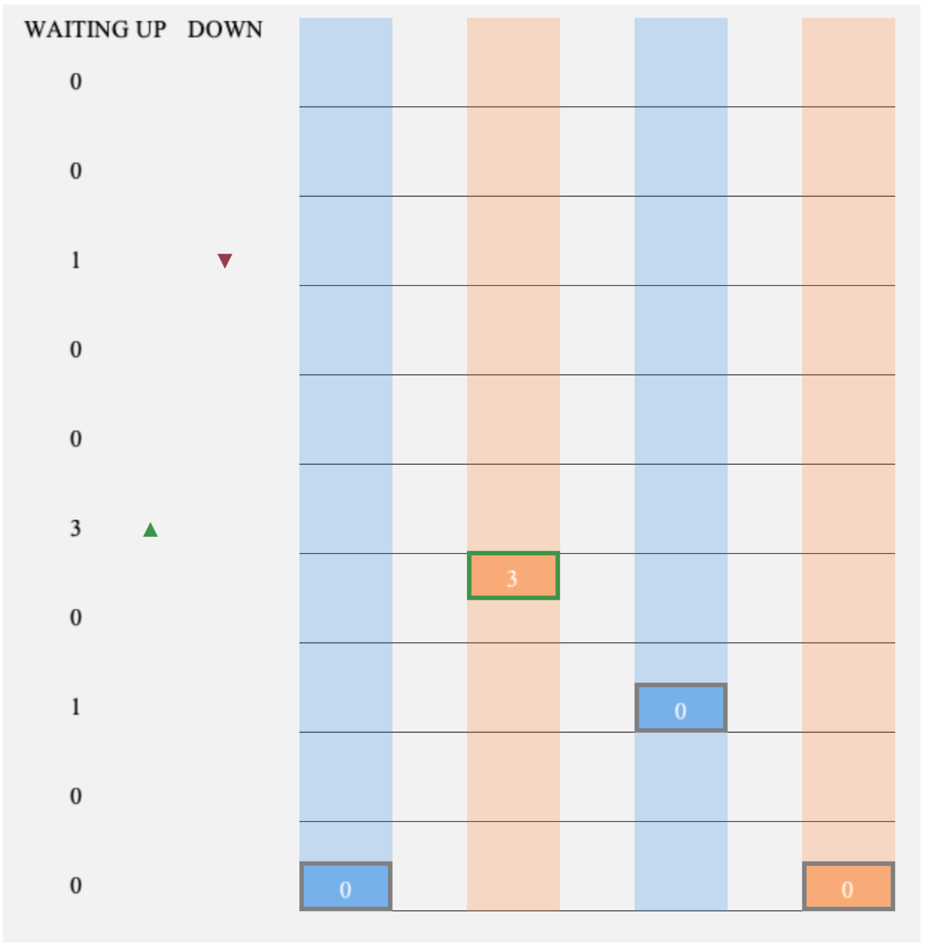
From a high level, the simulation is split into six fundamental classes, a step function, and the visualization.

The classes are as follows: Simulator, Building, Elevator, Floor, Person, and the Person Scheduler. The Simulator class runs the step function, serving as the most abstracted layer of the simulator. The Building class is one step deeper, handling the storage of elevators, floors, and measurement specifications of certain aspects of the building. The Elevator class contains variables for keeping track of passengers and velocity, as well as functions for physics and loading/unloading passengers. The Floor class stores the people waiting and queued floor button presses (up/down). The Person class stores the person's original floor, destination, waiting state, and waiting time.

The basic step function has several jobs. First, it checks if any elevators are going to finish their loading/unloading cycle in the time allotted. If so, it changes the allotted step time to exactly finish this cycle; this allows the reinforcement algorithm to determine the direction of travel. Next, it changes the position of each elevator to a new position based off the specified passage of time. With this it unloads people and loads people going in the previous direction. Finally, the step function increments states such as each person's waiting times.

The visualization simply gets fed the building as part of its initialization and updates the building/elevator/passenger states accordingly. This component uses a wrapper for OpenGL primitives to display the elevator's position, current state (the elevator outline), the elevator passengers, the number of people waiting on a floor, and the floor up/down button presses. Having a visualization is helpful to see that the physics are working properly and it is useful to compare the states of the reinforcement algorithm's performance as opposed to the SCAN algorithm's performance.

Figure 3: Example Of The Building Visualization



6.1.2 Reinforcement Learning

Interfacing with the Simulation

At the current stage of our project, we have been focusing on completing the simulation first since the RL algorithm depends on it. An algorithm for the RL algorithm, however, has been written and is implemented in Python and PyTorch. Once the simulation is complete, we will work on adapting this code directly to the simulation and performing three steps. These steps will combine the physics step function and reinforcement learning step function in different ways. Ultimately, the RL algorithm interfaces with the physics simulation as follows:

Algorithm 1: How the RL Algorithm Will Interface with the Simulation

```
Get the initial state.
while operating do
    Select an action to perform. Call the RL step function.
    RL Function:
            Calculate the next state by calling the physics step function
            some number of times.
            Calculate the reward for transition to the new state with the
            given action.
            Return the new state and reward.
    Save the state and reward in the replay buffer.
    if Training then
        Train the ML algorithm on samples from the replay buffer.
    end
end
```

Implementations

The RL algorithm-elevator interface will be implemented in three different ways which allow the reinforcement algorithm varying degrees of control. Due to safety reasons, each of these steps has varying pros and cons; this is because the elevator needs to ensure a maximum speed for safety (the elevator cannot directly control the force and slam the elevator full of people into the ground).

Implementations:

1. First, a simple problem with one elevator will be tackled as a proof of concept. Inside the RL step function, the algorithm will call the physics step function until the elevator reaches a given floor. Then, the algorithm will calculate a reward and return it and the new state. The RL algorithm will calculate an action which will be the next floor to go to.

- **Pros**: This will be very simple. The algorithm won't need to plan more than one RL step ahead. This will show that the elevator can learn on a simple problem.
- **Cons**: This algorithm can only be applied to one elevator. This is because elevators might not all reach their destination at the same time. Waiting for that to happen would be a huge waste of time.

2. In order to tackle multiple elevators, the physics and RL step functions will be merged together. Due to the safety constrains mentioned above, the elevator cannot change floors while traveling. In order to avoid this issue, a 2-element queue will be used. The elevator will constantly travel to the first element, and the RL algorithm can change the second element whenever it wants (even while traveling between floors). When an elevator arrives, the second element will become the first element.

    - **Pros**: By merging the physics and RL timesteps (only call the physics timestep once in the RL step function), the program will be able to have multiple elevators running in parallel. This is because the algorithm won't have to wait for all elevators to reach their destination before moving on (Implementation 1's method would necessitate doing so).
    - **Cons**: Since the RL algorithm can only change the second element (the next floor to travel to after the destination), the algorithm will receive a delayed reward for its actions. This delayed reward signal will make training slower and more difficult.

3. At Dr. Sadovnik's suggestion, we will also consider having an RL algorithm for each elevator. Each elevator will learn its own neural network and only take a new action when it reaches its destination like in Implementation 1.

    - **Pros**: The physics and RL timestep functions can be unmerged. The physics timestep will be called in a loop, and each elevator's unique RL step function will be called when that elevator reaches its destination. This would allow us to take a more direct approach like in Implementation 1 while using multiple elevators. This may also help scale up to many elevators, since each elevator will have its own specific neural network. (One neural network might take to long to learn to control several elevators by itself).
    - **Cons**: The program will not be able to control each elevator with one central neural network, which might cause communication issues between the elevators and reduce the efficiency improvement of the algorithm. However, the simplicity for each individual elevator might make this approach easier for learning a good algorithm.
      The implementation of this step is also much harder.

Reward Calculation

To calculate the rewards, we will first use negative people waiting as a reward, which is easy to implement and will incentivize the elevator to move people to their desired floors quickly. After this, we will also try changing the reward function to be the negative of the sum of the time each person has waited. This should further incentivize the algorithm to treat passengers fairly and not abandon someone for hours.

6.2 Best Practices

The algorithm has a maximum speed to ensure that the operations stay within safety parameters. This creates a harder problem for the reinforcement learning algorithm. However, due to neural networks black-box nature, it is necessary to exercise this level of caution. Specific implementation steps are discussed in the section above.

7 Test Plan

7.1 Testing the Simulation

7.1.1 Testing the Building Simulation

It is essential for the later parts of the project to have a robust elevator simulation in the models. Thus precise and complete tests are needed to ensure the robustness of the elevators.

To test this, a series of simulated buildings will be created to cover all of the possible elevator configurations and edge cases:

- Single elevator
- Multiple elevators
- Elevators covering different ranges of floors
- Building with floors that are not accessible by elevator

Each of these buildings will be fed a predefined list of passengers arriving at specific timings in order to test the simulation's capability to deliver passengers to all accessible floors in a building, and to deal with high load without failing.

This simulation test will run on a pass or fail basis, with the only condition being that after new passengers stop arriving, the simulation eventually reaches a state where there are no more waiting people. This pass should succeed for all buildings which are not intentionally designed to have inaccessible floors

7.1.2 Testing the Person Simulation

In order to truly acquire an adequate amount of accurate data and testing conditions for a machine learning algorithm to improve our results, an adequate simulation is required for the random coming and going of new passengers, accounting for differences in popularity of floors in a building and density of foot traffic depending on time of day. To do this, a simulator was created to generate passengers based on a probability density function (PDF) describing the likelihood of a button press for each floor, relative to the passage of time. In order to ascertain the accuracy of the person scheduling simulator, sufficient testing must be done.

Given a specific building and the set of PDFs for it, this test will run the person scheduling simulation, and monitor and analyze the queuing up of new passenger. After the simulation is run for a long enough duration, the actual average rates of button presses will be compared to the PDF, and an error value will be obtained. The test will pass if the error falls is an acceptable margin of ten percent.

7.2 Testing Reinforcement Algorithm

7.2.1 Testing Implementation

In order to test the capacity of the reinforcement algorithm to learn and make gradual improvements, the goal is simply to ascertain that the implementation is capable of reading the inputs generated by the simulation, and creating a meaningful output which can control the paths of the elevators.

To do this, a series of simulations will be run for different buildings in order to create a wide variety of input arrays, covering complex states for the building. The input arrays for these simulations will be fed to the reinforcement algorithm, and the output observed. The algorithm passes the test if every single input state results in a meaningful output.

7.2.2 Testing the Reward Function

The most essential element for good progression for the ML algorithm is the reward function, which processes an output generated by the algorithm and returns an evaluation score in order to gauge which outputs are better, and which are worse. Due to the nature of the optimization of waiting time, and the ambiguity in defining rules on how to prioritize the time of passengers in different circumstances, testing can only be done on situations where one output is clearly better than another.

For this test, several predetermined outputs will be passed into a matching simulation using the reward function being tested. The outputs will be chosen such that one output is always indisputably better than the other (e.g., one sends the elevator in the wrong direction, and the other sends the elevator to the correct floor). The score rewarded for each set of outputs will be compared, and the test passes if the better option(s) score higher than the worse option(s).

8 Deliverables

Upon completion of the project it is planned to have a working simulation for the elevators and reinforcement learning algorithms to find the optimal behavior of the elevator. Thus, Team 28 will use these as deliverables to work towards as various sections of the project are completed.

Since elevators are expensive, they will be unable to be purchased for this project. Additionally, the team's algorithms will be most useful for multi-elevator systems, which would require more elevators. The simulation will take several important parameters: number of elevators, speed of elevators, and weight capacity of elevators. Additionally, the amount of foot traffic will be parameter-based. This flexibility will allow for the simulation of nearly all real-world elevator systems will still retaining a reasonable budget. Additionally, Team 28 can measure elevator use within UTK buildings in order to replicate the real-world elevator systems here such as the Min Kao building or SERF elevator systems. Since an important aim of the team's project is to target very tall, multi-elevator systems, Team 28 will attempt to model elevator systems from well known towers elsewhere, such as the Cathedral of Learning in Pittsburgh. Finally, the group will test the team's algorithms on speculative elevator designs, which may be useful in future megastructures.

The next deliverable is the creation of a reward function, which will allow us to run the team's reinforcement learning algorithms. This is critical for the team's machine learning algorithms, since it tells the algorithm how it should respond to the simulation state. For example, the group can choose to prioritize how many passengers are outside an elevator door or to prioritize certain floors over others.

The other major deliverable in this project will be the modern deep reinforcement algorithms. This deliverable will use the prior deliverables of the simulation and reward functions in order to determine the optimal actions that an elevator should take at any given point in time. For example, algorithms such as deep Q-networks will be applied, which use neural networks for function approximation.

Since the team's project is mainly computational, these first deliverables will be in the form of several source code files written in the Python language. Finally, Team 28 will write and design a report on the results of the team's project. The report will highlight the team's methods and their effectiveness on various elevators and associated simulations. This deliverable is a critical step in communicating the results of the team's work to a wider audience.

9 Project Management

In order to successfully go about completing the deliverables, the team's group will break down the next semester into the following segments organized by the time and order of completion.

Table 14: Deadlines

Deadline	**Description**
Month 1: Create a simulation to handle the physics of elevator movement	The simulation should model people arriving, elevator travel time, and elevator weight. The state should be given to a reinforcement learning algorithm. Since this is the backbone of the team's system, the team's group will dedicate the first month to its creation.
Month 1.5: Find simulation parameters to model elevator systems within UT and other potential scenarios.	The group will model several test elevator systems from both real buildings and hypothetical future structures. It is important to have a solid collection of elevator systems to experiment on.
Month 2: Create a reward function	It is important to create a reward function which successfully captures the intent of the problem. The team's group need to balance rewards so that the elevator optimizes transporting people as efficiently as possible. The team's group will spend two weeks designing this function in preparation for the reinforcement learning algorithms.
Month 3: Implementation of modern deep reinforcement learning techniques to interface with the simulation and reward function	The team's group will spend 1 month implementing and modifying various algorithms from the literature. The team's group will also collect the results from these algorithms for the different elevator systems created in above deliverables.
Month 3.5: Generate a project report based on existing findings.	This will be the final project report for the class.

Since the group consists of five computer science majors, every member is capable of contributing similarly to the project. Carl Edwards has more experience in machine learning and is taking a graduate reinforcement learning class, so he will help steer the direction of the reinforcement learning algorithm deliverable production.

10 Budget

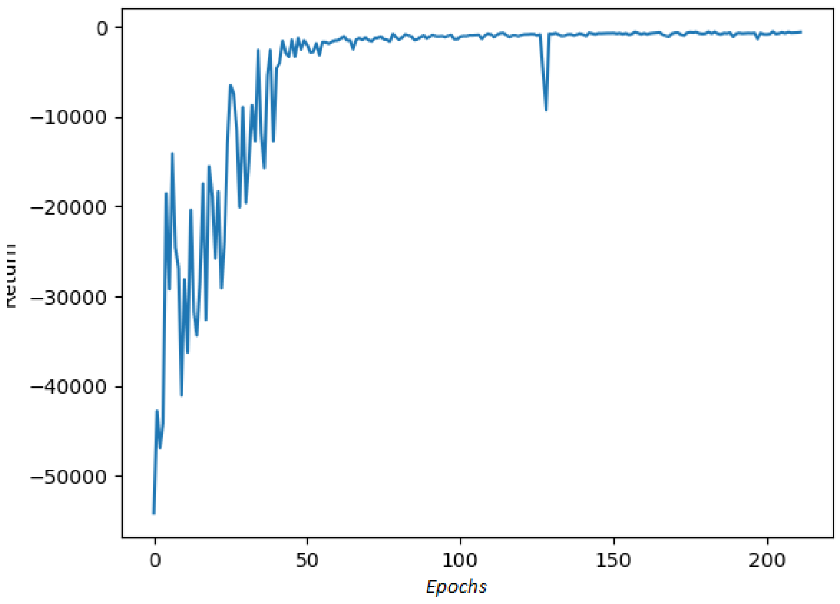
There is no designated budget for this project. The simulation and reinforcement learning are to be software only with only the use of open-source.

11 Summary of Progress

There are two major components to the project: the simulation and the reinforcement learning aspect. For the simulation, it is completely functioning. The simulation supports any number of elevators with any number of floors. Realistic physics are used with options to set acceleration/deceleration, max velocity, floor distance, etc. This is all displayed on an OpenGL visualization. Additionally on the visualization, one can see the floor button presses (up/down), the number of people waiting, the number of people on each elevator, and the current state of each elevator. Finally, the simulation uses a uniform poisson distribution for spawning people on floors with their desired destination floor.

As for the reinforcement learning component, we have achieved learning on a single elevator systems of 5 and 10 floors using negative people waiting as a reward and the desired floor as the action. Additionally, we also learn on 20 floors systems but solutions have a tendency to collapse. This is likely due to the way the algorithm interacts with the simulation being non-Markovian (there is a different amount of simulation time when we travel to floors farther away). We use [5] and [6] help determine hyperparameters. Initially, we tried to learn without episodes. This resulted in 5 floors working and sometimes 10. Unfortunately, 20 floors didn't work because people showed up faster than the initial random exploration policy could handle; this resulted in the Q-values essentially running away as the number of people increased to infinity. 20 floors worked when we reduced the rate people showed up to 20% of the previous value (from 0.5 to 0.1). The takeaway is that on taller elevator systems (20 floors), when the Poisson rate is the same as smaller buildings, the elevator has trouble keeping up at the beginning of learning. At lower Poisson rates it is also possible to learn better actions (and if Poisson rate is too high it won't learn at all).

Figure 4: This shows the return per epoch on a 10 floor system. The return approaches a value near zero, which indicates that few people are waiting during the episodes once it is trained.



To try to fix this, we switched to using fixed-length episodes. This produced unexpected results; the 20 floor elevator learned that less people showed up if it sits still (since only one second passes per when it doesn't move). Because of this, we switched to variable length episodes based on a fixed amount of simulation time. This seemed to help, but 20 still didn't work. We began to suspect that the elevator might not be physically fast enough to learn effectively or the RL step function isn't Markovian when interacting with the system. We tried several things: we tried a negative cumulative wait time reward function, we tried to add cumulative wait per floor as states, we added elevator position as a state, and we added the number of people on the elevator itself as a state too. We also tried to make the model a recurrent neural network so it would remember when people showed up. Unfortunately, this caused problems with variable batch sizes and the library we used (PyTorch). We also tried to use the change in people as a reward, but this doesn't work well because sometimes people show up and sometimes they don't (due to the Poisson process arrival scheme).

Finally, we realized that we could simplify the problem and make the amount of time that passes between RL steps more consistent by using only two actions (up and down) rather than 'the number of floors' actions. We found this to work better with the negative of the number of people waiting as the reward (we don't use the cumulative wait time per floor states either). 5 and 10 floors learn nicely. 20 floors begins to learn but it suffers from some type of solution collapse. It's interesting to note that the average action while learning was about 0.5. This is desirable because 1 is up and 0 is down, so an average of 0.5 indicates the elevator is moving up and down.

Figure 5: This shows the running average of actions taken at each step for an elevator system with 20 floors. This is the same experiment as Figure 19. Note the collapse in return happens when the action only goes up or down.

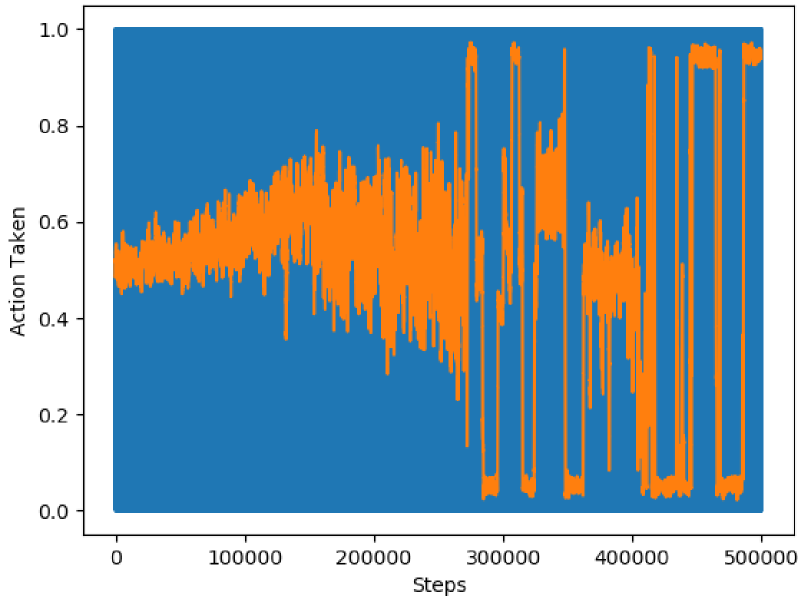
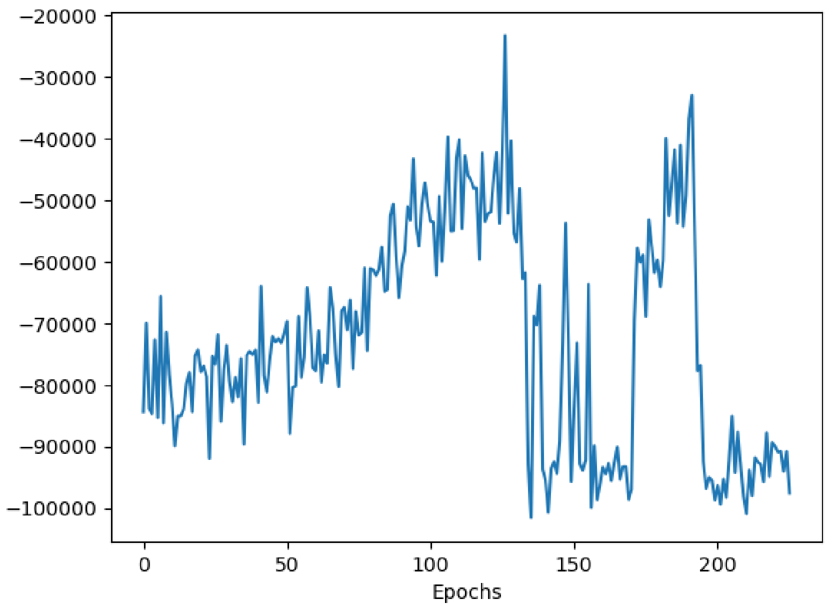


Figure 6: This shows the return for each epoch for an elevator system with 20 floors.

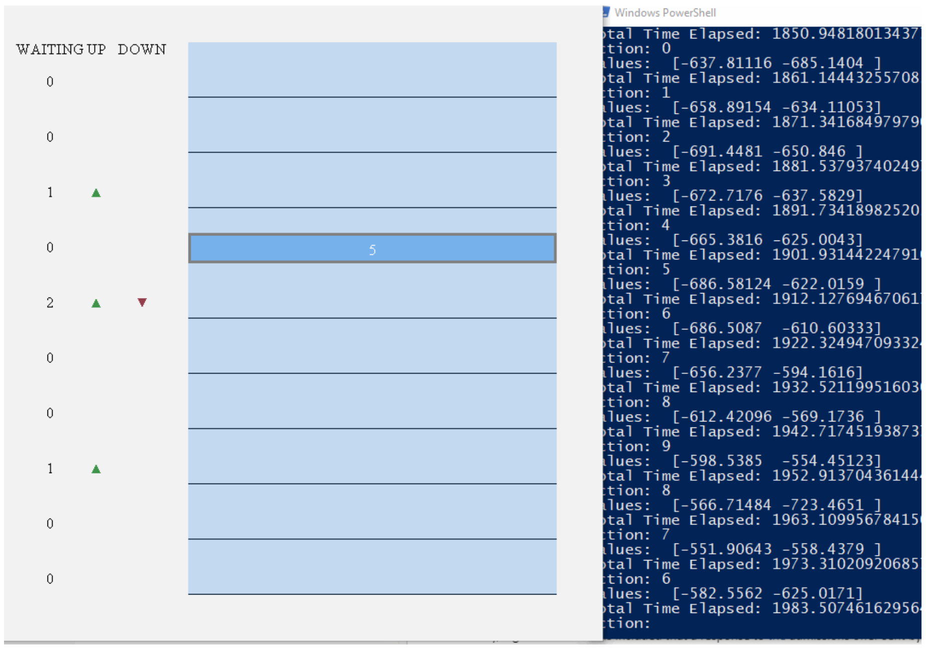


Whenever the solution collapsed the elevator started either only going up or only down. Long episodes seem to work better (20000 seconds or infinite) because people pile up which helps the elevator learn faster (bigger differences in values). However, for infinite episodes on large buildings people pile up before it can learn and it gets overwhelmed and doesn't learn (the Q function keeps changing to get bigger as more people show up. We balance this with 20000 second episodes so that the length advantage exists but we can apply what we learned by resetting.

The 20 floor solution collapse is theoretically impossible in a Markovian system in traditional reinforcement learning due to the policy improvement theorem [8], but due to the approximation function for Q and non-Markovian elements of the simulation it happened here in practice. Merging the RL step and simulation timestep function would probably contribute to fixing this. Also, although 5 and 10 floors worked, they would still abandon people on rare occasions. We think a hybrid algorithm between RL and classical algorithms might be the best bet for a consistent, semi-explainable algorithm. It might also be useful to remove the option to go up on the top floor and down on the bottom floor. Using a state-of-the-art model models might also help; we don't believe that policy gradient methods would help, however, though this is untested. Another future path for research is to fill the elevator up with 10-20 people and then stop letting them show up. After this, train the elevator on this until nobody is waiting. Repeat until fully trained.

The Covid-19 pandemic substantially delayed our project, but we managed to still achieve some results.

Figure 7: This is a visualization of the neural network making decisions for 10 floors. Check out the gif at: `https://raw.githubusercontent.com/EfficientElevator28/ReinforcementLearning/master/up_down_10.gif`



```
otal Time Elapsed: 1850.94818013437
tion: 0
alues:  [-637.81116 -685.1404 ]
otal Time Elapsed: 1861.14443255708
tion: 1
alues:  [-658.89154 -634.11053]
otal Time Elapsed: 1871.34168497979
tion: 2
alues:  [-691.4481 -650.846 ]
otal Time Elapsed: 1881.53793740249
tion: 3
alues:  [-672.7176 -637.5829]
otal Time Elapsed: 1891.73418982520
tion: 4
alues:  [-665.3816 -625.0043]
otal Time Elapsed: 1901.93144224791
tion: 5
alues:  [-686.58124 -622.0159 ]
otal Time Elapsed: 1912.12769467061
tion: 6
alues:  [-686.5087  -610.60333]
otal Time Elapsed: 1922.32494709332
tion: 7
alues:  [-656.2377 -594.1616]
otal Time Elapsed: 1932.52119951603
tion: 8
alues:  [-612.42096 -569.1736 ]
otal Time Elapsed: 1942.71745193873
tion: 9
alues:  [-598.5385  -554.45123]
otal Time Elapsed: 1952.91370436144
tion: 8
alues:  [-566.71484 -723.4651 ]
otal Time Elapsed: 1963.10995678415
tion: 7
alues:  [-551.90643 -558.4379 ]
otal Time Elapsed: 1973.31020920685
tion: 6
alues:  [-582.5562 -625.0171]
otal Time Elapsed: 1983.50746162956
tion:
```

12 References

[1] Gina Barney and Lutfi Al-Sharif. *Elevator traffic handbook: theory and practice*. Routledge, 2015.

[2] Robert H Crites and Andrew G Barto. Improving elevator performance using reinforcement learning. In *Advances in neural information processing systems*, pages 1017–1023, 1996.

[3] Robert H Crites and Andrew G Barto. Elevator group control using multiple reinforcement learning agents. *Machine learning*, 33(2-3):235–262, 1998.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

13 Appendix

Business model canvas is attached on the next page. Our Gannt chart is attached on the page after. The 3rd page is a test plan matrix. The 4th through 7th page is a preliminary patent application. The 8th page is the poster. The remaining pages are reinforcement learning results.

The Business Model Canvas

Designed for: Efficient Elevator Algorithm | *Designed by: Team 28* | *Date: 10/16/2019* | *Version: 2*

Key Partners

- Partners: elevator manufacturers for tall buildings (ex. Downtown hotels, offices)
- Suppliers: microcontroller suppliers (if additional hardware required)
- Resources (from partners): none* (run in simulations until actual implementation)
- Key activities (that partners perform): service/update existing elevators in certain buildings per our algorithm

Key Activities

- Key activities required by value propositions: creation or collecting data sets, algorithm and metric development, marketing to building owners and tenants (specifically companies), installation of additional hardware for customers
- Distribution channels: partnership(s) with building owners
- Customer relationships: periodic (yearly) check-ins with customers to see if that are satisfied, analysis of actual data to see if time savings have occurred (automated)
- Revenue streams: one-time installation fee and possible additional service fee

Key Resources

- For value proposition: existing example hardware used by the elevators that we will be working with
- For customer relationships: for later automated analysis of actual elevator data, potentially additional hardware and an internet connection will be required for data upload
- For distribution channels: website demonstrating effectiveness of algorithm
- For revenue streams: Subscription fee from clients for maintenance and updates

Value Propositions

- Value to customer: smaller* wait times for elevators leading to increased happiness (*exact measurement metric to what is a smaller wait time is not necessarily straightforward)
- Customer problems we solve: frustration from elevator users; the lack of elevators that were originally installed in buildings will be mitigated. These can lead to people/business more likely to rent
- Products/services to each customer segment: algorithm product to building owners
- Customer needs satisfied: better convenience/usability

Customer Relationships

- Elevator owners (personalized assistance)
  - Free remote debugging
  - Paid in-person debugging
- Elevator Users (active community that provides feedback if willing)
  - Will maintain online portal for submitting feedback on elevators (both positive and negative)
    - Link will be posted in each elevator

Channels

- Website with contact/purchasing information
  - Will include forum/portal for users to provide feedback on elevators (both positive and negative)
  - Will also include a separate portal for owners allowing them to:
    - Report issues/provide feedback
    - Download new updates (both functional and security)
    - Request in-person/remote assistance
- To increase awareness
  - Booths at appropriate conventions
  - Calls/emails to various businesses, apartment complexes, etc.
- To aid installation
  - Partnerships with appropriate electronics providers if necessary

Customer Segments

- Building owners
  - Ranging from corporations to landlords
    - Not very diverse
- Users
  - Anyone who uses the elevator
    - Ranging from workers to residents to visitors
    - Very diverse group

Cost Structure

- Key resources:
  - Website showing our product is a one-time implementation cost, plus small maintenance costs, plus a recurring hosting cost
  - The automated testing of our algorithm's effectiveness is a one-time implementation cost plus a recurring hosting cost
  - All other resources are hardware-based which is a fixed cost absorbed in the one-time fee that the customer pays per elevator
- Key activities not covered by resources:
  - Marketing campaign to potential customers

Revenue Streams

- One-time charge per elevator
  - Cost will vary significantly depending on the size/type of the elevator
  - Bulk pricing if multiple elevators are being fitted at once in the same building
  - Includes:
    - Fitting the algorithm to the elevator
    - Installing any additional hardware
    - Additional functional/security updates later on
- "Housecall" charges
  - Costs money if customers need in-person assistance
  - Potential subscription fee for maintenance and premium faster support

Efficient Elevator Project - Team 28

Task ID	TASK TITLE	START DATE	DUE DATE	January		February		March	
				1	15	1	15	1	15
1	Simulation	1/10/2020	2/15/2020	■	■	■	■	■	
1.1	Set Up Environment	1/10/2020	1/28/2020	🟦	🟦	🟦			
1.2	Research/Implement Elevator Physics	1/10/2020	2/3/2020	🟦	🟦	🟦	🟦		
1.3	Record Minkao Elevator Parameters	1/10/2020	2/3/2020	🟦	🟦	🟦	🟦		
1.4	Implement Elevator Visualization	2/1/2020	2/15/2020			🟦	🟦	🟦	
2	Reinforcement Algorithm	2/15/2020	3/15/2020			■	■	■	■
2.1	Determine Input State	2/15/2020	3/1/2020			🟦			
2.2	Determine Metrics	3/1/2020	3/15/2020			🟦			
2.3	Investigate Other Reinforcement Learning Algorithms to Use	3/1/2020	3/15/2020			🟦	🟦		
2.4	Implement Simple Heuristic	2/15/2020	3/1/2020				🟦	🟦	
2.5	Iterate on the Design Using Existing Tests	3/1/2020	3/15/2020					🟦	🟦
2.6	Test on Other Elevator Systems	3/1/2020	3/15/2020					🟦	🟦

Test	Target E.C.	Expected Results	Results	Remarks
Implementation Viability Test	Reinforcement Learning Implementation	Pass all test cases	Never ran	COVID-19
Optimal Output Comparison Test	Reward Function	Pass all test cases	Never ran	COVID-19
Correct Floor Accessibility Test	Building Simulation Implementation	Pass for all cases not intentionally designed to have	Never ran	COVID-19
PDF accuracy Test	Time-Dependent Foot Traffic Simulation Implementation	Consistently pass for cases with PDFs that generat	Never ran	COVID-19

Efficient Elevator Algorithm Patent

April 2020

TITLE: Efficient Elevator Scheduling via Reinforcement Learning
INVENTORS: Austin Day, Carl Edwards, Daniel Nichols, Sean Toll, Owen Barbour

FIELD OF THE INVENTION

The invention outlined in this patent pertains to a software improvement to augment the path choices of preexisting elevator systems. It achieves this by applying reinforcement machine learning to existing elevator algorithm concepts on an otherwise unchanged elevator.

BACKGROUND

Most elevators run a proprietary variation of the SCAN algorithm. The SCAN algorithm is a simple algorithm used in various applications, wherein the object in question moves in single-directional sweeps across its range of access points until it has satisfied all requests before it can change directions. For this specific case, once the elevator has started its motion up or down, it will continue in that direction until it has stopped at every floor that has had its button pressed in that direction. Once it has no more requests at any floors further down that trajectory, it has the freedom to go any direction to satisfy the next request for a floor.

This algorithm has been used since the introduction of some of the earliest mechanical elevators, and continues to be used for its effectiveness, with few changes outside of proprietary heuristic tuning to allow the algorithm to better suit the needs of more complex elevator setups and systems. The algorithm is nonetheless prone to certain shortcomings in the face of more complex scenarios, such as when presented with a choice, choosing a direction where passengers are nearer, or a direction where there is a higher probability of having many passengers, where choosing the wrong direction could waste time.

By implementing a well-tuned reinforcement learning element to the decision-making of an elevator system, the system is able to make more educated decisions based on a greater variety of factors, thus improving overall performance by decreasing the average time lost by people waiting on elevators.

BRIEF SUMMARY OF THE INVENTION

The presented invention comprises a novel algorithm for scheduling elevators in various structures. The algorithm is powered by reinforcement learning, specifically deep-Q learning, and a software simulation of the building to find optimal parameters. The invention is designed to work with both simple and complex structures with varying numbers of elevator shafts and elevators per shaft. While optimal scheduling is unknown, the simulation gives the reinforcement learning model access to all of its parameters and, with some training time, it can find a suitably optimal elevator control.

BRIEF DESCRIPTION OF THE DRAWINGS

In this document brief physical displays are given as an example to describe the proposed invention and its workflow, however, the invention is not limited to the likeness of the images.

Figure 1 shows the prescribed organization and workflow pertaining to this invention. The simulator, complex in nature, models each component of what might take place in a building. These parameters and state information are passed to a reinforcement learning algorithm which gives feedback to the simulation in order to control it. This cycle continues, while progress can be monitored through the output visualization program, which functions to evaluate the schedule from human eyes without finalizing the schedule in an actual building yet.

DETAILED DESCRIPTION OF THE INVENTION

The invention, and its novelty, lie in both the deep-Q learning algorithm and its intimate connection with the realistic simulation. The simulation serves the purpose of finding optimal parameters without having to train a learning algorithm, initialized with random weights, on an actual building.

The simulation is designed to model the building realistically, but give full control and modularity in its design. Most physically based parameters are accessible, such as elevator acceleration, number of floors, elevators, and shafts. Additionally, other abstract parameters, such as the rate at people pressing buttons, are modeled by customizable probability distributions, giving control to anything running the simulation. Paired intimately with the reinforcement learning algorithm the simulator provides a cheap and efficient way to learn optimal schedules for a building.

Powered by a reward function the deep-Q learning trains a deep learning model, which approximates the estimated return of each action from some state. The states are retrieved from the paired simulation (or the actual building in a deployed model). The proposed invention uses a 2 layer Multi-Layer Perceptron network with loss calculated via mean squared error.

The proposed loss function is written as

$$L_i\left(\theta_i\right) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} \mathcal{Q}(s', a'; \theta_i^-) - \mathcal{Q}(s, a; \theta_i)\right)^2\right]$$

In this equation, s is the current state, a is the action taken,, r is the reward for taking these actions, s' is the next state, and a' is the action taken from the next state. γ is the discounting factor. The Adam optimizer will be used to minimize this loss function. $\mathcal{Q}$ is the action-value function that will be learned (the Multi-Layer Perceptron). $U(D)$ is a uniform distribution of transitions in the replay buffer D. Θ are the model's parameters.

To effectively measure loss the model needs to take into an account an appropriate object. This is done via the reward function, which is modular and customizable in this invention. It is allowed to optimize waiting time, travel time, and any other desired objective (assuming appropriate mathematical properties). Given this modularity the invention is also claimed to be used for finding poor, or sub-optimal schedules. Since the objective can be set to longest average wait, for instance.

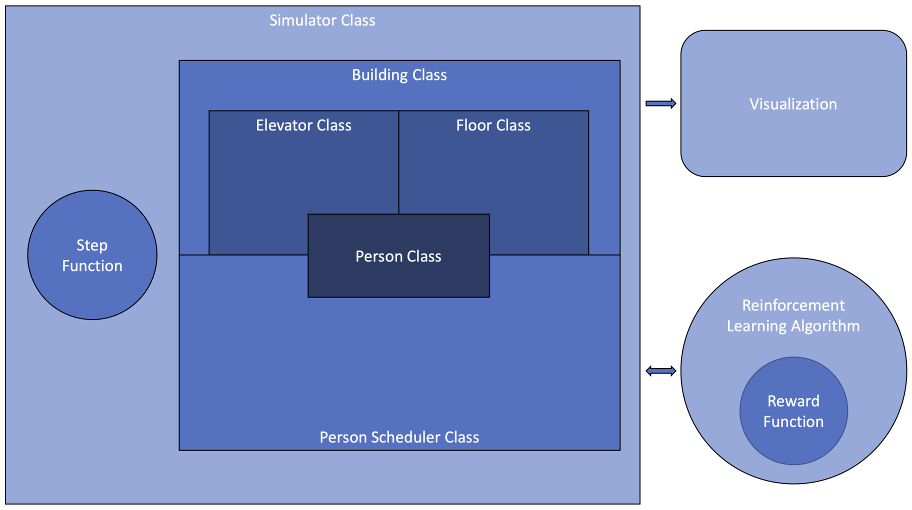
While this loss is standard in deep-Q learning, its use in elevator scheduling paired with a simulation is unique to this invention.

In implementation, this invention uses a proprietary code base developed by the authors in the open source, freely available Python language. The learning algorithm uses open source deep learning frameworks, which are freely available under the Modified BSD license.

ABSTRACT

The presented invention describes a novel approach towards combining advanced reinforcement learning techniques with fully customizable, modular simulations in order to discover optimal elevator scheduling for a desired building configuration. After configuring the simulation with the appropriate building setup, elevator parameters, and people frequencies the simulation, provided the desired optimization objective, will find an optimal schedule, which will be better than current SCAN style approaches to scheduling. With the invention is also included a custom visualization software to view elevator behavior before actually integrating with a real system. Once an optimal network is found, the product could be deployed to control a system in a building given main, central control system.

Figure 1: Workflow Overview



Efficient Elevator Algorithm

Owen Barbour, Austin Day, Carl Edwards, Daniel Nichols, Sean Toll

Abstract

Our goal was to improve elevator efficiency by running a custom reinforcement learning algorithm on an elevator simulation of our own creation. We had success learning elevator systems with a lower number of floors and a lower Poisson rate.

Overview

Elevators are not perfect, but they can be improved. There is currently a gap in the literature surrounding elevator optimization using modern reinforcement learning techniques. We attempted to fill in this gap. Our end goal was to create an elevator algorithm that could potentially handle future super-structures.

Solution

Our approach to this problem consisted of an elevator simulation and a reinforcement learning algorithm to run on the simulation.

Simulation:

- Coded in Python
- Realistic Physics
- Visualization Built-in (Figure 1)

Reinforcement Learning:

- Coded in Python
- Handles 5, 10, and 20 floors
- Reward Function: negative number of people

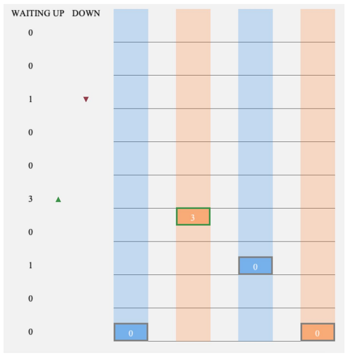


Figure 1 - Elevator Simulation

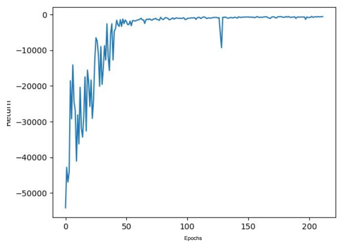


Figure 2 - Return vs Epochs Graph of the Reinforcement Learning for 10 floors

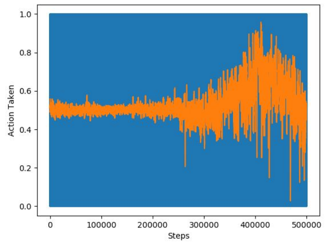


Figure 3 - Running Average of Actions vs Steps Taken of the Reinforcement Learning for 10 floors

Results

- Algorithm ran successfully with one elevator on 5 and 10 floors (Figures 2 and 3 show 10 floors)
- With 20 floors, the algorithm collapsed during training due to the non-Markovian nature of the simulation

Conclusion/Future Work

We successfully simulated an elevator. With limited success, we optimized the simulation using a reinforcement learning algorithm.

Regarding future development on this project idea, we could:

- Make simulation more Markovian
- Make hybrid reinforcement/classical algorithm to guarantee consistency

References

[1] Gina Barney and Lutfi Al-Sharif. *Elevator traffic handbook: theory and practice.* Routledge, 2015.
[2] Robert H Crites and Andrew G Barto. Improving elevator performance using reinforcement learning. In *Advances in neural information processing systems*, pages 1017–1023, 1996.
[3] Robert H Crites and Andrew G Barto. Elevator group control using multiple reinforcement learning agents. *Machine learning*, 33(2-3):235–262, 1998.
[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Figure 8: This shows the loss for each step for an elevator system with 5 floors. Note that all these figures show running average values as well.

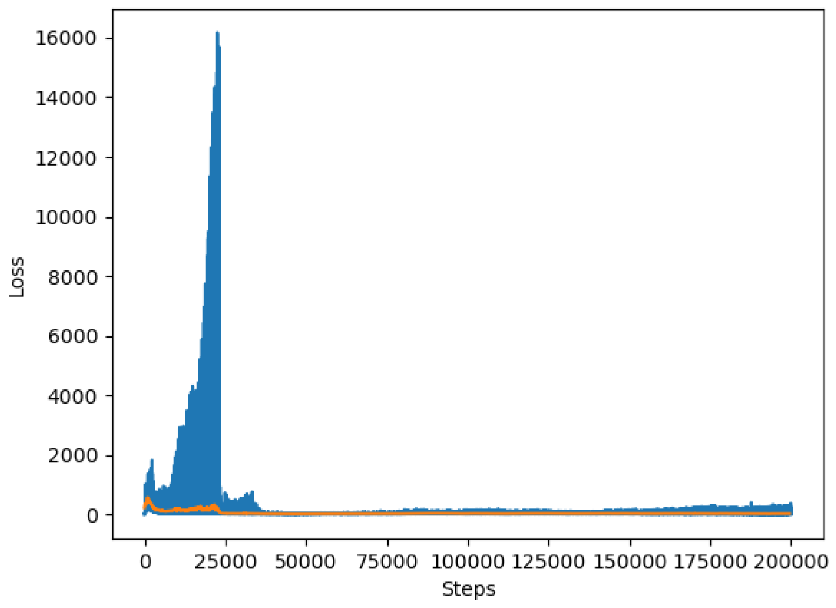


Figure 9: This shows the loss for each epoch for an elevator system with 5 floors.

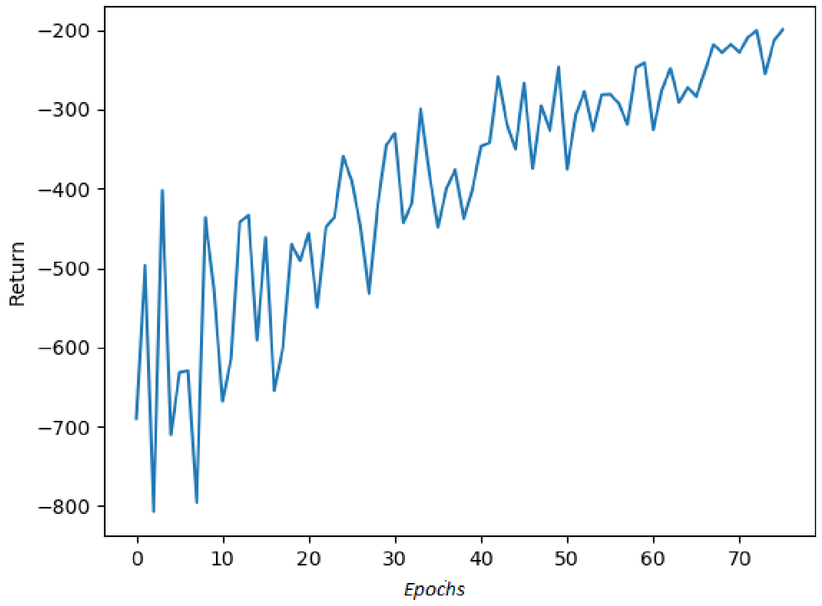


Figure 10: This shows the reward for each step for an elevator system with 5 floors.

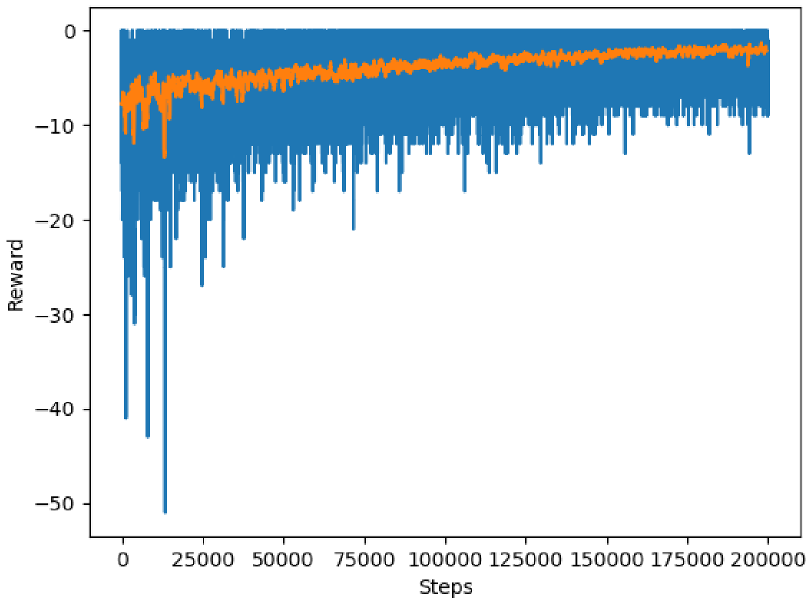


Figure 11: This shows the action for each step for an elevator system with 5 floors.

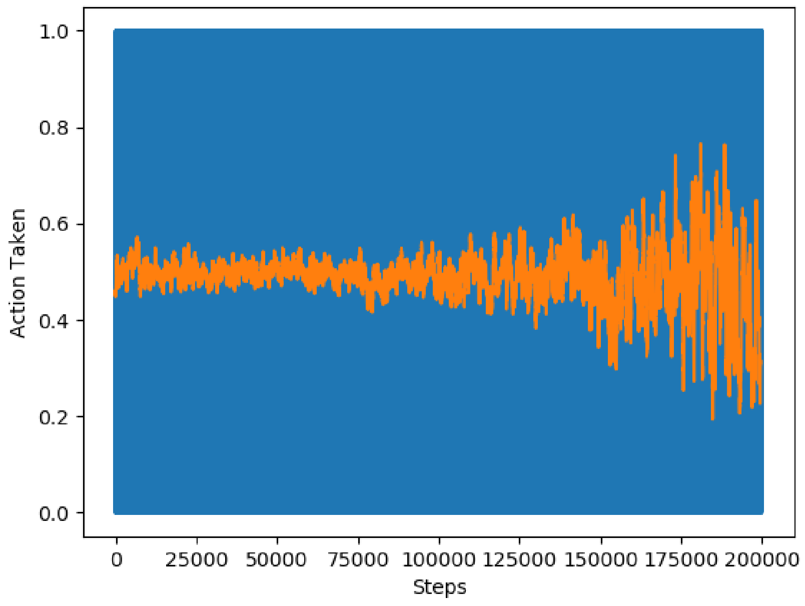


Figure 12: This shows the loss for each step for an elevator system with 10 floors.

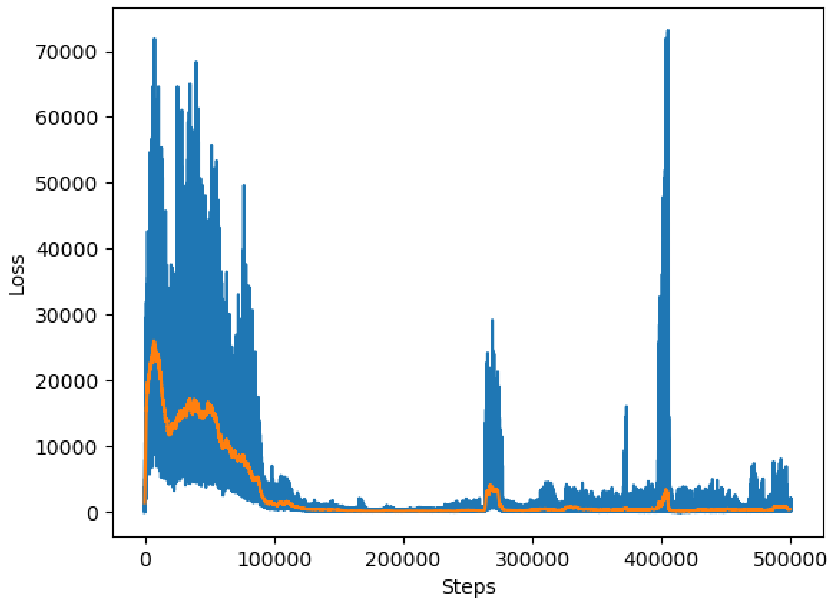


Figure 13: This shows the loss for each epoch for an elevator system with 10 floors.

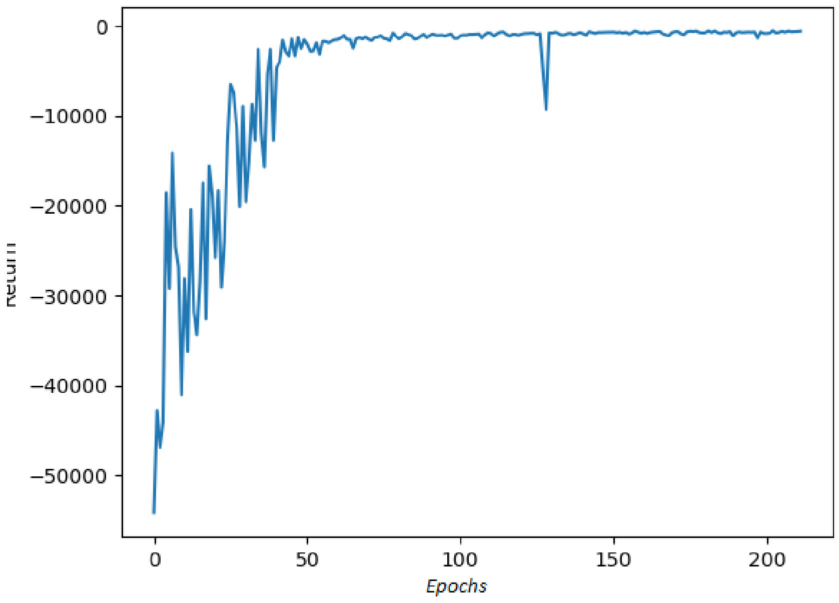


Figure 14: This shows the reward for each step for an elevator system with 10 floors.

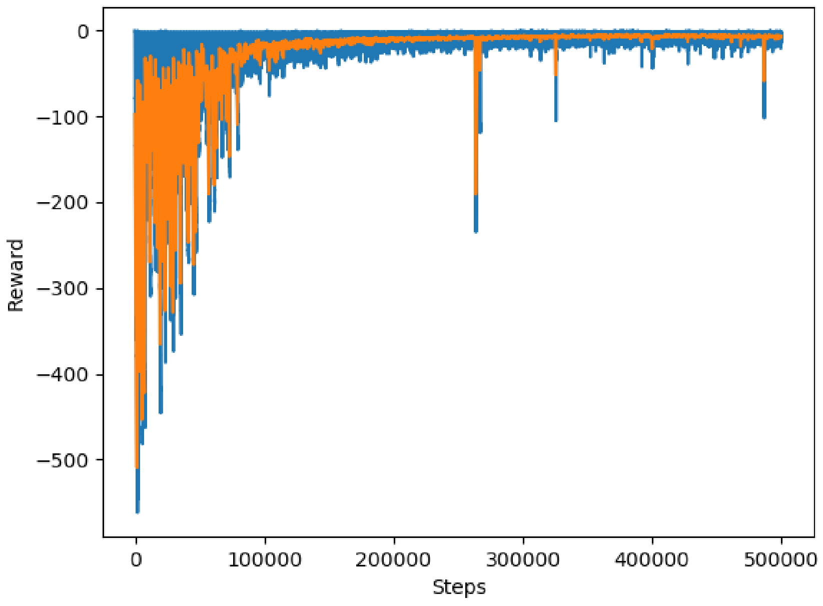


Figure 15: This shows the action for each step for an elevator system with 10 floors.

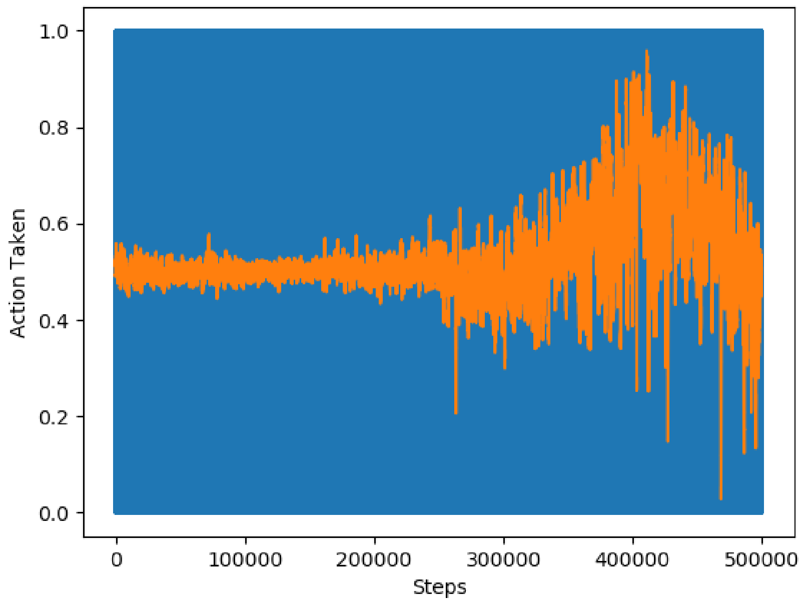


Figure 16: This shows the loss for each step for an elevator system with 20 floors.

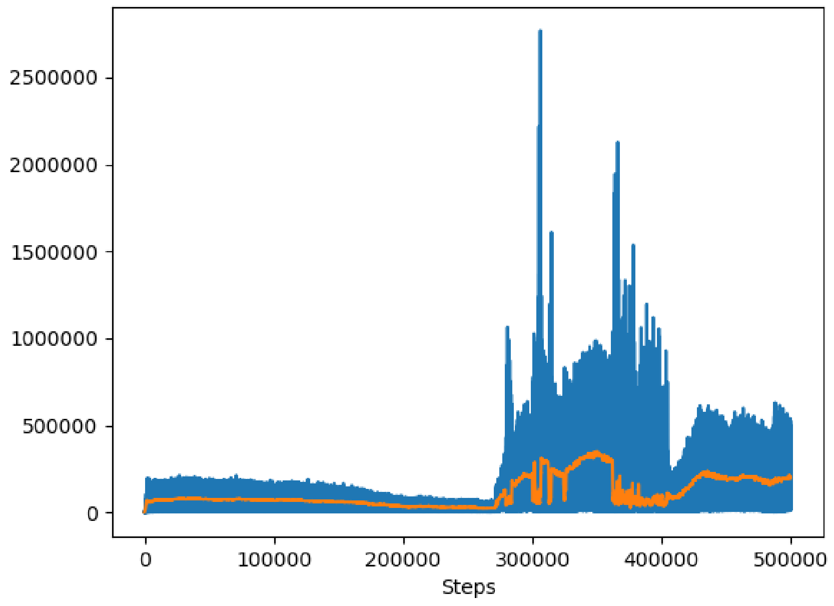


Figure 17: This shows the loss for each epoch for an elevator system with 20 floors.

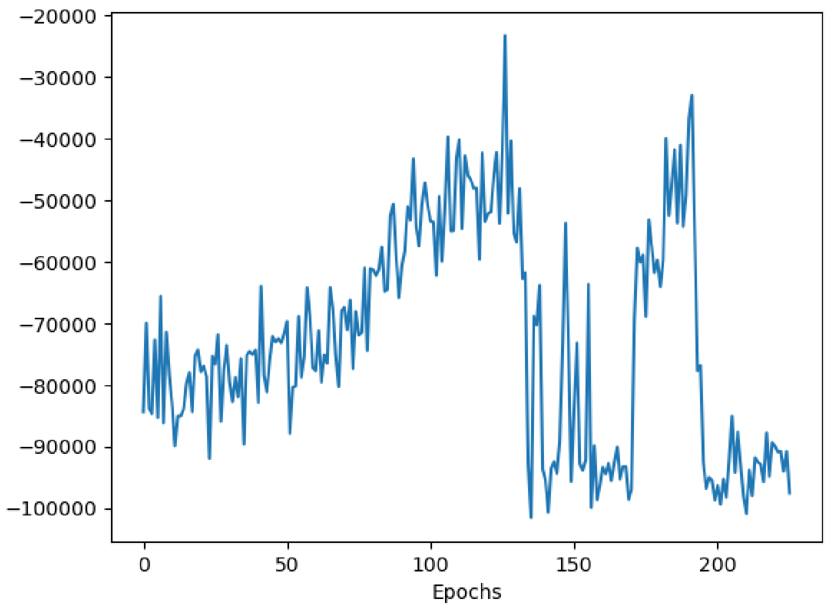


Figure 18: This shows the reward for each step for an elevator system with 20 floors.

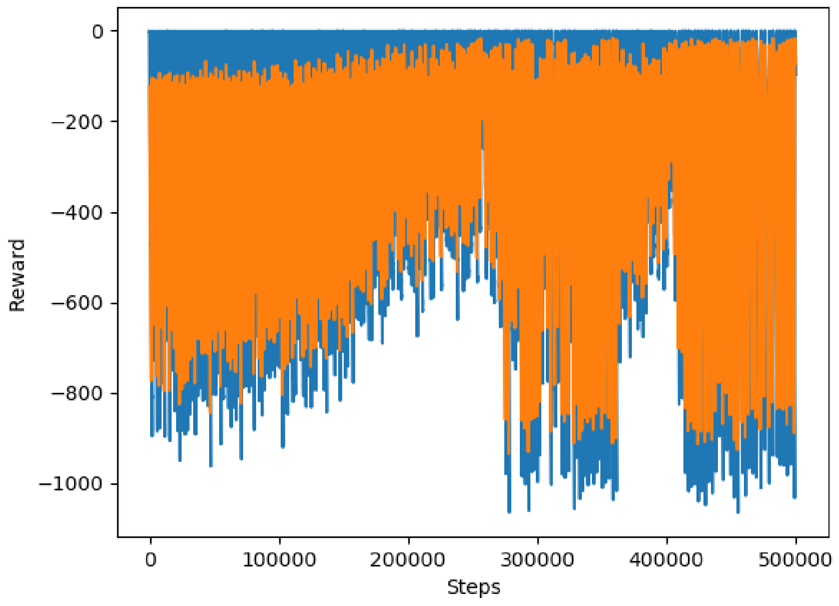
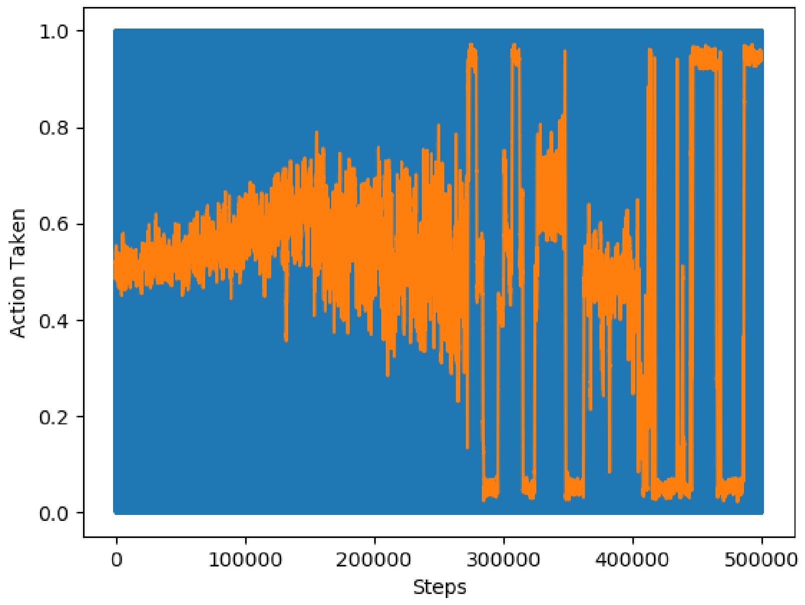


Figure 19: This shows the action for each step for an elevator system with 20 floors.


Action Taken
Steps