Logo, company name

Description automatically generated

**Michigan State University**

**Team Union Pacific**

Mobile Train Handling Simulator

Project Plan

Fall 2022

**Union Pacific Contacts**

Jeff Girbach

Adam Halley

**Team Union Pacific**

Adam Kasumovic

Darien Ford

Kevin Shin

Nhat Vong

Nicholas Shari

Reid Harry

**Contents**

[Executive Summary 3](#_Toc114672790)

[Functional Specifications 4](#_Toc114672791)

[Design Specifications 5](#_Toc114672792)

[Overview 5](#_Toc114672793)

[Game Designs 5](#_Toc114672794)

[User Interface (UI) 5](#_Toc114672795)

[Technical Specifications 10](#_Toc114672796)

[Overview 10](#_Toc114672797)

[System Architecture 10](#_Toc114672798)

[System Components 11](#_Toc114672799)

[Hardware 11](#_Toc114672800)

[Development Environment / Programming Language 11](#_Toc114672801)

[APIs 12](#_Toc114672802)

[Risk Analysis 13](#_Toc114672803)

[Testing Plan 14](#_Toc114672804)

[Schedule 15](#_Toc114672805)

Executive Summary

Founded in 1862 as part of the transcontinental railroad plan approved by Abraham Lincoln, the Union Pacific Railroad company, also known as Union Pacific, is one of the United States’ largest railroad companies. Union Pacific oversees 8,300 locomotives operating on over 32,200 miles of track, with most of the coverage located in the western United States, which includes both freight hauling and passenger train services. Union Pacific has almost 30,000 employees, many of which are locomotive engineers or train drivers.

Locomotive engineers have a heavy responsibility, overseeing anywhere between 3,000 to 18,000 tons of cargo, and possibly more. This requires maintaining a careful balance between the throttle and braking of the train, based on terrain conditions like elevation grade and weather, as well as the curvature of the railroad. Training locomotive engineers is crucial to all aspects of railroad travel, as failure could result in hundreds of thousands of dollars in company losses, or in the worst case, the loss of life. The crew for a given train on any day is assigned on-demand, so training must happen quickly. Previously, most of the training was done on the job, but with the advent of computer simulation systems, training can now be done safely behind a desk, allowing trainees to test their knowledge and gain experience without the use of an actual train. To that end, PS Technology, a subsidiary of Union Pacific, develops software-based railroad solutions, including realistic, physics-based simulators used to train locomotive engineers.

Together with PS Technology, Team Union Pacific is working to create a mobile train handling simulator, allowing crews to take their training with them. Currently, they must use proprietary and expensive hardware at a specific location, i.e., a train-like console. Along with those in the railroad industry, this simulator also gives the general public a fun and engaging way to learn about what goes into operating a modern train. Using PS Technology's proprietary train physics API, we seek to create a modified version of their existing simulator, allowing mobile users to simulate controlling the throttle and braking of a train with varying loads. This mobile simulator will be a useful, low-cost asset for trainees to test their knowledge and gain experience on the go.

Functional Specifications

Trains have been getting longer and heavier, so railroad companies must utilize distributed engines throughout their trains in order to push/pull cargo to their destination, also known as “distributed power”​. Distributed power requires specialized training to control, but full 3D simulators are costly and time-consuming to use (like their existing simulator)​. Our project simplifies these simulations into a mobile 2D game that still utilizes their advanced physics API while being easy to pick up and play anytime, anywhere​.

The simulator contains a selection of levels with varying terrains and trains. Trains may have up to several hundred cars and three locomotive groups (locomotive groups are adjacent locomotives that are controlled as one unit, so we will abstract them into one locomotive car in the simulator and refer to them as just “locomotive cars” or “locomotives” for the remainder of this document). The train cars are differentiated from locomotive cars by having different visuals than the locomotives.

When in the simulation, users can see information pertaining to the train, particularly the velocity of the train and weight of the cars (on demand), and most importantly the forces always acting on each of the cars (using color codes). In addition, they can see much more terrain ahead of the train than behind the train, so they have ample time to let any throttle changes take effect. There are on-screen throttle controls to control the speed of each of the locomotives that have 17 settings, with speeds going from -8 for braking (dynamic braking), to 0 for an idle position, to 8 for full acceleration. The number of throttles corresponds to the number of locomotives that are on the train. There is also a braking system that applies a percentage of braking from 0 to 100, with 0 being no braking and 100 being full brakes applied. This system is based on the real life “dynamic braking” feature of trains. To simplify controls, dynamic braking is implemented as negative throttle notches in increments of 12.5. An additional feature included is fencing, which allows for multiple locomotive groups to be controlled simultaneously. Also on the screen are pause and settings buttons. The pause button brings the user to a menu and lets them freeze the game at a given time, allowing them to evaluate their situation or analyze the information given to them so that they can take the most optimal next step depending on the situation. Also in the pause menu are options to adjust sound settings, restart the level, or quit to the main menu.

The main goal of the simulation is to have users constantly adjust the throttles on the train in order to minimize the forces acting upon the cars while considering factors such as the weight of the cars and the upcoming terrain. The most important forces include the buff and draft forces, where the buff is the force pushing the cars together and in on themselves (leading to buckling), and the draft is the forces pulling the cars apart (leading to stringlining). If any of these forces exceeds a threshold, the user will fail the simulation immediately, as this could possibly mean a catastrophe in the real world. This careful balancing act is like the responsibilities of an actual locomotive engineer, rewarding cautious users with success in their driving. It also allows experimenting users to test the limits of what they can do with the trains. As such, the simulator creates a mobile solution for in-training locomotive engineers to gain a simplified experience of what they must consider, as well as an engaging exposure to those who are interested in the profession. The simulator will be available on Windows devices.

Design Specifications

Overview

The Mobile Train Handling Simulator is designed to train locomotive engineers on handling distributed power trains by accurately simulating and displaying the buff and draft forces that happen on each individual car within the train. The train cars will change color depending on its force, blue representing buff force and red representing draft force. The intensity of the color reflects how much force the train car is experiencing. For example, a bright blue train car indicates a high buff force on the train car. Therefore, users will constantly see the train, represented as a colored series of continuous slim rectangles (each rectangle is a train car; they can be quite slim to fit up to 200 on a small screen), along with the terrain well ahead. The player can track their progress using the mileposts on the terrain or using the miles travelled UI. There is a weight graph located at the bottom-left of the screen to show the users the weight distribution of the train car. Since the train is always moving forward, they will be able to interact with the throttle controls in a timely manner so they can promptly minimize the potential harmful forces acting on the train cars and stay below the speed limit. Additionally, users can adjust the fencing to simultaneously control multiple locomotive groups at once. To successfully complete a level, users will manage the train to travel across 10 miles to reach to the next train station. A victory screen will appear, prompting to go to the next level or main menu and the user’s level progress is saved.

Game Designs

User Interface (UI)

Main Menu

A picture containing text, electronics, display, computer

Description automatically generated

Figure 1: Main Menu Screen

The main menu (Figure 1) is designed to be familiar and intuitive to all users. All menu buttons are large and spelled out. A distinct lack of symbols was used to get the user directly into the simulation.

Level Select

A screenshot of a computer

Description automatically generated with medium confidence

Figure 2: Level Select Screen

Once the user has pressed the start button, they are redirected to the level select screen (Figure 2). Here they will choose their premade level with its specified train and track combination. Details about the train such as number of locomotives and cars is displayed. An image of the level is displayed to give the user an idea of the challenges they would interact with during the level.

Game Scene

Graphical user interface

Description automatically generated

Figure 3: Game Scene Screen with Dialogue Box

The simulation (Figure 3) will open with a text box quickly explaining the level to the user. The level will feature solid ground to represent the track going left to right. The train will not be separated into viewable cars and engines. Instead, it will be one continuous line that will vary in color according to the force and stresses acting upon the cars. However, a weight distribution is implemented in the bottom left of the screen to indicate important areas of the train. Square mileposts are placed underground along the track to help the user judge the amount of distance they are covering.

Additional useful information will be displayed at the top of the screen for the user to use as a reference. Across the top of the screen is a color code for forces acting on the train. Blue being buff forces and red being draft forces. The top left-hand side of the screen will display time of day and distance traversed. The top right-hand side of the screen displays the train’s current speed and the track’s gradient. Next to those displays is a railroad speed limit sign which updates depending on the track’s location.

Graphical user interface, website

Description automatically generated

Figure 4: Game Scene Screen with Throttle Controls

.

Throttle control (Figure 4) will be performed using the up and down arrow buttons. These buttons are placed on the bottom of the screen to allow the user to easily control the train without blocking their vision of the screen while holding a mobile device. The buttons permit the user to cycle from an idle speed of zero up to a maximum speed of eight or a minimum of negative eight for dynamic braking.

Victory and Failure Transition Screen

A picture containing graphical user interface

Description automatically generated

Figure 5: Victory Transition Screen

After a user has successfully completed the level a victory screen (Figure 5) will pop up over the entire screen. This screen allows the user to continue playing onto the next level. Or offers them the choice to return to the Main Menu.

A screenshot of a video game

Description automatically generated

Figure 6: Failure Transition Screen

If the user fails the level for any reason a failure pop up (Figure 6) will take over the entire screen. This enables the user to retry the level by clicking on restarting. Or offers them the choice to return to the Main Menu.

Technical Specifications

Overview

The project revolves around the use of Unity, a free game development environment (Unity Editor) and game engine. Unity Editor allows developers to combine visual and auditory assets with C# scripts and the .NET framework into a project that can be built for virtually any platform. In addition, PS Technology (PST), a Union Pacific subsidiary, provides a Physics API in the form of compiled C# Dynamically Linked Libraries (DLLs) implemented as plugins into Unity Editor for use in simulating realistic train physics within the simulation. The target platform for end users is Windows. Additional software requirements include Visual Studio Community 2019 for its unique integration with Unity Editor thanks to Unity’s vast API.

System Architecture

Diagram

Description automatically generated

Figure 7: System Architecture Diagram.

At the heart of the system architecture is the Unity game engine. Unity will synthesize all the assets into a playable game as a .exe file for Windows machines. Visual Studio is what the team is using as their primary code editor because of the easily integrated debugging plugin for Unity. Unity’s codebase is built using the Microsoft .NET 4.X framework and can run scripts written in C# to provide the simulation with functionality. Finally, PS Technology is allowing the team to use their proprietary Physics API to simulate the forces always acting upon the train. All these technologies combined will keep the project performant, maintainable, and modular.

System Components

Hardware

Surface Pro

A tablet that runs Windows, making it very suitable for running and testing our builds by allowing us to see our frontends and UI on a large, portable device.

Development Environment / Programming Language

Unity

Unity acts as a catalyst that bridges the gap between C# scripting, art and sound assets, and working applications. Acting as a development environment that enables us to efficiently create the game with built-in development, testing, and iteration features, Unity is at the core of our system architecture.

Windows OS

While Unity can run on any of Windows, Mac, or Linux, Windows (preferably 10 or higher) is necessary for developing this project due to the use of DLLs, which are Microsoft’s proprietary dynamic library files that only reliably work on Windows machines.

Visual Studio Community 2019

The free version of Visual Studio 2019, Visual Studio Community 2019 is the IDE recommended by Unity for development as it integrates well with Unity’s API, providing unique and specific code linting for Unity along with many debug features that can run while playtesting the game.

C# and .NET 4.6

All scripting must be done in C# for both compatibility with Unity and the provided Physics API from PST. The .NET framework for the project is fixed at 4.6 or higher within Unity, as that is the necessary version for compatibility with the Physics API.

APIs

Unity API

Unity provides a MonoBehaviour class from which all scripts can inherit to gain access to the Unity API. The Unity API includes functions like Start() and Update(), which are called when the game starts and every frame, respectively. It also includes variables like Time.deltaTime, or the time elapsed between frames to ensure that the game runs the same regardless of the framerate. The Unity API makes game development much easier by providing the essentials that would need to be created otherwise.

PS Technology’s Physics API

Provided as .NET 4.6 C# project, PST’s Physics API contains source code, builds, and documentation for realistically and accurately simulating the physics of a consist, or the physical makeup of a given train, with distributed power. There are provided consist files (.cst files in CSV format) that go on provided track files (.trk files in XML format). Combinations of consist and track files are used to create levels, while the provided classes and functions in the API are used to simulate the resulting physics to be visualized in Unity. The result of building the C# project is a console application that runs solely on Windows, so compatibility of the unaltered project with other platforms is dubious.

Risk Analysis

Adjust to Screen Size for Different Devices

* **Difficulty:** Easy
* **Description:** The simulator will be run on devices of different sizes and resolution and it must be able to fit and look good on all of them.
* **Mitigation:** Utilize Unity’s built-in UI scaling and anchoring, and ensure that our game looks good on the most common aspect ratios by constantly testing with Unity Editor’s Game Window.

Fit Enough Train Cars on the Screen

* **Difficulty:** Medium
* **Description:** Since trains are realistically many cars long, the simulator would need to fit up to around 140 cars on the screen and make it so that the player can view and manage the cars.
* **Mitigation:** Represent the cars as very slim colored rectangles forming a contiguous snake-like rectangle. Usually, adjacent cars have similar forces acting on them, so groups of cars will still clearly show up as a gradient, indicative color. This way, all cars will fit on the screen.

Create the Tracks and Trains for the Levels

* **Difficulty:** Hard
* **Description:** All the physics is simulated internally (i.e., the physics API will simulate the hills and troughs, etc. even if we make it look like the train is moving along a flat surface) by using the provided and track (terrain) and train (consist) files in XML and CSV format, respectively. We need to create the levels so that they accurately depict what is happening internally with clear visuals.
* **Mitigation:** Develop each track stage automatically by parsing the track files for the elevations and mileposts and using the provided “track visualizer” tool to check if we are close and develop another parsing tool to convert a train CSV file into a Unity prefab (a reusable game object we can place in the game), likely with Python. This way, we can simply spawn in the correct train prefab on each of the track stages to account for all possible train-track combinations.

Scaling and Time

* **Difficulty:** Hard
* **Description:** Realistically, trains are quite slow to start up and their routes can last many hours. In addition, trains generally cannot handle inclines or declines of more than 5% (5% is a record--1.5% already presents challenges); however, such inclines or declines when drawn on a standard computer screen are barely noticeable by a user.
* **Mitigation:** On the horizontal axis, we will maintain the ratio between the train and track by implementing them at their “actual size” (1 Unity meter = 1 real-life meter) and scaling them down both by the same factor to fit on the screen. This will ensure that the train and track line up and are proportionate to each other. On the vertical axis, we will scale both the train and track up by a constant factor. Trains are much longer than they are tall, so vertical scaling will make them more visible on the screen than if they were actual size. Vertical scaling will also make slight changes in inclination, such as a 1% incline much more visible to users. We will also simulate the game in fifty times speed compared to real life, that is, one second in real life is fifty seconds in the game. This will ensure that users can progress through the simulator in reasonable time. Trains move and adjust *very* gradually in the real world, so users would still have ample time to react to inclination changes even when time is sped up.

Testing Plan

Since this simulation will be used for training purposes, stability, optimization, and intuitiveness are top priorities. Rigorous testing reinforces all three of these pillars. Testing the simulation is fast and easy inside the Unity Editor. While each of the team members write code, the project can enter “Play Mode” at the click of a button to see changes in action, and even simulate a plethora of screen sizes. Code reviews are organized during our weekly sprints in our Trello board to ensure all code is thoroughly tested for potential bugs. Our structure of a weekly build for Windows at the end of each sprint allows for regular, frequent testing to avoid any last-minute bugs.

Schedule

**Week 1 (9/1 - 9/4)**

* Teams Created and Projects Assigned
* Set Up GitLab Repository
* Set Up Slack for Team Communication
* Initial Team Meeting
* Contact Client

**Week 2 (9/5 - 9/11)**

* Initial Client Meeting with Jeff Girbach
* First Triage Meeting with Tommy Hojnicki
* Set Up Trello Board
* Technical Design First Draft
* Received Physics API From Client

**Week 3 (9/12 - 9/18)**

* Build Test App to Target Platforms
* Status Report Presentation
* Design System Architecture
* Refine UI/UX First Draft

**Week 4 (9/19 - 9/25)**

* Main Menu Screen
* Train Selection Screen
* Track Selection Screen
* Game Scene with UI Controls
* **Project Plan Presentation**

**Week 5 (9/26 - 10/2)**

* Smooth Train Movement
* Game Manager
* Train Prefabs
* Terrain Generation from .trk Files
* Add Force Visuals

**Week 6 (10/3 - 10/9)**

* Additional Track Prefabs
* Win and Lose Screens
* Pause Button
* Saving and Loading Progress
* Working Alpha Demo

**Week 7 (10/10 - 10/16)**

* **Alpha Presentation**
* Updates to Saving and Loading
* UI Updates
* Create Levels
* Update Train Sprite

**Week 8 (10/17 - 10/23)**

* UI Updates
* Add Tutorial Level
* Add Pause Menu
* Implement Force Threshold System
* Implement Progress Tracking

**Week 9 (10/24 - 10/30)**

* Implement Fences
* All Levels Added
* Create Game Guide/Tutorial
* Playable Build with Multiple Levels

**Week 10 (10/31 - 11/6)**

* Variable Speed Limits
* Sounds
* UI Updates

**Week 11 (11/7 - 11/13)**

* Status Report Presentation
* Beta Debugging and Refining
* Working Beta Build
* Create Project Video Storyboard

**Week 12 (11/14 - 11/20)**

* **Beta Presentation**
* Finalize Project Plan
* Create Project Video Script

**Week 13 (11/21 - 11/27)**

* Shoot and Edit Project Video
* Implement Stretch Goals
* User Test Game
* Debug and Refine Game

**Week 14 (11/28 - 12/4)**

* Shoot and Edit Project Video
* Status Report Presentation
* Final Debugging and Refinement
* Finalize Project Video

**Week 15 (12/5 - 12/11)**

* **Submit Project Video**
* **Submit All Project Deliverables**
* Design Day