Scope and Vision Updates and Formatting

Engineering Simulations with Game Development Tools Group 3

Team Members: Stanley Hale, Ezra McDonald-Mueller, Benny Xu, Greggory Hickman
Project Partner: Chris Patton

Abstract: This project is a simulation program focused on the physics of vehicles and how they interact with terrain and physical forces. Within this program, the user is able to move a vehicle around with a keyboard or controller, and watch how the vehicle behaves in its environment. Additionally, features such as random terrain generation and spatial audio are being added to make the experience more immersive. Currently in development are more additional features, including the ability to modify the vehicle's parameters during the program's runtime (weight, traction, color, adding more vehicles, etc.), and a system that will allow the user to record their vehicle's state to a file, which can then be loaded back into the program later at the user's convenience! This will allow a user to show their scenarios to other users who have this program, as well as save their work for later, if they need to take a break.

Change Log

Change Location	**Description**
Github Issue Column - 2.7.1	We got rid of this column due to us not utilizing the github issues feature. It doesn't really make sense for our current workflow as it just adds unnecessary steps.
Recording User Inputs - 2.1.2, 2.7.3, 2.8, 3.5, 3.6	We decided to not record user inputs for a replaying feature. This is due to time constraints as we have not yet invested enough time previously to understand this system. Now we believe we will not have enough time for this feature.
Multiplayer/P2P - 2.1.2, 2.4.1, 2.7.3, 2.7.5, 2.8, 2.9, 3.1, 3.2, 3.3, 3.4, 3.5, 3.7, 3.8.1, 3.8.2, 4	We decided to shift focus away from creating multiplayer support for this simulation program in favor of trying to achieve the higher priority stretch goals for our program, because the implementation was taking too long.

Table of Contents

Software Development Process (SDP)

1.1 Principles

- We will always do our best to respond to communications on our discord server and on our Microsoft Teams chat within 24 hours of a message being sent.
- We use a github project board for our team to keep track of what tasks still need to be completed, and who is currently working on what task, so that duplicate work can be avoided.
- The backlog will always have work items ready for the next two weeks at the minimum.
- All changes need to be developed in a separate git branch, preferably with a separate git branch for each major task.
- Once the feature, bug, or refactoring is done, a Pull Request is created from the separate branch to the main branch.
- Each Pull Request has to be reviewed by at least one team member before being merged.
- The Pull Request needs to comply with the Definition of Done (see later) and should be linked to the corresponding work item.

1.2 Process

- Backlog and Planning, once per week.
- GitHub Project Board, contains Ready, Backlog, In Progress, In review, and Done sections that task cards can be dropped into.
- Weekly check-in meeting with all team members and our Project Partner (Chris Patton), once per week.
- Weekly check-in meeting with all team members and our TA (Ananya Sundararajan), once per week.

1.3 Roles

- Subgroups
    - Terrain: Benny, Ezra
    - Multiplayer: Greggory
    - UI: Stanley
    - Audio: Stanley

We will all rotate the responsibility of leading the weekly meetings with our TA, with a different team member leading each week.

This list will expand/change over time as needed.

1.4 Tooling

Version Control	GitHub
Project Management	GitHub Issues and Projects
Documentation	README
Linting and Formatting	Rust with Visual Studio Code (https://code.visualstudio.com/docs/languages/rust)
CI/CD	GitHub Actions
IDE	Visual Studio Code
Graphic Design	LucidChart
Others	Bevy Engine (https://bevyengine.org), Rust (https://www.rust-lang.org)

1.5 Definition of Done (DoD)

- Acceptance criteria are validated
- Changes are merged to the main branch
- Documentation is updated, including deployment instructions if any
- Breaking changes are evaluated/avoided
- Demo is prepared for next TA/project partner meeting

1.6 Release Cycle

- Automatically deploy to staging every merge to main branch
- Deploy to production every release
- Release every three months
- Use semantic versioning `MAJOR.minor.patch`
  - Increment the `minor` version for new features
  - Increment the `patch` version for bug fixes
  - Increment the `major` version for breaking API changes

1.7 Environments

Environment	Infrastructure	Deployment	What is it for?	Monitoring
Production	GitHub Releases and the Main Branch	Release	Fully tested and verified builds that function as intended with no major bugs (hopefully)	Can be viewed on our public GitHub page under “Code”, and in our releases tab, once we have GitHub Actions set up.
Staging (Test)	GitHub Issues, Separate GitHub Branches	Minor Changes & Patches	New unreleased features and integration tests	Can be viewed on GitHub issues, and listed in the GitHub Project Board under “In progress” or “In review”.
Dev	Local (macOS and Windows)	Commit	Development and unit tests	N/A

Product Requirements Document (PRD)

2.1 Problem Description

At the moment, there isn't a very good free program that accurately simulates the physics of vehicles. If we were to create such a program and make it available to download for free, our program would be a great fit for many consumers.

2.1.1 Scope

This product will not be sold commercially, and a big reason this project is happening is because it will be used by our project partner, Chris Patton. Though, the project will also be open to the public to download for anyone else who wants it.

2.1.2 Use Cases

These user stories are adapted from the list of our project's basic requirements.

- The user will be able to use a keyboard or gamepad as a form of input for the vehicle.
- The user will modify vehicle parameters, like the weight of each part, the size of each part, the traction of the wheels, and more with an on screen GUI (stands for "Graphical User Interface", which is a window on the screen that the user can click and interact with).
- The user will be able to load a vehicle's parameters and seed from locally saved data.
- The user will be able to generate terrain to then start a game on.
- The user will be able to interact with the environment.
- The user will hear the vehicle's engine audio change in pitch depending on the acceleration of the vehicle.

2.2 Purpose and Vision (Background)

Our aim is to improve an existing interactive driving simulation that provides users with a realistic approximation of how real vehicles would act in certain environments, terrains, and conditions that the user can set up.

We want our program to be able to model the physics of vehicles accurately enough that in most cases, it would nearly identically show what a real-life car would do. Essentially, we want people interested in physics to be able to use this program to model specific situations for whatever purposes they may need it for.

2.3 Stakeholders

- Chris Patton (Project Partner)
    - Updated weekly
    - Update on progress made and get feedback on research/implementation
    - Can help make decisions and guide us:
        - New features and implementation
        - Project scope
        - Project timelines
        - Suggest priorities
    - Needed to make informed decisions:
        - Skill sets acquired/needed
        - Timeline and team members schedules
- Team Members
    - Updated weekly
    - Update on progress made and get feedback on research/implementation or testing

- Can help make decisions and guide us:
    - New features and implementation
    - Project scope
    - Project timelines
    - Assign priorities
- Needed to make informed decisions:
    - Design process
    - Testing
    - Skill sets acquired/needed
    - Timeline and team members schedules

2.4 Preliminary Context

2.4.1 Assumptions

- The game engine we are using for development, Bevy, is well-tested, reliable, free to use, and open-source.
- We will be developing the application using the Rust language, which is well-tested, reliable, free to use, and open-source..
- We will be using the following Bevy libraries, which will be well-tested, reliable, free to use, and open-source:
    - Rigid body dynamics library
    - Rigid body integrator library
    - Noise library
    - Grid Terrain for terrain mesh creation
    - Cameras for basic bevy camera controls
    - Flo_curves for bezier curves
    - Bevy_egui for egui integration into the bevy game engine
    - Noise crate for generating noise.
- Our program will work on Windows 10, Windows 11, macOS 14, Linux systems, and embedded systems like mobile devices.

2.4.2 Constraints

- Our program must be efficient enough that users with less-powerful computers are able to run it without using more than 80% of their CPU or GPU at any given time.
- We don’t have a budget, so we need to use resources that are free and open-source or create them ourselves.
- We have from September 2023 until April 2024, which is about seven months in total, to produce a version of the existing program with the basic requirements successfully implemented.

- Deliverable due dates are on canvas (some solo, some team oriented).
- Have 4 people to work on this project “part-time”.
- Limited by the compute power of team members’ development machines.

2.4.3 Dependencies

We’re dependent on the Bevy game engine being reliable, robust, and efficient.

- Bevy game engine
- Chris’s initial car demo: https://github.com/crispyDyne/bevy_car_demo
- We will be using the following Bevy libraries:
    - Matchbox
    - Rigid body dynamics library
    - Rigid body integrator library
    - Noise library
    - Grid Terrain for terrain mesh creation
    - Cameras for basic bevy camera controls
    - Flo_curves for bezier curves

2.5 Market Assessment and Competition Analysis

Alternatives to our program:

- **GTTrack (link to website):** This is a great tool that does most of what our program aims to accomplish, but it is very expensive (currently $899 USD as of 10/15/2023), and consists of a set of large, heavy equipment with software that runs on said equipment.
- **aVDS Advanced Vehicle Driving Simulator (link to website):** This is a great tool, but also requires an expensive set of equipment. Though the company that makes this product claims to deliver a custom product based on customer specifications, which no other company that I researched offers.
- **American Truck Simulator, Euro Truck Simulator 2, World of Trucks (link to website):** These games are limited to only cargo trucks, and they do not claim to provide a realistic truck-driving experience. Though, they do simulate road and highway traffic realistically.
- **BeamNG.drive:** Soft-body physics simulation for crashes and simulation of vehicle movement in realistic environments. Able to record and replay vehicle states from a file.
- **Car Physics Simulator:** Suspension and crash physics. Modify car properties.

2.6 Target Demographics (User Persona)

There are multiple reasons a user might want to use this program:

- The user is interested in vehicle designs, and needs a program to quickly and easily simulate a vehicle.
- The user drives or races professionally, and wants to use the program to test certain things that would be dangerous to attempt in real life.
- The user is a physics student, or at least a person with an interest in the physics involved with vehicles, and wants to use the program as a learning tool.

Here are a few user persona that fit into these demographics:

- Taylor is a 50 year old who works in robotics at a company, and they want to be able to simulate how some of their robots might function before building a prototype.
- Jessie is a 30 year old professional racer who needs to have a deep understanding of the physics involved with their car in order to stay safe.
- Charlie is a 18 year old high school student with an interest in both physics and vehicles.
- Tony is a 57 year old retired trucker who has little computer knowledge, but wants an immersive driving experience that he can record and look back at later.

2.7 Requirements

2.7.1 User Stories and Features (Functional Requirements)

User Story	Priority	Dependencies
As a user, I want the physics of the vehicles to be as accurate as possible to a real-life vehicle, so that I can use the program to predict how a real vehicle would behave.	Must Have	Rigid body dynamics and integrator library
As a user, I want the weight, traction, and other properties of specific parts of the vehicles to be customizable, so that I can make the vehicle in the simulation more similar	Must Have	TBD

to the real-life vehicle that I am trying to simulate.		
As a user, I want the environment to look realistic, so that I can better visualize the scenario happening in the real world.	Should Have	Noise library
As a user, I want to be able to save and load the settings and parameters of both the vehicle and environment in the program, so that I don't have to waste time manually recreating those conditions later.	Must Have	TBD
As a user, I want to be able to manually remap the controls on both keyboard and controller, so that I can use the control scheme that is most comfortable for me.	Could Have	TBD

2.7.2 Non-Functional Requirements

- The program should work equally well for keyboard users and controller users.
- The program should not have a significant increase in lag when intended features are used, within reason.
- The program should work perfectly on all different screen sizes, and in both windowed and fullscreen mode.
- If the program has a fatal error, it should crash and show an error report instead of simply freezing indefinitely.
- Because many different people will be working on this project, our code should be well-documented and well-commented, following good practices and standard variable naming conventions.

2.7.3 Data Requirements

- Specifically, vehicle parameters and terrain seeds will be saved into a readable format so that existing conditions can be recreated later

2.7.4 Integration Requirements

- This program will use the Bevy game engine, which uses Rust, which is a fast and efficient programming language. It's also free.

2.7.5 User Interaction and Design

- <u>Main Menu:</u>
    - <u>Settings Menu:</u>
        - Modify audio volume
        - Modify control input
        - Modify video settings
        - Modify vehicle parameters, there will be text fields for
            - Custom max/min speed
            - Mass of car
            - Gravity
            - Friction
            - Terrain intensity
            - Vehicle acceleration rate
    - Seed Text Field
        - Modify/choose a seed to generate terrain before starting the game
    - Play Button
    - Quit Button
    - Load seeds/generated terrain from seed list/files
- <u>In-Game Menu:</u>
    - View Controls
    - Settings Menu
        - Modify audio volume
    - Main Menu
    - Quit Button

2.8 Milestones and Timeline

- Visualize the motion of a vehicle in an environment.
    - This feature is technically already implemented, but we plan to improve the visuals throughout the next couple of months.

    - ➢ Expected timeline: April 2024
- ❖ Translate user inputs (mouse, keyboard, gamepad) to vehicle controls.
    - ➢ This feature is only partially implemented. We plan to have this feature complete as soon as possible, because the development of other features may depend on the completion of this feature.
    - ➢ Expected timeline: DONE
- ❖ Modify vehicle parameters (weight, size, etc) with an on screen GUI.
    - ➢ This feature should be completed within the first month, and gradually improved throughout the following month.
    - ➢ Expected timeline: April 2024
- ❖ Terrain Generation
    - ➢ Users will be able to choose a seed to procedurally generate the terrain.
    - ➢ The environment will include navigable terrain and obstacles.
    - ➢ Expected timeline: April 2024
- ❖ 3D Audio.
    - ➢ Vehicle engine audio that changes as vehicle acceleration changes:
    - ➢ Environmental audio: Dependent on Terrain Generation
    - ➢ Expected timeline: March 2024

2.9 Goals and Success Metrics

Goal	Metric	Baseline	Target	Tracking Method
The 3D visuals look realistic and visually pleasing	How well do the program's visuals allow the user to imagine the scenarios that take place in the simulation, in real life?	Crude visuals, with most surfaces being a solid color.	Realistic visuals, with accurate shading.	Survey
The UI is complete	Does the UI allow the user to use every intended feature?	Very little UI, if any at all	Menus and buttons for every feature.	Checklist
Simple UI	Average feature location time	30 seconds	10 seconds	Plausible Analytics
The vehicle	How accurately	Uses the game	Uses	Compare the

physics are realistic, and measurable	does the program simulate real life?	engine's default physics.	finely-tuned custom settings that makes the vehicle behave more realistically, and shows the velocities and accelerations of all implemented vehicles.	program's simulation to real life vehicle tests
Stability	Crash/Bugs during a single session	< 4 noticeable bugs, < 1 crash	No bugs or crashes	User reports
Product-market fit	How would you feel if you could no longer use this product?	Very disappointed < 40%	Very disappointed > 40%	Interview

2.10 Open Questions

Q: What is our target audience?

A: Engineers will use it as a tool for engineering projects. Theoretically, they might be used for Chris Patton's client.

Q: What kinds of vehicles will we be simulating?

A: Our team will be focusing on automobiles (cars, trucks, etc.)

Q: What operating systems will our program be able to run on?

A: Everywhere! Mac, Windows, and Linux.

Q: Are there any other good existing alternatives to the program that we are going to be developing that I didn't mention?

A: Yes, and they have been added to the "Market Assessment and Competition Analysis" section of this document.

Q: Will we be starting from scratch?

A: We will be starting from Chris Patton’s sample program.

2.11 Out of Scope

- We will only be implementing this program onto desktop and laptop computers on Windows, macOS, and Linux systems. We will not be porting the program to consoles or smartphones unless we have extra time leftover, and we think it would be viable.
- This section will be updated with additional entries once I have more information.
- Randomly generated racetracks that you can start on the map.
- There will be no weather simulation.
- There will be no interactable animals.
- There will not be many vehicle shapes/bodies.
- There will be no ray tracing.

PART III
Software Design and Architecture

3.1 Introduction

This document outlines the architecture for a Rust-based motor vehicle simulation running on the Bevy Game Engine. While the architecture for this program likely won't be complicated, it is still a good idea to make sure that it is well-defined so that future developers who work on the program are able to understand exactly how that different parts of the program work, making it easy for them to build upon the existing code.

Our project is entirely contained within a single application that runs on Windows, Mac, or Linux. Our project does not depend on any kind of database to function, and all data for the program is stored within the project files. Our program will have the ability to save and load states from files, which will also be stored locally.

3.2 Architectural Goals and Principles

The structure is intended to be as simple as possible, because it is only a single application with no subprocesses. We'll design the architecture to be robust, understandable, scalable, secure, and complete.

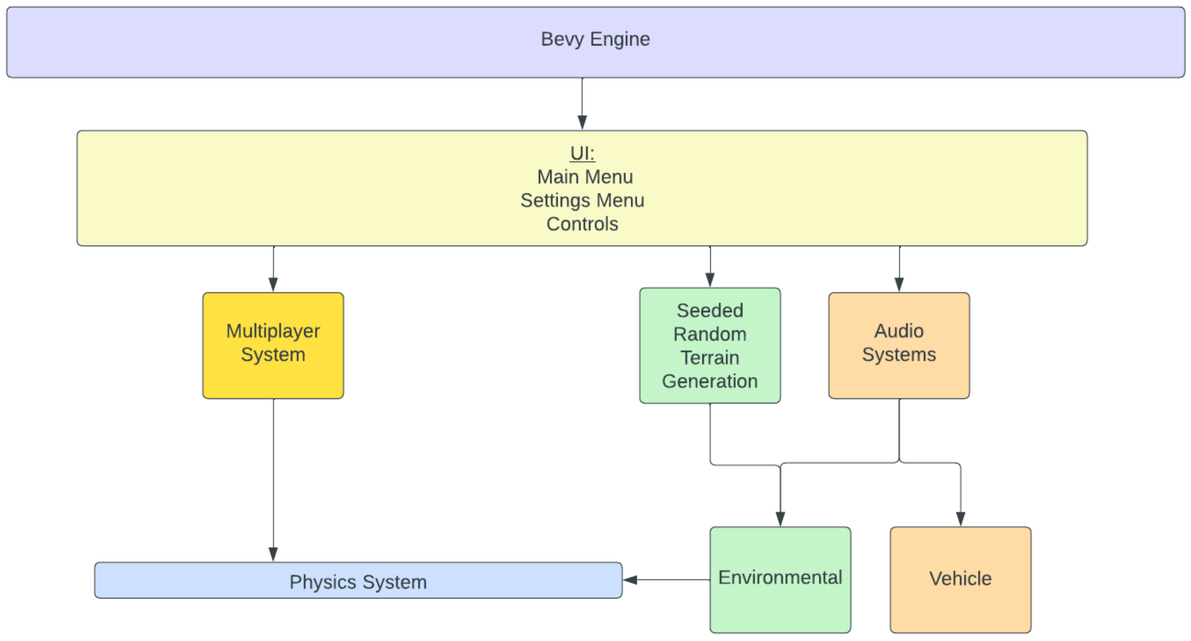
The architecture's priority is to be as simple as possible as the program gives the user options to change certain parameters while taking care of the essentials itself. Because this program will not be running any subprocesses or pulling data from anywhere else, we want to make the architecture focused on easy implementation of features, accessibility to other devs and ease of navigation.

3.3 System Overview

For a local multiplayer session:

The Application World and Renderer both run in parallel, and are synchronized once per frame. For more information, see this page: https://bevy-cheatbook.github.io/gpu/intro.html.

System Component Overview Diagram:



3.4 Architectural Patterns

A model-view-controller pattern would be accurate for this project as the game is broken into components with the camera, physics, and input controllers. Where the camera controller manages the scene views of the vehicle and the input component manages user inputs to control the vehicle and camera view.

3.5 Component Descriptions

- Bevy: Underlying basis for this project as it is the game engine that houses the core components for everything such as UI, Audio, 3D Graphics, etc.
- User Interface: Uses Bevy's UI systems to display and interact with content pertaining to Terrain, Physics, and Audio. Physics parameters, Audio settingswill be the primary UI interactions for this project, with additional focus on smaller features such as user interactions, display data, and navigation throughout software.

  [See Interface Design section for more details]
- Physics systems: This includes Bevy's basic physics systems as well as the ones included in Chris's demo. This will allow us to modify the physics parameters of the vehicles in real time using the UI.
- Multiplayer System: Multiple car entities in the game world can be generated and assigned different control systems.
- Terrain Generation: Terrain can be randomly generated using UI before starting the game. This would also interact with the Audio system through environmental Audio.
- Audio Systems:
  - Environmental: Specific terrain assets will also have Audio tied to them, such as forested areas having bird sounds, areas with water having sounds of flowing water. All with 3D spatial audio.
  - Vehicle: Simulate a simple X-piston engine in 3D spatial audio depending on vehicle parameters, such as acceleration. Simulate changing gears.

3.6 Data Management

All data and assets will be either stored on the user's machine. As for the file structure, we are using the standard Bevy structure, which consists of the root folder, which contains a "public" and "src" folder. The "src" folder contains most of the code, as well as the assets (like images, or models) that will be used in the program.

Data regarding the simulation such as;
- Car properties
- Generated terrain

Will be saved into a custom file for easy recreation and sharing of results

3.7 Interface Design

- <u>Main Menu:</u>
    - <u>Settings Menu:</u>
        - Modify audio volume
        - Modify control input
        - Modify video settings
        - Modify vehicle parameters, there will be text fields for
            - Custom max/min speed
            - Mass of car
            - Gravity
            - Friction
            - Terrain intensity
            - Vehicle acceleration rate
    - Seed Text Field
        - Modify/choose a seed to generate terrain before starting the game
    - Play Button
    - Quit Button
    - Load seeds/generated terrain from seed list/files
- <u>In-Game Menu:</u>
    - View Controls
    - Settings Menu
        - Modify audio volume
    - Main Menu
    - Quit Button

3.8 Considerations

3.8.2 Performance

With performance in mind, we are using the Bevy Engine which uses Rust, a very efficient language. We intend for our program to be able to run on desktop computers and laptops that are at least reasonably new (i.e. no computers from the 90's), and I don't foresee our program ever having any major performance issues so long as we use best practices in our coding, and keep everything well-documented.

Performance should be prioritized in regards to terrain generation as we would not want the user to have to wait 5+ minutes just to get in the game and start playing. We could monitor and record how performance is impacted by changes that we make to the generation system. Settings to optimize performance over visuals/audio may also be implemented to remedy over stressing.

The simulation should run at least 30 frames per second for most reasonably-new systems. This can be done by optimizing the code for efficiency. This can be done by optimizing code.

3.8.3 Maintenance and Support

Maintenance and support may be necessary for another capstone team in the future. Our team members would then be the ones to gather documentation for the project once we are done. If another capstone team will not take the project over we can always publish it as an open source project for community members to keep it alive if there is interest. Documentation of systems and how to add onto them would be vital for either outcome. We can gather feedback through surveys or a public issues board to gather data and prioritize them. We will also let our project partner, Chris Patton, know of anything else that he should be aware of.

3.9 Deployment Strategy

Deployment will be through Bevy builds that will be released on our github. We could set up a CI/CD system for building the project whenever there are merges with the main branch. This would utilize github's resources to build the project each time as it runs the scripts.

We will likely keep our project on Github using the Github Releases system, though our project partner, Chris Patton, may choose to release it elsewhere as well.

3.10 Testing Strategy

There is a testing framework for the Bevy Game Engine, but for our project, such a framework would not be very applicable. It would take much longer to build an automated testing framework for each feature than to manually test it ourselves, which would defeat the whole purpose of an automated testing framework. So, we will just be doing our testing manually.

Unit Tests:

- Individual tests will first be performed by the team member responsible for development if said feature.
- Once another member has also tested the feature and given the green light, we can begin integration testing.

Integration Tests:

- This can be done by any team member or outside source willing/wanting to test it.
- If everything interacts with everything as planned, we can finalize implementation of the added feature into the current build.
- Any issues will be reported, changes will be made, and integration testing can continue.

4. Glossary

CI/CD: Continuous Integration and Continuous Development

UI: User Interface