# Full Project Documentation

I.R.S. (CS Game Design Competition - Group 3)
Ethan Largent, Riley Elwell, Connor Olsen, Kevin Huynh
CS46X - CS Capstone

***Abstract***

A complete collection of the documentation from a senior capstone group titled: “CS Game Design Competition - Group 3” at Oregon State University. Tasked with creating a video game in which a player controls a vehicle that has ADAS properties attached to it and AI traffic appropriate to its setting, they chose to make a starship simulator game. The documentation consists of the product requirements, software design and architecture, and software development process.

# Table of Contents

# 1 Product Requirements

## 1.1 Problem Description

In the busy environment of a college campus, the demand for timely and efficient food delivery to students is critical. To address this need, we are developing an entertaining gaming experience where players take on the role of driving a robot tasked with navigating through the campus to deliver food orders to hungry students. The primary objective of the game is to deliver these orders to designated locations across the campus within the shortest time possible while avoiding obstacles such as pedestrians and vehicles.

### 1.1.1 Scope

The scope of our project is limited to a small portion of the OSU campus, where you operate a food delivery robot. You will be delivering food to various locations while avoiding cars and pedestrians (trains and other AI controlled robots are set as stretch goals). ADAS functions like LIDAR will be simulated through raycasts, and perform functions like automatic braking, collision detection, and curb detection.

### 1.1.2 Use Cases

The user starts the game at a random food delivery location and drives to a dorm building avoiding all collisions and in a timely manner. The user gains a good score and amount of coins for the delivery and is given a food delivery location to pick up food from. They are not given a time limit or collision penalty while driving to pick up the food.

The users tried to get to the dorm building very quickly, but hit many obstacles on the way. They get a poor score since the food got messy and thrown around on the trip.

The user has been playing for a while and has received a large sum of coins. They use these coins to buy cat ears for their robot that appear on their player as they play. They then go back to the menu to buy more items and arrange their outfit. Some of the items they wanted to buy may have required achievements to be made before purchase.

The user leaves the game in the middle of a session to run an errand. After a certain time being idle, the game returns to the main menu.

The user doesn’t start the game and instead goes into the menu to change up their settings. They save their settings and when they start the game their settings are in effect.

## 1.2 Purpose and Vision (Background)

Our purpose and vision is to alter the gaming landscape by providing an immersive and educational experience that merges the excitement of driving with the challenges of food delivery in a dynamic college campus environment. Through innovation and attention to detail, we aim to entertain players while also teaching skills in strategic decision-making, reflexes, and familiarity with Advanced Driver Assistance Systems (ADAS) technology. By evolving and expanding our game's features and content, we hope to drive positive change in how games are perceived and experienced.

## 1.3 Stakeholders

Potential Employers – no updates, no decisions

Competition Judges – no updates, no decisions

Project Partner – weekly updates, advice based off of our need

Project Groupmates – weekly updates, equal say in all decisions for the project and needs to be fully up to date with everything going on in the group

Project TA – weekly updates, guidance based off of our progress and goals

Expo Users – no updates, take note of desires for the project

Driving Instructors – request feedback after version releases, take note of desires for project

Driving Students – request feedback after version releases, take note of desires for project

For Enjoyment Users – request feedback after version releases, take note of desires for project

## 1.4 Preliminary Context

### 1.4.1 Assumptions

The game will run on Unity

We are free to use any tools we wish as long as it isn’t a nearly complete version of our project

We have at least 6 months dedicated to coding the project

The workload is split amongst four people

The game will be single player and doesn't need internet access for any essential features

The game is for PC at a minimum and further ports may be stretch goals

The game is in third person so you can see cute robot

The game wants all non-player characters to avoid collisions with other non-player characters, but will collide with the player if intervened with

### 1.4.2 Constraints

We are not to use paid tools not provided by OSU

The game is to run on an average PC

We have 6 months for the project

We have four people working on the project

Our group members each have at least 3 years of college experience

### 1.4.3 Dependencies

We need our workplace and tools setup on each group member's machine before we can begin development

We need a functional prototype before we can implement advanced features

We need our MVP complete before we can work towards our stretch goals

## 1.5 Market Assessment and Competition Analysis

Forza: The game looks very realistic and the driving controls feel realistic, however the game is also very forgiving with mistakes, has no new modern vehicle features, focuses on racing rather than real life driving practice and it makes itself feels like a videogame.

BeamNG.drive: The game does well to focus on driving in the real world and abiding by traffic laws. It also does well with having realistic visuals, driving controls, and collision physics. However, the game can exaggerate its collisions and it fails to demonstrate the ADAS features that new modern vehicles have.

iRacing: The game looks and feels very realistic, the driving controls feel realistic, the mistakes are unforgiving. However, the game has no new modern vehicle features, focuses on racing rather than real life driving practice, and there is a subscription based payment to play the game.

City Skylines: A great example of vehicle and traffic AI. However, this game is a city builder and doesn't do well with player controlled vehicles or much else from a drivers point of view.

Mario Kart: The game does well to create an enjoyable experience. However, the driving controls are very unrealistic, mistakes are horribly unforgiving, it makes itself feel very much like a video game, and it fails to demonstrate any of today's ADAS features realistically.

Grand Theft Auto: Provides a free roam driving experience with realistically stupid AI drivers. The driving physics is floaty and geared more towards fun than simulation.

Many game engines have quality physics systems and other tools that have no reason not to be used for our project.

## 1.6 Target Demographics (User Persona)

Archibald and Barthalamult are first year OSU students checking out the engineering expo. They are wanting to find a cool project to have fun with.

Chris and Debbie are employers looking to hire game developers at the OSU engineering expo.

In addition to the above examples, we have potential categories we could group users into:

Enjoyment vs Education

Racing vs Law Abiding

Casual vs Competitive

Platform (PC)

## 1.7 Requirements

### 1.7.1 User Stories and Features (Functional Requirements)

| User Story | Feature | Priority | GitHub Issue | Dependencies |
|---|---|---|---|---|
| As a driving student, I want to learn to drive at home, so that I can continue to learn even when I'm not in a car. | TBD | Must Have | TBD | N/A |

| | | | | |
|---|---|---|---|---|
| As a driving student, I want to have fun while learning to drive, so that I stay motivated with my learning. | TBD | Should Have | TBD | N/A |
| As a driving student, I want to practice driving cars before I get one, so that I can know what kind of car I should get and if I should get one. | TBD | Should Have | TBD | N/A |
| As a car buyer, I want to practice driving a variety of cars, so that I know what kinds of vehicles I prefer. | TBD | Should Have | TBD | N/A |
| As a driving instructor, I want to give my students an opportunity to learn with no risks, so that they can stay safe when they learn to drive. | TBD | Must Have | TBD | N/A |
| As a driving instructor I want to give my students good un-proctored study material, so that I don't have to spend as much time teaching. | TBD | Could Have | TBD | N/A |
| As a new gamer, I want to see what driving simulator games are all about, so that I can know if I like them or not. | TBD | Must Have | TBD | N/A |
| As a gamer, I want to have a realistic driving experience in a game, so that I can have fun and immerse myself. | TBD | Should Have | TBD | N/A |

| | | | | |
|---|---|---|---|---|
| As a gamer, I want to see how reckless I can drive without getting in an accident, so that I can vent my reckless driving intrusive thoughts. | TBD | Could Have | TBD | N/A |
| As a gamer, I want to have a change of pace and drive leisurely, so that I can have a low stress driving experience in my games. | TBD | Could Have | TBD | N/A |
| As an experienced gamer, I want to earn all of the achievements in a realistic driving simulator, so that my friends think I'm cool. | TBD | Could Have | TBD | N/A |
| As a student at the expo, I want to find a project that peaks my interest, so that I can become inspired in my education goals. | TBD | Should Have | TBD | N/A |
| As an employer at the expo, I want to see an impressive representation of a student's skills, so that I'm confident in their ability as an employee. | TBD | Should Have | TBD | N/A |
| As a judge for the game design competition, I want to find a realistic driving simulator with ADAS, so that I can have a winner for the competition. | TBD | Will Have | TBD | N/A |

### 1.7.2 Non-Functional Requirements

The quicker the response time the better, but it must feel fluid/be playable

The game should be stable with minimal bugs and no crashes

The game should have minimal frame drops

The game should run spectacularly acceptably on an average grade of PC

### 1.7.3 Data Requirements

We will need to handle a lot of assets and scripts to create our game

### 1.7.4 Integration Requirements

The game will likely be fully developed using Unity

### 1.7.5 User Interaction and Design

Start-up: Opening the game brings the user to a menu screen that gives the user several options from there: Play, Quit, and Settings. Each brings the user to the corresponding menu.

Play: Brings the user to a screen showing the robot in third person view on the OSU campus map ready to be controlled with player input. The world around them will change close to how it does in the real world (vehicles, pedestrians, etc.). Pressing escape brings the player to the Pause Menu and freezes all interactions on the map (Play may also bring users to a game mode screen that could change how the game plays before they are brought to the core of the game).

Pause Menu: Brings the user to a pause menu giving them several options: Resume, Restart, Settings, Return to Menu, and Quit. Resume will bring the user back to Play. Restart will reset the entire map from the beginning of their session (but still in Play). Settings brings the user to Settings. Return to Menu returns the user to the Startup screen. Quit makes the player quit the application, stopping all interactions.

Settings: Brings the user to a screen displaying all of the alterable settings in the game as well as a save button and a cancel button. Pressing save will change the users' settings to reflect what they set them to and send where they came from. Pressing cancel will send them to where they came from without saving their changes.

Quit: Stops all interactions between the player and the system, and stops the application.

## 1.8 Milestones and Timeline

1. Create Workspace
2. Learn UE5
3. Create a rough prototype – requires 1 and 2

    a. Basic graphics
    b. Basic map – requires a
    c. Basic vehicle behavior
    d. Basic traffic system
    e. Basic controls – requires b and c
    f. Basic non-player AI – requires b, c and d
4. Update vehicle behavior – requires 3
5. Update player control – requires 4
6. Develop non-player vehicle AI – requires 4
7. Complete a map layout – requires 3
8. Update map graphics – requires 7
9. Implement full traffic system – requires 7
10. Update vehicle graphics – requires 3
11. Update physics – requires 3
12. Create Menu/Settings/Pause Screens

## 1.9 Goals and Success Metrics

| Goal | Metric | Baseline | Target | Tracking Method |
|---|---|---|---|---|
| Increase user performance | Decrease in collisions over time | TBD | TBD | Plausible Analytics |
| Increase player base | Average active players | TBD | TBD | Plausible Analytics |
| Decrease bug occurrences | Number of reported occurrences | TBD | TBD | Player Reports |
| Product is market fit | How would you feel if you couldn't play the game? | TBD | TBD | Survey |

## 1.10 Open Questions

Q: Will we port the game to other consoles/controller schemes?

Q: What will each of our group members be focusing on?

Q: Are we planning to take this project beyond the class?

## 1.11 Out of Scope

No non-player humans involved in traffic systems

No custom advanced water physics or lighting

No focus put on racing

No multiplayer or cross-save

# 2 Software Design and Architecture

## 2.1 Introduction

This document outlines the architecture for a 3D driving-based simulation/game that uses ADAS heavily. The chosen architecture needs to ensure stability, performance, pseudorealism, realistic physics, and the ability to demonstrate creativity. It should also allow us to create our application for multiple operating systems (Windows, Linux, Mac).

## 2.2 Architectural Goals and Principles

The architecture should allow for:

- Integration with other program files (3D models, animations, scripts, etc.)
- Performance/Stability that allows the application to run at 30+ FPS on a typical machine
- Flexibility in features to allow for more design choices and scalability
- A great physics system will enable us to have collision physics in our simulation
- Good graphic capabilities to make realistic-looking vehicles and environments

## 2.3 System Overview

Our system will store our assets (models, art, graphics) and scripts (code) for the project inside of Unity. The game engine will mainly handle the connections, only requiring us to import and export necessary files. In terms of code specifically, we want to structure using OOP (in C#) to enhance readability, stability, and inheritance qualities.

## 2.4 Architectural Patterns

We would likely use a layered architecture for our project; these layers include:

- Presentation layer:

  Handles rendering of graphics, user interface, and input processing. Provides the visual and interactive aspects of the game (including audio), ensuring a responsive and immersive experience for players

- Game logic layer:

  Handles core game logic, such as physics simulation, vehicle behavior, and mission scripting. Ensures mechanics are consistent and player actions have the desired effects

  - AI drivers:

    The AI system will take in the world around it and use the vehicle I/O to control vehicles the same way a player would

  - ADAS systems:

Built on top of the vehicle I/O, adas features will be able to read information about a vehicle and adjust the vehicle's behavior accordingly

    - Robot Vehicle Controls & Logic
        - Similar drive to a tank
        - Using Unity's horizontal/vertical inputs
        - Simple logic for driving, ADAS features will be automatically enabled
    - Physics:
        - All vehicles will be dynamically affected by physics, implemented using Unity's Rigidbodies
        - Forces will be applied to different objects based on their velocities and directions when collisions occur (a car and robot for example)
- Data access layer (optional):

  Handles saving and loading game progress, player profiles, and other data. This would be needed if we decided we wanted a structured database to hold this type of information (or multiplayer).
- Network layer (optional):

  Handles the interaction between players and servers for multiplayer capabilities

## 2.5 Component Descriptions

- User Interface: Handles user interactions, displays data (vehicle speed, current objective, etc.), and provides navigation to other screens.
- Game Engine: Unity processes all requests the game requires for running automatically, allowing us to focus on the gameplay logic in our scripts and the graphical presentation we choose to use.
- Database/Serves: If we pursue multiplayer, we will likely need a server to handle the connections and a database to hold data for each user.

## 2.6 Data Management

This section does not apply to our project. We should not need a database to store user information inside the game. Unless we decide to implement a user-saved game system, multiplayer, or some other save mechanic, the game engine handles most of the work for us.

## 2.7 Interface Design

Unity provides several built-in options for UI, allowing the process to be fairly streamlined. We aim to keep the interface clean, utilize as little space as possible, and always be informative to the user. Without distracting from the actual gameplay, we want to have indicators for when the

ADAS features are active/inactive, a timer for the current delivery, the current score for the user, and a speedometer (or its equivalent).

## 2.8 Considerations

### 2.8.1 Security

Regarding our code, we will need to use good coding practices to avoid exposing important variables to the user. Also, we need to ensure our .gitignore file is set with the correct files, so nothing private gets leaked on GitHub, as it is a public repository. There may also be more security measures that need to take place. More security measures will be necessary if the game gets multiplayer, as P2P connections are generally insecure.

### 2.8.2 Performance

We want the performance of the actual game to be at least 30 frames per second, and this comes down to mainly the architecture (but also our utilization of the software and how many polygons our models contain). We want to ensure the game's necessary storage capacity is reasonable, assuming we decide to make it into an executable (local storage). We would also need server space to accomplish this if we pursue multiplayer.

### 2.8.3 Maintenance and Support

Future maintenance on this project would only likely be performed by our current team members. This project is changed every year, and teams start from scratch. Based on our overall vision for the end product, we will likely end development at the end of the year. However, if we were to provide updates later on, we would focus on new ideas for mechanics, bug fixes, or requested features from users who have tested the game for us.

## 2.9 Deployment Strategy

This section only somewhat applies to us because we have been told to focus on our own local machines. Our product will not be deployed anywhere else besides our own computers and will be demoed in this state at the Expo. Therefore, it seems that each environment will be essentially the same in our game engine or as an executable we are testing. Regarding the deployment strategy, we want a couple of releases each term, with new features, bug fixes, or patches. With these releases, we can also have users test the functions and gain feedback on the program.

## 2.10 Testing Strategy

As mentioned above, we will likely never ship our product past our local environments, unless we desire to. This means our testing will be done locally inside Unity or Windows computers.

The testing results may differ depending on our machines, which will help us compare benchmarks for potential users.

## 2.11 Glossary

ADAS - Advanced Driver Assistance System
UE5 - Unreal Engine 5
UI - User Interface
FPS - Frames Per Second
OOP - Object-Oriented Programming
PRD - Products Requirements Document
SDP - Software Development Process
SDA - Software Design and Architecture
P2P - Peer to Peer

# 3 Software Development Process

## 3.1 Principles

- We are to answer questions on our discord within 24 hours
- We are to show up on time to meetings
- We will use GitHub Project and Issues
- We will create issues in the project backlog
- Each field in an Issue should be filled out as completely as possible
- We are to update the status of our issues as we deal with them
- All changes to the project are made in their respective branches
- Our pull requests need to be appropriately linked with its respective issues
- Do not allow force-pushing on branches
- Non-Milestone Issues must be achievable in a single week
- Milestone Issues will get their own iteration

## 3.2 Process

We will…

- Be using a method very much like Scrum.
- Keep a large project backlog that contains everything we can think of to complete our projects.
- Assign items from our project backlog to our weekly iteration backlogs a week before the iteration begins.
- Work on the things in our current iteration during the week and update their status as we work on them. We will continue this process until we have completed our project backlog.
- We will meet with the TA every Monday at 2:15 to discuss our progress and ask for advice.
- Meet as a group every Wednesday at 2:15 to discuss our progress and do group assignments that require better communication.

## 3.3 Roles

In our group, we will have the following roles:

- Primary Artist
    - Riley
- Primary Audio Designer
    - Connor

- Primary UX/UI Designer
    - Kevin
- General Physics Developers
    - Split Among Everyone Case-to-Case
- Primary Gameplay Developer
    - Ethan
- Internal Vehicle Physics Developers
    - Split Among Everyone Case-to-Case
- Primary ADAS Simulation Developer
    - Connor

## 3.4 Tooling

| | |
|---|---|
| **Version Control** | GitHub and Unreal Source Control |
| **Project Management** | GitHub Issues and Projects |
| **Documentation** | Google Drive |
| **Test Framework** | Unity |
| **Linting and Formatting** | Visual Studio Code Extension |
| **CI/CD** | GitHub Actions |
| **IDE** | Visual Studio Code |
| **Graphic Design** | Google Slides |
| **Others** | Blender |

## 3.5 Definition of Done (DoD)

- All group members approve of the issues' completeness
- The associated feature branch is merged with the working branch
- The issue has its todo items complete and has been thoroughly tested
- The progress report has been updated to reflect the completion

## 3.6 Release Cycle

Deploy the working branch to an appropriate version branch and then to the release branch
Our versioning schematic (Major.minor.patch) goes as follows:

- Increment Major when the game has an entirely new feel
- Increment minor when a new feature is added, reset when Major increments
- Increment patch when a bug has been fixed, reset when minor increments or resets
    - Version 0 is our testable project
    - Version 1 is our minimal viable product
    - Version 2 is our completed stretch goal project

## 3.7 Environments

| Environment | Infrastructure | Deployment | What is it for? | Monitoring |
|---|---|---|---|---|
| Production | AWS through CI/CD | Release | Sleeping well at night | Prometheus, Grafana, Sentry |
| Staging (Test) | Render through CI/CD | PR | New unreleased features and integration tests | Sentry |
| Dev | Local (macOS and Windows) | Commit | Development and unit tests | N/A |