Design document for Delivery Trail

"Fast Wheels, Faster Meals!"

Game created by: Aleksi Halttunen, Arttu Saarela, Janne Kinnunen, Joona Laaksonen, Saku Karttunen

Overview

Delivery Trail is a delivery based driving game that is inspired by games like Trackmania and Crazy Taxi. The aim of the game is to deliver pizzas to the customers of the establishment that you are working at.

The gameplay revolves around completing levels as fast as you can. The levels are short and sweet but include some challenge and replay value as well. The amount of challenge is up to the player, since all the levels rank your performance and show you different time goals that you can try to hit. So if you are a new player, you don't need to go for gold and can instead enjoy the game more casually. More about this can be found at section Game Elements.

The goal is to create a game that is easy and fast to pick up, but one that will include enough replay value that more players will play through the levels more than once.

Technology

Game engine

The game engine of choice for the game is Unity. We chose Unity because it is used during this module's Game Programming course. The team is not very familiar with game development, so we did not want to add extra learning that would arise from another engine to our team.

Unity is also a good choice for our game, since it includes all sorts of nice features and lots of ready made assets from the Unity Asset Store.

Figure 1: Unity logo

Platforms

Delivery Trail will be developed for the PC market. It will include controller support as well, so in theory it could be ported to a console later. The game has Windows builds for both the x86 and ARM CPU-architectures, as well as Linux x86.

The target audience

This game is directed towards a wide range of players with different amounts of skill and gaming history. Because the game is built around the idea of “choose your own difficulty”, it can be played and enjoyed by many different kinds of gamers. The levels are short and sweet at a maximum of a couple of minutes. This makes the game easy to pick up and play without needing a large time or attention commitment. The player does not need to put full focus into the game and can instead enjoy trying to beat the records while listening to a podcast or watching a TV show.

The more casual crowd can have fun driving through the levels and progressing one level at a time. Meanwhile the more competitive gamers have the chance to clear all of the gold medals for the levels. The more speedrun centric people can also try to get the absolute best times they can and drive against the times of other people. This would be possible by creating an online leaderboard. That would be a nice feature but we are not sure if we will have time to develop it. More on this can be found in the Game Elements -section.

Game concepts

The game revolves around small, easy to start levels. The length of these levels are always under 2 minutes. You play as a delivery driver that has to get the delivery to the customer as fast as you can! The customer satisfaction is then based on the amount of time taken.

The game is easy to pick up and play because the basic concept is simple and does not require lots of brain processing to enjoy. This way the game can be played while enjoying other forms of media for a more casual and fun experience. The hardcore gamers can instead focus harder on trying to beat their times and on getting the best customer satisfaction.

The game will also include a scoreboard of the best times on the levels. When the player finishes the level, they will get their time on the board. This board would likely be filled with times gotten during the development. If there is enough time, it could also implement an online component.

How does it differ to other existing games?

The game differs form other games by being easier and faster to just pick up and play. It respects your time by not including anything unnecessary to the gameplay loop, like cutscenes. Perfect for the busy and multitasking player of today!

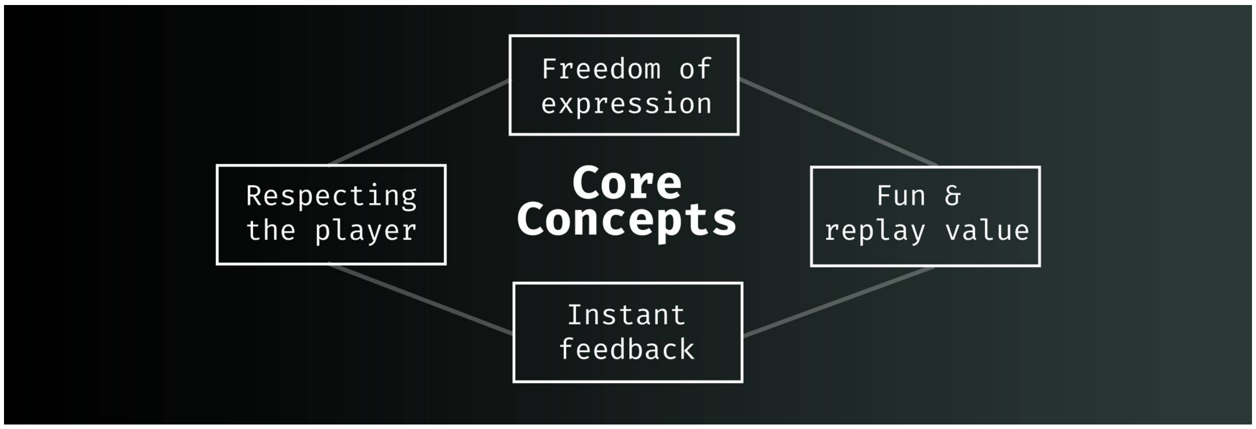


Figure 2: The core concepts of the game

On top of this, the gameplay loop is designed to be simple, while providing potential for it's own community of time trialing fans.

By keeping the theming fun and lighthearted, it allows for fun to be had while playing it without taking itself too seriously. This also has the chance of creating fun gameplay moments that have the potential to interest people to show it to their friends. This includes streamers and content creators for example.

Why is it better than the competitors?

The game is better than the competition because it respects the player and lets them create their own level of challenge. It does not expect the levels to be beaten in a singular way, providing possibilities for creativity and expression in the gameplay.

On top of this, it respects the player's time and does not require your entire attention to have fun and enjoy.

How do you plan to make money with it?

There are multiple possibilities for micro-transactions. By using multiple of these, the game could have a better chance at being financially profitable. The following ideas try not to disturb the gameplay and will **NOT** affect your individual level success.

1. Level progression help

The level progression could be tied to your customer satisfaction score from the previously played levels. This way the player would need to achieve a specified level of success before continuing to the next levels. The customer satisfaction could then be bribed via in-game micro-transactions that would let you progress without succeeding in the prior delivery levels.

2. Skins and cosmetics

Skins for the delivery car would provide the player with chance to stand out from other players. These skins could be purchased with the game progression currency (like the customer satisfaction score) or bought with in-game premium currency (like the bribes in the previous paragraph). This way they do not get in the way of the gameplay, but provide players with the possibility of expression and status. Examples for games that use the same kinds of tactics include the following:

- **Team Fortress 2 |** Hats, weapons, emotes
- **Fortnite |** Skins, emotes, music

- **Counter Strike 2 |** Skins, stickers, event passes
- **World of Warcraft |** Cosmetic items

3. In-game advertisements

Finally, the game levels could include big advertisement boards with purchasable advertisement spots. The start of the level could have these banners as well. This way the monetization would not interfere with the gameplay and the game world would stay feeling consistent and grounded in reality. You can see these ad spots and banners all over in motorsport like in the following reference pictures.

- Example 1: **WRC Rally |** Jyväskylä, Harju
- Example 2: **Formula 1 |** Monaco, Main straight
- Example 3: **Trackmania |** Formula E collaboration

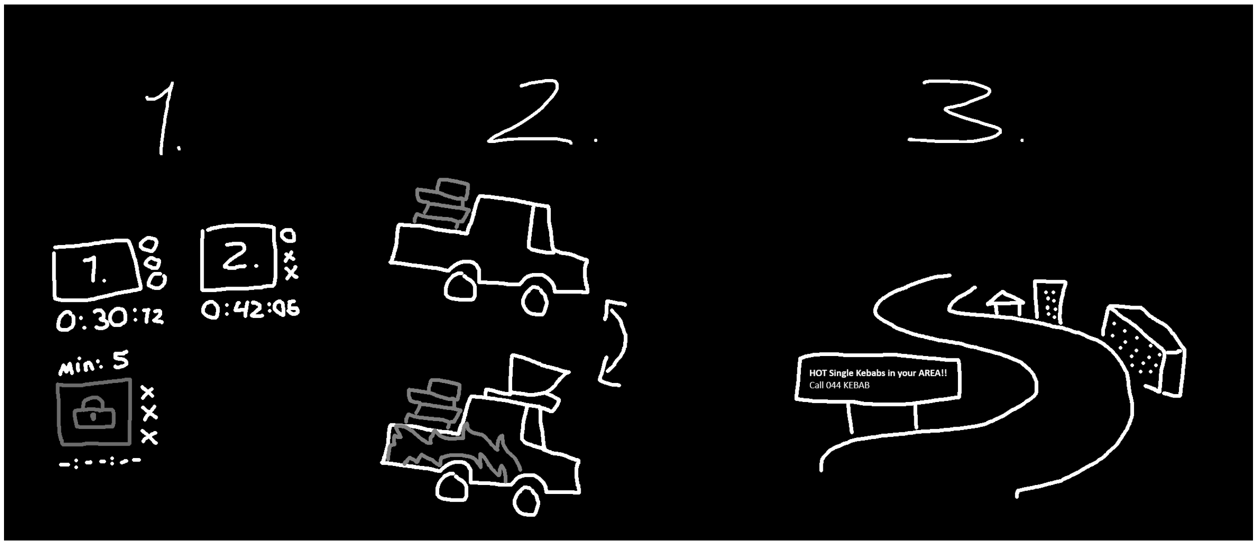


Figure 3: The monetization ideas

Executive summary (elevator speech)

The game is fun, fast paced and includes player creativity and freedom for expression. In today's entertainment climate, people have millions of things fighting for **100%** of their attention. We are not asking for **100%** of the attention, but around **50%** instead. This way the player can have the chance to catch up on the new seasons of their favorite show or listen to the new episode of the podcast their friend recommended to them.

By providing the player with creative expression through gameplay freedom, we will be able to hook the player's interest. The hooked player could even be then interested in improving and going for world records on singular levels in the future, thanks to the instant reward mechanism that the personal time improvement gives.

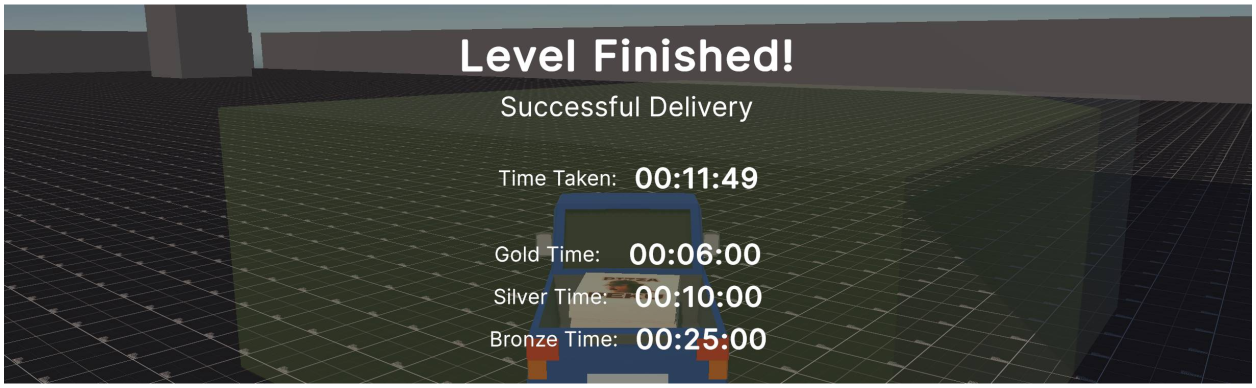
Planned lifespan

The lifespan of the game is very varied. Player's that are not interested can pick up the game, try it out and put it down if the gameplay loop does not interest them. Instead the players that love speedrunning, personal improvement and gameplay freedom that the game gives can be hooked on it for a long time to come. With the expectations that there will be other players to compete against in the leaderboards.

Gameplay

Game Elements

Time The main element of the game is time. This is used in each level as a player specified difficulty setting. This means that the game does not ask you to choose a difficulty setting straight out. Instead the the player chooses which medals will they try to go for. This way the playing field stays even for all levels of players while not affecting the experience of players with different skill levels. The next image shows an example of this. The end of the level shows all of the time requirements that there are, so that the player knows what to go for on their next run.



Level Finished!

Successful Delivery

Time Taken: 00:11:49

Gold Time: 00:06:00

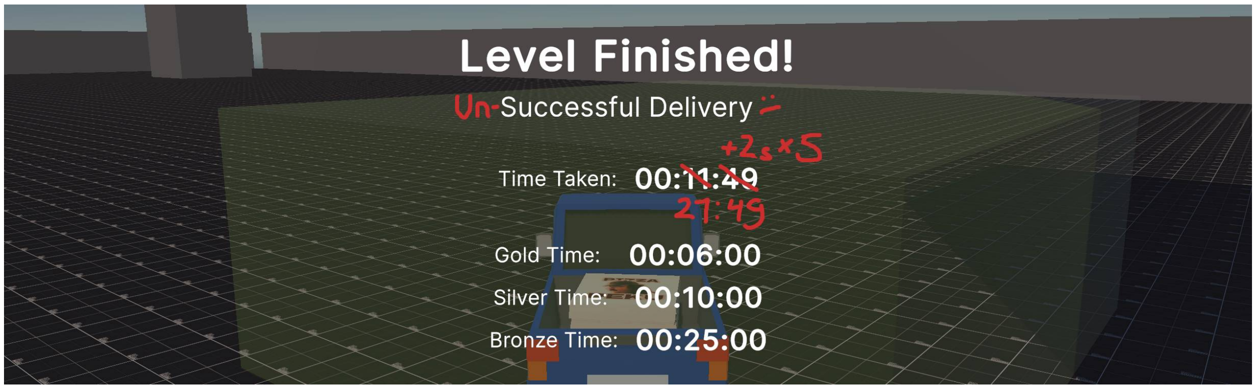
Silver Time: 00:10:00

Bronze Time: 00:25:00

Figure 4: Example of different rewards based on time taken to complete levels

This element could also include a secret highly difficult time that would be shown to the player once they get the gold time. This time could be the best time of the creators of the level for example.

Clean execution The other element planned for the game is for discouraging the player from hitting obstacles like walls or solid props. When the player would hit another obstacle, they would lose one of the pizza boxes from the top of their car. At the end of the level, the missing pizza boxes would count up to penalty time given.



Level Finished!

Un-Successful Delivery :-(

+2s ×5

Time Taken: 00:~~11:49~~

27:49

Gold Time: 00:06:00

Silver Time: 00:10:00

Bronze Time: 00:25:00

Figure 5: Example of a penalty given at the level result screen

These are the main elements of the gameplay that the game uses to rank your success and customer satisfaction.

Game Mechanics

Simplified explanation The car is programmed to use multiple different components that all handle their respective parts of the car. These components include the powertrain (motor and force transfer), suspension and steering. As you can see, these components are very closely aligned to their real world counterparts. This was possible with the use of real physics in the programming. The physics and formulas used will be gone through in the other headings of the Game Mechanics part.

We wanted the parts of the car to be split to their own parts, so that we could modify the code within them more easily. It also makes the editor experience of working with the car simpler and more organized.

The main goal with the feel of the car is to have it be fun to drive. The physics are using real formulas, but the values are all adjustable, so that they can be made more arcade-y. Because the most important part is that the game is fun to play. To achieve this, the steering includes a steering angle curve that is tied to the speed of the car. This way if you are driving faster you steer less. This makes the car especially more controllable on keyboard and mouse. This is possible because, just like in the real world, the faster you drive the more change your small impacts have. This same feature is also available on many real world cars of the last 5 years.

The car uses the brake pedal for reversing. This means that the player does not need to know how to change gears to go back. This was quite a bit harder to code than the alternative, but the end result is more intuitive and straight forward.

Lastly, the game includes a respawn button. This way the player can easily revert back to an older state of the level if they are stuck. The respawn mechanic uses the game's checkpoint system to revert the player's car's rigidbody to a prior state. This state is collected when the player goes through a checkpoint. Because the checkpoint system saves the player's rigidbody values, the player will be able to respawn with the same speed, direction and rpm as when they hit the checkpoint. This makes it easier to design levels, since you do not need to make sure that the player is able to continue from the checkpoint if they have respawned.

Car suspension The suspension of the car is programmed by our team. It uses raycast to count the amount of force that needs to be given to the car to keep it off the ground. The properties of it can be changed. It includes the following editable properties:

- Spring Stiffness
- Rest Length
- Spring Travel
- Damper Stiffness (zeta value)

The suspension is comprised of 4 dampened springs. These springs use basic physics calculation for calculating the force upwards to keep the car off the ground. All 4 raycast suspensions have their own calculations and work independently from one another. For calculating the dampened spring force, we use the following formula:

```
SpringForce = Offset * Stiffness
DampeningForce = SpringVelocity * DamperStiffness
Force = SpringForce - DampeningForce
```

For getting the damper stiffness, we calculate it using the Zeta value given.

```
DamperStiffness = (2 * sqrt(SpringStiffness * CarMass)) * Zeta
```

The component can be found in the following link and it includes code comments with formulas and extra info

Suspension.cs -script

(LibreTexts, n.d.) (AshDev, 2024) (Toyful Games, 2022)

Powertrain The powertrain of the car is in its own component/script. It includes the following editable parameters:

- Drivetrain (FWD, RWD or AWD)
- Minimum RPM
- Maximum RPM
- Peak Horse Power
- Power Curve
- Engine Braking Multiplier

The drivetrain is used to choose which tires the forward force should be given to. In fwd the drive divided by 2 and is only given to the front wheels and in awd the force is divided by 4 and distributed evenly between all 4 tires.

The forward force is calculated using the rpm and the power curve. The power curve is a built in unity element called Animation Curve. This way we can graphically add a formula that is used to calculate the horsepower of the car at a given rpm value. Animation Curve makes this very easy, since we don't need to come up with our own mathematic formulas and just do it visually. (Game Dev Guide, 2021)

The formula for calculating the force is the following:

```
HorsePower = PowerCurve.Evaluate(rpm / maxRpm);
Torque = (HorsePower * 5252) / rpm;
TorquePerTire = 2; // or 4 if AWD is on
// for each tire
Force = TorquePerTire / tire.radius;
```

The engine braking is then calculated by the following formula. But only if the brake or gas is not pressed

```
Force = CarMass * EngineBrakingMultiplier;
Deceleration = Force * (PowerCurve.Evaluate(rpm / maxRpm) * 3);
```

The component can be found in the following link and it includes code comments with formulas and extra info

Powertrain.cs -script

Steering The steering component is used for rotating the front tires as well as handling the sideways traction of the tires. This way the car does not slide sideways erratically. It had the following editable parameters.

- Steering Curve
- Max Steering Angle
- Sideways Grip Multiplier

The steering curve is also an Animation Curve that is used to calculate the current maximum amount that the front tires can be rotated sideways. This is used so that in higher speeds, the steering does not turn as fast as in low speed. This is done so that the car will stay more controllable on keyboard. Since the keyboard does not give analogue values. Without this calculation, the car would feel weird to drive. (Game Dev Guide, 2021)

This addition was inspired by modern cars with this same functionality as well as Crazy Taxi. We did not want our car to control better than in that game, where the controls were erratic to say the least.

The sideways grip is calculated per tire using the simple formula of:

```
TireGrip = tire.GripCurve.Evaluate(CurrentSpeed);
desiredVelocityChange = (ChangeInVelocity / Time) * TireGrip;
```

The component can be found in the following link and it includes code comments with formulas and extra info

Steering.cs -script

(Toyful Games, 2022)

Braking The braking component uses the following editable parameters:

- Break Force
- Break Balance

The break force is the force that the brakes give in newtons per square meter (N/mˆ2). This is used to calculate the desired deceleration of the car when the brake is pressed.

The brake balance is used to split the brake force between the front and rear axles. It can also be referred to as front brake bias, since the value depicts how much of the brake force should be put to the front tires.

The formula for calculating the deceleration is the following:

```
Deceleration = BreakForce / CarMass;
FrontBrakeForce = Deceleration * BreakBalance;
RearBrakeForce = BreakForce - FrontBrakeForce;
```

The braking component also handles the reverse gear, which is activated when the car is still and the player presses the brake button down.

The component can be found in the following link and it includes code comments with formulas and extra info

Braking.cs -script

Respawning When the player presses the respawn action on their controlscheme of choice, they will be teleported back to the latest checkpoint. If no checkpoints are present or have been reached, the player respawns at the spawn point of the game. If this respawn position is the start of the game, the level timer also starts over. This way you do not need to go to the pause menu to restart every time.

The underlying system that makes respawning possible is the checkpoint system. This system is part of the `GameManager` singleton class, that is present in all of the level scenes. The system is a simple dynamic list of `Checkpoint` class instances. This checkpoint class includes the following values inside.

```
    public float rpm { get; private set; }
    // all the information about the rigidbody at the time of the checkpoint
    public Vector3 position { get; private set; }
    public Quaternion rotation { get; private set; }
    public Vector3 linearVelocity { get; private set; }
    public Vector3 angularVelocity { get; private set; }
    public Vector3 inertiaTensor { get; private set; }
    public Quaternion inertiaTensorRotation { get; private set; }
    public float linearDamping { get; private set; }
    public float angularDamping { get; private set; }
```

This is a lot, but it is needed, since the rigidbody component has a lot of moving parts. When the player hits the checkpoint all of these values are extracted form the rigidbody and stored inside the `Checkpoint` instance. This way, it is as easy as applying these to the player's rigidbody when they respawn.

This was not as simple in practice, but the end result we got worked well enough. It uses lots of waiting for the physics engine to synchronize. Without these physics frame waits, the car would go crazy. The applying of these forces is done twice in a row. We do not know why it is needed twice, but for some reason, it stabilizes the respawning system nicely.

For more information on the system, please take a closer look at the following code files. They include lots of documentation and extra info on the inner workings of this system.

Assets/Code/Level/Checkpoint.cs, Assets/Code/Level/CheckpointController.cs, Assets/Code/Vehicle/CarContro

User interface

The game's art style is low poly. Which means that the assets' topology is very minimal. The user interface is more modern, which gives the game a nice contrast of classic and modern.

The levels are a mix of urban and more rural areas. In the future the levels could get more and more absurd and more unrealistic and wacky. The story would mix with the gradual incrementing of the absurdness.

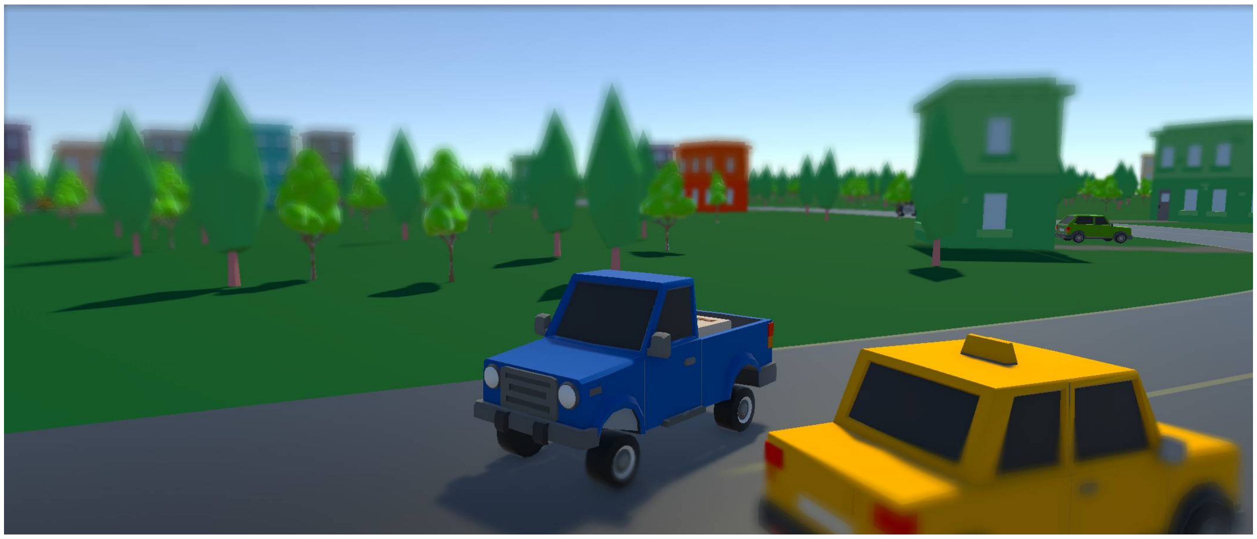


Figure 6: Screenshot of the game

Menus

- Title screen / Main menu
- Level selection menu
- Level finish menu

Controls

The game includes the following main actions:

- Acceleration
- Braking
- Steering
- Camera Movement

These controls are added with Unity's Input Manager toolkit. This allows us to add controls to them without touching the code again. This makes it very easy to add controller support alongside mouse and keyboard. We can also add multiple mappings to a single action, so that it is easy to support for example W,A,S,D and the Arrow Keys. (Unity, n.d-a)

Controller bindings

- **Acceleration**: Right trigger
- **Braking**: Left trigger
- **Steering**: Left analogue stick
- **Camera Movement**: Right analogue stick
- **Respawn**: Top button
- **Pause**: Start button

Keyboard and mouse bindings

- **Acceleration**: W | Up-arrow | Left mouse button
- **Braking**: S | Down-arrow | Right mouse button
- **Steering**: A (left) D (right) | Left and right arrow keys
- **Camera Movement**: Mouse movement
- **Respawn**: R
- **Pause**: Escape key

Error handling

In our game code, we use Assertion-esque checks that help identify errors and their fixes if they come up. This was inspired by the TigerStyle programming of TigerBeetle. It also lends itself nicely for automated testing in the future. Try-Catch is used to the best of our abilities where it is possible, but due to the extremely little amount of time for this project, error handling is not the highest priority. Instead the project is prioritizing documentation over it. (TigerBeetle, 2024)

Story

The main character of the game is you, the player controlling the car. The game also includes characters like the boss of the fast food establishment. The customers could also have dialogue that they say during the deliveries, but that is a lower priority.

The characters you deliver the food to are extremely nit picky and whiny. They have different personality types. These could include the following:

- **The micro-manager**: Keeps telling the player how he would do it. *"You should take this shortcut here, it's way faster!"*.
- **The old man**: An old person that thinks all young people are lazy. *"Back in my day this pizza would be coming out of me by now!"*.
- **The picky one**: A person that does not know how to choose and keeps correcting their wants. *"I reeeally want pineapple on my pizza ... Hey, remember the pineapple, can you take it off actually? Thank youuuu"*.

Another idea for the story is that your boss is constantly asking more and more of you. Sending you to weirder and weirder places. This way the game could end up at very abstract situations that are very unrealistic but fun.

As established in the previous sections, the game does not prioritize the story. It is likely that due to the incredibly small development time, that there is no story at all. Other than deliver the pizzas and make the customers happy.

Game progression

The game progresses one level at time. The later levels could be locked behind a specified customer satisfaction threshold. These levels can be played non-linearly otherwise.

Development plan and scheduling of the process

Priorities

High priorities

- Create 3 levels
- Create a title screen and a level selector menu
- Make the car feel good to control
- Add sound effects to the game
- Timing system that shows your improvement to prior times on the level

Mid priorities

- Penalty system for collisions with obstacles
- Replay system that would allow to see how the level went
- 1 or 2 extra levels
- Online leaderboards for levels

Low priorities

- Story and character dialogue
- Level progression thresholds
- More through error handling

Updates

The game will likely not receive updates after the game development module. However, if there is enough interest, then more levels could be easily added to the game post-launch. With other content like micro-transactions and whatnot.

References and extra information

References

Ash Dev. 8.3.2024. *Arcade Car Controller Part 1 | Suspension tutorial Unity | AshDev*. https://www.youtube.com/watch?v=sWshRRDxdSU

Game Dev Guide. 29.10.2021. *Why Animation Curves In Unity Are So Useful*. https://www.youtube.com/watch?v=Nc9x0LfvJhI

LibreTexts. n.d. *15.6 Damped Oscillations*. https://phys.libretexts.org/Bookshelves/University_Physics/Univers
Mechanics_Sound_Oscillations_and_Waves(OpenStax)/15%3A_Oscillations/15.06%3A_Damped_Oscillations)

Unity. n.d-a. *Input Manager*. https://docs.unity3d.com/Manual/class-InputManager.html

Batiati. R., Kladov. A., Schwartz. E. & Swayne. L. 5.11.2024. *Tiger Style*. https://github.com/tigerbeetle/tigerbeetle/blob/main/docs/TIGER_STYLE.md

Toyful Games. 9.7.2022. *Making Custom Car Physics in Unity (for Very Very Valet)* https://www.youtube.com/watch?v=CdPYlj5uZeI

Links

- Trackmania trackmania.com
- Crazy taxi Crazy Taxi - Wikipedia