Interim Report:

Aquario Kart: Double Splash!!

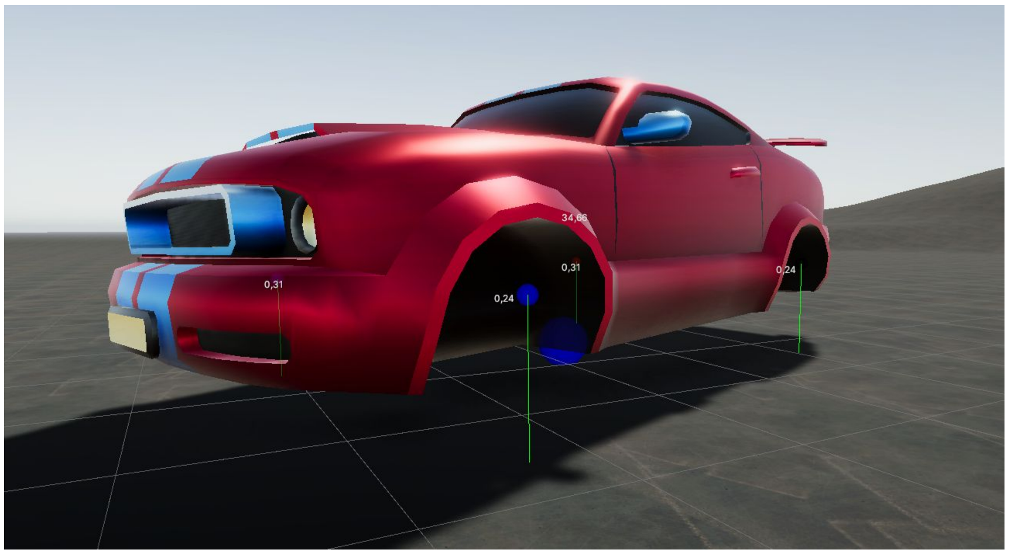
December 2020

Team Markedly Minified Olives (MMO)
Mark Pilgram
Min-Shan Luong
Oliver Jung

Task Progress

Vehicle

Our vehicle handling model uses Ray-cast based springs pushing our vehicle upwards in place of actual wheels. This is to say that our cars don't actually touch the ground under normal driving conditions, they are more like hover cars with visual wheels dangling to touch the ground. This approach gives us a lot of control in how our vehicle handles under various scenarios, while still providing a highly-physics driven driving sensation.

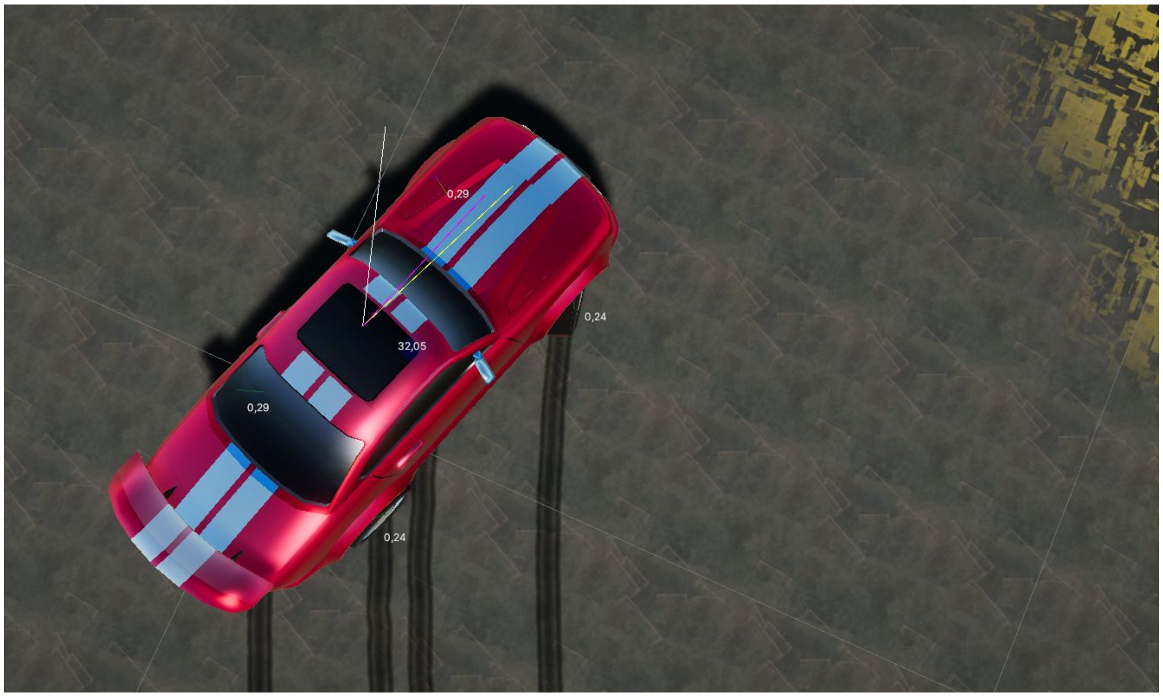


Debug view of the springs holding the vehicle above the ground with the (visual) wheels disabled. The values next to the wheel raycasts display the compression ratio of the associated spring.

Essentially the vehicle handling model is split into two halves: One for regular driving, the other for drifting.

During regular driving, the wheels try to behave in a way you'd expect a wheel to behave: The wheel allows (most) forward movement, while trying to stop sideways movement. Rotating a wheel thereby influences the direction in which force exerted to counter sideways movement. On the other hand, acceleration and brake forces resulting from player inputs are applied in the forward and backward direction respectively. We have several variables that control the strength of all these forces as well as how far a steering wheel can turn, some of which are dependent on the vehicle's current velocity.

When drifting, the influence of individual wheels on the handling is greatly reduced, essentially just the spring forces keeping the vehicle above the ground remain. While drifting, the vehicle essentially tries to rotate itself so that it's at an angle towards its current movement vector. The more steering input the player provides, the greater an angle the vehicle tries to achieve. While in this mode, Velocity is converted from the current direction of movement towards the direction the vehicle is pointing in, with the amount converted depending on the angle of the vehicle to its velocity vector.



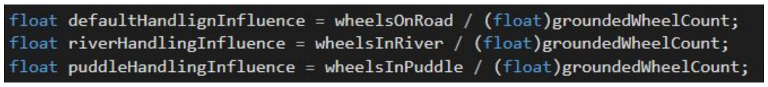
Drift Debug View. The white line displays the current direction of movement, the yellow line the drift target angle (determined by player input and the vehicle's drift parameters) and the pink line the vehicle's current forward vector.

Drift mode is entered by the player tapping the brakes while making a turn at sufficient velocity, or by landing at an angle with sufficient velocity. To exit drift mode the car needs to match its forward vector with the direction of the velocity vector within a certain window, while keeping its angular velocity sufficiently low. This allows the player to choose between changing drift direction from left to right by switching their steering input direction or exit the drift by steering straight ahead. The vehicle will also automatically exit the drift if certain parameters are no longer met, for example when the vehicle is too slow. By drifting, a player can make turns while losing less speed than they would making the same turn in the regular driving mode. Drifting also significantly reduces the negative impacts shallow water sources (more on those later) have on vehicle handling.

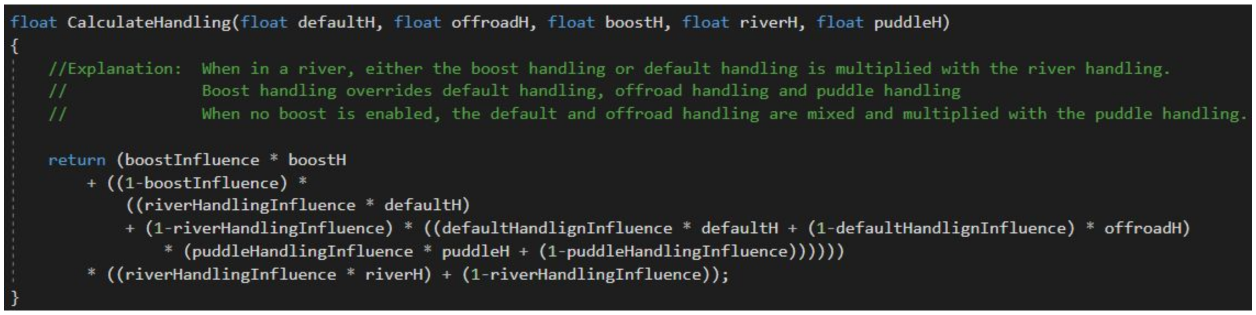
Certain handling parameters of the vehicle can be influenced by different terrain types. Essentially the vehicle has a set of handling modifiers for various conditions, namely

- Regular driving
- Driving outside of the race course
- Driving while a speed boost is active
- Driving through a large body of water (e.g., a river)
- Driving over a shallow wet surface (e.g., a puddle)

Some of these conditions are dependent on global states (e.g., the vehicle has a speed boost active), others on states at individual wheels (e.g., a wheel is on/off-road and/or in a puddle). The state of the vehicle is however handled on a global level, meaning that all wheels operate under the same parameters. These global parameters are constructed by a complex series of interpolations of both local and global states. The reason for this approach is that we want our game to still have a relatively arcade-y feel that is easy to pick up, and having individual wheels behave differently could lead to adverse effects.



```
float defaultHandlignInfluence = wheelsOnRoad / (float)groundedWheelCount;
float riverHandlingInfluence = wheelsInRiver / (float)groundedWheelCount;
float puddleHandlingInfluence = wheelsInPuddle / (float)groundedWheelCount;
```



```
float CalculateHandling(float defaultH, float offroadH, float boostH, float riverH, float puddleH)
{
    //Explanation:  When in a river, either the boost handling or default handling is multiplied with the river handling.
    //              Boost handling overrides default handling, offroad handling and puddle handling
    //              When no boost is enabled, the default and offroad handling are mixed and multiplied with the puddle handling.

    return (boostInfluence * boostH
        + ((1-boostInfluence) *
            ((riverHandlingInfluence * defaultH)
            + (1-riverHandlingInfluence) * ((defaultHandlignInfluence * defaultH + (1-defaultHandlignInfluence) * offroadH)
                * (puddleHandlingInfluence * puddleH + (1-puddleHandlingInfluence))))))
        * ((riverHandlingInfluence * riverH) + (1-riverHandlingInfluence));
}
```

Interpolation is fun. Enjoy figuring this one out. Or just read the comment above the calculation…

Visual aspects of the vehicle (wheel positioning and rotation) are handled independently of the physical aspect. When on the ground, wheels currently rotate with the speed of their forward movement, meaning that we don't yet simulate wheels losing traction. For aerial rotational velocity, a system had to be put in place that simulates momentum, acceleration and brake forces being applied to the wheel in the air.

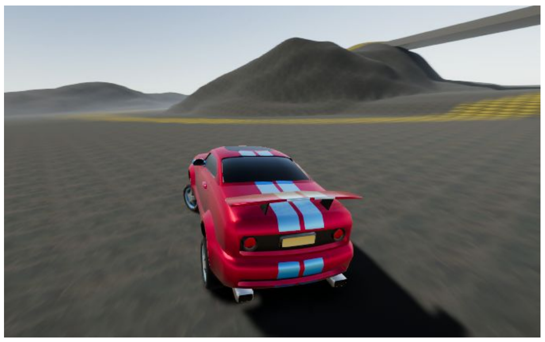
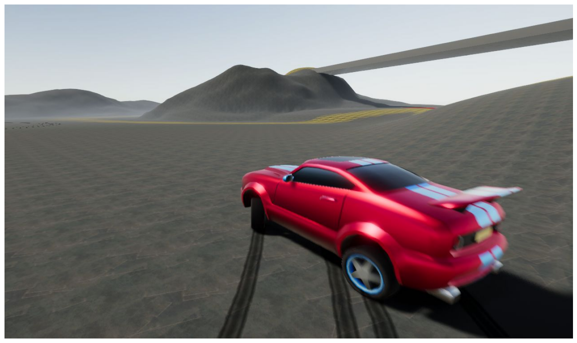
Camera

The camera is another important aspect influencing the driving sensation in our game. Camera motion is vital to communicating information about the vehicle's state. Currently we offer three camera views.

The primary view is a third-person chase camera that follows the vehicle. This camera was designed around the principle that there are two "rings" surrounding the vehicle, one of them closer, the other further away. These rings prevent the camera from intersecting the car, instead the camera circles around the car when the direction of movement is changed. The camera will always try to point in the car's direction of movement. As the car gains speed, the camera transitions from the inner towards the outer ring, shifting the focus from the car to the environment ahead. The rotation of the rings is influenced by either the ground normal or the direction of movement when the car is in the air, a vector which smoothly transitions over time for an equally smooth experience.

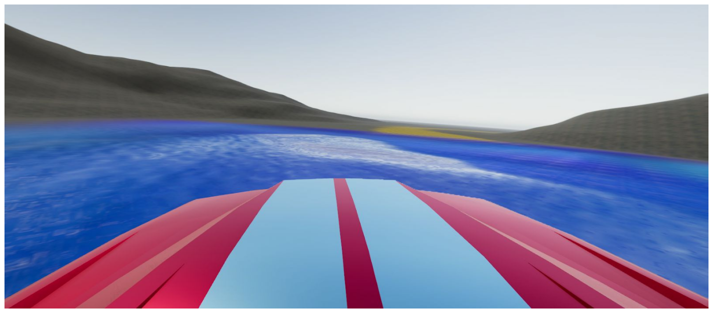
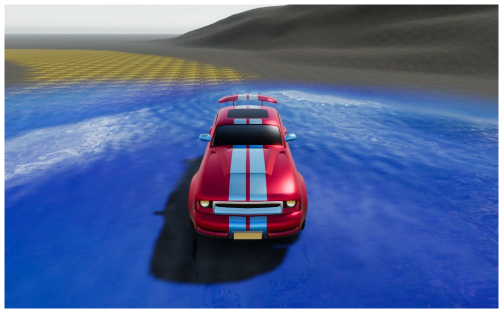
For additional smoothing, the ground normal is constructed using the (up to four) wheel-spring raycasts of the highlighted vehicle, granting us an additional measure to prevent imperfections on the ground causing unexpectedly large camera movements. The camera can also roll slightly to highlight when the vehicle is driving along a slope. Great care was put into the camera following the vehicle's movements, while not rolling over to be below the vehicle should it land on its roof.

The third-person chase camera also plays a vital role in communicating the car's drift state. As can be seen below, a clear distinction is made between a regular turn and a drift turn.

The second camera view is a bonnet camera. For actual driving it's probably less useful than the third-person camera, but its "slightly loose gopro camera" design greatly highlights changes in the vehicle's movement, though it is harder to distinguish between regular turns and drifts from this view. Switching between these two views is a seamless experience, with the camera even lifting above the vehicle's interior to avoid potential clipping issues.

Finally we have a rear-view camera. Unlike the chase-camera, this camera provides a fairly static view of what's behind the player's vehicle, regardless of whether the player is driving forward or backward. This camera view exists in parallel to the other two, meaning that its position and rotation is updated simultaneously with whichever of the first two views is active, to allow an instant transition between the regular driving view and the rear view while maintaining continuity - If the camera switches from the chase camera to the rear view camera and back while in a drift, the chase camera can continue to apply all its smoothing operations to the ground normal vector and drift offset while its view is inactive.

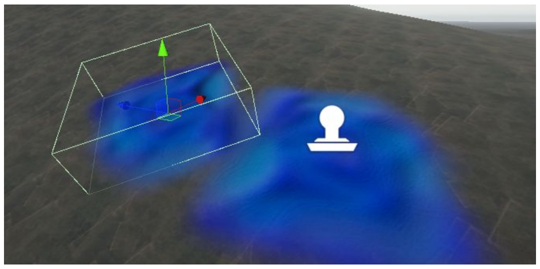



Bonnet camera (left) and rear view camera (right)

Water System

As mentioned earlier, vehicle handling can be influenced by a variety of factors, including small and large bodies of water Each wheel-spring has a collider that can interact with the colliders of water sources, letting the spring know about its current water-state. Interactions with water sources are passed on to the vehicle's water tank, which fills when at least one wheel is interacting with some kind of water.

Currently the water tank only empties when the vehicle tries to accelerate on ground, though we do of course want to add items that consume water later on. The fill-level of the water tank increases the vehicle's top speed.



Water sources can be either static or dynamic. Static water sources (rivers or puddles) remain present no matter what, whereas dynamic ones (puddles only) gradually disappear as vehicles collect water from them. These dynamic puddles are displayed using gradually shrinking decal projectors aligned with the ground.

Vehicle AI

Our AI driven vehicles use the same vehicle handling model as the player vehicles. The main goal of an AI vehicle is to reach each of the predefined checkpoints in a given order (more information on the predefined checkpoints below in section "Checkpoints").

However, there are obstacles located all over the race track that would hinder our AI vehicles to directly drive to the given target location. In order to detect and avoid these obstacles, we equipped our AI vehicle with sensors (raycasts directly and diagonally to the front) that influence the vehicle's steering and driving speed. At the moment, the obstacle avoidance system has three times more influence on steering behaviour than the checkpoints to successfully bypass obstacles in most cases.

Nevertheless, there are some cases where our AI vehicle inevitably crashes into obstacles e.g. due to high driving speed. In this case, if the vehicle is stuck for a certain amount of time in front of an obstacle, it reverses until its sensors don't detect anything anymore and thus should be able to bypass the obstacle afterwards.

Just as the player vehicle, our AI vehicle is capable of drifting. An AI vehicle enters a drift whenever the angle between its front vector and its vector to the next checkpoint is too large. If not already interrupted, the AI vehicle drifts until it is properly oriented (=small angle) towards the next checkpoint.

Checkpoints

In order to follow a given path our AI uses a set of checkpoints distributed along the path. These checkpoints can be arranged as a curve or as a loop and represent the rough shape of the path. Each of the checkpoints knows its neighbours and guides the AI to the next coordinate by providing information about the next checkpoint and upcoming points of interest (e.g. water sources or boost pads). In return, the AI informs the checkpoint system about newly created dynamic puddles.

Furthermore, the checkpoints enable tracking the progress of each car while checking their local progress between the current and the next point. For simplicity, the overall length of the track is calculated by the sum of the single distances between the connected checkpoints. The local progress is measured by projecting the player's position (relative to the last checkpoint) onto the connecting vector from the last to the next point and comparing the magnitudes of the resulting vector and the connecting vector.

Scenery

In keeping with the water-theme our racing track is located on an island. For creating the basic landscape we used Unity's terrain system whereas the roads are created with the help of a customized version of the RoadArchitect-asset from the Unity asset store. In order to add more detail to the environment we are using free assets from the Unity asset store including 3D models, character animation, and textures. In general, we aim for a cartoony art style in our game.

Challenges

High vertical-velocity upright landings

One problem with the current handling model is that our vehicle can come to a rather abrupt stop if it lands on the ground too hard. This is probably due to its collider intersecting the ground too much from one time-step to the next for even a low-friction physics material to save it. Unfortunately continuous collision detection is not an option in this case, as it leads to many unintended collisions during regular driving. While the vehicle's collision can be calculated continuously, the springs holding it off the ground can not.

Increasing the fixed update rate seems to have unintended consequences on the vehicle's handling, despite forces being applied in a fixed-delta time dependent manner, so that approach would require the handling parameters of the vehicle to be re-adjusted significantly, while also increasing the CPU's workload.

As long as our track(s) don't feature too hard falls this shouldn't be too much of a problem, but some other solutions might be to try to slow the vehicle down just before it hits the ground, though this would have to be done in a way that's not too obvious to the player.

Camera Jitter

As our vehicles are rigidbodies, their movement isn't continuous between fixed updates. Of course, rigidbody interpolation is an option that we are using, however not all aspects of the vehicle are necessarily interpolated by this between frames, which can be a problem when the framerate of the game is higher than the update rate of the physics engine (as is generally the case). When designing the third-person chase camera in particular, a lot of effort needed to be put into using and smoothing the right values in the right places to achieve perfectly smooth and jitter free motion for both the camera and the vehicle in its view.

Vehicle AI Obstacle Avoidance

One of the most challenging parts of creating a driver AI is to balance the strength of steering, braking, and acceleration. To achieve satisfying results we probably need to take various parameters into account as for example the sensor distance to obstacles, the vehicle's current velocity, the tangent direction at a sensor's hit point etc.

Design Revisions

NONE!