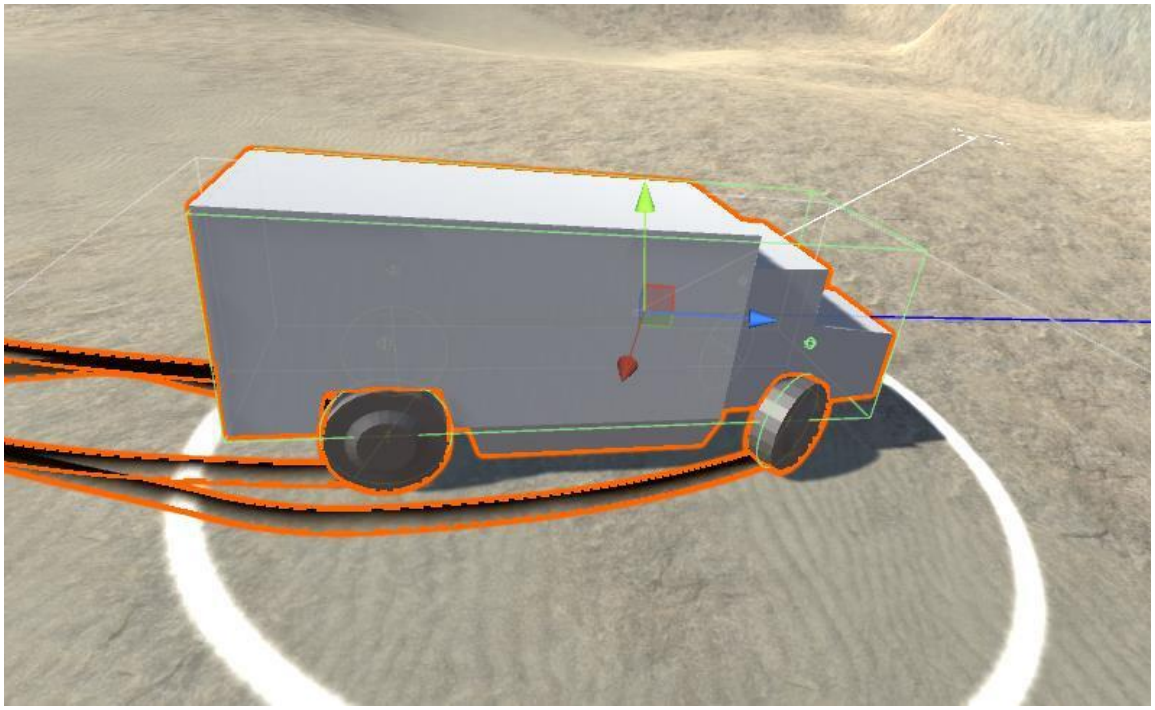
RTS Wheeled Vehicle Controller

--- SINCRESS ---

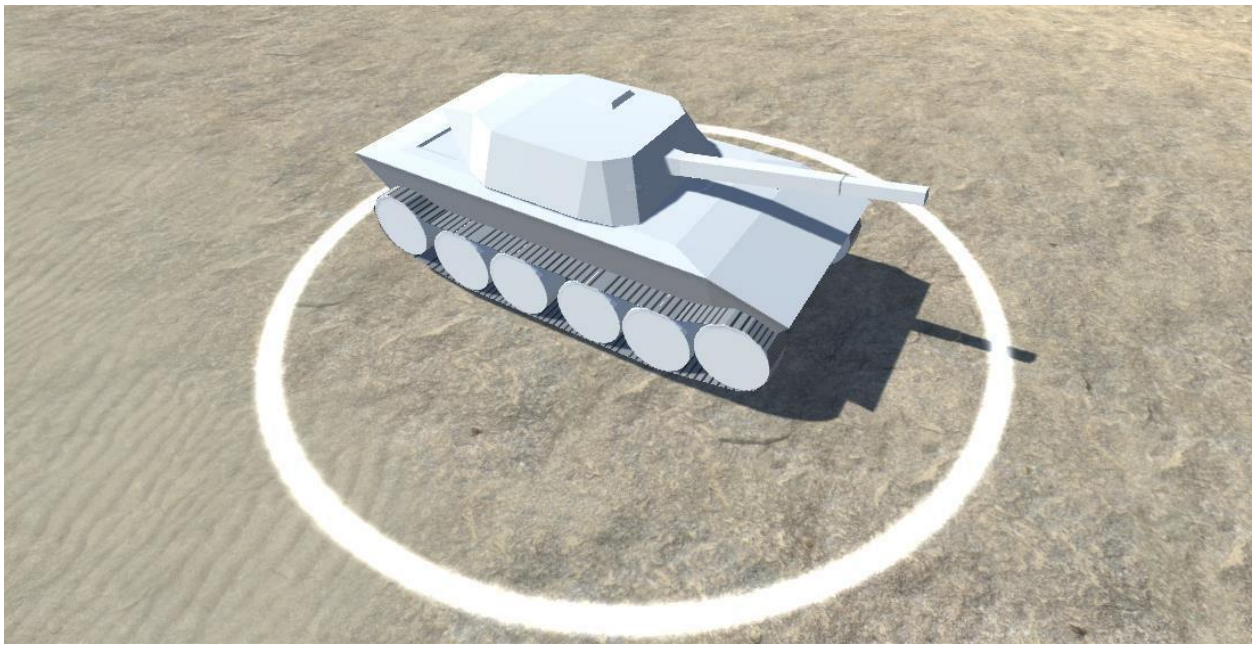
Realistic wheeled vehicle behavior for RTS games

Welcome to the documentation of the RTS Wheeled Vehicle Controller package for Unity3D. This document describes the example scene, the functionality and the scripts of the provided. To find a description of a script, just search its name in this document. Thank you for selecting the *RTS Wheeled Vehicle Controller* for your project, I hope you enjoy using this package and integrating it into your project.



1. Overview

This framework is designed to give your strategy game project realistic behavior for wheeled vehicles. Included in the project is a mockup vehicle which demonstrates the minimal example of implementation. The vehicle relies on Unity's *WheelColliders* to facilitate movement and has **two scripts** which control its movement. There is another script attached to the Main Camera which performs unit selection by means of mouse clicking or dragging. These vehicles do not need a **NavMesh** to move, however, a NavMesh agent is used for pathfinding. A Pathfinder prefab is included to provided to facilitate the navigation.



2. Example Scene

The example scene is in the root of the project folder. It showcases the features of this package in a minimal example. The *CameraController* script is attached to the **Main Camera** object, controlling the camera movement and clamping it to stay above the terrain. The **Terrain** object is a mesh for the vehicles to drive on, and can be replaced with any mesh which has a collider.

Each vehicle has a *VehicleController* and a *VehicleInput* script attached to it. The ***VehicleController*** script contains vehicle properties (acceleration, speed, torque, braking,

etc.) and methods which add throttle and steering input to the vehicle. Vehicles with wheels such as trucks, cars and APCs use the *WheeledVehicleController* class and tracked vehicles (tanks) use the *TrackedVehicleController*. The ***VehicleInput*** script handles input to the vehicle in two forms: manual keyboard input (W,A,S,D to drive the vehicle) and automated input which moves the vehicle to a position on the terrain. To tweak vehicle settings refer to the *VehicleController* script on the vehicle GameObject and the WheelColliders on its wheels. There is a rigidbody component attached to the vehicle which enables it to move using the Wheel Colliders. In its sub-hierarchy the vehicle has four WheelColliders and physical wheel models and a *Projector* which shows the player whether the unit is currently selected for input. The *Projector* projects a circular texture on the terrain below the unit when it has been selected. The image below shows the Example Scene hierarchy and the two main scripts attached to a vehicle GameObject.

PATHFINDING NOTICE:

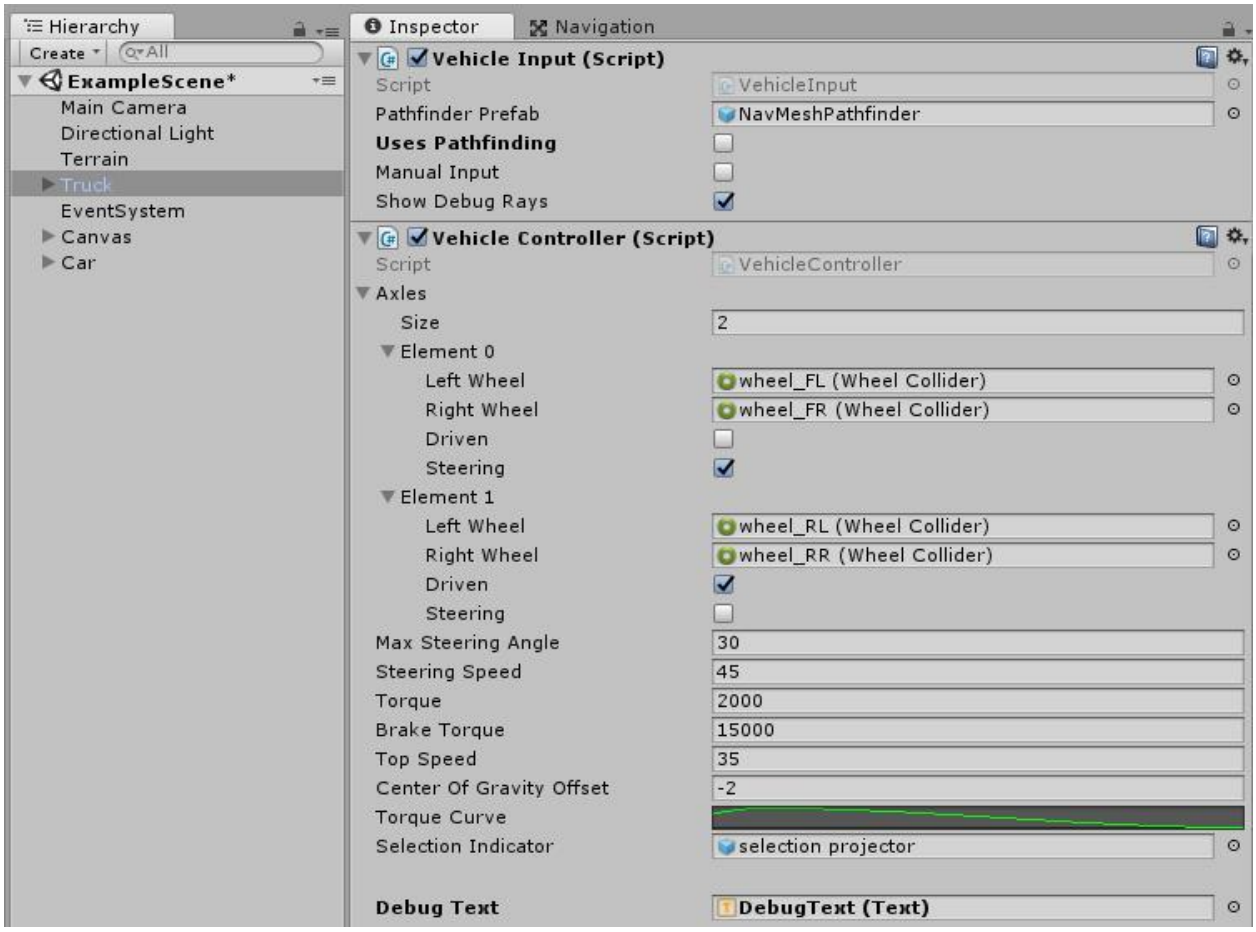
The pathfinding system in RTS Vehicle Controller is currently in development and therefore may not be as refined as expected. The foundations are here for you to build on, however if you do not like the navigation system you can turn it off using the **UsesPathfinding** boolean on the *VehicleInput* script. Future releases of this asset will feature improved pathfinding.

3. Scripts

The following section describes scripts which are relevant to the package and explains how individual settings affect vehicle behavior.

SelectUnits.cs

Controls selection of units by means of mouse click and drag. When the user clicks on the game screen, a raycast is performed from the camera and the clicked objects is checked. The script keeps track of player's units in its *selectableUnits* array and notifies them when a destination is given to selected units by right clicking on the terrain. This script



also contains GUI drawing functionality used to draw the selection rectangle which appears when the left mouse button is held and dragged on the screen. Additionally, when the player right clicks on the terrain, a **Destination Prefab** is spawned on the position. It is a projector with a texture which denotes that a destination is given to selected units and will be destroyed after one second.

CameraMovement.cs

Based on given movement speed and zoom increments, *CameraMovement* controls the basic viewport movement on the terrain. A terrain size is given and its size is used to clamp the camera movement to within the boundaries of the terrain. The camera's height above the ground can also be changed by scrolling the mousewheel.

VehicleController.cs

Abstract class which contains vehicle properties and methods used to process steering and throttle input. It is implemented by *WheeledVehicleController* and *TrackedVehicleController* classes.

Fields:

`public List<Axle> **Axles** - Each axle has a right and left wheel and a property which tells if it's a drive axle and if steering input affects it`

`public float **Wheelbase** – Distance between the first and the second axle, used to compute the turn radius`

`public float **TurnRadius** – Calculated as wheelbase/Sin(MaxSteeringAngle)`

`public float **MaxSteeringAngle** – Max angle of wheels in degrees`

`public float **SteeringSpeed** – Steering speed in degrees per second (for manual mode) public float **Torque** – Engine torque applied at drive wheels`

`public float **BrakeTorque** – Brake torque applied at all wheels public int **TopSpeed** – Maximum theoretical speed`

`public AnimationCurve **TorqueCurve** – The curve which describes the relation between torque and speed (usually torque decreases with speed)`

`public GameObject **SelectionIndicator** – Reference to the selection texture projector public bool **IsSelected** – If selected, vehicle receives input and movement orders public Text **DebugText** – UI Text used to log details`

private float **steeringAngle** – Current steering angle in degrees private float **speed** – Current vehicle speed

private Vector3 **lastPos** – Vehicle position in the last frame, used to calculate speed

private Vector3 **currentDestination** – The current given destination

private bool **destinationGiven** – Whether the current destination is actual and should be driven towards

TrackedVehicleController specific properties:

public MeshRenderer TrackLeft, TrackRight – References to the MeshRenderers of the left and the right track

public Material TrackMaterial – Reference to the material which is instantiated and applied to the track MeshRenderers. The X offset of the material is changed when the vehicle moves to simulate track movement

private float currentMaxSteeringAngle, currentSteeringSpeed – The current values of steering and throttle input used for smooth transitions (to avoid vehicle "drifting" and make it feel more realistic)

Methods:

public void **SetThrottle**(float throttle) - Sets the throttle input to the vehicle, clamped to range of [-1,1]

public void **SetSteeringAngle**(float input) - Sets the steering angle based on input in range of [-1,1], then clamped to fit the MaxSteeringAngle. Called by FixedUpdate() in VehicleInput.cs

public void **SetDirectSteeringAngle**(float angle) - Sets the actual wheel steering angle in degrees. Called by FixedUpdate() in VehicleInput.cs. Used by automatic throttle input.

public void **SetSelected**(bool value) - Toggles the vehicle selection flag. If selected, the vehicle will project a circle around it.

public void **SetHandbrake**(bool isOn) - Applies brake torque to all wheels

private void **ApplyLocalPositionToVisuals**(WheelCollider collider) - Applies the wheel collider rotation to the physical models of wheels.

protected void **DriveForwardToTarget**(float angle, float distance) – When the destination is ahead or far away, drive forward while steering towards it

`protected void` **`ReverseToTarget`**`(float angle, float distance) – When the destination is (almost) directly behind the vehicle and very close, simply reverse into destination`

`protected void` **`ReverseUntilOutsideRadius`**`(float angle, float distance) – When the destination is close to the vehicle (inside its turn radius) and not directly ahead, reverse into a position from which it will be possible to drive forward to the destination`

`public void` **`UpdateVehicle`**`() – Updates the vehicle movement while it is controlled by the AI. Called by FixedUpdate in VehicleInput.`

`public abstract void` **`SetDirectSteeringAngle`**`(float angle) - Sets the direct steering angle by providing the angle of the steered wheels in degrees.`

VehicleInput.cs

Performs input to the vehicle, both manual (using W,A,S,D) and automatic (when right-clicks sets a destination on the terrain). Provides input to the **VehicleController** script using abovementioned methods.

Fields:

`public GameObject` **`PathfinderPrefab`** `– If pathfinding is used, a NavMeshAgent is instantiated and the vehicle follows it`

`public bool` **`UsesPathfinding`** `– If true, a pathfinder prefab is followed by the vehicle, otherwise the vehicle drives straight to the destination`

`public bool` **`ManualInput`** `– If manual input is used, WASD controls the unit, otherwise a destination can be given using rightclick (when selected)`

`public bool` **`ShowDebugRays`** `– Draw the forward and destination vectors in Scene view private` **`VehicleController`** `controller – Reference to the vehicle controller private float` **`throttle, steering`** `– The current throttle and steering value`

Methods:

`public void` **`SetDestination`**`(Vector3 position) - Gives the vehicle a destination on the map to move to.`

I hope you find this package helpful and easy to use in your project. If you have questions or feedback please contact me at sincress@gmail.com. Please rate this Asset and comment on the Unity Asset Store, it means a lot.

Sincress