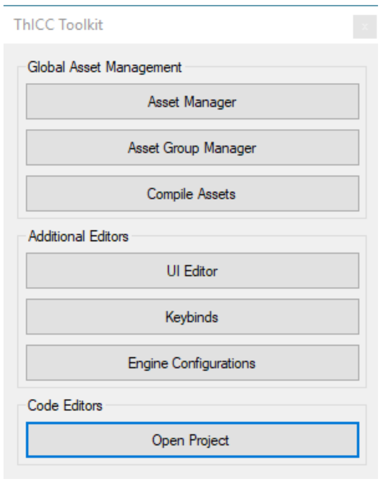
ThICC Engine Pipeline Documentation

Contents

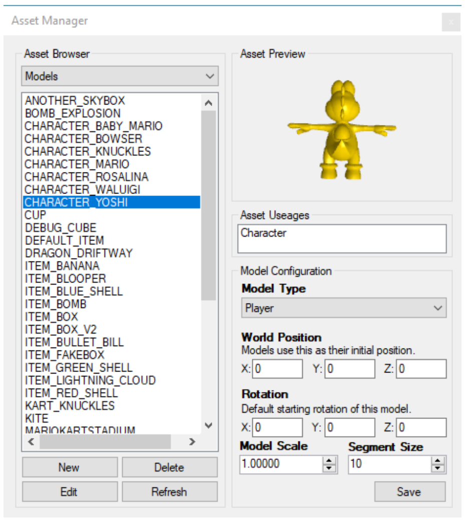
Introduction



The ThICC Engine is a 3D game framework intended for creating racing games. The engine is designed to use multi-part maps to create a playable level. Levels can be loaded dynamically during runtime with high complexity mesh collision and simple auto-generated bounding box colliders per in world object.

Content is easily customisable through the engine's toolkit which provides an entire ecosystem for importing, managing, using, and deleting assets through a usage tagging system. Additional configurations are also customisable through the toolkit for functionality such as character controllers, fonts, and cameras.

All assets are stored in the root "DATA" folder while developing. This folder is then auto-copied to any build folders (excluding non-required files) when any changes are detected by the asset compiler. The asset compiler is called automatically at program build time, or when manually triggered in the main toolkit window. A cache is stored per build type by the asset compiler to determine if files should be copied. The cache can be overridden from within the toolkit if required.



Models

ANOTHER_SKYBOX
BOMB_EXPLOSION
CHARACTER_BABY_MARIO
CHARACTER_BOWSER
CHARACTER_KNUCKLES
CHARACTER_MARIO
CHARACTER_ROSALINA
CHARACTER_WALUIGI
CHARACTER_YOSHI
CUP
DEBUG_CUBE
DEFAULT_ITEM
DRAGON_DRIFTWAY
ITEM_BANANA
ITEM_BLOOPER
ITEM_BLUE_SHELL
ITEM_BOMB
ITEM_BOX
ITEM_BOX_V2
ITEM_BULLET_BILL
ITEM_FAKEBOX
ITEM_GREEN_SHELL
ITEM_LIGHTNING_CLOUD
ITEM_RED_SHELL
KART_KNUCKLES
KITE
MARIOKARTSTADIUM

Asset Useages
Character

Model Configuration
Model Type
Player

World Position
Models use this as their initial position.
X: 0 Y: 0 Z: 0

Rotation
Default starting rotation of this model.
X: 0 Y: 0 Z: 0

Model Scale 1.00000
Segment Size 10

Any imported assets can be previewed within the asset manager. Certain assets can be edited after import, and some provide resource specific configurations for certain properties.

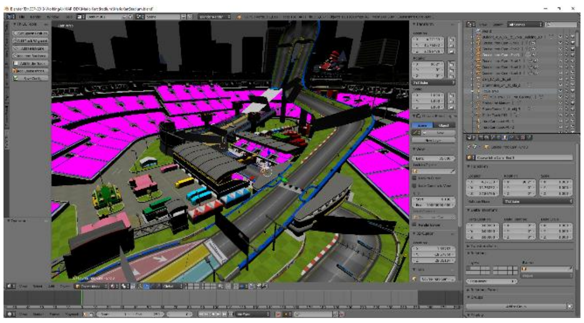
Code is compiled through Visual Studio, the project for which can be opened from the toolkit – by selecting "Open Project" on the main tool window.

The engine is designed for portability and has an optimised build target intended for use in arcade machines. This build utilises custom control glyphs, an alternate input binding system, and exclusive maps – all of which are defined through the toolkit.

Development Environment

The recommended development environment for the ThICC Engine is as follows:

- Blender 2.79 with the ThICC Blender Add-On installed
- Visual Studio 2017 (version 15.7+)
- Windows SDK 10.0
- .NET Framework 4.6.1+



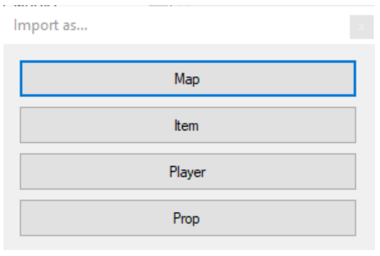
The ThICC Blender Add-On can be found within the "TOOLS/BlenderPlugin" folder and is required for producing custom map content for the engine.

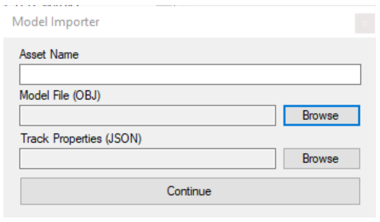
Importing new assets

A range of assets can be imported through the toolkit for use in-engine. These assets are all tracked for usages to ensure that they are not deleted while in use. Similarly, it is easy to spot non-used assets with this system for easy clean-up when coming to ship.

Importing a model

Models in OBJ format can be imported to the ThICC Engine as a variety of types. Model types define their intended use. All model types except “Map” generate a box collider. The model type “Map” will instead generate a mesh collider. This mesh collider can be utilised in-engine to provide complex collide-able surfaces derived from the imported model.





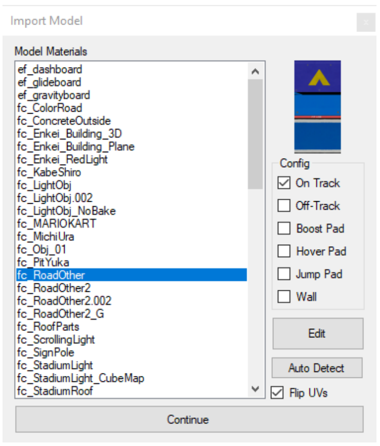
To import a model:

- Open the asset manager
- Select “New”
- Select your model type
- Enter an asset name, and locate the model’s OBJ file
- If importing type “Map”, locate your track properties JSON (see *creating a map*)
- Press continue
- Configure your materials (see *editing an imported model*)
- Press continue

The import process may take some time as vertex data is parsed and material information is processed. The engine’s toolkit utilises JSON heavily for configuration files, and also uses JSON for an interim format when parsing a model’s MTL (see *model importer interim config*). This interim JSON file also stores other useful importer data for animations, material indexes, and metallic configurations. OBJ is converted to SDKMESH using DirectXMesh, all textures are converted to the DDS format.

When importing, a model’s geometry is parsed to generate collision information. If importing type “Map”, all materials selected as “in playable area” (see *editing an imported model*) will be processed into a COLLMAP resource for the engine’s mesh collision systems. The toolkit requires triangulated models to generate mesh collision, and will auto-fix any degenerate triangle values.

Models utilise multiple formats for use in the engine, being JSON (configuration files), SDKMESH (the mesh file), OBJ+MTL files (stored for asset previews and editing), COLLMAP (a proprietary mesh collider data file), DDS (materials), and ThICC (a proprietary material configuration file). Models are stored within the “DATA/MODELS” folder in their own sub-folder, named after the model’s asset name. Within this folder all materials, model versions, and configurations are stored.


Import Model

Model Materials

ef_dashboard
ef_glideboard
ef_gravityboard
fc_ColorRoad
fc_ConcreteOutside
fc_Enkei_Building_3D
fc_Enkei_Building_Plane
fc_Enkei_RedLight
fc_KabeShiro
fc_LightObj
fc_LightObj.002
fc_LightObj_NoBake
fc_MARIOKART
fc_MichiUra
fc_Obj_01
fc_PitYuka
fc_RoadOther
fc_RoadOther2
fc_RoadOther2.002
fc_RoadOther2_G
fc_RoofParts
fc_ScrollingLight
fc_SignPole
fc_StadiumLight
fc_StadiumLight_CubeMap
fc_StadiumRoof

Config

☑ On Track
☐ Off-Track
☐ Boost Pad
☐ Hover Pad
☐ Jump Pad
☐ Wall

Edit

Auto Detect

☑ Flip UVs

Continue

For more information on COLLMAP files, see *Collision Map (COLLMAP Files)*. For more information on ThICC files, see *Material Properties (ThICC Files)*.

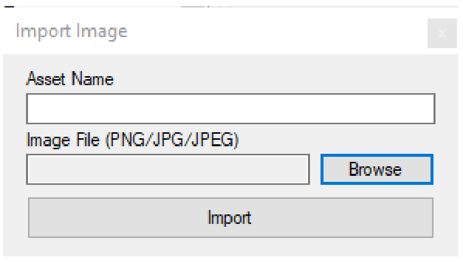
To customise a model’s configuration, find it within the asset manager and select it. The asset’s configuration will display beneath the preview box. Press “Save” to keep any changes made to the parameters.

Importing images

The ThICC Engine allows for image imports for custom 2D sprites. Accepted formats include JPG, JPEG, and PNG. The engine can automatically handle image transparency. Imported images generate a JSON configuration file containing image dimensions and a key for X/Y position. This was intended to be used in the UI editor, however its functionality was not finished in time so sprites are placed in code.

To import an image:

- Open the asset manager
- Select "Images" from the type dropdown, and press "New"
- Select your image and enter the asset name
- Press import



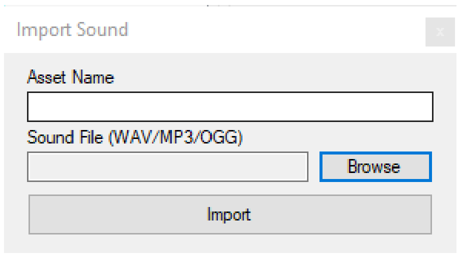
Images are converted internally using DirectXTex to DirectX DDS files formatted as R8G8B8A8_UNORM 2D. Their configuration files are stored as JSON. All images are stored in the folder "DATA/IMAGES".

Importing sounds

The ThICC Engine also allows for sound imports for custom soundtracks and sound effects. Accepted formats include WAV, MP3, and OGG. Sound properties are stored in a per-sound JSON configuration file, these configurations cover: looping, volume, pitch, and pan.

To import a sound:

- Open the asset manager
- Select "Sounds" from the type dropdown, and press "New"
- Select your sound and enter the asset name
- Press import



Sounds stored in the folder "DATA/SOUNDS" and are converted internally to WAV using FFmpeg. The WAV files used by the engine can be of any given rate. All WAV files have their own JSON configurations which are updated through the tool. They are read by the engine's sound systems directly to configure sound properties for use.

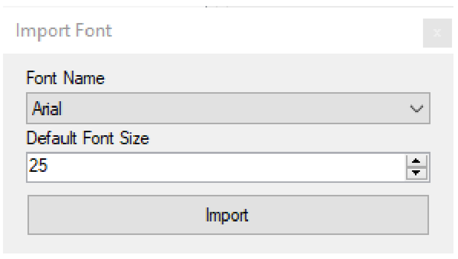
To customise a sound's configuration, find it within the asset manager and select it. The asset's configuration will display beneath the preview box. Press "Save" to keep any changes made to the parameters.

Importing fonts

The ThICC Engine supports custom fonts for displaying text at runtime. These are not assets imported to the tool as files, rather a selection from the system's already installed fonts. To use a custom font with the engine, the font must be installed to the system to be visible for import.

To import a font:

- Open the asset manager
- Select “Fonts” from the type dropdown, and press “New”
- Select your system-installed font from the dropdown
- Select a default size for the font to display at
- Press import


Import Font

Font Name: Arial

Default Font Size: 25

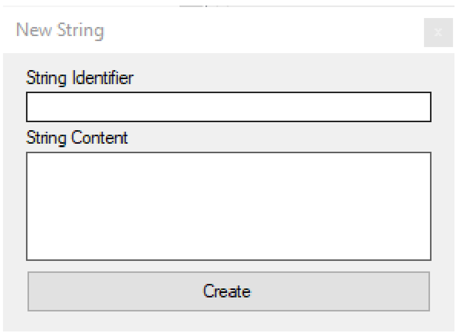
Fonts auto-generate a bitmap preview image to display in the preview window. System fonts are imported using the DirectX MakeSpriteFont tool, which also generates the preview. All fonts are stored in the SPRITEFONT format, a spritesheet of converted font glyphs. Higher imported font sizes are better for higher resolutions, as these can be scaled down to improve the pixel density of a glyph sprite. All fonts have a JSON configuration file, however this is unused in-engine and used exclusively by the toolkit. Fonts are stored in the “DATA/FONTS” folder.

Creating strings

All text strings in the ThICC engine are read from a common JSON file, sorted by language. In the current engine implementation, the toolkit only supports English – however it is very easy to expand this functionality to support multiple languages for future localisation.

To create a string:

- Open the asset manager
- Select “Strings” from the type dropdown and press “New”
- Enter an identifier for your string
- Enter the content of the string in English
- Press create


New String

String Identifier

String Content

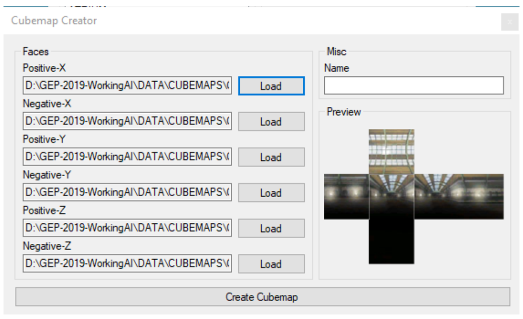
The string is created within the “LOCALISATION” JSON file in the folder “DATA/CONFIGS” and an extra entry is stored in the toolkit’s JSON file for string useage information, “LOCALISATION_INUSE”, which stores usage information to prevent deletion if being used.

Creating cubemaps

Cubemaps can be created to use per map for the ThICC Engine’s image based lighting system and reflections. Cubemaps take six images, each used for a face of the cube – those being positive and negative X,Y,Z. A preview of the generated cubemap is displayed in the editor tool. Irradiance and radiance cubemaps are automatically generated from the given images.

To create a cubemap:

- Open the asset manager
- Select “Cubemaps” from the type dropdown and press “New”
- Enter a name for the cubemap
- Load each face of the cubemap into its respective position
- Press create (this will take some time)


Cubemap Creator

Faces

Positive-X: D:\GEP-2019-WorkingAI\DATA\CUBEMAPS\

Negative-X: D:\GEP-2019-WorkingAI\DATA\CUBEMAPS\

Positive-Y: D:\GEP-2019-WorkingAI\DATA\CUBEMAPS\

Negative-Y: D:\GEP-2019-WorkingAI\DATA\CUBEMAPS\

Positive-Z: D:\GEP-2019-WorkingAI\DATA\CUBEMAPS\

Negative-Z: D:\GEP-2019-WorkingAI\DATA\CUBEMAPS\

Misc

Name

Cubemaps have irradiance and radiance variants when imported which are used in the engine to give metallic and non-metallic effects to models at runtime. The radiance (blurred) variant is generated using a bitmap blurring algorithm, while the irradiance variant uses the raw images given to the toolkit, resized to 700px squares. Both cubemap variants are stored as DirectX DDS images in the BC6H_UF16 format (cube), converted by the DirectX TexAssemble tool. Cubemaps are stored in the "DATA/CUBEMAPS" folder.

Utilising imported assets

Once imported, assets can be assigned to be used within the ThICC Engine through the toolkit's asset group manager. This process automatically tags any selected assets as "in use" and prevents them from being deleted. Assets tagged as "in use" are also easy to spot, making it possible to clear up any unused assets before shipping a project with the ThICC Engine.

Creating a map definition

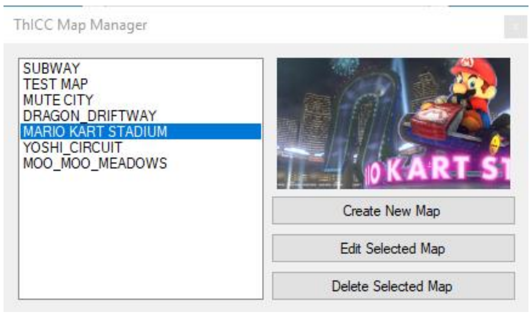
A map can be defined within the toolkit to be used within the engine. Defining a map requires a number of assets, including:

- 1 string
  - Map title in menus
- 3 images
  - Map preview in menus
  - Cubemap
  - Skybox
- 2 models
  - Map model (with collision – E.G. main track model)
  - Decoration (without collision – E.G. extended skybox models)
- 4 sounds
  - Soundtrack intro
  - Soundtrack loop
  - Final lap intro
  - Final lap loop

Also able to be defined is the "cup" that the map is in (the group) and the platform that the map is intended to be accessed on. Selecting "Arcade" will default the map to arcade exclusivity, and vice-versa for the "PC" option. Maps require a codename for reference in other locations.

To define a map:

- Open the asset group manager
- Select "Maps"
- Select "Create New Map"
- Fill out the required information (assets are selected by pressing "Load")
- Press create.



This data is compiled to the "MAP_CONFIG" JSON within the "CONFIGS" directory, and auto-loaded within the engine – able to be accessed via the service locator.

Maps configurations can be edited and deleted from the map manager window.

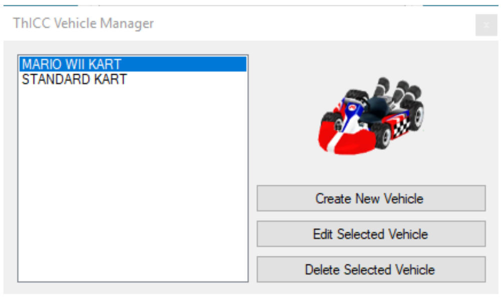
Creating a character/vehicle/cup definition

Characters, vehicles, and cups (map groups) can be defined through the tool for use in-engine. These groups allow multiple assets to be used for a single purpose - for example, a character will have sound(s), a model, and a name. Asset groups pull from existing assets, however the browser window will give you the option to import a new asset if required for the intended purpose.

All asset groups require a codename which is used internally within the toolkit.

To define a character, vehicle, or cup:

- Open the asset group manager
- Select the appropriate group editor
- Select “Create New”
- Fill out the required information (assets are selected by pressing “Load”)
- Press create



MARIO WII KART
STANDARD KART

This data is compiled to the “CHARACTER_CONFIG”, “VEHICLE_CONFIG”, and “CUP_CONFIG” JSON files respectively, and similar to maps, auto loaded within the engine – able to be accessed via the service locator.

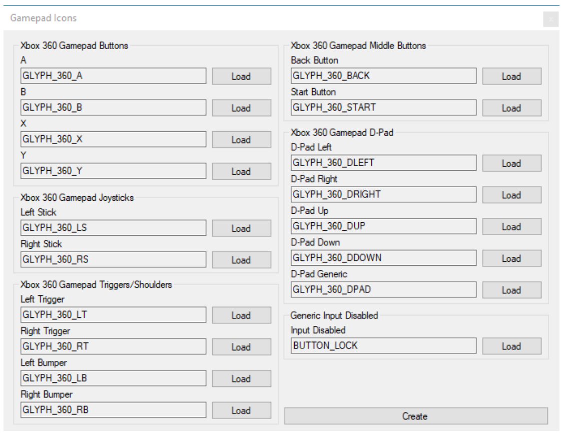
These can all be edited and deleted through their manager window, as well as the initial creation. Editing a configuration is merge-safe for teams using version control software.

Assigning input glyphs

Input glyphs are used within the ThICC Engine to relay to users the correct input to press at any given time. Glyphs are totally customisable and vary from platform to platform. On the PC, Xbox 360 gamepad glyphs are used (these also map to Xbox One), and on the arcade, arcade input glyphs are used. Input glyphs relate to their associated keybind – to define or edit a keybind see *defining keybinds*.

To assign input glyphs:

- Open the asset group manager
- Select “Gamepad Input Icons” for PC, or “Arcade Input Icons” for arcade
- Fill out the required information (assets are selected by pressing “Load”)
- Press create



Xbox 360 Gamepad Buttons
A: GLYPH_360_A
B: GLYPH_360_B
X: GLYPH_360_X
Y: GLYPH_360_Y

Xbox 360 Gamepad Joysticks
Left Stick: GLYPH_360_LS
Right Stick: GLYPH_360_RS

Xbox 360 Gamepad Triggers/Shoulders
Left Trigger: GLYPH_360_LT
Right Trigger: GLYPH_360_RT
Left Bumper: GLYPH_360_LB
Right Bumper: GLYPH_360_RB

Xbox 360 Gamepad Middle Buttons
Back Button: GLYPH_360_BACK
Start Button: GLYPH_360_START

Xbox 360 Gamepad D-Pad
D-Pad Left: GLYPH_360_DLEFT
D-Pad Right: GLYPH_360_DRIGHT
D-Pad Up: GLYPH_360_DUP
D-Pad Down: GLYPH_360_DDOWN
D-Pad Generic: GLYPH_360_DPAD

Generic Input Disabled
Input Disabled: BUTTON_LOCK

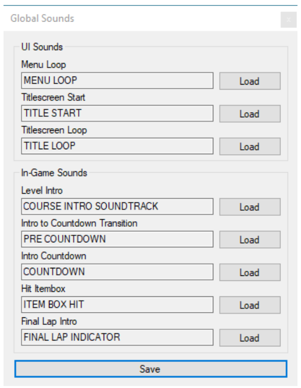
All glyphs must be defined in order to build for the platform. It is recommended to follow Microsoft’s standards for the Xbox controller glyphs.

Any inputs that are disabled per-platform can display a special disabled glyph, this is defined with in the “Gamepad Input Icons” editor window, and applies to both platforms.

The glyph configuration is stored in JSON to the “CONTROLLER_GLYPH_CONFIG” in the “CONFIGS” folder it is automatically handled in-engine by the ThICC Engine’s keybind manager.

Defining global sounds and common models

The ThICC Engine utilises common sounds and models across maps at runtime for features that are not bespoke to the current map. These assets are used for a number of situations such as menus, common mechanics, debug markers, and base skyboxes.



Global Sounds

UI Sounds

Menu Loop: MENU LOOP

Titlescreen Start: TITLE START

Titlescreen Loop: TITLE LOOP

In-Game Sounds

Level Intro: COURSE INTRO SOUNDTRACK

Intro to Countdown Transition: PRE COUNTDOWN

Intro Countdown: COUNTDOWN

Hit Itembox: ITEM BOX HIT

Final Lap Intro: FINAL LAP INDICATOR

To define a global sound or common model:

- Open the asset group manager
- Select “Shared Global Sounds” or “Common In-Game Models” as appropriate
- Fill out the required information (assets are selected by pressing “Load”)
- Press create

If defining common models, the skybox model will have all materials overridden at runtime (albedo and emissive) to the current map’s skybox texture, as defined in the map asset group. It is important for this reason that the selected model is appropriate for the intended use.

If defining global sounds, any sound slots that are labelled as “loop” should likely be configured as looping within the asset manager – to learn how to do this, see *importing sounds*.

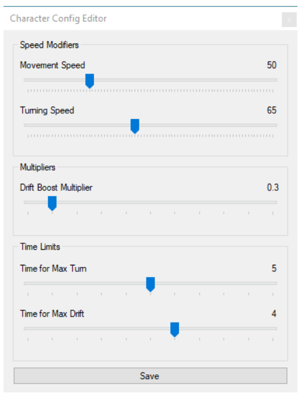
Changing configurations

A range of engine configurations are also able to be customised through the toolkit, such as character controls and camera configurations. These configurations are saved to JSON and read by the engine at runtime to define certain functionalities. The ThICC Engine is designed to be as data-driven as possible, centred around the toolkit.

Character control parameters

Character controls are able to be customised through the toolkit to modify certain behaviours at runtime, and how controls act on the player character. This is similar to Unity’s Character Controller.

To modify character configurations:



Character Config Editor

Speed Modifiers

Movement Speed: 50

Turning Speed: 65

Multipliers

Drift Boost Multiplier: 0.3

Time Limits

Time for Max Turn: 5

Time for Max Drift: 4

- Open the engine configurations window
- Select “Character Controller”
- Modify the parameters as required
- Press save

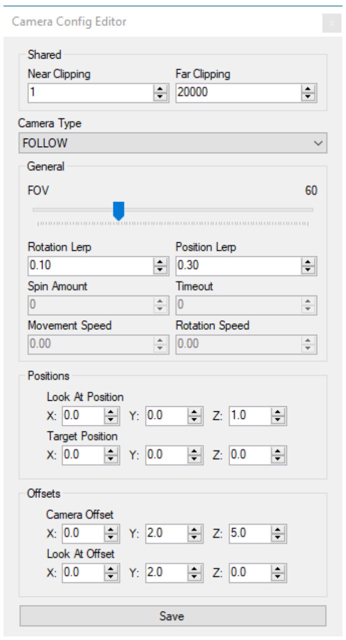
Each parameter has a different effect on the character’s controls:

- Movement speed
  - The compound movement speed for a character
- Turning speed
  - The compound turning speed for a character
- Drift Boost Multiplier
  - The multiplier given to a character’s speed based on the time spent drifting
- Time for Max Turn
  - The amount of time to take before a character is at their maximum turn speed
- Time for Max Drift
  - The amount of time to take before a character is at their maximum drifting speed

Camera parameters

Camera configurations allow for customisation of a range of parameters per each camera type. Although designed to be used as a racing engine, the ThICC Engine provides seven camera types, including:

- Follow
  - Lerps behind the player in a fixed position
- Orbit
  - Orbits around the player automatically
- Cinematic
  - Performs zero to four pre-defined camera lerps (see *creating a map*)
- First
  - First person perspective
- Independent
  - Follows the player with free movement control
- Back Facing
  - Facing back from the player's forward direction
- Debug
  - "Freecam" mode which allows movement around the environment freely



Camera Config Editor

Shared

Near Clipping	Far Clipping
1	20000

Camera Type: FOLLOW

General

FOV 60

Rotation Lerp	Position Lerp
0.10	0.30
Spin Amount	**Timeout**
0	0
Movement Speed	**Rotation Speed**
0.00	0.00

Positions

	X:	Y:	Z:
Look At Position	0.0	0.0	1.0
Target Position	0.0	0.0	0.0

Offsets

	X:	Y:	Z:
Camera Offset	0.0	2.0	5.0
Look At Offset	0.0	2.0	0.0

Configurations for near and far clipping are global to all of these camera types, this defines the minimum and maximum render area for the camera's view. A lower far clipping distance will improve performance and is a recommended optimisation. The engine cap for far clipping is 50,000 units. Further camera type specific parameters include:

- FOV [all types]
  - The field of view of the camera, valid for a range of 30 to 130
- Rotation Lerp [all types]
  - The lerp applied to rotations
- Position Lerp [all types]
  - The lerp to apply to positions
- Spin Amount [orbit/independent]
  - The amount to spin per second
- Timeout [cinematic]
  - The time before switching to the next cinematic camera group
- Movement Speed [debug]
  - The speed of the camera's free movement over position
- Rotation Speed [debug]
  - The speed of the camera's free movement over rotation
- Look At Position [all types]
  - The default position to look at
- Target Position [all types]
  - The default position to set to
- Camera Offset [all types]
  - The offset of the camera's position from the parent
- Look At Offset [all types]
  - The offset of the camera's look target

To edit the camera configurations:

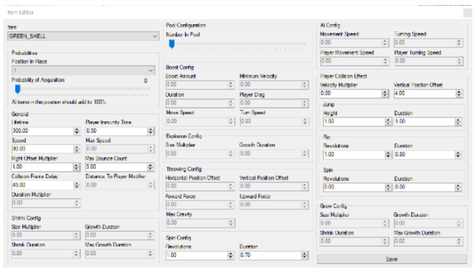
- Open the engine configurations window
- Select “Camera Configurations”
- Modify the parameters as required
- Press save

Editing items

The ThICC Engine provides a number of in-game special items able to be acquired through the item boxes placed within a map (see *creating a map*). Items are customisable through the engine toolkit to tweak a number of values in their configurations and the assets that they use.

To edit item configurations:

- Open the engine configurations window
- Select “Item Config Editor”
- Modify the parameters as required (by selected item type)
- Press save



Item Editor

Item: GREEN_SHELL

Probabilities
Position In Race: 1
Probability of Acquisition: 0
All items in this position should add to 100%.

General
Lifetime: 300.00
Player Immunity Time: 0.50
Speed: 80.00
Max Speed: 0.00
Right Offset Multiplier: 1.00
Max Bounce Count: 5.00
Collision Frame Delay: 45.00
Distance To Player Modifier: 0.00
Duration Multiplier: 0.00

Shrink Config
Size Multiplier: 0.00
Growth Duraton: 0.00
Shrink Duration: 0.00
Max Growth Duration: 0.00

Pool Configuration
Number In Pool

Boost Config
Boost Amount: 0.00
Minimum Velocity: 0.00
Duration: 0.00
Player Drag: 0.00
Move Speed: 0.00
Turn Speed: 0.00

Explosion Config
Size Multiplier: 0.00
Growth Duration: 0.00

Throwing Config
Horizontal Position Offset: 0.00
Vertical Position Offset: 0.00
Forward Force: 0.00
Upward Force: 0.00
Max Gravity: 0.00

Spin Config
Revolutions: 1.00
Duration: 0.70

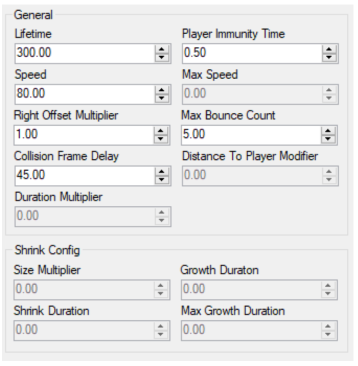
AI Config
Movement Speed: 0.00
Turning Speed: 0.00
Player Movement Speed: 0.00
Player Turning Speed: 0.00

Player Collision Effect
Velocity Multiplier: 0.00
Vertical Position Offset: 4.00
Jump
Height: 1.50
Duration: 1.00
Flip
Revolutions: 1.00
Duration: 0.80
Spin
Revolutions: 0.00
Duration: 0.00

Grow Config
Size Multiplier: 0.00
Growth Duration: 0.00
Shrink Duration: 0.00
Max Growth Duration: 0.00

Save

There are a number of configurations able to be edited within the item editor. While all are shown in the editor at any one time, only the enabled inputs apply to the currently selected item, and the others will be ignored. Item configurations are:

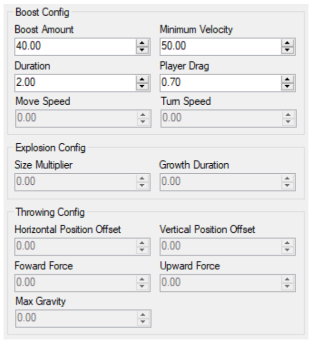


General
Lifetime: 300.00
Player Immunity Time: 0.50
Speed: 80.00
Max Speed: 0.00
Right Offset Multiplier: 1.00
Max Bounce Count: 5.00
Collision Frame Delay: 45.00
Distance To Player Modifier: 0.00
Duration Multiplier: 0.00

Shrink Config
Size Multiplier: 0.00
Growth Duraton: 0.00
Shrink Duration: 0.00
Max Growth Duration: 0.00

- General
    - Lifetime - the duration of the item, item is destroyed once lifetime is reached
    - Player Immunity Time - the amount of time the item holds a pointer to the player that used it - prevents collisions with the using player as soon as the item is used
    - Speed - the speed that the item moves at when used
    - Max Speed - the maximum speed the item can move at
    - Right Offset Multiplier - the offset multiplier applied to the items world matrix when used, making appear to the right of the player
    - Max Bounce Count - the number of times an item bounces before being destroyed
    - Collision Frame Delay - the frame delay between item and wall collision responses - used to stop wall collisions being checked multiple times per hit
    - Distance To Player Modifier - the threshold distance between a red shell and its target player (squared)
    - Duration Multiplier - the duration of the item, multiplied
- Grow/Shrink Config
    - Size Multiplier - the multiplier applied to the player's scale to grow/shrink to
    - Growth Duration - the time taken for the player to grow
    - Shrink Duration - the time taken for the player to shrink
    - Max Growth/Shrink Duration - the duration the player spends in the modified size
- Pool Configuration
    - Number In Pool – the number of item meshes spawned into the preloaded mesh pool when entering a new environment

- Boost Config
  - Boost Amount – multiplied by the player's normalised velocity which is then added to the player's velocity to give them a speed boost
  - Minimum Velocity - the velocity given to the player before they boost if they're under this velocity
  - Duration - the duration of the boost
  - Player Drag - the amount of drag applied to the player when they boost
  - Move Speed - the move speed given to the player
  - Turn Speed - the turn speed given to the player

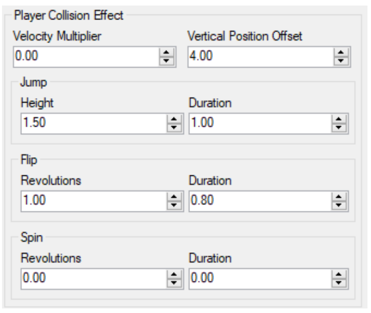


Boost Config
Boost Amount 40.00
Minimum Velocity 50.00
Duration 2.00
Player Drag 0.70
Move Speed 0.00
Turn Speed 0.00

Explosion Config
Size Multiplier 0.00
Growth Duration 0.00

Throwing Config
Horizontal Position Offset 0.00
Vertical Position Offset 0.00
Foward Force 0.00
Upward Force 0.00
Max Gravity 0.00

- Explosion Config
  - Size Multiplier - the multiplier applied to the explosion's scale to get the scale to grow to
  - Growth Duration - the time take for the explosion to grow
- Throwing Config
  - Horizontal Position Offset – the horizontal position the item is set to before throwing
  - Vertical Position Offset – the vertical position the item is set to before throwing
  - Forward Force – the forward force applied to the item thrown
  - Upward Force – the upward force applied to the item thrown
  - Max Gravity – the change in gravity to apply a “floaty” effect to thrown items
- Spin Config
  - Revolutions - the number of revolutions spun per update tick
  - Duration - the time taken for all the revolutions set
- AI Config
  - Movement Speed – the regular movement speed of the item AI
  - Turning Speed – the regular turning speed of the item AI
  - Player Movement Speed – the movement speed of an item AI which affects the player
  - Player Turning Speed – the turning speed of an item AI which affects the player
- Player Collision Effect
  - Velocity Multiplier - the multiplier applied to the players velocity when they collide with the item - can either slow down or speed up the player
  - Vertical Position - the vertical offset applied to the player's world matrix causing the mesh to launch upwards
  - Jump Height - the height that the player's animation mesh lerps to when colliding with an item
  - Jump Duration - the time taken for the player to jump
  - Flip Revolutions - the number of revolutions that the player's animation mesh flips when colliding with an item
  - Flip Duration - the time taken for all the revolutions set
  - Spin Revolutions - the number of revolutions that the player's animation mesh spins when colliding with an item
  - Spin Duration - the time taken for all the revolutions set

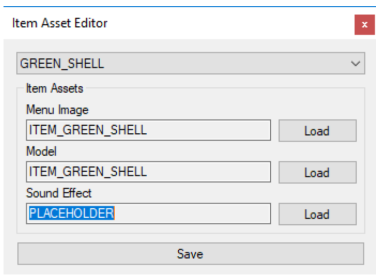


Player Collision Effect
Velocity Multiplier 0.00
Vertical Position Offset 4.00
Jump
Height 1.50
Duration 1.00
Flip
Revolutions 1.00
Duration 0.80
Spin
Revolutions 0.00
Duration 0.00

Item assets can be selected by using the asset grouping tool:

- Open the asset group manager
- Select "In-Game Items"
- Select the assets as required (by selected item type)

Assets used for items are tagged as "Items" use. They are referenced in the item's JSON file "ITEM_CONFIG", and sound assets are stored in the global "SOUNDS_CONFIG" JSON file.



GREEN_SHELL

Menu Image: ITEM_GREEN_SHELL

Model: ITEM_GREEN_SHELL

Sound Effect: PLACEHOLDER

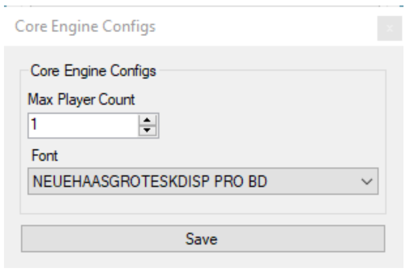
Core engine parameters

Configurations for other core engine parameters can be edited through the toolkit for completeness in the data-driven system. This includes:

- Maximum player count (caps the in-game player count, allows a choice of 1-4)
- Font (used for all UI text)

To edit the core engine parameters:

- Open the engine configurations window
- Select "Core Engine Configurations"
- Modify the parameters as required
- Press save



Max Player Count: 1

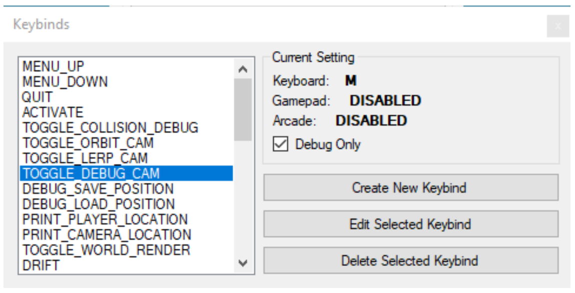
Font: NEUEHAASGROTESKDISP PRO BD

Defining keybinds

Inputs are handled in the ThICC Engine through a cross-platform keybind manager, which can be configured through the toolkit. Inputs are defined through a single identifier and can be defined for keyboard, gamepad, and arcade control schemes. Arcade controls take effect in the engine's arcade build, while keyboard and gamepad take effect in the game's PC build. Keybinds can be tagged as a "debug keybind" which will disable the input's functionality in any release builds. This is handy to disable any keys which enable debug options in-engine (such as the debug camera, see *camera parameters*) which may accidentally be skipped by any pre-processor directives.

To create a new keybind:

- Open the keybinds window
- Select "Create New Keybind"
- Enter a name for the keybind – this will be utilised in code when calling the keybind manager
- If the keybind is debug only, select the appropriate option
- Select the appropriate inputs for keyboard, gamepad, and arcade – if not required, leave as "DEFAULT"
- Press create

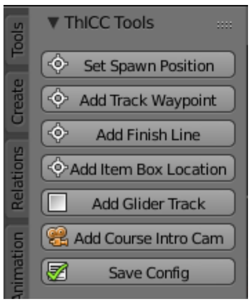


MENU_UP
MENU_DOWN
QUIT
ACTIVATE
TOGGLE_COLLISION_DEBUG
TOGGLE_ORBIT_CAM
TOGGLE_LERP_CAM
TOGGLE_DEBUG_CAM
DEBUG_SAVE_POSITION
DEBUG_LOAD_POSITION
PRINT_PLAYER_LOCATION
PRINT_CAMERA_LOCATION
TOGGLE_WORLD_RENDER
DRIFT

Keyboard: **M**
Gamepad: **DISABLED**
Arcade: **DISABLED**
☑ Debug Only

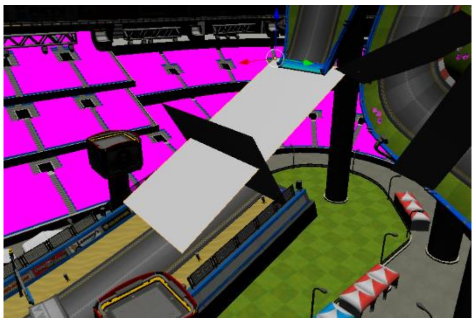
As well as the ability to create, keybinds can be both edited and deleted from the keybinds window.

Creating a map



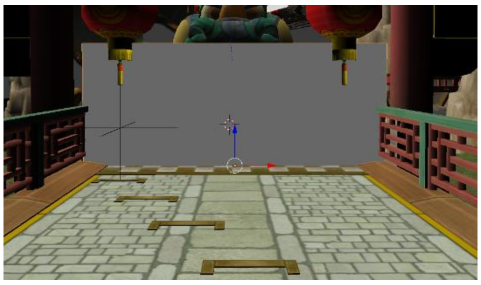
The ThICC Engine utilises open source tool Blender as its level editor. To develop maps for the ThICC Engine, the ThICC Blender Add-On is required (see *development environment*). This plugin unlocks a number of functions within Blender for adding scripted content handles that the engine utilises. The plugin includes:

- Set Spawn Position
  - Set the spawn position for all players within the map (max of one per map)
- Add Track Waypoint
  - Creates a waypoint used to calculate the player's current position in the race, the AI pathing, and lap progression
- Add Finish Line
  - Create a finish line waypoint variant which defines the finish line, used to increment the current lap if players have hit all waypoints (max of one per map)
- Add Item Box Location
  - Adds an item box location to the map, at runtime this will spawn an item cube which will give players a power-up when collided with
- Add Glider Track
  - Adds a glide plane used as a positional helper when players are in glider mode at runtime (max of one per map)
- Add Course Intro Cam
  - Adds a group of two cameras and a look-at point for creating scripted course introductions, tied to the cinematic camera system (see *camera parameters*)
- Save Config
  - Saves all added markers from the plugin to JSON for importing alongside a map (see *importing a model*)



Glider tracks, track waypoints, and the finish line are all represented by meshes in the Blender scene. For this reason, maps must be exported in OBJ format **before** adding these elements. For this reason, the typical ThICC map creation workflow should be: produce map, export map as OBJ, add waypoints and other required marker data through the plugin, export the configuration file.

The standard for placing waypoints is to place them so that the centre of the waypoint is level with the track surface. This can be easily achieved in Blender by moving the transform handles to the centre of the waypoint with the shortcut "CTRL+SHIFT+ALT+C". Waypoints should be placed in the order that they are expected to be collided with. This is the order the engine will use to calculate the related mechanics.



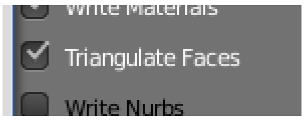
Item box locations should be placed so that their marker partially intersects with the ground for the most reliable collision height.

Spawn positions should be placed slightly above the track surface to avoid spawning in or below the track's surface.

To preview the output of the course intro cams, press "NUMPAD 0". "CTRL+NUMPAD 0" will switch the active camera preview. Cameras focus on a "look at point" – to realign cameras, move this point rather than rotating the camera object. The time to lerp between both specified camera positions can be configured in the toolkit (see *camera parameters*).

There is a recommended maximum poly-count of ~200k when creating maps for the ThICC Engine. If your map begins to exceed this, take advantage of the "decoration" model which allows for scenery alongside the main track in each map (see *creating a map definition*).

If creating a model which is going to have collision enabled in the engine, select "triangulate faces" when exporting the model's OBJ file from Blender. Failure to do this will result in an error in the toolkit, stopping you from importing the model.

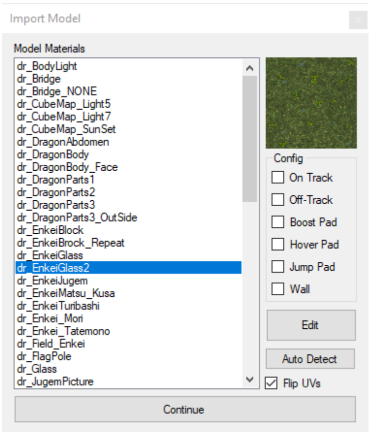


Write Materials
☑ Triangulate Faces
☐ Write Nurbs

Editing an imported model

With the ThICC Engine, imported models can be edited after-the-fact. This allows for tweaks to material properties after initial import and is useful to refine the model's visual style. To reach the edit window:

- Open the asset manager
- Select a model from the list
- Press "Edit"



Import Model

Model Materials

dr_BodyLight
dr_Bridge
dr_Bridge_NONE
dr_CubeMap_Light5
dr_CubeMap_Light7
dr_CubeMap_SunSet
dr_DragonAbdomen
dr_DragonBody
dr_DragonBody_Face
dr_DragonParts1
dr_DragonParts2
dr_DragonParts3
dr_DragonParts3_OutSide
dr_EnkeiBlock
dr_EnkeiBrock_Repeat
dr_EnkeiGlass
dr_EnkeiGlass2
dr_EnkeiJugem
dr_EnkeiMatsu_Kusa
dr_EnkeiTuribashi
dr_Enkei_Mori
dr_Enkei_Tatemono
dr_Field_Enkei
dr_FlagPole
dr_Glass
dr_JugemPicture

Config
☐ On Track
☐ Off-Track
☐ Boost Pad
☐ Hover Pad
☐ Jump Pad
☐ Wall

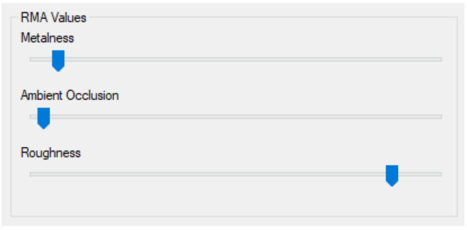
Edit
Auto Detect
☑ Flip UVs
Continue

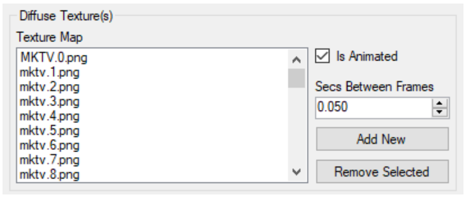
From this window all materials within the model are listed. If you select a material you will receive a preview of its diffuse (albedo) material in the top right of the window. Below that, you will see its collision config (this is disabled for models not of type "Map", see *importing a model*). This window holds the option to flip a model's UV information, this is particularly useful if you are importing a model to the engine from an alternate coordinate scheme (E.G. OpenGL). There is also the option to "Auto Detect" – this will search all materials in the model and attempt to pick the best collision parameters based on its metadata.

To edit a selected material press "Edit". This will open the material editor for the selected material.

Within the material editor there are a number of configurations for different material properties:

- RMA Values
  - Metalness – how metallic the material is
  - Ambient Occlusion – how big of an AO effect should be applied
  - Roughness – how rough the material seems
- Diffuse Texture(s)
  - Diffuse textures – minimum of 1, unlimited maximum
    - If using more than one, an animation time must be specified





Diffuse Texture(s)

Texture Map

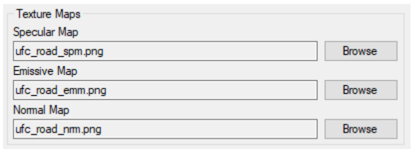
MKTV.0.png
mktv.1.png
mktv.2.png
mktv.3.png
mktv.4.png
mktv.5.png
mktv.6.png
mktv.7.png
mktv.8.png

☑ Is Animated

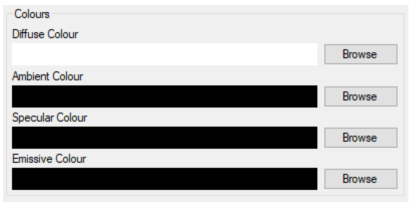
Secs Between Frames
0.050

Add New
Remove Selected

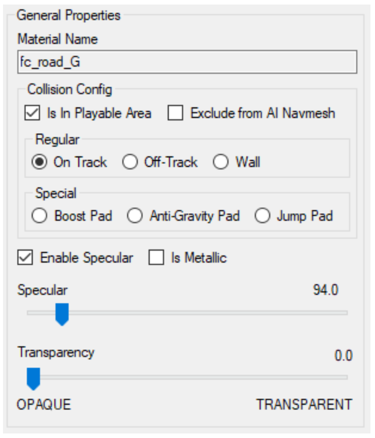
- Texture Maps
  - Specular Map – the specular texture file for defining glossy areas
  - Emissive Map – the emissive texture file for defining perceived illumination
  - Normal Map – the normal texture file for defining extra perceived extrusion details


Texture Maps
Specular Map: ufc_road_spm.png
Emissive Map: ufc_road_emm.png
Normal Map: ufc_road_nrm.png

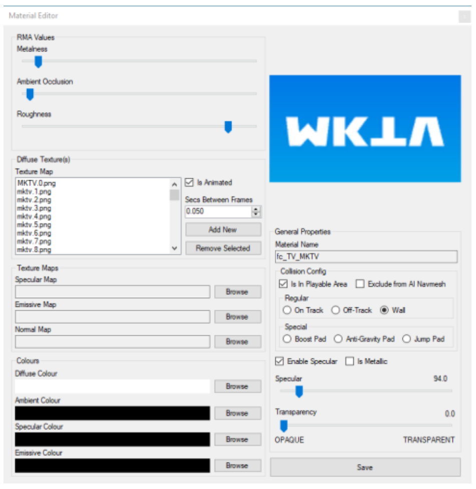
- Colours
  - Diffuse Colour – a colour modifier applied to the diffuse map(s)
  - Ambient Colour – a colour modifier applied to the material as an ambient source
  - Specular Colour – a colour modifier applied to the specular gloss
  - Emissive Colour – a colour modifier applied to the emissive map


Colours
Diffuse Colour: ⬜
Ambient Colour: ⬛
Specular Colour: ⬛
Emissive Colour: ⬛

- General Properties
  - Material Name – this cannot be edited, matches the material name in Blender
  - Collision Config – this allows the definition of collision parameters for "Map" model types (see *importing a model*)
    - Is In Playable Area – should the material be collidable?
    - Exclude from AI Navmesh – if in playable area, should the AI ignore it?
    - On Track – this material is part of the track
    - Off Track – this material is off of the track, a speed modifier is applied
    - Wall – this material forms part of a wall, is not traversable
    - Boost Pad – a special speed boost pad
    - Anti-Gravity Pad – a special type intended for anti-gravity pads
    - Jump Pad – a special pad type to be placed before glider sections (see *creating a map*)
  - Enable Specular – enable/disable the specular glossy effects
  - Is Metallic – allow the engine to render the material as shiny metallic?
  - Specular – a value from 0-1000 to define the specular intensity
  - Transparency – a value from 0-1 to define how transparent the material is (0 = opaque, 1 = transparent).


General Properties
Material Name: fc_road_G
Collision Config
[✓ Is In Playable Area] Exclude from AI Navmesh
Regular: [✓ On Track] Off-Track Wall
Special: Boost Pad Anti-Gravity Pad Jump Pad
[✓ Enable Specular] Is Metallic
Specular 94.0
Transparency 0.0

Material properties are stored in the importer's interim JSON file which is kept in memory while editing, and saved out when changes are confirmed. The material properties are then converted back to MTL when the import process starts up again after editing is complete (see *importing a model*).


Material Editor
RMA Values
Metalness
Ambient Occlusion
Roughness
Diffuse Texture(s)
Texture Map: MKTV.0.png, mktv.1.png, mktv.2.png, mktv.3.png, mktv.4.png, mktv.5.png, mktv.6.png, mktv.7.png, mktv.8.png
[✓ Is Animated]
Secs Between Frames: 0.050
Texture Maps
Specular Map:
Emissive Map:
Normal Map:
Colours
Diffuse Colour: ⬜
Ambient Colour: ⬛
Specular Colour: ⬛
Emissive Colour: ⬛
General Properties
Material Name: fc_TV_MKTV
Collision Config
[✓ Is In Playable Area] Exclude from AI Navmesh
Regular: On Track Off-Track [✓ Wall]
Special: Boost Pad Anti-Gravity Pad Jump Pad
[✓ Enable Specular] Is Metallic
Specular 94.0
Transparency 0.0

Collision map (COLLMAP files)

One of the key features of the ThICC Engine is its support for complex mesh collision, achieved through a proprietary “COLLMAP” format in the engine pipeline. The COLLMAP file stores easy to parse geometry information which is generated through the toolkit for use in the engine’s collision scripts.

The geometry information is stored in groups of 4-byte float values within a binary file. The binary file can contain as many groups as required. The number of groups within the binary file is specified as a 4-byte integer value at the first position in the file, making up the first value of the collision file’s header.

Following the group count in the header are group sizes. The number of group size counts in the header can be determined from the number of groups that are contained within. Group sizes are defined by a 4-byte integer. The size of a group is not its size in memory, but the number of items it contains. A calculation of size * 4 (size of float) can be performed to work out the group offsets within the file.

Each group’s intended use can be determined by its index within the file. The group’s index relates to an enum shared between the engine’s editor tool and game client (CollisionType) which describes the intended use case: for example: wall, or track. This makes the entire file extremely scale-able and completely modular for adding new collision groups in future projects. For wider use, a version number may be required to be added to the header signature to ensure backward compatibility with older collision group definitions.

Material properties (ThICC files)

The ThICC Engine utilises per-material properties which are applied at a low level during render time. These properties are:

- “Is metallic”
    - This option changes the IBL textures applied to the material at render time, giving the illusion of something being metallic if required. Materials marked as metallic will have a more detailed cubemap handed to the IBL system to give the illusion of a higher reflection property.
- Animated diffuse maps
    - Animated diffuse maps are applied at render time based on the current frame time. All available diffuse maps to flipbook between are specified for a material in the editor toolkit, along with an animation time. If provided, the render function then applies the correctly timed diffuse texture to the mesh part before rendering.

These properties are achieved through binary files using a proprietary extension of “.ThICC”. Each file has its own header signature and method of reading. Both share a common start to the file, being the “ThICC” signature to verify the file’s legitimacy and version. This header is structured as such:

- File version number (integer) – allows for backwards compatibility
- Identifier (5 chars, creating “ThICC”) – used to verify file’s legitimacy

Each ThICC file then breaks into its specific headers and contents…

REFLECTION.ThICC

This ThICC file defines the “is metallic” property in the material editor (see *editing an imported model*), and is formatted as such:

- Number of materials (integer)
- [for each material]
    - Is metallic (bool)

ANIMATION.ThICC

This ThICC file defines the animated diffuse maps (see *editing an imported model*), and is formatted as such:

- Number of materials (integer)
- [for each material]
    - Material index in model (integer)
    - Number of textures to animate (integer)
    - Time between textures (float)
    - [for each texture]
        - Length of texture filename (int)
        - Texture filename (a series of chars, of length previously specified)

Material properties (misc)

As well as ThICC files to define custom low level properties that are bespoke applications for the ThICC Engine, a number of other common properties are utilised such as:

- Emissive maps
- Specular maps
- Normal maps
- Ambient colour
- Diffuse colour
- Specular colour
- Emissive colour
- RMA
    - Metalness
    - Ambient Occlusion
    - Roughness

These configurations are applied to the model for use in-engine through DirectXMesh’s conversion to SDKMESH2 from OBJ/MTL. The commands in the model’s MTL file define each of these properties, which is compiled by the toolkit from the interim JSON configuration file. An imported model’s MTL is parsed to JSON to handle the extra engine configuration properties that are not included in the standard MTL definitions. When re-compiling the map, the toolkit parses its JSON configuration back out to MTL for DirectXMesh’s use.

An RMA map of metalness/AO/roughness is compiled by the toolkit utilising the RGB channels in the glTF2 standard, as required by DirectX’s specifications. As part of the development of this project, two new MTL commands were proposed: “map_RMA” and “map_ORM” – both implemented into DirectXMesh to allow for the ability to define PBR RMA maps in a model’s material file.

Model importer interim config

The toolkit's model importer utilises a JSON file named "IMPORTER_CONFIG" as an interim configuration file while importing models. This file is stored within each model's directory inside "DATA/MODELS". The interim file pulls model material commands from the imported OBJ's MTL file and stores them within a JSON object. Inside this JSON object, engine-specific configurations are also registered which the toolkit reads from when building the ThICC material property files. Alongside any pulled MTL commands, the following JSON keys are added:

- "ThICC_METALLIC" – a bool value which is written to the "REFLECTION.ThICC" file when a model is compiled
- "ThICC_COLLISION" – an object of bools defining collision parameters: this matches the enum found within the engine and toolkit and order should not be changed
    - Any collision types not specified in this object are interpreted as false in-engine
- "ThICC_INDEX" – the material's index within the OBJ file for ordering graphical effects within the engine's effect factories
- "ThICC_ANIMATION" – an object of the diffuse map files to animate (if only one is listed, no animation is present) – when compiled to "ANIMATION.ThICC" at asset compile time, all image extensions are replaced with DDS
- "ThICC_ANIMATION_ENABLED" – a bool value that specifies if this material is animated, toolkit defaults this to false unless two or more diffuse maps are supplied
- "ThICC_ANIMATION_TIME" – the time between animation frames, only utilised if material is animated, is compiled to "ANIMATION.ThICC" at asset compile time

It is strongly advised, as with all JSON configuration files, that you do not manually edit the contents. This could lead to compilation issues or graphical bugs at runtime.

Asset file structure

- DATA
    - CONFIGS (all configuration files and string definitions)
    - CUBEMAPS (all cubemaps)
    - FONTS (all fonts)
    - IMAGES (all images for sprite use)
    - MODELS (all models, stored in their own folders)
    - SHADERS (all shaders – depreciated)
    - SOUNDS (all sounds)

The DATA folder in the toolkit's working directory includes development assets and additional executables for compiling resources. These are not copied to the build directories by the asset compiler, so will not ship with any projects created with the ThICC Engine.

Creating UI elements

Since the ThICC Engine was created under a time constraint, unfortunately its UI editor was not finished in time for current release. As a result of this, all UI is hard-coded in the engine's game-specific scripts, going against the data-driven goal of the project. That said, all UI within the game will automatically scale to any resolution the game is launched at automatically. The engine will also handle on-the-fly window resizing.

To work with the resolution scaling system, the UI must be programmed as if the game is running at 720p. This is our “virtual” resolution which we scale up/down from depending on the resolution that the user selects within the launcher.

Images can be imported through the engine’s toolkit for use in UI (see *importing images*) and referenced in code through the GameFilepaths system which automatically generates a filepath internally using the asset file structure (see *asset file structure*) when given an asset name and type.

Credits

The ThICC Engine utilises a number of libraries and additional tools for its pipeline, these are:

- FFmpeg (GNU Lesser Public General Public License)
- TexAssemble (MIT License)
- DirectXMesh (MIT License)
- TexConv (MIT License)
- Newtonsoft Json.NET (MIT License)
- Costura Fody (MIT License)