# Best Engine Documentation

## Core::

Core is the central control object for the entire engine. It contains all the entities in the scene as well as the input manager and environment variables.

### std::shared_ptr<InputManager> Input

Input stores all button presses for the engine in separate vectors. Core holds the variable for the input manager from which the inputs can be accessed

### std::vector<std::shared_ptr<Entity> > entities

This vector stores a pointer to every entity in the scene

### std::shared_ptr<Environment> environment

This points to the environment for the engine, which locks the fps to 60 max and keeps the delta time for each frame

### std::weak_ptr<Core> self

This is a self pointer for core. Weak pointers are used for self and up the hierarchy to avoid memory leaks

### Static std::shared_ptr<Core> initialise()

This static function creates the core and must be the first thing called in any program using this engine. It sets up the self pointers and all necessary variables for core to function

### std::shared_ptr<Entity> addEntity(std::string name)

### std::shared_ptr<Entity> addEntity(std::string name, std::shared_ptr<Transform> position)

This function will add an entity to the vector with the name passed in. the overload will additionally set the entity's transform to the one passed in. if a transform is not provided it will default to 0 position, 0 rotation and 1 scale.

### std::shared_ptr<Entity> getCurrentScreen()

A simple getter for the camera core is currently rendering to

### std::shared_ptr<Entity> findEntity(std::string name)

Function to find an entity with the specified name in the vector

### Void start()

This function runs the main game loop and calls all the entity tick functions. It must be called at the end of initialisation in int main() to begin the program

## Component::

Base class for the components in the engine where the majority of the user's code will be written

### std::weak_ptr<Entity> entity

Pointer to the entity the component is attached to for moving up the hierarchy

### std::weak_ptr<Component> self

Pointer to self

### std::shared_ptr<Entity> getEntity()

Returns the entity that the component is attached to

### std::shared_ptr<Screen> getCurrentScreen()

Return the current main camera of core. Used to reduce work for end user

### Virtual void onTick()

Called every frame to update the component. This can be overridden by the user to have logic happen every frame

### Virtual void onInitialise()

Called when the component is created. Can be overridden by the end user

### Virtual void onRender()

Called when the component’s entity is rendered. Can be used to change rendering

### Virtual void onCollision(Collider other)

This function is called when the object collides with another. Other is the collider for the object the collision was with

## Entity::

An entity is any object in the scene. They must all have a transform and renderer component and are given one by default

### std::weak_ptr<Entity> self

Self weak pointer

### std::weak_ptr<Core> core

Pointer up the hierarchy

### Std::string name

Stores the name of the entity

### std::vector<std::shared_ptr<Component> > components

Vector containing all the components attached to this entity

### std::shared_ptr<T> addComponent()

### std::shared_ptr<T> addComponent(std::shared_ptr<T> component)

Adds a component of the type specified to the entity. If a premade component is not passed into the function then its variables will begin at null as there will be no initialization by default

### std::shared_ptr<T> getComponent()

Gets a component of the type specified. Entities should never have more than one component per derived type

### std::shared_ptr<Core> getCore()

Returns a pointer to core

### std::shared_ptr<Screen> getCurrentScreen

Returns the current screen in core

## Transform::

Component containing positional data of the entity

### Glm::vec3 position

Stores a 3D vector containing the position of the entity

### Glm::vec3 rotation

Stores a 3D vector containing the rotation of the entity

### Glm::vec3 scale

Stores a 3D vector containing the scale of the entity

### Glm::mat4 getModel()

Returns a 4x4 model matrix which can transform a point from object space to world space

## Collider::

Component for testing collisions with other entities

### Virtual glm::vec3 findFurthestPoint(glm::vec3 direction)

Overload function for different types of colliders

### Bool mesh

False if the collider is a sphere. True if not

### Float maxDis

The maximum distance from the center of an object to one of its points. Used for the broad stage of collision detection

## MeshCollider::

Derived from collider. Specific type of collider defined by a set of vertices

### std::vector<glm::vec3> vertices

Contains each vertex for the mesh collider

## InputManager::

Object that stores all keyboard and mouse inputs for the user to check against

### Bool getKeyboardInput(int key)

Checks the keys pressed for the key passed in. Returns true if input is found

### Bool findKeyUp(int key)

Checks if the key passed in was released this frame

### Bool findKeyDown(int key)

Checks if the key passed in was pressed down this frame

### Bool getMouseInput(int button)

Checks the mouse buttons pressed for the button passed in. Returns true if input is found

### Bool findMouseUp(int button)

Checks if the button passed in was released this frame

### Bool findMouseDown(int button)

Checks if the button passed in was pressed down this frame

## Environment::

Object containing deltaTime and performing frame rate locking

### Float deltaTime

Stores the time the previous frame took

## Mesh::

Object for handling the rendering mesh and texture of objects

### Void initialise(std::string path)

Initialises the mesh from the file path passed in

### Void addTexture(std::string path)

Gets the texture from the file in the path passed in adding it to the mesh

## Screen::

Component storing all variables needed for the perspective matrix

### Int getWidth()

Returns the width of the screen

### Int getHeight()

Returns the height of the screen

### Float getFOV()

Returns the FOV angle of the screen

### Float getNear()

Returns the near plane of the screen

## Float getFar()

Returns the far plane of the screen

## Glm::mat4 getPerspective()

Returns a perspective matrix for the camera