Marcin Lusa

Space Ship Simulatior

Project documentation
2th May 2017

VS solution structure

Solution is build from three projects:

- ReactPhysics3d - physics library used for collision detection. www.reactphysics3d.com
- Engine - general purpose mini engine/framework written on the occasion of working on test game
- Game - code implementing test game. Only few classes related to the game logic.

Interesting things

This paragraph contains description of some architectural choices and solution that may be interesting.

GameObject

Engine is using component approach to build a game. All spawned objects are of GameObject class and GameObject is just a container for components. All components derive from Component class. Components available in the engine:

- TransformComponent - used to hold position, rotation and scale of an object
- TransformMoverComponent - used to move TransformComponent in constant way
- TransformRotatorComponent - used to constantly rotate TranformComponent
- MeshRendererComponent - used to render Mesh
- ColliderComponent - used to assign colliding volume to an object
- CameraComponent - used to calculate view and projection matrices for rendering
- LightSourceComponent - components used to represent light in the game world

Components related to the test game:

- SpaceShipController - used to control ship movement and shooting
- MeteorController - used to manage spawning and despawning meteors

BaseGame and systems

BaseGame is the engine class that contains entry point for the game. It’s also container for systems. BaseGame manages lifetime of systems. Following systems are implemented:

- RenderingSystem - it’s responsible for rendering
- InputSystem - responible for reading user input from keyboard
- PhysicsSystem - responsible for collision detection. It’s build on ReactPhysics3d
- UpdateManager - broadcast Update event each frame.

Factories

ResourceFactory is a template class the helps manage lifetime of resources shared by different objects in the game. It reads resource based on path and caches the result so next call with the same path uses cached value. Implemented factories:

- MeshFactory - reads meshes from .obj file.
- MaterialInstanceFactory - reads MaterialInstances
- MaterialDefinitionFactory - reads MaterialDefinitions
- CollisionShapeFactory - creates collision shapes used to check collisions

Pools

ObjectsPool is a template class that helps to create pool of objects. It’s parametrized by factory function, used to create instances of pooled objects. Class is used to manage meteors and projectiles shot by the space ship.

Events

Implementation of observer design pattern. In the game it’s used to broadcast frame based updates and to broadcast collision detection.

Configuration

Some properties of the game can be changed by editing **Game\Resources\GameConfig.txt** and **Game\Resources\BaseConfig.txt**

System

This paragraph describes details about systems in the engine

Rendering

Rendering is done by following pieces:

- MateriadDefinition - defines what fragment and vertex shader should be used
- MaterialInstance - defines what MaterialDefinition should be used and what texture should be applied
- Mesh - vertices, uvs and normals
- CameraComponent - calculated View and ProjectionMatrices
- LightSourceComponent - used to define light source
- MeshRendererComponent - render components based on Mesh and MaterialInstance
- RendererSystem - calls MeshRendererComponents to render themselves

Physics

PhysicsSystem is a wrapper for ReactPhysics3d. It’s build from following pieces:

- CollisionShapeDescription - describes properties of collider
- ColliderComponent - component attached to a GameObject, parametrized by CollisionShapDescription
- PhysicsSystem - contains collection of ColliderComponents, checks collisions and broadcast events when collision occurs

Input

Input system is build from two pieces:

- InputContext - represents context of inputs ex steering ship.
- InputSystem - contains stack of InputContexts and updates one that is on top.