Content

Intro

Space gravity 2D makes much easier to build custom 2D solar system scenes with realistic gravitational motion. It can be used for implementing various gameplay types.
Main features of this asset are: two different systems for simulating gravitational motion and tool for editing velocity vectors in unity scene window.

Two systems of gravitational simulation:

- Newtonian attraction calculation (N-body problem).
- Keplerian motion (2-body problem or 'RailMotion') is exactly predicted motion on static orbit, which behaviour is as if no outer gravitation exists. This method can be very useful for moving planets and moons whose orbits are not dynamically changing, as it has better performance and precision on long terms periods.

Every object with CelestialBody component can be setted to use one of this two types of motion, but on collision it will be always reseted to N-body type until collision ends.

Every CelestialBody on scene will be shown with arrows which can be dragged by mouse. Its most usable with editor orbits drawing activated. Arrows can represent velocity relative to body's attractor and relative to global space.

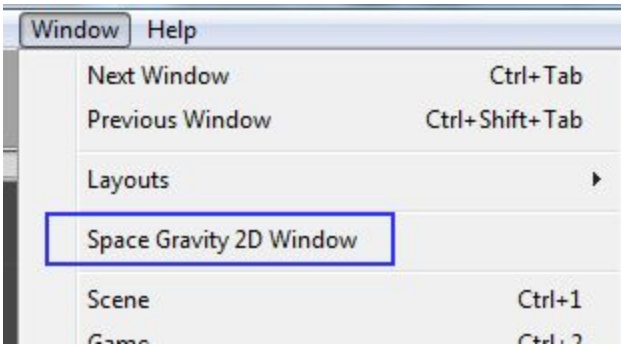
Components overview

There are two main components: CelestialBody and SimulationControl.

CelestialBody contents body parameters and must be attached to all bodies that have to move on orbits. Rigidbody2D is required.
It is functional only when object with SimulationControl component exists on same scene.

Component icon will be visible in scene view, and you can disable it in “gizmo” menu of scene window by clicking on highlighted icon

SimulationControl holds all global properties and performs simulation for scene. To edit settings is better to use Space Gravity 2D window.



When this window is opened, new SimulationControl object will be created, if its not exists yet. It is also Orbit displaying line renderer can be customized in it.

PredicitonSystem allows to predict and display orbits for all bodies on scene. All orbits are displayed in global space.
Attach this component to, for example, simulation control gameobject and setup parameters for current scene scale.
PredictionSystemTarget is optional component that allows to change prediction orbits parameters and enable/disable orbits display for certain celestial bodies.

OrbitDisplay component is second way to display orbits with separated parameters for every celestial body which it is attached to.

SphereOfInfluence is component for making zones of interest for attractors.
It is better to attach this component to child of CelestialBody, as it will create its own circle collider. Make sure this collider is marked as trigger.

CelestialBodyKeyboardController is created as simplest celestialbodies controller.

Step by step guide

1. Attach SimulationControl component to any gameobject on the scene or just open SpaceGravity2d window, it will be created automatically.
2. Attach CelestialBody component to gameobject.
3. Configure body parameters, scale of attached sprite or mesh object, add colliders.
4. Create second CelestialBody (it may be duplicate). Set mass lower than first body mass. Select motion type.
5. Use ‘find nearest attractor’ button in second’s body inspector or manually drag first body to Attractor property of second body to create planetary system. Button ‘make circle orbit’ may help to quick setup.
6. Configure distance between bodies, their velocities with help of editor scene arrows, Gravitational parameters and TimeScale in SpaceGravity2D window.
7. Add other bodies according to your wishes.

It is possible to not use central attractor, but in this case all bodies have to use only nonRailMotion.
Note that Attractor property in CelestialBody component is required only for RailMotion and for displaying orbits. If you prefer to not use RailMotion at all, you don't need to set Attractor for bodies.

Collisions

For allowing collisions you can use standard Collider2D components, but different types of collisions, requires setup components to achieve the correct behavior of objects, and it is quite complex topic.
First of all it is necessary to point out that keplerian bodies have the option of ignoring all collisions, it avoids any impact on their movement and make their motion static. If necessary to achieve the same behavior of dynamic newtonian bodies, you can try to use a child object with a static collider.

However, continuous collisions are not behave correctly, so for planetwalking effect often better to disable CelestialBody component on smaller objects and attract them by another system, for example, by standard component Point Effector 2D.

Scripting

To access components via script, first you need to add SpaceGravity2D namespace.
Some of methods that might be useful:

CelestialBody class:
public void AddExternalVelocity(Vector2 inputVelocity) - Apply non-gravity acceleration to body. Result is affected by simulation timescale.

public void SetAttractor(CelestialBody attr, bool checkIsInRange, bool instant) - Set attractor at the end of current frame.
if checkIsInRange is true, attractor max gravity range will be checked before assignation.
if instant is true, assignation will be executed before method return;

public Vector2[] GetOrbitPoints(int pointsCount, float maxDistanceForHyperbolicCase) - Get array of points of current orbit relative to attractor.

public Vector2[] GetOrbitPointsRestricted(int pointsCount, bool oppositeDirection, float fromAngle, float toAngle, float maxDistanceForHyperbolicCase) - Get array of points of current orbit with some additional parameters.

public void SetPosition(Vector2 newPos) - Set new world space position with orbit data update.

public void AddExternalForce(Vector2 forceVector, bool asImpulse) - Apply force to body

methods for changing orbit parameters:
public void SetEccentricity(double e)
public void SetMeanAnomaly(double m)
public void SetEccentricAnomaly(double e)
public void SetArgumentOfPeriapsis(double r)

Contacts

For support and requests feel free to write on itanksp@gmail.com