Putt Putt In Space with Universal Gravitation

Isaac McDaniel, Daniel Xing

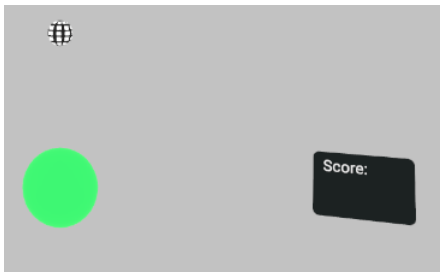
Motivation and Inspiration

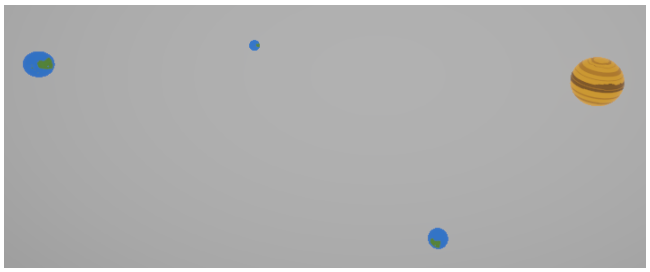
Our project is inspired by previous games that incorporate gravity into their gameplay, including the many phone-based 2D gravity simulators and Kerbal Space Program. Since VR allows the user to experience depth and interact with 3D space, we envision a game that incorporates gravity in an immersive way that's not feasible with conventional displays and controls.

Final Implementation

Environments

We have developed two play areas which contain obstacles, starting and destination planets, and the SpacePuck, which is the "golf ball" the player tries to send to the target. The first environment is a tutorial environment with just a source and destination planet and is intended to get the player used to the controls of our game. The second environment includes a mix of objects and obstacles, including three rocky planets, a gas giant, and a small white dwarf star. Masses of rocky planets are small relative to the gas giant and star so that correctly launching the SpacePuck requires precision and finesse.

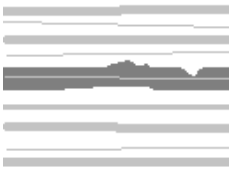


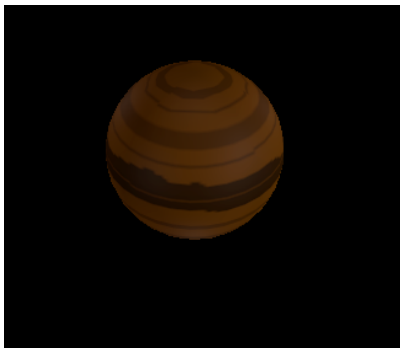


Obstacles

There are currently 2 types of obstacles, which are both prefabs which can be instantiated many times. First, the **star** prefab contains a small white sphere (radius 0.01, about the size of a bead) and six point lights which surround it on each side. Getting a good-looking star required trial and error, since placing a single light at the center of the sphere illuminated the star's surroundings but not the surface of the star. Moving to several light sources around the sphere illuminated the surface unevenly, with clear reflections of the light in the surface of the sphere. Increasing the lights to 6 and reducing the smoothness of the sphere evened out the lighting effects, allowing for a star with the same appearance when viewed from any angle. The uniform reflections make it seem to the user that the light comes from the sphere itself, instead of external sources. The star has a mass of 5.

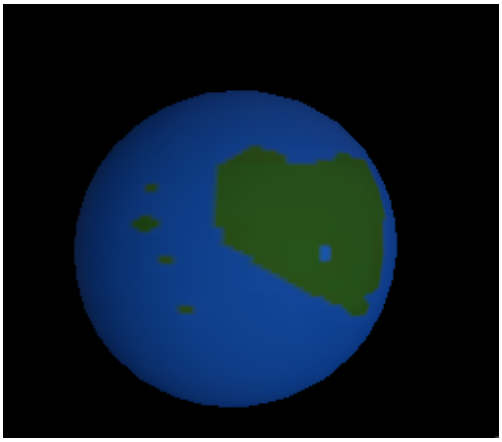
The other obstacle is a large **gas planet** (radius 0.4). This consists of a solid-colored orange sphere and a black-and-white texture which applies the signature striped pattern of gaseous planets. The texture was designed in black-and-white so that it could be quickly reused in new obstacle designs, since a sphere of any color could use this texture to instantly produce a new gas planet. As an example, though the current obstacle has an orange sphere which produces a look similar to Jupiter, a blue sphere could be used instead to prepare a planet which resembles Neptune or Uranus.





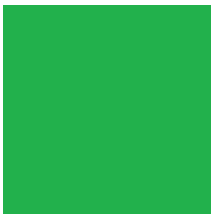
Rocky Planets

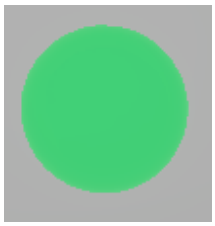
The starting planet, where the SpacePuck begins, and the destination planet, where the player must send the SpacePuck, are both represented as Earth-like, rocky planets with oceans. These use the same prefab consisting of a white sphere and a texture which paints the world map onto the sphere.These represent the starting and destination planets as well as intermediate landing and relaunch points for the player to progress along.



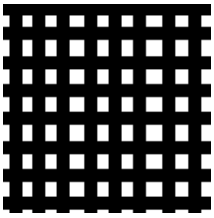
Since there will be multiple identical copies of this prefab, it is necessary to distinguish the starting and intended destination planets. To do this, we added a script which highlights these planets during the aim/launch stage by changing their material. The destination planet is given a checkered pattern, while the planet currently housing the SpacePuck becomes solid green. Since the launch phase is not implemented yet, our test script highlights the planets upon game start and reverts them to their original appearance when the ‘A’ button is pressed and released.

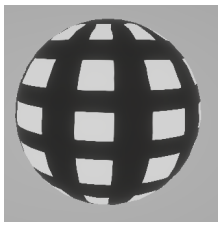
Starting Planet:





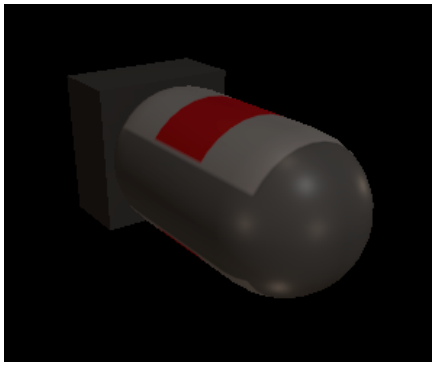
Destination Planet:





SpacePuck

The SpacePuck is the “golf ball” the player must deliver to the destination planet. It is the only object upon which gravity acts, which requires it to have a script attached. The SpacePuck’s orientation is set to always point in the direction of motion. While not necessarily realistic (breaks conservation of angular momentum), we decided that the visual effect would aid the player, and that preservation of angular momentum is not a critical feature to the game.



Skybox

Since the setting of the game is in space, it was necessary to replace Unity's default sunny blue sky. Skyboxes can be created by creating a dedicated skybox material which requires 6 textures, one for each face of the cube which encloses the virtual environment. Initially, we made textures out of images of starry skies, but the images were unrecognizably blurry in-game, so these were replaced by solid black backgrounds.

During testing, we used a black skybox to simulate space, but found that contrast between game objects and the skybox was too low and could pose problems for those with vision impairments. We decided to switch to a light gray skybox to improve celestial visibility.

Navigation

Since the play area is large and the player's physical environment may be quite small, we added a "click and drag" navigation feature. The player can move the play area around them by pressing and holding the left index trigger and moving the left controller. While the button is held down, the play area will translate (but not rotate) along with movements of the left controller. This allows the player to use their left hand to bring objects of interest into reach/view. The SpacePuck will be aimed and launched with the right hand.

Universal gravity

We use Unity's built in RigidBody.AddForce() function to implement Newtonian physics. All celestial objects that exert a gravitational force on the SpacePuck are kept in the same parent GameObject. All celestial objects contain a mass property used for calculations. Forces are calculated by iterating through all celestial objects and calculating each object's force contribution to SpacePuck. They are then applied to SpacePuck using AddForce(). In the current implementation, only SpacePuck moves, all other celestial objects are static.

To ensure decent physics accuracy, we put force calculations inside FixedUpdate(), since Unity physics calculations are also done at the same time as FixedUpdate. Additionally, we enabled motion interpolation of SpacePuck's RigidBody so that SpacePuck's motion appears smooth.

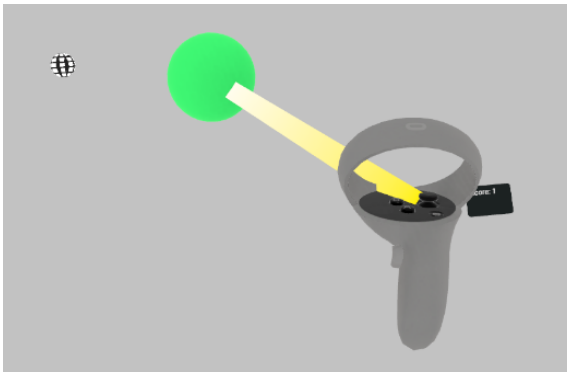
UX

We added a hop counter to keep track of the number of strokes needed to reach the destination. We considered keeping the score counter in a fixed location relative to the headset but found that this would limit the already-limited FoV of the Oculus headset, so we elected to display the score counter on a floating plane fixed in world space.

Gravity, controls

The interim version hardcoded the SpacePuck's starting position and velocity, and collisions with planets are not treated in any special way. The final version places the SpacePuck on a planet and its initial launch direction and impulse is user controlled through a pool-like interface. The starting/current planet that SpacePuck is on is highlighted in green.

To launch the SpacePuck, the player first holds the right controller near the launchable planet and presses the main right trigger. A ray connecting the source planet and the controller appears, and then the trigger is released the SpacePuck is launched in the direction of the ray at a velocity proportional to how far away the controller is from the launch planet. In flight, the SpacePuck rotates so that it is always pointing forward, in the direction of motion. Collisions with celestial objects places SpacePuck back into launch mode either on the object that SpacePuck lands on or on the planet the SpacePuck launched from, depending on whether or not the landed planet is a rocky planet.



If the SpacePuck becomes lost in space or moves for a long time without landing (this was a major problem which slowed down gameplay), we have added Relaunch and Reset buttons which will end the SpacePuck's flight. The Relaunch (A) button ends the current flight as if the SpacePuck had struck an obstacle, incrementing the stroke count and returning the SpacePuck to the previous launch point. The Reset (B) button resets the level entirely. The score is cleared to 0 and the SpacePuck is placed back on the original starting planet.

We also added a course correction system to accelerate/decelerate the SpacePuck while in space, since it's possible for the SpacePuck to end up in stable orbits around especially large planets, and will never land. To add an additional dimension of realism to the game, we allow the player to speed up/slow down the SpacePuck using the right thumbstick: pushing up increased velocity in the prograde direction and pulling back does the opposite.

Division of Work

Isaac

- Art (Planets, Obstacles, SpacePuck)
- Level Design
- UI: Launch Controls, Level Transition, Relaunch/Reset Buttons, Controller Representations

Daniel

- Gravity Scripts
- UI: Score display and next-level indicator
- In-flight course correction capability

Future work

Environment, UX

To aid the player in route planning, we could implement some sort of orbit trail and predicted path given a user's intended velocity impulse. Additionally, some sort of visualization of the 3D spheres of influence of each planet would also aid in route planning.

Gravity, controls, scoring

Given infinite time, implementing N-body physics for celestial objects so that they move like in a real planetary system would be very cool. This would require a significant investment into balancing the planetary system if planets aren't on rails or else planets could be thrown off into deep space while the player is trying to navigate the object.

The current orbital course correction system is very generous, with infinite propellant and very high thrust. A future implementation with limited propellant and thrust would add an additional dimension of skill to the game. Scoring could also take into account total propellant mass expended.