extends CharacterBody2D

# To be replaced with a system to detect the nearest planet once we instance multiple planets in the scene.

@onready var collider = $"PlayerCollider"

@onready var g = $"gravity"

var planets = {}

var current\_planet

# Player movement constants

const MASS = 10

const MAX\_FLYING\_SPEED = 2000

const FLYING\_ACCELERATION = 200

const WALK\_SPEED = 5000

const FUEL\_BURNING\_SPEED = 10

const MAX\_FUEL = 1000

const FEUL\_REFUELING\_SPEED = 20

const A\_SINGLE\_PROPEL = 10000

const MAX\_JUMP\_SCENES = 50000

# Stores all the potential states that the player can be in, will be useful later to simplify complicated tasks.

enum states {

GROUNDED,

FALLING,

FLYING

}

var current\_state = states.FLYING

# Relative directions on the current planet.

var relative\_up = Vector2.UP

var relative\_down = Vector2.DOWN

var relative\_right = Vector2.RIGHT

var relative\_left = Vector2.LEFT

var rel\_up\_last\_frame = relative\_up

var player\_input = Vector2.ZERO

var input\_velocity = Vector2.ZERO

var gravitational\_pull = 0.0

var grounded\_gravitational\_pull = 0.0

var acceleration = Vector2.ZERO

# Fuel System

var fuel = MAX\_FUEL

#Jump System

var jump\_scenes = 0

func \_process(delta):

# Sets the rotation to be facing the right way relative to what is up on the planet's surface.

rotation = Vector2.UP.angle\_to(relative\_up)

func \_physics\_process(delta):

# absolute gravitational influence

var gravitational\_force = set\_gravity()

# player input

match current\_state:

states.GROUNDED:

relative\_up = (global\_position - current\_planet.global\_position).normalized()

relative\_down = -relative\_up.normalized()

relative\_left = Vector2(-relative\_up.y, relative\_up.x).normalized()

relative\_right = -relative\_left.normalized()

velocity = velocity.rotated(velocity.angle\_to(current\_planet.position))

rotation = Vector2.UP.angle\_to(relative\_up)

move\_grounded(delta)

velocity = relative\_down \* grounded\_gravitational\_pull + relative\_left \* (player\_input.x \* WALK\_SPEED)

# check whether to add fuel

if current\_planet.has\_fuel:

if fuel < MAX\_FUEL:

fuel += FEUL\_REFUELING\_SPEED \* delta

fuel = clamp(MAX\_FUEL, 0, fuel)

if not is\_grounded():

current\_state = states.FALLING

position += relative\_down

propel(delta)

if Input.is\_action\_just\_pressed("jump"):

current\_state = states.FALLING

states.FLYING:

move\_floating(delta)

propel(delta)

velocity = clamp\_vector(input\_velocity, MAX\_FLYING\_SPEED) \* delta

states.FALLING:

move\_floating(delta)

propel(delta)

velocity = (clamp\_vector(input\_velocity, MAX\_FLYING\_SPEED) + gravitational\_force) \* delta

if is\_grounded():

current\_state = states.GROUNDED

print("Fuel Amount: ", fuel)

print("jump\_scenes: ", jump\_scenes)

print("player\_input: ", player\_input)

move\_and\_slide()

# set gravity direction based on planet influence

func set\_gravity():

var gravity = Vector2.ZERO

for i in planets:

var direction = (planets[i].global\_position - global\_position).normalized()

var influence = ((25000 \* MASS \* planets[i].MASS) / (position.distance\_squared\_to(planets[i].position) \* 2))

gravity += direction \* influence

#print(gravity.length())

g.target\_position = gravity

return gravity

# Manipulates velocity according to gravitational formula.

func handle\_gravity(delta):

gravitational\_pull = ((2000 \* MASS \* current\_planet.MASS) / position.distance\_squared\_to(current\_planet.position)) \* delta

# Manipulates velocity according

func move\_floating(delta):

player\_input = Vector2(Input.get\_axis("move\_left", "move\_right"), Input.get\_axis("move\_up", "move\_down")).normalized()

if (fuel > 0) && (not player\_input == Vector2.ZERO):

fuel -= FUEL\_BURNING\_SPEED \* delta

if fuel > 0:

input\_velocity += player\_input \* FLYING\_ACCELERATION

func propel(delta):

if Input.is\_action\_just\_pressed("jump"):

fuel -= 5

jump\_scenes += A\_SINGLE\_PROPEL

jump\_scenes = clamp(MAX\_JUMP\_SCENES, 0, jump\_scenes)

if not jump\_scenes == 0:

match current\_state:

states.GROUNDED:

velocity += jump\_scenes \* jump\_scenes \* player\_input

states.FALLING:

velocity += jump\_scenes \* jump\_scenes \* player\_input

states.FLYING:

velocity += jump\_scenes \* jump\_scenes \* player\_input

jump\_scenes -= 10

# Manipulates velocity according to the player's lateral input (left, right).

func move\_grounded(delta):

player\_input.x = Input.get\_axis("move\_left", "move\_right") \* delta

# Checks if the player is grounded using the floor collision raycast.

func is\_grounded():

return $RayDown.is\_colliding()

# Clamps a vector at a given maximum length (will be useful when handling movement).

func clamp\_vector(vector, maximum\_length):

if vector.length() > maximum\_length:

return vector.normalized() \* maximum\_length

return vector

func \_on\_player\_area\_area\_entered(area):

if area.is\_in\_group("PlanetGravity"):

if current\_state != states.FALLING:

current\_state = states.FALLING

var p = area.get\_parent()

planets[p.get\_name()] = p

elif area.is\_in\_group("OrientField"):

current\_planet = area.get\_parent()

if current\_state != states.GROUNDED:

current\_state = states.GROUNDED

func \_on\_player\_area\_area\_exited(area):

if area.is\_in\_group("PlanetGravity"):

var p = area.get\_parent()

planets.erase(p.get\_name())

if planets.is\_empty():

current\_state = states.FLYING

elif area.is\_in\_group("OrientField"):

current\_planet = null

current\_state = states.FALLING