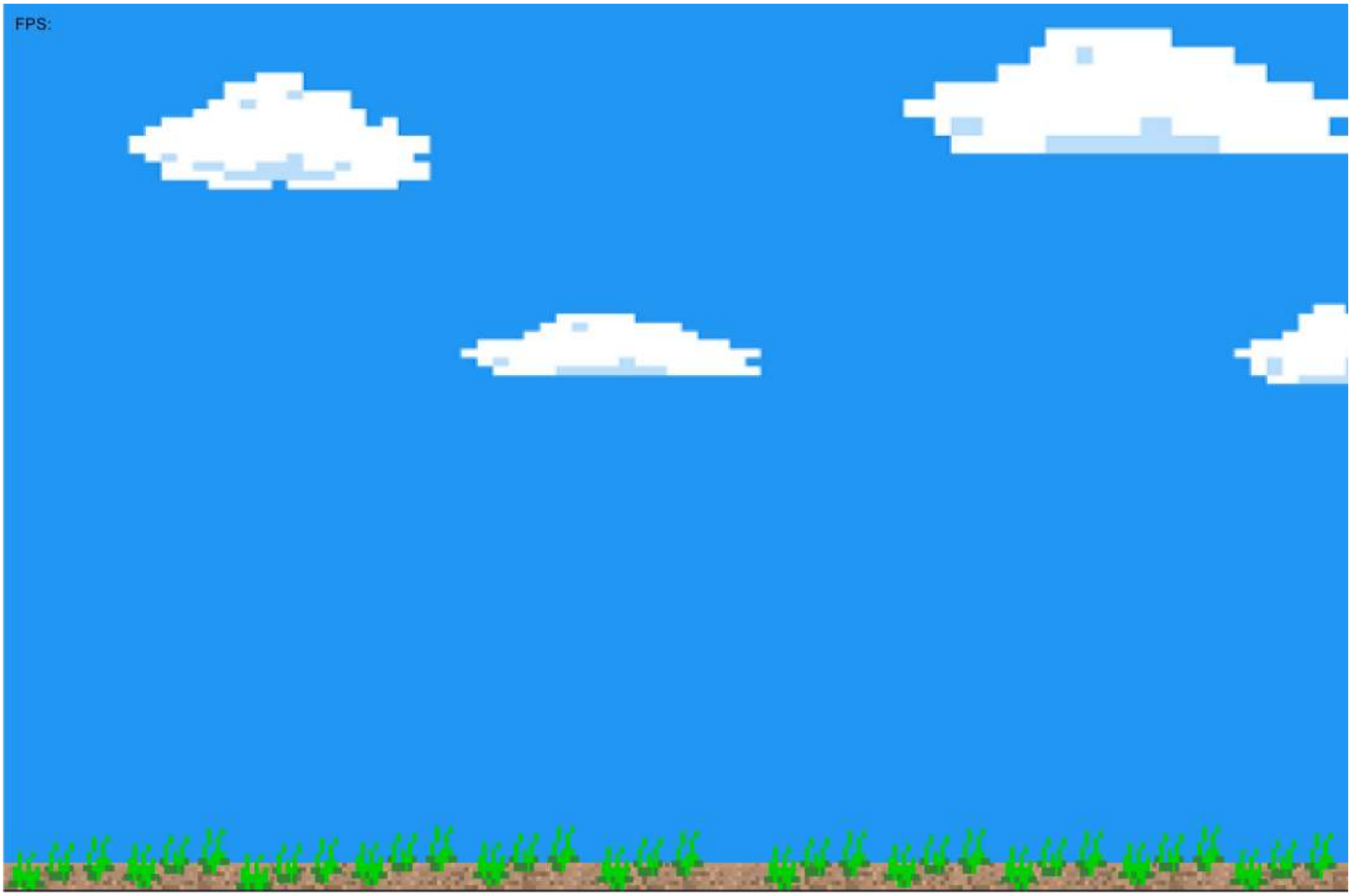
Contents


```
%FARM EQUIPMENT SIMULATOR 2021
%BUY NOW!

%Created by:
%Adam Rike             | Scoring, project management, website management
%Jonah Robles          | Sprites and background, docs
%Ranga Rutiser Sundar  | Physics and input, title/pause/crash screens, docs

%gameMain handles all game logic and the main game loop. It refers to
%Const.m for constants that need to be set.
function gameMain
```

```
clear
clc

%All constants are in Const.m.
```

Initialize Game with SpriteKit

```
%Initialize Game object
G = SpriteKit.Game.instance('Title', 'Farm Equipment Simulator 2021', 'Size', Const.windowSize);

%Initialize Background object
if Const.debugBlueScreenBG
    %If blue screen is enabled, do that
    bkg = SpriteKit.Background(Const.blueScreenBGImg);
else
    bkg = SpriteKit.Background(Const.backgroundImg);
end

bkg.Scale = Const.backgroundScale;
```



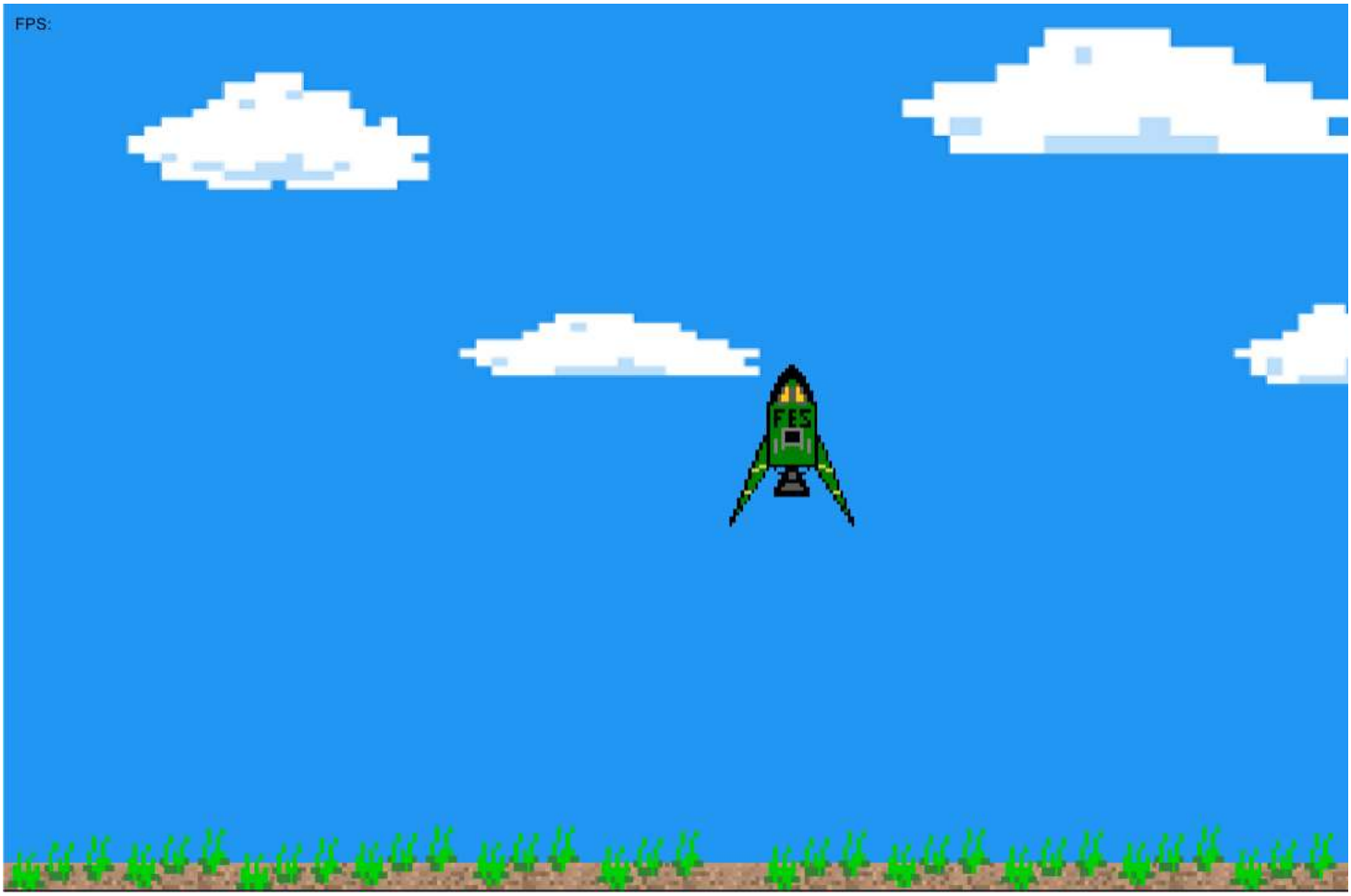
Create Sprites

```
%{
SpriteKit Framework Crash Course:
The SpriteKit Framework is used here to display most of the game on
screen.
We first create a Game object that will handle displaying our Sprites and
Background and creating a MATLAB Figure.
We then create a Background object with a .png image. It can be scrolled in
four directions and will automatically tile the image to fill the Figure
window.
We then create Sprite objects by passing in a .png image. Sprites are used
to represent the rocket, the cows, and the title/pause/crash screens.
The Sprite objects have a few built-in properties:
-Location: Location is a (x,y) vector in pixels that represents where the
image should be drawn on screen. If we change this in action(), SpriteKit
takes care of moving the Sprite.
-Angle: Angle is a scalar value in degrees that represents the rotation of
the Sprite. SpriteKit will take care of displaying the image properly if we
change this value.
-State: Different States can show different images. This is used to enable
and disable the cows (they have a state where they are invisible, and we
check state when they collide with the rocket - if they are invisible, we
do nothing.)

We also add our own custom properties using the addprop() function. This
allows us to store any value we want as a property of the Sprite object.
Using this method, the rocket can remember its fuel mass, gravity, thrust,
and more. Virtually all values used for physics are stored to the Sprite
this way.
%}

%Rocket and cow (gameplay sprites)
%createRocket and createCow take care of initializing the different states
%and properties for the rocket and cow sprites and return a Sprite object.
%They retrieve values from the Const file to refer to (for example, initial
%states or what images to use).
```

Create rocket Sprite object

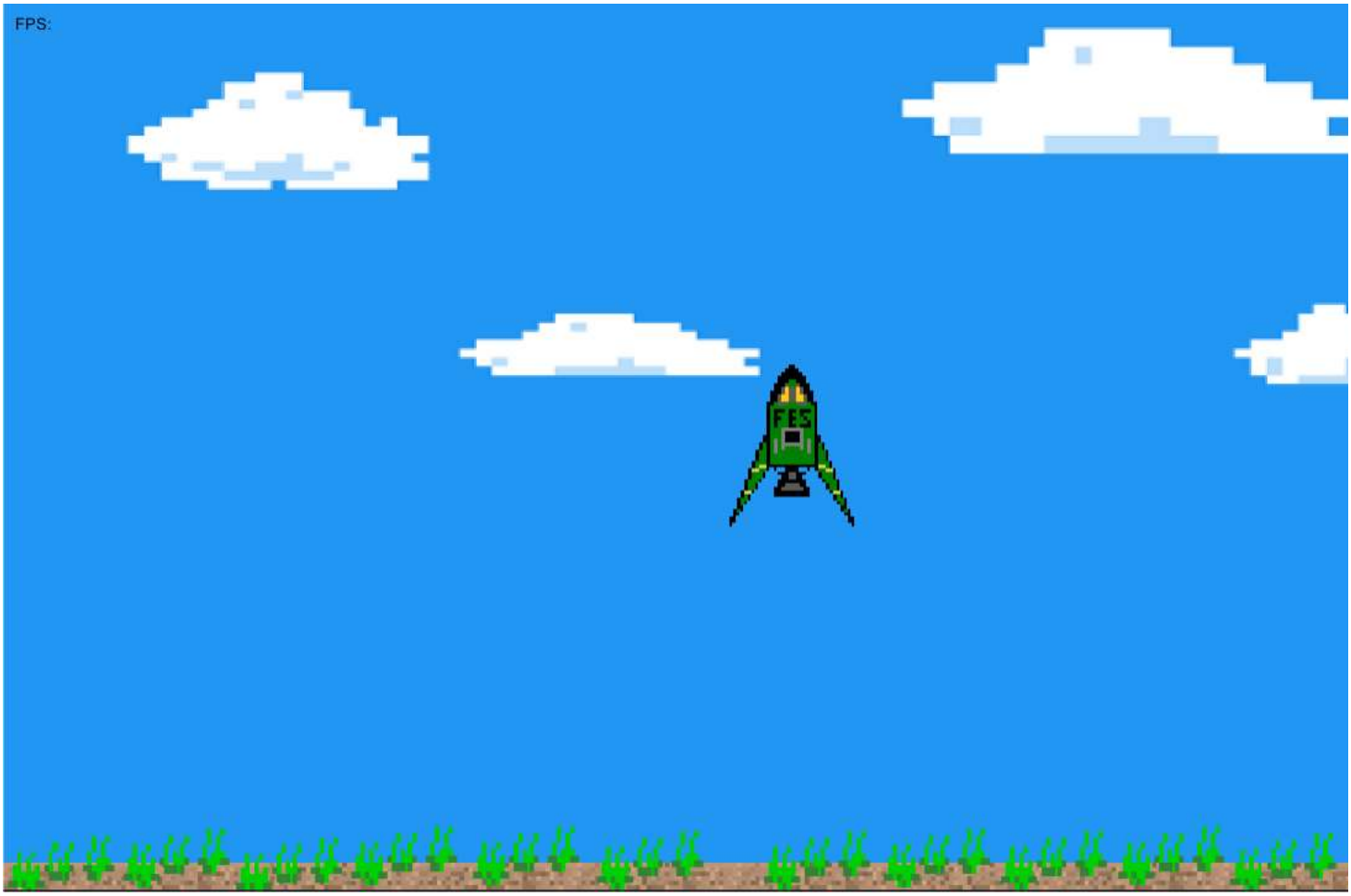
```
rocket = SpriteKit.Sprite('rocket');

%Set up the default state for the rocket and give it its image
```

```
rocket.initState('hide', Const.noneImg, true); %hide rocket if needed
rocket.initState('crash', Const.crashedRocketImg, true); %exploded rocket

%Different throttle states. These allow displaying a different sized
%frame depending on the thrust value.
rocket.initState('thrust0', Const.rocketImg, true); %Cut throttle (no flame)
rocket.initState('thrust1', Const.rocketThrust1Img, true); %Low throttle
rocket.initState('thrust2', Const.rocketThrust2Img, true); %Mid throttle
rocket.initState('thrust3', Const.rocketThrust3Img, true); %High throttle

%Give the rocket its necessary properties. These are per rocket in case
%multiplayer is wanted in the future or in case multiple rocket models
%would be wanted.
addprop(rocket, 'velocity'); %1x2 velocity meters per second
addprop(rocket, 'propMass'); %propellant mass kg
addprop(rocket, 'dryMass'); %dry mass kg
addprop(rocket, 'mass'); %total mass, kilograms
addprop(rocket, 'fuelRate'); %fuel rate at max throttle kg/s
addprop(rocket, 'maxThrust'); %maximum thrust, N
addprop(rocket, 'throttle'); %throttle, 0 to 1
addprop(rocket, 'throttleBuffer'); %key buffer for throttle keys
addprop(rocket, 'rotBuffer'); %key buffer for rotation keys
addprop(rocket, 'altitude'); %altitude, meters
addprop(rocket, 'maxPropMass'); %maximum prop mass, kg
addprop(rocket, 'specialBuffer'); %key buffer for pause, etc. keys
%Game state is stored in the rocket so that it is easier to access for
%other functions. As they are being passed the rocket anyway, it allows
%them to read the game state without needing to pass it in explicitly.
%Game state is stored as a string and regulates whether physics are
%processed and what is shown on screen.
addprop(rocket, 'gameState'); %game state: "play" "pause" "crash" "title" "tut1"
addprop(rocket, 'zeroAltLocPx'); %zero altitude location in pixels
addprop(rocket, 'score'); %game score

%Initialize rocket physics and give it its values
rocket.Location = Const.startingPosition;
rocket.velocity = Const.startingVelocity;
rocket.State = 'thrust0';
rocket.Scale = Const.rocketScale;
rocket.altitude = Const.startingAltitude;
rocket.propMass = Const.startingPropMass;
rocket.dryMass = Const.dryMass;
rocket.mass = Const.startingPropMass + Const.dryMass;
rocket.fuelRate = Const.fuelRate;
rocket.maxThrust = Const.maxThrust;
rocket.throttle = Const.startingThrottle;
rocket.maxPropMass = Const.maxPropMass;
rocket.gameState = Const.startingGameState;
rocket.zeroAltLocPx = rocket.Location(2) - rocket.altitude * Const.pixelsPerMeter;
rocket.score = 0;
rocket.Depth = Const.foregroundDepth;

%These need to be initialized to 0 (empty) because they store control
%inputs.
rocket.throttleBuffer = 0;
rocket.rotBuffer = 0;
rocket.specialBuffer = 0;

%Timer for flame animation if used.
flameAnimTimer = 0;
```



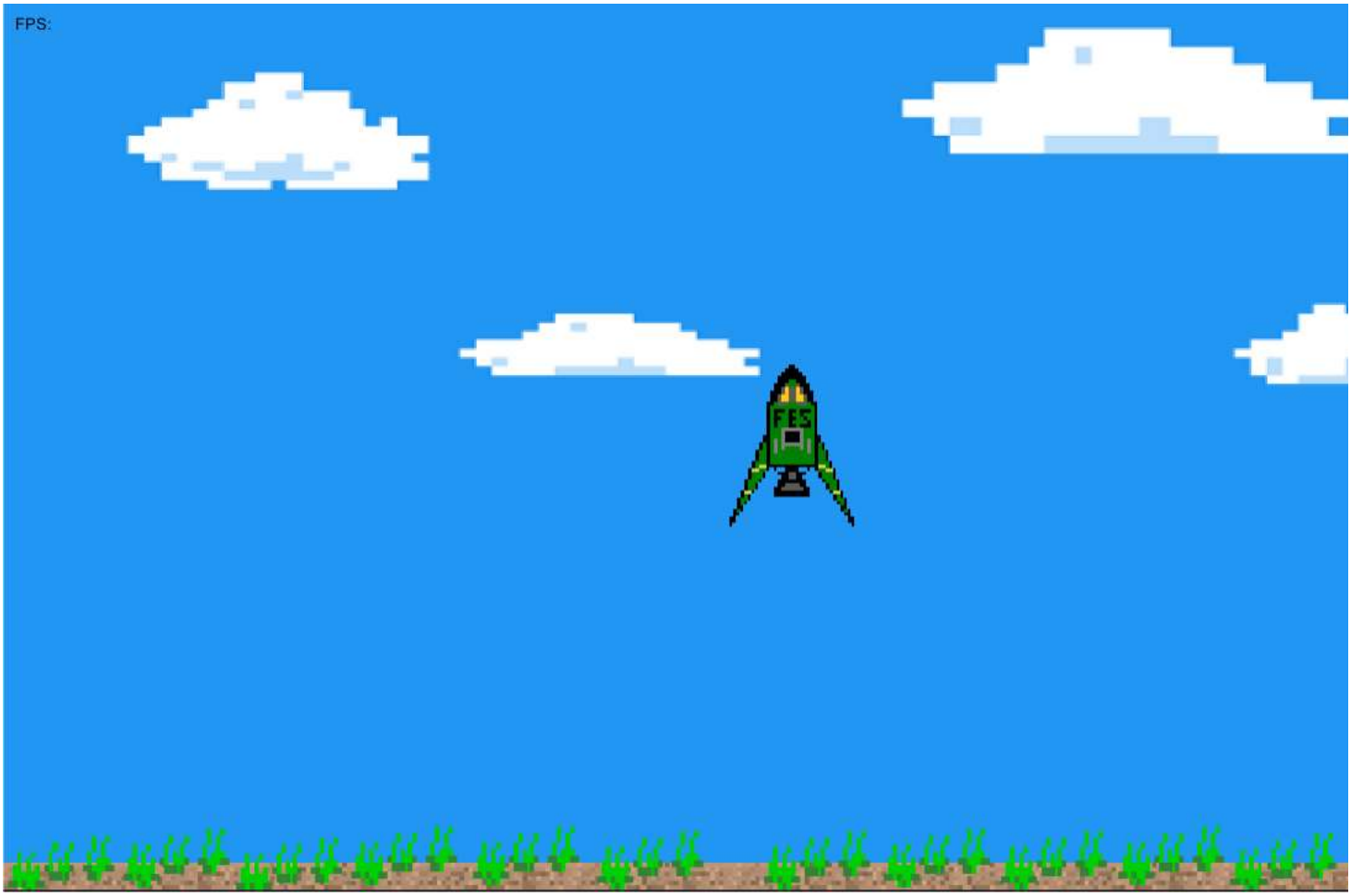
Create cow Sprite object

```
cow = SpriteKit.Sprite('cow');

%Set up state
cow.initState('on', Const.cowImg, true);
cow.initState('off', Const.noneImg, true); %when the cow has been collected, this allows it to disappear
cow.initState('fly', Const.cowFlyImg, true); %flying cows!
cow.initState('tractor', Const.crashedTractorImg, true); %when your cow is a tractor.

%Give it its properties
addprop(cow, 'propAmt'); %amount of propellant in the cow, kg
addprop(cow, 'xToNextCow'); %distance until next cow, meters
cow.State = 'off'; %Disable the cow to start
cow.Scale = Const.cowScale; %set the scale of the sprite (shouldn't really be needed)
cow.Depth = Const.foregroundDepth; %set to foreground

cow.propAmt = Const.cowPropMass; %Amount of propellant in the cow
cow.xToNextCow = randi(Const.cowRandVals); %meters since last cow collected
```



Set up exhaust manager and title sprite.

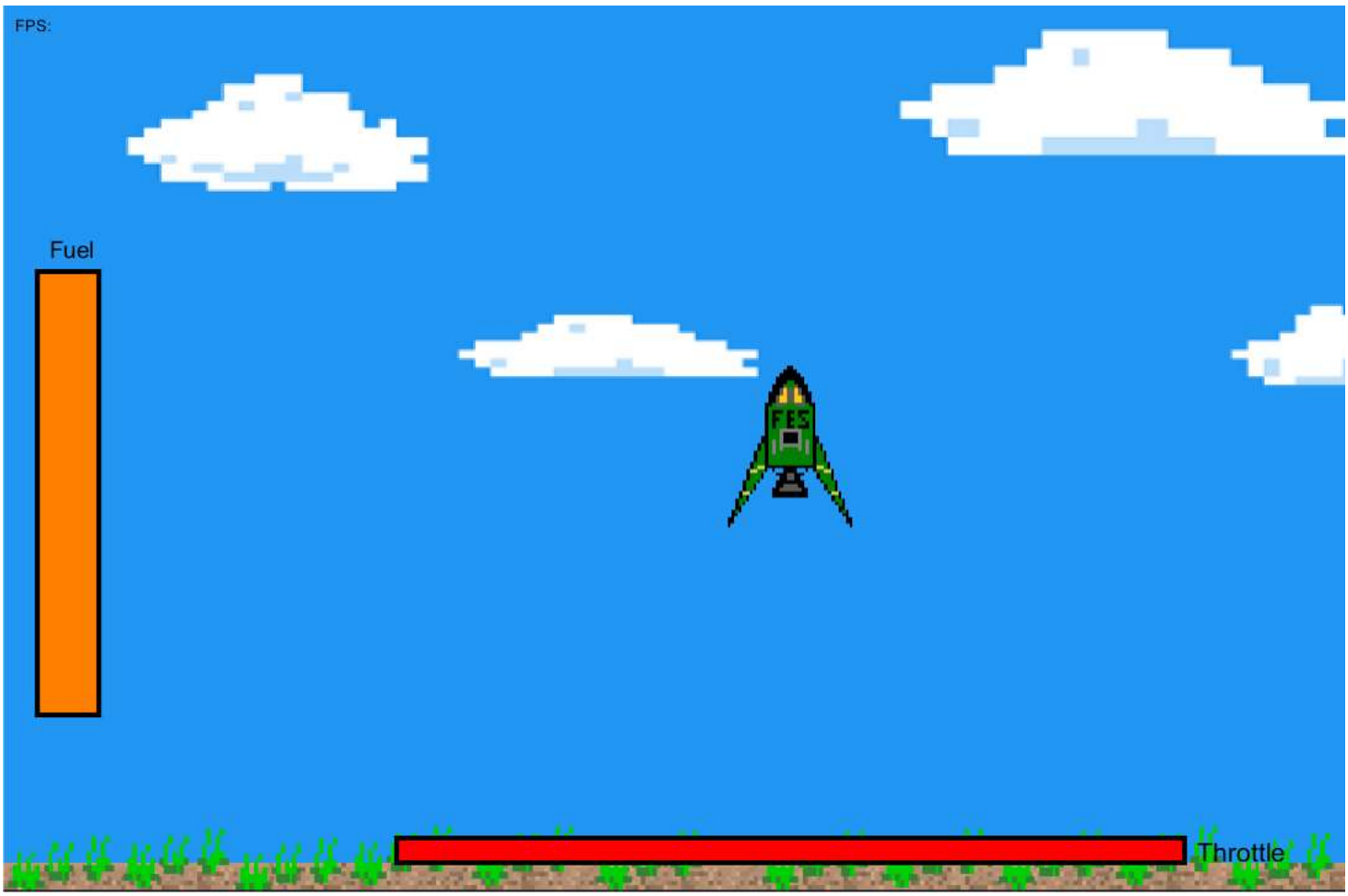
```
%Create the exhaust manager. This object handles updates and physics for
%all exhaust particles.
exhaustMgr = ExhaustMgr(rocket);

%Having these on different depths results in collision not being processed
%rocket.Depth = 0;
%cow.Depth = 0;

%Title/pause screen sprite. Whenever the title, pause, or crash screens are
%visible, this sprite is taking up the whole screen.
titleSprite = SpriteKit.Sprite('title');

%Initialize states for the title sprite.
titleSprite.initState('titleScreen', Const.titleScreenImg, false);
titleSprite.initState('pauseScreen', Const.pauseScreenImg, true);
titleSprite.initState('crashScreen', Const.crashScreenImg, true);
titleSprite.initState('tut1', Const.tutorial1Img, false);
titleSprite.initState('hide', Const.noneImg, true);

titleSprite.Depth = 100; %Make sure this is always on top of all other sprites
```



Create non-SpriteKit UI elements

```
%For these UI elements, it's easier to use MATLAB's built in functions to
%draw them.

%Create fuel gauge
%Outline rectangle that shows the maximum size
fuelGaugeRect = rectangle('Position',...
    [Const.fuelGaugeOffset, Const.fuelGaugeWidth,...
    Const.fuelGaugeMaxHeight]);

%Set the fuel gauge height based on how much propellant the rocket has so
%that it is accurate on the first frame.
fuelGaugeHeight = Const.fuelGaugeMaxHeight *...
    rocket.propMass / rocket.maxPropMass;

%Fill rectangle that shows how much propellant you have. Its height is
%changed as the propellant is consumed.
fuelFillRect = rectangle('Position',...
    [Const.fuelGaugeOffset, Const.fuelGaugeWidth,...
    fuelGaugeHeight]);

%Set colors for fill and edge and set line widths
fuelFillRect.FaceColor = Const.fuelGaugeFillColor;
fuelFillRect.EdgeColor = Const.fuelGaugeEdgeColor;
fuelFillRect.LineWidth = Const.fuelGaugeLineWidth;

fuelGaugeRect.EdgeColor = Const.fuelGaugeEdgeColor;
fuelGaugeRect.LineWidth = Const.fuelGaugeLineWidth;

%Draw the label for the fuel gauge
fuelText = text(Const.fuelTextX, Const.fuelTextY, "Fuel");
fuelText.FontSize = 14;

%Create throttle gauge outline rectangle
throtGaugeRect = rectangle('Position', Const.throtGaugeRectPos);
throtGaugeRect.EdgeColor = Const.throtGaugeEdgeColor;
throtGaugeRect.LineWidth = Const.throtGaugeLineWidth;

%Create throttle gauge fill rectangle
throtFillRect = rectangle('Position', Const.throtGaugeRectPos);
throtFillRect.Position(3) = rocket.throttle * Const.throtGaugeWidth;
throtFillRect.EdgeColor = Const.throtGaugeEdgeColor;
throtFillRect.FaceColor = Const.throtGaugeFillColor;
throtFillRect.LineWidth = Const.throtGaugeLineWidth;
```

```
%Throttle gauge label text
throtText = text(Const.throtTextX, Const.throtTextY, "Throttle");
throtText.FontSize = 16;

%Altitude text, initialized to blank. It will be updated when playing.
altitudeText = text(Const.altTextX, Const.altTextY, "");
altitudeText.FontSize = 16;

%Text that shows the game score. It will be updated when playing.
scoreText = text(Const.scoreTextX, Const.scoreTextY, "");
scoreText.FontSize = 16;

%Debug text for debugging. It will be updated when playing.
debugText = text(Const.debugTextX, Const.debugTextY, "");
```



Run Game

```
%Set up the key buffering system. Whenever a key is pressed, this function
%is called. It will store a value in the rocket's buffer variables
%depending on what key is pressed, and on the next frame the game will read
%this value to determine what to do.
G.onKeyPress = {@bufferKeys, rocket};

%Hide the title sprite
titleSprite.State = 'hide';

%Specifying @action tells G.play what to run every frame. In this case,
%action handles everything for the game other than displaying the sprites
%and background, which are handled by the SpriteKit Framework.
G.play(@action); %Run game
```

Action to be run every frame

```
function action

    %Different game states.
    %Valid game states: 'title', 'play', 'pause', 'crash'
    %Note that once title is left it is not designed to be returned to.
    switch rocket.gameState

        case 'tut1' %tutorial page 1 (if more pages are added)
```

Tutorial Page

```
            exhaustMgr.killAllParticles(); %kill all particles so they are not visible.
            rocket.State = 'hide'; %hide the rocket
            cow.State = 'off'; %hide the cow
            titleSprite.State = 'tut1'; %show tutorial page

            %Hide fuel gauge
            fuelGaugeRect.Visible = 'off';
            fuelFillRect.Visible = 'off';
            fuelText.Visible = 'off';

            throtGaugeRect.Visible = 'off';
            throtFillRect.Visible = 'off';
            throtText.Visible = 'off';

            altitudeText.Visible = 'off';
            scoreText.Visible = 'off';

            %Take special buffer key inputs
            switch rocket.specialBuffer
                case 2 %Spacebar (go to title)

                    %Update fuel gauge fill height
                    fuelGaugeHeight = Const.fuelGaugeMaxHeight * rocket.propMass / rocket.maxPropMass;
                    fuelFillRect.Position(4) = fuelGaugeHeight;

                    %Set the rocket's game state so that next frame the
                    %game will go to title
                    rocket.gameState = 'title';
                case 3 %q key, stop game
                    G.stop(); %stop gameplay execution
                    close(gcf); %Close the current figure (the game)
            end

            %Clear the key buffers
            rocket.throttleBuffer = 0;
            rocket.rotBuffer = 0;
            rocket.specialBuffer = 0;
```

```
        case 'title' %When on the title screen
```

Title Screen

```
            exhaustMgr.killAllParticles(); %kill all particles so they are not visible
            rocket.State = 'hide'; %hide the rocket
            cow.State = 'off'; %hide the cow
            titleSprite.State = 'titleScreen'; %show title screen

            %Hide fuel gauge
            fuelGaugeRect.Visible = 'off';
            fuelFillRect.Visible = 'off';
            fuelText.Visible = 'off';

            throtGaugeRect.Visible = 'off';
            throtFillRect.Visible = 'off';
            throtText.Visible = 'off';

            altitudeText.Visible = 'off';
            scoreText.Visible = 'off';

            %Take special buffer key inputs
            switch rocket.specialBuffer
                case 2 %Spacebar (start game)
                    %Update fuel gauge fill height
                    fuelGaugeHeight = Const.fuelGaugeMaxHeight * rocket.propMass / rocket.maxPropMass;
                    fuelFillRect.Position(4) = fuelGaugeHeight;

                    %Set the rocket's game state so that next frame the
                    %game will start
                    rocket.gameState = 'play';
                    rocket.State = 'thrust0'; %don't be invisible
                case 3 %q key, stop game
                    G.stop(); %stop gameplay execution
                    close(gcf); %Close the current figure (the game)
                case 4 %h key, go to tutorial
                    rocket.gameState = 'tut1';
            end

            %Clear the key buffers
            rocket.throttleBuffer = 0;
            rocket.rotBuffer = 0;
            rocket.specialBuffer = 0;
```

```
        case 'pause' %When paused
```

Pause Screen

```
            titleSprite.State = 'pauseScreen';
            %Check for resuming or quitting the game.
            switch rocket.specialBuffer
                case 1 %esc key, resume game
                    rocket.gameState = 'play';
                    rocket.specialBuffer = 0;

                case 3 %q key, quit game
                    G.stop();
                    close(gcf);
            end
```

```
        case 'play' %In gameplay
```

```
            %Hide title/pause screen
            titleSprite.State = 'hide';

            %Make fuel gauge visible unless hide UI is enabled
            if ~Const.debugHideUI
                fuelGaugeRect.Visible = 'on';
                fuelFillRect.Visible = 'on';
                fuelText.Visible = 'on';

                %Make throttle gauge visible
                throtGaugeRect.Visible = 'on';
                throtFillRect.Visible = 'on';
                throtText.Visible = 'on';
                altitudeText.Visible = 'on'; %altimeter text
                scoreText.Visible = 'on'; %Score text
            else
                %Hide UI
                fuelGaugeRect.Visible = 'off';
                fuelFillRect.Visible = 'off';
                fuelText.Visible = 'off';

                throtGaugeRect.Visible = 'off';
                throtFillRect.Visible = 'off';
                throtText.Visible = 'off';

                altitudeText.Visible = 'off';
                scoreText.Visible = 'off';
            end
```

Rocket Physics

```
            %This isn't necessary
            %rocket.State = 'thrust0'; %Show the rocket

            %Check for pausing the game.
            if rocket.specialBuffer == 1
                rocket.gameState = 'pause';
                rocket.specialBuffer = 0;
            end

            %Debug key handling
            if rocket.specialBuffer == 99
                %Forcibly spawn a cow to test spawning behavior
                if Const.debugForceSpawnCow
                    cow.State = 'off'; %turn the cow off so it can be respawned
                    %spawnCow takes care of enabling the cow and moving it
                    %to its spawn location.
                    spawnCow(rocket, cow);
                end
            end

            %The magnitude of the rocket's velocity
            rocketSpeed = sqrt(rocket.velocity(1)^2 + rocket.velocity(2)^2);
            %If the rocket goes below the crash altitude, set the game
            %state to crashed.
            if (rocket.Location(2) <= Const.crashAlt)
                if rocketSpeed >= Const.crashSpeed
                    rocket.gameState = 'crash';
                else
                    rocket.Location(2) = Const.crashAlt;
                    rocket.velocity(2) = 0;
                end
            end

            rocket.specialBuffer = 0;

            %Read key inputs from the buffer and update throttle

            %w is increase, s is decrease
            switch(rocket.throttleBuffer)
```

```
    case 1 %z key, max throttle
        throttleVal = 1;
    case 2 %x key, cut throttle
        throttleVal = 0;
    case 3 %w key, increase throttle
        %Only if incremental throttle is enabled
        if Const.incThrottle
            throttleVal = rocket.throttle + Const.throttleInc * Const.frameTime;
        else
            throttleVal = rocket.throttle;
        end
    case 4 %s key, reduce throttle
        if Const.incThrottle
            throttleVal = rocket.throttle - Const.throttleInc * Const.frameTime;
        else
            throttleVal = rocket.throttle;
        end
    otherwise
        throttleVal = rocket.throttle;
end

%Constrain throttle between 0 and 1.
if throttleVal > 1
    throttleVal = 1;
end

if throttleVal < 0
    throttleVal = 0;
end

%Assign the working value to the output
rocket.throttle = throttleVal;
rocket.throttleBuffer = 0; %Clear the throttle buffer

%Handle rotation inputs. Right is d and left is a.
switch(rocket.rotBuffer)
    case 2 %d key, rotate right
        rotationVal = rocket.Angle - Const.rotationInc * Const.frameTime;
    case 1 %a key, rotate left
        rotationVal = rocket.Angle + Const.rotationInc * Const.frameTime;
        %fprintf("rotating left, rotation is now %f deg\n", rotation);
    otherwise
        rotationVal = rocket.Angle;
end

%Constrain rotation between 0 and 360 by adding or subtracting 360.
while rotationVal >= 360
    rotationVal = rotationVal - 360;
end

while rotationVal < 0
    rotationVal = rotationVal + 360;
end

%Assign the working value to the output.
rocket.Angle = rotationVal;
rocket.rotBuffer = 0; %clear rotation buffer

%Handle engine physics: check for out of propellant, calculate thrust vector,
%and subtract propellant.
if rocket.propMass <= 0
    rocket.propMass = 0;
    thrust_v = [0,0];
else
    %Calculate thrust
    thrust_s = rocket.maxThrust * rocket.throttle; %scalar thrust value, Newtons
    thrust_v = [thrust_s * sind(rocket.Angle),...
        thrust_s * cosd(rocket.Angle)]; %vector thrust force, Newtons

    %amount of propellant consumed this frame
    propConsumed = rocket.fuelRate * rocket.throttle * Const.frameTime;

    %subtract consumed propellant
    rocket.propMass = rocket.propMass - propConsumed; %kg

    %Make sure the rocket's propellant doesn't go into the
    %negative
    if rocket.propMass < 0
        rocket.propMass = 0;
    end

end

%Update the rocket's total mass with the new propellant mass
rocket.mass = rocket.propMass + rocket.dryMass; %kg

%Calculate the forces acting on the rocket
gravityForce = [0, rocket.mass * Const.gravity]; %force of gravity, Newtons

netForce = thrust_v + gravityForce; %net force, Newtons
```

```
%a = f/m
acceleration = netForce / rocket.mass; %acceleration, m/s^2

%dv = a*dt
rocket.velocity = rocket.velocity + acceleration * Const.frameTime; %m/s

%Rough approximation of friction. Does the job.
rocket.velocity = rocket.velocity * Const.frictionMultiplier; %m/s

%dx = v * dt
delta_pos = rocket.velocity * Const.frameTime; %change in position this frame, meters

rocket.altitude = rocket.altitude + delta_pos(2); %increment altitude

%The altitude is above the ground level, so it is offset here
%to display properly.
rocket.Location(2) = rocket.altitude * Const.pixelsPerMeter + Const.zeroAlt;
```

Check for crashes or landing

```
%Landing is commented out for now because it was not working
%correctly.
%The magnitude of the rocket's velocity
%rocketSpeed = sqrt(rocket.velocity(1)^2 + rocket.velocity(2)^2);
%If the rocket goes below the crash altitude, set the game
%state to crashed.
if (rocket.Location(2) <= Const.crashAlt)
    %if rocketSpeed >= Const.crashSpeed
        rocket.gameState = 'crash';
    %elseif rocket.velocity(2) < 0
    %   rocket.Location(2) = Const.crashAlt;
    %   rocket.velocity(2) = 0;
    %end
end
```

Scoring and Display

```
if Const.flameAnim
    flameAnimTimer = flameAnimTimer + 1;

    if (rocket.throttle <= Const.throttle0cutoff) || rocket.propMass <= 1
        rocket.State = 'thrust0'; %engine off
    elseif flameAnimTimer >= Const.flameAnimTime
        flameAnimTimer = 0;
        switch rocket.State
            case "thrust0"
                rocket.State = 'thrust3';
            case "thrust1"
                rocket.State = 'thrust2';
            case "thrust2"
                rocket.State = 'thrust3';
            case "thrust3"
                rocket.State = 'thrust1';
        end
    end

end
if ~Const.flameAnim
    %Show the rocket throttle states: If the rocket throttle is
    %between certain values, it shows a different size flame.
    %Also if the rocket is out of propellant
    if (rocket.throttle <= Const.throttle0cutoff) || rocket.propMass <= 1
        rocket.State = 'thrust0'; %Engine off
    elseif rocket.throttle <= Const.throttle1cutoff
        rocket.State = 'thrust1'; %Small flame
    elseif rocket.throttle <= Const.throttle2cutoff
        rocket.State = 'thrust2'; %Medium flame
    else
        rocket.State = 'thrust3'; %Large flame
    end
end

%Update the particles
exhaustMgr.updateParticles([delta_pos(1), 0]);

%Update altitude text
altitudeText.String = sprintf("Altitude: %.0f m", rocket.altitude);

%Update game score display
%If rocket is below scoring threshold, count score
if (rocket.altitude < Const.altitudeScoreCutoff)
    %Score due to angle (flatter is better)
    angleScoreAdd = cosd(rocket.Angle) * (Const.angleMultiplier);
    %Score due to altitude (lower is better)
    altScoreAdd = (Const.altitudeScoreCutoff - rocket.altitude) * (Const.altitudeMultiplier);
    %Add to rocket score
    rocket.score = rocket.score + angleScoreAdd + altScoreAdd;
end
```

```
scoreText.String = sprintf("Score: %.0f", rocket.score);

%Update fuel gauge
%Calculate the height of the gauge based on how much propellant
%the rocket has
fuelGaugeHeight = Const.fuelGaugeMaxHeight * rocket.propMass / rocket.maxPropMass;
fuelFillRect.Position(4) = fuelGaugeHeight;

%Update throttle gauge
%width based on throttle value
throtFillWidth = Const.throtGaugeWidth * rocket.throttle;
throtFillRect.Position(3) = throtFillWidth;
```

Background

```
%The scroll function requires a positive value. We store change
%in position as a signed value, so we need to figure out its
%direction and then take the absolute value.

%Scroll the background horizontally. It requires a positive value, so we do this.
if delta_pos(1) < 0
    bkg.scroll('right', abs(delta_pos(1) * Const.pixelsPerMeter));
elseif delta_pos(1) > 0
    bkg.scroll('left', abs(delta_pos(1) * Const.pixelsPerMeter));
end

%Vertical scroll can be disabled (and is as of 12/2/21). We
%originally had vertical scrolling, so we're leaving the code
%to enable it if wanted. Some other things may not work well
%with vertical scrolling though.
if Const.backgroundVerticalScroll
    %Scroll the background vertically. It requires a positive value, so we do this.
    if delta_pos(2) < 0
        bkg.scroll('up', abs(delta_pos(2) * Const.pixelsPerMeter));
    elseif delta_pos(2) > 0
        bkg.scroll('down', abs(delta_pos(2) * Const.pixelsPerMeter));
    end
end
```

Cow Handling

```
%Add the x-movement to the cow's counter for respawning
cow.xToNextCow = cow.xToNextCow - abs(delta_pos(1));

%If the rocket has travelled far enough since the last cow or is low on fuel,
%spawn a new one
if (cow.xToNextCow <= 0) || rocket.propMass < Const.propCowPity
    spawnCow(rocket, cow);
end

if cow.State ~= "off"
    %Scroll the cow. We move it by the change in position in
    %meters. It does not move vertically.
    cow.Location = cow.Location + [1, 0] .* delta_pos * Const.pixelsPerMeter;
    %fprintf("cow location %i, %i", cow.Location(1), cow.Location(2));


    %Check that the cow is not way off screen
    if cow.Location(1) < -Const.cowKillMargin
        cow.State = 'off';
    elseif cow.Location(1) > Const.windowSize(1) + Const.cowKillMargin
        cow.State = 'off';
        %Removed kill for above and below screen
    elseif cow.Location(2) > Const.windowSize(2) + Const.cowKillMargin
        %cow.State = 'off';
    elseif cow.Location(2) < -Const.cowKillMargin
        %cow.State = 'off';
    end

    %Check the cow's collisions in case it hits the rocket.
    [collide, target] = SpriteKit.Physics.hasCollision(cow);
    if collide %If it has collided with another Sprite
        switch target.ID %What Sprite it has collided with
            %Give the rocket propellant if it hits the cow
            case 'rocket'
                %If the cow is actually a tractor, crash the
                %rocket.
                if cow.State == "tractor"
                    rocket.gameState = "crash";

                    %Only give propellant if the cow is enabled (otherwise,
                    %it's a 1x1 transparent png and its position is irrelevant
                elseif cow.State ~= "off"
                    %Give the rocket more propellant
                    rocket.propMass = rocket.propMass + cow.propAmt;

                    %Make sure the rocket doesn't exceed its maximum
                    %propellant mass
                    if rocket.propMass > rocket.maxPropMass
```

```
                                rocket.propMass = rocket.maxPropMass;
                            end

                            cow.xToNextCow = randi(Const.cowRandVals); %Reset the counter for next cow spawn
                            cow.State = 'off'; %Disable the cow
                        end
                    end
                end
            end
```

```
    case 'crash' %Rocket crashed/game over
        exhaustMgr.updateParticles([0,0]);

        %Hide fuel gauge
        fuelGaugeRect.Visible = 'off';
        fuelFillRect.Visible = 'off';
        fuelText.Visible = 'off';

        %Hide throttle gauge
        throtGaugeRect.Visible = 'off';
        throtFillRect.Visible = 'off';
        throtText.Visible = 'off';

        altitudeText.Visible = 'off'; %Hide altitude text
        scoreText.Visible = 'on'; %Show score text

        %          don't vw
        %Crash rocket and hide cow
        rocket.State = 'crash';
        %cow.State = 'off';

        titleSprite.State = 'crashScreen'; %Show crash screen

        %Handle inputs:
        %Close the window with the q key
        if rocket.specialBuffer == 3
            G.stop(); %Stops the game from processing any further
            close(gcf); %Closes the current figure (gcf = get current figure)
        end

        %Restart the game with the space key
        if rocket.specialBuffer == 2

            %Reset rocket to its starting values
            rocket.Angle = Const.startingAngle;
            rocket.Location = Const.startingPosition;
            rocket.velocity = Const.startingVelocity;
            rocket.State = 'thrust0';
            rocket.altitude = Const.startingAltitude;
            rocket.propMass = Const.startingPropMass;
            rocket.dryMass = Const.dryMass;
            rocket.mass = Const.startingPropMass + Const.dryMass;
            rocket.throttle = Const.startingThrottle;
            rocket.gameState = Const.restartGameState;
            rocket.score = 0;

            %Reset cow
            cow.State = 'off';
            cow.xToNextCow = randi(Const.cowRandVals); %meters since last cow collected

            %Clear input buffers
            rocket.rotBuffer = 0;
            rocket.throttleBuffer = 0;
            rocket.specialBuffer = 0;

        end

    end

    %This was just for debug and won't be used in normal gameplay.
    %Basically, it puts together a debug string based on all enabled debug
    %features and then displays that to screen. It doesn't process some of
    %the line returns properly but it isn't going to be used so we don't
    %need to fix that.
    debugString = "";

    if Const.debugShowRocketPos %Show the rocket's position values
       debugString = sprintf("Rocket Position %.0f, %.0f \n ", rocket.Location(1), rocket.Location(2));
    end

    if Const.debugShowGameState %Display the game state
        debugString = debugString + "Game State " + rocket.gameState + " \n";
    end

    if Const.debugShowCowPos %Display the cow's position
        debugString = debugString + sprintf("Cow Position %.0f, %.0f \n", cow.Location(1), cow.Location(2));
    end

    debugText.String = debugString; %Update the debug text to show the desired string
end
```

```
end
```

Published with MATLAB® R2021b