



Design Document

Revised: 30 April 2017

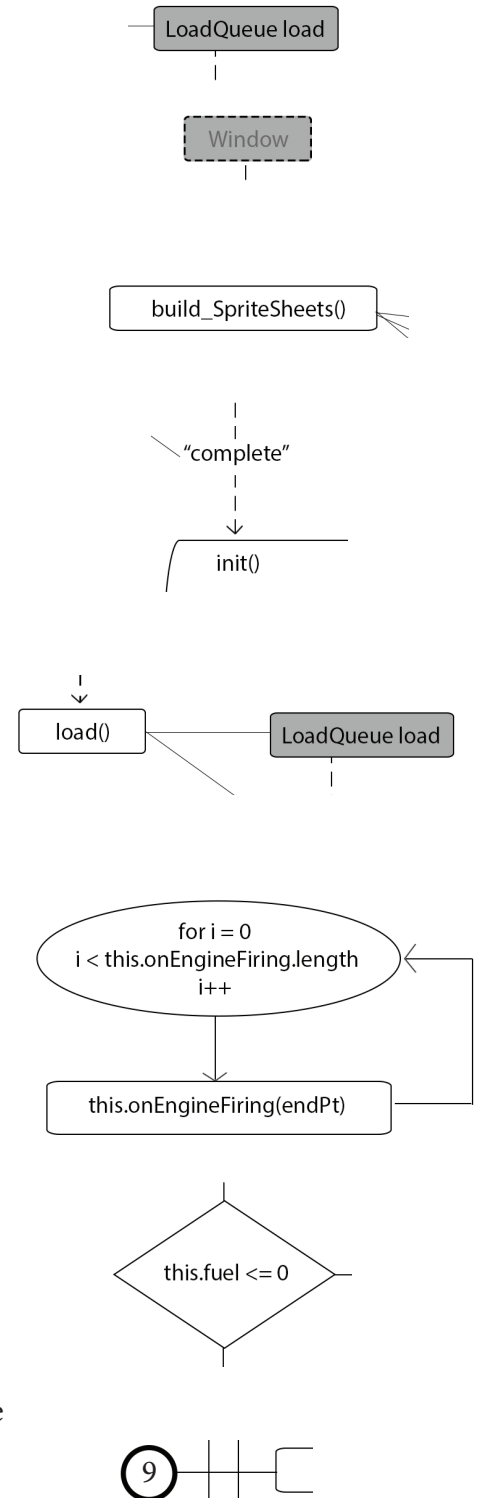
Introduction

This document contains a series of diagrams illustrating game flow for Rocket Lander!. Rocket Lander! is a JavaScript game that utilizes the CreateJS library. The game code is made up of objects, functions, and events. Each component type is represented graphically.

- Objects are greyed squares. Solid line and black text represent the initialization of the object. Dashed line and greyed text represent a reference to the existing object.
- Functions are white squares. Functions can contain properties, nested functions, or function calls.
- Event names are given in “quotation marks”.
- Dashed lines represent the relationship between an object and a function called in the case of an event. (i.e. “complete” event triggers `init()`)
- Solid lines represent a relationship between components.

For example, the `load()` function initializes the `LoadQueue` queue object, and sets up the function `init()` to trigger when the `LoadQueue` event “complete” occurs.

- Oval shapes represent loop statements.
- Diamond shapes represent decision statements.
- Circles with numerical values are considered keys, and lead to a separate diagram that provides more information for the keyed item. Numerical values correspond to the page numbers listed on the lower righthand pages of this document.



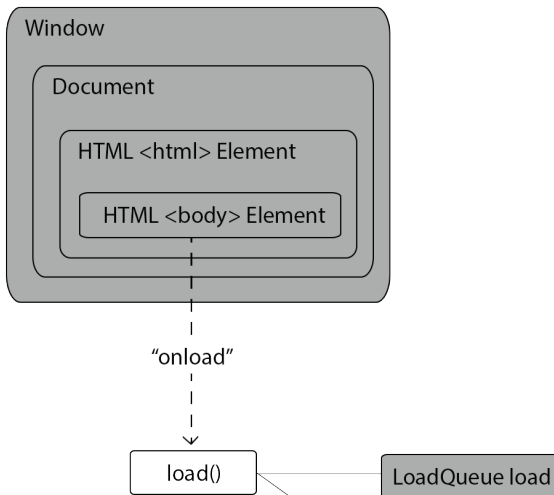


Diagram 1: Game Loading

The game's `load()` function is triggered by the "onload" event of the HTML <body> Element. It initializes a `LoadQueue` object, which loads all images needed for the game.

Upon completion of this loading, `init()` function is called. This function initializes all of the global game objects, sets the user controls, displays the loading screen, and starts the game in a gameover state.

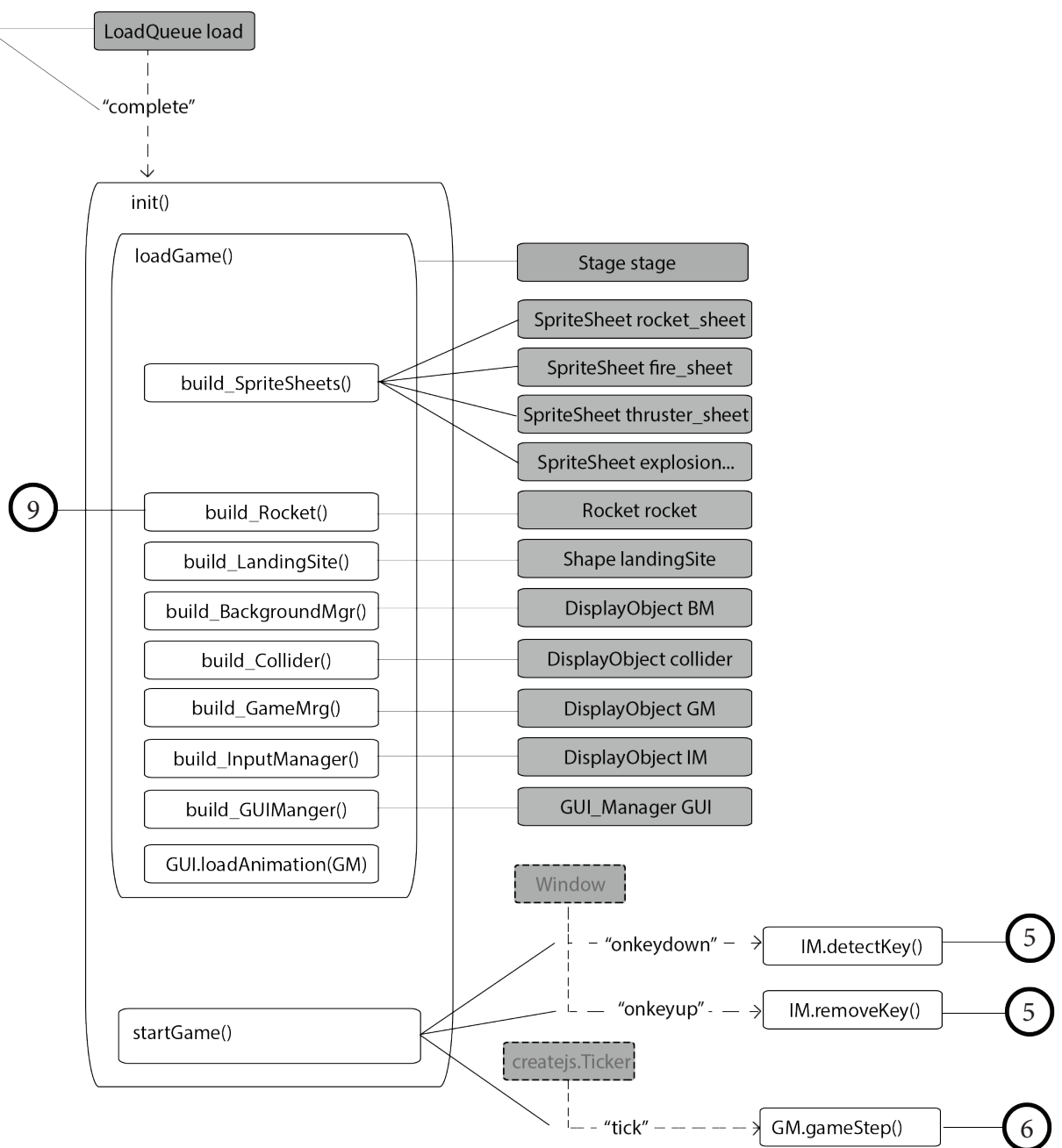


Diagram 2: Input Manager

Input Manager is an encapsulated object with two functions: detectKey() and removeKey()

IM.detectKey(e) When window detects “onkeydown” event, it triggers this function.

IM.removeKey(e) When window detects “onkeyup” event, it triggers this function.

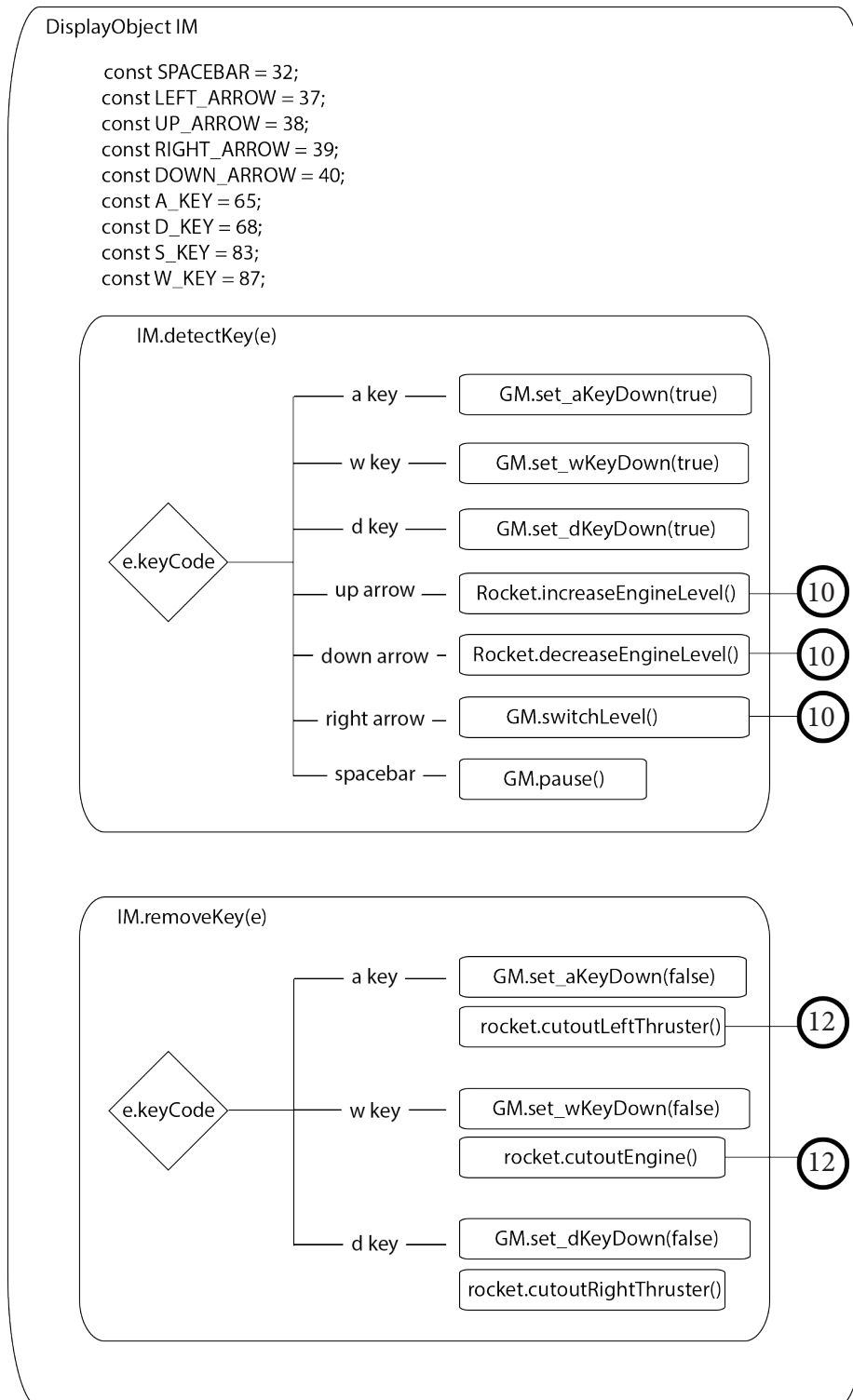


Diagram 3: GM.gameStep()

Function checks whether createjs.Ticker is paused. If it is not paused, checks whether game is over. If game is not over, function updates and renders the game.

Whether or not it is gameover, stage is updated.

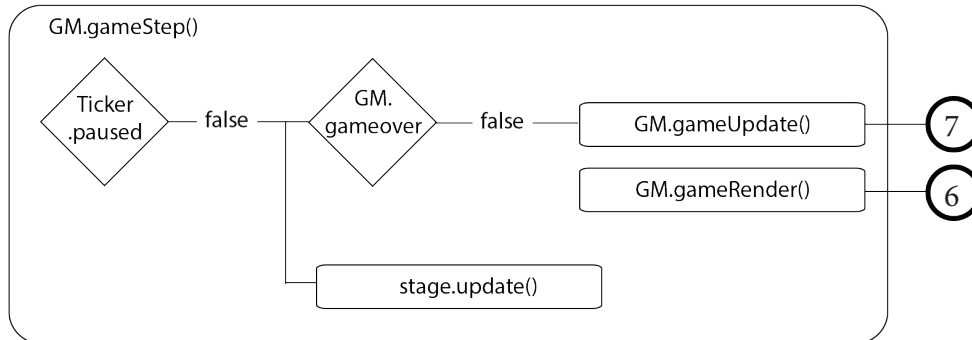


Diagram 4: GM.renderGame()

Function calls methods of game objects to render their updated positions. Currently the function only renders the rocket.

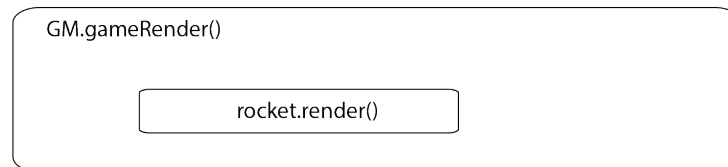


Diagram 5: GM.restartGame()

Function starts a counter to ensure Tween and reset is only called once. Gameover value is set to true to prevent updateGame and renderGame from triggering. Function creates a Tween that performs a non-visible change to a global object in the game, in order to create a 2500 ms timing event. After waiting for this amount of time, Tween calls `GM.reset()` which resets rocket values and uses `build_Rocket()` to reposition existing object.

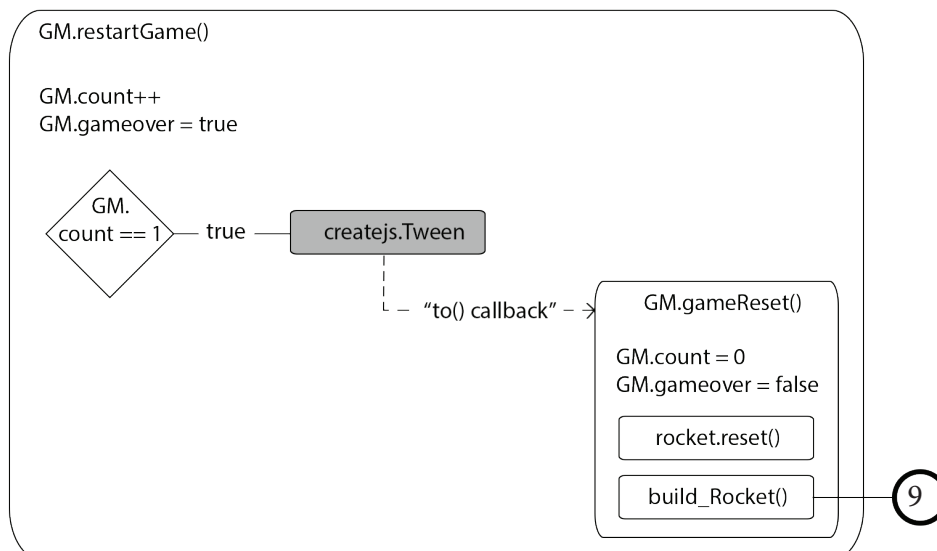


Diagram 6: GM.updateGame()

Game manager contains the variables wKeyDown, aKeyDown, dKeyDown. In updateGame(), checks are done to see whether a given key is currently being held down, which would trigger specific rocket functions.

Game manager then performs update for rocket, collider, and gui manager.

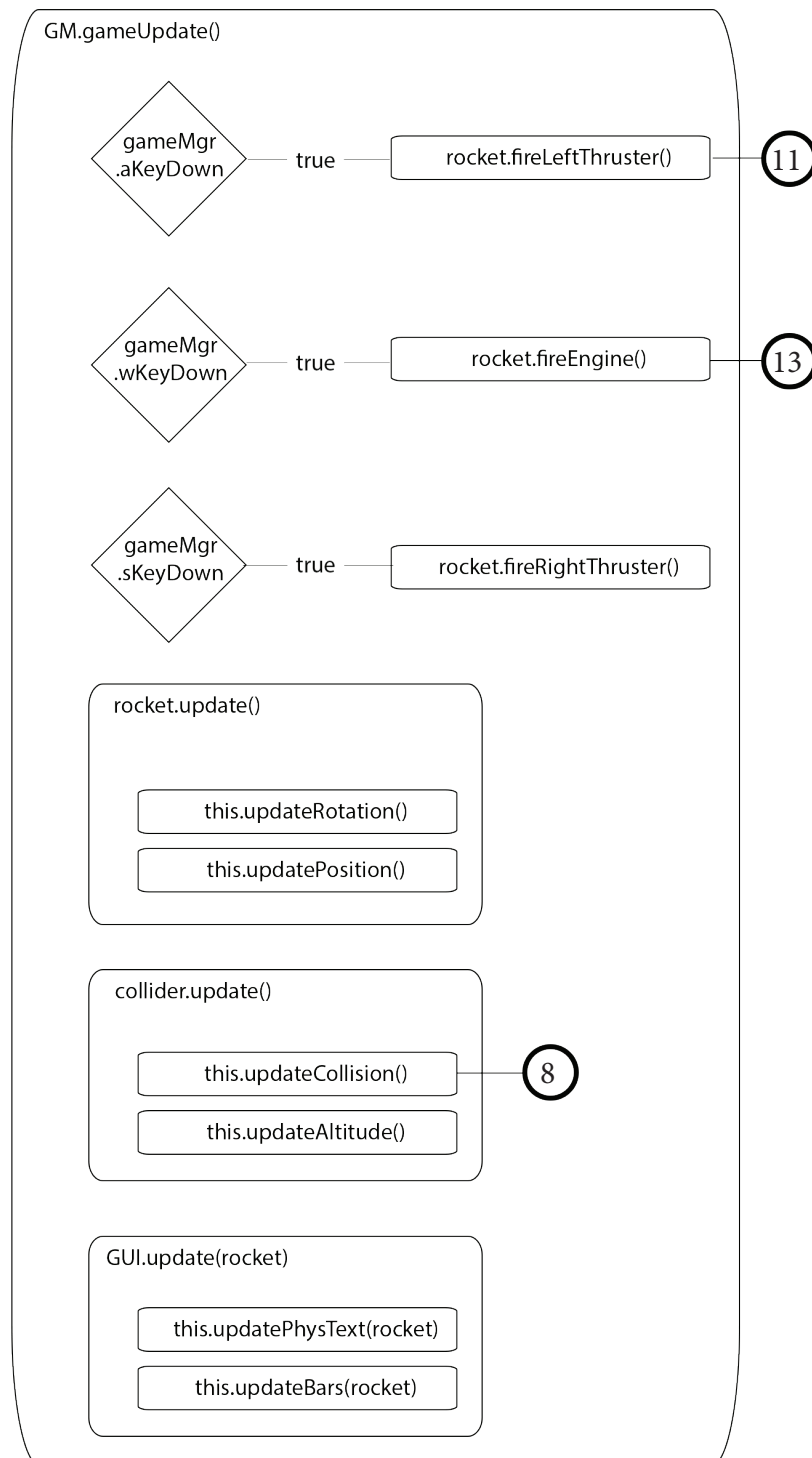


Diagram 7: collider.updateCollision()

Collider keeps track of the altitudes of rocket and landingSite objects, and triggers actions when a collision between them occurs. If bottom of the rocket's landing legs and the top of the landingSite are in the same horizontal plane, y-range is good. If the y-range is good, other parameters are checked. If rocket is landing over the landingSite, x-range is good. If the rocket's rotation, horizontal speed, and vertical speed are within a specified range, the collider can proceed with a landing event. Otherwise, proceed with a crash event.

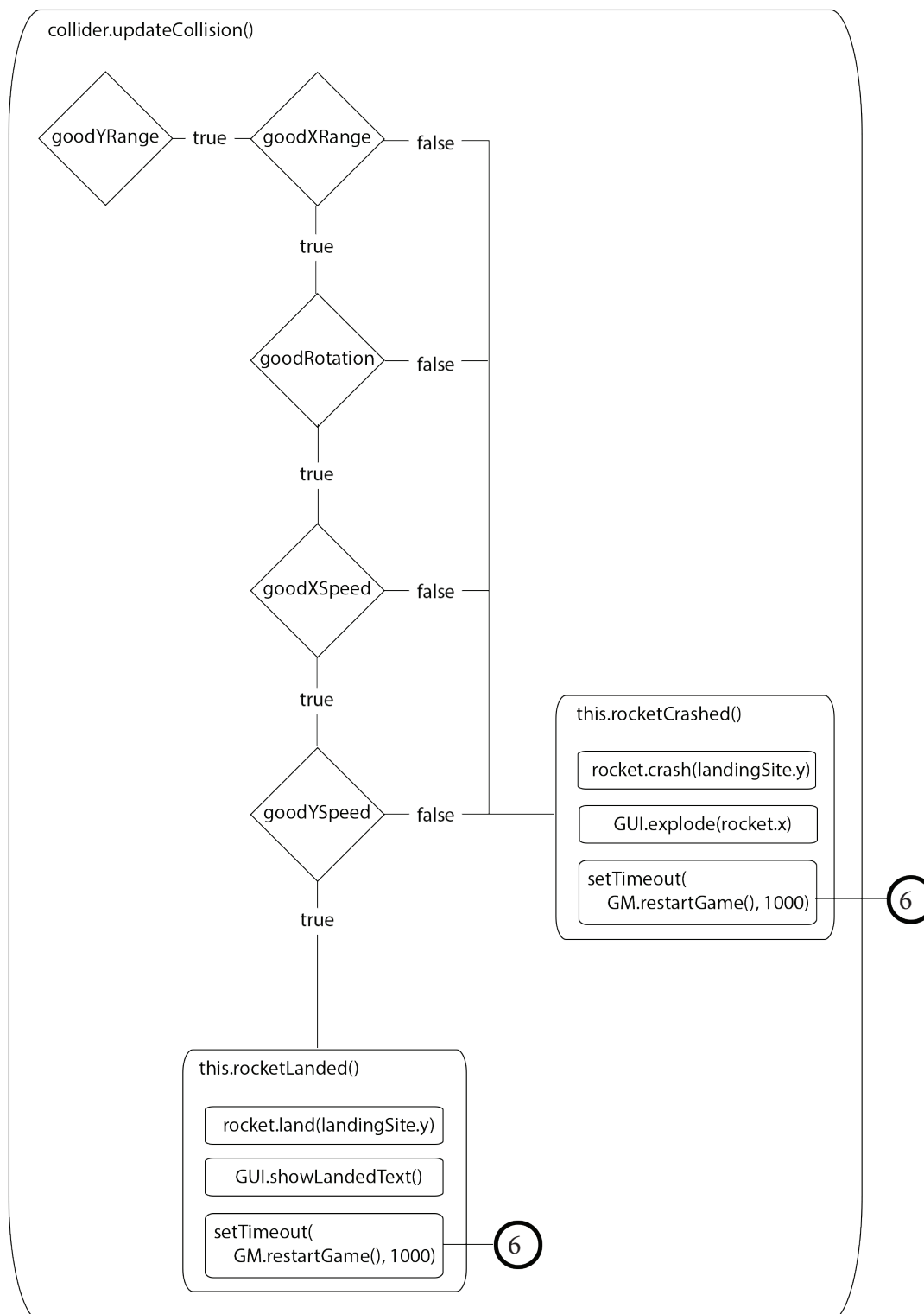


Diagram 8: build_Rocket()

This function will initialize the global rocket object and position it on the stage.

First, function calculates a random horizontal position and random angle for the rocket (the rocket begins in the same vertical position every attempt - out of view). Then if the rocket has already been initialized (in the case of a second landing attempt), the rocket is repositioned.

If the rocket has not yet been initialized, function calls the rocket constructor. Then functions are added to the rocket's event listeners, to be called for specific events that occur in the rocket. Then the rocket is positioned.

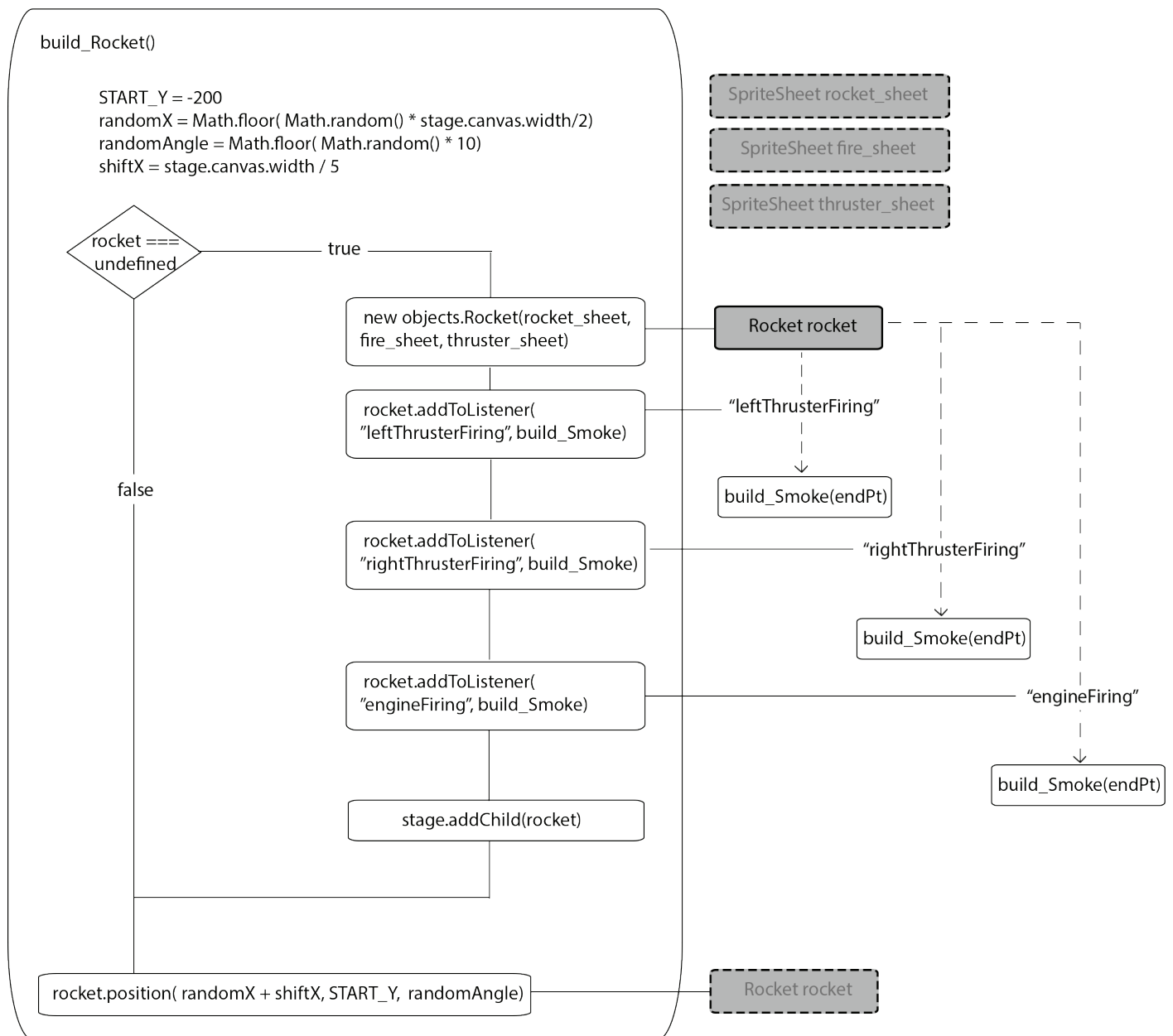


Diagram 9: GM.switchLevel()

Function toggles between the levels of the game using modular arithmetic. Then calls a function of background manager and landingSite.

Background manager has function to choose which background images to show.

Landing Site stores a reference to a graphics command object for the drawRect() command, and redraw() changes the shape of the landing site based on the level.

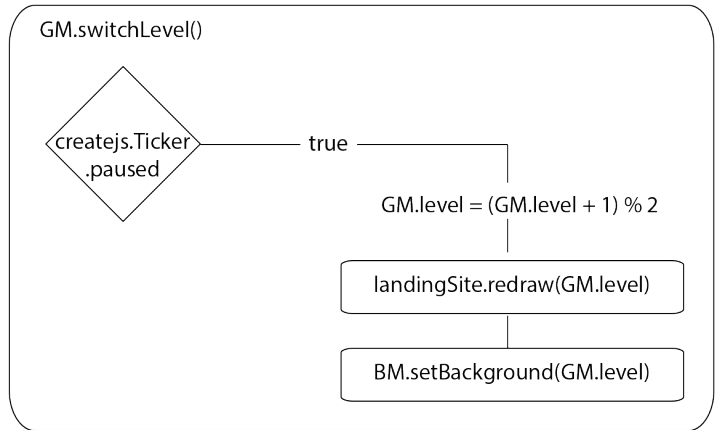


Diagram 10: rocket.increaseEngineLevel()

This function lowers the rocket's engine level property within established range.

Engine level plays a key role in rocket animations and movement.

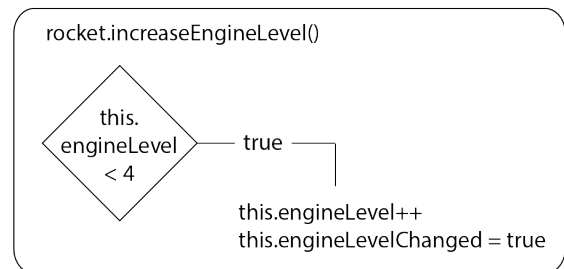


Diagram 11: rocket.decreaseEngineLevel()

This function raises the rocket's engine level property within established range.

Engine level plays a key role in rocket animations and movement.

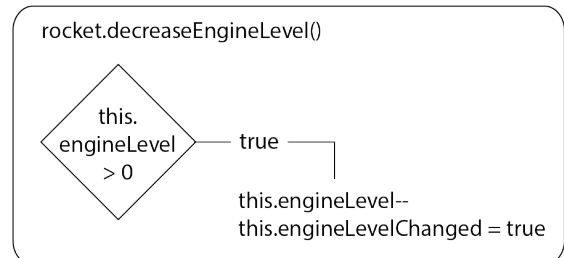


Diagram 12: rocket.fireLeftThruster()

Function checks whether this thruster is already firing and whether there is monopropellant (thruster fuel) remaining. It updates animations and torque values accordingly. Then it calls all functions that were stored in the event listener for this event. Called functions are sent a parameter representing the end point of the tip of the thruster animation.

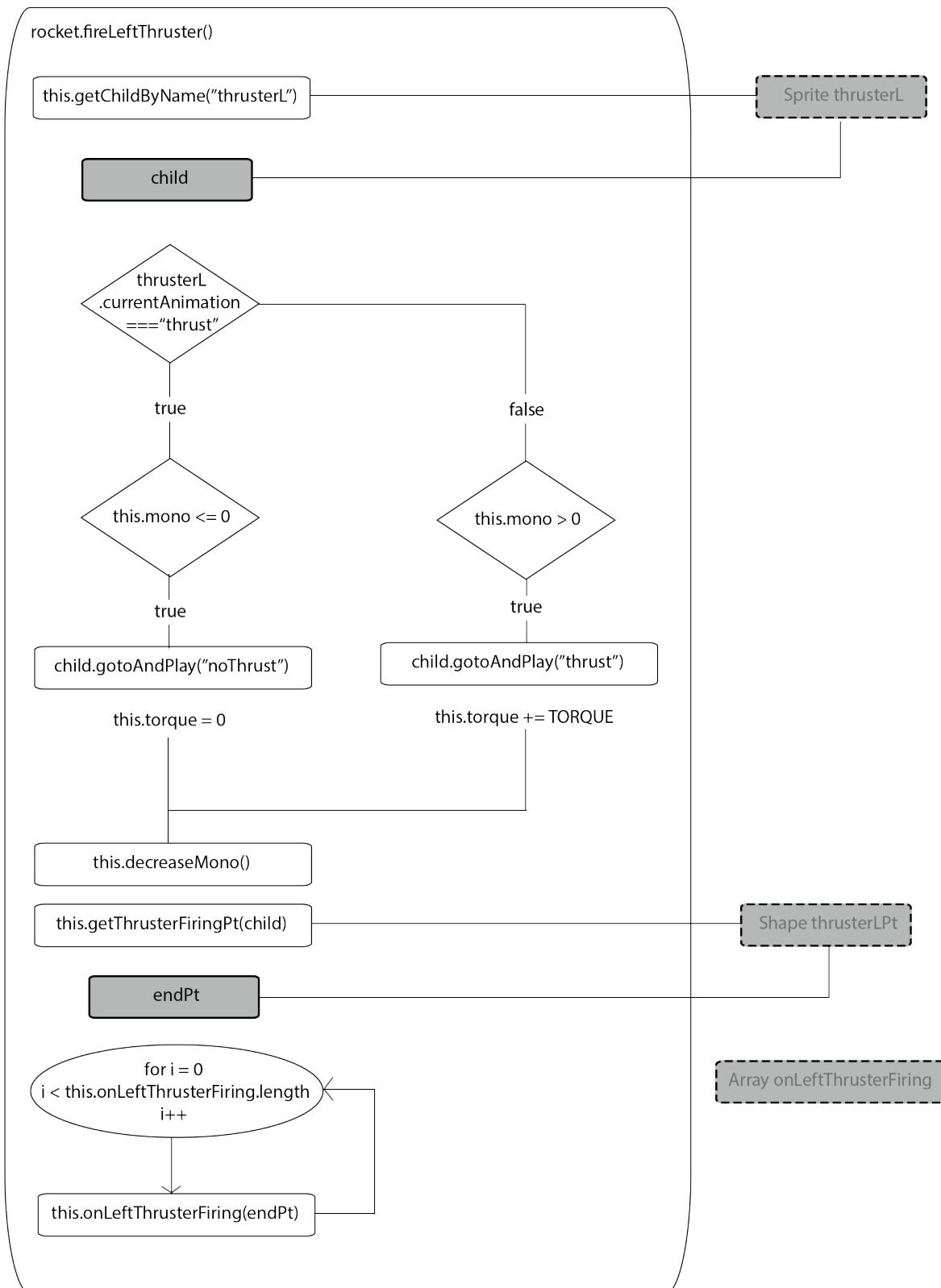


Diagram 13: rocket.cutoutLeftThruster()

Function checks whether this thruster has firing animation. If it does, it changes the animation and torque values.

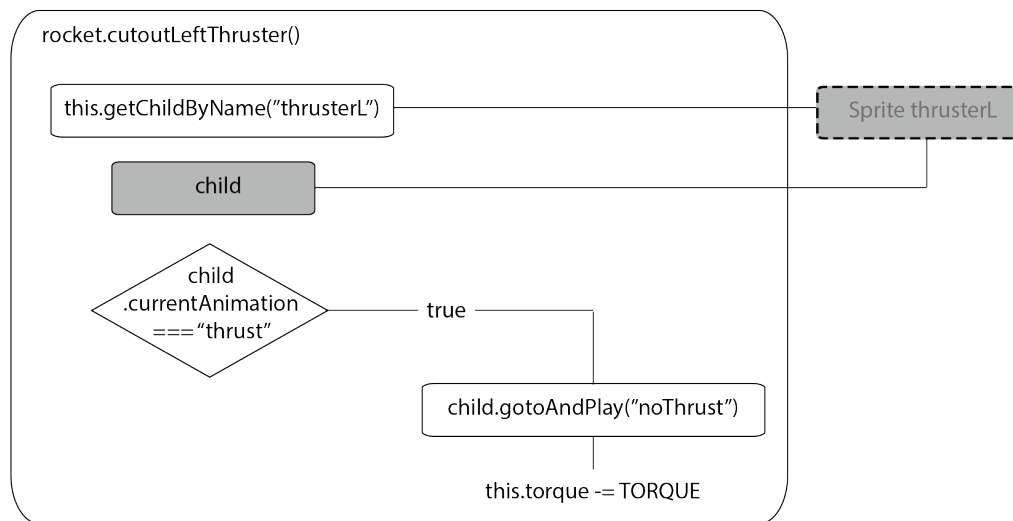


Diagram 14: rocket.cutoutEngine()

Function checks whether engine has a firing animation. If it does, it changes the animation and thrust values.

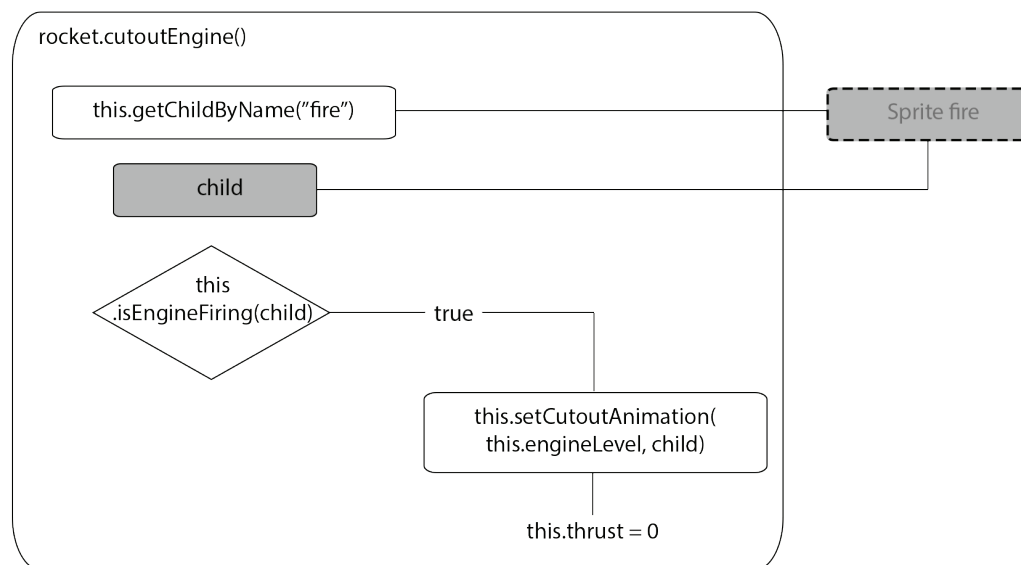
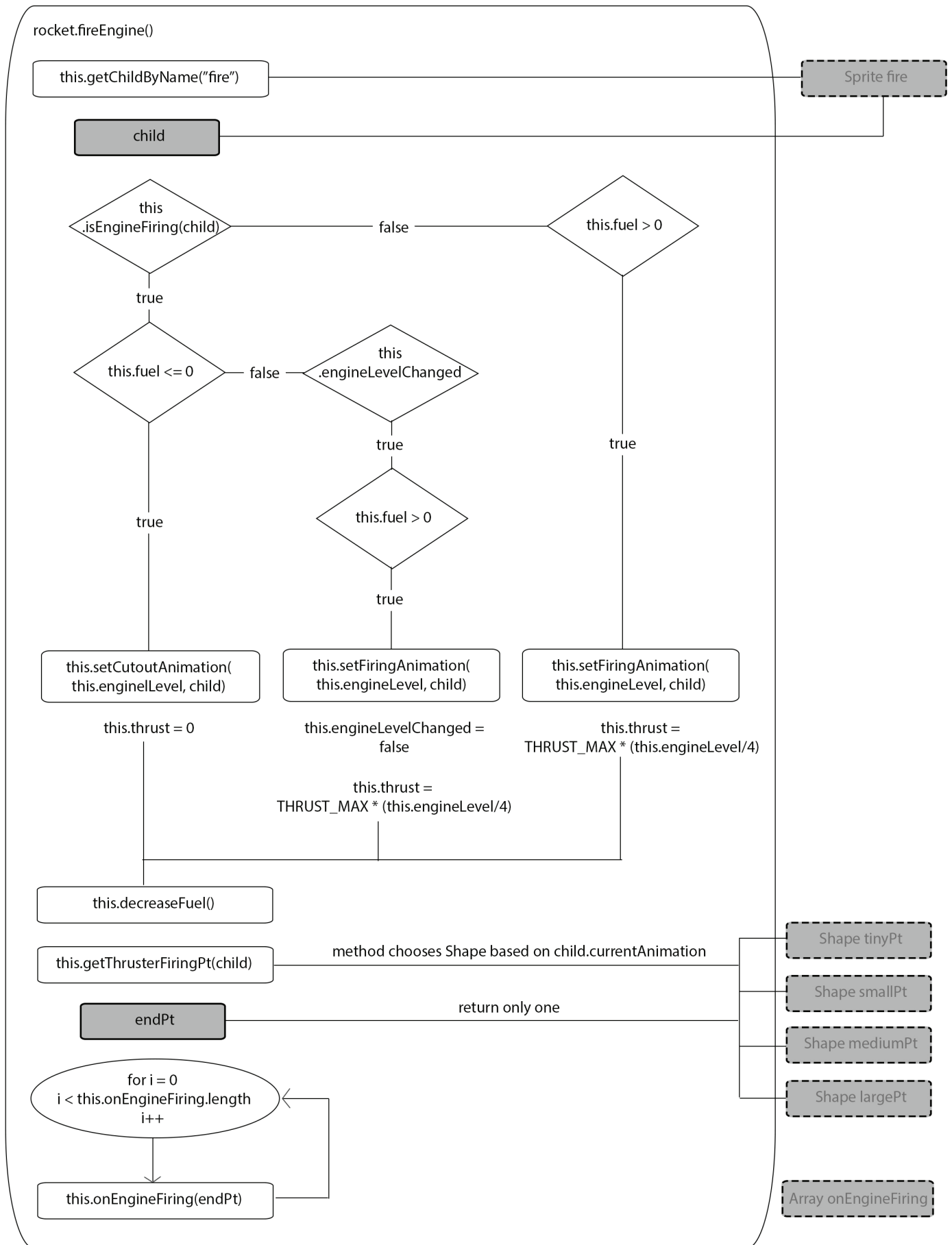


Diagram 15 (opposite page): rocket.fireEngine()

Function checks whether this thruster is already firing and whether there is fuel remaining. It also checks whether the engine level has changed if the engine was already firing. It updates animations and thrust values accordingly. Function then determines the correct reference to the end point of the tip of the current engine firing animation. Then it calls all functions that were stored in the event listener for this event. Called functions are sent a parameter representing the end point of the tip of the engine firing animation.



Window window

HTMLCanvasElement

CreateJS createjs

Ticker

LoadQueue queue

Falcon9.png

Smoke.png

OceanSlice.png

Falcon9Fire.png

Loading.png

Earth.png

Falcon9Thrusters.png

Ocean.png

EarthSlice.png

PauseScreen.png

Explosion.png

Stage stage

Rocket rocket

Shape tinyPt

Sprite body

Shape smallPt

Sprite legs

Shape mediumPt

Sprite fire

Shape largePt

Sprite thrusterL

Shape thrusterLPt

Sprite thrusterR

Shape thrusterRPt

Bitmap earth

DisplayObject collider

Bitmap earthslice

DisplayObject GM

Bitmap ocean

DisplayObject BM

Bitmap oceanslice

DisplayObject IM

Bitmap smoke

Shape landingSite

GUI_Manager GUI

Container bars

FillBar fuelBar

Text label

Shape border

Shape fill

FillBar monobar

Text label

Shape border

Shape fill

Container physBox

Shape textBox

Text physText

Bitmap pauseScreen

Bitmap loadScreen

Text landedText

Text crashedText

Text pauseHint

Sprite explosion

SpriteSheet thruster_sheet

SpriteSheet explosion_sheet

SpriteSheet rocket_sheet

SpriteSheet fire_sheet

Diagram 16: Game Objects

Rocket Lander! contains a number of game objects:

1. Window represents the global namespace created when a browser opens a tab.
2. HTMLCanvasElement is the canvas element CreateJS.Stage is placed inside.
3. CreateJS is a reference to the CreateJS library
4. createjs.Ticker is the library's method of handling time. This object is not instantiated by our code, but is referenced.
5. Our game utilizes multiple PNG images for its graphics.
6. queue is a createjs.LoadQueue that provides ways of loading PNG images.
7. stage is a createjs.Stage, which is the container used to store and display all other game objects:
 - earth and ocean are createjs.Bitmap's which hold the image used for a game level background
 - earthslice and oceanslice are createjs.Bitmap's which contains a horizontal section of the background image used with the given game level. By positioning this object in front of the rocket, all rocket fire animations that extend past the level of the ground / landingSite are hidden from view. This makes the ground feel solid.
 - Collider is an object that encapsulates functionality related to tracking the visible game objects and triggering events when they collide.
 - GM, or the game manager, is an object that encapsulates the functionality of the game flow.
 - BM, or background manager, is an object that encapsulates creating and displaying the game backgrounds.
 - IM, or input manager, is an object that encapsulates functionality related to user input.
 - landingSite is a createjs.Shape which defines the location the rocket can land. It will be hidden from view by default, and the player will rely on background images for the visual cue of where to land.
 - rocket_sheet, fire_sheet, explosion_sheet, and thruster_sheet are createjs.SpriteSheet's which contains the data and image used to animation the rocket and explosion in the event of a crash.
 - smoke is a single instance of a Bitmap object which has the appearance of a puff of smoke.
 - Rocket extends createjs.Container, and contains all the game objects and functionality related to the rocket.
 - createjs.Sprite body, legs, fire, thrusterL, and thrusterR each animation a different part of the rocket.
 - createjs.Shape tinyPt, smallPt, mediumPt, largePt, thrusterLPt, and thrusterRPt are hidden references to the end point of their respective animations. This allows smoke to be generated at the correct positions when the engine or thrusters are firing.

References are stored inside the rocket so that as the rocket moves and rotates, these shapes do as well. Accessing a reference provides the current location of the Shape object without additional calculations.

- GUI extends createjs.Container, and contains all game objects and functionality related to the user interface.
 - pauseScreen is the image displayed when the game is paused.
 - loadScreen is the image displayed at the start of the game before gameplay begins.
 - createjs.Text pauseHint, crashedText, and landedText display text to the user.
 - Sprite explosion contains the animation shown whenever the rocket crashes.
 - physBox is a container that holds a rounded rectangle Shape object and physText, which displays the stats of the rocket.
 - FillBar extends createjs.Container, and contains all objects and functionality related to displaying the fuel remaining visually. It contains a Text object, and two Shape objects.

Diagram 17: Game States

Although Rocket Lander! was not designed as a game state machine, gameplay can be subdivided into a number of states:

- The game begins in the loading state, where all images used in the game are loaded.
- After loading state comes initialize state, where game objects are created.
- Once game objects are created and added to the stage, the game is in run state. Here the rocket is moving through space and players control the rocket.
- At any time during run state, players can pause the game.
- The game can only return to run state when in pause state.
- If the rocket has landed or crashed, game changes to gameover state. Here specific animations are shown, and the player cannot fly the rocket.
- Once the ending sequence has been completed, game enters restart stage. Restart stage allows the game to reset the rocket's values and position, the fuel values, and other game variables. Once restart has been completed, game returns to run state.

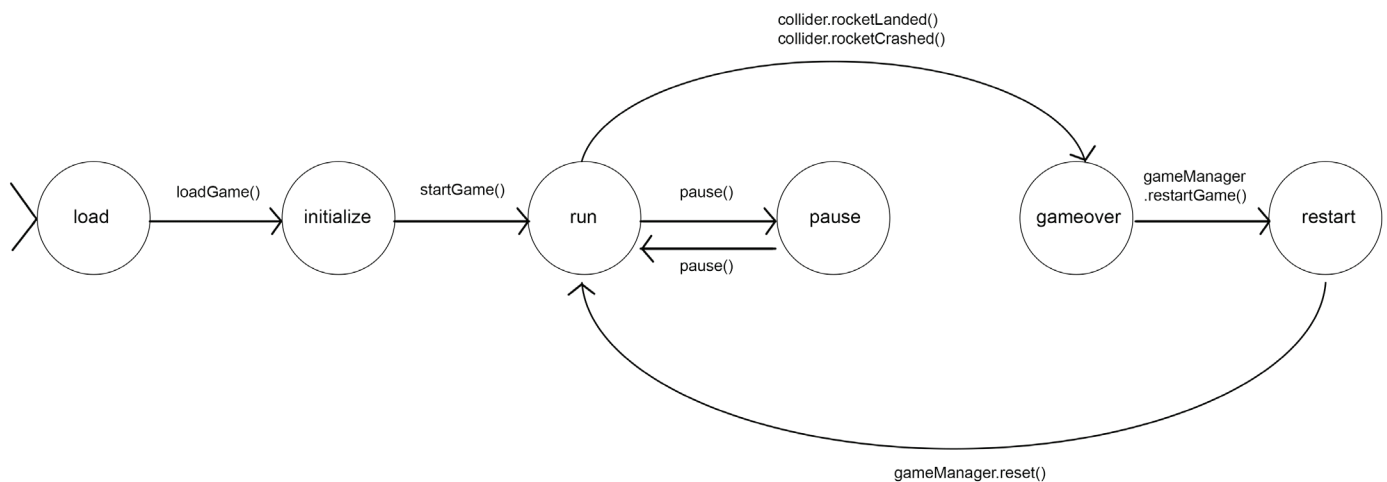


Diagram 18: Engine Animation States

In Rocket Lander! the rocket engine has a number of animations, depending on the situation.

- noFire: No flame animation is shown. The engine begins in this state.
- tinyFire: Smallest flame animation is shown as a cycle.
- smallFire: Slightly larger flame animation is shown as a cycle.
- mediumFire: Second largest flame animation is shown as a cycle.
- largeFire: Largest flame animation is shown as a cycle.
- cutLargeFire: Largest flame animation leads to mediumFire after one animation cycle
- cutMediumFire: Second largest flame animation leads to smallFire after one animation cycle
- cutSmallFire: Second smallest flame animation leads to tinyFire after one animation cycle
- cutTinyFire: Smallest flame animation leads to noFire after one animation cycle

There are multiple events that result in a change in animation:

1. FireEngine() is called when engine is not firing AND engineLevel is not 0.
State changes from noFire to tinyFire, smallFire, mediumFire, or largeFire, depending on the engineLevel.
2. FireEngine() is called when engine is firing AND engineLevel has changed.
State changes from current state to one state to the left or one state to the right, depending on engineLevel.
3. FireEngine() is called when engine is firing AND rocket fuel level reaches zero.
State changes from tinyFire, smallFire, mediumFire, or largeFire to cutTinyFire, cutSmallFire, cutMediumFire, or cutLargeFire, respectively.
4. CutoutEngine() is called when engine is firing.
State changes from tinyFire, smallFire, mediumFire, or largeFire to cutTinyFire, cutSmallFire, cutMediumFire, or cutLargeFire, respectively.

