



Design Document

Revised: 5 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.

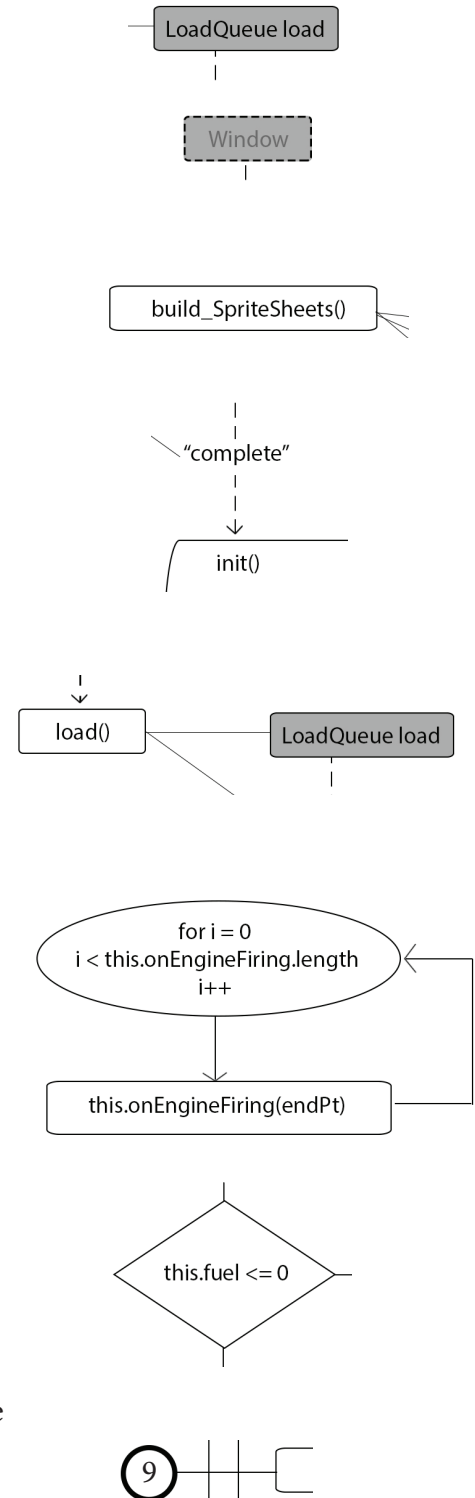


Diagram 1: Game Loading

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, and starts the game.

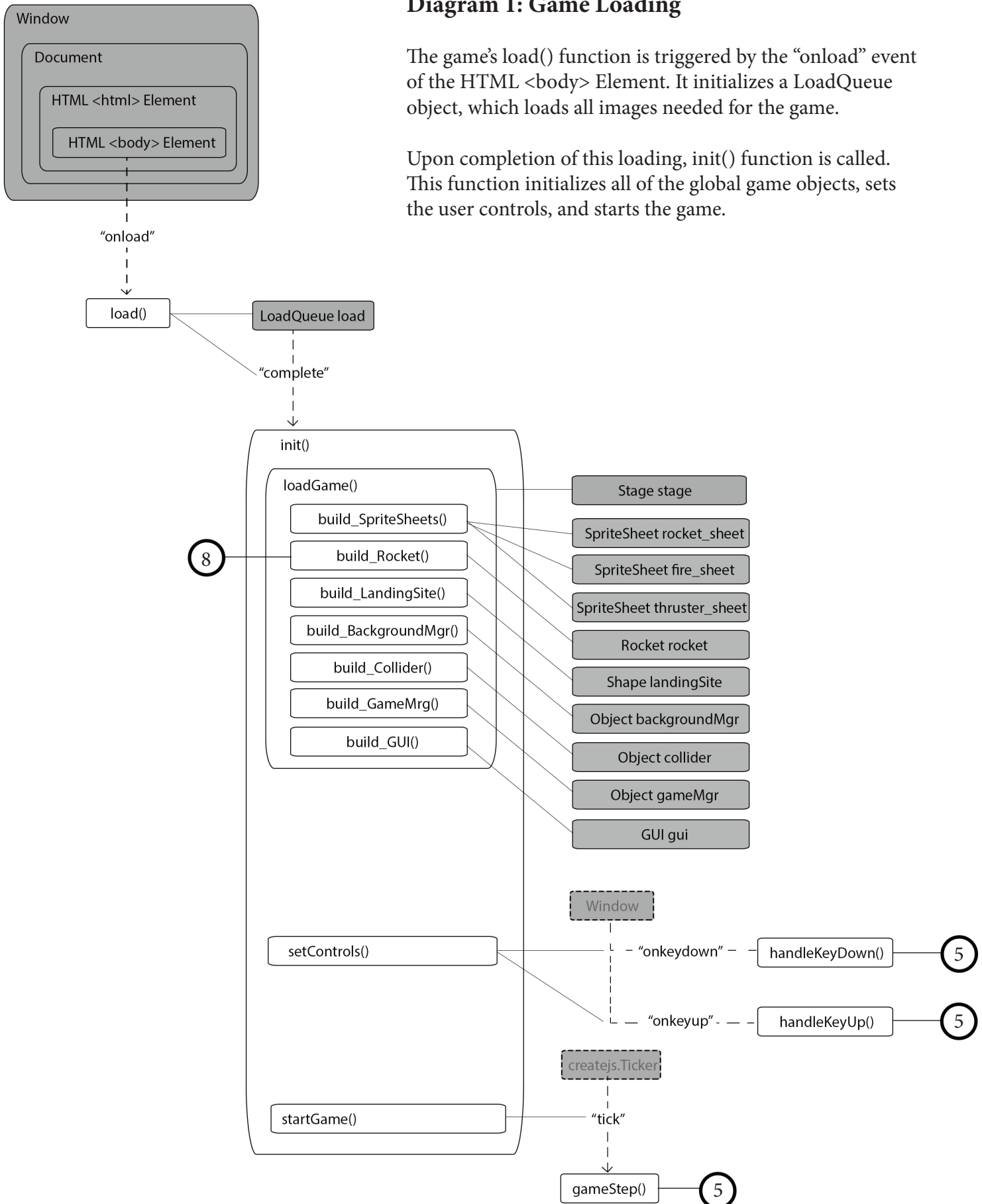


Diagram 2: handleKeyDown(e)

When window detects “onkeydown” event, it triggers handleKeyDown(e). Depending on the key pressed, different functions are called.

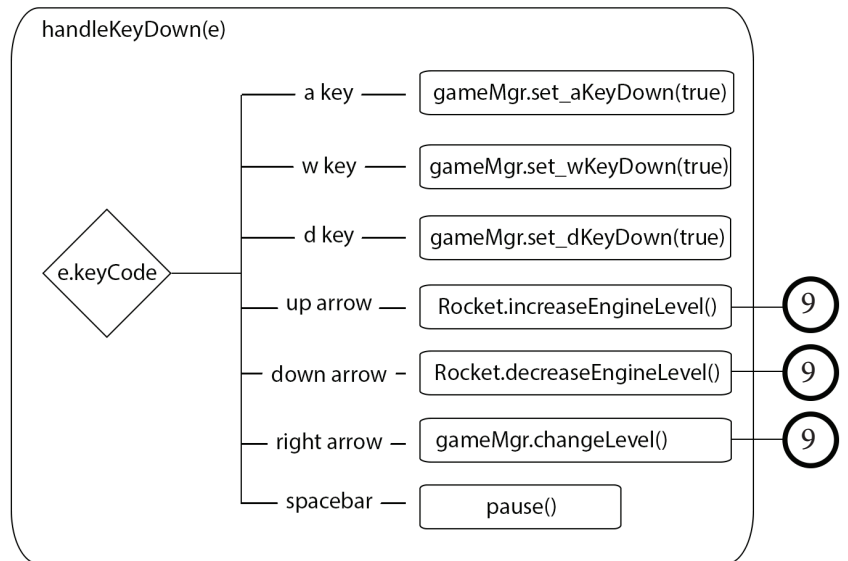


Diagram 3: handleKeyUp(e)

When window detects “onkeyup” event, it triggers handleKeyUp(e). Depending on the key pressed, different functions are called.

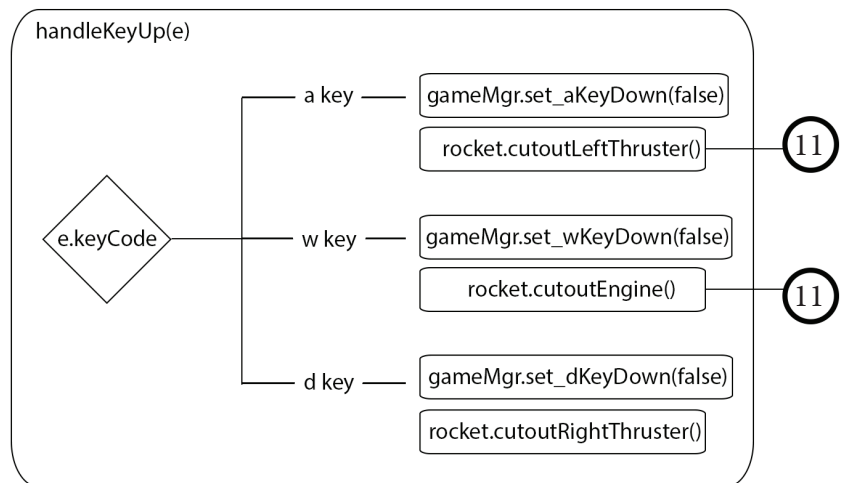


Diagram 4: gameStep()

This global function checks whether createjs.Ticker is paused. If it is not paused, updates the game manager and stage.

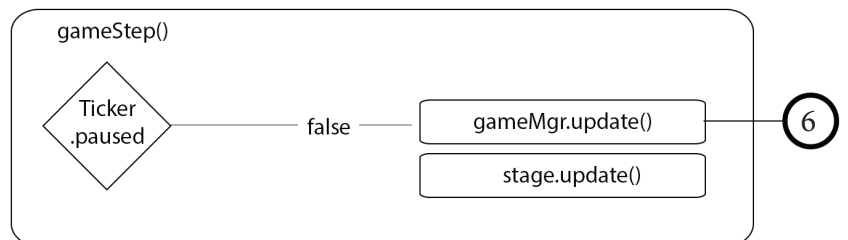


Diagram 5: gameMgr.update()

Game manager has a property gameover. If this internal property is not false, game manager checks other properties. Each of these other properties trigger specific rocket functions. These functions need to occur once every gameStep (other rocket functions can be called once when “onkeydown” event is triggered). Game manager then performs update for rocket, collider, and gui. Finally, game manager renders the rocket.

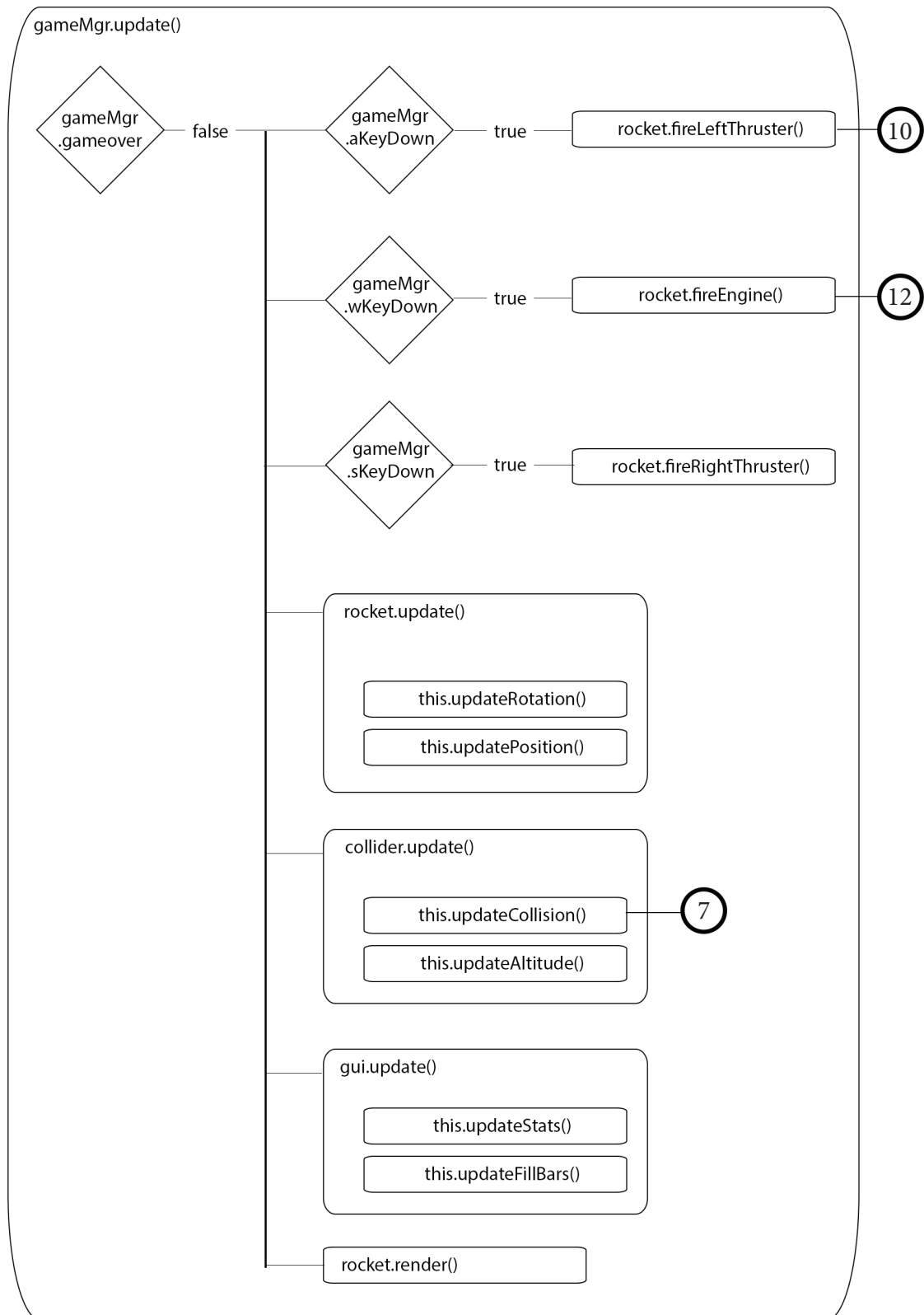


Diagram 6: collider.updateCollision()

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, the collider proceeds with a crash event.

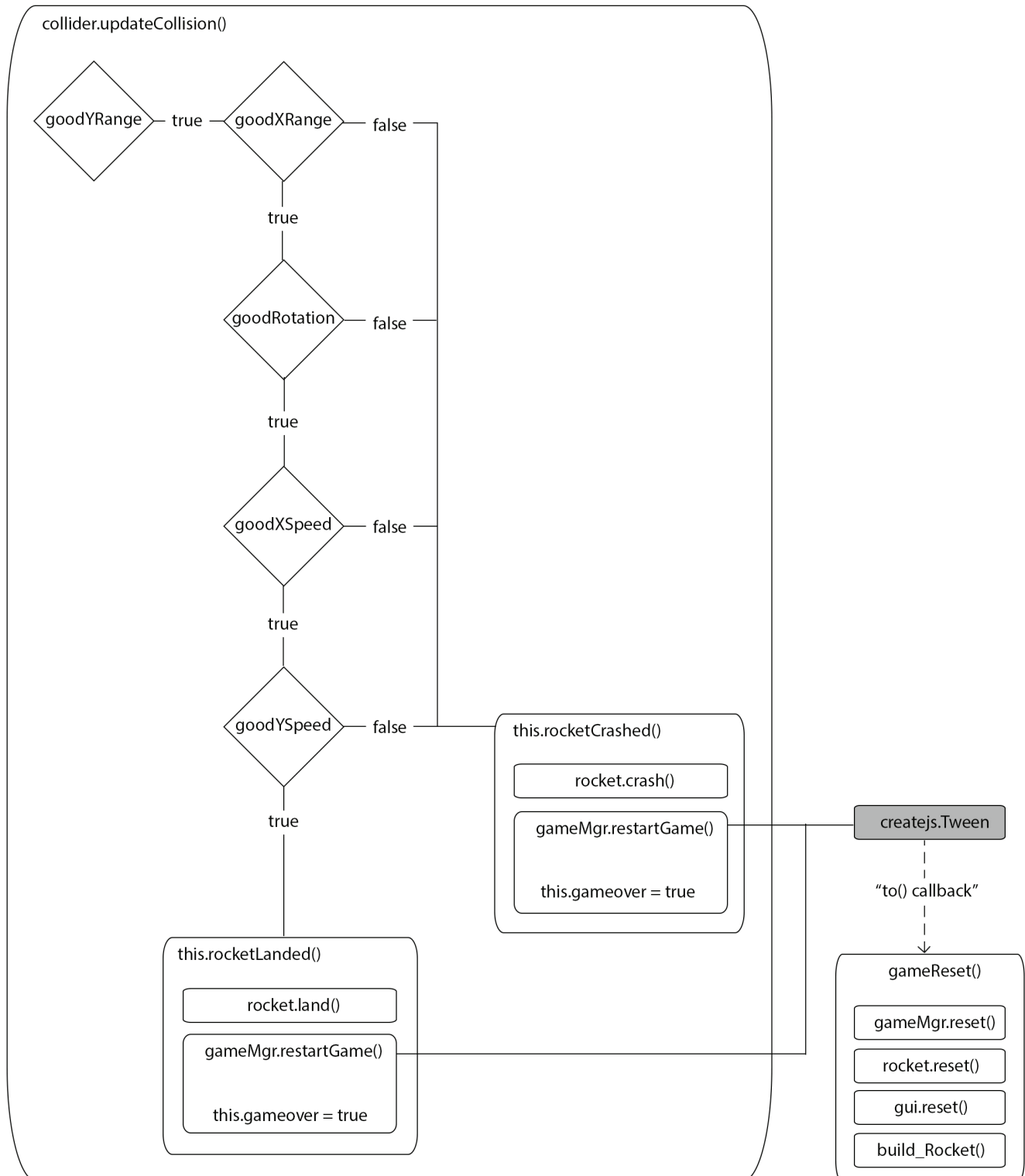


Diagram 7: build_Rocket()

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.

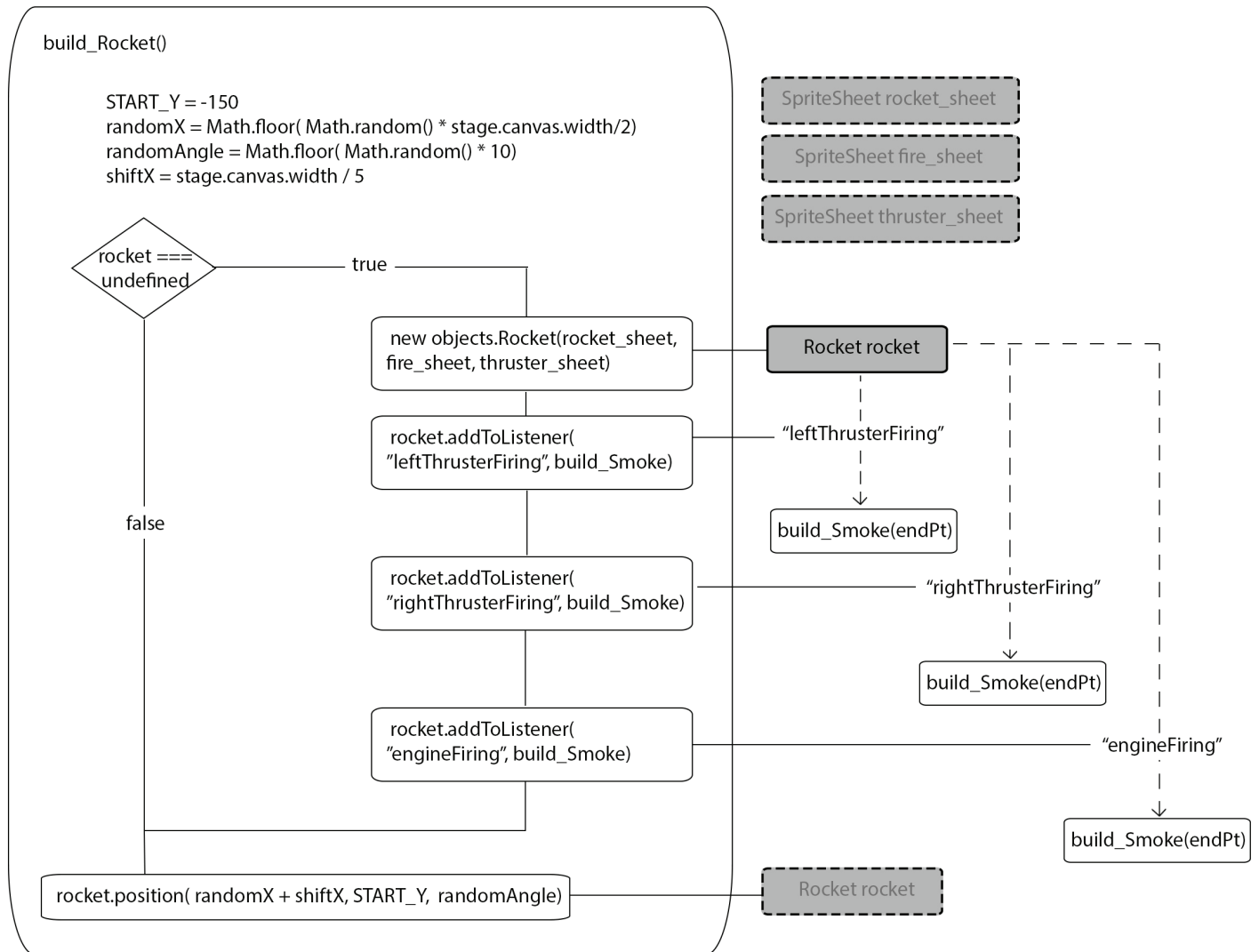


Diagram 8: gameMgr.changeLevel()

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

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

GUI has function to choose which text to display (and in what color), depending on the level.

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.

Finally global function gameReset() is called.

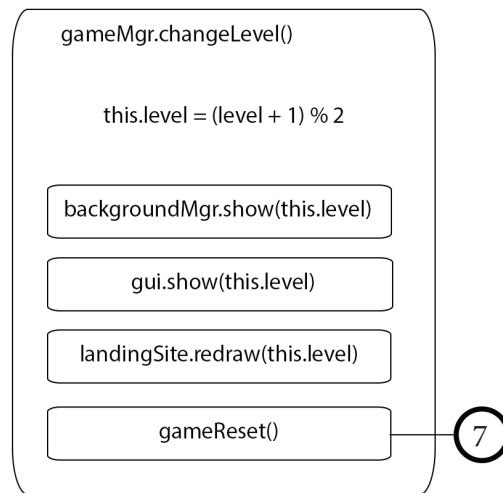


Diagram 9: rocket.increaseEngineLevel()

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

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

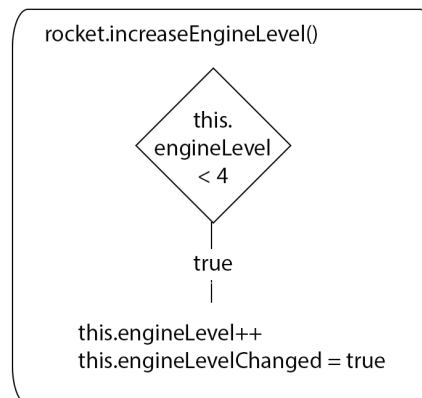


Diagram 10: rocket.decreaseEngineLevel()

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

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

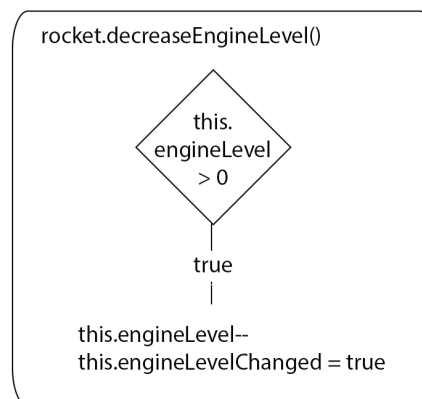


Diagram 11: rocket.fireLeftThruster()

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.

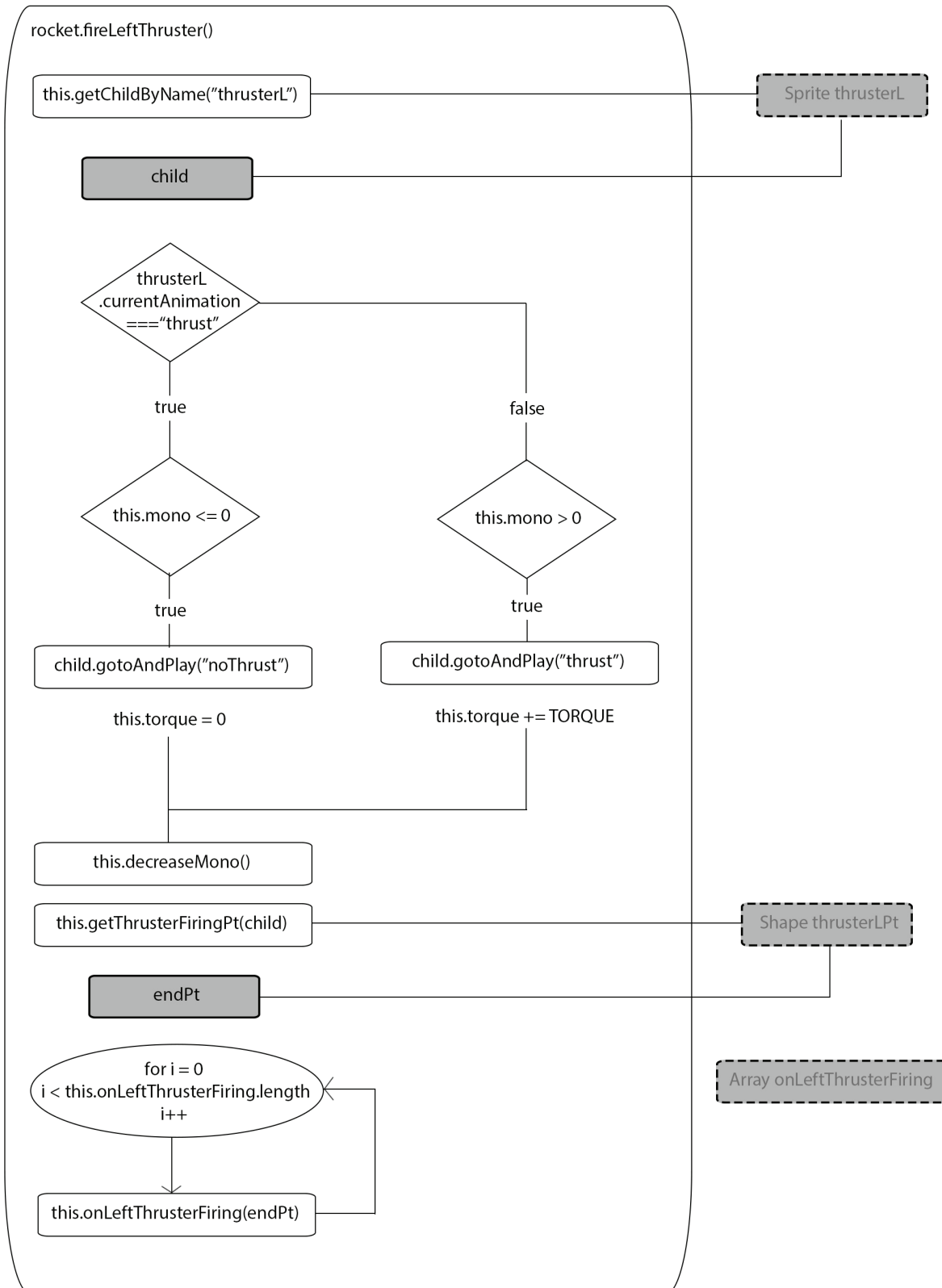


Diagram 12: rocket.cutoutLeftThruster()

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

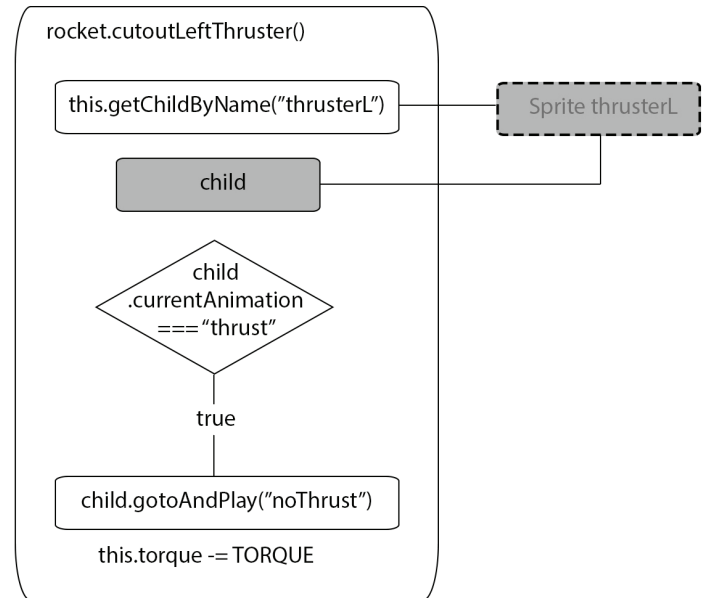
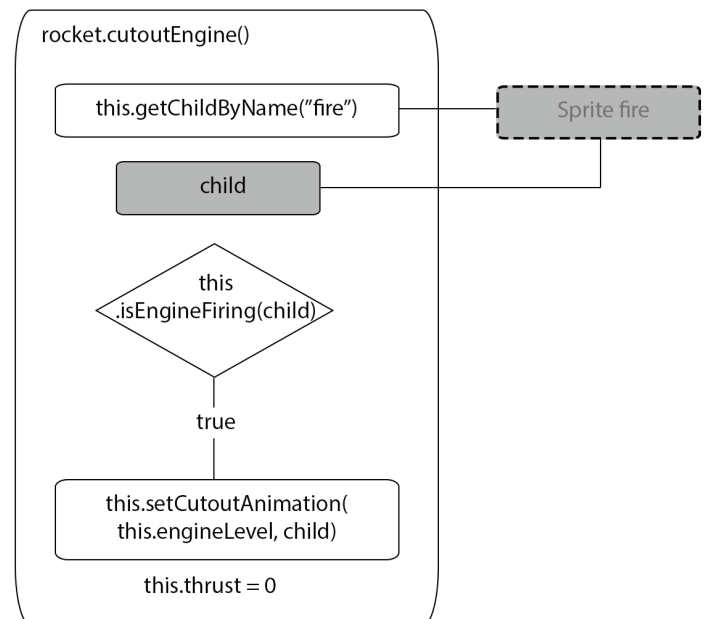


Diagram 13: rocket.cutoutEngine()

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



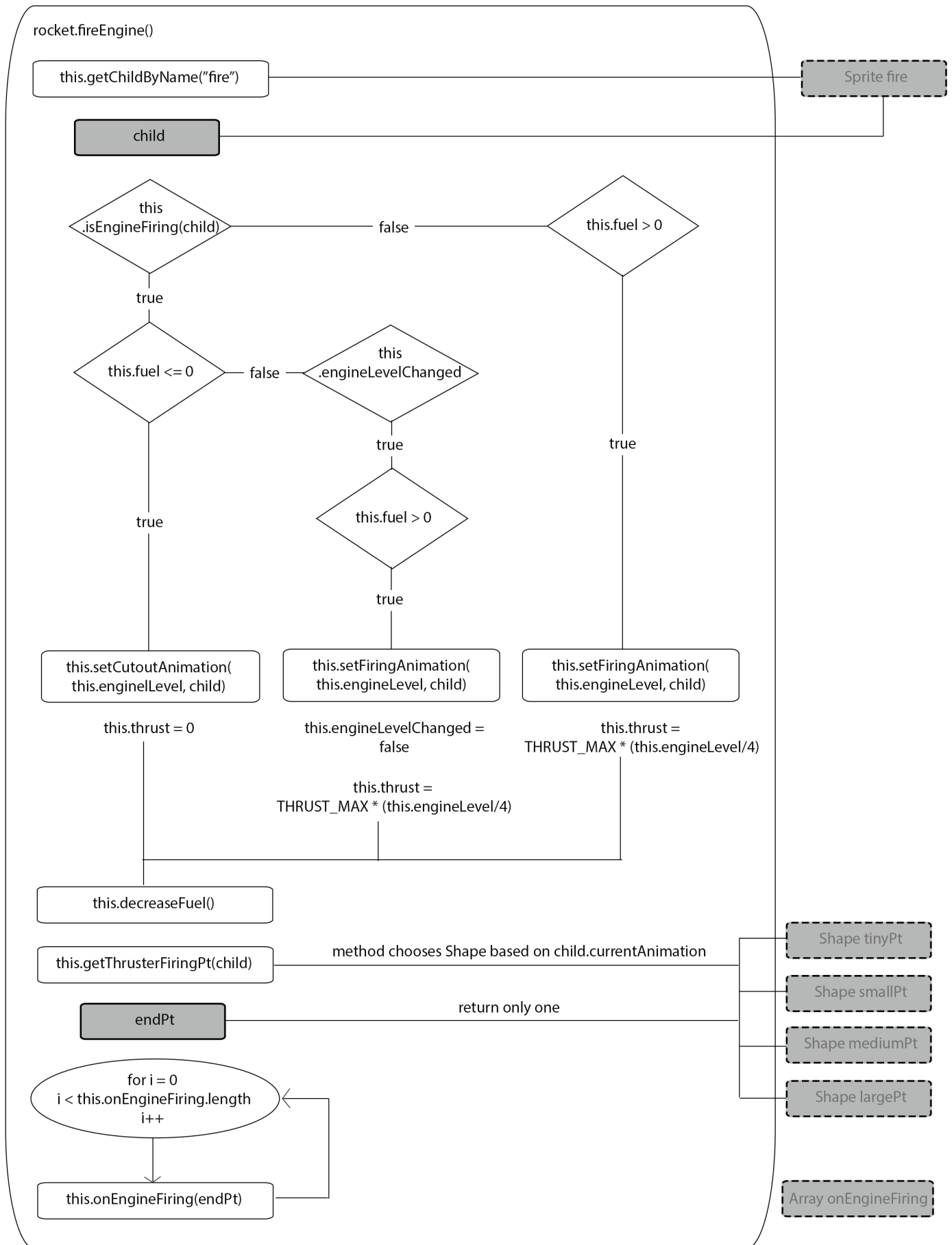


Diagram 14: `rocket.fireEngine()`

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.

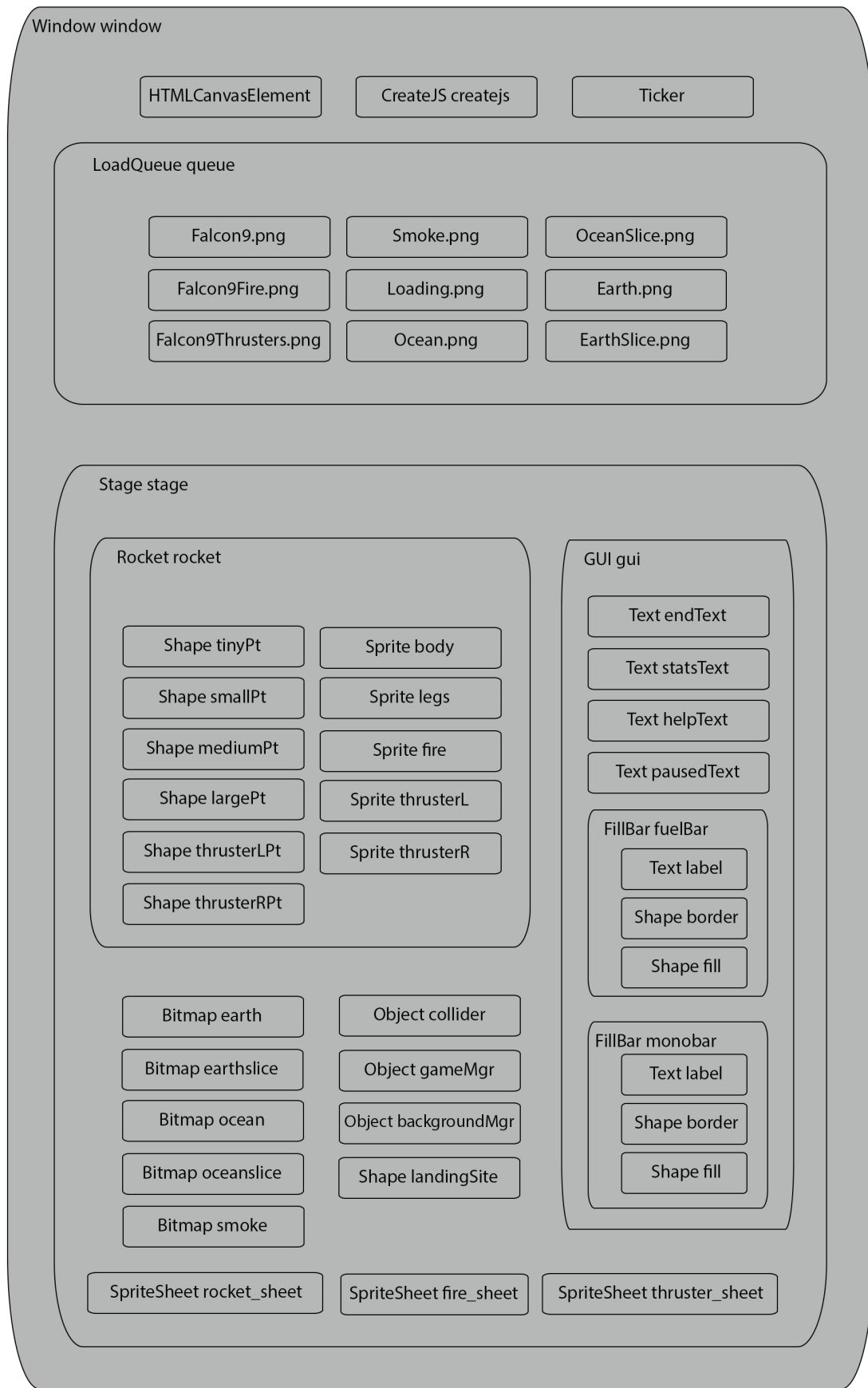


Diagram 15: Game Objects

Rocket Lander! contains a number of game objects:

- Window represents the global namespace created when a browser opens a tab.
- HTMLCanvasElement is the canvas element CreateJS.Stage is placed inside.
- CreateJS is a reference to the CreateJS library
- createjs.Ticker is the library's method of handling time. This object is not instantiated by our code, but is referenced.
- Our game utilizes multiple PNG images for its graphics.
- queue is a createjs.LoadQueue that provides ways of loading PNG images.
- 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.
 - gameMgr is an object that encapsulates the functionality of the game stage and update.
 - backgroundMgr is an object that encapsulates the functionality of displaying the game backgrounds.
 - 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, and thruster_sheet are createjs.SpriteSheet's which contains the data and image used to animation the rocket.
 - 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.
 - createjs.Text endText, statsText, helpText, and pausedText display text to the user.
 - 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.