Drone Delivery

by Jack Morgan and Ryan Nelson

NDSU Spring 2017 | CSCI 313 Project | Last Updated: 27 Feb 2017

Drone Delivery is a platform-style JavaScript web-browser game. It is built inside a DOM Canvas object in HTML and utilizes the createjs library.

TABLE OF CONTENTS

0. Terminology

1. Narrative

2. Objective

3. Scene

4. Game Components

4.1 Actors

4.2 Environmental Forces

4.3 Game Rules

4.4 Game Events

4.5 Game Platform

4.6 Game Controls

5. Game Design

5.1 Game Design (Basic)

5.1.1 Objects

5.1.2 Variables

5.1.3 Functions

5.2 Game Design (Advanced Topics)

5.3 Game Design (Collision Detection)

6. Bugs

7. Game Expansion Ideas

0. TERMINOLOGY

- Movable object: object that can be moved or picked up using keyboard / mouse input. Movable objects will include the Drone and the Parcel.
- Pickup object: object that can be picked up and carried by the Drone.

1. NARRATIVE

"It's 2050 and delivering packages by Drone is all the rage! You have just enrolled in Drone Flying School to prepare for a career as a drone pilot. To successfully complete your training, you'll need to guide Junior, your drone at hand, through a series of obstacle courses.

Each course will test your ability to navigate around these hazards, think critically, and deliver your package on time. Future courses may unlock opportunities to upgrade your drone and fly in style."

2. OBJECTIVE

For each course, players must pick up and deliver The Parcel to the Drop Zone of each course before time runs out on the Game Timer, while avoiding all hazards.

Complete all courses to graduate from Drone Flying School as a pilot.

3. SCENE

The game is laid out as a single platform, with the entire course visible. The background will be themed according to each particular course. Game title is displayed outside and above this field.

An overlay will display the following information during gameplay: - Time remaining for delivery - Current course number

4. GAME COMPONENTS

4.1. ACTORS

Player-Controlled

- Drone
  - Junior is a flying drone that players control on screen.
  - He has twin engines and the ability to pick up certain objects.
  - Junior is destroyed if he collides with a hazard.
  - Junior is not affected if he collides with a wall, platform, or edge of the screen.

NPCs

- The Parcel
  - The Parcel can be picked up by the Drone.
  - The Parcel is destroyed if it collides with a hazard.

    - The Parcel is not affected if it collides with a wall, platform, or edge of the screen.

Hazards

- The Ocean
    - Forms the bottom of the course, and prevents Junior from landing.
- Birds
    - Birds fly back and forth in either a horizontal or vertical pattern.
    - They each pause at the end of a movement, then turn around to fly the other way.

Neutrals

- Walls
    - Form vertical boundaries neither the Drone nor any movable object can pass through.
- Platforms
    - Form horizontal boundaries neither the Drone nor any movable object can pass through.
- Game Timer
    - show the remaining amount of time

Positives

- Drop Zone
    - Each course has a highlighted area called the Drop Zone.
    - If Junior flies into the Drop Zone while carrying The Package, the course is successfully completed.
    - Nothing occurs if Junior flies into the Drop Zone without The Package.

4.2. ENVIRONMENTAL FORCES

Solidity

- No game objects (including the Drone and The Parcel) can go outside the canvas
- The Drone and pickup objects (i.e The Parcel) cannot pass through a neutral object
- All pickup objects bounce horizontally off neutral objects and the canvas edges

Gravity

- All pickup objects (i.e The Parcel) fall downward if dropped in the air
- The Drone falls downward if he is not using engine power

4.3. GAME RULES

Game Start Rules

- Junior and the Parcel will be positioned on platforms inside the course at the start of each attempt.
- Each course begins paused.

Game Ending Rules

- If the Drone lands in the Drop Zone while carrying The Parcel before the Game Timer runs out, the course is won
- If either the Drone or The Parcel are destroyed, player loses the course
- If the Game Timer reaches 0:00 before delivery, player loses the course

4.4. GAME EVENTS

Game start

- At the start of each course, a Gameplay Explanation is shown.
- The Game Timer countdown begins when the player presses SPACEBAR at the start of each course.

Game Ending

- If the course is won or lost, the game is paused and a message is displayed

Control Events

- Feedback from the keyboard and mouse is used to move the Drone in each course

4.5. GAME PLATFORM

Players are expected to have the following equipment: - Computer with web browser - Keyboard - Mouse

4.6. GAME CONTROLS

Drone

Players control a Drone using the keyboard and mouse.

- A-KEY: if the Drone is not landed, move Drone left
- D-KEY: if the Drone is not landed, move Drone right
- LEFT MOUSE: click and hold to make the Drone fly upward; release the mouse button to let the Drone drift downward.

Pickup / Drop Actions

Players control interaction between the Drone and The Parcel through the keyboard.

- SPACEBAR: if the Drone has landed with its grabber (black area) on The Parcel, pick up The Parcel. If the Drone is carrying The Parcel, drop The Parcel.

Game Mechanics

Players control game mechanics through the keyboard.

- ESC: pauses the game if the game was going; unpauses the game if the game was paused.
- SPACEBAR: if the game is showing the Gameplay Explanation, start the game; if the game is

paused, causes the game to restart.

5. GAME DESIGN

5.1 GAME DESIGN (BASIC)

The game is composed of JavaScript objects, variables, and functions.

5.1.1. OBJECTS

Objects are composed of standard classes of the createjs library.

- createjs.Ticker
- stage (Stage)
- dContainer (Container)
- sky (Bitmap) (i.e. background image)
- birds (Sprite) (each course includes multiple birds)
- text (Text) (game includes multiple Text objects)
- drone (Shape)
- ocean (Shape)
- walls (Shape) (each course includes multiple walls)
- dropZone (Shape)

About createjs.Ticker:

createjs.Ticker is a built-in component of Stage. It triggers a "tick" event based on a given framerate. For Drone Delivery we use 60 frames per second, so the "tick" event occurs 60 times per second. Game mechanics are based on this "tick" event.

About dContainer:

dContainer is a container that can contain children. At game start, dContainer contains the Drone. The Drone cannot be removed from dContainer. If the Drone "picks up" a pickup object (i.e. The Parcel), that object is also added to dContainer. If the Drone "drops" an object it was carrying, that object is removed from dContainer.

Keyboard and mouse interactions move dContainer. Moving dContainer moves all of its children as well.

About walls

Wall objects are used to create platforms in Drone Delivery. The object type is exactly the same, except for a platform the object is wider than it is tall. (We found it unnecessary to create a separate method and object for platforms).

5.1.2. VARIABLES

- gameObjectsArr (array containing all objects the Drone can interact with)

- movingArr (array containing all objects that user can interact with that are currently moving through the air (i.e. Drone, The Parcel, etc.))

How Variables Are Used

- Once the game has started, the game loops through all objects in movingArr, updating and rendering their positions.
- Updating movingArr object positions involves performing collision detection against all objects in the gameObjectsArr.

Variable Rules:

- If The Parcel or a similar pickup object are picked up, it is removed from the gameObjectsArr.
- If The Parcel or a similar pickup object are dropped, it is added to the movingArr.
- If The Parcel or a similar pickup object land on a horizontal neutral surface, it is removed from the movingArr and added to the gameObjectsArr.

5.1.3. FUNCTIONS

Each function will be described in detail in the implementation section. Drone Delivery contains the following functions:

startup functions

- load()
- init()
- buildGame()
- startGame()
- restartGame()

game input

- detectKey(e)
- removeKey(e)
- moveUp(e)
- moveDown(e)

game mechanics

- runGame(e)
- pauseGame(e)
- setCourseOver(scenario)
- restartGame(e)

game GUI

- buildGUI()
- buildPauseMenu(color)

- buildPauseRect(w, h, color)
- buildGameTimer(color)
- convertTime(ms)
- updateTimer(t)
- buildStartupMessage()

game courses

- buildCourse(number)
- buildCourse1()

game objects

- buildBackground(target)
- buildBird(x,y,w,h)
- buildWall(x,y,w,h,color)
- buildDropZone2(x,y,w,h,color)
- buildDrone()
- buildContainer()
- buildParcel()
- buildOcean(n, h, depth, a, b)

collision detection

- detectCollision(target, nextX, nextY)
- revisePosition(target, cObject, nextX, nextY, revisedArr)
- detectEdgeOfFrame(target, nextX, nextY)
- mostRestrictive(target, revisedArr, pt)

game actions

- checkPickup(target)
- pickup(target)
- drop(target)
- neutralResponse()
- hazardResponse()
- dropZoneResponse()

movable object update / rendering

- calcNextPosition(target)
- performCollisionDetection(target, nextX, nextY)
- getChildClone(child)
- performPositionRevision(target, collisionArr, nextX, nextY)
- updatePosition(target)
- renderPosition(target)
- detectLanding(target)

- updateChildrenBounds(container, cX, cY)

animation

- movePropellers()
- moveWaves(e)

5.2 GAME DESIGN (ADVANCED TOPICS)

Object Properties

Each Shape or Bitmap object is a DisplayObject in createjs. DisplayObjects have certain default properties, and Drone Delivery uses some of these properties for key game functionality.

- alpha
  - the transparency of the DisplayObject
  - 0 for totally transparent
  - 1 for totally opaque
- name
  - Used in performPositionRevision() to check whether the DropZone was collided with
  - Used in revisePosition() to update "landed" property of originals of a clone
- x
  - the x position of DisplayObject, relative to its container
- y
  - the y position of DisplayObject, relative to its container

Dynamically Injected Properties

JavaScript suppports dynamically injected properties for objects. Drone Delivery utilizes this feature extensively to store object data. Dynamically injected properties in Drone Delivery include:

- carried
  - used to flag whether the parcel is being carried by the Drone
- cloneOf
  - used to store a reference to the original object that a clone represents
  - used in getChildClone() and revisePosition()
- curves
  - used to store reference to bezierCommands array that contains all bezier graphics command objects used to construct the Ocean
- direction
  - used to indicate whether movable object is moving right or left or neither
- height
  - used to store the height in pixels of the given object
- isContainer
  - used to flag whether this object is a container or not
  - used in collision detection functions

- landed
  - used to indicate whether movable object has landed on a horizontal surface
- nextX
  - used to store the future x-position of given object
- nextY
  - used to store the future y-position of given object
- onCollision
  - used to store a reference to a function
  - function can be called by using onCollision()
- speedX
  - used to store current horizontal speed of given object
  - speed is never negative (only direction changes)
- speedY
  - used to store current vertical speed of given object
  - speed is never negative (only direction changes)
- width
  - used to store the width in pixels of given object
- up
  - used to flag that the Drone is moving upward
- xPropeller
  - used to store the current x-position of the animated band on the drone's propellers

Note about onCollision

It is possible to set a function as a property of an object. Certain objects have the property "onCollision". The idea is that if an object is collided into, we call

.onCollision();

and the function referenced there is called. For the case of a bird, onCollision is set to hazardResponse. If a bird is collided into by a moving object, the hazardResponse() method will be called, which ends the course.

Graphics Command Objects

Each Shape object in createjs has a graphics property by default, a reference to a graphics object. Inside this object is a stack of graphics command objects (GCO). It begins empty by default, but as a drawing function is called (i.e. beginFill, drawRect, drawCircle, etc.) a GCO to the graphics stack. This GCO is an object with a given type (Graphics.Circle, for drawCircle).

A reference to this object can be stored and changes can be made to any GCO later. If a change is made, the next time the stage is updated and the Shape is redrawn, any GCO changes will be visible.

In Drone Delivery this ability is used to animate the Drone's propellers, by keeping a reference to the small thin bar that moves horizontally on each propeller to simulate revolutions. The position of the bar can be changed by referencing the GCO that created the bar, and changing the x-coordinate provided to that GCO.

Event Listeners

With createjs, if a second event listener is added for the same event and references the same function, two events (two calls to the function) will be triggered when the event occurs.

To avoid this duplication issue, in Drone Delivery we remove the old event listeners when they are no longer needed.

In JavaScript Window class, there are two ways to add event listeners. - window.addEventListener("event", functionToCall); - window.onevent = functionToCall;

setInterval() / clearInterval()

To provide timed animations that are independent of the createjs.Ticker, Drone Delivery uses window.setInterval(functionToCall, timeInMilliseconds).

To stop these animations when the game is paused, Drone Delivery uses window.clearInterval(functionToCall).

e = !e ? window.event : e

This code is needed to ensure browser compatibility with Microsoft Internet Explorer. For some reason, if the window triggers an event with IE, it may not be represented as the proper object.

bounds

Drone Delivery uses object bounds in collision detection. Each object's bounds is abstracted as a Rectangle object. For collision detection, the game checks whether two Rectangles (reflecting two different game objects) have collided.

Shape objects are not given bounds by default. Bounds must be manually added and updated. Bitmap objects are given bounds by default.

drawing position

A graphics object can use methods to draw things like rectangles. When drawRect() is called, it requires four arguments: x, y, width, height. For a graphics.Rectangle, x,y is defaulted to the upper left corner. By setting x,y to 0,0, this draws the object in the 0,0 position relative to the graphics object.

A graphics object is added to a Shape object (and the Shape is added to the Stage for display). The Shape object (extending DisplayObject) has its own x,y properties. These x,y indicate where the object's registration point will be relative to the Stage.

- When graphics rectangle x,y is set to 0,0, and Shape x,y is set to 100,100, the graphics rectangle will be drawn with upper left corner at 100,100 in the Stage.

- When graphics rectangle x,y is set to 100,100 and Shape x,y is set to 100,100, the graphics rectangle will be drawn with upper left corner at 200,200 in the Stage.

Graphics object vs. Shape.graphics

Graphics can be added to Shapes in two ways. - Graphics object is created first, and graphics command objects are added to the graphics objects. Later, the Shape is instantiated with the graphics object as a parameter. - Shape object is created first. The default constructor instantiates a blank graphics object and stores reference to it through the Shape.graphics property. Graphics command objects can be added through the property.

Bezier curves

- When cp1x and cp2x are horizontally aligned, they create a bezier curve like a sine curve
  - curve is symmetric about the x-position and mirrored about the y axis (if cp1x and cp1y are both set at 10, then the x-axis is at 10)
  - the y-axis is the middle value between cp1y and cp2y (if cp1y is 0 and cp2y is 20, y-axis would be at 10)
- When bezier curve starts and ends at the same y-position, it forms a curve symmetric about the y-axis
  - second half is mirrored about the y-axis
- When the bezier curve has the y value of its two control points symmetric about the starting y and ending y of the curve, there is symmetry as well

5.3 GAME DESIGN (COLLISION DETECTION)

Collision detection for dContainer is partly based upon its children. Drone Delivery is designed so that dContainer cannot be moved to a place where one of its children collides with a game object. This is accomplished in the following way: - For each child: + determine if that child has any collisions in its next calculated position + for any such collisions, determine the most restrictive revised position the child can be moved to (this is done by considering all possible collisions that the child has at the given moment, finding a revised point for each collision, and keeping only the most restrictive) - After all children: + determine the most restrictive revised position for dContainer, based on the most restrictive revised position for each of its children + keep only the most restrictive + use it to set the final revised position for dContainer

6. Bugs

Bug 3.01

If Drone lands on surface while carrying Parcel, then lets go of Parcel, then grabs again, sometimes the Parcel sinks below the surface, and when Drone grabs it again, there is an issue where eventually dContainer is getting the call to move to position -30,-33. Has something to with the shiftX, shiftY in Step 2 B of the updatePosition() method. For now, we have mitigated it by only choosing to update to point values that are greater than or equal to zero.

7. Game Expansion Ideas

- Ability to lift Rocks and other movable objects, and use these other objects to solve the course
    - hold down a switch to open a door
    - destroy an enemy
- Add concept of a battery level
    - flying the Drone lowers battery
    - also include Power Ups to add more battery, so Drone can fly longer
    - battery visualization will appear at the top of the screen
- Power Ups
    - invicibility from hazards
    - invisibility (go through walls)
- Additional courses
- Concept of Lives
    - player gets a set number such as 5
    - each time a course is lost, player loses a life
    - each time a course is won, player gains an extra life
    - game only ends if player runs out of lives
- Drone Upgrades
    - after a certain number of levels, given option to choose a faster drone or one that can carry heavier loads
    - improvements indicated by different color for drone body
- Performance points
    - player could gain or lose points based on how precisely they navigate a given course (how many walls they hit, etc.)
- Accrued damage
    - drone could be slightly damaged each time it hits a wall
    - too many hits could destroy the drone
    - could require spending points to repair drone after each course