```
/* ------------ GLOBAL HELPER VARAIBLES ------------ */
// Difficulty Helpers
let astProjectileSpeed = 3; // easy: 1, norm: 3, hard: 5

// Game Object Helpers
let currentAsteroid = 1;
// I omitted constants
// Movement Helpers
let LEFT = false;
let RIGHT = false;
let UP = false;
let DOWN = false;

// TODO: ADD YOUR GLOBAL HELPER VARIABLES (IF NEEDED)
// I omitted constants
// game states
let firstTimePlaying = true;
let gamePaused = true;
let onReadyPage = false;
let playerIsDead = false;

let score = 0;
let startingDanger = 20;
let asteroidSpawnRate = 800;
let danger = 20;
let level = 1;

let portal;
let shield;
let hasShield = false;

let collectAudio = new Audio("./src/audio/collect.mp3");
let dieAudio = new Audio("./src/audio/die.mp3");

// intervals
let moveSpaceshipInterval;
let createAsteroidsInterval;
let spawnPortalInterval;
let spawnShieldInterval;
let increaseScoreInterval;
let checkCollisionInterval;

/* ---------------------- MAIN ----------------------- */
$(document).ready(function () {
  // jQuery selectors
  game_window = $('.game-window');
  game_screen = $("#actual-game");
  asteroid_section = $('.asteroidSection');
  // hide all other pages initially except landing page
  game_screen.hide(); // Comment me out when testing the spawn() effect below

/* --------- ASSIGNMENT 2 SELECTORS BEGIN ---------- */
  // buttons
  playButton = $('.landing-play-button');
  settingsButton = $('.landing-settings-button');

  settingsDifficultyButton = $('.settings-difficulty-button');
  settingsCloseButton = $('.settings-close-button');

  // pages
  landingPage = $('.landing-container');
  tutorialPage = $('.tutorial-container');
  settingsPage = $('.settings-container');

  // other
  slider = $('.settings-slider');
  volumeElement = $('.settings-volume-value');
  volumeElement.text(slider.val());
  collectAudio.volume = slider.val() / 100;
  dieAudio.volume = slider.val() / 100;

/* ---------- ASSIGNMENT 2 SELECTORS END ----------- */
// TODO: DEFINE YOUR ASSIGNMENT 3 JQUERY SELECTORS HERE
  // buttons
  tutorialButton = $('.tutorial-button');

  pauseButton = $('.pause-button');
  resumeButton = $('.resume-button');
  restartButton = $('.restart-button');
  exitButton = $('.exit-button');

  startOverButton = $('.restart-button');

  // pages
  gamePage = $('#actual-game');
  getReadyPage = $('.get-ready-container');
  pausePage = $('.pause-container');
  restartPage = $('.restart-container');
  gameOverPage = $('.game-over-container');

  // other
  scorePanel = $('.score-panel-container');

  scoreElement = $('.score');
  dangerElement = $('.danger');
  levelElement = $('.level');

  restartText = $('.restart-text');

  spaceship = $('.spaceship');

  finalScoreElement = $('.final-score');
});


/* ----------------- EVENT HANDLERS ----------------- */
// Keydown event handler
document.onkeydown = function (e) {
  if (e.key == 'ArrowLeft') LEFT = true;
  if (e.key == 'ArrowRight') RIGHT = true;
  if (e.key == 'ArrowUp') UP = true;
  if (e.key == 'ArrowDown') DOWN = true;
  // I added this
  if (e.key == "Escape") {
    pauseGame();
  }
}

// Keyup event handler
document.onkeyup = function (e) {
  if (e.key == 'ArrowLeft') LEFT = false;
  if (e.key == 'ArrowRight') RIGHT = false;
  if (e.key == 'ArrowUp') UP = false;
  if (e.key == 'ArrowDown') DOWN = false;
}

/* --------- ASSIGNMENT 2 EVENT HANDLERS BEGIN ------ */
$("html").on("click", ".landing-settings-button", function(event){
  settingsPage.css("display", "flex");
});

$("html").on("click", ".landing-play-button", function(event){
  hideAllPages();

  if (firstTimePlaying) {
    firstTimePlaying = false;
    tutorialPage.css("display", "flex");
  } else if (!gamePaused) {
    showGetReadyPage();
  } else {
    gamePage.css("display", "flex");
    scorePanel.css("display", "flex");
    pauseGame();
  }
});

$("html").on("click", ".settings-difficulty-button", function(event){

settingsDifficultyButton.removeClass('settings-selected-difficulty-button');
  $(this).addClass('settings-selected-difficulty-button');

  switch (this.innerHTML) {
    case 'Easy':
      astProjectileSpeed = 1;
      startingDanger = 10;
      asteroidSpawnRate = 1000;
      break;
    case 'Normal':
      astProjectileSpeed = 3;
      startingDanger = 20;
      asteroidSpawnRate = 800;
      break;
    case 'Hard':
      astProjectileSpeed = 5;
      startingDanger = 30;
      asteroidSpawnRate = 600;
      break;
    default:
      alert(`Invalid difficulty "${this.innerHTML}"!`);
  }
});

$("html").on("click", ".settings-close-button", function(event){
  settingsPage.css("display", "none");
});

$("html").on("input", ".settings-slider", function(event){
  volumeElement.text(slider.val());
  collectAudio.volume = slider.val() / 100;
  dieAudio.volume = slider.val() / 100;
});

/* ------- ASSIGNMENT 2 EVENT HANDLERS END ---------- */
// TODO: ADD MORE FUNCTIONS OR EVENT HANDLERS (FOR ASSIGNMENT 3) HERE
$("html").on("click", ".tutorial-button", function(event){
  hideAllPages();
  showGetReadyPage();
});

$("html").on("click", ".pause-button", function(event){
  if (!gamePaused && !playerIsDead) {
    pauseGame();
  }
});

$("html").on("click", ".resume-button", function(event){
  resumeGame();
});

$("html").on("click", ".restart-button", function(event){
  pausePage.css("display", "none");
  restartPage.css("display", "flex");
});

$("html").on("click", ".exit-button", function(event){
  hideAllPages();
  playButton.text("Resume game!");
  landingPage.css("display", "flex");
});

$("html").on("click", ".restart-yes-button", function(event){
  restartGame();
});

$("html").on("click", ".restart-no-button", function(event){
  restartPage.css("display", "none");
  pausePage.css("display", "flex");
});

$("html").on("click", ".start-over-button", function(event){
  hideAllPages();
  landingPage.css("display", "flex");
});

// helper functions
function showGetReadyPage() {
  onReadyPage = true;

  score = 0;
  danger = startingDanger;
  if (startingDanger == 10) {
    astProjectileSpeed = 1;
  }
  if (startingDanger == 20) {
    astProjectileSpeed = 3;
  }
  if (startingDanger == 30) {
    astProjectileSpeed = 5;
  }
  level = 1;

  scoreElement.text(score);
  dangerElement.text(danger);
  levelElement.text(level);

  pauseButton.css("display", "none");
  spaceship.css("display", "none");
  if (portal) {
    portal.remove();
  }
  if (shield) {
    shield.remove();
  }

  gamePage.css("filter", "brightness(100%)");
  gamePage.css("display", "flex");
  getReadyPage.css("display", "flex");
  scorePanel.css("display", "flex");

  setTimeout(() => {
    onReadyPage = false;
    getReadyPage.css("display", "none");

    spaceship.css("top", "50%");
    spaceship.css("left", "50%");
    spaceship.css("translate", "-50% -50%");

    startGame();
  }, gameReadyOccurrence);
}

function hideAllPages() {
  landingPage.css("display", "none");
  tutorialPage.css("display", "none");
  settingsPage.css("display", "none");
  gamePage.css("display", "none");
  getReadyPage.css("display", "none");
  pausePage.css("display", "none");
  restartPage.css("display", "none");
  gameOverPage.css("display", "none");
}

function startGame() {
  gamePaused = false;
  playButton.text("Play game!");

  pauseButton.css("display", "flex");
  spaceship.css("display", "flex");

  moveSpaceshipInterval = setInterval(moveSpaceship, 1);
  createAsteroidsInterval = setInterval(createAsteroids, asteroidSpawnRate);
  spawnPortalInterval = setInterval(spawnPortal, portalOccurrence);
  spawnShieldInterval = setInterval(spawnShield, shieldOccurrence);
  increaseScoreInterval = setInterval(increaseScore, scoreIncreaseOccurrence);
  checkCollisionInterval = setInterval(checkCollision, 1);
}

function pauseGame() {
  gamePaused = true;
  pausePage.css("display", "flex");
  gamePage.css("filter", "brightness(20%)");

  stopGame();
}

function resumeGame() {
  pausePage.css("display", "none");
  gamePage.css("filter", "brightness(100%)");
  startGame();
}

function restartGame() {
  restartPage.css("display", "none");
  restartText.css("display", "flex");
  setTimeout(() => {
    restartText.css("display", "none");
    showGetReadyPage();
  }, restartTime);
}

function stopGame() {
  clearInterval(moveSpaceshipInterval);
  clearInterval(createAsteroidsInterval);
  clearInterval(spawnPortalInterval);
  clearInterval(spawnShieldInterval);
  clearInterval(increaseScoreInterval);
  clearInterval(checkCollisionInterval);
}

function moveSpaceship() {
  if (LEFT) {
    if (hasShield)
      spaceship.attr("src", "./src/player/player_shielded_left.gif");
    else {
      spaceship.attr("src", "./src/player/player_left.gif");
    }

    let newPos = Math.max(parseFloat(spaceship.css("left")) - spaceshipSpeed, minPersonPosX);
    spaceship.css("left", newPos);
  }

  if (RIGHT) {
    if (hasShield)
      spaceship.attr("src","./src/player/player_shielded_right.gif");
    else {
      spaceship.attr("src", "./src/player/player_right.gif");
    }

    let newPos = Math.min(parseFloat(spaceship.css("left")) + spaceshipSpeed, maxPersonPosX);
    spaceship.css("left", newPos);
  }

  if (UP) {
    if (hasShield)
      spaceship.attr("src", "./src/player/player_shielded_up.gif");
    else {
      spaceship.attr("src", "./src/player/player_up.gif");
    }

    let newPos = Math.max(parseFloat(spaceship.css("top")) - spaceshipSpeed, minPersonPosY);
    spaceship.css("top", newPos);
  }

  if (DOWN) {
    if (hasShield)
      spaceship.attr("src", "./src/player/player_shielded_down.gif");
    else {
      spaceship.attr("src", "./src/player/player_down.gif");
```

```
    }

    let newPos = Math.min(parseFloat(spaceship.css("top")) +
spaceshipSpeed, maxPersonPosY);
    spaceship.css("top", newPos);
  }

  if (!LEFT && !RIGHT && !UP && !DOWN) {
    if (hasShield) {
      spaceship.attr("src", "./src/player/player_shielded.gif");
    } else {
      spaceship.attr("src", "./src/player/player.gif");
    }
  }
}

function createAsteroids() {
  spawn();
}

function spawnPortal() {
  let randomPos = getRandomPosition();

  const objectString = "<img class='portal' src = 'src/port.gif'/>";
  asteroid_section.append(objectString);

  portal = $('.portal');
  portal.css("top", randomPos.y);
  portal.css("left", randomPos.x);

  setTimeout(() => {
    if (portal) {
      portal.remove();
    }
  }, portalGone);
}

function spawnShield() {
  let randomPos = getRandomPosition();

  const objectString = "<img class='shield' src =
'src/shield.gif'/>";
  asteroid_section.append(objectString);

  shield = $('.shield');
  shield.css("top", randomPos.y);
  shield.css("left", randomPos.x);

  setTimeout(() => {
    if (shield) {
      shield.remove();
    }
  }, shieldGone);
}

function increaseScore() {
  score = parseInt(scoreElement.text()) + 40;
  scoreElement.text(score);
}

function checkCollision() {
  if (portal && isColliding(portal, spaceship)) {
    portal.remove();

    level++;
    astProjectileSpeed *= 1.5;
    danger += 2;

    levelElement.text(level);
    dangerElement.text(danger);

    collectAudio.play();
  }

  if (shield && isColliding(shield, spaceship)) {
    hasShield = true;
    shield.remove();

    collectAudio.play();
  }
}

function getRandomPosition() {
  return {
    x: getRandomNumber(minPersonPosX, maxPersonPosX),
    y: getRandomNumber(minPersonPosY, maxPersonPosY)
  };
}

/* ----------------- GAME FUNCTIONS ----------------- */
class Asteroid {
// I omitted the constructor
function spawn() {
  const asteroid = new Asteroid();
  move(asteroid);
}

function move(asteroid) {
  // create an interval to move an Asteroid (i.e. repeatedly update
an Asteroid's position)
  const astermovement = setInterval(function () {
    // HINT: Consider checking collision and other game states here
    if (onReadyPage) {
      asteroid.id.remove();
      clearInterval(astermovement);
      return;
    }
    if (gamePaused || playerIsDead) {
      return;
    }
    if (isColliding(asteroid.id, spaceship)) {
      if (hasShield) {
        hasShield = false;
        asteroid.id.remove();
        clearInterval(astermovement);
        return;
      }
      else {
        playerIsDead = true;
        stopGame();
        spaceship.attr("src", "./src/player/player_touched.gif");
        dieAudio.play();

        setTimeout(() => {
          playerIsDead = false;
          hideAllPages();
          finalScoreElement.text(score);
          gameOverPage.css("display", "flex");
        }, playerDeadTime);
      }
    }

    // update Asteroid position on screen
    asteroid.updatePosition();
    // determine whether Asteroid has reached its end position
    if (asteroid.hasReachedEnd()) { // i.e. outside the game border
      // remove this Asteroid from DOM (using jQuery .remove()
method)
      asteroid.id.remove();
      // clear the interval that moves this Asteroid
      clearInterval(astermovement);
    }
  }, AST_OBJECT_REFRESH_RATE);
}

/* ---------- Additional Utility Functions  --------- */
// Are two elements currently colliding?
function isColliding(o1, o2) {
  // I've omitted isOrWillCollide
  return isOrWillCollide(o1, o2, 0, 0);
}

// Get random number between min and max integer
function getRandomNumber(min, max) {
  return (Math.random() * (max - min)) + min;
}
```

Notes:

1/14/25

- Fitt's law- time = $a + b * \log(D / S + 1)$
  - D = distance and S = size
  - easier to move mouse to something that is closer and bigger. log(D/S)
- Steering law- time = $a + b * D / W$
  - D = distance (length of tunnel) and W = width (width of tunnel)
  - harder to navigate dropdown menu that is long and wide
- People can process visual and audio input at the same time
- Miller's law- 7 +- 2 chunks can be held in working memory

- Gulf of execution- discrepancy between what a user wants to do and what they can actually do
- Gulf of evaluation- discrepancy between what the user knows/understands about the state of the system and the actual state of the system

- Affordance- what a system lets you do
- Signifier- signals where and how interaction should take place
- Metaphor- way to create a signifier
- Learned association

- Learnability- Can you get better at using the interface?
- Efficiency- Can you get the job done quickly + well?
- Discoverability- Can you find new/existing UI features/tools?
- Understandability - Do you understand what is happening?

1/16/25

- CSS selector- select all elements that have a specific tag or…
  - # - ID
  - . - class
- Selectors can be a little more complicated
  - p, #helloWorld selects all <p> tags and the element with ID "helloWorld"
  - p #helloWorld selects an element with ID "helloWorld" that's inside a <p> tag
  - p#helloWorld selects all <p> tags with ID "helloWorld"
- Position property values
  - static (default)
  - relative- position element relative to its normal/original position
    - top: 10px moves the element 10 pixels down from its original position
  - fixed- element stays in the same position even if the page scrolls
    - right: 0 positions the element 0 pixels away from the right edge of the screen
  - sticky (probably won't use this one often)
  - absolute- position element relative to its nearest ancestor that has a specified position.
    - "Specified position" means you explicitly wrote out the element's position property and assigned it to something besides static.
    - Common use: assign position: relative to the parent element, then assign position: absolute to the child element. Then you can position the child element inside the parent element much like you can position an element with a fixed position inside a webpage.
- With flexboxes (edit position of elements that are children of this container):
  - Choose how items are positioned horizontally using the "justify-content" property
  - Choose how items are positioned vertically using the "align-items" property
  - Choose whether items should be in a row or a column with the "flex-direction" property (default is row, column reverses justify-content and align-items)
    - Specific children elements can override this using the "align-self" property
  - Choose whether items wrap using the "wrap" property (whether they all go on one line or multiple lines)
  - Choose how much space is between lines using the "align-content" property

11/21/25

- JavaScript can be embedded in HTML, but only in the <head> tag or at the end of <body>.
- Alternatively (recommended), you put it in a separate file much like styles.css. Do <script src= "(source here)"></script>.
  - Can be put in <head> or at the end of <body>. If in <head>, it runs before parsing the HTML. If at the end of body (recommended), it runs after parsing the HTML. If you really want to put it in <head> though but make the code run after parsing the HTML, you can write "defer " before "src".
- In a button, you can add an "onclick" attribute and set it equal to a line of code (probably a call to a function that was defined in your javascript file) that will run when the button is clicked
- JavaScript selectors
  - document.getElementById("str")
  - document.getElementsByClassName("str")
- In JavaScript, functions are first-class objects (can be stored in a variable, passed to or returned from a function)
- Functions can be object constructors. Just use "this.propertyName" to assign values to properties of the new object being constructed. Then you can create the object by doing something like "let person = new Person("alan")" where Person is the name of the function.

1/23/25

- jQuery
  - $('h1') selects all h1 tag elements
  - $('#item') selects the element with ID 'item'
  - $('.items') selects all elements with class 'items'
  - $(':button') selects all elements with type 'button'
  - $('[href]') selects all elements with an href attribute
  - $('[value="Submit"]') selects all elements whose value attribute has value "Submit"
  - $(document) selects whole document
  - $('#item').html() gets the element's text
  - $('#item').click(callbackFunc) causes callbackFunc to be called every time the element is clicked
  - $('#item').css("property", "value") sets the element's property to the specified value
  - $('#item').append(element)

1/28/25

- One browser process per window (think: everything above and including the bookmarks bar), one renderer process per tab (think: everything below the bookmark bar)
- Browser process knows user inputs. Once it detects an event, it tells the renderer process that the event occurred, and then the renderer process handles that event.
- AJAX allows you to update parts of a webpage without reloading the whole page.
- There is an event loop that executes code. First it runs all the synchronous code, then it runs all the code in the microtask queue (promises), then it runs all the code in the macrotask queue (everything else, like setTimeout, setInterval, ajax). Before code/functions enter either of these queues, we first wait on the web API to return a response.
  - For example, while we're waiting for a promise to resolve, that promise is just waiting on the web API. But once it's resolved, if there is a callback function that is called once that promise resolves, it would go in the microtask queue.

1/30/25

- Want to make systems **usable and useful** by focusing on users, needs, requirements. It's iterative.
- Four stages of Human-centered design- need finding, iteration, prototyping, testing
- Can try to understand users through observation, surveys, or interviews (either structured, semi-structured, or unstructured).
  - Ask the right people (daily users, manufacturers, experts) the right questions (why do you do that/want this feature?)

2/4/25

- Ask targeted, specific questions- easier to answer than general questions
  - Bad example: In what conditions would you prefer the University shuttle over other transportation methods?
  - Good example: Recall the last time you preferred to use other transportation method over the shuttle service. Why did you do so?
- Qualitative data is important to help you know your user's needs
- Affinity diagrams:
  - 1. Capture the main idea of each interview. Your (team's) interpretation
    - Each interpretation note should contain one idea, be self-contained, relate to the participant's motivation, and use first-person pronouns (e.g "I").
      - Bad example:
      - Good example:
  - 2. Group interviews that are similar to each other into clusters called affinity groups
  - 3. Label each affinity group
    - Group hierarchichally: data (white) > category (blue) > theme (pink) > insight (green)
      - 3-7 white notes per blue note
      - Pink and green labels should be about an issue or area of concern
    - Make labels based on the data. Don't try to make categories that fit your pre-conceived labels.

2/6/25

- IDEO and Double diamond are two design processes
- IDEO rules:
  - 1. Defer judgement
  - 2. Encourage wild ideas
  - 3. Build on others' ideas
  - 4. Stay focused on topic
  - 5. Conversations take place one at a time
  - 6. Be visual
  - 7. Go for quantity
- Speed dating- discover what users like and dislike
  - Done by showing users storyboards
    - 3-4 panels
    - Being handrawn is fine
    - Include some riskier/less rational designs
    - Storyboards do not need to focus on *how* the technology works, but more so on how it's used/user experience
    - Ask them what they like/dislike about the proposed solution; do not ask them about how they would want to improve it. Do not consider it an evaluation of the proposed design.
      - Bad questions: Do you think an Uber-like ride-sharing system would be helpful to UM students who find themselves in such a situation? [Show a picture of the proposed solution with a detailed screen design]. Which features described here would you have the most trouble using?
      - Good questions: Have you ever found yourself nervous about a safe way to get home from north campus in the evening? [Show a picture of the proposed storyboard/solution]. Which aspects of the solution do you find compelling?
  - Goals: need validation and getting user's thoughts on proposed solutions

2/11/25

- A prototype is:
  - A series of screen sketches
  - A storyboard
  - Powerpoint slides
  - A simulation
  - A physical rough model
  - A piece of software with limited functionality
- Purpose: can see what it's like, test out new ideas, get feedback, communicate effectively
- Can be low-fidelity or high-fidelity
- A wireframe is a rough sketch of a user interface
- A wizard-of-oz model uses a human to mock certain functionality
- Figma is the most popular prototyping application
- Prototyping is iterative- you can start with a high or low fidelity prototype, and make more as you go on

2/13/25

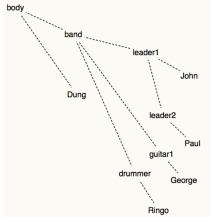
- Figma pages- like snapshots of layouts. Think of it like a git commit.
- A frame is like an HTML div
- Constraints let you specify where an element should lie relative to the edges of the element it is inside.
- Auto layout is like a CSS flexbox.
  - Rectangle select multiple elements and make it an auto layout to group them together.
- A component is like a Unity prefab.
  - Instances can be created by holding the alt key and holding down the mouse and dragging out of the component
  - Can view in Assets tab
  - Don't use them in your designs, only instances of them
- Prototyping lets you define interactions.
  - Go to the prototype tab, hover over an element, click the + button, and drag to another component
  - From there, you can define the interaction much like a JavaScript event
  - Click the present button to test it out
- Variants of components can be made, and interactions can be made between a component and its variants
- Variables are only available with the education or paid plan

HTML example:

```
<head>
  <title>Asteroids</title>
  <!-- JS -->
  <script src='scripts/jquery.min.js'></script>
  <script src='scripts/page.js'></script>
  <!-- CSS -->
  <link rel="stylesheet" type="text/css" href="style/index.css">
</head>

<div class='settings-volume-container column'>
   <div class='settings-volume-header'><span
   class='bolded'>Volume:</span><span
   class='settings-volume-value'></span>
   </div>
   <input type="range" min="0" max="100" value="50"
   class='settings-slider'>
</div>

<div class='landing-header'>
    <img class='landing-asteroid' src="./src/asteroid.png">
    <div>Asteroids</div>
    <img class='landing-asteroid' src="./src/asteroid.png">
</div>
```

CSS example:

```
.settings-container {
  position: absolute;
  z-index: 1;
  top: 2%;
  bottom: 2%;
  left: 25%;
  right: 25%;

  color: white;
  font-size: 50px;
  background-color: darkslateblue;
  border: 3px solid white;

  display: none;  /* can be toggled to flex */
  justify-content: space-between;
}
```




```
<html>
<head>
  <title> Meet the Beatles </title>
</head>
<body>
  <div id="band" class="content">
    <div id="leader1" class="leader ">
      John
      <div id="leader2" class="leader guitarist">Paul</div>
    </div>
    <div id="guitar1" class="guitarist">George</div>
    <div id="drummer">Ringo</div>
  </div>
  Dung
</body>
</html>
```