Applied Math 2 - Asteroids Game Project

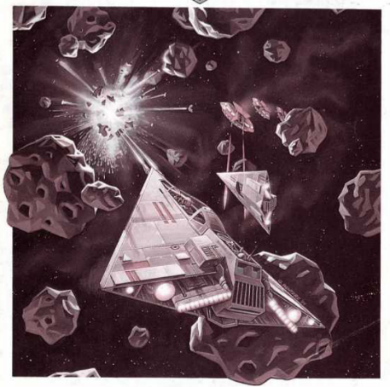
# Project Structure

We’ve now spent well over a term learning about math as it applies to the domain of game development. It’s now time to work on a small project that will incorporate the skills and technologies you’ve been studying.

This is a “Choose Your Own Adventure” project. Each feature you complete “levels up” your mark, but you don’t need to complete them all. Level up your mark until you’re satisfied with your project or until you run out of time. There are no partial marks. A feature must be fully implemented as specified to receive the associated marks. This project is worth 50% of your final grade in this course.

# Project Objective

It’s time to use your Javascript, P5.js, and applied math skills to build a simple video game. You will be implementing the classic game Asteroids, originally released by Atari in 1979. (Kyle was two. You did not exist.)

[You should start by playing a few rounds of the game here](https://games.aarp.org/games/atari-asteroids).   
  
While playing, start to think about the different applied math topics you will need to leverage to implement this game.

Asteroids Gameplay

* The player controls a triangular ship.
* The player can rotate the ship and fire its engine to move.
* The ship eventually comes to a stop when the engine is off.
* The player can teleport the ship through hyperspace to a random location on the screen.
* The player can fire bullets from the front of the ship.
* The objective of the game is to destroy asteroids and saucers.
* The player loses a life when:
  + Their ship crashes into an asteroid.
  + Their ship crashes into a saucer.
  + Their ship is shot by a saucer.

# Basic Asteroids Features (0.5 Marks Each ➡️ 3 Marks total)

* 0.1 - A player can control the rotation of their ship using two keyboard keys. Rotation stops when no keys are pressed.
* 0.2 - A player can activate the ship’s engine by using a keyboard key resulting in a force propelling the ship forward. (The ship’s existing velocity will be affected by this force.)
* 0.3 - The ship will slow down and eventually stop once the player stops firing the engine.
* 0.4 - The player can teleport to a random location on the screen. (They will insta-die if they teleport into another game object.)
* 0.5 - The game starts with a set number of large asteroids, each moving in a random direction. (All asteroids move at the same speed.)
* 0.6 - The player can fire bullets in the forward direction of the ship by hitting a key. Bullets that do not hit anything within a set timeframe will also be removed from play.

# Intermediate Asteroids Features (1 Mark Each ➡️ 13 Marks Total)

* 1.1 - The player loses a life when:
  + They collide with an asteroid of any size.
  + They collide with an enemy saucer. (If saucers are implemented.)
  + They are shot by an enemy saucer. (If saucers are implemented.)
* 1.2 - When the player loses a life, their ship is returned to the centre of the screen.
* 1.3 - When returned to the centre of the screen after losing a lift, the ship will be invincible for a short period of time. There will be some sort of visual indication that the ship is invincible.
* 1.4 - The player destroys asteroids (and saucers if implemented) by shooting them with a bullet or by colliding with them.
* 1.5 - The player’s score increases when they destroy:
  + Large Asteroid: 20 Points
  + Medium Asteroid: 50 Points
  + Small Asteroid: 100 Points
  + Large Saucer: 200 Points (If implemented.)
  + Small Saucer: 1000 Points (If implemented.)
* 1.6 - When a large asteroid is hit, it breaks into two medium asteroids. When a medium asteroid is hit, it breaks into two small asteroids. Small asteroids disappear when shot.
* 1.7 - The player gains an extra life for every 10,000 points they score.
* 1.8 - The game ends when the player loses all their lives. (They start with three lives.)
* 1.9 - The player’s score and number of lives remaining are displayed on the canvas.
* 1.10 - The movement of all game objects (ship, asteroids, saucers, and bullets) will wrap-around the screen edges while maintaining their momentum.
* 1.11 - There are different sound effects for the engine, shooting, the presence of saucers, the destruction of objects, hyperspace jumps.
* 1.12 - You’ve got music playing on a loop in the background of your game.
* 1.13 - The smaller asteroids move slightly quicker than medium ones. Medium asteroids are quicker than larger ones.

# Advanced Asteroids Features (2 Marks Each ➡️ 30 Marks Total)

* 2.1 - At set scoring intervals, an enemy saucer will fly across the canvas and shoot at the player.
* 2.2 - More often than not, the enemies will be large saucers with very poor aim.
* 2.3 - Occasionally small saucers will arrive. They will have decent aim.
* 2.4 - The aim of the small saucers improves as the player’s score increases.
* 2.5 - When the player’s score is high enough, the small saucers will improve their aim by taking the player’s current velocity into consideration when aiming.
* 2.6 - Saucer bullets can also destroy asteroids.
* 2.7 - Saucers are destroyed when they collide with asteroids.
* 2.8 - There is a level system. (Play [the sample game](https://games.aarp.org/games/atari-asteroids) to see how they implement levels.)
* 2.9 - Screen shake has been implemented and used when ships and asteroids are destroyed. (It’s recommended that you only attempt linear shake. See [this video](https://www.youtube.com/watch?v=tu-Qe66AvtY) and [the associated slide deck](https://www.gdcvault.com/play/1023146/Math-for-Game-Programmers-Juicing). This can be implemented using the p5.js translate() function.)
* 2.10 - Each bullet fired applies a small amount of knock-back force to the ship.
* 2.11 - Players can record their names and their top scores on a locally saved leaderboard. This leaderboard should be accessible through an initial main menu or when the game is lost. (See: [storeItem()](https://p5js.org/reference/#/p5/storeItem))
* 2.12 - You’ve coded a way to restart the game after the player wins or loses. When the game restarts all aspects of the game should be reset, including the score, number of lives, number and size of asteroids, etc.
* 2.13 - Your game begins with a title screen that includes a play button to start the game.
* 2.14 - You've implemented custom particle systems for asteroid and ship explosions.
* 2.15 - Your ship to asteroid, saucer to asteroid, and ship to saucer, collisions are handled by nearly pixel-perfect polygon-to-polygon collisions. (In other words, not just rectangular or circular hit-boxes. [You can research polygon-to-polygon collisions here](https://www.jeffreythompson.org/collision-detection/).)

# Mind Melting Asteroids Features (10 Marks)

* 3.1 - Small saucers navigate around the canvas by firing four engines (top, bottom, left, right) while actively avoiding collisions. ***They do not rotate.*** Their motion is physics based, like the ship. These saucers can fire any of their four engines at any time, including multiple engines at the same time. Engines do not have variable thrust; an engine is either fully off or fully on. Whether an engine is on or off should be visible to the player. These enemies cannot remain motionless, and the engine firing collision avoidance system must work. These saucers should be able to navigate around the canvas without crashing while firing at the player’s ship. You must be able to demonstrate all this is a manner that makes it clear that everything is working as required. (Do not attempt this feature until you have nearly completed the project. Remember, there are no partial marks. You can only receive the associated marks if you fully implement the feature.)

# Additional Tasks (2 Marks Each ➡️ 10 Marks Total)

* 4.1 - 200+ word reflection on the problems and discoveries you made while coding this project.
* 4.2 - Separate classes have been used to implement the various game elements (asteroids, player ship, enemies, etc). Minimum four classes. Rather than using globally scoped variables, all game element state is stored in properties of specific objects.
* 4.3 - You’ve designed a reasonable class hierarchy amongst several of your classes.
* 4.4 - You submit a GitHub repo of your project, showing a minimum of 32 commits made throughout the project. (This will mean that you’ve developed the project using VSCode as an IDE and not using the p5.js web editor. This will be covered in class, but [you can get a sneak peek of the workflow here](https://youtu.be/9v9-7ncp2vw).)
* 4.5 – Your game has been deployed to itch.io so that it can be easily played. The itch.io page is professionally polished and includes images, gifs, and explanatory text.

# Project Marking Opportunities

There will be four in-person marking opportunities for this project:

* March 6, 2025
* March 13, 2025
* March 20, 2025
* March 27, 2025

# Feature Marks and Project Grade

**Your project grade will be: 100 \* [marks for your completed work] / 56**

The basic project features listed above are worth 1 mark each.

The advanced project features listed above are worth 2 marks each.

The additional tasks listed above are worth 2 marks each.

The mind melting project feature listed above is worth 10 marks. (I fear the amount of work required for this feature might exceed the reward!)

**SUPER IMPORTANT:** One of the learning outcomes for this project is your ability to implement a complex game to spec while managing your time. As such, during each marking session you may have a maximum of 20 marks worth of features evaluated. This means if you left everything to the last minute and were only marked during the final marking opportunity, your maximum mark on this project would be 20/56.

# Choose Your Own Adventure

There are a total of 66 marks available, but you only need 56 marks to get 100% on the project. This means you can choose to implement the features that interest you the most, and you can stop working on the project when you are satisfied with how high you have levelled up your mark.

The type of graphics you use is not defined by the assignment. You can implement the original line-based graphics or use image sprites. 3rd party image sprites are permitted with attribution.

**IMPORTANT:** For each marking session come prepared with a list of features you believe you have completed. This list should be categorized by basic, advanced, mind melting, and additional tasks. This list can be submitted as a Word/Text file, a Notion page, or within a Learn submission comment.

# Tips, Gotchas, and Best Practices

## Don’t Name Any of Your Classes “Object”

P5.js has an internal class called “Object”. This means that you cannot create any of your own objects with this name. If you do, p5.js will crash when the sketch loads.

## Dealing with User Input Involving Multiple Key Presses

Dealing with simultaneous keypresses in p5.js can lead to weird results, especially when using the keyPressed() callback function or the isKeyPressed global Boolean variable. Instead, it is recommended that you use the keyIsDown() predicate function to detect keyboard input.

if (keyIsDown(65)) { // The 'a' key is being pressed.  
 // rotate counter-clockwise  
}  
  
if (keyIsDown(68)) { // The 'd' key is being pressed.  
 // rotate clockwise  
}

if (keyIsDown(87)) { // The 'w' key is being pressed.  
 // Turn on engine.  
} else {  
 // Turn off engine.  
}

## Objects are Passed by Reference in JavaScript

When passing around objects (like P5.Vectors) you need to know that in Javascript objects are passed by reference. If you aren’t careful, you could end up with multiple objects that accidentally reference the same data.

For example, imagine you want to spawn a bullet at the same position as your ship. If your ship’s position is a P5.Vector, you will want to instantiate your bullet object with a copy of this vector. Thankfully, P5.Vectors have a .copy() method that makes this easy to do.

## The Order of Your Script Tags in the index.html Matters

When implementing inheritance in a p5.js project the order of your script tags in your index.html file matters.

For example, if you have an Actor class (in actor.js) and a Ship class (in ship.js) where Ship looks something like this:

class Ship extends Actor {  
 // Implementation details are not important.  
}

Then you will want to ensure that the actor.js is included in the index.html document \*before\* the ship.js file:

<script src="actor.js"></script>  
<script src="ship.js"></script>

## Performance Boost for p5.js

If you went a little wild with your project and are suffering from poor performance, you can get a small perf boost by adding the following line of code to the top of your sketch.js file:

p5.disableFriendlyErrors = true;

# No Generative AI Usage

No part of this project should be generated by an AI. Your source code should be a genuine reflection of your personal effort and understanding. That said, feel free to brainstorm approaches and architectures with a generative tool like ChatGPT or Claude, just don’t ask it to code your project for you. :)

# 3rd Party Source Code

Many versions of Asteroids exist online as pre-written source code or explained in blog and video form. Resist the urge to reference and/or duplicate Asteroids source code that you did not write. The code for your project should be 100% your own.