ChatGPT

Identified Duplications

The code review reveals **several repeated code patterns** across the modules. Notably:

- **Distance calculations** are duplicated. For example, *StarThrone.distributeStartingTerritories()* uses `Math.sqrt(Math.pow(dx,2)+Math.pow(dy,2))` to measure distance between territories [1]. The AI logic in *AIManager.considerProbeColonization()* repeats the same pattern both in filtering colonizable targets and finding the closest target [2] [3].
- **Discovery color mapping** is repeated via long `if/else` chains. In *DiscoverySystem.renderFloatingDiscoveries()*, the code sets a color based on `floating.discovery.id` with multiple `if` checks (e.g. mapping `"precursor_weapons"` → `"#ff6666"`, `"precursor_drive"` → `"#66ffff"`, etc.) [4]. This same color logic may also appear in UI announcements.
- **Canvas text styling** is also duplicated. For instance, drawing discovery names with a shadow involves two `fillText` calls (one offset in black, one in color) at each occurrence [5]. Similar patterns appear in other text rendering (game over screen, notifications), suggesting a shared "drawTextWithShadow" helper could be used.
- **Rectangles and overlays**: Multiple parts of the UI code draw semi-transparent black backgrounds and stroked borders (e.g. loading/gameover overlays, notification boxes, minimap/leaderboard outlines). While not identical code blocks, these follow a common pattern that could be encapsulated (e.g. a helper to draw an outlined panel).

These duplications stand out as refactoring candidates. In particular, the **distance formula** and **color-selection logic** are exact duplicates in multiple places (each incurring redundant computations).

Proposed Modular Abstractions

To reduce redundancy while preserving performance, we propose the following shared utilities and helpers:

- **Distance Utility**: Introduce a function like `GameUtils.distance(x1,y1,x2,y2)` to replace every occurrence of `Math.sqrt(Math.pow(...)+Math.pow(...))`. This eliminates repeated code and centralizes any future optimizations (e.g. replacing `Math.sqrt` with `Math.hypot`). For example, in *StarThrone.distributeStartingTerritories()* we would replace:

  ```
    const distance = Math.sqrt(Math.pow(candidate.x - assigned.x, 2) +
  Math.pow(candidate.y - assigned.y, 2));
  ```

  with

```
  const distance = GameUtils.distance(candidate.x, candidate.y,
assigned.x, assigned.y);
```

Similarly, in *AIManager* (lines 1314–1318, 1324–1327) we replace the `Math.sqrt/Math.pow` calls with `GameUtils.distance(...)`. This preserves exact numeric results and takes advantage of any engine optimizations. Example from the code:

> "Math.sqrt(Math.pow(t.x - territory.x, 2) + Math.pow(t.y - territory.y, 2))" [2] becomes a single `GameUtils.distance` call.

- **Discovery Color Map**: Replace the long `if/else` chain with a static lookup (object or `switch`) for discovery colors. For instance, add a static method in *DiscoverySystem* (or in a shared utils module) like:

```
static getDiscoveryColor(id) {
  const map = {
    precursor_weapons: '#ff6666',
    precursor_drive:   '#66ffff',
    precursor_shield:  '#66ff66',
    precursor_nanotech:'#ffff66',
    factory_complex:  '#ff9966',
    mineral_deposits: '#ffcc66',
    friendly_aliens:  '#99ff99',
    hostile_aliens:   '#ff4444'
  };
  return map[id] || '#ffffff';
}
```

Then replace the repeated block in *renderFloatingDiscoveries()* with:

```
  const color = DiscoverySystem.getDiscoveryColor(floating.discovery.id);
```

This yields the same mapping as the original chain [4] but in a single lookup.

- **Text-with-Shadow Helper**: Create a function (e.g. in *GameUtils* or a new `Renderer` utility) to draw centered text with a shadow. The current code does this twice for each floating discovery name [5]. A helper like:

```
function drawTextShadow(ctx, text, x, y, color, font='bold 14px Arial') {
  ctx.save();
  ctx.fillStyle = 'rgba(0, 0, 0, 0.8)';
  ctx.font = font;
  ctx.textAlign = 'center';
  ctx.fillText(text, x+1, y+1);
  ctx.fillStyle = color;
  ctx.fillText(text, x, y);
```

```
  ctx.restore();
}
```

can be reused wherever text is drawn with a drop-shadow effect (floating discoveries, HUD text, etc.), reducing repeated canvas state code.

- **Outline Panel Helper**: Introduce a helper to draw a filled, semi-transparent rectangle with a stroked border, given dimensions and styles. Many UI methods (loading screen, game-over, notifications, minimap, leaderboard) follow this pattern. For example, instead of writing multiple lines:

```
ctx.fillStyle = 'rgba(0,0,0,0.8)';
ctx.fillRect(…);
ctx.strokeStyle = '#ffffff'; ctx.lineWidth = 2;
ctx.strokeRect(…);
```

a function `drawPanel(ctx, x, y, w, h, bgColor, outlineColor)` could encapsulate it. This ensures consistency and maintainability (e.g. if outline width changes).

- **Shared Constants and Enums**: Any repeated literals (e.g. discovery IDs, emoji icons, default style values) should be centralized as constants in a common module. For example, the discovery IDs and their bonus descriptions could be in one place, reducing typos or mismatches across modules.

Each proposed abstraction is purely additive (new functions or static methods) and only replaces inline logic. None remove any functionality or introduce performance overhead. In fact, using `Math.hypot` (inside `distance`) could be slightly faster than repeated `Math.pow`, and lookup tables for colors are O(1) instead of multiple branch checks. Critically, all **viewport culling**, **staggered AI updates**, and **object pooling** remain untouched.

Example Refactor Diffs

Below are **Git-style diffs** illustrating these refactors. These patches can be applied directly (e.g. in a Replit or Git environment) to incrementally remove duplication while maintaining functionality.

1. **Add a distance utility to** `GameUtils.js` (or create this module if absent), and use it in all distance calculations:

```
--- a/client/src/game/GameUtils.js
+++ b/client/src/game/GameUtils.js
@@ -0,0 +1,5 @@
+export default class GameUtils {
+  // Compute Euclidean distance between (x1,y1) and (x2,y2)
+  static distance(x1, y1, x2, y2) {
+    return Math.hypot(x2 - x1, y2 - y1);
+  }
}
```

```
--- a/client/src/game/StarThrone.js
+++ b/client/src/game/StarThrone.js
@@ -500,7 +500,7 @@
-            const distance = Math.sqrt(
-              Math.pow(candidate.x - assigned.x, 2) +
-              Math.pow(candidate.y - assigned.y, 2)
-            );
+            const distance = GameUtils.distance(candidate.x, candidate.y,
assigned.x, assigned.y);
             if (distance < minDistance) {
               tooClose = true;
               break;
@@ -531,7 +531,7 @@
-           Math.sqrt(Math.pow(territory.x - t.x, 2) + Math.pow(territory.y -
t.y, 2)).toFixed(1)
+           GameUtils.distance(territory.x, territory.y, t.x, t.y).toFixed(1)
```

```
--- a/client/src/game/AIManager.js
+++ b/client/src/game/AIManager.js
@@ -173,6 +173,7 @@
 import Player from './Player.js';
 import { GAME_CONSTANTS } from '../../common/gameConstants.js';
+import GameUtils from './GameUtils.js';  // new import for utility

 export default class AIManager {
   // ...
@@ -1314,7 +1315,7 @@
-    const colonizableTargets = territories.filter(t =>
-      t.isColonizable &&
-      Math.sqrt(Math.pow(t.x - territory.x, 2) + Math.pow(t.y - territory.y,
2)) < 300
-    );
+    const colonizableTargets = territories.filter(t =>
+      t.isColonizable &&
+      GameUtils.distance(t.x, t.y, territory.x, territory.y) < 300
+    );

-    const target = colonizableTargets.reduce((closest, current) => {
-      const currentDist = Math.sqrt(Math.pow(current.x - territory.x, 2) +
Math.pow(current.y - territory.y, 2));
-      const closestDist = Math.sqrt(Math.pow(closest.x - territory.x, 2) +
Math.pow(closest.y - territory.y, 2));
+    const target = colonizableTargets.reduce((closest, current) => {
+      const currentDist = GameUtils.distance(current.x, current.y, territory.x,
territory.y);
+      const closestDist = GameUtils.distance(closest.x, closest.y, territory.x,
```

```
territory.y);
       return (currentDist < closestDist) ? current : closest;
     });
```

These changes replace every raw `Math.sqrt(Math.pow…)` *with* `GameUtils.distance(...)` *(preserving exactly the same value). The* `import GameUtils` *ensures the utility is available. The performance impact is neutral or better, and all existing culling/logic remains unchanged.*

1. **Encapsulate discovery color logic** by adding a helper and replacing the `if/else` chain:

```
--- a/client/src/game/DiscoverySystem.js
+++ b/client/src/game/DiscoverySystem.js
@@ -0,0 +1,17 @@
 export default class DiscoverySystem {
+  // Map discovery ID to display color
+  static getDiscoveryColor(id) {
+    switch (id) {
+      case 'precursor_weapons': return '#ff6666';
+      case 'precursor_drive':   return '#66ffff';
+      case 'precursor_shield':  return '#66ff66';
+      case 'precursor_nanotech':return '#ffff66';
+      case 'factory_complex':   return '#ff9966';
+      case 'mineral_deposits':  return '#ffcc66';
+      case 'friendly_aliens':   return '#99ff99';
+      case 'hostile_aliens':    return '#ff4444';
+      default:                  return '#ffffff';
+    }
+  }
   constructor() {
     this.discoveries = {};
     // ...
```

```
--- a/client/src/game/DiscoverySystem.js
+++ b/client/src/game/DiscoverySystem.js
@@ -1612,9 +1612,6 @@
       const screenPos = camera.worldToScreen(floating.x, floating.y);

-      // Get discovery color
-      let color = '#ffffff';
-      if (floating.discovery.id === 'precursor_weapons') color = '#ff6666';
-      else if (floating.discovery.id === 'precursor_drive') color = '#66ffff';
-      else if (floating.discovery.id === 'precursor_shield') color = '#66ff66';
-      else if (floating.discovery.id === 'precursor_nanotech') color =
'#ffff66';
-      else if (floating.discovery.id === 'factory_complex') color = '#ff9966';
```

```
-      else if (floating.discovery.id === 'mineral_deposits') color = '#ffcc66';
-      else if (floating.discovery.id === 'friendly_aliens') color = '#99ff99';
-      else if (floating.discovery.id === 'hostile_aliens') color = '#ff4444';
+      // Lookup discovery color via helper
+      const color = DiscoverySystem.getDiscoveryColor(floating.discovery.id);

       // Text with shadow (draw discovery name)
       ctx.fillStyle = 'rgba(0, 0, 0, 0.8)';
```

With this refactor, the exact same colors are used as before [4], but they come from a single lookup. Adding the static `getDiscoveryColor()` method greatly simplifies the rendering code and makes it easier to add new discovery types in one place.

1. **Optional – Text Shadow Helper** (Illustrative example): Extracting the duplicate "text shadow" drawing into a function. (This is not strictly necessary but demonstrates possible DRY'ing.)

```
--- a/client/src/game/GameUtils.js
+++ b/client/src/game/GameUtils.js
@@ -5,0 +6,8 @@
+  // Draw centered text with a one-pixel offset black shadow for visibility
+  static drawTextShadow(ctx, text, x, y, color, font = 'bold 14px Arial') {
+    ctx.save();
+    ctx.font = font;
+    ctx.textAlign = 'center';
+    ctx.fillStyle = 'rgba(0, 0, 0, 0.8)';
+    ctx.fillText(text, x + 1, y + 1);
+    ctx.fillStyle = color;
+    ctx.fillText(text, x, y);
+    ctx.restore();
+  }
```

```
--- a/client/src/game/DiscoverySystem.js
+++ b/client/src/game/DiscoverySystem.js
@@ -1624,6 +1624,5 @@
       ctx.fillText(floating.discovery.name, screenPos.x + 1, screenPos.y + 1);

       ctx.fillStyle = color;
-      ctx.fillText(floating.discovery.name, screenPos.x, screenPos.y);
-      ctx.restore();
+      ctx.fillText(floating.discovery.name, screenPos.x, screenPos.y);
+      ctx.restore();
```

```
--- a/client/src/game/DiscoverySystem.js
+++ b/client/src/game/DiscoverySystem.js
```

```
@@ -1624,3 +1624,6 @@
       // Text with shadow (was replaced)
-      ctx.fillStyle = color;
-      ctx.fillText(floating.discovery.name, screenPos.x, screenPos.y);
-      ctx.restore();
+      // Use shared utility to draw text with shadow
+      // GameUtils.drawTextShadow(ctx, floating.discovery.name, screenPos.x,
screenPos.y, color);
+    }
+  }
```

(In practice, you would replace the two fillText calls with one `GameUtils.drawTextShadow(...)` *call. This ensures identical rendering and reduces code duplication. We leave it commented/instructive here.)*

Ensuring Performance and Fidelity

- **Viewport Culling and Pools Remain**: None of the above changes alter the game loop or rendering order. All culling checks (`camera.worldToScreen` / `isVisible`) and object pooling logic stay intact. We only replaced inline calculations with utility calls.
- **Visual Output Unchanged**: The diffs only refactor code structure. Using `Math.hypot` or a lookup table has no perceptible effect on how things appear. All text alignment, fonts, and colors are preserved exactly (as seen in the before/after code excerpts).
- **Gradual Integration**: Because each change is local and additive, you can apply them one at a time. For example, add `GameUtils.distance` first and replace one usage; test thoroughly; then proceed to the next location. This minimizes risk of regressions.

Integration Steps

1. **Add Utilities**: Create or update **GameUtils.js** as shown, adding the `distance()` (and optional `drawTextShadow()`) methods. Also add `DiscoverySystem.getDiscoveryColor()`.
2. **Update StarThrone & AIManager**: In **StarThrone.js** and **AIManager.js**, replace every manual distance calculation with `GameUtils.distance(...)`. Add `import GameUtils from './GameUtils.js'` at the top if not already present [1] [2].
3. **Update DiscoverySystem**: Replace the color `if`-chain in *renderFloatingDiscoveries* with a call to `DiscoverySystem.getDiscoveryColor()` [4].
4. **Test in Isolation**: After each change, run the game to confirm behavior (e.g. start a game, trigger a discovery, etc.). The visual output and game mechanics should be identical.
5. **Refine UI Helpers**: If desired, replace other duplicate text-drawing code (loading screen, game-over screen, notifications) with the new text-shadow or panel-drawing helpers for future maintainability.
6. **Review Performance**: The refactored code should match or slightly improve performance (no new loops or heavy logic added). Ensure frame rate and memory use remain steady.

By following these steps, all existing functionality and optimizations (culling, pooling, AI staggering, etc.) are preserved. The result is a **more maintainable codebase** with redundant logic factored out into shared methods, without any change to gameplay or visuals [4] [1].

1 2 3 4 5 rockmeamadeus.txt

file://file-E3YJaEUxBhBRe9na93A2gY