ChatGPT

Designing Companion Task Delegation – Lessons from RimWorld, Tropico & Schedule 1

Overview of Task Delegation in Management Sims

In management/survival games, players often **assign tasks or roles to AI characters** (colonists, citizens, companions) and then rely on those NPCs to carry out the work automatically. Games like **RimWorld**, **Tropico**, and **Schedule 1** demonstrate different systems for delegating jobs (e.g. hauling, cooking, hunting, building) and handling them over time. Key design elements include how the player **issues orders or sets priorities**, how the AI executes tasks in the background, and what **risks or variable outcomes** come with delegation. Understanding these systems provides valuable insights for designing a companion-task system in *Old Trail*, an HTML5 roguelike survival RPG, where the player can entrust survival duties to AI companions.

Task Assignment Mechanics in Key Games

RimWorld: Priority-Based Work Scheduling

In RimWorld, each colonist has a **Work tab** where the player toggles which job types they can do and sets priority levels (1–4) for each. This creates a **priority matrix**: colonists will do all jobs marked with priority 1 (highest) before any priority 2 jobs, etc., moving left-to-right for ties. By default, jobs are either on or off, but with manual priorities the player can fine-tune a colonist's focus (for example, Doctor at priority 1, Cooking at 2, Hauling at 3, and never Hunting). Colonists then autonomously choose tasks based on these settings and current needs in the colony. Players can also **override** the AI temporarily by selecting a colonist and right-clicking a specific task ("Prioritize working on X"), which is useful in emergencies. This system emphasizes *planning over micromanagement* – you set the roles and the simulation handles minute-to-minute decisions.

User Experience: RimWorld's UI helps the player assign and monitor work effectively. The Work tab is a clear grid of pawns vs. task types, with visual cues about skill: a job box is shaded or outlined to indicate the pawn's expertise (e.g. red outline for very low skill, gold for high skill [1]). Hovering shows the exact skill level, and even a subtle **"crunching" sound cue** plays if you assign someone with abysmal skill to a task (warning of likely inefficiency) [2]. Once tasks are assigned, the player can see what each colonist is currently doing via their inspect panel or icons (e.g. a hauling icon). RimWorld also notifies the player of problems – for instance, if no one is assigned to a critical work type or a colonist is idle, a yellow alert appears. This feedback-rich UI ensures the player understands the AI's behavior and can adjust priorities on the fly.

Automation and Execution: Colonists perform delegated jobs continuously in the background as time progresses, as long as the work is available and they aren't sleeping or on break (RimWorld even has a separate **Schedule** tab where you can assign each hour of a colonist's day to Work, Recreation, Sleep, etc., to control their routine). The game's AI scheduler evaluates all pending tasks (like "haul this item to

stockpile" or "tend to injured ally") and assigns them to free colonists based on priority and proximity. Importantly, colonists **lack global efficiency insight** – they strictly follow priority rules even if it means doing something inefficient (e.g. hauling an item from across the map because Hauling is priority 1, before doing closer tasks of lower priority). This is a deliberate design trade-off: it rewards players for thoughtful priority planning and base layout, and it creates interesting challenges (colonists aren't perfectly optimal robots).

Tropico: Indirect Control and Workforce Management

The Tropico series puts you in charge of a city/nation, where task delegation is **less direct** but still crucial. You don't assign specific tasks to individual citizens; instead, you manage the economy and infrastructure so that NPC workers fill jobs and complete tasks **autonomously**. Delegation in Tropico happens by **constructing buildings** (which create job slots) and setting broad parameters like wages, work modes, or priority levels for construction projects. For example, if you place a Farm or Clinic, citizens with the required skills will take those jobs and perform farming or healthcare tasks over time. You can prioritize a building's construction so builders focus on it first, or raise a factory's budget to attract more workers and increase output.

System Design: Tropico abstracts tasks at the building level – each building has a staff that handles its function (growing crops, producing goods, etc.). As the player, you influence **who works where and how effectively**, rather than micromanaging daily chores. Key mechanics include education and specialization: certain jobs require educated workers (e.g. high school for managers, college for professors). If you lack specialists, productivity suffers or the position stays empty until you train or import someone. This creates a *specialist dependency* dynamic: your economy might rely on one good engineer or a few high-skill farmers. The game offers ways to mitigate this, like hiring foreign experts or enacting edicts to improve worker training. It's wise to maintain some redundancy – for instance, have multiple people with high Construction skill – because if your only top builder leaves or dies, building projects slow down significantly.

Risk/Reward Dynamics: While citizens in Tropico generally don't "fail" at tasks in the way RimWorld pawns might, their **efficiency and output vary** with their skill, happiness, and working conditions. A "skilled builder" in Tropico erects structures *much faster* than an unskilled one, so keeping experienced builders is gold. However, since you can't directly control individuals, you manage risk indirectly through policies. If you underpay or mistreat workers, they may become unhappy – Tropico models this with **strikes and protests**. Workers on strike abandon their duties, halting production at that building. The player is alerted (your advisor Penultimo will announce the strike), and you face a dilemma: pay the workers more, send the army to break up the strike (which can injure or kill protesters), or endure the shutdown until it resolves. This is a clear risk/reward scenario: neglecting worker needs can cripple your economy at a bad time, whereas investing in their happiness (through higher wages, safety edicts, housing, etc.) has upfront costs but reduces the chance of disruptive strikes. Additionally, disasters or events can remove key people – for example, rebels might kill your best generals, or a disease could decimate your educated populace – forcing the player to adapt by training replacements or shifting strategies. Tropico thereby demonstrates **macro-level task delegation**: you set up systems and incentives, and must anticipate that if a critical link in the chain fails, the simulation will make you deal with the fallout (often via narrative events like strikes, rebel attacks, or election pressures).

UX Considerations: Tropico's UI leans on dashboards and overlays. You have an almanac with data on jobs, unemployment, and individual worker satisfaction. Each building's panel shows its workers, efficiency, and

any issues (e.g. “Job Quality low” or “No worker – position unfilled”). Icons will appear in the world for problems (a protest sign above a building on strike, or a warning icon if a factory lacks raw materials). To delegate tasks effectively, players often use these cues: for example, seeing a construction site with no progress might prompt you to check if your construction offices are understaffed or too far away, or if builders are stuck satisfying needs. The game also lets you **assign a building manager** (a specific citizen who gives a bonus, like increased efficiency or decreased accident risk). Choosing a manager is a form of micro-delegation through the UI – you pick a person from a list, based on their traits, to fill that role. Overall, Tropico’s interface focuses on **high-level control** and feedback rather than fine manipulation, which suits its broader scale.

Schedule 1: Direct Task Roles & Automation

Schedule 1 is a satirical management sim (centered on a clandestine drug business) that offers another perspective on task delegation. Initially, the player-character can perform all tasks (growing plants, cooking product, packaging, selling) themselves, but this quickly becomes overwhelming. The game encourages hiring **NPC workers** to take over these duties. There are distinct employee types – e.g. Cleaners (who pick up trash), Botanists (plant and harvest crops), Handlers (transport goods and package drugs), and Chemists (synthesize the product). Once hired, these NPCs will **handle tasks on your behalf**, continuously and without further micromanagement, as long as they have the needed tools or stations. The player’s job is to assign them properly and keep the operation running by supplying materials and paying wages.

Task Assignment Mechanics: Schedule 1 uses an in-world, interactive approach to delegation. The player is given a **“management clipboard”** item which, when equipped, allows them to configure NPC assignments. To assign a worker to a task or area, you click on the NPC with the clipboard and then click on target objects in the environment. For example, to set up a **Cleaner**, you place a few trash bins in your building and use the clipboard to mark which areas (bins) the Cleaner is responsible for – they will automatically roam around those bins and keep those areas clean. For a **Botanist**, you similarly assign them to specific plant pots (up to 8 pots per botanist) by clicking the pots with the clipboard; the botanist will then water and harvest those crops continuously. You can even assign a storage shelf as their supply source so they automatically grab seeds and soil from there. Each worker type has a similar setup: Handlers get assigned to haul between certain stations (or deliver products to specific outputs), and Chemists are assigned to lab stations (each chemist can run up to 4 lab stations at once). This clipboard system is powerful but not immediately intuitive – it’s essentially a **diegetic UI** (a tool within the game world) for linking workers to the things they should work on [3] . The benefit is an immersive feel of “managing your crew” by physically walking around and setting up their duties, rather than navigating a spreadsheet menu.

Automation and Time: Once assignments are made, Schedule 1’s workers operate in **real-time** alongside the player. For example, if you’ve assigned a chemist to a meth cooking station, they will continuously produce batches as long as ingredients are available, freeing the player to focus on other tasks (like making deliveries or expanding the business). Delegated tasks persist over time – workers even persist across game sessions (they stay hired and assigned when you return). However, the player must remember to **pay their wages** by depositing money in each worker’s pay box regularly. If a worker isn’t paid, they won’t do any work and will just stand around idle (though notably, they *don’t quit and leave* just for not being paid). This introduces a light resource-management aspect: you can intentionally stop paying a worker during slow periods to save money (they’ll wait around unpaid, costing nothing, until you fund them again). It’s an interesting design choice that effectively lets the player “pause” a delegation without firing the NPC – useful early on when you might hire more people than you can constantly afford.

Risk/Reward Factors: Initially, Schedule 1 had relatively low risk in task delegation – employees always succeed at their jobs (a botanist will never randomly fail to water plants, for instance), and there wasn't much that could go wrong except running out of supplies or money. In fact, one of the game's early criticisms was that once you set up a smooth operation, **there was "no friction" or evolving challenge** to threaten your growing empire. The developers addressed this by introducing a **Rival Cartel** in an update: now, as your operation grows, events can occur such as rival gang members ambushing you or robbing your dealers, and police interventions ramping up. These changes mean that while your hired workers automate the grindy tasks (which is the reward – you earn income hands-free), you're now exposed to new risks like losing inventory or having NPCs temporarily taken out of commission (e.g. a dealer who is robbed loses all the product and cash they were carrying). If a key worker is incapacitated or jailed due to such events, the player might have to step in personally or hire replacements, adding a layer of strategy around **redundancy and security**. There's also a subtle skill element: the player character can technically do tasks more profitably in some cases (e.g. manually recycling trash yields money, whereas a Cleaner NPC who does it saves time but forgoes that extra cash). So the player must weigh the efficiency gain of delegation against potential downsides (slightly lower yield or new threats that target your operation).

Interface and Feedback: Schedule 1's approach to UI is unique in that many controls are in-world. Assignments are viewed by inspecting the NPC's "clipboard menu" – you see a list of stations or items they're linked to. This can be a bit opaque until you learn it. To help, the game provides visual indicators like colored outlines on objects when using the clipboard (for example, a dotted line connecting a botanist to each pot you assign them, as shown in the clipboard tutorial image). The NPCs themselves have color-coded elements (their bed sheets change color based on role, and their name tag shows their job), so you can tell at a glance who is a botanist vs. a chemist. Progress feedback is mostly emergent: you'll see plants growing and being harvested, products accumulating on shelves, trash bins being emptied, etc. If something is wrong – say, production has stopped – it's on the player to investigate (maybe the chemist ran out of ingredients because you forgot to assign a Handler to restock them). This design trusts the player to notice issues in the world, fitting the immersive style of the game. However, a few modern QoL features exist, like text messages from NPCs: for instance, after the Rival Cartel update, if a dealer is robbed, they will text you about the incident, serving as a direct alert. This mix of implicit and explicit feedback teaches the player to keep an eye on the operation, even when tasks are delegated.

Risk, Reward, and Failure in Delegated Tasks

One common theme across these games is the **risk/reward trade-off of delegation**. Letting NPCs handle chores spares the player effort and often increases efficiency, but it introduces variability and potential failure states that the player must manage. Below is a comparison of how each game handles outcomes and failures when tasks are assigned to AI characters:

Game	Skill Impact & Outcomes	Failure States & Risks	Consequences if Key NPC Lost
RimWorld	NPC skill directly affects speed and success. Low-skill workers are slow and may produce poor results (e.g. low-quality items). Some tasks have explicit failure chances – e.g. a low Construction skill can fail building (wasting materials), low Cooking causes food poisoning, low Medical can lead to surgery failure.	Injuries or incidents can occur during tasks. Hunting or taming animals has risk of the animal attacking if the pawn's skill is low (animal revenge chance increases). Construction failures waste resources; a doctor's failure can outright kill a patient. Mental breaks from low mood can also remove a pawn from work (they wander or go berserk).	Most tasks can be done by others at lower efficiency if a specialist is unavailable. However, if no one has a required skill, certain actions become very risky or impossible (e.g. complex surgeries without a good doctor). The game forces adaptation: you might train another colonist (slowly), or accept suboptimal outcomes. A lost expert creates drama but is recoverable – e.g. you can acquire prosthetics if your only builder lost an arm, or recruit/ convert a new expert over time.
Tropico	Workers have education levels rather than explicit "skill stats," but higher education or experience yields higher efficiency (faster production, more output). As the player, you can improve this via policies (education edicts, better pay to retain experienced workers). There's no concept of "failing" a task due to low skill – it just happens slower or not at all if no worker is qualified.	Failures in Tropico come from systemic issues or events. If workers are unhappy (low liberty, low wages, poor housing), they may strike, halting the task entirely until resolved. If you push them too hard (or ignore needs), you risk rebellion – in which case tasks stop because your island is in chaos! Also, random events like fires, crime, or export ships being delayed can disrupt progress.	If a key worker is gone (dies, emigrates, switches jobs), the building they staffed will slow down or stop. The player can respond by training new personnel (build schools, internships) or by importing experts (at a cost). Tropico often softens the blow by making new immigrants arrive periodically – so a vacant job eventually gets filled, just maybe not with the same efficiency. The bigger danger is if a whole class of specialists is missing (e.g. no doctors island-wide); then the player must urgently build education or face cascading issues (sickness untreated, etc.).

Game	Skill Impact & Outcomes	Failure States & Risks	Consequences if Key NPC Lost
Schedule 1	Workers don't have individual skill levels; they perform their role at a fixed effectiveness. The "skill" factor is mostly on the player's side (your character has a chosen specialty like better at trading or hunting), but employees just either do the task or not. Thus, outcome variability is low – a chemist always produces the same quality product. The main limiter is how many tasks one worker can handle (caps like 8 plants per botanist, 4 stations per chemist, etc. to simulate workload strain).	Early on, risks were minimal aside from running out of money or supplies. With updates, external threats add failure states: e.g. a **dealer NPC can be robbed** by rivals (you lose inventory and profits), or police can arrest the player (temporarily removing you from the operation). Workers might stop if unpaid, but they won't leave – they just idle uselessly (the "failure" in that case is on the player for forgetting to pay, resulting in lost productivity). There's no injury/health system for workers in Schedule 1, so you won't have an employee get sick or die randomly. The primary risks are **financial (losing money/product)** and **security (losing time recovering from raids or arrests)**.	If a key character (like the player or an important dealer) is unavailable, some parts of the operation halt. For example, if you (the only one who can drive deliveries) get jailed, distribution stops for a few days. If your only botanist isn't working, your supply of raw ingredients will dry up. The game allows hiring multiple workers of each type as redundancy, but early on you might have just one of each. Losing them means you must either do the task yourself (less efficient) or quickly hire a replacement (which costs money and time to assign). Because workers never permanently quit even if unpaid, you at least won't lose them to attrition – you have control over when they work or not. The Rival Cartel threats introduce the idea that you might need **backup resources** (e.g. extra stash of cash, or a weapon for self-defense) to handle periods when a worker's role is compromised by an event.

Lessons from Failures: These games teach that delegation doesn't mean you can ignore the task entirely – you trade direct control for strategic management. A few design takeaways:

- **Ensure the player has some Mitigation Options:** If a companion in *Old Trail* gets injured or fails at a task, the player should have ways to respond. In RimWorld, if your cook causes food poisoning, you can assign someone else to cook or switch to simple meals; if a hunter gets hurt, you can tend their injuries and maybe restrict hunting until healed. Similarly, *Old Trail* could let the player adjust by changing companion roles on the fly or using their own character to fill in during emergencies. Designing **fallback roles or cross-training** (perhaps each companion has a primary skill but a capable secondary skill) can prevent total collapse when one member is out.
- **Communicate Risk Upfront:** A common thread is that players are warned about low-skill assignments (RimWorld's red outline and crunch sound [1]) or about deteriorating conditions (Tropico's strike alerts). *Old Trail* should clearly telegraph if, say, you assign a companion with poor hunting skills to go hunt – perhaps a tooltip like "Chance of injury is high due to low skill" or an icon indicating risk. This allows the player to make informed choices about risk/reward (maybe you *still* send the novice hunter because you desperately need

food, but at least you knew the risks).
- **Embrace Narrative Consequences:** Delegated tasks failing can generate memorable stories – which is great for a survival RPG. RimWorld's emergent narratives (the one-eyed doctor botching a surgery and killing the patient, leading to colony grief) or Tropico's political dilemmas are part of their appeal. *Old Trail* can incorporate this by turning companion failures into events: e.g. "John went fishing but was bitten by a snake – he's laid up injured," or "Elena, while scavenging, found extra materials but also attracted a wolf pack." This adds depth to the world and ties the outcomes of automated tasks back into the player's narrative. Just be sure to keep such events fair and allow recovery – if every delegated task ends in disaster, players will simply never use the system. It's about occasional **drama and trade-offs**, not constant punishment.

Designing a Companion Task System for *Old Trail*

With the above insights, we can outline how a companion task delegation system might work in a browser-based survival RPG like *Old Trail*. The goal is to let players **assign survival tasks to AI companions** (for example, gathering firewood, hunting game, cooking meals, guarding camp) and have those tasks carried out over time, while maintaining the game's pace and challenge. Below are design recommendations drawn from the best practices of RimWorld, Tropico, and Schedule 1, tailored to an HTML5/Weebly context:

1. Define Clear Roles and Priorities

Give each companion a **role or set of tasks** they can focus on, and let the player define who does what. This could be as simple as a role selection (e.g. mark a companion as "Hunter", "Forager", "Guard", etc.) or a small priority list per companion similar to RimWorld's work priorities. For a survival RPG with a small party, you might not need numeric priorities – possibly a **toggle system** (each companion is either assigned or not assigned to a task type, and if multiple are assigned, they share the workload or the highest-skill one leads). What's crucial is that the player can adjust these assignments at any time (especially if conditions change or someone is injured). A **default AI** can fill in logical behavior (e.g. if nobody is explicitly the cook, maybe the least busy person will attempt it), but the player should feel they have direct control to optimize tasks.

Example: A possible UI is a "Party Management" screen listing companions down one side and tasks across the top (like a mini RimWorld work tab). The player checks boxes to assign roles – e.g. Alice ✔ Hunting, ✔Guard duty; Bob ✔Foraging, (X) not on Hunting. Under the hood, you then script the AI to automatically perform those tasks when appropriate (Alice will go hunt whenever there's daylight and game around, unless an urgent threat appears, etc.). If more granularity is needed, allow setting one task as high priority for each companion (similar to "current assignment") while others are secondary. This keeps it simple for a web UI while still offering control.

2. Skill-Based Outcomes and Progression

Following RimWorld's lead, integrate the companions' **skills or stats** into task outcomes. If companions have RPG-like attributes (e.g. a Hunting skill, Cooking skill, Survival skill), use those to modify the results of tasks: a higher skill yields faster completion, better quality results, and lower risk. Conversely, a low-skilled attempt might consume more time or resources, or have a chance of failure. For instance, a skilled cook makes a filling meal out of few ingredients, whereas a novice might spoil some ingredients or make a meal that only partially satisfies hunger. As companions perform tasks, they could **improve their skills over time**

(like RimWorld's experience gain by doing), rewarding the player for delegating and nurturing their team. This also encourages specialization – you might want one companion to keep hunting regularly to become an expert marksman.

However, *Old Trail* should also plan for **critical failures**: what is the chance a task goes horribly wrong and what happens? Borrowing from these games: maybe while chopping wood there's a small chance a companion gets a minor injury (cut arm) which then needs attention, or a hunting trip could rarely trigger a dangerous encounter (bear attack). The probability of such events can be tied to skill (high skill = lower chance of disaster) and perhaps terrain or scenario difficulty. This creates a risk element that keeps delegated tasks interesting – you can't just send someone out 100 times and assume 100% success; there's always that 1% chance of trouble that makes you consider, "Should I maybe go with them or prepare a backup plan?"

Make sure to surface these probabilities in the UI/UX. For example, if the player is about to send a companion on a multi-day foraging quest, you might show a brief "Expected outcome: 80% chance successful forage, 15% returns empty-handed, 5% chance of injury" based on that companion's stats and current conditions. This is similar to how some strategy games present mission odds. It manages expectations and allows the player to make informed decisions (perhaps they'll wait until the companion heals or equip them with better gear to improve odds).

3. Monitoring and Feedback Loop

When tasks are delegated, the player shouldn't be left guessing about their progress. Implement a **clear feedback loop** for ongoing tasks. This could include: - **Status icons or text** next to each companion's name (e.g. "Hunting...", "Cooking", "Idle") so at a glance the player sees what everyone is doing. This is akin to RimWorld's pawn status or Tropico's building info, but simplified for a small party.
- A **progress bar or timer** if tasks take substantial time. For instance, if building a shelter takes 3 in-game hours, an icon over the construction site or a progress bar in the UI can show how far along it is. In an HTML5 UI, you might update this with CSS animations or incremental fills – lightweight and effective visually.
- **Log messages or narrative updates** for key events. Old Trail already has a log system (`addLog` in the code) for events like hunting outcomes. Leverage that for companion tasks: e.g. "Day 12, 3:00 PM – Alice sets off to hunt." then later "Day 12, 6:00 PM – Alice returns with 15 lbs of game meat, but she sprained her ankle." This way, even if the player was busy managing inventory or traveling, they can read what happened. It's very much in the spirit of Oregon Trail's event narration and keeps the player engaged with the results of automated actions.
- **Alerts for problems**: If a delegated task cannot proceed, notify the player. Schedule 1, for example, doesn't explicitly pop up alerts when a worker is idle due to no supplies, but *Old Trail* might benefit from a subtle cue like the companion's task status turning red or a "Needs Attention" flag. E.g. "Bob could not complete cooking – no firewood" could appear, prompting the player to supply wood or reassign Bob to gather wood instead. This prevents frustration where the player assumes things are being handled when they actually stalled.

From a UX perspective, especially on Web/Weebly, keep these indicators **lightweight and responsive**. A simple HTML/CSS overlay or a small info panel can suffice; avoid cluttering the screen. The player should be able to scan and tell if all is well (all companions show green or normal statuses) or if something needs intervention (a red icon, an exclamation mark, or a log entry about trouble). Since the game runs in a

browser, using familiar web UI patterns (tooltips, colored badges, etc.) is a good approach – these are easy to implement and users intuitively understand them.

4. Flexible Intervention and Overrides

Even with a robust AI, players often want the ability to **step in and micromanage in critical moments**. RimWorld allows manual prioritization by draft commands; Tropico lets you quick-build or personally oversee construction via El Presidente; these ensure that when the player needs something done *now* or in a very specific way, they can make it happen despite the usual AI routine. In *Old Trail*, consider features to: - **Immediately assign a task to a companion**: e.g. a button "Prioritize: [Companion] do [Task] now." If a sudden situation arises (wolves circling camp), the player might click "Prioritize: John – defend camp" which would override John's current activity and have him stand guard with his weapon. This is essentially a high-level command that temporarily bumps a task to the top of that companion's queue. After it's done or after a certain time, John can revert to his normal duties. This way, delegation doesn't mean loss of control when it matters; the player can still orchestrate specific responses.
- **Reassign roles on the fly**: The system should be dynamic – if your cook falls ill with fever, the player might toggle another companion to be the cook for a while. There should be little friction in doing so (maybe as easy as clicking a checkbox or selecting from a dropdown of roles). The UI could warn if the replacement is unskilled ("[Companion] has no experience cooking [1] "), but ultimately allow it. This mirrors real survival scenarios where people have to step up beyond their usual roles when someone is out of commission.
- **Cancel or pause tasks**: If the player decides a delegated task is no longer safe or needed, they should be able to cancel it. For instance, you sent a companion to gather berries, but then a storm hits – you might want to call them back early. A "Recall" or "Stop Task" command would tell that companion to abandon what they were doing and return to base. In implementation, this might simply set their current goal to null, causing them to re-evaluate (and perhaps default to an idle or safety behavior). Schedule 1 had a rough equivalent with the wage system – not paying was effectively pausing the worker – but in *Old Trail* a direct cancel button is more straightforward for players to use.

Implementing these interventions in HTML5 might involve event-driven programming: e.g. attach onclick events to UI buttons that set a companion's state in your gameState. Since you likely already maintain game state for companions (as seen in `gameState.companions` array in the code), you can integrate flags like `companionStatus = "idle"/"busy: taskname"/"urgent override"`, etc. The AI loop can check those flags each tick to decide what action to take. This logic is lightweight and can run easily in JavaScript on the client.

5. Companion Morale and Condition

For a survival RPG, the human element is important. While Tropico deals with morale in a city-wide sense, *Old Trail* can incorporate **companion morale or health** affecting task performance. If a companion is starving, exhausted, or upset (maybe your decisions affect their mood), they might work slower or even refuse tasks (like a mini-strike). This adds a layer of strategy: you can't just work your people 24/7; you need to rest and feed them. You could represent this with simple stats like Energy and Mood for each companion. Low energy could automatically reduce their efficiency or increase failure chances (just as a tired RimWorld pawn works slower and is more prone to mistakes). Low morale could trigger events – perhaps a companion with very low morale might confront the player or consider leaving (if your design allows for that drama).

Be sure any such system is communicated and balanced. For example, a companion's portrait or name could have a small face icon indicating mood (happy, neutral, sad). Tooltips or a status page could list effects: "Tired: -20% to efficiency" or "Injured: cannot hunt until healed." This intersects with the risk system: if John got that snakebite, now he's injured which lowers his move speed and effectively takes him out of hunting until treated. The player then must adjust assignments (maybe have someone else hunt or reduce travel pace while John heals). These cascading effects simulate the Oregon Trail vibe (where if one member falls ill, the whole party might slow down) and make the simulation richer.

Crucially, because *Old Trail* is an HTML5 game possibly played in shorter sessions, consider **how persistence is handled**. If a companion is mid-task or recovering when the player saves/quits (or the Weebly page unloads), you'll want to save that state (e.g. "Alice is 50% through forage task" or "Bob has 2 days left of injury"). Using `localStorage` or IndexedDB on the browser can preserve these states so that when the player returns, the game picks up where it left off. This is especially important if tasks are long-running over in-game days.

6. Adaptation to Browser Performance & Frameworks

Implementing a colony-sim style AI in a browser is entirely feasible, but efficiency matters. Keep the number of simulated actors modest (which is naturally the case in *Old Trail* with a small party, not hundreds of NPCs). Use time steps wisely – e.g. update companion AI maybe 4–10 times per second (or even once per second for coarse tasks) rather than every millisecond, which is plenty for a strategy game pace. JavaScript can handle pathfinding and decision-making for a few agents easily, but avoid very heavy loops. If complex calculations are needed (pathfinding on a large map or simulating environment over time), consider using Web Workers to offload those so the UI thread isn't blocked. Web Workers allow a separate thread to run JavaScript – you could, for instance, have a worker simulate the world state tick by tick and send back updates to the main thread for rendering. This keeps the game feeling smooth, which is key in a web embed where any lag might be interpreted as the page freezing.

In terms of frameworks: if *Old Trail* is currently built with direct DOM and vanilla JS (as the code snippet suggests), you might continue that approach for the UI, especially since it seems to be working. But for more game-like interactivity (canvas rendering, animations, maybe a mini tile-based map), you could integrate a game library. **Phaser 3** is a popular choice for 2D HTML5 games and can manage a game loop, sprites, and physics easily. It's somewhat heavy if you only need UI and simple logic, but it excels at orchestrating game states and could handle things like a world map view where companions move around. Phaser also supports structured data and might simplify collisions or pathfinding if needed (with plugins). On the other hand, if the game remains mostly UI + text driven (like Oregon Trail style), a full game engine might be unnecessary overhead. In that case, using **PixiJS** just for rendering any animations or dynamic canvas elements could be beneficial as a lightweight option [4] [5]. Pixi gives you fast WebGL rendering for things like drawing a travel route or showing weather effects without needing a full engine.

Since the game is embedded on Weebly, also mind **responsive design**. The HTML/CSS approach shown in the file (with flex containers and percentage widths) is good – ensure that any new UI (like a task assignment panel or companion status display) scales or reflows on different screen sizes. Weebly sites might be viewed on tablets or phones, so test that the delegation interface is tappable and legible on smaller screens. Using standard HTML inputs/buttons for the task UI is actually a plus here (they are easily recognized and can be styled for mobile). If you do go with a canvas-based solution for parts of the game, use CSS or meta tags to handle scaling that canvas on mobile.

Finally, consider leveraging **existing patterns** for saving and updating data in a web game. Since Weebly might not allow a complex server backend, the game likely runs entirely client-side. This means saving to localStorage and perhaps exporting/importing save files for players. Make sure the companion system's state is part of that save data (their roles, any ongoing task progress, stats, etc.). You might use a JSON object stored as a string in localStorage for simplicity. And to avoid issues with accidental refresh/navigation, you could include an "Are you sure you want to exit? Unsaved progress will be lost." if the player tries to navigate away mid-session (or better, auto-save frequently so progress isn't lost).

UI & UX Best Practices Summary

To tie everything together, here are a few **best practices for the task delegation UI/UX** in *Old Trail*, distilled from the analysis:

- **Clarity and Simplicity:** Use clear labels and icons for tasks. A small panel titled "Companion Tasks" with checkboxes or dropdowns is immediately understandable. Avoid burying task assignment in too many sub-menus; it should be accessible in one or two clicks from the main travel screen (maybe a "Manage Party" button opens the assignments overlay). Short paragraphs or tooltips can explain what each task does, especially since players might not be familiar with terms – e.g. "Foraging – searches for edible plants and water" as a tooltip when hovering "Forage" task.
- **Visual Feedback for Skill Fit:** Imitate RimWorld's approach of showing skill fit at the point of assignment. For example, when the player tries to assign a task, you could visually indicate the companion's expertise: show their skill number, or color the text (green = good, red = poor). Even a message like "(Novice)" or "(Expert)" next to the task name in the UI can guide players to delegate wisely. This prevents frustration from unknowingly assigning someone unqualified – the game gently warns them upfront [1].
- **Non-Intrusive Alerts:** Integrate notifications in a way that suits the narrative tone. A hardcore alert box might break immersion, so consider diegetic or integrated alerts – e.g. a companion might *say* "I can't find anything out here!" in the log if their foraging turned up nothing, or an icon on the travel map might blink where a companion is in trouble. Tropico's approach of showing an icon over the problematic building or using the advisor's voice could translate into *Old Trail* as perhaps a sound cue (a whistle or shout when a companion needs help) along with a text log entry. As a web game, you could even use subtle animations (shake the companion's portrait if their status turns critical, etc.). The idea is to catch the player's attention without being annoying.
- **Testing and Iteration:** Because this is for an HTML5 game, you have the advantage of rapid iteration and gathering player feedback (possibly through web analytics or direct comments). See how players actually use the delegation system. If you find, for instance, that players never use a certain task or always ignore delegation in favor of doing things manually, that might indicate the UI is too cumbersome or the incentives aren't right. Use that data to tweak the design – maybe you need to streamline the assignment process or increase the benefit of delegating tasks. The best systems from other games often went through such tuning (RimWorld's devs added the manual priority mode because players wanted more control; Tropico added edicts to balance out issues with workers leaving jobs, etc.).

Conclusion

Designing a companion task delegation system for *Old Trail* can dramatically enrich the gameplay by incorporating strategy, planning, and emergent narrative – all while relieving the player from excessive

micromanagement. By learning from RimWorld's robust priority system, Tropico's management of worker happiness and indirect control, and Schedule 1's on-the-ground assignment tools (and the importance of introducing risk to automated tasks), we can create a balanced system. The companions in *Old Trail* should feel like real partners: they have strengths to leverage and weaknesses to mind, they can surprise you with both successes and mishaps, and they require a bit of care and direction from the player to truly shine.

By implementing clear role assignment, skill-based outcomes with meaningful risk/reward, a user-friendly interface for orders and feedback, and keeping everything performant in a browser environment, *Old Trail*'s delegation mechanics can become a standout feature. It will allow players to experience the drama of a wagon party trekking into the wilderness where everyone's contribution matters – and when someone falters, it's up to you as the leader to adapt and survive. With thoughtful design, the system will not only **handle tasks over time** but also deliver the kind of storytelling and player agency that define great survival RPG experiences.

Sources: The above analysis is informed by game design knowledge and specific examples from RimWorld [1], Tropico, and Schedule 1, including their documented mechanics and player guides. These principles have been adapted to fit the context of an HTML5 embedded game, ensuring both depth and approachability in the final design.

---

[1] File:Work priority example.jpg - RimWorld Wiki
https://rimworldwiki.com/wiki/File:Work_priority_example.jpg

[2] Work - RimWorld Wiki
https://rimworldwiki.com/wiki/Work

[3] How to use the management clipboard in Schedule 1 | PC Gamer
https://www.pcgamer.com/games/sim/schedule-1-management-clipboard/

[4] [5] Advanced HTML5 Game Development for Weebly_ A Comprehensive Guide.pdf
file://file-4icAQUg29rqWATz8a4sCDT