ChatGPT

Idle Fantasy RPG/MMO (Phaser.js) – Features & Development Plan

Key Features

- **Idle Mechanics:** Offline progression and automated battles (the game earns resources or fights while the player is away) [1]. Players gain gold, mana, etc. over time and through auto-combat. This follows classic idle-game design (like *AdVenture Capitalist*, which introduced offline earnings for player progress [1]).
- **RPG Stats & Progression:** Character attributes (Strength, Intelligence, etc.), hit points, mana, and damage values. Experience points and leveling unlock new abilities or stat increases. Players feel the "never-ending escalation" typical of idle RPGs [2] [1].
- **Equipment & Inventory:** Multiple gear slots (weapon, armor, accessories). Items have stat bonuses and rarity. An inventory system for consumables and loot. Players can manage gear (equip/unequip) and see stat changes in real time.
- **Dungeons & Combat:** Diverse dungeon areas with waves of monsters and bosses. Each encounter grants loot and XP. Combat is mostly automated: the player's character (and party, if any) fights monsters using stats and skills without manual input.
- **Save System (Local & Cloud):** Local storage for game state (using HTML5 `localStorage`). Ensure players can quit and later "resume seamlessly from where they left" [3]. Implement cloud saves (e.g. RESTful API or services like Firebase) as an optional feature to sync across devices. Phaser's Registry plugin can export game state to localStorage or a web API [4].
- **Desktop Experience:** Wrap the Phaser game in Electron (or NW.js) for a full desktop app [5]. This allows custom window sizes, title, and file system access. (Phaser provides a "Phaser Runtime" template for Electron integration [5].)
- **Multiplayer (Optional):** Plan for future real-time or asynchronous multiplayer. Could use WebSockets/Socket.io or a service like Firebase Realtime Database. Features might include cooperative dungeons, trading, or leaderboards.

Development Plan

1. **Project Setup & Scenes:** Create a Phaser 3 project (using npm or Phaser Editor). Define key Scenes (`Boot`, `Preload`, `MainMenu`, `GameScene`, etc.) by extending `Phaser.Scene` (Phaser docs). In `preload()`, load all assets; in `create()`, build the scene. For example, load images with keys:

   ```
   this.load.image('hero', 'assets/hero.png');
   this.load.image('background', 'assets/castle_bg.png');
   ```

   The first parameter is an **asset key** used in code [6]. In `create()`, place sprites:

```
this.add.image(400, 300, 'background');
this.add.sprite(100, 200, 'hero');
```

This key-based system means you can swap art later by replacing files (and keeping the same key) [6].

2. **Asset Pipeline & Art:** Collect open-license medieval fantasy assets (sprites, tiles, icons). Use CC0 sources like Kenney's assets or vetted OpenGameArt packs. For example, use a castle background (as shown below) for scenery.

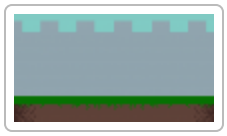

Figure: Sample medieval castle background (open-source). Use similar CC0/PD art for environments and UI elements.

3. In `preload()`, load backgrounds, character spritesheets, UI icons, etc.
4. To swap assets, simply update the image file and path in your loader; reference by the same key [6].

5. **User Interface (HUD and Menus):** Design HUD elements (health bars, resource counters, buttons) using Phaser GameObjects or DOM. For example, preload icon images and use `this.add.image()` or `this.add.text()` for UI. Make interactive buttons for inventory and settings (`setInteractive()`). Display stats and resource values on-screen. The UI should reflect the medieval theme (e.g. ornate frames, fantasy fonts from free assets).

6. **Idle Resource & Combat System:**

7. **Resource Generation:** Implement a timed loop to add resources. For example:

```
this.time.addEvent({ delay: 1000, callback: () => { player.gold +=
goldPerSecond; }, loop: true });
```

This yields passive income. Adjust rates via upgrades.

8. **Automated Combat:** In `GameScene.update()`, process combat logic. Spawn enemies (Phaser Physics Groups) in waves. On each tick, automatically apply player damage to enemies and vice versa. Handle enemy death: grant loot and XP. Because this is idle, allow the player to watch battles or switch away while it runs.

9. **Offline Progress:** When the player exits, record the timestamp and current state. On next load, calculate elapsed time and simulate gains (resources and even experience from auto-fights). As the Phaser saving tutorial notes, saving progress lets players "leave the game, shut down their browser, then come back... and resume seamlessly" [3].

10. **Player Stats and Leveling:** Model the player's stats in a JavaScript object (or class). Include base attributes (e.g. STR, AGI, INT) and derived stats (HP, attack power, magic power). When the player gains XP and levels up, increase stats or allow skill point allocation. Display updated stats in the UI. Store stats in a central game-state object that will be saved/loaded.

11. **Inventory and Equipment:**

12. **Item Data:** Define item templates (type, stats, sprite key). Use JSON or JS classes.
13. **Inventory Structure:** Use an array or list to hold item instances. In the UI, create a grid of inventory slots (Sprites or DOM elements). Load item sprites with `this.load.image('item_sword', '...')`.
14. **Equipping Gear:** Define equipment slots (weapon, armor, etc.). When the player clicks or drags an item onto a slot, update the player's stats by adding the item's bonuses. E.g. equipping a sword adds to damage.

15. **UI Interaction:** Make inventory icons interactive (`sprite.setInteractive()` and handle click/drag events). To swap art later, replace the spritesheet or images, keeping the keys consistent [6].

16. **Dungeons & Levels:**

17. Create dungeon scenes or tilemap levels. Use Phaser Tilemap (or static images) to design rooms.
18. When the player enters a dungeon, switch to a Dungeon scene. Spawn enemies appropriate to the dungeon level. After clearing waves, allow exit or transition to next level.
19. Boss battles: spawn a stronger enemy with unique attack pattern. Rewards are greater.

20. Ensure the game constantly offers progression (new dungeons unlock on level up or item finds).

21. **Game Loop & Animations:** In each game loop update: handle movement, combat ticks, and resource timers. Use Phaser Animations: define animations (`this.anims.create(...)`) for player attack, enemy death, etc. Play animations on sprites as actions occur. Physics/overlap checks (if using Arcade physics) manage interactions. The game's polish comes from smooth animations and UI feedback.

22. **Saving and Loading:** Use Phaser's data systems or direct Web Storage. Periodically (or on change), serialize the game state (player stats, inventory, current level, resources) into JSON and save:

    ```
    localStorage.setItem('gameSave', JSON.stringify(gameState));
    ```

    On startup, check for existing `localStorage` data and `JSON.parse()` it to restore state. Phaser's Registry plugin also provides exporting to localStorage or a RESTful API [4]. For cloud save, send the JSON to a backend endpoint (`fetch('/api/save', { method: 'POST', body: saveData })`) and load similarly. Ensure saved data includes timestamps to calculate offline progress.

23. **Desktop Packaging:** Use Electron to wrap the game as a desktop application. The official "Phaser Runtime" template makes this easy [5]. Clone the template, replace its game files with yours. In

Electron's `main.js`, set the window size/title. This lets your Phaser game run outside the browser with native app behavior (local file access, etc.).

24. **Optional Multiplayer:** If enabling multiplayer later, set up a server (e.g. Node.js). Use WebSocket libraries (Socket.io) to sync player actions (like trading items or grouping for dungeons). Structure game so that the core logic (stats, inventory) can be sent over the network. You might reuse the save system for persistent player accounts on the server.

25. **Testing & Iteration:** Continuously test each system. Verify that idle gains feel rewarding and not too fast/slow. Ensure equipment correctly modifies stats. Check save/load thoroughly (the Phaser saving tutorial emphasizes resuming "seamlessly" [3]). Profile performance: use Phaser's debug tools for physics, and optimize sprite counts. Collect feedback to balance the RPG progression curve.

By following this plan, you'll implement a full-featured medieval fantasy idle-RPG/MMO game in Phaser. Each system (stats, equipment, combat, save) is built step-by-step. Asset loading uses keys so that art can be swapped easily [6]. Offline progression is handled via saved timestamps [3] and Phaser's storage/registry tools [4]. The result will be a desktop-ready idle fantasy game with MMORPG-style depth.

Sources: Concepts and code examples are based on the Phaser 3 docs and tutorials [6] [3], and established idle/MMO game design principles [2] [1] [4] [5].

---

[1] [2] Incremental game - Wikipedia
https://en.wikipedia.org/wiki/Incremental_game

[3] How to save and load player progress with localStorage - Dynetis games
https://www.dynetisgames.com/2018/10/28/how-save-load-player-progress-localstorage/

[4] [5] Phaser World Issue 65
https://phaser.io/newsletter/issue-065

[6] Making your first Phaser 3 game
https://phaser.io/tutorials/making-your-first-phaser-3-game/part2