Концепт и ТЗ

Привет, мне надо создать некую интерактивную онлайн игру. Будет она выглядеть как сайт на react/nodejs, который бы, на первой стадии, хостился с на github pages с моего аккаунта.

Игрок только один - пользователь. Игра посвящена инвестициям и способности человека правильно аллоцировать активы. Механика следующая:

Есть три раунда.

В начале каждого раунда человеку докладывается описание некоторой ситуации, что происходит или будет происходить в мировой экономике.

Например: "производители золота решили создать свой аналог ОПЕК, который сразу выпустил пресс-релиз о намерениях сократить производство"

Эти ситуации должны быть

1) С долей юмора, быть нескучными

2) Мы их с тобой отдельно сгенерим

3) Должны быть четко идентифицируемы с точки зрения влияния на классы активов - какие активы будут точно в плюсе, какие точно в минусе, какие - нейтрально отреагировали бы на данное событие, произойди оно в реальном мире.

4) Должны быть достаточно емкими, чтобы пользователь не тратил по 10 минут на прочтение.

Далее человеку предлагается движением ползунков определить аллокацию активов после прочтения новости. Классы активов определим отдельно, но как минимум, туда должны входить акции (возможно, разбить на венчур/не венчур, growth/value), облигации (investment grade/high yield), фьючерсы на товары, фьючерсы на валюту.

Далее человек нажимает кнопку "Раунд".

На основе выведенного пользователю системой события, игра заранее знает - что в данной конкретной ситуации золото вырастет в цене, остальные активы более-менее нейтрально будут. Для каждого заранее созданного события мы заранее, видимо в json, укажем "правильный ответ" - какие типы активов вырастут, какие будут нейтрально, какие упадут.

После нажатия на кнопку система генерирует случайные блуждания, которые как бы моделируют поведение цен на каждый класс активов. Если мы знает, что при данном событии должен быть рост цен на актив А - случайное блуждание будет иметь положительное матожидание приростов. Если нейтральное влияние на класс активов А - то будет матожидание ровно 0, если отрицательное влияние - отрицательное матожидание. Дисперсии приростов классов активов мы зададим в файле, они должны быть близки к реальным.

По итогам симуляции и на основе выбранной аллокации - пользователю выводится график совокупной динамики портфеля и каждого из активов. Пользователю объясняется (данное объяснение должно быть заготовлено заранее) - какие виды активов выросли, какие упали и почему.

Пользователю дается 3 раунда игры. По итогам игры идет сравнение с первоначальным капиталом и делается ввод - насколько человек успешен в инвестициях (без сильного унижения)

Представь, что я - домохозяйка, которая не сильно понимает в технологиях программирования, но может копировать/вставлять код куда надо. Код я планирую писать с помощью ИИ (chatGPT) - на соответствующих подшагах пиши, какой конкретно запрос я должен подать ИИ, чтобы получить требуемый код. Пиши прямо с самого начала и до самого конца - какие директории создать, как периодически сохранять проект на гитхабе, какие файлы создать и что в них скопировать и тд.

Ниже приведён подробный план из 60 шагов, который проведёт тебя от нуля до полноценного интерактивного сайта-игры. План составлен максимально подробно, чтобы даже если ты не специалист в программировании, можно было копировать и вставлять готовый код, используя помощь ИИ (например, ChatGPT). Каждый шаг включает, что нужно сделать, какие директории создать, когда сохранить проект на GitHub, и какие конкретно запросы можно отправлять ИИ для получения нужного кода.

Подготовительный этап

1. **Анализ требований:**
   Прочитай описание проекта: игра с 3 раундами, где игрок получает новость (с юмором) о мировой экономике, выбирает аллокацию активов с помощью слайдеров, запускается симуляция динамики активов, выводится график и итоговое объяснение.
2. **Регистрация на GitHub:**
   Если у тебя ещё нет аккаунта, зарегистрируйся на GitHub.
3. **Создание репозитория:**
   На GitHub создай новый публичный репозиторий, например, с именем `investment-game`.
4. **Клонирование репозитория:**
   На компьютере клонируй репозиторий с помощью команды в терминале:

```
git clone https://github.com/твой-аккаунт/investment-game.git
```

6. **Переход в директорию проекта:**
   Перейди в папку `investment-game`.
7. **Инициализация Node.js проекта:**
   В корневой директории запусти:

```
npm init -y
```

   Это создаст файл `package.json`.

9. **Создание директории для фронтенда:**
   В корне проекта создай папку `client`.
10. **Инициализация React-приложения:**
    Перейди в папку `client` и запусти:

```
npx create-react-app .
```

    Это создаст стандартную структуру React-проекта.

12. **Структурирование фронтенда:**
    Внутри `client` создай дополнительные папки:
    - `src/components` – для компонентов
    - `src/assets` – для изображений и других медиа
    - `src/utils` – для утилит и вспомогательных функций
    - `src/styles` – для файлов стилей
13. **Настройка .gitignore:**
    В корне проекта создай или обнови файл `.gitignore`, включив в него стандартные исключения для Node.js и React (например, папки `node_modules`, `build`, `.env`).
14. **Первый коммит:**
    Сохрани текущий прогресс:

```
git add .
git commit -m "Initial project setup with React and Node.js"
git push origin main
```

Проектирование основных компонентов

12. **Определение компонентов игры:**
    Выдели следующие основные компоненты:
    - **Landing Page** – стартовая страница
    - **NewsRound** – компонент для отображения новости
    - **AssetAllocation** – компонент для выбора аллокации активов (слайдеры)

- **Simulation** – компонент, запускающий симуляцию случайных блужданий
- **PortfolioChart** – компонент для отображения графиков
- **RoundExplanation** – компонент для пояснения результатов раунда
- **FinalResult** – итоговая страница с оценкой успеха

13. **Создание схемы взаимодействия:**
    Нарисуй схему (на бумаге или в любом графическом редакторе) – как будут переходить состояния между компонентами и как они будут взаимодействовать через общее состояние.
14. **Формирование новостных ситуаций:**
    Придумай несколько примеров новостей (с юмором). Например:
    "Производители золота решили создать свой аналог ОПЕК и объявили о намерении сократить добычу – ждем рост цены на золото!"
15. **Подготовка файла с событиями:**
    В корне или в отдельной папке (например, `data`) создай файл `events.json`, где будут храниться описания событий и «правильный ответ» (напр., какие активы растут, какие падают, а какие нейтральны).
16. **Запрос к ИИ для events.json:**
    Спроси ChatGPT:

    > «Напиши пример файла events.json, где описаны события с влиянием на активы (например, золото растёт, акции нейтральны, облигации падают), чтобы можно было использовать его в игре.»
    > Скопируй полученный пример в файл `events.json`.

Реализация компонентов

17. **Создание компонента NewsRound:**
    В папке `src/components` создай файл `NewsRound.js`.
18. **Запрос к ИИ для NewsRound.js:**
    Спроси ChatGPT:

    > «Создай React-компонент NewsRound.js, который считывает описание новости из файла events.json и отображает его на странице.»
    > Скопируй полученный код в `NewsRound.js`.

19. **Проверка компонента NewsRound:**
    Импортируй и отобрази NewsRound в основном приложении (например, в App.js) для проверки работы.
20. **Установка react-router-dom для маршрутизации:**
    Перейди в папку `client` и установи библиотеку:

```
21.  npm install react-router-dom
```

22. **Настройка маршрутизации:**
    Спроси ChatGPT:

    > «Напиши пример настройки маршрутизации в React с использованием react-router-dom для приложения с тремя раундами.»
    > Скопируй полученный код и интегрируй его в файл `src/App.js`.

23. **Создание компонента AssetAllocation:**
    В `src/components` создай файл `AssetAllocation.js`.
24. **Запрос к ИИ для AssetAllocation.js:**
    Спроси ChatGPT:

«Создай React-компонент AssetAllocation.js, который отображает слайдеры для выбора аллокации активов (например, акции, облигации, фьючерсы).»
Скопируй полученный код в `AssetAllocation.js`.

25. **Создание файла стилей для AssetAllocation:**
    В папке `src/styles` создай файл `AssetAllocation.css`.
26. **Запрос к ИИ для CSS:**
    Спроси ChatGPT:

    «Напиши CSS стили для компонента AssetAllocation.js с простым и понятным дизайном.»
    Скопируй полученные стили в `AssetAllocation.css` и подключи его в `AssetAllocation.js`.

27. **Создание компонента Simulation:**
    В `src/components` создай файл `Simulation.js`.
28. **Запрос к ИИ для Simulation.js:**
    Спроси ChatGPT:

    «Создай React-компонент Simulation.js, который принимает выбранную аллокацию активов и моделирует их динамику с помощью случайных блужданий.»
    Скопируй полученный код в `Simulation.js`.

29. **Создание утилиты для случайных блужданий:**
    В `src/utils` создай файл `randomWalk.js`.
30. **Запрос к ИИ для randomWalk.js:**
    Спроси ChatGPT:

    «Напиши функцию на JavaScript, которая генерирует случайное блуждание с заданным матожиданием и дисперсией.»
    Скопируй полученный код в `randomWalk.js`.

31. **Интеграция randomWalk в Simulation:**
    В файле `Simulation.js` подключи и используй функцию из `randomWalk.js` для моделирования динамики активов.
32. **Выбор библиотеки для графиков:**
    Реши, какую библиотеку использовать (например, Chart.js). Перейди в папку `client` и установи её:

```
33.  npm install chart.js react-chartjs-2
```

34. **Создание компонента PortfolioChart:**
    В `src/components` создай файл `PortfolioChart.js`.
35. **Запрос к ИИ для PortfolioChart.js:**
    Спроси ChatGPT:

    «Напиши React-компонент PortfolioChart.js, используя Chart.js (или react-chartjs-2) для отображения динамики портфеля и отдельных активов.»
    Скопируй полученный код в `PortfolioChart.js`.

36. **Интеграция графика в Simulation:**
    В `Simulation.js` импортируй PortfolioChart и отображай график после симуляции.

37. **Создание компонента RoundExplanation:**
    В `src/components` создай файл `RoundExplanation.js`.
38. **Запрос к ИИ для RoundExplanation.js:**
    Спроси ChatGPT:

    > «Создай React-компонент RoundExplanation.js, который выводит заранее подготовленные объяснения результатов раунда (какие активы выросли/упали и почему).»
    > Скопируй полученный код в `RoundExplanation.js`.

39. **Создание компонента FinalResult:**
    В `src/components` создай файл `FinalResult.js`.
40. **Запрос к ИИ для FinalResult.js:**
    Спроси ChatGPT:

    > «Напиши React-компонент FinalResult.js, который отображает итоговый результат игры и сравнивает финальный капитал с первоначальным, при этом не унижая пользователя.»
    > Скопируй полученный код в `FinalResult.js`.

Управление состоянием и логикой игры

39. **Создание глобального состояния:**
    В `src/utils` (или создай папку `src/context`) создай файл `GameContext.js` для хранения состояния игры (выбранные аллокации, результаты раундов и т.д.).
40. **Запрос к ИИ для GameContext.js:**
    Спроси ChatGPT:

    > «Напиши пример реализации GameContext.js с использованием Context API для хранения состояния игры (например, текущий раунд, выбранные аллокации, результаты симуляции).»
    > Скопируй полученный код в `GameContext.js`.

41. **Оборачивание приложения в GameContext:**
    В `src/index.js` или в `src/App.js` оберни приложение в `<GameContext.Provider>`.
42. **Настройка логики перехода между раундами:**
    Определи, как будут сохраняться результаты каждого раунда и переходить к следующему. Планируй обновлять состояние в GameContext после каждого раунда.
43. **Создание обработчика кнопки "Раунд":**
    В компоненте AssetAllocation или Simulation добавь кнопку «Раунд», которая будет запускать симуляцию.
44. **Запрос к ИИ для обработчика события:**
    Спроси ChatGPT:

    > «Напиши обработчик клика для кнопки 'Раунд' в React, который запускает симуляцию (вызывает функцию случайного блуждания) и обновляет состояние игры через Context API.»
    > Скопируй полученный код в нужный компонент.

45. **Обновление состояния после симуляции:**
    После завершения симуляции сохрани результаты (например, динамику портфеля) в GameContext.

Тестирование и доработка интерфейса

46. **Локальное тестирование:**
    Запусти приложение в режиме разработки командой:
```
47. npm start
```
    и проверь, как работают все компоненты.

48. **Отладка ошибок:**
    Если возникнут ошибки – вноси исправления в соответствующие файлы, руководствуясь сообщениями в консоли.
49. **Проверка навигации между раундами:**
    Убедись, что маршрутизация (react-router-dom) корректно переключает страницы (NewsRound, AssetAllocation, Simulation, RoundExplanation, FinalResult).
50. **Настройка отзывчивости (responsive design):**
    Проверь, как выглядит интерфейс на разных устройствах, при необходимости добавь медиа-запросы в CSS.
51. **Доработка дизайна:**
    Добавь дополнительные стили, если необходимо – можешь попросить ИИ:

    > «Напиши улучшенные CSS стили для компонента [название компонента] для современного и чистого интерфейса.»

Документация и контроль версий

51. **Создание README:**
    В корневой папке проекта создай файл `README.md` с описанием проекта, инструкциями по запуску и основными функциональностями.
52. **Документирование кода:**
    В каждом компоненте добавь комментарии, чтобы было понятно, что делает каждая часть кода.
53. **Периодическое сохранение изменений:**
    После значимых изменений делай коммиты в Git:
```
54. git add .
55. git commit -m "Описание изменений"
56. git push origin main
```
57. **Создание веток (опционально):**
    Если планируешь экспериментировать, создай отдельные ветки для новых фич.

Развертывание на GitHub Pages

55. **Установка gh-pages:**
    В папке `client` установи пакет для деплоя:
```
56. npm install gh-pages --save-dev
```
57. **Настройка package.json для деплоя:**
    В файле `client/package.json` добавь следующие строки:
    - Параметр `"homepage": "https://твой-аккаунт.github.io/investment-game"`
    - В секцию `scripts` добавь:
    - `"predeploy": "npm run build",`
    - `"deploy": "gh-pages -d build"`
58. **Проверка сборки проекта:**
    В папке `client` выполни:

```
npm run build
```

чтобы убедиться, что сборка проходит без ошибок.

60. **Деплой на GitHub Pages:**
    В папке `client` выполни:

```
npm run deploy
```

    Это загрузит сборку на GitHub Pages.

62. **Проверка сайта:**
    Перейди по адресу, указанному в параметре `homepage`, и проверь работу сайта.
63. **Финальные правки и анонс:**
    Проведи финальное тестирование, собери обратную связь (если есть тестовые пользователи) и внеси доработки. Обнови документацию в README, чтобы другие (или ты сама в будущем) понимали, как поддерживать проект.

Следуя этим шагам, ты получишь полностью рабочий сайт-игру, где пользователь сможет проходить раунды, выбирать аллокацию активов, видеть симуляцию динамики и получать пояснения к результатам. В каждом из шагов используй конкретные запросы к ChatGPT для получения готовых фрагментов кода, копируй их в соответствующие файлы и не забывай периодически сохранять изменения в GitHub. Удачи в разработке!