演習 1 - GitHub Copilotコーディングエージェント

技術的負債に悩まされない組織はほとんどありません。未解決のセキュリティ課題、更新が必要なレガシーコード、実装の時間が取れずバックログに積まれたままの機能要望などが典型です。GitHub Copilotコーディングエージェントは、コード更新や機能追加といったタスクを自律的に実行するための機能です。エージェントの作業が完了すると、人間の開発者がレビューできるドラフト PR を作成します。これにより、単調な作業をオフロードして開発を加速し、開発者はより本質的な作業に集中できます。

本演習では、Copilotコーディングエージェントを用いて次の点を学びます。

- コード生成のための環境をカスタマイズする
- 安全性を担保しながら操作を実行する
- スコープが明確な Issue の重要性を理解する
- Issue を Copilot に割り当てる

シナリオ

Tailspin Toys には、すでに対処したい技術的負債があります。サイトの初期版を作成した外部委託の開発者が、ドキュメントをほぼ残していません。まずはアプリケーション内のすべての関数に docstring（または同等のドキュメント）が付与されている状態にしたいと考えています。

加えて、デザインチームはゲーム管理の UX 構築に着手できる状態にしたいと考えています。完全な実装はまだ不要ですが、受け入れテストに使えるエンドポイントは必要です。具体的には、ゲーム API に「作成・更新・削除」を行うためのエンドポイントが必要です。これは現状のボトルネックですが、他にも優先度の高い Issue がある状況です。

これらはどちらも優先度が下がりがちなタスクの好例であり、Copilotコーディングエージェントに割り当てるのに最適です。Copilot に任せて非同期で進めることで、開発者は他の作業に集中し、後から Copilot の成果をレビューして期待どおりか確認できます。

GitHub Copilotコーディングエージェントの概要

GitHub Copilotコーディングエージェント は、人間の開発者と同様にバックグラウンドでタスクを実行できます。人間の開発者と働くときと同じように、GitHub の Issue を Copilot に割り当てることで開始します。割り当てられると、Copilot は進捗を追跡するためのドラフト PR を作成し、環境をセットアップしてタスクの実行を開始します。作業中や完了後にセッションの内容を確認でき、提案がレビュー可能になれば、PR 上であなたにメンションが届きます。

スコープが明確な指示の重要性

つい魔法のように感じてしまいますが、GitHub Copilot は魔法ではありません。短い一文二文の指示で、AI がすべてのタスクを完璧にこなすわけではありません。実際には、単純に見える作業であっても、詳細に踏み込むと多くの複雑さがあります。

そのため、Copilotコーディングエージェントへのタスクの割り当て方には配慮が必要です。通常は AI ペアプログラミングとして Copilot と協調するのが最良です。タスクが大きくても小さくても基本は同じで、段階的に進め、学び、試行錯誤しながら調整します。生成 AI を導入しても、ソフトウェア開発の基本原則は変わりません。

Copilotコーディングエージェント用の開発環境セットアップ

コード作成には、誰が書く場合でも、適切な環境と準備のためのスクリプトが必要です。これは Copilot にタスクを割り当てる場合でも同じで、Copilot もソフトウェアエンジニアと同様の手順で作業します。

Copilotコーディングエージェントには事前に実行される専用ワークフロー（**.github/workflows/copilot-setup-steps.yml**）を設定できます。これにより、必要な開発ツールや依存関係へアクセスできるようにします。本ラボでは事前に設定済みで、Copilot が Python、Node.js、クライアントとサーバーの依存関係にアクセスできるようにしています。

```
name: "Copilot Setup Steps"

# リポジトリの Actions タブからセットアップ手順を試すためのトリガー
on: workflow_dispatch

jobs:
  copilot-setup-steps:
    runs-on: ubuntu-latest
    # 必要最低限の権限を設定します。Copilot の操作には Copilot 専用のトークンが付与されます。
    permissions:
      # 依存関係のインストールなどでセットアップ中にリポジトリをクローンする場合は `contents: read` が必要です。
      # セットアップでクローンしない場合は、この手順の後で Copilot が自動的に行います。
      contents: read
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      # Backend setup - Python
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.13"
          cache: "pip"

      - name: Install Python dependencies
        run: ./scripts/setup-env.sh

      # Frontend setup - Node.js
      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: "22"
          cache: "npm"
          cache-dependency-path: "./client/package.json"

      - name: Install JavaScript dependencies
        working-directory: ./client
        run: npm ci
```

他の GitHub ワークフローファイルと同じように見えますが、いくつか重要な点があります：

- 単一のジョブ **copilot-setup-steps**が含まれています。このジョブは、Copilot が Pull Request で作業を開始する前に GitHub Actions 上で実行されます。
- また、 **workflow_dispatch** トリガーを追加しており、これによりリポジトリの Actions タブからワークフローを手動で実行できるようになります。これは、 Copilot が実行するのを待つ代わりにワークフローが正常に動作するかをテストするのに便利です。

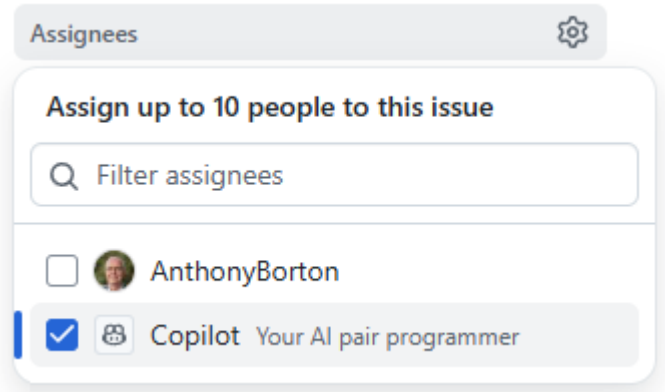
コードドキュメントの改善

すべての開発者と組織がドキュメントの重要性を理解している一方で、ほとんどのプロジェクトでは情報が古くなっているか、そもそも不足していることが多いです。これはしばしば放置される技術的負債の一種であり、生産性を低下させ、コードベースの保守や新しい開発者のチーム参加を難しくします。幸いなことに、Copilot はドキュメント作成に長けており、これは Copilotコーディングエージェントに割り当てるのに最適な課題です。エージェントはバックグラウンドで必要なドキュメントを生成します。今後の演習でその生成物をレビューします。

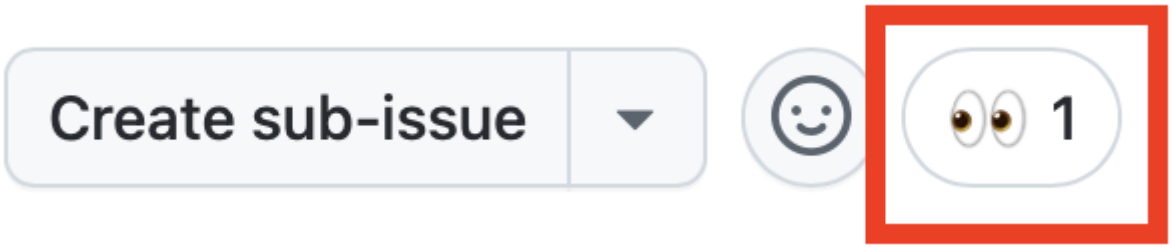
1. **新しいブラウザタブ**を開き、github.com 上の自分のリポジトリに戻ります（ヒント： se-copilot-workshops/*yourhandle*)。
2. **Issues** タブを開きます。
3. **New issue** を選んで新規 Issue ダイアログを開きます。
4. テンプレート選択が表示されたら **Blank issue** を選びます。
5. **Title** を コードのドキュメンテーションが不足している に設定します。
6. **Description** を次の内容に設定します。

   > 私たちの組織では、すべての関数に `docstring` （または同等のもの）を付けることを義務付けています。しかし、最近の更新ではこの基準が守られていません。既存のコードを更新して、すべての関数やメソッドに `docstring` （または同等のもの）が含まれるようにする必要があります。

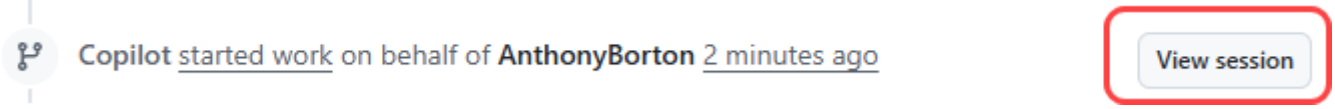
7. **Create** を選択して Issue を作成します。Create ボタンを確認するために下にスクロールする必要がある場合があります。
8. 画面右側の **Assignee** を選択し、このリポジトリの関係者の検索ボックスを開きます。
9. **Copilot** を選択し、Issue の担当者に設定します。



10. ページ内の他の箇所をクリックして担当者ウィンドウを閉じます。しばらくすると、Issue の最初のコメントに 👀 が表示され、Copilot が作業中であることが示されます！



11. **Pull Requests** タブを開きます。
12. 新しく作成された Pull Request を開きます。そのタイトルは **[WIP]: コードのドキュメンテーションが不足している** のようになっているはずです。
13. **数分待つと**、Copilot が進捗を示すチェックリストを PR に追加します。
14. そのタスクリストを確認します。Copilot はこのリストを使って進捗を共有します。
15. タイムラインを下にスクロールすると、Copilot が作業を開始した更新が表示されます。
16. **View session** ボタンを選択します。



> [!IMPORTANT] 画面を更新しないと表示が変わらない場合があります。

17. ライブセッションをスクロールして、Copilot がどのように問題を解決しているかを確認してください。コードの調査や状態の理解、適切な計画を立てるために一時停止して考える様子、そして実際にコードを作成する様子が含まれます。

これは数分かかる可能性があります。 Copilotコーディングエージェントの主な利点の一つは、非同期でタスクを実行できることにより、私たちが他の作業に集中できるようにする点です。この機能を活用して、別のタスクを Copilotコーディングエージェントに割り当て、その間にアプリケーションに機能を追加するコードを書いていきます。

新しいエンドポイントを作成してゲームを変更できるようにする

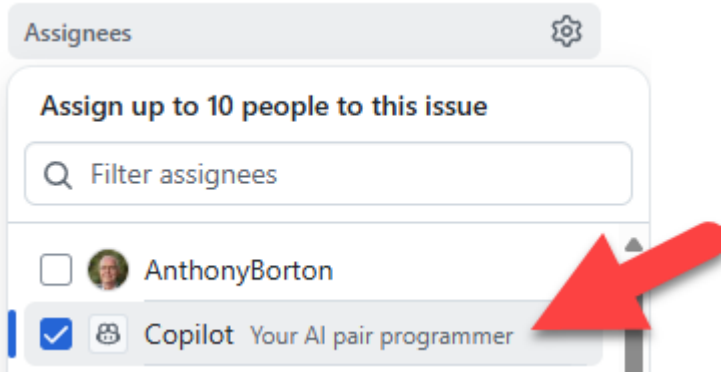
前述したように、GitHub Copilotコーディングエージェントの大きな利点の一つは作業を分担できることです。あなたは一連のタスクに集中し、エージェントは別のタスクに集中できます。デザインチーム向けにゲームを変更するためのエンドポイントを作成する作業は長時間かからないかもしれませんが、それでも他の作業に使える時間です。そこで、この作業を Copilotコーディングエージェントに任せましょう！

1. github.comの自分のリポジトリに戻ります。
2. **Issues** タブを開きます。
3. **New issue** を選んで新規 Issue ダイアログを開きます。
4. テンプレート選択が表示されたら **Blank issue** を選びます。
5. **Title** を ゲームを作成・編集するエンドポイントを追加する に設定します。
6. **Description** を次の内容に設定します：

```
- Games API に、ゲームの作成・更新・削除を行う新しいエンドポイントを追加する
- すべての新規エンドポイントに適切なエラーハンドリングを実装する
```

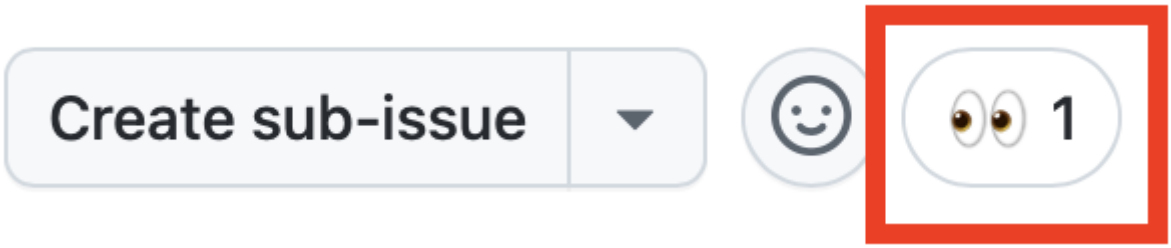
- すべての新規エンドポイントに対しユニットテストを作成する
- PR を作成する前に、すべてのテストがパスしていることを確認する

7. Copilot がうまく作業を行うことができるような十分な指示を与えている点に注目してください。
8. Issue 右側の **Assignee** ボタンを見つけます。
9. **Assignee** をクリックして担当者選択のダイアログを開きます。
10. **Copilot** を選択します。



11. **Create** を選択して Issue を作成します（下部にスクロールが必要な場合があります）。
12. 作成した Issue が表示されます。

まもなく、最初のコメントに 👀 が表示され、Copilot が作業を開始したことが分かります！



> [!IMPORTANT] 画面を更新しないと表示が変わらない場合があります。

13. **Pull Requests** タブを開きます。
14. 生成された PR（例: **[WIP]: ゲームを作成・編集するエンドポイントを追加する**）を開きます。
15. 数分後、Copilot が作業内容のチェックリストを PR に追記します。Copilot はこのチェックリストで進捗を共有します。
16. タイムラインを下にスクロールすると、Copilot が作業を開始した更新が表示されます。
17. **View session** ボタンを選択します。

18. ライブセッションをスクロールし、Copilot がどのように問題を解決しているかを確認できます。コードの探索や状態の把握、思考のステップ、計画の策定、コード作成の様子が見られます。

Copilot は現在、あなたのリクエストに対して着実に作業中です。Copilotコーディングエージェントはソフトウェアエンジニアと同様に動作するため、常時監視は不要で、完了後にレビューすれば十分です。ここからはコード作成と他の機能に目を向けましょう。

まとめと次のステップ

本レッスンでは、AI の同僚である GitHub Copilotコーディングエージェント を紹介しました。Issue を Copilot に割り当てて非同期に作業させ、技術的負債の解消・新機能の作成・フレームワーク移行の補助などに活用できます。

以下のトピックを学びました：

- コード生成環境のカスタマイズ
- 安全に作業が行われていることの確認
- スコープが明確な Issue の重要性
- Issue の Copilot への割り当て

バックグラウンドでコーディングエージェントが動いている間に、外部サービスと連携する MCP サーバーの活用 に進みましょう。Copilotコーディングエージェントは MCP サーバーも利用できます が、次は Codespace に戻り、Copilot のエージェントモードと MCP を組み合わせて試してみます。

リソース

- Copilotコーディングエージェントの概要
- GitHub の Issue を Copilot に割り当てる
- Copilotコーディングエージェント用セットアップワークフローのベストプラクティス

次の演習に進むには以下をクリックしてください。 次の演習: Copilot のエージェントモードと GitHub MCP サーバーでバックログを準備する