Teaching and Learning with Scratch: Semi-Structured Interview Protocol

1. **Background Info**
    a. Which age groups do you predominantly work with?
    b. Do you work through a specific organization? Does that organization have standards through which you teach programming?
    c. How long have you been teaching computer science to children?
    d. How long have you been learning computer science yourself?
    e. Are there any features in any other interfaces you've taught with (Code.org, etc). that you found especially helpful with teaching computational thinking?
    f. What are some methods you've used to teach computational thinking concepts that have worked best for students (e.g., analogies)?
2. **Instructional Paradigm**
    a. How do you introduce computational thinking concepts? Do you provide direction instruction?
    b. What types of assignment materials and scaffolding do you provide students to supplement Scratch?
    c. If Scratch offered the capability to embed those materials and scaffolding in the environment, would you use it?
3. **Gameful Learning**
    a. What types of games do your students most frequently create in Scratch
    b. How are game construction assignments structured so that students learn computational thinking concepts?
    c. If Scratch offered puzzles and games that students could play in order to learn computational thinking, would you assign those games?
    d. Would you design them?
4. **Formative Assessment**
    a. What types of formative feedback do you provide students?
    b. How frequently do you provide formative feedback to students?
    c. If you could provide objectives for students to accomplish within Scratch, would you do so?
    d. If Scratch could provide hints based on students' progress, how would you prefer those hints be delivered? Should the hints arrive just-in-time or on-demand?
5. **Summative Assessment**
    a. How do you assess students' Scratch projects? Do you use rubrics?
    b. Have you considered using automated assessments to complement manual inspection?
    c. How do you assess students' mastery of computational thinking?
    d. How do you differentiate instruction for students at different levels of mastery?
6. **Call to Action**
    a. Which elements of Scratch make teaching computational thinking easier?
    b. How could students learn computational thinking more effectively and efficiently from an environment like Scratch?

c. How could teaching with an environment like Scratch be simpler?