Beckham Carver

Final Project Proposal

Web Application Development

CS 4210

11/12/23

Part 1: Project Proposal

1. **Project Title:** Wiki Surfer

2. **Project Description:**

    - Wiki Surfer is a way to gamify interacting with Wikis. Wiki Surfer translates wiki pages into platformer levels and allows you to ‘surf’ along them traveling between pages. This creates dynamic level generation where information and the format of its presentation is treated as level design. This translation from one medium to another creates an experimental environment for studying how the layout of information unintentionally affects gamified interaction.

3. **Features and Functionality:**

    - Level generation: by inputting a link to a Wiki page, Wiki Surfer then transforms its elements into level elements, ie. platforms, pipes, and obstacles. You can then traverse the page as a character, entering links to new pages.
    - Character control: you control a ‘surfer’ character which allows you to traverse the transformed page.
    - Timer: either dynamically created, or predefined, you have a goal ‘link’ to find and must traverse to it as quickly as possible. The time from one link to another (or a chain of links) will be your score.
    - Level creation: when you’ve found a good WIki page for platforming, you can set a starting link, and goal link(s) to create a ‘level’ and upload it to the site.

4. **Information to Store in the Database:**

    - Premade levels: link to a starting Wiki, and the expected link(s) to visit and the ending link. As well as a goal time to beat, name, and maker, and country.
        i. Can optionally host level.html files for non Wiki levels, ie. test levels
    - Uploaded levels: identical to premade levels, with attributes for likes/dislikes, and time/date uploaded
    - Highscores: stored similarly to classic arcade games, each level contains a highscore table that stores the top 10 best times, user selected name, and country for those times.

5. **Database Integration:**

- Being that the app needs minimal information stored in the DB, I plan to use something lightweight such as MongoDB.
- On the main page the DB will be queried for premade levels to display for selection, as well as a ‘user’ made levels tab.
- The user levels will be displayed along premade levels, and could be sorted by highest rated or newest.

6. **Technology Stack:**

    - I plan to use PixiJS as the behind the scenes game engine, as it has solid text support- and it has integrated support for React. PixiJS uses JS and HTML5.
    - I also plan to use Express to host the server, as that’s been our usage in class.
    - As mentioned earlier I’ll use MongoDB for storing levels and user data.
    - For loading Wiki’s I’m hoping I can just get by with FETCH requests and build from the returned HTML. If not, nearly every Wiki in the world follows the open source standard wikimedia which has an API… in PHP.

7. **Credentials Management:**

    - Users will not have accounts to hold with the arcade/early-internet theme, therefore users cannot directly interface with the DB, so the server will need to be packaged with credentials for the DB. Requests to post to the DB will be sent to the server which will handle uploading.

Part 2: Development Plan

1. **Project Timeline:**

    - I won't have time to start in earnest until roughly Nov 18th, so I'll plan to work primarily during thanksgiving week. My goal will be to have a bare minimum platformer completed before the 22nd to get feedback. With its features benign similar to the platformers shown in PixiJS examples and have it construct a basic level from a local level.html file
    - After that point it will be crunch time, I will need to focus on reaching minimum functionality of the necessary features before tuning or refining them.

2. **Development Stages:**

    - Create a platformer: using the examples in PixiJS create a basic platformer that uses a level.html as input.
    - Level linking: create level elements from links in the level.html that point to a new level.html (later a FETCH) and load it when touched.
    - Level selection: create a section of the main page that interacts with the server to get a new level.html, later will fetch to DB
    - Dynamic levels: finalize functionality to create levels from Wiki fetch requests, add input from users for Wiki links.
    - Integrate DB: and integrate with the selection page add sorting if time allows, with a page for users to send level structures, this will need to verify that the structure is valid before upload.

3. **Design and User Interface &**
4. **Interactivity and Navigation:**

    - The UI will contain two primary sections, a game window section and a static section.
        i. The static section will contain links to premade or popular levels, as well as an input form to start from.
        ii. If communication with PixiJS game states are easy to implement, there could also be a dynamic ledger in the static section, containing the current and previous sites/links visited and timer. Otherwise these can be displayed in game.
        iii. The game window section will contain the level constructed from the input link or selected level. It will look primarily like a standard Wikipedia page with additional space between elements, and borders denoting platforms.

iv. In its simplest form, the game window will simply allow traversal, and level linking, if time allows additional obstacles can be parsed from the Wiki
v. Beyond that interaction is primarily described in the project description; you control a character and traverse a Wiki site, traveling through links (pipes likely) as you see them.
vi. For traversing vertically between sections of text, I will have to decide the best method through (limited) testing. My candidates are to give the side boundaries collision and jump reseting for climbing/sliding, or adding some elevator platform system similar to Mario titles.

Part 3: Risks and Challenges

1. **Identify Potential Risks:**
    - Standard Wikipedia usage of PHP may be a large obstacle when trying to parse for level creation.
    - Dynamic generation of any type will have bugs, and the timeline for testing/finding them will be short.
2. **Mitigation Strategies:**
    - Explain how you plan to address these risks and challenges. What contingency plans do you have in place?
    - If Wikipedia is too gross and API to work with, I can look to other sites for content to transform, such as University sites. I can also work directly from the HTML and ignore additional formatting or API features.
    - For dynamic generation, the premade levels will be an easy way to curate for parsable links/pages. I also have the option to send level.html files directly to the user, and utilize premade html exclusively and modify how users upload levels to accompany the shift in style.