How will I complete this project?

This project is connected to the Developing Scalable Apps with Python courses, but depending on your background knowledge you may not need the entirety of the course to complete this project.

Game API Project Overview

In the Developing Scalable Apps with Python course you learned how to write platform-agnostic apps using Google App Engine backed by Google Datastore.
In this project you will use these skills to develop your own game! You will write an API with endpoints that will allow anyone to develop a front-end for your game. Since you aren't required to write a front-end you can use API explorer to test your API.Let's get started!

Task 1: Explore the Architecture

Get the skeleton 'Guess a Number' application up and running. Read through the code and documentation, and test the endpoints with API explorer. Make sure you understand how different entities are created, how they work together, and the overall flow of a game. Create a `User` or two and play a few games. Make sure to take a look at the admin Datastore viewer to check out the various entities. The datastore is typically at http://localhost:8000, but you may need to access it on another port.

Task 2: Implement Your Own Game

Come up with a new game to implement! This could be a more advanced guessing game such as Hangman, or a simple two player game like Tic-Tac-Toe. Note: Implementing a copy of Guess a Number, such as Guess a Date, will not be acceptable, we want you to be creative!
Consider how the existing endpoints will work with the new game - you'll need to modify the models, forms, and resource containers, but the general structure should stay roughly the same so that Games can be created, moves played, and the game state updated and stored according to the rules.
Here are some ideas to get you started:

One-Player Games:
- Hangman)
- Solitaire)
- Concentration)

Two-Player Games:
- Tic-Tac-Toe
- Battleship)
- Mancala
- Boggle
- War)
- Yahtzee - This could be a one player game too

Note for two-player games: These could be designed so that two players can play against each other, or they could be single player games played against an AI player of your design.

Score Keeping

Define what a "score" for each game will be and keep this data in your database. For example in 'Guess a Number' the Score model stores the number of guesses taken before the number was found. Two player games do not need to implement this feature.
You can record any other data that you think is interesting or relevant to your particular game.

Task 3: Extend Your API

A well engineered backend should be extensible. Let's test that theory and get additional practice working with the Datastore by implementing several new endpoints.
You may need to customize them somewhat to fit the specifics of your game.
Ensure that each new endpoint uses an appropriate HTTP Method. For example accesses to data should be be performed with GET requests.
Finally, these endpoints should be documented in your README just like the ones for Guess a Number.
- get_user_games
- This returns all of a User's active games.
- You may want to modify the `User` and `Game` models to simplify this type of query. Hint: it might make sense for each game to be a `descendant` of a `User`.

- cancel_game
- This endpoint allows users to cancel a game in progress. You could implement this by deleting the Game model itself, or add a Boolean field such as 'cancelled' to the model. Ensure that Users are not permitted to remove completed games.

- get_high_scores
- Remember how you defined a score in Task 2? Now we will use that to generate a list of high scores in descending order, a leader-board!
- Accept an optional parameter `number_of_results` that limits the number of results returned.
- Note: If you choose to implement a 2-player game this endpoint is not required.

- get_user_rankings
- Come up with a method for ranking the performance of each player. For "Guess a Number" this could be by winning percentage with ties broken by the average number of guesses.
- Create an endpoint that returns this player ranking. The results should include each Player's name and the 'performance' indicator (eg. win/loss ratio).

- get_game_history
- Your API Users may want to be able to see a 'history' of moves for each game.
- For example, Chess uses a format called PGN) which allows any game to be replayed and watched move by move.
- Add the capability for a Game's history to be presented in a similar way. For example: If a User made played 'Guess a Number' with the moves: (5, 8, 7), and received messages such as: ('Too low!', 'Too high!', 'You win!'), an endpoint exposing the game_history might produce something like: [('Guess': 5, result: 'Too low'), ('Guess': 8, result: 'Too high'), ('Guess': 7, result: 'Win. Game over')].
- Adding this functionality will require some additional properties in the 'Game' model along with a Form, and endpoint to present the data to the User.

Task 4: Improve Notifications

In the skeleton Guess a Number project, a cron job and associated handler have been created (see cron.yaml and main.py). This sends an hourly reminder email to every User with an email address to try out 'Guess a Number'. This is probably annoying the users.
Modify the SendReminderEmail handler so that this reminder email is only sent to users that have incomplete games (or some other logic that makes sense to you). Make sure to update the message to reflect this.

Optional Improvements:

- If you're feeling ambitious you can implement more sophisticated notifications. For example: "If the User has not made a move in an active game for more than 12 hours, send a reminder email that includes the current game state."

- If you created a two-player game, you can implement a turn notification system! When one user makes a move, add a task to the task queue to notify the User's opponent that it's their turn. You can use the `SendReminderEmail` handler in main.py as a template. Remember that you will need to pass parameters to identify the Game and User that should receive the reminder. Don't forget to update `app.yaml` with the new Handler listing. Finally, consult Google App Engine documentation for Using Push Queues in Python.

Task 5: README and API Documentation

Be sure to document your game. Your README file should include:

- Instructions for playing the game
- Detailed descriptions of each endpoint

Remember, you are documenting an API that another programmer may want to use as the basis for a web or mobile app. An api user should not need to read the source code to understand how to use it. You may follow the format of 'Guess a Number' for your README.

Reflect on Your Design

Document your design decisions by answering the following questions:

- What additional properties did you add to your models and why?
- What were some of the trade-offs or struggles you faced when implementing the new game logic?

These answers should be in a file Design.txt. Your responses can be in paragraph form or bulleted lists. This document should be around 500 words long.
View it here - https://github.com/udacity/FSND-P4-Design-A-Game
Please note: While launching chrome to test your API, you will have to launch it using the console as follows:
[path-to-Chrome] --user-data-dir=test --unsafely-treat-insecure-origin-as-secure=http://localhost:port