Hello World Proposal: Side Scroller Platformer Game

After starting the implementation of my ‘Hello World’, I realized my original proposal wasn’t going to be feasible. The point at which I feel will be infeasible to implement in time is the addition of the AI race-mode.

In my original proposal, I proposed that I make a side scroller platformer game that had multiple levels and difficulties to choose from. In the ideas for the various modes/difficulties I proposed a time-trial mode, which implemented an AI that the player was to compete against to make it to the finish line first. However, as of starting implementation of the game, I have found that it will be difficult for me to meet the deadline of the Thanksgiving while implanting this, since I have no prior experience in creating AI. Additionally, in my proposal, I suggested the possibility of adding in a level maker; for the level maker, I propose that it can replace the AI in priority, since it seems more feasible given the time constraints. However, I still put it as merely a possibility and not a certainty of getting added before Thanksgiving.

My main plan forward is as follows (this is what follows after the code I’ve already implemented):

1. Create platforms with collision detection, along with a start and goal platform for level completion. As well as obstacles, like spikes or kill platforms.
2. Using a .txt file as a way to store and read the platform information for levels.
3. Create a GUI for level selection, application quitting, other options (i.e. don’t know what the buttons will be, could be something like instructions), and game start-up. Also plan to use this to add buttons for going back to level selection from in-game level (i.e. quit level).

\*\*\*Everything after this is only a possibility if there is enough time left\*\*\*

1. Time high score, achievements, currency for purchase of skins/backgrounds (list may not all be implemented in the same step, like currency possibly being added after step 6.).
2. Difficulty and mode implementation (i.e. faster left and right speeds, time trial, etc.).
3. Level maker.
4. Game mode for competition against an AI.

The main library I will be using is pygame, which can handle everything mentioned in the definite plan. However, I also plan to use the pygame\_gui library to do the GUI for the game, since it allows for easier and cleaner implementation of buttons, graphics, and etc.

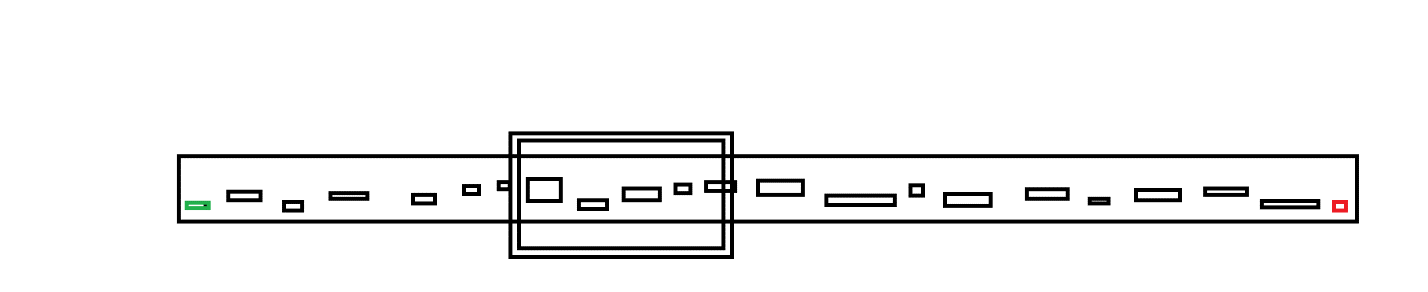
For the levels, their information will be stored as a .txt file in one of the folders in the project folder, most likely named “levels”, which will be the positions and type of platforms present in each level. For example, if I have a simple level with a start platform a floating platform and a goal platform, it would be stored as something like:

0 0 400

1 50 400

2 100 400

The first line reads: make a start platform (0) at coordinates (0, 400) on the screen where the tuple is in the form of (x, y). The second line reads: make a floating platform (1) at (50, 400). The last line reads: make a goal platform (2) at (100, 400). In this way, I could store levels in a compact way and quickly and easily read them in using text file I/O. Additionally, it allows for easy writing of levels since it would simply write to a text file and store it in a certain area to be later played under a possible “custom levels” section. (As a note, the example for what I could write in the text file for a level is just a basic example, I would add in more numbers to determine the overall size of the platform as well as other factors that I haven’t thought of yet.)

For levels that are longer than the screen is wide (i.e. have platforms with coordinates that are off the screen in the x-direction), there would be a way to implement it so that the method that writes the platforms to the screen would check to see if the x-coord is greater than the coordinate range of the screen and keeps track of which portion of x-values should be looked at. That is, if I move my character past a certain distance, the platforms from behind, at a certain distance, get unloaded, and the platforms up ahead, at a certain distance, get loaded onto the screen. To better image this, I have this poorly drawn image: 

In the box is all the platforms that are loaded at one time and the others aren’t loaded until either their left-most x-coord is inside the box, or their right-most x-coord is inside the box. This is how I would implement the levels.