15-112 TP3 No Updated Sections

15-112 TP2 Updated Sections (all other sections remained unchanged)

(sarahc2)

Project Description

My Term Project is called: Ho Ho Home: the Santa Maze Game.

Background Story

It is the last few hours of Christmas, and there are only a couple more presents left to deliver. However, Santa's reindeer just contracted a novel XMAS-20 virus that can only affect reindeer (don't worry, all of Santas are showing positive signs of recovery), and they can't help Santa deliver his last presents! Even though the reindeer told him the fastest way, Santa immediately forgot and must now figure out his way to town, i.e. solve the maze, on his own. Help Santa redeem his bad memory and prove that he's still got what it takes to deliver presents to every last kid!

Player Perspective

Players will begin by being given the background story and instructions. From there, players will choose a sled based on what features of the game they want to play with. Based on the chosen sled, the player will enter the sled's respective mode. Though each mode has different features (i.e. compromised visibility or a pathfinding Grinch), the ultimate goal for each is still the same - solve the maze as quickly and accurately as possible. Santa loses 10 presents every minute it takes to solve the maze, and the final screen will display how many presents out of 100 Santa was able to deliver. The telephone line, aka "cheat mode," can be enabled where Santa "calls the reindeer to ask for the correct path," i.e. the solution path will be revealed through a series of dots.

Structural Plan

main.py:

- Calls active screen
- Each screen will be a subclass of a superclass: MyApp that contains parameters per screen & sets whatever screen the user is on to be "active"

gameScreens.py:

- Title Screen, Story/Instructions, Sled Descriptions, *ability to call different game modes*, Final Screen
- Buttons that allow the user to freely move between screens
- Essentially will have all of the project "fluff" and draw functions, i.e. what makes the game more than just a maze

mazeModes.py:

- Includes the 3 sled modes with different maze attributes

mazeGenerationAndSolution.py:

- Includes all functions that have anything to do with creating and solving the maze

Algorithmic Plan

I found the trickiest part of my project so far to actually be the movement of the Grinch (moving the Grinch to the location of the sleigh). This meant that I needed to create a new solution list that included the ordered path from the Grinch's cell to the sleigh's cell. One issue I had was that my maze dictionary (mapping nodes to successive nodes) was one directional, i.e. if my recursive backtracking mapped point A to B to C, then my dictionary would not recognize that C was connected to B which was connected to A. Hence getting from any

arbitrary point to another was very difficult if that path included any bit that was backtracked, i.e. the path was not a part of the maze generation's first get-go. I spent honest hours and hours trying to figure out and implement a solution to this complex problem (the solution seems pretty evident now but coming up with it was much harder than anticipated), and ultimately, I figured it out! Here's what I did:

From my previous work, I knew that I had a list of the solution from the sleigh's starting point (0,0) to the end cell of the maze (n-1,n-1) for an nxn maze. So, my idea was essentially to split the path from the Grinch to sleigh into two lists: (1) Grinch to end, and (2) Sleigh to end. Then, with these two paths, I could loop through each until I found a common point, i.e. the cell of which their paths intercepted (let's call it cell X), and cut off the remaining points of each list (duplicates). Finally, given two lists (Grinch -> cell X, sleigh -> cell X), I could pop cell X from one of the lists to remove a duplicate (Grinch -> cell X, sleigh -> cell before cell X), reverse the sleigh to cell X list (Grinch -> cell X, cell before cell X -> sleigh), and finally combine them to produce a full solution (Grinch -> cell X).

Now my problem was, how do I get from any given cell, i.e. the Grinch's cell or the sleigh's cell (let's generalize this to cell Y), to the end cell with only a one-directional maze dictionary and a solution list (let's call this list S) from (0,0) to (n-1,n-1)? Instead of solving this question, I further cut down my problem to finding a solution from cell Y to any point in the list S. If I could find a path from cell Y to a coordinate in list S (let's call this solution list L), I could just concatenate list S, from that coordinate to the end, to my list L. Thus I would be left with a working solution from any arbitrary cell Y to the end cell.

Hence, my complex problem could be cut down to "simply" being able to find a list L from an arbitrary cell Y to a cell in list S (let's call cell Z), given a one-directional maze dictionary. This issue was difficult again because of the reason mentioned above: that cell A being mapped to cell B was not recognized as cell B being mapped to cell A given the directionality of my maze dictionary. Hence, I solved this problem by first creating a "flippedDict" which mapped all the values of my maze dictionary to their keys. Then using the my maze solver function from TP1 that implemented recursive backtracking to find a solution list, I first tried to solve cell Y -> cell Z using my original maze dictionary. If this didn't work, then the path from cell Y -> cell Z could be found using my maze solver function but this time with my flippedDict.

Finally to summarize, a solution list L from an arbitrary cell Y to cell Z in list S implied that I could find my two paths, Grinch to (n-1,n-1) and sleigh to (n-1,n-1). Given these two paths, I could loop through, find their point of intersection, and combine the two lists to find one complete path from the Grinch to the sleigh. Yay!

I implemented this function (along with its countless helper functions) by calling it every time the timer was fired and either the Grinch or the sleigh's position was changed. The solution list gave me insight on which direction the Grinch needed to move at any given moment in order to get to the sleigh, and thus was my pathfinder algorithm.

15-112 TP1 Project Proposal

(sarahc2)

Project Description

My Term Project is called: Ho Ho Home: the Santa Maze Game.

Background Story

It is the last few hours of Christmas, and there are only a couple more presents left to deliver. However, Santa's reindeer just contracted a novel XMAS-20 virus that can only affect reindeer (don't worry, all of Santas are showing positive signs of recovery), and they can't help Santa deliver his last presents! Even though the reindeer told him the fastest way, Santa immediately forgot and must now figure out his way to town, i.e. solve the maze, on his own. Help Santa redeem his bad memory and prove that he's still got what it takes to deliver presents to every last kid!

Player Perspective

Players will begin by being given the storyline, aka instructions. From there, players will customize their "Santa" and choose a sled based on what features of the game they want to play with (The new, modern sled is bigger / loses presents at a slower rate, and has better headlights / visibility. However, it attracts the Grinch which will take all of the presents if he finds Santa. The older, more discreet sled is smaller / loses presents faster and can only see a small radius around the sled, but this sled does not attract the Grinch because of its small size. The final third sled is No Presents, Just Vibes. Santa accidentally forgot the presents at the North Pole and has decided to cut his losses, not turn back, and see if he can find his way to the houses just for the fun of it. The Grinch is irrelevant in this mode because without presents, Santa has nothing the Grinch wants anyway.). Based on the sled the player chooses, they will enter the sled's respective mode. Though each mode has different features, the ultimate goal for each is still the same - solve the maze as quickly and accurately as possible. The faster and more accurately the maze is solved, the more presents Santa was able to successfully deliver to the quiet households. The telephone line, aka "cheat mode," can be enabled where Santa is allowed to call the reindeer to ask for the correct path (the AI solved solution will be highlighted on the maze).

Competitive Analysis

For my competitive analysis, I found three main online games that were related to my TP:

1. Standard Maze Challenges: The basic goal behind these games is similar to the fundamental goal of my project: solve the maze. However, my TP will include various difficulty features, e.g. limited visibility, enemy chaser, etc., and be centered around a Christmas storyline.
2. Pacman: Pacman is similar to one of the game modes, i.e. sled choices in my TP. Both games involve an enemy (i.e. ghost or Grinch) following the player (i.e. pacman or Santa) in some sort of maze setting. However, my project will differ from Pacman because the game ends when Santa has exited the maze, whereas Pacman ends when he eats all of the dots.
3. Finally, MapCrunch: MapCrunch is a game on GoogleMaps that places the player in a random location's street view, and the player must find his or her way back to an airport in order to survive. This game is similar to one of the game modes of my TP because both involve wandering through a maze setting with limited visibility and an aim to reach a certain location. My TP will be different from MazeCrunch in the fact that the paths of my project are of randomly generated mazes and they change every time the game restarts.

Structural Plan

Main python file:

- Title Screen, Story/Instructions, Santa Customization, Sled Descriptions, *ability to call different game modes*, Final Screen
- Buttons that allow the user to freely move between screens
- Each screen will be a subclass of a superclass: MyApp that contains parameters per screen & sets whatever screen the user is on to be "active"
- Essentially will have all of the project "fluff" and draw functions, i.e. what makes the game more than just a maze

Maze file:

- Superclass: Random Maze Generation, Maze Solver
- Subclasses: Each sled mode will be its own class and implement different features based on the mode (one will have a Grinch, a timer, and presents remaining; another will have only a radius of visibility surrounding Santa, a timer, and presents remaining; and the last will just be Santa exploring the maze on his own)

Algorithmic Plan

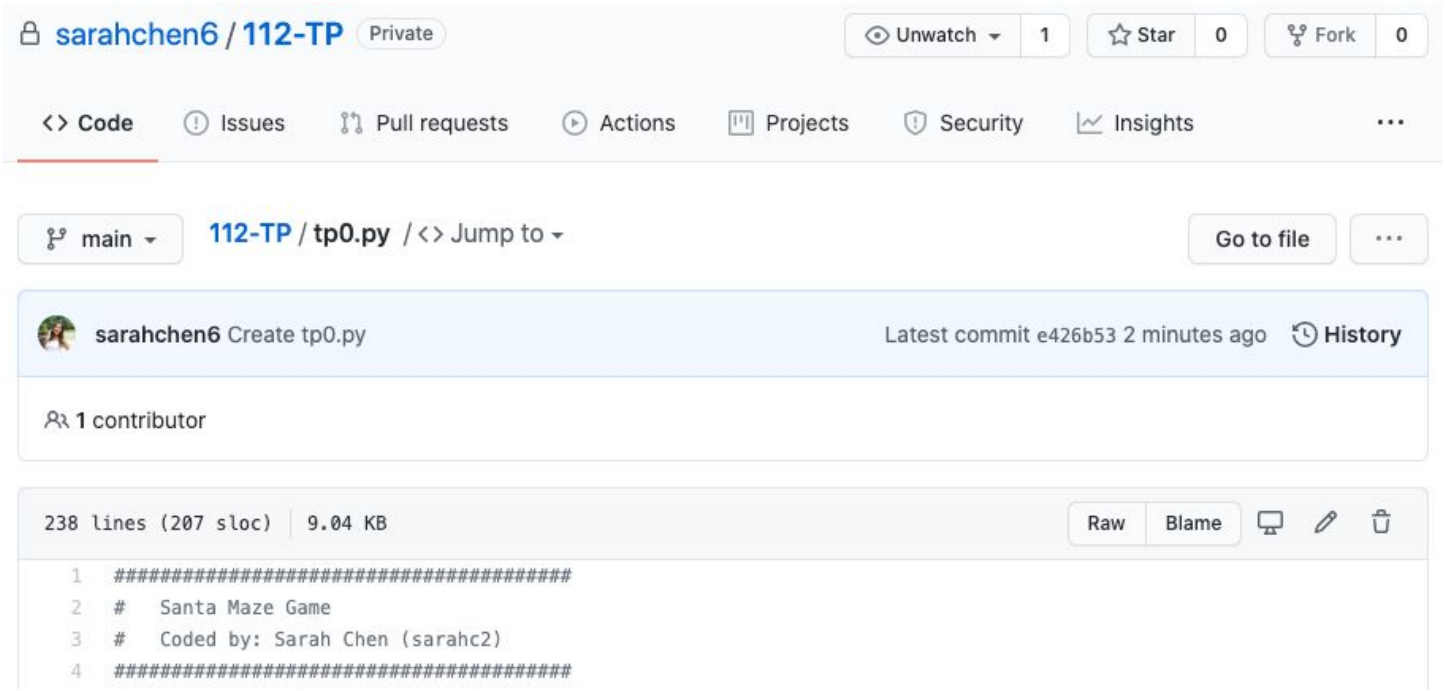
The trickiest part of my project will most likely be creating and solving randomly generated mazes. In order to do this, I will use dictionaries and recursive backtracking along with Graph Theory principles (e.g. nodes and edges). I plan on having the dictionary map each coordinate point to a randomly selected neighbor that is not already a key in the dictionary, i.e. it is not connected to another point already. If all neighboring points have connections, I will use recursive backtracking to return to the most recent node with a free neighbor and continue the process until all nodes / coordinates are connected. Hence, I must keep track of the order of the visited coordinate points (backtracking) and which coordinates map to which (dictionary that eventually will lead to drawing functions). I expect solving for a solution to be along the same lines of this recursive backtracking concept. Another difficult aspect of my project that I foresee is the implementation of different game modes. I expect to accomplish this through OOP and the use of classes / superclasses. Coupled alongside my maze-generating algorithms, I think that implementing these elements into a visually-appealing user experience will be a major feat.

Timeline Plan

Date	11/28	11/30 (TP1)	12/01	12/03	12/05 (TP2)	12/07	12/09 (TP3)
Finished Features	Random Maze Generation	AI Maze Solver & Maze Display	1-2 out of 3 sled modes	3 out of 3 sled modes	Story, Santa Customization, Updated Design Doc	Ensure everything is visually appealing & features actually work	ReadMe File & Project Demo

Version Control Plan

I am using a GitHub repository to back up my code.



Module List

I am not planning on using any additional modules/hardware/technologies.