“In the Beginning”
Design Proposal

Project Proposal

Project Description

“In the beginning” is a 3D game where users can generate an initial 3D land terrain which serves as a canvas for their world to come. In the beginning, the user will have starting items, a lake located in the valleys of the 3D terrain they created, infinite rocks, and infinite shrubs. By strategically placing the rocks and plants, the user can discover new world elements like trees and fire to build their way to more complex elements. And in the end, they will have a whole world that started from a blank canvas and a body of water!

Competitive Analysis

Minecraft, Little Alchemy, and Bakery Story (produced by Storm8 studios) are existing projects very similar to what “In the Beginning” strives to be.

(1) Minecraft relies on taking basic building materials and combining them to make new ones. Users can explore a world from the first person point of view. The game can be a therapeutic session to design fun houses or it can be a stressful speed run to beat the final boss as fast as possible.

(2) Little Alchemy is like Minecraft, but the only user task is to craft by dragging items onto a 2D white board. Of the other two, Little Alchemy is the most straightforward, putting things next to each other and getting new things. The difficulty of the game lies in what each recipe consists of.

(3) Unlike Little Alchemy, Bakery Story takes place on a 2.5D map, a diamond shaped room with an angle so that the user is looking down at the world. Players buy decorations and place various stoves to make unique pastries.

An example of a world in Bakery Story

Here are 5 aspects of my game that I wish to optimize: Interactive Element Generation, Buildable Crafting, User POV, Gaming Adrenaline, and Reward Rate.

(1) Buildable Crafting (BC) is the idea that one item or 2+ items can lead to further discoveries. Having Buildable Crafting gives the user purpose and a goal to reach which is to build them all.

- Minecraft and Little Alchemy do this perfectly.
- Bakery Story has a little sense of buildability has certain parts are required to build stoves, but the acquiring of the parts take far too long and is not central to the game.

(2) Interactive Elements will be defined as the level of interaction that the user has with various objects in the game. Without it, the user really has nothing to do or anything “tangible” to handle thus making the game boring. This is different from Buildable Crafting because a game can have BC but the items do not have very many applications.

- Minecraft does this perfectly as there are random elements that the user has a multitude of usages for. Each interaction is unique, sometimes has cool animations (like killing the final boss), and the feeling it gives the user is one of wonder and excitement for what else exists.
- Little Alchemy has the element of adding things together to get another object, which is exactly what I strive to recreate. However, I hope to add onto the experience by having special interactions. Like when diamonds are discovered, there will be a star shower across the sky.
- Bakery Story only allows the user to buy and place decorations and bake goods. The process is very similar and the interactions become boring too quickly. To avoid this, I need to make the interactions varied.

(3) Gaming Adrenaline is the vague term for the amount of stress that a user feels when playing it. Some users enjoy the amount of stress while others enjoy a more smoothing experience. I hope to evoke feelings of the latter option.

- Minecraft can vary from stressful to calming. But for my game I hope to only have calm moments, no chaotic fights with swarms of monsters or stress from losing one's progress.
- Little Alchemy and Bakery Story have little adrenaline.

(4) User Control (UC) is the ability for the user to determine how their game experience will be. For instance, Minecraft has user control in the sense that players can make their own houses, choose where to go, what to farm, etc. However it lacks UC in the sense that finding diamonds or other items take painstakingly long.

- Minecraft has user control in the sense that players can choose basically whatever they want to do. However the main lack of control that is some rare items are far too rare to acquire making the user feel impatient.
- Little Alchemy is very difficult for a first time player to figure out each recipe as there are no hints, leaving the user in the dust
- Bakery Story gets full points because the user can always expect what is to happen and controls what their shop looks like.

(5) Reward Rate is the pace at which the game progresses and how option the user gets prizes. Having the right tempo is important to keep the user occupied to keep playing while not giving them too much to the point where they become bored.

- Minecraft is too addictive to the point where users can't stop playing for long periods of time until they experience Minecraft burnout. But being able to
- Little Alchemy reward depends on how fast the user can figure out the recipes, but the reward is not stimulating enough to keep someone hooked for long periods of time.
- Bakery Story gives out many but low value rewards towards the beginning of gameplay but then gives few but big rewards at the end. This progression makes it so that seasoned players are either really into the game, or casual players lose interest and drop the game.

	Interactive Elements	Buildable Crafting	Gaming Adrenaline	User Control	Reward Rate
Minecraft	10	10	6	7	8
Little Alchemy	7	10	8	4	4
Bakery Story	5	5	9	10	2

What my game should do:

1) Interactive elements:
    a) mirror the placement of those things near each other like Minecraft and Little Alchemy but on a 2.5 platform.
    b) Add onto Little Alchemy by having special events trigger when a recipe is crafted
2) Buildable Crafting:
    a) Have a good amount 7+ elements to discover/interact with
3) Gaming Adrenaline
    a) There will be no monsters or loss of any sort happening in my game, we are looking to minimize the jump scares
4) User Control
    a) I still want to have some randomness like what animals spawn, but nothing should be too spontaneous like finding diamonds in Minecraft which has a 0.0846% chance of happening.
5) Reward Rate
    a) For the safe of the demo I will speed things up, but altogether the reward time delays should give consistent and a variety rewards without making the user bored of making too much progress or having too little

Structural Plan

There will be 3 guaranteed files: UpdateCells (currently named grayScott, runs and redraws the game), ProjectionOperations (contains functions to convert between 3D to 2D and other perception handling), and WorldElements (contains all the classes for all the possible world elements). If time allows it, I will add WeatherTimeCoordinator (controls the settings of the in game world). In case other features beyond my MVP are not in the scope of these files, I will most likely create a different file as well.

UpdateCells handles the main progression of the game. It loads the starting screen, calls the map generating functions, and necessary helper functions from the other helper files. There are two phases, the 3D map generation and the element placement/discovery phase. Those are separated by `app.drawLine`, inside the model.

ProjectionOperations contains code that helps with all perception related challenges and functions relating to matrix algebra. Some functions like `matrixMultiplication()`, `centerOf4Coords()`, and `magnitudeOfVector(v)` perform the simple operations. But `pointTransformer(point)` is a perception related function that takes in a 3D point in (X,Y,Z) form and converts it to a 2D point using matrix operations and planar homography/image geometry.

World Elements will have every possible world element because they are so intertwined with each other, having one file would be the most compact. Each world element has a `self.coords` attribute which contains the four points that contain it. The class methods will include `checkSurroundings()` and `checkTime()`. `checkSurroundings()` will compare the current object to nearby ones to see there are any it can craft itself with . `checkTime()` keeps track of time dependent items like Flowers, Fruit, Salmon, and more which only after if it's precursor has existed for a certain amount of time.

Algorithmic Plan

There are 2 complex parts to my project:

1. Representing 3D data in a 2D fashion, while making it look 3D without MatPlotLib
2. Gray Scott Reaction and Diffusion without matplotlib or numpy

1: Representing 3D data in a 2D fashion, while making it look 3D without MatPlotLib

How I approached it:

1) Went to OH with Professor Kosbie, who told me to look up projections and camera perspective (30 minutes)
2) Watched lectures, read PowerPoints and textbooks about Planar Homography (8 hours across 2 days)
3) Set up my initial matrices for transforming a single data point
4) Try using that matrix for a set of points and continued to adjust the matrices until I achieved my desired output (2 hours)

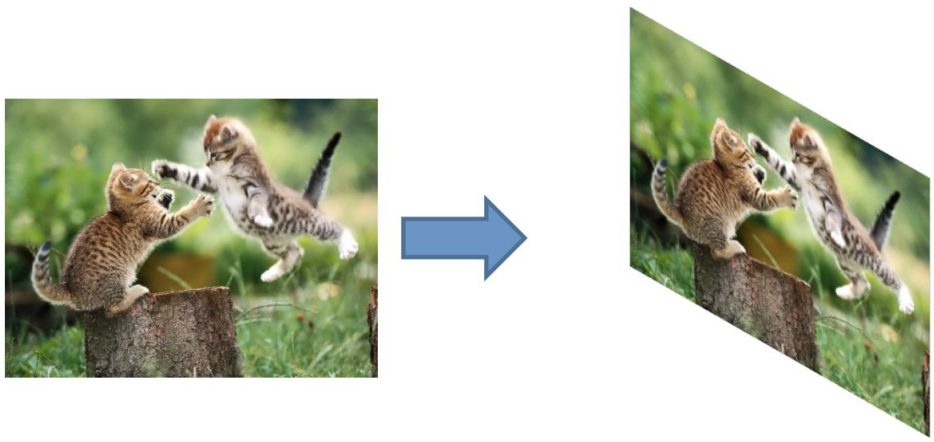
My final algorithm works like so:

Input: a list containing [X, Y, Z, 1]

```
def pointTransformer(point):
    #set k1 matrix
    sin = math.cos(math.pi*5/6)
    cos = math.sin(math.pi*5/6)
    k1 = [[cos, -1 * sin , 50], \
         [sin,  cos      , 50], \
         [0  ,  0      , 1] ]

    #set k2 matrix
    k2 = [[17,5,0,0], \
         [5 ,17, 0,0], \
         [0, 0,1, 0] ]

    k = matrixMultiplication(k1, k2)

    newPoint = matrixMultiplication(k, point)
    #x translation
    newPoint[0] = newPoint[0][0] + 70
    #y translation
    newPoint[1] = newPoint[1][0] + 350
    return newPoint
```

The k1 matrix rotates my board by 5pi/6 radians, and the column containing (50, 50, 1) dictates the “camera position”/focal point of the viewer.

The k2 matrix scales my data by 17 otherwise my graph would be very small since the point (x,y) would be just the row and col+some number. The 5 shears my image in a fashion like so (but in a different direction):



I keep the two separate just in case I want to tweak a certain transformation, ultimately I multiply the two together to make the final k matrix. Then I apply the 3D to 2D transformation to the individual point, and translate it by 70, 380 so that the point is in frame.

- Second Tricky Part: Gray Scott Reaction and Diffusion without matplotlib or numpy

Since I could not use modules, I made python do all the arithmetic by scratch.

$$A' = A + (D_A \nabla^2 A - AB^2 + f(1-A))\Delta t$$
$$B' = B + (D_B \nabla^2 B + AB^2 - (k+f)B)\Delta t$$

I simulated a Gray Scott Reaction and Diffusion system. The first part marked with “1” initializes two boards to change the concentration of particles A and B.

In part two I simulate diffusion which is the $D_A \nabla^2 A$ and $D_B \nabla^2 B$. For each cell it diffuses out 0.2 of its current concentration to adjacent neighbors and 0.05 to diagonal neighbors. First I go through each cell, then go through each possible direction of that cell, check if the cell exists on the board. Then if the direction is adjacent, diffuse 0.2 and 0.5 if the direction is diagonal.

In part three I apply the diffusion changes to each cell, as well as the rest of the equation.

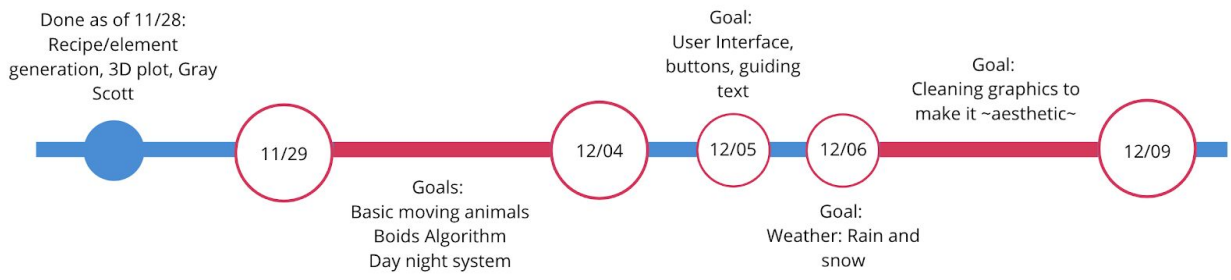
- AB^2 signifies the reaction that A+B produces 3B particles. So I subtract that amount from A, and add it to B
- f(1-A) is like a simple system to make sure the concentration of A never exceeds 1, otherwise I feed (hence f to represent the feeding constant) a negative amount of A to lower its concentration
- -(k + f) is the kill rate, but in other sources they just use k*B so I went with that instead
- Delta T is the amount of time that passes between each step.

I also have a min and max implement to make sure that the values of B and A stay in range of (1,1].

```
def grayScottRD(app):
1   deltaBoardA = make2dList(app.rows, app.cols)
    deltaBoardB = make2dList(app.rows, app.cols)
    directions = [(-1, -1), … , (1, 1) ]

2   #DIFFUSION
    #construct delta matrices
    for row in range(app.rows):
        for col in range(app.cols):
            for direction in directions:
                if neighborExists(app, direction, row, col):
                    # adjacent
                    if direction in [(-1, 0), (0, -1), (0, 1), (1, 0)]:
                        deltaA = app.dA*app.boardA[row][col] #dA = 0.2*0.2
                        deltaB = app.dB*app.boardB[row][col]
                    # diagonal
                    if direction in [(-1, -1), (-1, 1), (1, -1), (1, 1)]:
```

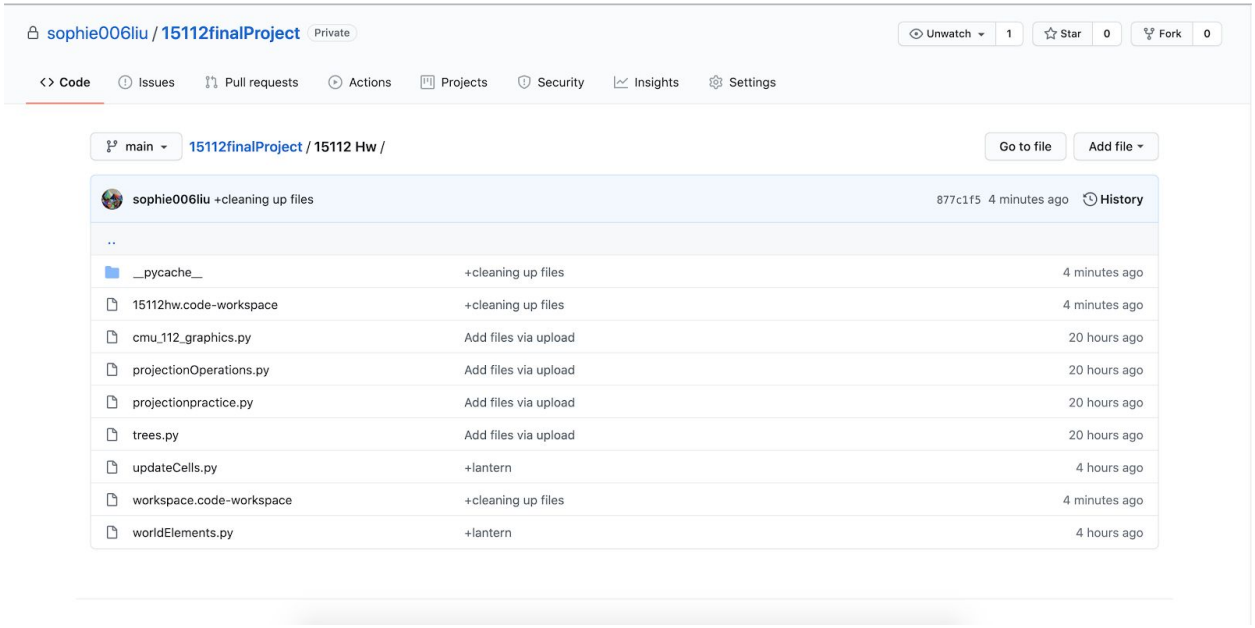
```
                        deltaA = app.dA_diagonal*app.boardA[row][col]
                        deltaB = app.dB_diagonal*app.boardB[row][col]
                    # which cell receives a change of delta
                    deltaRow, deltaCol = row+direction[0], col+direction[1]
                    deltaBoardA[deltaRow][deltaCol] += deltaA
                    deltaBoardB[deltaRow][deltaCol] += deltaB
                    # subtract from the center
            deltaBoardA[row][col] -= app.boardA[row][col]
            deltaBoardB[row][col] -= app.boardB[row][col]

3   #apply deltaA to the current boardA
    DA = 1.0
    DB = 0.5
    DT = 1.0
    epsilon = 0.0000000000001
    for row in range(app.rows):
        for col in range(app.cols):
            A = app.boardA[row][col]
            B = app.boardB[row][col]
            app.boardA[row][col] = min(1.0, max(epsilon, A +
(DA*deltaBoardA[row][col] - A*B*B + app.f*(1.0-A)) * DT))
            app.boardB[row][col] = min(1.0, max(epsilon, B +
(DB*deltaBoardB[row][col] + A*B*B - (app.k*B)) * DT))
```

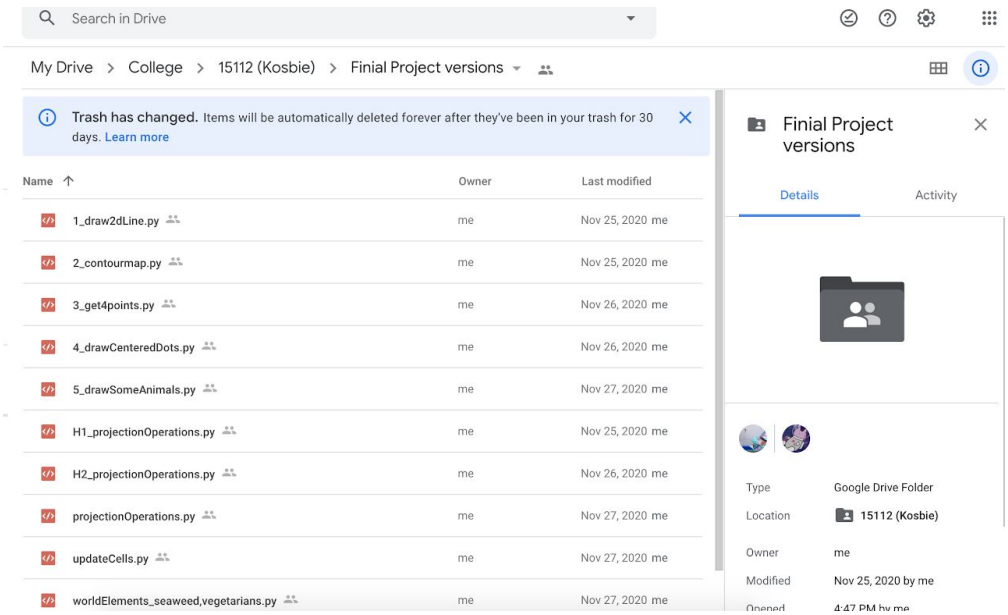
Timeline



Version Control Plan

I am formally using GitHub for the first time. In case things go wrong I have some manually saved versions in a Google Drive Folder.

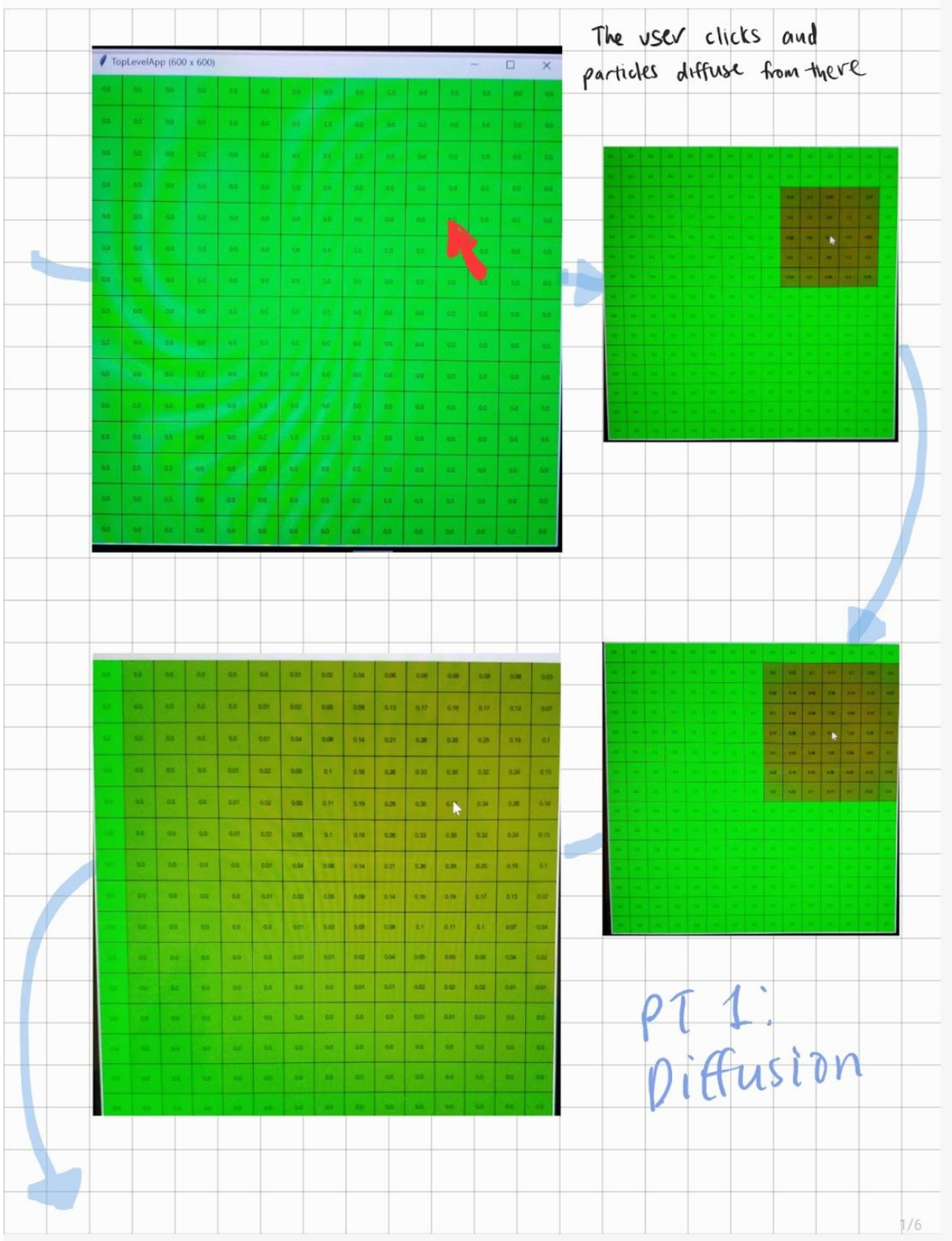


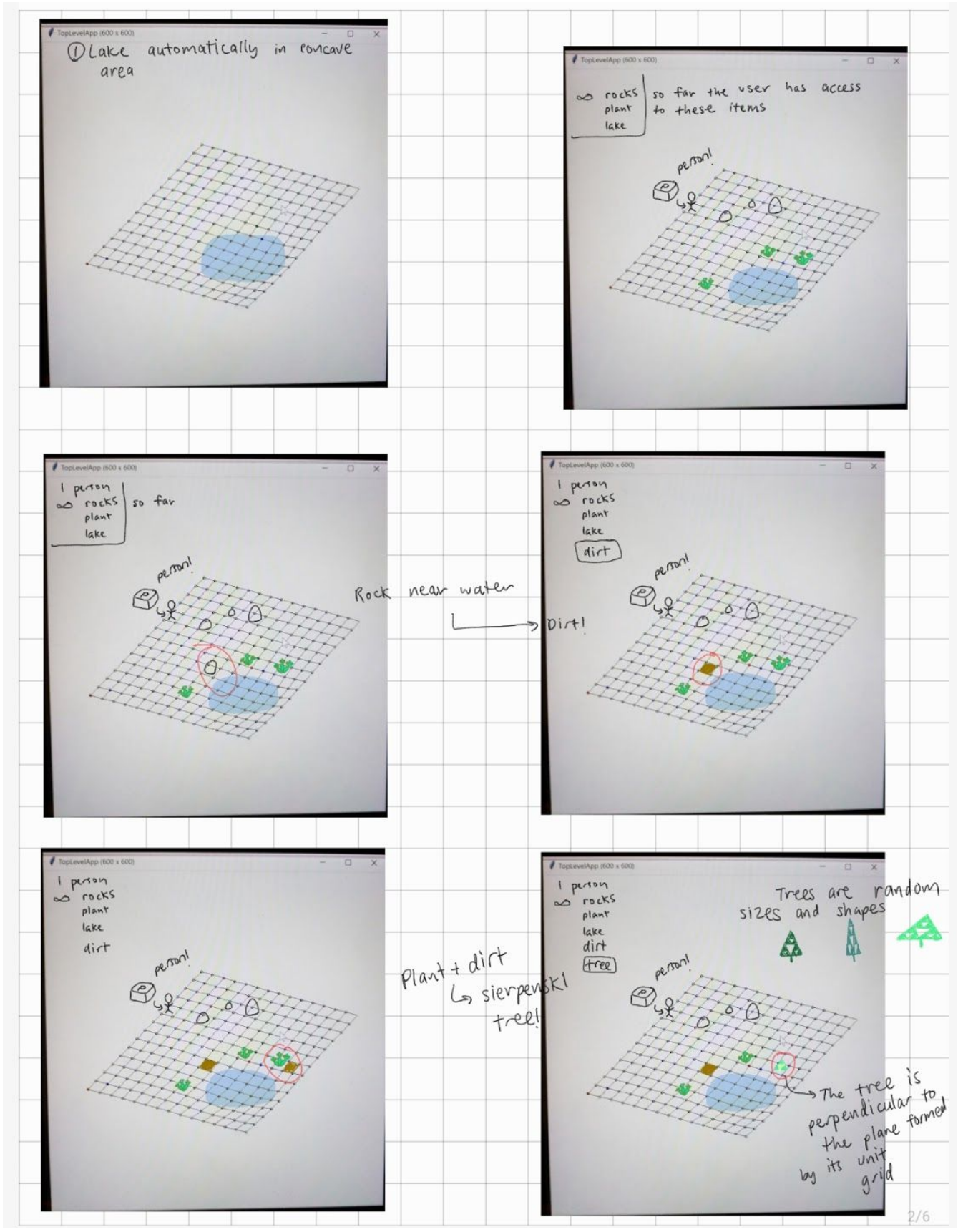


Module List

No modules here!

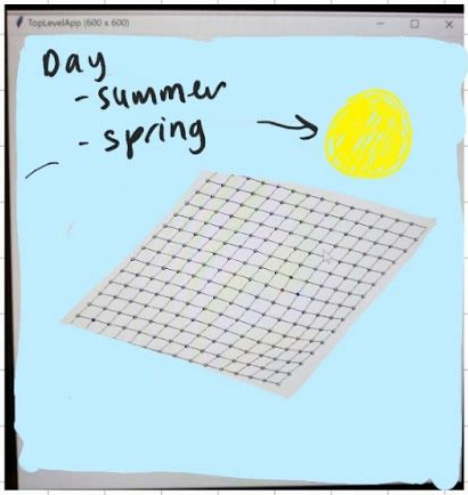
StoryBoard


TopLevelApp (600 x 600)
The user clicks and particles diffuse from there
PT 1: Diffusion
1/6



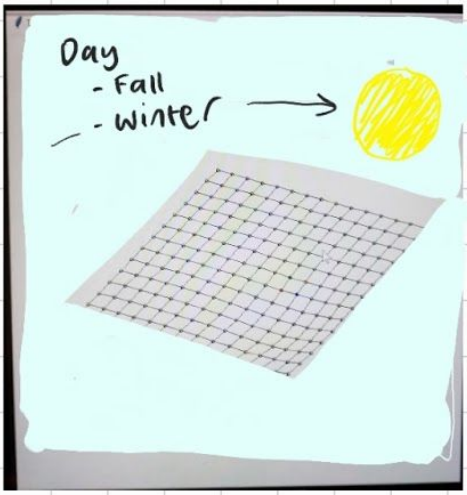
Note: There are more recipes in the game, these are just a few

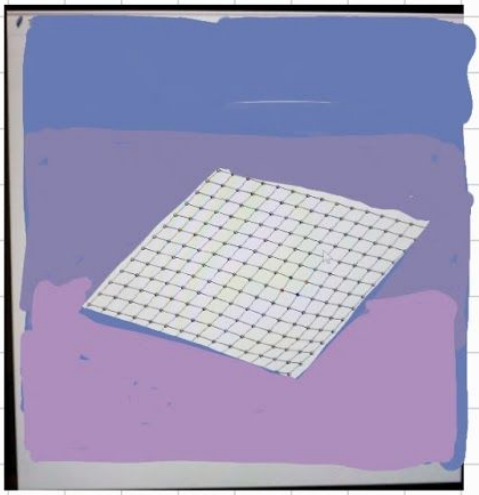
Day/Night in various seasons



* The animated sun will moved across the sky

Night is the same for all seasons





TP2 Update

No major changes have been made, but for some points on clarification are:

- I changed the Gray Scott colors to match the overall color scheme of the game, but I think that the colors are a little less smooth. To circumvent that let's just say when the user clicks on an area in the board that area will flash those colors and proceed to diffuse.

- People will not be spawned. If they are, it will be low priority. I think I am going to focus more on the graphics quality and trying to make the game look more put together.

TP3 Update

- Something went wrong in my code., so I had to use an old version. As of right now, the color scheme in TP2 is gone. As well as the directions.

- The new animal "Dog" can now use the A* algorithm to path find its way to a rabbit. If too many rabbits spawned at once, the dogs will compete for the same rabbit.

- The user can expand their map into the screen, and drag their mouse outside of the board to shift their perspective, scroll along the board.