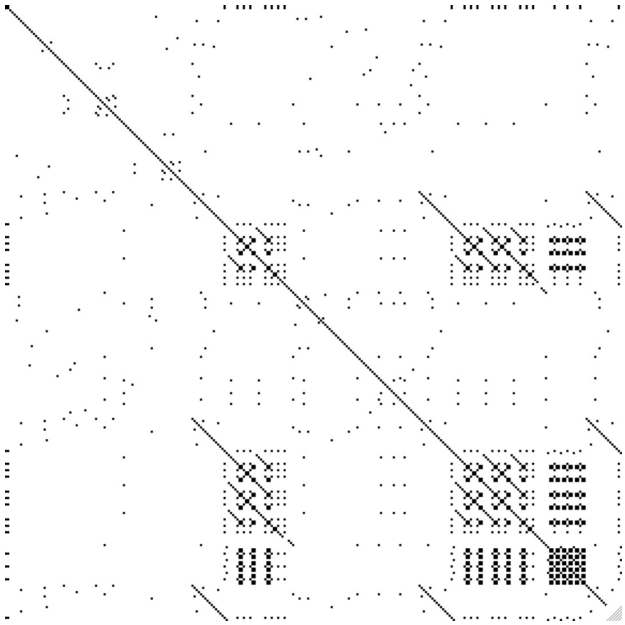
Repetition in Lyrics

Goals

The purpose of this assignment is to learn to

1. Access lyrics data through BRIDGES.
2. Manipulate a `ColorGrid` object.
3. Show repetition patterns in songs.

You will generate a visualization that looks like that!



Programming part

Task

In this assignment, the objective is to pull a song from Bridges, split the lyrics into individual words, and compare each word against every other word to check for repetition.

From these lyrics, you will be building a matrix, or a `ColorGrid` in this case, where every row and every column represents a sequential word in the song's lyrics.

Upon finding repetition, you will be setting the pixel at that location to a color of your choice at that point in the grid.

Basic

1. Open your scaffolded code.
2. Plug in your credentials.
3. Complete the TODO's.
4. Run and visualize the code.

Build a ColorGrid

1. Plug in your credentials.
2. Think of any song which contains words.
3. Query `Bridges` for said `Song`. For example

- in Java

```
Song mySong = Bridges.getSong("My Favorite Song", "Optional Artist String");
String lyrics = mySong.getLyrics();
```

- in C++

```
Song mySong = DataSource::getSong("My Favorite Song", "Optional Artist String");
auto lyrics = mySong.getLyrics();
```

- in Python

```
so = get_song("My Favorite Song", "Optional Artist String")
song = so.get_lyrics()
```

4. Pass these lyrics through the provided helper function, which will clean up and split the lyrics into an array of squeaky clean `String`s.
5. Initialize a `ColorGrid` with the dimensions the size of the array returned from the helper function.
6. Iterate over the split lyrics array, checking to see if there is any repetition. For example, if word 1 is the same as the word 6, you would color the pixel at (1, 6), and later on at (6, 1).
7. After filling out your grid, set it as the data structure on your Bridges object, and run the code.

Help

for Java

ColorGrid documentation

Color documentation

DataSource documentation

Song documentation

Bridges class documentation

for C++

ColorGrid documentation

Color documentation

Song documentation

DataSource documentation

For Python

Bridges documentation

Color documentation

ColorGrid documentation

Song documentation

BRIDGES Game API Tutorial

Goals

To teach students the functions that allow them to interact with their games.

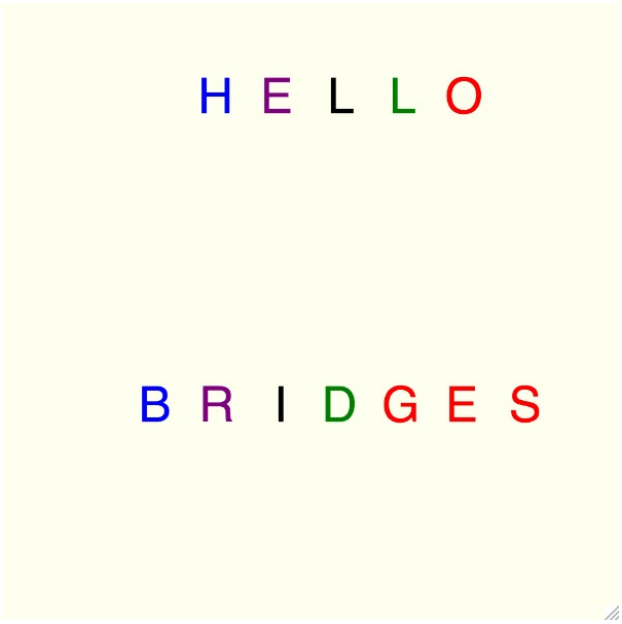
1. Understand the main variables used to run a bridges game.
2. Learn about the initialize() method.
3. Understand ways students can implement their own methods and how to traverse the grid using for loops.
4. Learn about the gameLoop() method and the 10 different controls available in non-Blocking games.
5. Prepare the main method and run a game.

ToDo

1. Paint the screen a single color and display a set message
2. Upon a key press, display a different message

Check comments in the scaffold for details.

Sample Output



Variables

Sets the size of the students grid. Grids are limited to 1024 cells. The largest square grid available is 32 x 32.

- int gridColumns
- int gridRows

Variables that will change the color and symbols despalayed in the game:

- NamedColor myBackgroundColor
- NamedColor mySymbolColor
- NamedSymbol mySymbol

Functions

initialize():

- A required bridges function. This runs before the game loop begins allowing the students an oppertunity to set the colors and symbols of their game before it starts.

gameLoop():

- A required bridges function. A recursive method that will run until the game is over or is disconnected. Students can call their own methods here to make their games more complex.

Boolean values that are true when a specific key is being pressed:

- keyUp()
- keyDown()
- keyLeft()
- keyRight()
- keyW()
- keyS()
- keyA()
- keyD()
- keyQ()
- keySpace()

Documentation

You can check the generic Game Tutorial

Java

NonBlockingGame

NamedColor

NamedSymbol

C++

NonBlockingGame

NamedColor

NamedSymbol

Python

NonBlockingGame

NamedColor

NamedSymbol

Bugstomp Game

Goals

The purpose of this assignment is to learn to

1. Move a character sprite around a 2D grid to step on randomly generated bug sprites.
2. Use loops, 2D arrays, and conditional statements to move the character and check if he has stepped on a bug sprite.

Programming

Tasks

- Initialize a 2D array the size of your game grid.
- Initialize the NamedColors of your character, bugs, and background.
- Determine the starting point of your character on the grid.
- Create a function that uses keypress events to move the character around.
- Create a function that will generate bugs in a random place on your grid.
- Create a function that will check if the character has collided with a bug.
- Create a function that removes bugs after some amount of time.
- Create a win condition.

More Details to Get You Started

Key Press Events - NonBlocking Games

- keyUp()
- keyDown()
- keyLeft()
- keyRight()
- keyQ()
- keySpace()
- keyW()
- keyS()
- keyA()
- keyD()

Variables, Colors, and Sprite Symbols

- NamedSymbol.symbolname;
- NamedColor.colorname;
- drawSymbol(column, row, NamedSymbol, NamedColor);
- setBGColor(column, row, NamedColor);

Important Functions

- The gameLoop() function loops until the game is over.
- quit() stops the game.
- start() starts the game and calls the initialize() function once before it starts the gameLoop() function.
- render() sends your updated game grid to the server once. This is a blocking game function.

Help

For Java

NonBlockingGame documentation

For C++

NonBlockingGame documentation

For Python

NonBlockingGame documentation

Assignment 23: Patterns

Goals

1. Learn how to modify and display a ColorGrid to Bridges
2. Practice using simple loops to display patters to a ColorGrid

Programming

Setup

- Download the Bridges library from the Bridges website http://bridgesuncc.github.io/
- Follow the Getting Started tutorial for the IDE you are using http://bridgesuncc.github.io/bridges_setup.html
- Enter the assignment number, username, and api key values into the main method of the class file given to you for this assignment.
- Run your code and follow the link in the output to view your grid.
- Make changes as needed to adjust how your grid looks.

Assignement

- Use Bridges ColorGrid to display simple pattern
- Call ColorGrid.set to set the color of a pixel in the grid
- Loop through the grid and set each pixel to match a pattern

Examples

- Checker board, display 2 colors in an alternating checker board pattern
- Frame, display a frame around the entire board with the center filled
- Quadrants, Cut the image in four quadrants of different colors
- Or make a customized pattern of your own

Help

for C++

ColorGrid documentation

Color documentation

Bridges documentation

for Java

ColorGrid documentation

Color documentation

Bridges class documentation

for Python

ColorGrid documentation

Color documentation

Bridges documentation

Mountain Paths - Determining a path of low elevation through a mountain

Goals

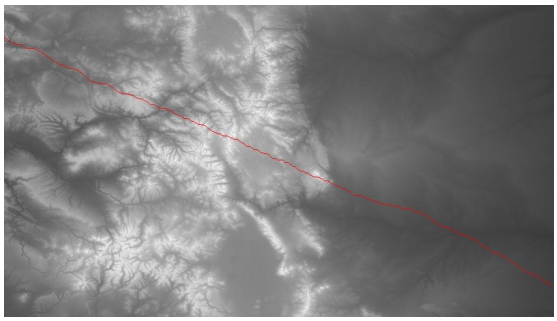
1. Working with 2D images of elevation maps
2. Understanding Greedy algorithms and its application to a real-world application
3. CS concepts: 2D array addressing, greedy algorithms

Source

This assignment is adapted from a **Nifty assignment from 2016 proposed by Baker Franke. See **Source

Description

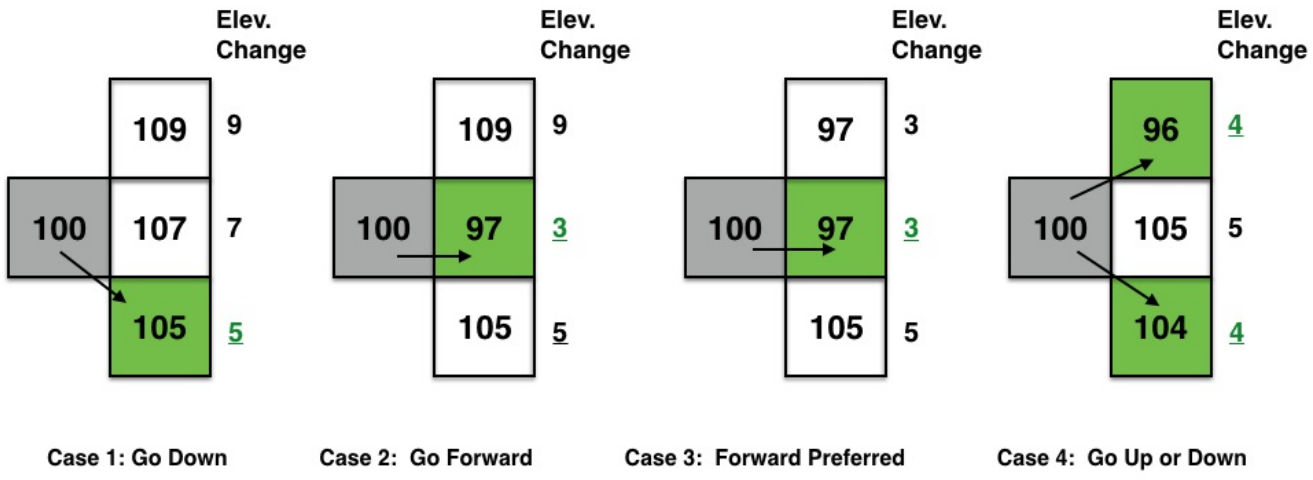
You are given elevation data of a mountainous region in the form of a 2D array of integers (see example image below of one of the datasets). Your goal is to find a path that takes you through the points with the lowest elevation changes, in an effort to minimize the overall effort in walking through the path. For this you will use a `greedy' approach and make local decisions in determining the successive points in the path.



Input gray scale image of a terrain with gray shades mapped to elevation. Elevation ranges from low (dark shades) to high (lighter shades)

The image above shows a mountainous region; lighter regions are higher elevation, and the red line shows the path taken by a walker.

Algorithm To Determine the Path:



The figure above (reproduced from **Nifty) illustrates 4 different cases for making a decision in determining the next point in the path.

You will use a greedy strategy to determine successive points along the path. Figure above shows how the algorithm

applies the greedy strategy to a pixel (with grayscale values shown). The algorithm looks to the 3 choices and picks the pixel that
causes the smallest change (least effort to walk). The idea is to start from an edge of the image (say the leftmost column), then make moves based on the pixels to the right, each time choosing the pixel that results in the smallest change. Your goal is to reach the right edge of the image.

Tasks

1. *get dataset, visualize as an image* You will use BRIDGES to define a lat/long range (rectangular bounding box of your choice) to access the elevation data (returned in an object containing the elevation values). You will need to calculate maximum elevation value, as you will need to scale the values to the (0-255) range to display the image in a ColorGrid. (Backup plan is to read from a PPM image, if there are any issues accessing data from an external source)

2. *Display the image using BRIDGES*. Once you have acquired the dataset, scale the values to 0-255 range and convert that to an integer (divide each value by the largest and scale to 255). Use the BRIDGES ColorGrid class to hold the image. This can then be visualized directly in BRIDGES. Check the tutorial for the ColorGrid object that illustrates the calls needed for visualization. The ColorGrid class has the methods to load colors for each pixel and can take separate R,G, B and Alpha components. For grayscale images (like the one above), R=G=B.

3. *Compute the Lowest Elevation Path*. Implement the greedy algorithm on the image.
   Choose a pixel in the left most column, somewhere in the middle region. Your program will determine the points in the path that exhibit the smallest change (see figure above) in elevation and draw this path in a distinct color (like red). Pixels in the path will have their values changed to this color (for instance, use (255, 0, 0) for red.
   As you compute these low elevation points, modify your color grid to draw the red pixels.
   You need to keep track of the pixel addresses and the image height and width, so that you dont go past the boundaries of the dataset (grid). Note that if you are on the boundaries of the grid, your choices will be reduced.

4. *Display the image.* Again, use BRIDGES to display your final image with the chosen path (see example above). You can have a simple User Interface to specify the starting point and rerun your program to display different paths.

Variants

One can make variants of this assignment. Indeed, the greedy algorithm presented above is a heuristic; it does not return the path that sees the lowest change of elevation across the entire mountain. It only makes a local choice. Here are some possibilities:

1. One can find the path that always goes right and that minimizes the total change of elevation using Dynamic Programming. Propose a Dynamic Programming algorithm, and implement it to highlight the right-going path of minimal total change of elevation.

2. If one does not always go right, then the problem is akin to a Shortest Path problem. Adapt Dijkstra's algorithm and implement it to highlight the path of least change of elevation.

3. Graduate students can consider the problem of optimizing simultaneously the distance traversed and the total change of elevation as a bi objective optimization problem.

Additional Help:

BRIDGES Team: Contact the BRIDGES team for any issues with the BRIDGES API. This is an active project.

for Java

DataSource

ColorGrid documentation

Color documentation

Bridges class documentation

for C++

DataSource

ColorGrid documentation

Color documentation

Bridges Class documentation

for Python

ColorGrid documentation

Color documentation

Bridges documentation

DataSource