Omar Cruz Pantoja
801-14-1672
Prof. Remi Megret
CCOM-4995 Computer Graphics

Build a Block

Game description

Developed a 2D puzzle game using javascript and canvas of HTML. The purpose of the game is to fill in a silhouette using primitive geometries with colors. The user will start off in a main menu screen which the user will decide whether playing the game or having a quick tutorial in a how-to-play section.

I. how to play screen- will contain an animation representing what the user will be doing (click and dragging colored pieces and setting them on the silhouette). The animation will be composed of three rectangles which will be animated to complete the shown silhouette. This screen will also contain the purpose of the game as well as the buttons given in each puzzle(called level in game).

Ii. Play screen- the user will be prompted to a screen with a selection of levels, for now, from level 1 to level 3. Each level will contain the same figures (colored pieces) but a different silhouette. Once the user selects a level, a black colored silhouette will be displayed with 7 colored geometrical figures. In order for the user to complete the level, he must cover up most of the black pixels in the silhouette. The user will have 5 triangles, a rectangle and a parallelogram (tilted rectangle) which will click and drag to relocate to a new position within the canvas. In the case the user wants to rotate a figure, he must select the figure and use the rotate buttons displayed in the screen. Once 98% of the black pixels in the canvas has been covered by the colored geometries the level will be completed and prompted to a new level .

Architecture and design

Since we are using canvas API, we have two variables containing the canvas information and the context, which allows us to draw and add event listeners. We've got 3 main events: mouseup, mousedown, mousemove.

I. mouseup- Will be responsible to deal with relocating each figure when dragging to a new position

Ii. mousemove- Will be responsible to deal with drawing figures around when being dragged

Iii. mousedown- Will be responsible to make sure mouseup/mousemove parameters are set in order to move around a figure, will identify which buttons are pressed (if any), all interactions in the game have something to do with this callback

In order to store the figures we've used a similar data structure for both the silhouette and each figure. The structure is composed of instances of two class called rectangle and triangle. In each of these classes we save the vertices of each geometry which will be used to find the new vertices position's when a figure is rotated (using rotation's formula). Both of these class also have similar functions, the only difference is that rectangle function has 4 vertices, while triangles only have 3. These functions include their respective vertices, a set rotation function, fill color, dimensions, type, rotation angle and skew if needed.

All our buttons are "custom" made in the terms that hitboxes are developed by me, meaning i have control over what will be selected or not, depending on the mouse's coordinates in the canvas plane. (developed isInsideRectangle/Ellipse/Triangle) Each of these will verify that the X Y coords of the mouse are inside the closed figures. In the how-to-play screen we have a animation which will be what the user will be based off in order to play the game and uses basically the same idea as the play part, but without the use of the class rectangle, because in the animation we don't rotate the figures.

How it works

The way the game works is by reading each pixels within a range in the canvas(to reduce having to calculate unnecessary data),and count all the #000003 color pixel within this data. We've decided to use #0000003 because it allows us to bring other materials or scenes with color black and the output won't affect directly the decision making of the program. Once 98% of the #000003 pixels have been covered, the user will be prompted an alert indicating they've passed to next level. The main data structure works off an array of class Triangle and Rectangle instances, this array is the one passed to the ctx to draw the new positions of each figure(in the case something changed). As mentioned before both the silhouette and the colored pieces uses the same structure, which simplifies the use of different data structures. How do we rotate each figure? We mentioned that each geometrical figure vertices information is saved in the data structure, meaning that when we go rotate a figure, we use the mathematical function taken from the rigid matrices and relocate update the vertices information applying the formula.

- How do we drag a figure?

First a figure must be clicked (mouse down), if the mouse is still pressed after the initial mouse down, and dragged around, the figure will be dragged but it's original content won't be updated up till mouseup isn't called anymore, which means we're no longer dragging any figure but working either with its rotation or it's on an expected position.

- How do we know we clicked a figure?

By using some mathematical basic contexts we check if a point is inside any geometrical figure(in this case, ellipse, rectangle) The parallelogram is the only

different because we separate the parallelogram into two triangles to identify if a point is inside any it's triangles.