Project Specification - Derivatiles

Names: Anish Lakkapragada, Hank Hsu, Abhishek Nambiar
Period: 2

Overall Description

Our game is for users to practice their calculus before the very difficult AP Calculus BC exam. One of the most crucial parts of calculus is differentiation; through our game players will be able to practice differentiation of a function from the first to tenth order (their choice) on a list of functions. They can also practice on functions a friend (who enters them on a different computer) gives them.

Our game has a GUI that displays a function f(x), which we have simplified to only be polynomials with an added/subtracted sin(x) or cos(x). For example, this could be "3x^2 + x^2 + sin(x)." The user will choose how many orders of derivatives they want to practice on (1-10 range) and a grid will be displayed each of which containing a tile, which displays a mathematical function. The grid have the number of order rows and 5 columns, and each row represents answer choices for the given order. Each row will have 4 randomly generated answer choices, and one of the five slots wil contain the correct answer.

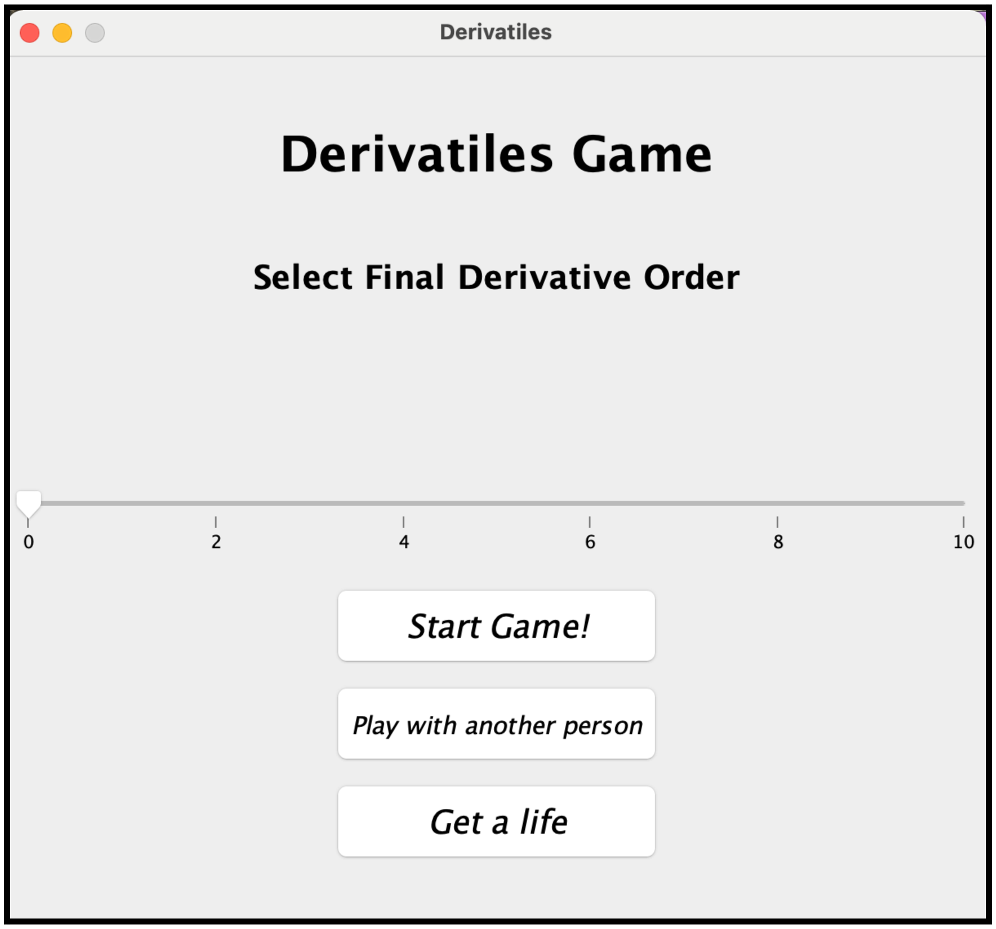
The bottom row will store mathematical expressions of the first derivative of f(x) (given in the GUI), and the second-bottom will store expressions of the second derivative of f(x), and so on. Users will start at the bottom row and move along the tiles using WASD or Arrow Keys. They will go up on a tile to choose its mathematical expression to answer the derivative of the function for the current order (or row.) They will progress up the grid and choose tiles to select the answers for higher order derivatives (i.e. users select functions for first derivative on bottom row, users select functions for second derivative on second-bottom row).

When a user is selects a function (i.e. goes up on a tile) on the top of the grid, meaning that they have answered all orders of derivatives of this function, they will go to a random tile on the bottom row to differentiate on a new function. They have the option to go back to the welcome screen at any time.

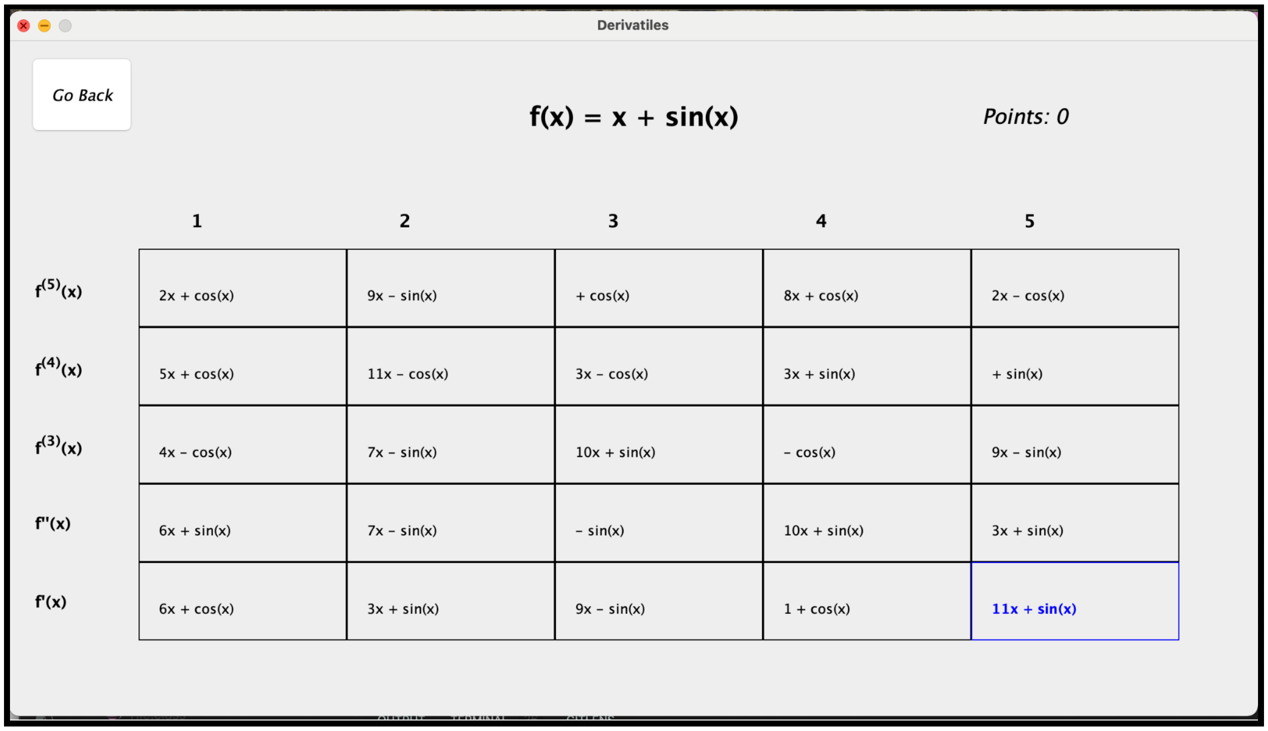
However, if the user is playing in the multiplayer mode - a user on another computer will connect to the main computer (who is playing the game) and enter in a GUI the function they want to send to the main computer. The user on the main computer will receive the functions from the other player's computer, and play on them using the same grid idea. However, they will wait to get a new function from the other player (on the other computer) when they reach the top row.

Game Demonstration

When the user first runs the game, they are met with a Welcome Screen (screenshot below.) The Welcome Screen asks the user how many derivatives orders they want to play up to (regardless of mode) and whether they want to play the vanilla game or play with someone else. The user may also choose to get a life (last button), which closes the application safely.



If the user chooses to start the game in single player mode (first button on welcome screen), they are launched into another screen. The screen displays a grid of tiles, each of which contains a function. The user is placed on a random column in the bottom row. They have to use their arrow keys (or WASD) to move along the grid. They should go up on the tile that represents the correct function for the given derivative order. For example, for the screenshot below, the user should go up on “1 + cos(x)”, “ - sin(x)”, “ - cos(x)”, “+ sin(x)”, “+ cos(x).” Note that the selected tile (currently the 4th column on the bottom row) has a blue, bold font and border.

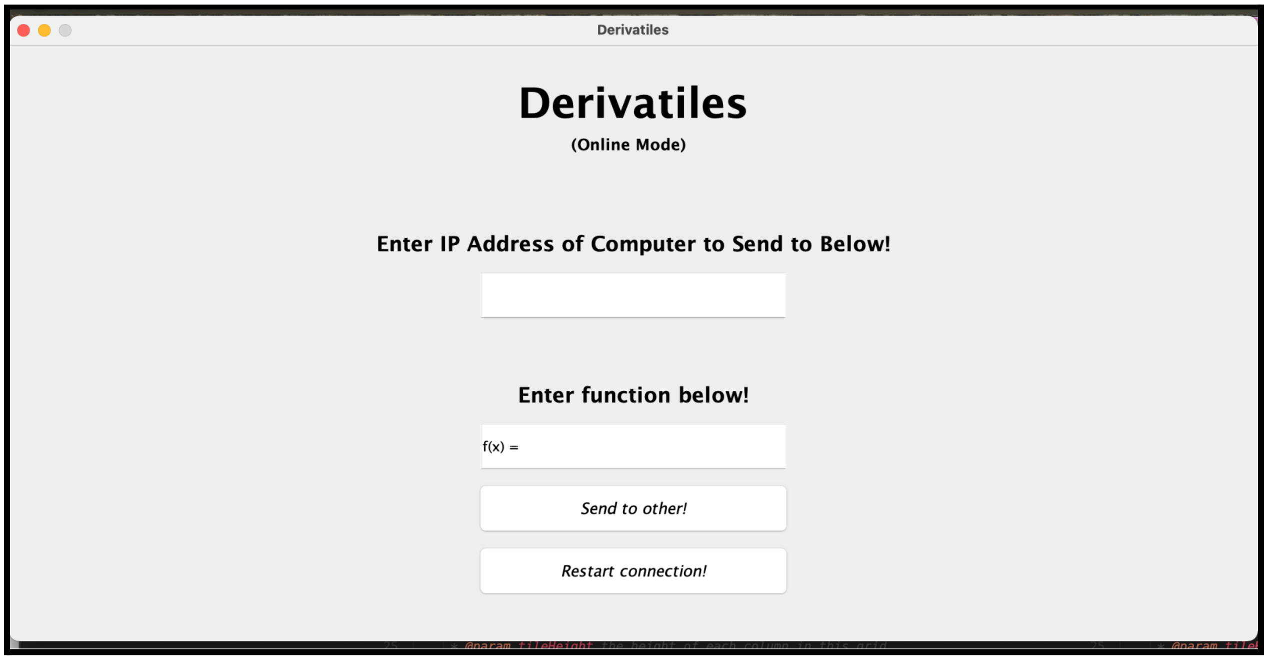


f(x) = x + sin(x)

Points: 0

	1	2	3	4	5
$f^{(5)}(x)$	2x + cos(x)	9x − sin(x)	+ cos(x)	8x + cos(x)	2x − cos(x)
$f^{(4)}(x)$	5x + cos(x)	11x − cos(x)	3x − cos(x)	3x + sin(x)	+ sin(x)
$f^{(3)}(x)$	4x − cos(x)	7x − sin(x)	10x + sin(x)	− cos(x)	9x − sin(x)
f''(x)	6x + sin(x)	7x − sin(x)	− sin(x)	10x + sin(x)	3x + sin(x)
f'(x)	6x + cos(x)	3x + sin(x)	9x − sin(x)	1 + cos(x)	**11x + sin(x)**

If the user presses the back button on the top left screen, they will go back to the welcome screen page. There they can choose to play with another person (the middle button), where they will be launched into a screen with an empty grid (no functions displayed.) In this mode, the main computer playing the game with the grid receives functions to play on by another computer. In order for the main computer to get functions to play with, there must be another computer (connected on the same network) to send functions to it. This can be done by running another executable file we wrote, the “FunctionSender” class. A screenshot of what this looks like is below.



Here the user can enter the IP address of the main computer they want to send functions to in the top text field, and in the bottom half type the function to send and send it by clicking the “Send to other!” button. If the main user decides to go back into the single player mode, the connection maybe dropped between the two computers, thus if the main computer chooses to go back into multiplayer mode the user on the computer sending functions can restart the connection by hitting the bottom button.

Classes Overview

Major Classes

Game – Serves as the main class where the game is played. Extends JFrame and functions as the screen the player plays on. Contains the main frame where the content is rendered. Implements KeyListener for user movement across the grid. Uses a *WelcomeScreen* (to display the welcome screen)*,* uses *BoardState, and TileManager* to update the game (when a question is complete or the user moves).

Differentiate – It uses symbolic differentiation (differentiate functions in String form) and is used to generate the correct derivatives in the grid. Uses a hashmap to store the derivatives of basic power functions from x^2 to x^100). Searches for key characters and strings (“x” “^”, “sin” and “cos”) to be able to recognize functions.

BoardState – It keeps track of the player score and increments or decrements if as a player gets answers right or wrong. It also generates a grid of functions (with both the right and randomly generated wrong answers. Game calls *Boardstate*’s getGrid() method to update the grid with the next problem.

GUI Classes

Tile – It draws an individual tile that is part of the grid. It is black unless it is selected by the player (then it is blue).

Tile Manager – Stores a 2D array of *Tiles* in the grid. Stores the current user location on the grid, and repaints the grid accordingly when instructed to.

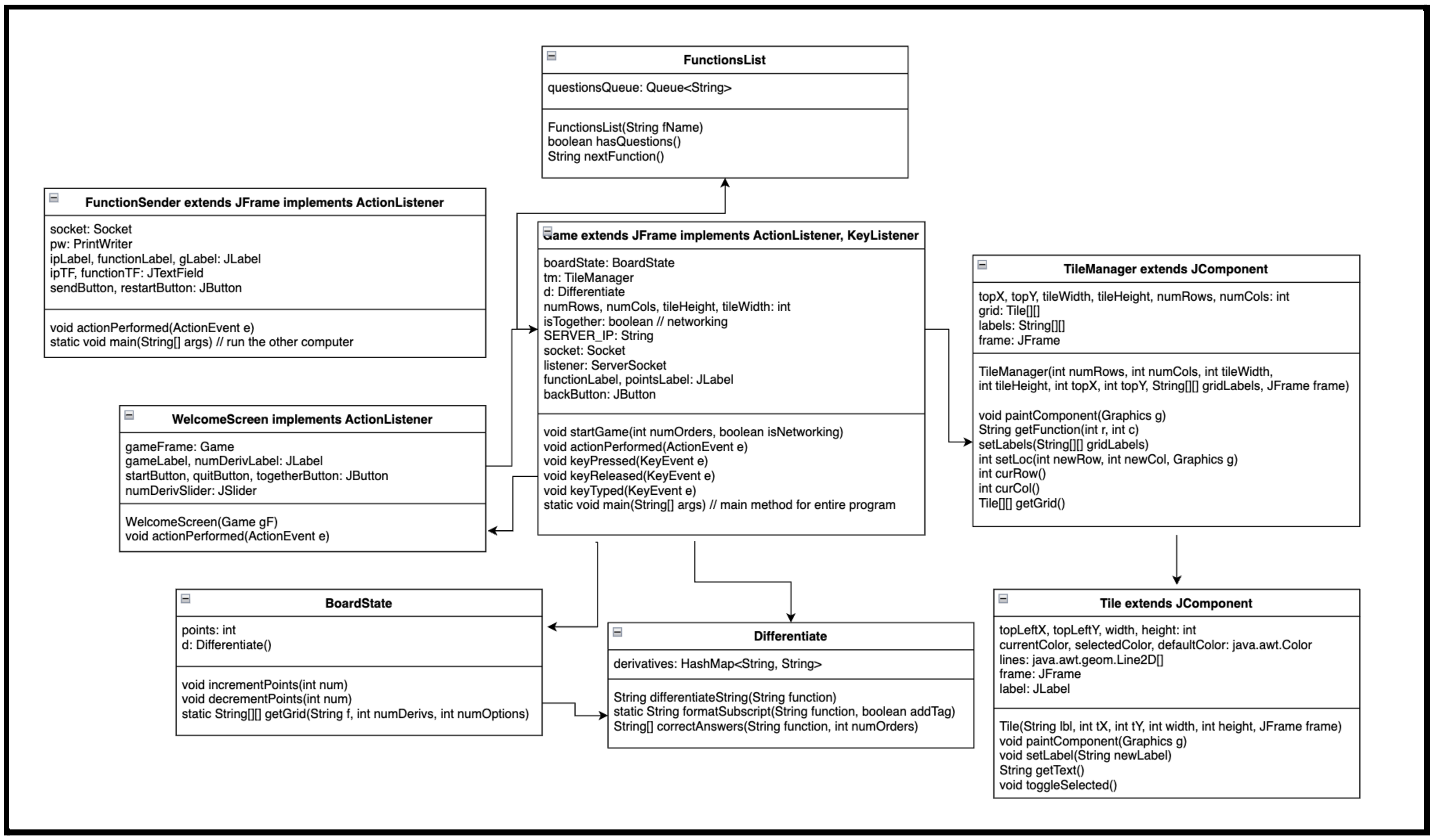
WelcomeScreen – Represents the welcome screen of the game, and uses the *Game* as a parameter in order to paint welcome screen components/objects onto the *Game* Jframe welcome screen. Offers a play button, a multiplayer button, and a quit button

FunctionList – provides a list of hardcoded functions from a Text file that is put into the grid eventually when the user chooses to play on single player mode. Provides an iterator to be able to loop through the functions.

Networking Class

FunctionSender – allows a separate user that is connected to the same network to be able to send functions (by typing them) for the other person to differentiate. When the main method of this class is run, a GUI pops up for the user to type the IP address of the computer they want to send to (must be currently on multiplayer mode) and the function they want to send.

Classes Relationship Diagram (UML)


FunctionsList
questionsQueue: Queue<String>
FunctionsList(String fName)
boolean hasQuestions()
String nextFunction()
FunctionSender extends JFrame implements ActionListener
socket: Socket
pw: PrintWriter
ipLabel, functionLabel, gLabel: JLabel
ipTF, functionTF: JTextField
sendButton, restartButton: JButton
void actionPerformed(ActionEvent e)
static void main(String[] args) // run the other computer
Game extends JFrame implements ActionListener, KeyListener
boardState: BoardState
tm: TileManager
d: Differentiate
numRows, numCols, tileHeight, tileWidth: int
isTogether: boolean // networking
SERVER_IP: String
socket: Socket
listener: ServerSocket
functionLabel, pointsLabel: JLabel
backButton: JButton
void startGame(int numOrders, boolean isNetworking)
void actionPerformed(ActionEvent e)
void keyPressed(KeyEvent e)
void keyReleased(KeyEvent e)
void keyTyped(KeyEvent e)
static void main(String[] args) // main method for entire program
TileManager extends JComponent
topX, topY, tileWidth, tileHeight, numRows, numCols: int
grid: Tile[][]
labels: String[][]
frame: JFrame
TileManager(int numRows, int numCols, int tileWidth, int tileHeight, int topX, int topY, String[][] gridLabels, JFrame frame)
void paintComponent(Graphics g)
String getFunction(int r, int c)
setLabels(String[][] gridLabels)
int setLoc(int newRow, int newCol, Graphics g)
int curRow()
int curCol()
Tile[][] getGrid()
WelcomeScreen implements ActionListener
gameFrame: Game
gameLabel, numDerivLabel: JLabel
startButton, quitButton, togetherButton: JButton
numDerivSlider: JSlider
WelcomeScreen(Game gF)
void actionPerformed(ActionEvent e)
BoardState
points: int
d: Differentiate()
void incrementPoints(int num)
void decrementPoints(int num)
static String[][] getGrid(String f, int numDerivs, int numOptions)
Differentiate
derivatives: HashMap<String, String>
String differentiateString(String function)
static String formatSubscript(String function, boolean addTag)
String[] correctAnswers(String function, int numOrders)
Tile extends JComponent
topLeftX, topLeftY, width, height: int
currentColor, selectedColor, defaultColor: java.awt.Color
lines: java.awt.geom.Line2D[]
frame: JFrame
label: JLabel
Tile(String lbl, int tX, int tY, int width, int height, JFrame frame)
void paintComponent(Graphics g)
void setLabel(String newLabel)
String getText()
void toggleSelected()

*Note: Game and Welcome Screen both “have” and use each other, thus they both have arrows directed to each other. The above diagram only shows public methods, for brevity.

Structural Design

Description	**Java Structure**	**Class Found In:**
Grid of tiles to be rendered on the GUI	Tile[numRows][numCols]	TileManager
Storing functions in String form and their derivatives (e.g. “x^3” to “3x^2”)	HashMap<String, String>	Differentiate
(Temporarily) storing the	HashSet<String>	BoardState

unique answer choices for each row		
Storing the functions in the text file to be played on	Queue<String>	FunctionList

Explanation for Data Structures:

We chose a 2D array for the grid of functions because it is the most natural way to represent a matrix of values. Our application will frequently require calling methods of an individual tile, thus we will need to access them quickly; 2D arrays can do this fast in O(1) time.

For storing functions and their derivatives, we chose a HashMap. The HashMap will store a function (e.g. polynomial, “cos(x)”, “sin(x)”) and return its derivative, such as “3x^2” for “x^3”.

A TreeMap may be more space-efficient, but since speed is a bigger priority for us, we choose to use a HashMap instead of O(1) access. We also do not need an iterator in alphabetical order for our application, so there is not too much benefit of using a TreeMap compared to a faster HashMap.

When creating the randomly-generated answer choices for each row, we needed to make sure that they all were unique in the row (no duplicates.) One efficient method to do this was with a set. We had to decide between a HashSet and a TreeSet, and because we wanted to prioritize speed and didn’t need the elements in the set stored in any order - we choose to use a HashSet, as adding is roughly O(1).

We decided to store the hardcoded functions the user would play on (in single player mode) in a Queue of Strings. This is because when we read the functions (in String form) in the Text File, the first line we read in should be the first function given to the user (first-in, first-out.) The most efficient way to deal with this sort of task, also noting that we only need to access to the next function in the front of the “line”, is a Queue, as insertion and adding in a Queue is O(1).

Major Level Classes Specifications

Game Class (extends JFrame implements ActionListener, KeyListener)

- Attributes
    - BoardState boardState
    - TileManager tm
    - Differentiate d
    - int numRows, numCols, tileHeight, tileWidth
    - boolean isTogether
    - String SERVER_IP
    - Socket socket
    - ServerSocket listener
    - JLabel functionLabel, pointsLabel
    - JButton backButton
- Methods
    - public Game(TileManager manager)
    - void startGame(int numOrders, boolean isNetworking)
    - void actionPerformed(ActionEvent e)
    - static void main(String[] args) // run the entire application
    - void keyReleased(KeyEvent e)
    - void keyPressed(KeyEvent e)
    - void keyTyped(KeyEvent e)

WelcomeScreen Class

- Attributes
    - Game gameFrame
    - JLabel gameLabel, numDerivLabel
    - JButton startButton, quitButton, togetherButton
    - JSlider numDerivSlider
- Methods
    - WelcomeScreen (Game gF)
    - void actionPerformed(ActionEvent e)

Differentiate Class

- Attributes
  - HashMap<String, String> derivatives
- Methods
  - String differentiateString(String function)
  - static String formatSubscript (String function, boolean addTag)
  - String[] correctAnswers (String function, int numOrders)

FunctionSender Class (extends JFrame, implements ActionListener)

- Attributes
  - Socket socket
  - PrintWriter pw
  - JLabel ipLabel, functionLabel, gLabel
  - JTextField ipTF, functionTF
  - JButton sendButton, restartButton
- Methods
  - void actionPerformed (ActionEvent e)
  - static void main (String[] args) //run the other computer

FunctionsList Class

- Attributes
  - String fileName // gets files
- Methods
  - public FunctionsList(String fileName)
  - static String differentiate(String function)
  - String nextFunction() // returns the "base function"
  - boolean hasQuestions()

Tile Class (extends JComponent)

- Attributes
  - int topLeftX, topLeftY, width, height
  - java.awt.Color currentColor
  - java.awt.geom.Line2D[] lines
  - final Color selectedColor, final Color defaultColor
  - JFrame frame
  - JLabel label
- Methods
  - public Tile(String label, int tX, int tY, int height, int width)
  - void paintComponent(Graphics g)
  - void setLabel(String newLabel)
  - String getText()
  - void toggleSelected() //changes color to indicate player has left/entered this tile

TileManager Class (extends JComponent)

- Attributes
    - int topX, topY, tileWidth, tileHeight, numRows, numCols
    - Tile[][] grid
    - String[][] labels
    - JFrame frame
- Methods
    - public TileManger(int numRows, int numCols, int tileWidth, int tileHeight, int topX, int topY, String[][] gridLabels, JFrame frame)
    - void paintComponent(Graphics g)
    - setLabels(String[][] gridLabels)
    - int setLoc (int newRow, int newCol, Graphics g)
    - int curRow()
    - int curCol()
    - Tile[][] getGrid()

BoardState

- Attributes
    - int points
    - Differentiate() d
- Methods
    - void incrementPoints (int num)
    - void decrementPoints (int num)
    - static String[][] getGrid(String f, int numDerivs, int numOptions)