Jack Schlosser

Mr. Spicer

IB Compsci. 2

22 February 2016

**Criterion A: Planning**

**Defining the Problem**

A great way to engage the mind with an educational idea is through game. Games that challenge the player to think about numbers in unique and innovative ways are not only fun but highly educational. A great example of this is a game that has the player manage money and compete against a computer that also manages money. In an attempt to create my own version of a game that engages the user in a videogame using math and intuition, the aim of my project is to create a computerized lemonade stand game.

To achieve this, my computer science teacher has agreed to be my supervisor (advisor) and my classmate has agreed to test the product for me (client).

**Rationale for Solution**

The reasons for creating the game are to explore how different numbers and multipliers interact, to develop strategy skills when faced with a situation of managing money, and to ultimately have fun. First, the game uses an algorithm that the user can master over time to calculate how different upgrades impact their game. Understanding these different upgrades and figuring out how the algorithm can best benefit them is key to their success in the game and also has real life impacts. Second, because the game can be repeated and is essentially the same scenario over and over, users can develop strategies to beat the game at different levels, teaching them how to navigate issues of money management once they know how their money is spent. Finally, the game pits the user against a competitive AI and intrigues the user to continue playing and feel rewarded whenever they beat the AI at different levels of increasing difficulty.

Out of all programming languages, this project is best conducted in Java.

* Graphical User Interface – Jframe gives the program a user interface, complete with buttons. A useful element of the interface is action listeners that change interface values as the user interacts with the game.
* Command Prompt – Java can display instructions using the command prompt for the user while allowing the user to type in responses to the program to initiate different scenarios.
* Algorithm Development – Java’s in depth opportunities with number manipulation make it the best language for coding a program that deals heavily with mathematics.
* Simplicity – Java is a widely known and simple language that uses software almost every computer has in order to run.

**Success Criteria**

After discussing with my client, a successful system will:

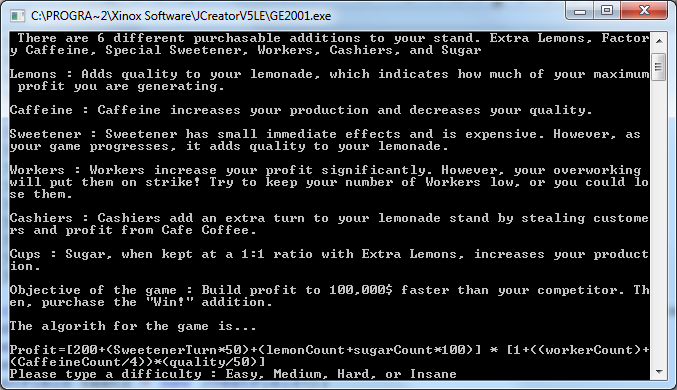
* Make effective use of a GUI that accurately represents the progression of the user and the computer, as well as being visually acceptable.
* Enable a winning condition for the user.
* Construct AI that progress in difficulty at a feasible rate.
* Develop an algorithm for the game that is complicated enough to encourage experimentation, but easy enough to understand to encourage calculation.
* Be able to identify when a player wins, and display which player won (AI or user).

**Criterion B: Design**

**Object Design**

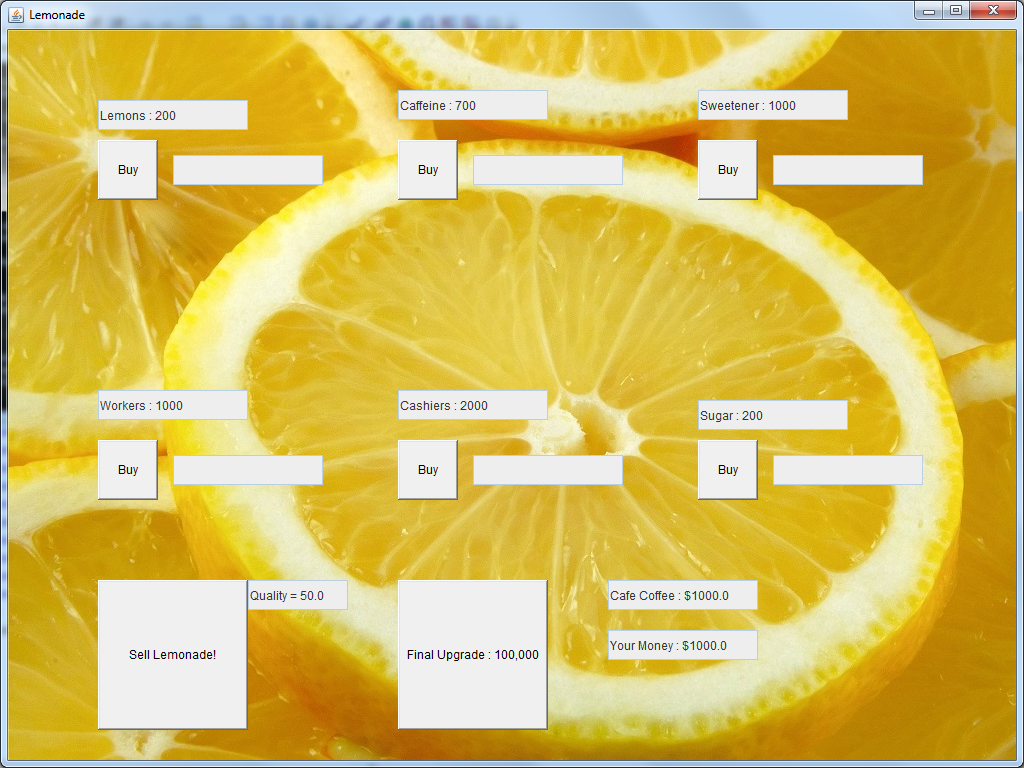
The program operates in two interfaces. First, the user reads a rule set offered in the command prompt, then types in a difficulty to play the game on.

Figure 1: Command Prompt Rules

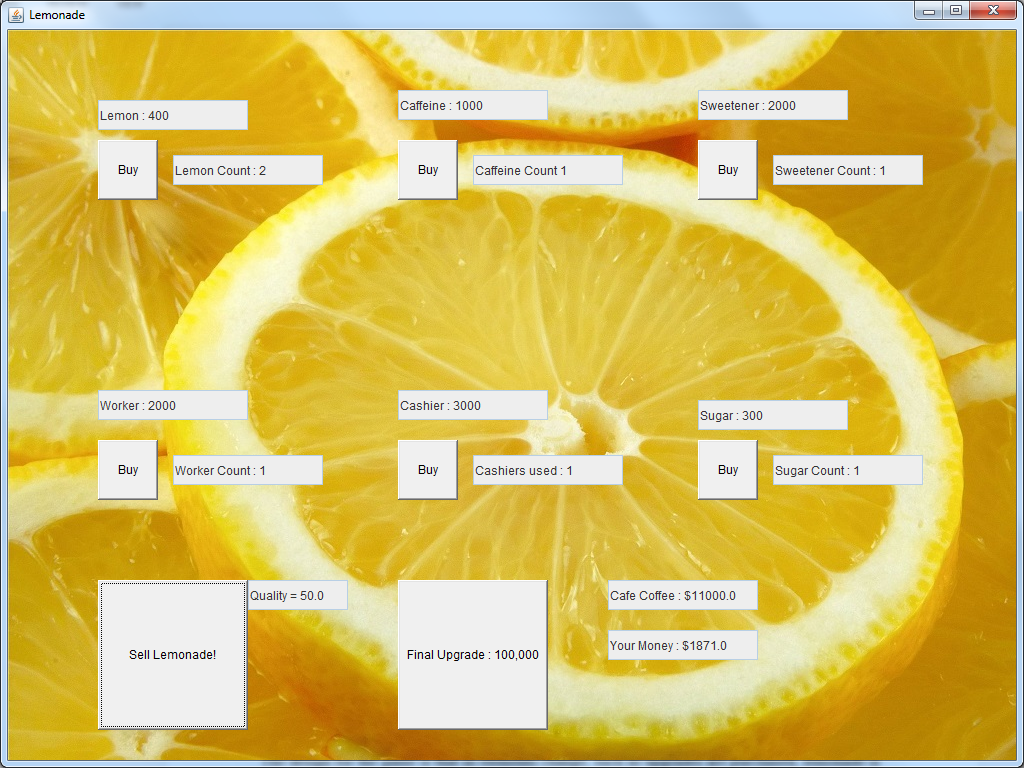


Then, the game opens the playable GUI. User can now click buttons to purchase upgrades or pass their turn. The GUI includes a button for each upgrade offered, a button that indicates selling lemonade and ending a player’s turn, a button that indicates a user winning the game, and two text fields that display the user’s and the computer’s current money. A lemon background was chosen to complement the theme of the game.

Figure 2: Graphical User Interface



The design for the game is that as elements change, such as upgrades are purchased, lemonade is sold and turns pass, the changes are displayed in text boxes near their respective button.

Figure 3: Graphical User Interface Interaction

When the game finishes, the user is met with a generic winning popup that displays the difficulty that the user has beat the game on.

Figure 4: Winning JFrame

**Processing Flowchart**

The flowcharts outline the logic of the program.

Start Insane game

Start Hard game

Start Normal game

Start Easy game

Choose Difficulty

Game winning screen appears

Buy Final Upgrade

Yes

User loses game

Yes

AI has $100,000?

No

AI Gains profit based on difficulty

Algorithm Calculates user profit

User sells lemonade to end turn

User has $100,000?

No

User purchase upgrades?

**Start game based on difficulty**

Upgrade Cost is subtracted from money

Yes

Activate Special Effect (If any)

**Development Pseudocode**

These methods are written in pseudocode to understand the premise behind the program in a simplistic manner.

**Choose Difficulty**

String Input //Checks user input

Int AIProfit = 1000 //Basic profit AI makes every turn. This changes based on difficulty.

System Println “Enter difficulty: Easy, Normal, Hard, or Insane”

Bufferedreader Input

If Input = “Easy”

AI Profit = AI Profit - 500

If Input = “Normal”

AI Profit = AI Profit

If Input = “Hard”

AI Profit = AI Profit + 1000

If Input = “Insane”

AI Profit = AI Profit + 2000

The user is told difficulty options and the AI’s profit multiplier is increased accordingly.

**Set Up GUI**

Button “X” //Declares buttons, multiple names used for buttons. X delineates each type.

setLayout(null)

BufferedImage = background.jpg //Sets background image

This.setContentPane(background) //adds background image

JTextField “X” //Declares JTextField, multiple names used for text fields. X delineates each type.

SetBounds Button (W,X,Y,Z) //Declares position (XY Axis) and shape of Button

SetBounds JTextField (W,X,Y,Z) //Declares position (XY Axis) and shape of JTextfield

Add(Buttons)

Add(Textfields)

Seteditable(Texfields)=false //Makes text fields static unless changed by program.

All buttons are declared for their name, their bounds assigned. All textfields are assigned a name, their bounds assigned. Then, all elements are added to the GUI.

**Purchase Upgrade, Main Method/GUI**

Int Money //Current user money amount

Int Cost of Upgrade //Price of the upgrade user is purchasing

String Upgrade //Checks what upgrade is being purchased

Actionlistener “e” Button //Checks if button is pressed

Then, this information is passed onto the Button class that branches off the single actionlistener “e” into different paths based on what button was pressed

**Purchase Upgrade, Button Class**

If Actionlistener Button “e” source=Upgrade && Money > Cost of Upgrade

Money = Money – Cost of Upgrade

Cost of Upgrade = Cost of Upgrade + X /

Textfield Display “Money”

Textfield Display “Cost of Upgrade”

This displays how the game subtracts cost of an upgrade from the user’s money, then displays the new values.

**End Turn, Main Method/GUI**

Int Money

Int AIMoney //How much money AI has

Int AIProfit = 1000 //How much the AI makes per turn

JTextField Money,AIMoney //Displays amount of Money and AIMoney on GUI

Actionlistener End Button

**End Turn, Button Class**

If Actionlistener e = Button

Money = Money + Profit Algorithm

AIMoney = AIMoney + AI Profit

AIMoney > 100,000

Jpane Output “You have lost.”

System Exit

Textfield Display “Money”

Textfield Display “AIMoney”

This displays how the game conducts the “Sell Lemonade” button, where the user passes their turn. The button adds profit to the user’s money pool while increasing the AI’s profit. Additionally, if the AI has won at the end of the turn, the program will display that the AI won.

**End Game, Main Method/GUI**

Actionlistener Final Upgrade //Checks if Final Upgrade Button is pressed

**End Game, Button Class**

If Actionlistener “e” Source = Final Upgrade && Money > 100,000

Jpane Output “You have beaten the game at “ + Difficulty + “Difficulty.”

System Exit

This displays how the game interacts with the player when they attempt to purchase the final upgrade.

**Record of tasks**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Number | Planned action | Planned Outcome | Time Estimated | Target Completion Date | Criterion |
| 1 | Decide on project | Decide on program that fits level of expertise and interests me | 1 Week | 12/1/15 | A |
| 2 | Discuss with advisor | Discuss with advisor to see if program is suitable | 1 Day | 12/8/15 | A |
| 3 | Research languages | Research necessary code to make program. | 1 Week | 12/15/15 | A |
| 4 | Draw GUI for program | Draw trial GUi to outline program’s visual experience | 2 Days | 12/17/15 | A |
| 5 | Define program criteria | Construct a criteria for what the program should strive to achieve | 1 Week | 12/24/15 | A |
| 6 | Complete coded GUI | Complete a coded version of only the interface in java | --- | 1/5/16 | B |
| 7 | Complete algorithmic design of program | Design the code of the program using algorithms | 1 Week | 1/12/16 | B,C |
| 8 | Code program | Code program | 1 month | 2/12/16 | C |
| 9 | Develop program | Finish coding program | 1 Week | 2/19/16 | C,D |
| 10 | Test with client | Test program with an objective 3rd party client | 2 Days | 2/21/16 | C,D |
| 11 | Improve program | User the client’s input on the program to make changes | 1 Week | 2/28/16 | D |
| 12 | Give program to advisor | Give these changes to advisor | 2 Days | 3/1/16 | D,E |
| 13 | Feedback from advisor | Hear feedback from advisor | 1 Day | 3/2/16 | E |
| 14 | Implement ideas for improvement | Think of ideas you can take, feedback from advisor, implement into code | 1 Week | 3/9/16 | B,C |
| 15 | Finish IA evaluation | Write complete evaluation of project | 1 week | 3/16/16 | All |

**Test Plan**

|  |  |
| --- | --- |
| Action to Test | Method of Testing |
| Make effective use of a GUI that accurately represents the progression of the user and the computer, as well as being visually acceptable. | Test if variables are being displayed correctly on screen through command prompt output. Ask client if program needs visual changes, make these changes. This includes making sure that all buttons work correctly and impact the game in their intended manner and all text fields update based on changes to variables within the program. |
| Test difficulty of program, make it challenging yet able to complete. | Place test values on difficulties, play through game with client, hear feedback on each difficulty of the game, make changes to balancing based on feedback. |
| Develop an algorithm for the game that is complicated enough to encourage experimentation, but easy enough to understand to encourage calculation. | Test a trial algorithm, run program with client to see if the algorithm is easily understandable, pinpoint specific points of algorithm that could be improved, make improvements to algorithm based on feedback. |
| Be able to identify when player or AI wins. | Run program to completion to check if end game screens are activating when the game is finished. |
| Enable a winning condition for the user. | Run program to completion to check if the user can operate the winning condition. |

**Criterion C: Development**

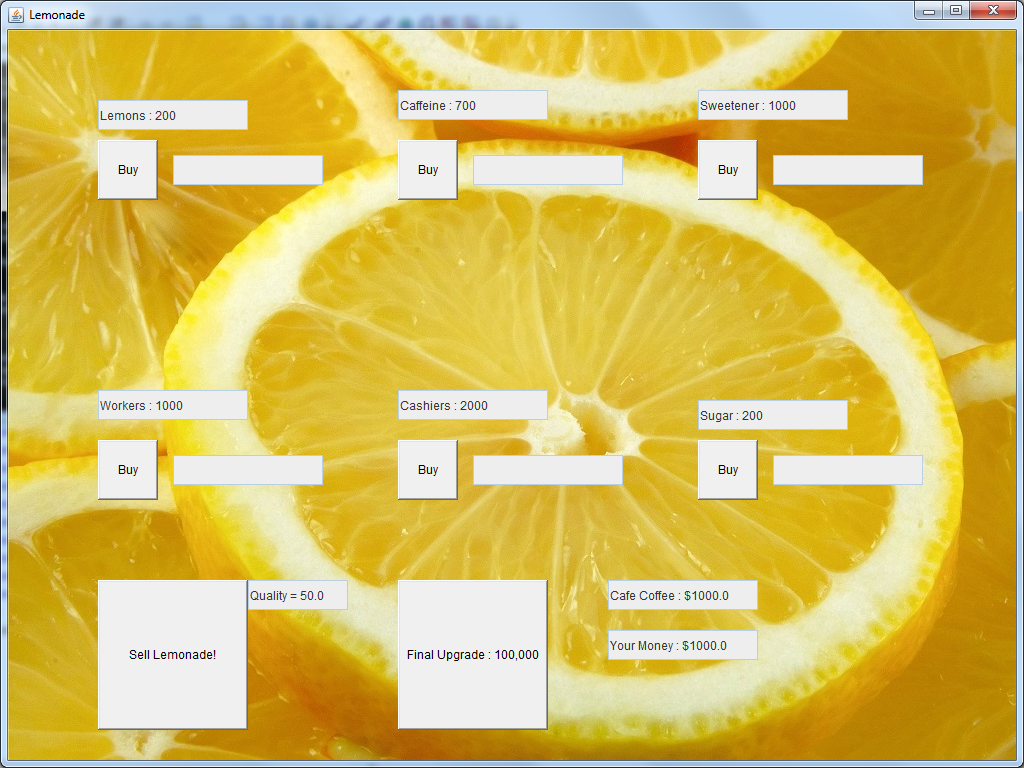
**List of Techniques**

The program uses the following techniques

* Graphical User Interface
* Artificial Intelligence
* Algorithmic calculations
* Object Inheritance
* Object-oriented programming

**Graphical User Interface**

The GUI allows the user to interact with the program visually and through the mouse rather than through the command prompt with their keyboard. It provides an atmosphere for information to be easily displayed to the user with simplistic access.



Window options

Textfields display amounts visually

Button allows user to purchase upgrades

Button allows user to end their turn

**Artificial Intelligence**

The Artificial Intelligence in the game is very simplistic. The AI operates at the end of every turn, gaining a set amount to their variable AIMoney. This AI could easily be expanded to conduct more intricate moves, such as purchasing upgrades on its own and its profit being randomly generated.

**Algorithm**

The majority of the program operates off a profit algorithm that calculates the user’s profit when they end their turn. The algorithm is as follows.

Profit=[200+(sweetenerTurn\*50)+(lemonCount+sugarCount\*100)] \* [1+((workerCount)+(caffeineCount/4))\*(quality/50)]

The algorithm has been balanced and altered over testing phases to help promote fairer gameplay. Here is an example of a previous algorithm that was used in the program.

Profit=[100+(SweetenerTurn\*100)+(lemonCount+sugarCount\*100)] \* [((workerCount)+(CaffeineCount/2))\*(quality/50)]

The algorithm was changed to prevent a bug where the profit multiplier section was not above 1, making the multiplier actually decrease profit. Furthermore, the upgrade caffeine was off balance for gameplay and promoted the user only buying that upgrade to win the game. To counter balance this, the caffeine’s impact on the profit algorithm was halved, making it more valuable for players to explore other aspects of the game to win.

**Inheritance**

The program makes use of inheritance through the extension of two crucial methods and the extension of one class: WindowListener, ActionListener, and JFrame

public class trialgame extends JFrame implements WindowListener,ActionListener

WindowListener allows for an interface that receives window events within the GUI that the program creates for the user. ActionListener, a far more crucial object, receives the input from pressed buttons within the program. Extending this is critical to the program running as the program’s main aspects cannot operate without the ActionListener “e” function.

Finally, extending JFrame makes the program a subclass of JFrame, operating in default under parameters JFrame operates under. This makes it simple to program the graphical user interface.

**Object-Oriented Programming**

This program makes use of object-oriented programming by seeing the program as constant interactions between different objects. The objects that the program interacts with are the ActionListeners from the buttons being pressed, the JTextFields that display the amounts, and the Jframes that operate the graphical user interface. As elements of each of these objects change, they interact with each other, as most of the program is active objects rather than passive entities.

**Criterion E: Evaluation**

**Meeting the Success Criteria**

After discussing with my client, a successful system will:

* Make effective use of a GUI that accurately represents the progression of the user and the computer, as well as being visually acceptable. – *This was met by constructing a GUI that makes it simple for the user to play and comprehend the game.*
* Enable a winning condition for the user. – *This was met by implementing the final upgrade button.*
* Construct AI that progress in difficulty at a feasible rate. – *This was met by implementing four different difficulties, each with varying success rates for the user. Harder difficulties are much more tedious to beat but it is possible.*
* Develop an algorithm for the game that is complicated enough to encourage experimentation, but easy enough to understand to encourage calculation. – *This is done through the profit algorithm that encourages the user to explore how all the variables in the program interact with their profit.*
* Be able to identify when a player wins, and display which player won (AI or user). – *This is done through having two winning conditions; One for the player and one for the AI. The player wins through buying the final upgrade. The AI wins by reaching $100,000.*

**Recommendations for Further Improvement**

After deliberation with the client, the thesis of the feedback was that the program needed a more multifaceted approach to playing. Although the game offered multiple paths of victory, the game revolving around an algorithm was simplistic but not as intriguing as adding multiple elements to play. Because the program is extendable, more nuanced approaches to the program could be implemented, such as more varied upgrades and more elements similar to the “Quality” element that makes the player manage their resources more intricately.

There was not an issue with AI difficulty, but rather that the player relied on luck to beat the AI rather than skill, due to the reliance on random number generation. Although some luck adds variance to the game and makes it engaging, forcing reliance on luck to see victory is unhealthy and should be changed by reducing the impact of random generation and increasing the amount of variables to master necessary to defeat the AI instead of increasing the amount of random generation that must fall in favor of the user.

The client suggested adding new interactive elements to the GUI. The program could be improved visually by adding new textures to the button to make them match the lemonade theme and by adding actions or sounds that take place when buttons on the screen are pressed. For example, when the user sells lemonade, a sprite of money could flow down the GUI with cash register sounds playing for the user. Adding visual and audio queues could make the game more captivating as well as keeping the user engaged in every action that they make.

The client suggested an opportunity for two player interaction. This would be a difficult addition as it would require two entirely different GUIs to interact, even though their information is encapsulated in their own method. What could be possible is to remove the aspect of the AI and simply have users share turns on the program. However, this would be clunky and eliminate the idea of freely controlling the flow of the game. This is an area that could be explored further.

Finally, the client suggested that the user be able to save their information to a text file to be accessed at a later time and to be able to see a recap on their actions when the game is complete to study ways they could improve their gameplay. For the first suggestion, a simple saving and loading method could be added to the program with a button that activates this method added to the GUI. For the second suggestion, an output could be read in the command prompt when the user completes the game that analyzes how much profit they were generating per turn, how many different upgrades they bought, and how many turns it took for them to complete the program.

Most feedback indicated the client and advisor were pleased with the gameplay. They were both intrigued to continue playing the game at different difficulties once learning how the game operated. The game was competitive for them and encouraged them to continue trying until they finally beat the game at the highest difficulty. This was rewarding for both testers as they had felt like they understood the function of a fairly complicated algorithm. They both saw definite improvements in how they approached the game with every play through and could pinpoint changes that they needed to make to progress to the next difficulty. Ultimately, the program achieved its goal of trying to encourage users to continue playing and giving a sense of reward when the user finally conquers the program’s AI.

Overall, multiple changes could be made to the gameplay intricacies and ease of access for the user, but the general reaction to the program has been positive. The GUI was successful in showing the user their progress and the finalized algorithm was tested and well balanced to ensure good gameplay. Users were pleased with how easy the game was to understand and how simplistic the goal of the game was, even if more elements could have been added to make the game more interesting and nuanced.

Appendix 1: Rules

Welcome to Lemonade Stand, the game! In this game, you are attempting to make a lemonade stand to compete against your competitor, Cafe Coffee.

There are 6 different purchasable additions to your stand. Extra Lemons, Factory Caffeine, Special Sweetener, Workers, Cashiers, and Sugar

Lemons: Adds quality to your lemonade, which indicates how much of your maximum profit you are generating.

Caffeine: Caffeine increases your production and decreases your quality.

Sweetener: Sweetener has small immediate effects and is expensive. However, as your game progresses, it adds quality to your lemonade.

Workers: Workers increase your profit significantly. However, your overworking will put them on strike! Try to keep your number of Workers low, or you could lose them.

Cashiers: Cashiers add an extra turn to your lemonade stand by stealing customers and profit from Cafe Coffee

Cups: Sugar, when kept at a 1:1 ratio with Extra Lemons, increases your production.

Objective of the game: Build profit to 100,000$ faster than your competitor. Then, purchase the "Win!" upgrade.

**Appendix 2: Code**

**Start Class**

//Start Class

public class trialgame extends JFrame implements WindowListener,ActionListener {

Random randomGenerator = new Random();

JTextField text = new JTextField();

JTextField text2 = new JTextField();

JTextField text3 = new JTextField();

JTextField text4 = new JTextField();

JTextField text5 = new JTextField();

JTextField text6 = new JTextField();

JTextField money = new JTextField();

JTextField name = new JTextField("Lemons : 200");

JTextField name2 = new JTextField("Caffeine : 700");

JTextField name3 = new JTextField("Sweetener : 1000");

JTextField name4 = new JTextField("Workers : 1000");

JTextField name5 = new JTextField("Cashiers : 2000");

JTextField name6 = new JTextField("Sugar : 200");

double quality = 50;

JTextField qualityField = new JTextField("Quality = "+quality);

Button b;

Button b2;

Button b3;

Button b4;

Button b5;

Button b6;

Button click;

Button saveButton;

Button loadButton;

Button finishgame;

private double count = 1000;

int[] boundsText={165,125,150,30};

int[] boundsButton={90,110,60,60};

int lemonCount=0;

int CaffeineCount=0;

int SweetenerCount=0;

int workerCount=0;

int cashierCount=0;

int sugarCount=0;

int lemonCost=200;

int CaffeineCost=700;

int SweetenerCost=1000;

int workerCost=1000;

int cashierCost=2000;

int sugarCost=200;

int SweetenerTurn = 0;

double multi = 1+((workerCount\*2)+(CaffeineCount/4))\*(quality/50);

double gain = 200+((SweetenerCount+SweetenerTurn)\*50)+(lemonCount+sugarCount\*100);

int coffee=1000;

int coffeegain=1000;

JTextField competitor = new JTextField("Cafe Coffee : $"+coffee+".0");

String difficulty="easy";

**GUI Class**

//GUI Class

public trialgame(String title) throws Exception{

System.out.println("Welcome to Lemonade Stand, the game! In this game, you are attempting to make a lemonade stand to compete against your competitor, \"Cafe Coffee\"."+

"\n\n"+" There are 6 different purchasable additions to your stand. Extra Lemons, Factory Caffeine, Special Sweetener, Workers, Cashiers, and Sugar\n");

System.out.println("Lemons : Adds quality to your lemonade, which indicates how much of your maximum profit you are generating.\n");

System.out.println("Caffeine : Caffeine increases your production and decreases your quality. \n");

System.out.println("Sweetener : Sweetener has small immediate effects and is expensive. However, as your game progresses, it adds quality to your lemonade.\n");

System.out.println("Workers : Workers increase your profit significantly. However, your overworking will put them on strike! Try to keep your number of Workers low, or you could lose them.\n");

System.out.println("Cashiers : Cashiers add an extra turn to your lemonade stand by stealing customers and profit from Cafe Coffee.\n");

System.out.println("Cups : Sugar, when kept at a 1:1 ratio with Extra Lemons, increases your production.\n");

System.out.println("Objective of the game : Build profit to 100,000$ faster than your competitor. Then, purchase the \"Win!\" addition.\n");

System.out.println("The algorith for the game is...\n");

System.out.println("Profit=[200+(SweetenerTurn\*50)+(lemonCount+sugarCount\*100)] \* [1+((workerCount)+(CaffeineCount/4))\*(quality/50)]");

System.out.println("Please type a difficulty : Easy, Medium, Hard, or Insane");

BufferedReader input=new BufferedReader(new InputStreamReader(System.in));

difficulty = input.readLine();

super(title);

setLayout(null);

addWindowListener(this);

BufferedImage bf = ImageIO.read(new File("background1.jpg"));

this.setContentPane(new backImage(bf));

b = new Button("Buy");

b2 = new Button("Buy");

b3 = new Button("Buy");

b4 = new Button("Buy");

b5 = new Button("Buy");

b6 = new Button("Buy");

finishgame = new Button("Final Upgrade : 100,000");

click = new Button("Sell Lemonade!");

text.setBounds(boundsText[0],boundsText[1],boundsText[2],boundsText[3]);

b.setBounds(boundsButton[0],boundsButton[1],boundsButton[2],boundsButton[3]);

boundsText[0]=boundsText[0]+300;

boundsButton[0]=boundsButton[0]+300;

text2.setBounds(boundsText[0],boundsText[1],boundsText[2],boundsText[3]);

b2.setBounds(boundsButton[0],boundsButton[1],boundsButton[2],boundsButton[3]);

boundsText[0]=boundsText[0]+300;

boundsButton[0]=boundsButton[0]+300;

text3.setBounds(boundsText[0],boundsText[1],boundsText[2],boundsText[3]);

b3.setBounds(boundsButton[0],boundsButton[1],boundsButton[2],boundsButton[3]);

boundsText[0]=boundsText[0]-600;

boundsButton[0]=boundsButton[0]-600;

boundsText[1]=boundsText[1]+300;

boundsButton[1]=boundsButton[1]+300;

text4.setBounds(boundsText[0],boundsText[1],boundsText[2],boundsText[3]);

b4.setBounds(boundsButton[0],boundsButton[1],boundsButton[2],boundsButton[3]);

boundsText[0]=boundsText[0]+300;

boundsButton[0]=boundsButton[0]+300;

text5.setBounds(boundsText[0],boundsText[1],boundsText[2],boundsText[3]);

b5.setBounds(boundsButton[0],boundsButton[1],boundsButton[2],boundsButton[3]);

boundsText[0]=boundsText[0]+300;

boundsButton[0]=boundsButton[0]+300;

text6.setBounds(boundsText[0],boundsText[1],boundsText[2],boundsText[3]);

b6.setBounds(boundsButton[0],boundsButton[1],boundsButton[2],boundsButton[3]);

click.setBounds(90,550,150,150);

finishgame.setBounds(390,550,150,150);

qualityField.setBounds(240,550,100,30);

competitor.setBounds(600,550,150,30);

name.setBounds(90,70,150,30);

name2.setBounds(300+90,60,150,30);

name3.setBounds(300+300+90,60,150,30);

name4.setBounds(90,300+60,150,30);

name5.setBounds(300+90,300+60,150,30);

name6.setBounds(300+300+90,300+70,150,30);

money.setBounds(600,600,150,30);

money.setText("Your Money : $"+ count+"");

add(b);

add(b2);

add(b3);

add(b4);

add(b5);

add(b6);

add(click);

add(text);

add(text2);

add(text3);

add(text4);

add(text5);

add(text6);

add(qualityField);

add(money);

add(name);

add(name2);

add(name3);

add(name4);

add(name5);

add(name6);

add(competitor);

add(finishgame);

text.setEditable(false);

text2.setEditable(false);

text3.setEditable(false);

text4.setEditable(false);

text5.setEditable(false);

text6.setEditable(false);

qualityField.setEditable(false);

money.setEditable(false);

name.setEditable(false);

name2.setEditable(false);

name3.setEditable(false);

name4.setEditable(false);

name5.setEditable(false);

name6.setEditable(false);

competitor.setEditable(false);

b.addActionListener(this);

b2.addActionListener(this);

b3.addActionListener(this);

b4.addActionListener(this);

b5.addActionListener(this);

b6.addActionListener(this);

click.addActionListener(this);

finishgame.addActionListener(this);

}

**Background Image Class**

class backImage extends JComponent {

Image i;

//Creating Constructer

public backImage(Image i) {

this.i = i;

}

//Overriding the paintComponent method

@Override

public void paintComponent(Graphics g) {

g.drawImage(i, 0, 0, null); // Drawing image using drawImage method

}

}

**Button Activation Class**

//Button Action Class

public void actionPerformed(ActionEvent e) {

if (e.getSource()==b&&count>=lemonCost&&lemonCount<=sugarCount+1){

lemonCount++;

text.setText("Lemon Count : "+lemonCount);

count=count-lemonCost;

if (quality<100)

quality=quality+5;

lemonCost=lemonCost+100;

qualityField.setText("Quality = "+quality);

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

name.setText("Lemon : "+lemonCost);

}

if (e.getSource()==b2&&count>=CaffeineCost){

CaffeineCount++;

text2.setText("Caffeine Count " + CaffeineCount);

count=count-CaffeineCost;

CaffeineCost=CaffeineCost+300;

quality=quality-10;

qualityField.setText("Quality = "+quality);

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

name2.setText("Caffeine : "+CaffeineCost);

}

if(e.getSource()==b3&&count>=SweetenerCost){

SweetenerCount++;

count=count-SweetenerCost;

SweetenerCost=SweetenerCost\*2;

if (SweetenerTurn==0)

SweetenerTurn=1;

name3.setText("Sweetener : "+SweetenerCost);

text3.setText("Sweetener Count : " +SweetenerCount);

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

}

if(e.getSource()==b4&&count>=workerCost){

workerCount++;

count=count-workerCost;

workerCost=workerCost+1000;

int workerRandom = randomGenerator.nextInt(10-workerCount);

if (workerRandom==1)

workerCount=0;

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

text4.setText("Worker Count : "+workerCount);

name4.setText("Worker : "+workerCost);

}

if(e.getSource()==b5&&count>=cashierCost){

cashierCount++;

text5.setText("Cashiers used : "+cashierCount);

coffee=coffee-2000;

count=count-cashierCost;

cashierCost=cashierCost+1000;

name5.setText("Cashier : "+cashierCost);

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

competitor.setText("Cafe Coffee : $"+coffee+".0");

}

if(e.getSource()==b6&&count>=sugarCost&&sugarCount<=lemonCount+1){

sugarCount++;

text6.setText("Sugar Count : "+sugarCount);

count=count-sugarCost;

sugarCost=sugarCost+100;

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

name6.setText("Sugar : "+sugarCost);

}

if(e.getSource()==finishgame&&count>=100000){

JOptionPane.showMessageDialog(null,"You crushed the competition! Nice job winning on "+difficulty+" difficulty!");

System.exit(0);

System.out.println("You crushed the competition! Nice job winning on "+difficulty+" difficulty!");

}

if (e.getSource()==click){

double multi = 1+((workerCount)+(CaffeineCount/4))\*(quality/50);

double gain = 200+(SweetenerTurn\*50)+(lemonCount+sugarCount\*100);

SweetenerTurn=SweetenerTurn+(SweetenerCount/2);

System.out.println("Your profit generated this turn: "+(gain\*multi));

count=count+(gain\*multi);

int countInt;

countInt = (int)count;

money.setText("Your Money : $"+countInt+".0");

if (difficulty.equalsIgnoreCase("easy"))

coffee=coffee+1000;

if (difficulty.equalsIgnoreCase("medium"))

coffee=coffee+2000;

if (difficulty.equalsIgnoreCase("hard"))

coffee=coffee+3000;

if (difficulty.equalsIgnoreCase("insane"))

coffee=coffee+4000;

competitor.setText("Cafe Coffee : $"+coffee+".0");

if (coffee>=100000){

JOptionPane.showMessageDialog(null,"Cafe Coffee beat you to $100,000! You lose this time.");

System.exit(0);

}

}

}

public void windowClosing(WindowEvent e) {

dispose();

System.exit(0);

}

public void windowOpened(WindowEvent e) {}

public void windowActivated(WindowEvent e) {}

public void windowIconified(WindowEvent e) {}

public void windowDeiconified(WindowEvent e) {}

public void windowDeactivated(WindowEvent e) {}

public void windowClosed(WindowEvent e) {}

**Main Method**

//Main Method

public static void main(String[] args) throws Exception{

trialgame myWindow = new trialgame("Lemonade");

myWindow.setSize(1024,768);

myWindow.setVisible(true);

}

}