Documentation Template

|  |
| --- |
| ***You MUST provide evidence showing how the problem has been decomposed, how the components have been developed and trialled, and of how they have been assembled and tested to create a final, working outcome.*** |

Getting Lucky

**Game Summary**

Choose number of rounds 10-30, objective is to reach the specified money goal based off the rounds within the number of rounds.

The money goal = rounds \* 10

Each round user gets to choose 4 options, Loot, Steal, Job, Shop. Choosing any options except shop will pass a round. E.G. Round 1: User chooses Loot, game now moves on to round 2.

**Loot:** gives you a chance to get $10 but also a chance to not get any money that round.

(Generate random, 80% chance to gain $10 and 20% chance to get nothing that round.)

**Steal:** gives you a chance to get a percentage of current balance or a chance to lose percentage of balance that round.

(50% chance to lose 30% of current balance or 50% chance to gain 40% of current balance)

**Job:** Nothing to lose but gives user a math question like 2+2 to get a small amount of money.

($8 if user gets question right and if wrong user gains nothing during that round.)

**Shop:** User can access a shop which displays a menu of upgrades with costs.

Upgrades can be bought multiple times

Shop sells upgrades:

**Loot Upgrade:** Increase reward from loot by $3 for $10 (e.g. loot upgrade bought once = loot now give $13)

**Steal Upgrade:** Increase percentage to get money from steal by 10% for $15(60% chance to gain and 40% chance to lose)

**Job Upgrade:** Increase reward from job by $5 for $20 (e.g. job upgrade bought once = job now gives $13)

When game ends user gets asked if they want to play again.

### Outline / Decomposition

*Please write down your task decomposition here (a numbered list is a good idea)*

1. Base Component
2. Yes and No checker
3. Instructions
4. Round Mechanics (asks for rounds and checks for valid input and ends game when rounds end or game quit)
5. Game checks for loot, steal, job, shop and XXX each round.
6. Generate random for loot
7. Workout percentage of balance for steal and 50/50 chance to lose or gain money
8. Generate random question for job
9. shop display menu of upgrades with cost
10. Looping the game / ask user to play again

### Flowchart

*Diagram

Description automatically generated*

### Version Log

### *<https://github.com/Fange-Wu/Getting-Lucky>*

### Component Testing

**Yes / No checker Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| Have you played before? User input: Yes | Game Starts |
| Have you played before? User input: No | Displays instructions and starts game |
| Have you played before? User input: Maybe  Have you played before?  User input: Nope  Have you played before?  User input: 1saf | <error> Please choose y / n    <error> Please choose y / n    <error> Please choose y / n |

**Rounds checker Test Plan (ask user to input number of rounds between 10 – 30)**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| Choose number of rounds:  User input: 10 | Game continues with 10 rounds |
| Choose number of rounds:  User input: 30 | Game continues with 30 rounds |
| Choose number of rounds:  User input: 15 | Game continues with 30 rounds |
| Choose number of rounds:  User input: 7  Choose number of rounds:  User input: 12.5  Choose number of rounds:  User input: 34  Choose number of rounds:  User input: asd | Displays error and asks questions again    Displays error and asks questions again    Displays error and asks questions again    Displays error and asks questions again |

**Choice checker Test Plan (Checks for the inputs <loot>, <steal>, <job>, <shop>**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| User Input: loot  User Input: steal  User Input:  job  User Input:  Shop  User Input:  xxx | You chose <user choice> |
| User Input:  LOOT  User Input:  STEAL  User Input:  JOB  User Input:  SHOP  User Input:  XXX | You chose <user choice> |
| User Input:  123  User Input:  Question  User Input:  ???  User Input:  exit | ERROR, please choose Loot, Steal, Job, Shop or XXX to quit  Ask question again |

**Loot Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| Choose loot 10 times | Loot should be chosen 10 times with a 20% chance to get no money and if they do get money, they gain $8 + current balance |

**Steal Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| Choose steal 10 times | Steal is looped 10 times with a 50/50 chance to lose 30% of current balance or gain 50% of money. (Balance is currently $100 for testing purposes). |

**Steal Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| Steal ran 10 times.  **Changes:** Changed the gained money from steal from 50% of balance to 40% of balance. I felt like gaining 50% of money is too much. |  |

**Job V1 Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| User Input: Correct Answer  User Input: Wrong Answer  User Input: sdk (Invalid input) |  |

**Job V2 Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| User Input: Correct Answer  User Input: Wrong Answer  User Input: k (Invalid input)  User Input: ‘:’ (Invalid input)  User Input: !@# (Invalid input)  User Input: Correct Answer |  |

**Shop Test Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| User Input: food  Expected: Invalid input / asks question again  User input: 1  Expected: Asks question again  User input: loot / steal / job (with sufficient money)  Expected: Takes money from balance according to the cost and displays menu again if they want to buy more.  User input: loot / steal / job (with insufficient money)  Expected: Outputs not enough money and calculates how much money the user is missing and asks question again.  User input: exit  Expected: exits the shop / breaks the while loop |  |

**End Game Test**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| User input: xxx  Expected: Asks user if they want to play again  If user didn’t meet money goal within the rounds played  Expected: Displays you lost and asks user if they want to play again  If User won / met money goal  Expected: Displays win and asks if user wants to play again |  |

**Play Again Plan**

|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| User input: no  User input: yes  User input: invalid |  |

### Assembled Outcome Testing

|  |  |
| --- | --- |
| **Test plan** | **Proof** |
| Input: maybe  Expected: asks question again  Input: yes – program continues  Input: 15 rounds  Input: loot  Input: loot  Input: loot  Input: shop  Input: loot (buys loot upgrade)  Takes $10 from balance  Input: exit  Input: steal  Got caught so loses 30% of current balance so minus $3 of current balance  Input: steal  Got caught so loses 30$ of current balance  Input: steal  Stole money so gains 40% of current balance  Input: loot  Since loot upgraded loot gives $13 instead of $10  Input: loot  Input: loot  Input: loot  Input: loot  Input: steal  Input: steal  Input: loot  Output: do you want to play again  Input: Maybe  Expected: asks question again  Game Loops | Text  Description automatically generated  Text  Description automatically generated  Text  Description automatically generated |

### 

### Usability Testing

*Write a list of things improvements which need to be made based on your usability testing. Then write down what you changed.*

Family Member – She said the shop was confusing/annoying to access because she had to scroll up to read what the upgrades did in the rules section. The shop was also clumped together so it was hard navigate.

**Changes:** *To change this I added what the upgrades did to the shop menu and made empty lines to separate the shop items so it was easier to look at. I deleted the message that output how much the item costed because it wasn’t necessary. Made the format look neater*

Friend – He found a problem with the shop, he said he couldn’t buy the upgrades when the balance was equal to the cost of the item and the job would break the program when he entered letters. He also didn’t know that upgrades could be bought multiple times he also got stuck in the shop because he didn’t know how to exit.

**Changes:** Fixed job and made job code more efficient and fixed shop problem by adding an equal sign. Problem was that I missed the equal sign. Job outputs incorrect input when typing letters unlike before. I also added that upgrades could be bought multiple times in the shop menu and telling the user to type exit to exit the shop.

Post Usability Test…

**Shop changes + fixes**

The upgrades now can be bought now when the balance is equal to the price of the upgrade. Made the overall look more organized and better and added text telling the user that upgrades can be bought multiple times and typing exit to exit the shop.

|  |  |
| --- | --- |
| ***Before:*** | ***After*** |
| ***Text  Description automatically generated*** | *Text  Description automatically generated* |

**Job fix**

Fixed the code so that when text was inputted into job it wouldn’t break the program. It asks the same question again when an invalid input is inputted.

|  |
| --- |
| ***Before:*** |
|  |
| ***After:*** |
| *A picture containing text, sign, red, close  Description automatically generated* |

Social and End User Considerations…

**Who are your target audience / users?**

*Classmates / People who know how to code and play a text based game.*

**Why did you need to ensure that your task was suitable for your chosen audience?**

*I needed to ensure my game was suitable for my chosen audience so that it wasn’t boring and was a game that could be played simply.*

**How did you ensure that your task was suitable for your chosen audience?**

*I ensured it by making my game easy to use and having instructions that could be easily understood and with the help of a organized layout with the text.*

**Why do we honour copyright?**

*Copyright protects people creations from being used without the owner’s permission and this helps people who create original work like music and songs. Copyright protects their reputation.*

**How have you honoured copyright?**

*I have honoured copyright by not exactly copying someone else’s code. For example taking someone’s code and changing it up so it’s not exactly like its original. Changed it so that it fits my program.*

**Why do you need to make your quiz easy to use?**

*The user won’t be confused or stuck throughout the game. The user will be able to understand everything about your game and make it to the end with the problems. Easy to use because you don’t want a user quitting your game because he couldn’t understand it or it was to hard to use.*

**How did you make your quiz easy to use?**

*I made my game easy to use by using clear instructions and text and an easy looking and simply layout. Includes instructions so that the user is able to make it through the game. Including error messages and repeating the question when the user inputs an invalid input so that the user can know what to actually input.*