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.*** |

### **Outline / Decomposition**

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

1. Ask the user if they have played the game before, if not then show the instructions.
2. Ask user to pay an initial amount for how many rounds they want to play, $1 per round, maximum amount of money the user can spend is $10 per session
3. Generate a random token that is either a zebra, horse, donkey or unicorn. If the token is a unicorn, the user wins $5, if it is a horse or donkey, they win $0.50 and if it is a donkey, they don’t win anything.
4. Work out how much money the user won / lost and how much they have left each round.

### **Flowchart**

*Please show a developed flowchart of your program below (you may use draw.io to create your flowchart)*

*Diagram, engineering drawing

Description automatically generated*

### **Version Log**

*Your version log should go here. Annotated screenshots are a good idea at this point*

**Instructions Component**

*Text

Description automatically generated*

**Trial 1 of the yes/no checker for the instructions (01\_yes\_no\_v1.py)::**

This piece of code worked for both UPPER and lower-case characters but was inefficient. It has five if / elif / else statements and quite a bit of repeated code.

*Text

Description automatically generated*

**Trial 2 of the yes/no checker for instructions (01\_yes\_no\_v2.py)::**

I used the ‘or’ command to combine my ‘yes’ / ‘y’ and ‘no’ / ‘n’ statements and this code is more efficient compared with trial #1.

*Text

Description automatically generated*

**Trial 3 of the yes/no checker (02\_yes\_no\_v1.py):**

I made the code from trial 3 into one function which makes it easier to ask more than one yes / no question in a program. I will use this function in my Lucky Unicorn outcome.

**Amount Played Component**

**Text

Description automatically generated**

**Trial 1 of the amount played component (04\_how\_much\_v1.py):**

This program follows the results for the expected values from the test plan, the code works but would be more efficient in a function for reusability in the future.

Text

Description automatically generated

**Trial 2 of the amount played component (04\_how\_much\_v2.py):**

This program works in the same way as trial #1 but by placing the code from the previous trial into the function, I can reuse it easily and I can collapse the function so that lines of code do not fill my screen and make the code complicated. I will use this function in my Lucky Unicorn outcome.

**Token Generator Component:**

Text

Description automatically generated

**Trial 1 of token generator component (05\_how\_much\_v1.py):**

This program generates tokens completely randomly, but the user balance has not been implemented as of yet.

Text

Description automatically generated

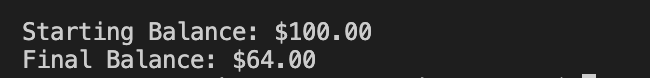
**Trial 2 of token generator component (05\_how\_much\_v2.py):**

This program generates random tokens and also contains the balance of the user. It has also come to my attention that in the long run, the house loses money instead of making money.

Text

Description automatically generated

**Trial 3 of token generator component (05\_how\_much\_v3.py):**

I fixed the loss of profit issue faced by the previous house by putting the unicorn token in the list once and putting all the other tokens in three times each:

Results for 100 rounds, 10% chance of a unicorn, 30% chance for each of donkey, zebra and horse

Text

Description automatically generated

**Trial 4 of token generator component (05\_how\_much\_v4.py):**

The code has been reduced to ten rounds, where chance of unicorn is 5%, chance of donkey is 30% and chance of horse / zebra is 65%. If we run the program for more than 100 rounds, the house advantage becomes clearer. But with less rounds the user will experience some wins and losses, so that they will think the odds are in their favor.



In my ‘else’ statement, to decide between a horse and zebra, I set it up so that if the number that was generated was even, the token was a horse, otherwise it was a zebra.

### **Component Testing**

*Show that you have tested each component here. You should have a test plan and then screenshot proof for each component. You should also include notes that justify the major decisions you made.*

**Instructions Component**

**Yes / No Checker Test Plan:**

Graphical user interface, text, application

Description automatically generated

**Yes / No Checker Expected Values Test Plan:**

Table

Description automatically generated

**Yes / No Checker Testing:**

Text

Description automatically generated

All cases worked as expected (code has been looped to make testing easier)

**Instructions (and Yes / No checker) Testing:**

Text

Description automatically generated

Test Case 1 (maybe then ‘yes’) – output is as expected

Text

Description automatically generated

Test Case 2 (no) – output is as expected

The code has been continually refined as at the start it was case sensitive by accepting only lowercase answers and being inefficient. Then the code was enhanced to accept lowercase/UPPERCASE and became efficient through the use of functions.

**Amount Played Component**

**Number Checker Test Plan:**

Table

Description automatically generated

**Number Checker Testing:**

Text

Description automatically generated

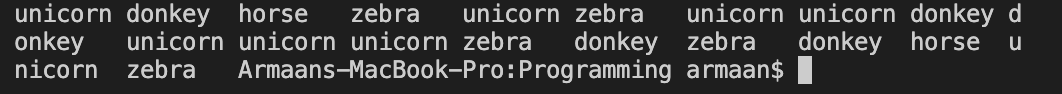
All tests passed. Program works correctly for unexpected, expected and boundary cases (code has been looped to make testing easier).

**Token Generator Component**

**Token Generator Test Plan:**

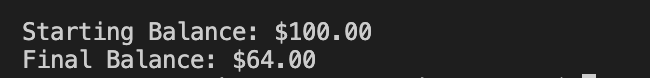
|  |  |
| --- | --- |
| **Test Case** | **Expected Values** |
| Run the program (random) | Program should output 20 tokens with at least one horse, donkey, zebra and unicorn. Order of tokens should be random. |
| If user starts with $100  Run program (house advantage) | User should lose money (financial balance should be less than $100) |

**Token Generator (random) Testing:**



Of the 20 looped tokens, there is at least one unicorn, zebra, horse and donkey. The program works as expected.

**Token Generator (house advantage) Testing:**



In the long run, in this case 100 rounds, the house has an advantage and gains profit.

Results for 100 rounds, 10% chance of a unicorn, 30% chance for each of donkey, zebra and horse.

### Assembled Outcome Testing

*Please show testing for your assembled outcome below. This should include a test plan followed by screenshot proof*

### Usability Testing

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

### Post Usability Test…

*Show that your post usability testing program works correctly*

### Social and End User Considerations…

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

*Answer here*

**How have you honoured copyright?**

*Answer here*

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

*Answer here*