Use Cases by Ian McKechnie

Use Case 1

Put a die in the players bag

Primary actor: Active player

Goal: Instantiate a die object, then put it in the player objects bag.

Stakeholder list: The active player

Initiating event: Player wants to add a die to their bag

Main Success Scenario:

1. Active player instantiates a player object with name “Jasmine”
2. Active player instantiates a die object with the colour black and 6 sides
3. Active player sets the description of the die to a description of the die
4. Active player calls store on the die object
5. Active player calls the load method on the player object with the same description used on the die

Exceptions:

1. Active player does not want a name
    a. Active player leaves the arguments blank when instantiating the player object
2. Active player has different die configuration than black with six sides
    a. Active player changes the arguments when creating the die
3. Active player wants a different amount of arguments given to the die
    a. Active player adds or removes keys and values to the hash list given as the dies description

Use Case 2

Player moves randomizer objects to the cup and throws them

Primary actor: Active player

Goal: Move randomizers from the cup to the bag and throw the cup

Stakeholder list: The active player

Initiating event: Player decides they want to throw the randomizers

Main Success Scenario:

1. Active player moves all elements over to cup from their bag
2. Active player calls the throw method and throws the cup
3. Active player records results of the throw with the results object
4. Active player puts randomizers back into their bag from their cup using the same description as they did in step 1

Exceptions:

1. Active player only wants to move subset of elements
    a. The active player would pass the subset item description to the loads method

Use Case 3

Player wants to compare different results from consecutive throws

Primary actor: Active player

Goal: Compare the results of the randomizers after making multiple throws

Stakeholder list: The active player

Initiating event: Player decides to see how different throws affect outcomes

Main Success Scenario:

1. Active player moves all elements over to cup from their bag
2. Active player calls throw method
3. Active player calls throw method
4. Active player calls throw method
5. Active player puts randomizers back in bag using same description as in step 1
6. Active player calls results passing arguments {} for all randomizers and 2 for the throw history
7. Player compares the results of the 1st, 2nd, and 3rd throw

Exceptions:

5. Active player wants to have more throws
    a. Active player continues calling the throw method
6. Player wants different subset of throw history
    a. Active player calls results method with {} and 1
    b. Active player receives throw history from oldest and second oldest throws

Use Case 4

Player wants the sum from a throw

Primary actor: Active player

Goal: To have the sum of all sides up after throwing the randomizers

Stakeholder list: The active player

Initiating event: Player wants the sum of all items thrown from cip

Main Success Scenario:

1. Active player instantiates a coin object
2. Active player instantiates a die object
3. Active player instantiates a die object
4. Active player adds all randomizers to their bag
5. Active player moves all items from their bag to their cup
6. Active player calls throw method on the cup
7. Active player moves all randomizers back to bag from cup using the same description from step 5
8. Active player calls sum method with argument {} to see the sum of the results of all the randomizers thrown

Exceptions:

4. Active player wants more randomzziers
    a. Active player continues instantiating randomizers
8. Active player only wants sum from the dice
    a. Active player changes the description being passed to sum method to {‘item’ => :die}