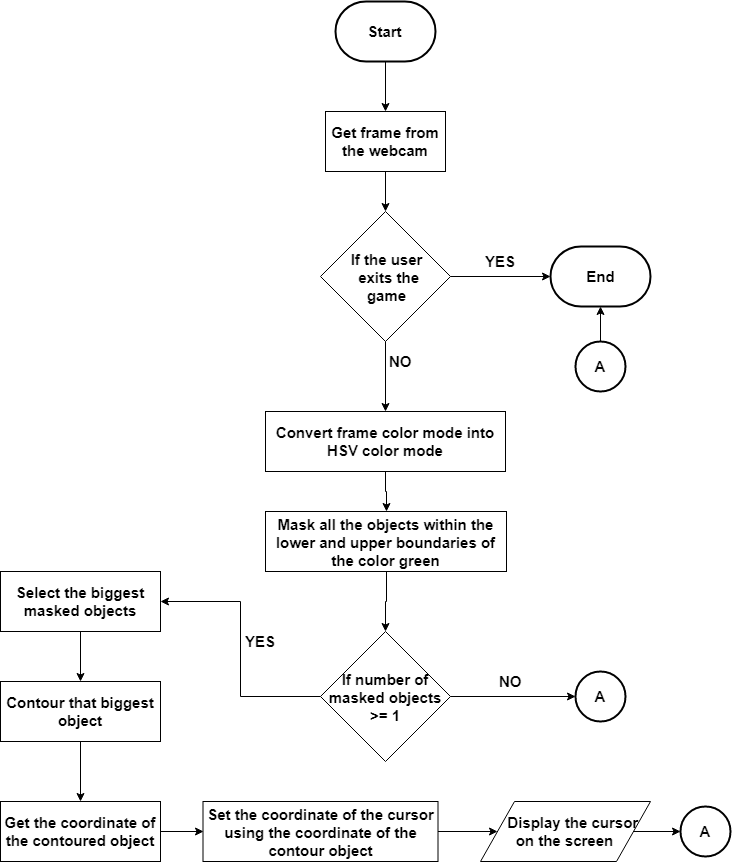
1. adding bonus points;
2. tracking the coordinates of sword’s location; and
3. graphical user interface

After the researchers tested and performed a number of dry-runs, they found out that the objectives of the game were met. These were as follows:

**Specific Objective 1: Discuss the developed system’s process of detecting and tracking the movement of the sword.**



*Figure* 5.1 Process of detecting and tracking the movement of the sword

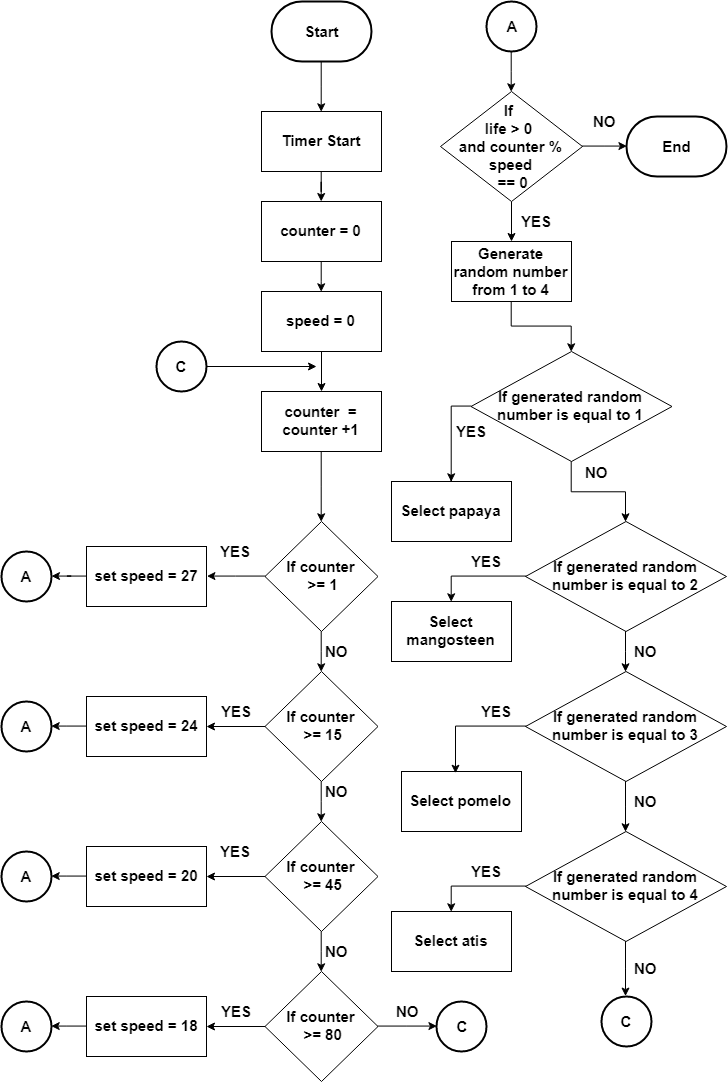
Figure 5.1 showed the algorithm on how the system detected and tracked the movement of the sword. First, the system gets the frame from the webcam. After which, the frame was converted from Red Green Blue (RGB) colour space into Hue Saturation Value (HSV) colour space in order to detect the green color from the lightest to the darkest colour. For the colour space conversion, the function cv2.cvtColor (input image, flag) was used to determine the type of conversion. Descriptions in terms of HSV are often more relevant. Thus, for the flag cv2.COLOR\_BGR2HSV to be used, it should be converted because the RGB components of an object’s colour in a digital image were all correlated with the amount of light hitting the object, and therefore with each other, image descriptions in terms of those components make object discrimination difficult.

Then, the system masked all the objects that were within the lower and upper boundaries of the green colour in the HSV colour space. These colour boundaries allowed the system to detect the green ball of the sword, provided that at least one or more objects with the area has been contoured. After which, the system took the coordinates of the biggest contoured object to represent the position of the cursor on the screen of the developed system.

**Specific Objective 2: Design the algorithms used to randomly select:**

**a. objects such as**

**a.1 fruits**

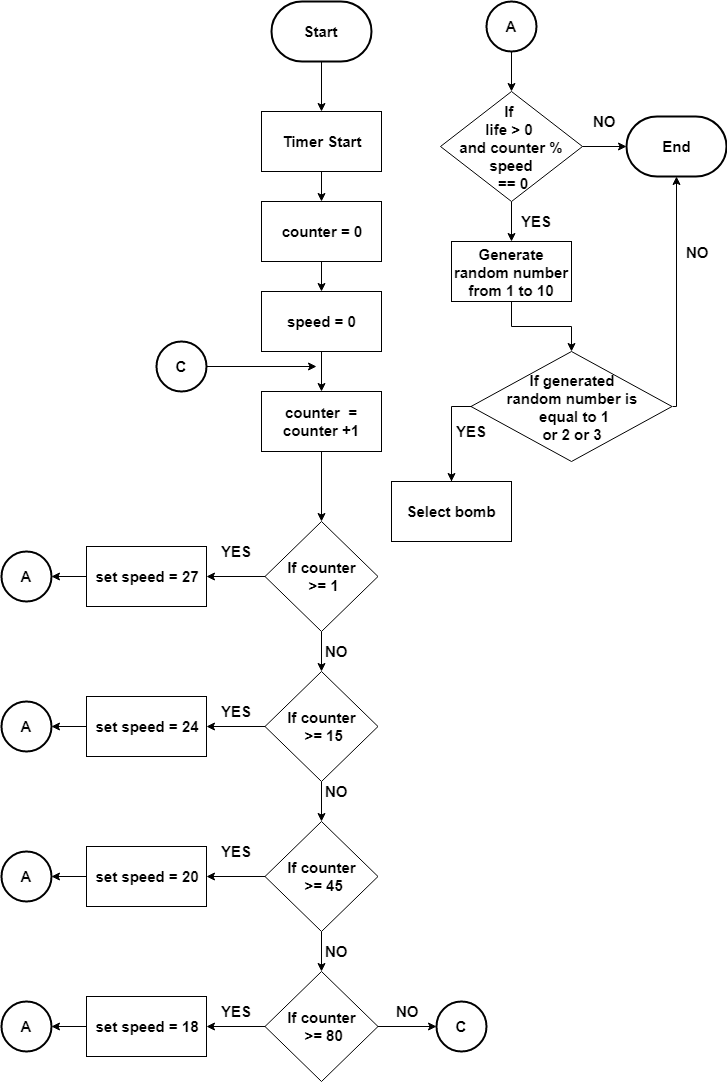


*Figure* 5.2.1 Algorithm used to randomly select fruits

Figure 5.2.1 showed the algorithms that were used to randomly select the fruits to be dropped in the game. First, the system started its timer then the variable *counter* was initialized to 0. This variable was responsible for the timer of the game. The variable *speed* was initialized to 0. It was responsible on how often the objects such as fruits and bomb dropped on the game. After that, the developed system generated a counter that equals to counter plus 1, which means that if the counter is greater than or equal to one, the set speed of the falling fruits were equal to 27, the higher the number of the variable the lesser the objects dropped. If not, the counter proceeded to another condition that, if the counter is greater than or equal to 15, the set speed of the system is equal to 24. However, if the counter is greater than or equal to 45, the set speed of the falling fruits is 20, else the counter continue to generate. If the counter is greater than or equal to 80, the set of speed of the falling fruits is 18, which means that the speed of the falling fruits was slowly fastened every time the player earned the highest points. If it does not qualify the criteria mentioned above, the developed system would go over to initialize from the start of the program followed by the counter re-initialization, and then the counter goes through the whole process again.

In generating different fruits, if the life is greater than 0 and the counter modulus speed is equal to zero, the developed system generated random numbers from one to four. If the generated number is equal to one, the developed system selects the papaya fruit. Else, if the generated number is equal to two, the developed system selects mangosteen fruit. Moreover if the generated number is equal to three, the developed system selects pomelo fruit. Otherwise, if the generated number is equal to four, the developed system selects atis fruit. If life is equal to zero, then the developed system ends.

**a.2 bombs**

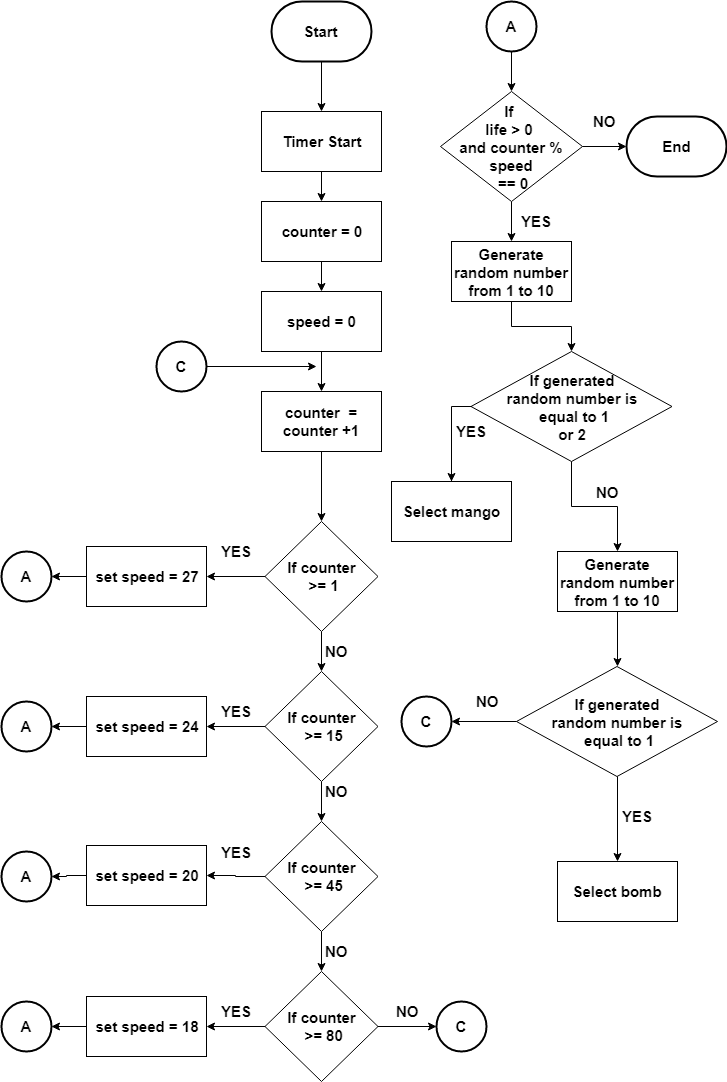


*Figure* 5.2.2 Algorithm used to randomly select bombs

Figure 5.2.2 showed the algorithms that used to randomly select the bombs. First, the system started its timer. The variable *counter* was initialized to 0. This variable was responsible for the timer of the game. The variable *speed* was initialized to 0. It was responsible on how often the objects such as bomb dropped on the game. After that, the developed system generated a counter that is equal to counter plus 1, which means that if the counter is greater than or equal to one, the set speed of the falling bomb were equal to 27. The higher the number of the variable, the lesser the objects dropped. If not, the counter proceeded to another condition that, if the counter is greater than or equal to 15, the set speed of the system is equal to 24. However if the counter is greater than or equal to 45, the set speed of the bomb is 20, else, the counter continue to generate. If the counter is greater than or equal to 80, the set of speed of the falling bomb is 18, which means that the speed of the falling bomb was gradually increasing. If it does not qualify the criteria mentioned above, the developed system would go over to initialize from the start of the program followed by counter re-initialization, and then the counter goes through the whole process again.

In generating bombs, if the life is greater than 0 and the counter modulus speed is equal to zero, the developed system generated random numbers from one to ten. If the generated number is equal to one, two or three, the bomb dropped. If the algorithm does not satisfy these conditions, then it ends.

**a.3 Bonus points**

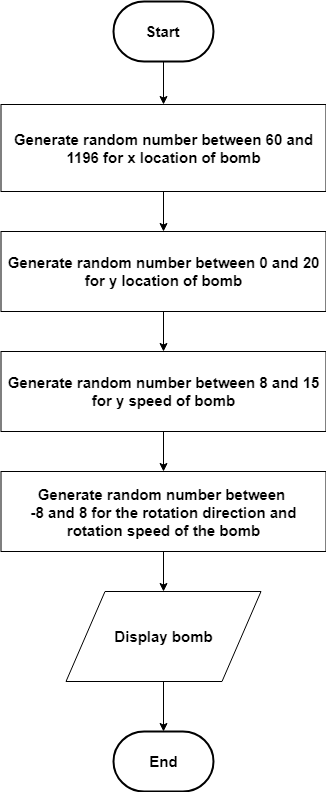


*Figure* 5.2.3 Algorithm used to randomly select special fruits with bonus points

Figure 5.2.3 showed the algorithms that were used to randomly select the special fruits to be dropped in the game. First, the system started its timer. The variable *counter* was initialized to 0. This variable was responsible for the timer of the game. The variable *speed* was initialized to 0. It was responsible on how often the objects such as special fruits dropped on the game. After that, the developed system generated a counter that is equal to counter plus 1, which means that if the counter is greater than or equal to one, the set speed of the falling special fruits were equal to 27, the higher the number of the variable the lesser the objects dropped. If not, the counter proceeded to another condition that, if the counter is greater than or equal to 15, the set speed of the system is equal to 24. However if the counter is greater than or equal to 45, the set speed of the bomb is 20, else the counter continues to generate. If the counter is greater than or equal to 80, the set of speed of the falling special fruits is 18, which means that the speed of the falling special fruit was gradually increasing every time the player earned the highest points. If it does not qualify the criteria mentioned above, the developed system would go over to initialize form the start of the program followed by counter re-initialization, and then the counter goes through the whole process again.

In generating special fruits, if the life is greater than 0 and the counter modulus speed is equal to zero, the developed system generated random numbers from one to four. If the algorithm does not satisfy this condition, then it ends. But if the condition is satisfied the program generated random numbers from one to ten. If the generated number is equal to one or two, mango is selected, else, durian is selected. If the conditions were not satisfied, then the process repeated from the start.

**b. object’s location**



*Figure* 5.2.4 Algorithm used to randomly select the object’s location

Figure 5.2.4 showed that the location of the object such as bomb is illustrated. First, the developed game generated random numbers between 60 and 1196 for “x” location of the bomb. The bomb location appeared only within these specified values. Going above or below the specified numbers made the bomb appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the bomb. This is for the head start of the bomb. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the bomb. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the bomb that starts to drop within the range of “x” and “y”.

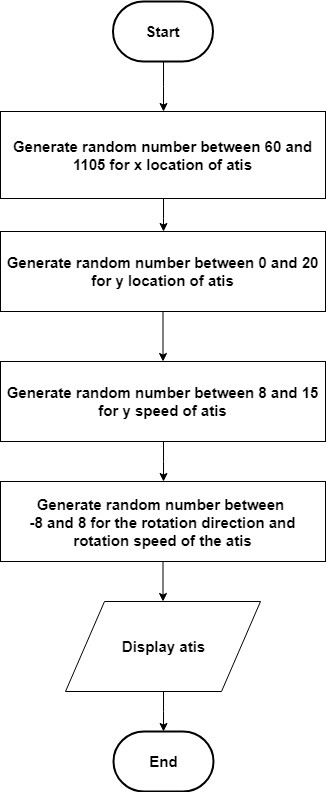


Figure 5.2.5 Algorithm used to randomly select the atis’ location

Figure 5.2.5 showed that the location of the object such as atis is illustrated. First, the developed game generated random numbers between 60 and 1105 for “x” location of the atis. The atis’ location appeared only within these specified values. Going above or below the specified numbers made the atis appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the atis. This is for the head start of the atis. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the atis. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the atis that starts to drop within the range of “x” and “y”.

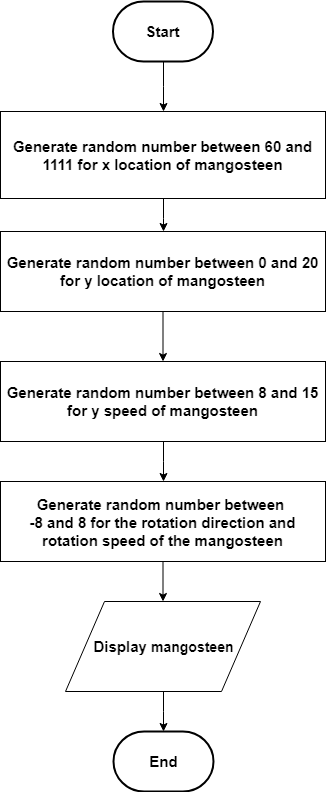
****

Figure 5.2.6 Algorithm used to randomly select the mangosteen’s location

Figure 5.2.6 showed that the location of the object such as mangosteen is illustrated. First, the developed game generated random numbers between 60 and 1111 for “x” location of the mangosteen. The mangosteen’s location appeared only within these specified values. Going above or below the specified numbers made the mangosteen appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the mangosteen. This is for the head start of the mangosteen. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the mangosteen. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the mangosteen that starts to drop within the range of “x” and “y”.

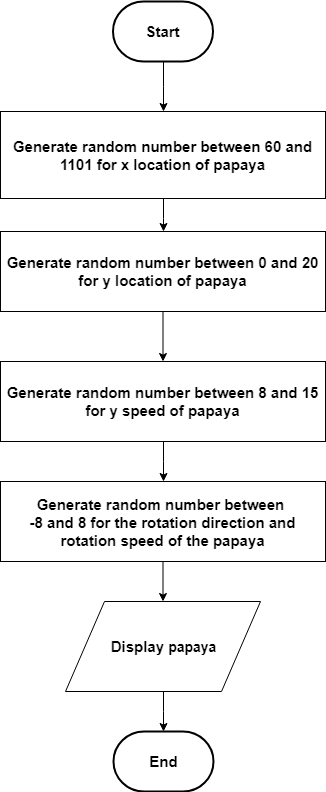


Figure 5.2.7 Algorithm used to randomly select the papaya’s location

Figure 5.2.7 showed that the location of the object such as papaya is illustrated. First, the developed game generated random numbers between 60 and 1101 for “x” location of the papaya. The papaya’s location appeared only within these specified values. Going above or below the specified numbers made the papaya appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the papaya. This is for the head start of the papaya. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the papaya. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the papaya that starts to drop within the range of “x” and “y”.

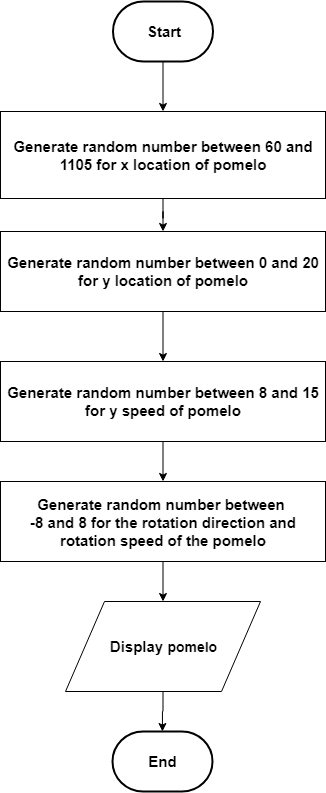


Figure 5.2.8 Algorithm used to randomly select the pomelo’s location

Figure 5.2.8 showed that the location of the object such as pomelo is illustrated. First, the developed game generated random numbers between 60 and 1105 for “x” location of the pomelo. The pomelo’s location appeared only within these specified values. Going above or below the specified numbers made the pomelo appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the pomelo. This is for the head start of the pomelo. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the pomelo. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the pomelo that starts to drop within the range of “x” and “y”.

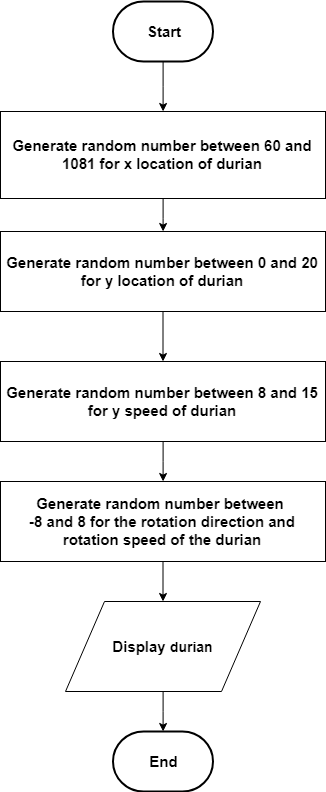


Figure 5.2.9 Algorithm used to randomly select the durian’s location

Figure 5.2.9 showed that the location of the object such as durian is illustrated. First, the developed game generated random numbers between 60 and 1081 for “x” location of the durian. The durian’s location appeared only within these specified values. Going above or below the specified numbers made the durian appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the durian. This is for the head start of the durian. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the durian. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the durian that starts to drop within the range of “x” and “y”.

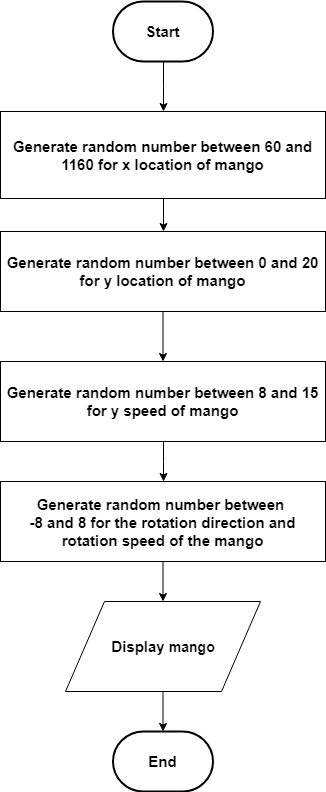
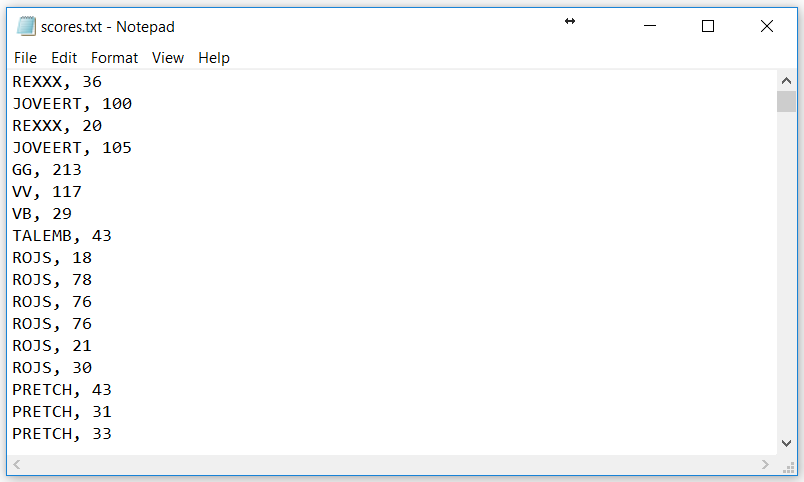


Figure 5.2.10 Algorithm used to randomly select the mango’s location

Figure 5.2.10 showed that the location of the object such as mango is illustrated. First, the developed game generated random numbers between 60 and 1160 for “x” location of the mango. The mango’s location appeared only within these specified values. Going above or below the specified numbers made the mango appeared beyond the screen. Then game generated random numbers between zero and twenty for “y” location of the mango. This is for the head start of the mango. The greater the generated number, the bigger the head start. The dropping of the fruits could appear only with the distances stated in the game. Then it generated random numbers between negative eight (-8) and positive eight (+8) for the rotation direction and rotation speed of the mango. If the result is negative, the rotation is counter clockwise and when the result is positive, the rotation is clockwise. If the generated number is near to zero, its rotation is slower. Then the screen displayed the durian that starts to drop within the range of “x” and “y”.

**Specific Objectives 3: Use flat file database in storing player’s information such as name and score.**



*Figure 5.3.1* Structure of Flat File Database of the developed system

Figure 5.3.1 shows the database structure of the JoRex: An Artificial Intelligence (AI) Motion-based Fruit Razor Computer Game. The information of the players is stored on a flat file database. Each line of the text file holds a single record separated by a delimiter such as comma. In the proposed game, the table schema holds the name and score played by the player.

**Specific Objectives 4: Test the accuracy and friendliness of the developed system in terms of:**

1. **Accuracy**

Table 5.4.1 Extent of Accurateness of the developed system (n=246)

|  |  |  |
| --- | --- | --- |
| Item | Mean | Description |
| 1.The developed system accurately cuts the following objects: |  | |
| a. Fruits |  | |
| a.1 durian | 3.89 | Strongly Agree |
| a.2 mango | 3.88 | Strongly Agree |
| a.3 mangosteen | 3.88 | Strongly Agree |
| a.4 papaya | 3.88 | Strongly Agree |
| a.5 pomelo | 3.87 | Strongly Agree |
| a.6 sugar fruit | 3.89 | Strongly Agree |
| b.Bombs | 3.82 | Strongly Agree |
| 2. The developed system accurately adds the corresponding points of the following objects when cut: |  | |
| a. Fruits |  | |
| a.1 durian | 3.90 | Strongly Agree |
| a.2 mango | 3.86 | Strongly Agree |
| a.3 mangosteen | 3.87 | Strongly Agree |
| a.4 papaya | 3.89 | Strongly Agree |
| a.5 pomelo | 3.90 | Strongly Agree |
| a.6 sugar fruit | 3.88 | Strongly Agree |
| 3. The developed system gives bonus points on the following special fruits when cut: |  | |
| a. Durian (points multiplier for 10 seconds) | 3.85 | Strongly Agree |
| b. Mango (5 points) | 3.82 | Strongly Agree |
| 4. The developed system easily detects the coordinates of sword’s location. | 3.69 | Strongly Agree |
| 5. The developed system generates an accurate total score at the end of the game. | 3.65 | Strongly Agree |
| 6. The developed system ends the game when the player cuts the bomb. | 3.77 | Strongly Agree |
| **Overall** | **3.84** | **Strongly Agree** |

*Table 5.4.1* shows the extent of accurateness of the developed system. All items in Table 5.4.1 were described as Strongly Agree. On one hand, items 2.a.1 and 2.a.5 which are durian and pomelo respectively, obtained the highest mean of 3.90. This means that the objects which are durian and pomelo have a greater chance of falling in a game. On the other hand, item 5 which is the developed system generates accurate total score at the end of the game got the lowest mean of 3.65, also interpreted as strongly Agree.

The over-all mean is 3.84 described as Strongly Agree. This means that the proposed game is accurate. The respondents approved that the developed system was accurate based on their evaluation.

1. **user-friendliness**

Table 5.4.2Extent of user-friendliness of the developed system (n=246)

|  |  |  |
| --- | --- | --- |
| Item | Mean | Description |
| 1. The developed system has distinct features for each fruits**/**objects. | 3.62 | Strongly Agree |
| 2.The developed system has clear instructions on the following menus: |  | |
| 1. About | 3.70 | Strongly Agree |
| 1. Help | 3.67 | Strongly Agree |
| 1. Score | 3.76 | Strongly Agree |
| 3. The developed system can be easily manipulated by the player using the sword. | 3.67 | Strongly Agree |
| 4. The developed system has a good graphical user interface. | 3.88 | Strongly Agree |
| **Overall** | **3.71** | **Strongly Agree** |

*Table 5.4.2* shows extent of user-friendliness of the developed system. On one hand, item 4 which is the developed system has a good graphical user interface obtained the highest mean of 3.88. This implies that the developed system has a good graphical user interface. On the other hand, Item 1 which is the developed system has distinct features for each fruit/object got the lowest mean of 3.62, but was still interpreted as strongly agree.

The over-all mean of 3.71 described as Strongly Agree means that the developed system is user-friendly. Most of the respondents approved that the developed system was user-friendly based on their evaluation.

Table 5.4.3 Summary of Overall Results of Effectiveness of the developed system

|  |  |  |
| --- | --- | --- |
| **Description of the System** | **Mean** | **Interpretations** |
| Accuracy | 3.84 | Strongly Agree |
| User-Friendliness | 3.71 | Strongly Agree |
| Average | 3.75 | Strongly Agree |

*Table 5.4.3* shows the summary of overall results of the effectiveness of the developed system. Based on the average mean of 3.75 shown in Table 5.5.3. the respondents Strongly Agree on the effectiveness of the developed system in terms of accuracy (3.84), and user-friendliness (3.71).