**Design**

**Problem Description (provided by Professor):**

In this program you will be mimicking a game of Battleship. Your game board is a line of 10 slots. You have to implement 3 methods. The following methods must be implemented:

1. void setBoard(int \*)

This method sets up the human’s game board. The method prompts the user for 2 slots to place the “ship”. The program should ensure that the slots are consecutive. In other words, your ship cannot be placed at slot 2 and at slot 6. Since the board has only 10 slots, valid slots are in the range of 0 to 9.

1. void setComputerBoard(int \*)

Exactly the same as the setBoard method except that the position of the computer’s ship is set randomly. The computer’s ship must also be in consecutive slots.

1. int playGame(int \*, int \*)

This method should do the following steps:

1. Have the computer pick at random a slot to “fire” at. If the slot the computer picked on the human’s board is a ship, a “HIT” message is displayed. If the slot on the human’s board is empty, a “MISS” message is displayed. Print out the computer’s board and the human’s board as described at the bottom of the page.
2. After the computer goes, have the human pick a slot to “fire” at. If the slot the human picked on the computer’s board is a ship, a “HIT” message is displayed. If the slot on the computer’s board is empty, a “MISS” message is displayed. Print out the computer’s board and the human’s board as described at the bottom of the page.
3. Repeat steps 1 and 2 until there is a winner. The computer wins if both of the human’s ship slots are hit. The human wins if both of the computer’s ship slots are hit.

The playGame method returns a 0 if the computer won. The playGame method returns a 1 if the human won.

The computer board is printed off with the following characters:

S – printed off in the slots where the computer’s ship is located.

M - printed off in the slots where the human guessed incorrectly.

H - printed off in the slots where the ship is located and where the human hit.

\* - printed off in all other slots

The human board is printed off similarly.

**Identifier Dictionary:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier | Category | Data Type | Value | Description |
| GRAVITY | Constant | float | 442.2 | Gravity number used in calculating FPS |
| SPEED | Constant | int | 16 | Speed used in calculating FPS |
| NUMBER\_OF\_TRIES | Constant | int | 5 | Number of tries the user gets |
| MIN\_SPEED | Constant | int | 14 | The minimum speed your cargo falls |
| MAX\_SPEED | Constant | int | 15 | The maximum speed your cargo falls |
|  |  |  |  |  |
| weight | Variable | float | User | Weight of cargo in ounces |
| size | Variable | int | User | Size of the parachute in square feet |
|  |  |  |  |  |
| fps | Variable | float | Calculation | Rate of fall in feet per second |
|  |  |  |  |  |

Algorithm:

Main():

This program will take the user’s weight of cargo and the parachute size and calculate the rate of fall in feet per second to see if it falls correctly to the ground.

1. The user will be prompted to enter a value for the weight of the cargo in ounces.
   1. There will be an error check here to make sure the value is positive.
   2. If the number is not positive, the user will try again.
2. The value will be stored as variable “weight”.
3. The number of tries will be displayed on the next line -(try attempts 1 through 5).
4. The user will be prompted to enter a value for the size of the parachute in square feet.
   1. There will be an error check here to make sure the value is positive.
   2. If the number is not positive, the user will try again.
5. The value will be stored as variable “size”.
6. The rate of fall in feet per second will be calculated using the user’s weight and size.
7. The value will be stored as variable “fps”.
8. FPS will be displayed as text: “Falling at \_\_\_\_\_\_\_ per feet/sec”.
9. If the FPS is between 14 and 15, it will display that the descent is perfect.
10. If the FPS is higher than 15, the cargo will “SPLAT” and the user will try again.
11. If the FPS is lower than 14, the cargo will “DRIFT” and the user will try again.
12. The program will stop after 5 attempts.

Functions():

* GetCargo: Determines whether the number of cargo is a positive number
  + Returns a positive number for cargo
  + Returns an error message to input a positive number
* GetChuteSize: Determines whether the number for the parachute size is a positive number
  + Returns a positive number for the parachute size
  + Returns an error message to input a positive number
  + The parameter passed to the function is the “try number” which is between 1 and 5.
* ComputeFPS: Determines the rate of descent
  + Returns the speed.
  + The parameter passed to it are the weight of the cargo and the size of the parachute
  + Formula: rate of fall in feet per second is fps = (442.2 \* sqrt (X / 16 )) / Y;
    - Where X is the weight of the cargo in ounces and Y is the size of the parachute in square feet.

Work Log

|  |  |  |
| --- | --- | --- |
| Date: | Time: | Task: |
| 10/8/2015 | 1 hour | Coding |
| 10/11/2015 | 2 hours | Design, Algorithm & Coding |
| 10/12/2015 | 2 hours | Design, Coding, Debugging |
| 10/15/2015 | 2 hours | Debugging and Summary |

Summary

This program was pretty straight forward. I had some trouble getting the “number of tries” to print correctly with a for-loop so I changed it to a while-loop. It was still printing twice. I had an infinite loop for the parachute size. I also added more constant variables to make the code easier to read. Overall, I really do not like working in Gargamel but this is how it must be.

You get 5 chances to land your cargo safely. Good Luck!

Enter the weight of cargo in ounces, please: 34

Try: 1

Enter the parachute size in square feet, please: 0

Enter a positive number for the parachute size in square feet, please: 23

Falling at 28.027

SPLAT! We're going to need a bigger chute. Please try again.

Try: 2

Enter the parachute size in square feet, please: 40

Falling at 16.115

SPLAT! We're going to need a bigger chute. Please try again.

Try: 3

Enter the parachute size in square feet, please: 50

Falling at 12.892

DRIFTING AWAY! The parachute is too big. Please try again

Try: 4

Enter the parachute size in square feet, please: 45

Falling at 14.325

NICE LANDING!!! You won the game!

12:04:12:jcapaz@MathCSProd01:~/cs3335/HW3$

You get 5 chances to land your cargo safely. Good Luck!

Enter the weight of cargo in ounces, please: 120

Try: 1

Enter the parachute size in square feet, please: 90

Falling at 13.456

DRIFTING AWAY! The parachute is too big. Please try again

Try: 2

Enter the parachute size in square feet, please: 80

Falling at 15.138

SPLAT! We're going to need a bigger chute. Please try again.

Try: 3

Enter the parachute size in square feet, please: -85

Enter a positive number for the parachute size in square feet, please: 85

Falling at 14.247

NICE LANDING!!! You won the game!

12:07:30:jcapaz@MathCSProd01:~/cs3335/HW3$

You get 5 chances to land your cargo safely. Good Luck!

Enter the weight of cargo in ounces, please: 1000

Try: 1

Enter the parachute size in square feet, please: 100

Falling at 34.959

SPLAT! We're going to need a bigger chute. Please try again.

Try: 2

Enter the parachute size in square feet, please: 0

Enter a positive number for the parachute size in square feet, please: 300

Falling at 11.653

DRIFTING AWAY! The parachute is too big. Please try again

Try: 3

Enter the parachute size in square feet, please: 250

Falling at 13.984

DRIFTING AWAY! The parachute is too big. Please try again

Try: 4

Enter the parachute size in square feet, please: 240

Falling at 14.566

NICE LANDING!!! You won the game!

12:08:25:jcapaz@MathCSProd01:~/cs3335/HW3$