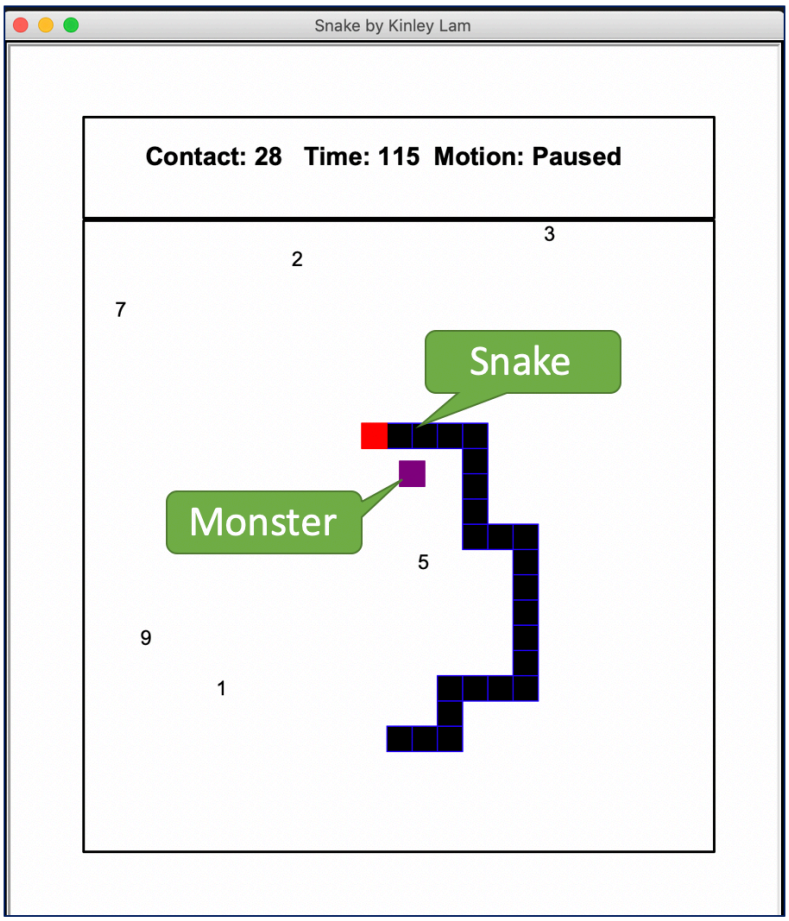
Snake – 2021

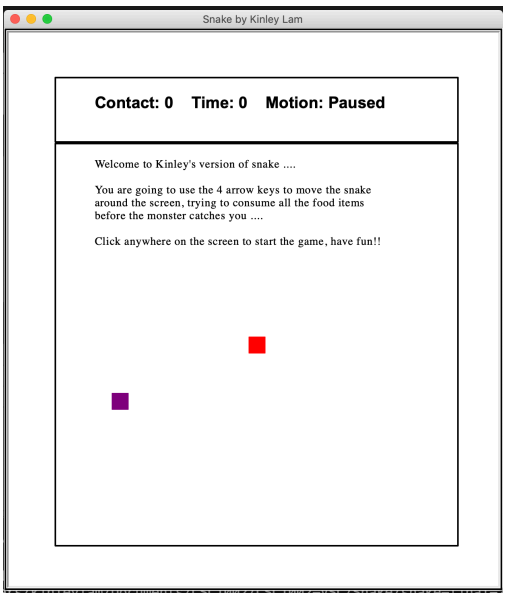


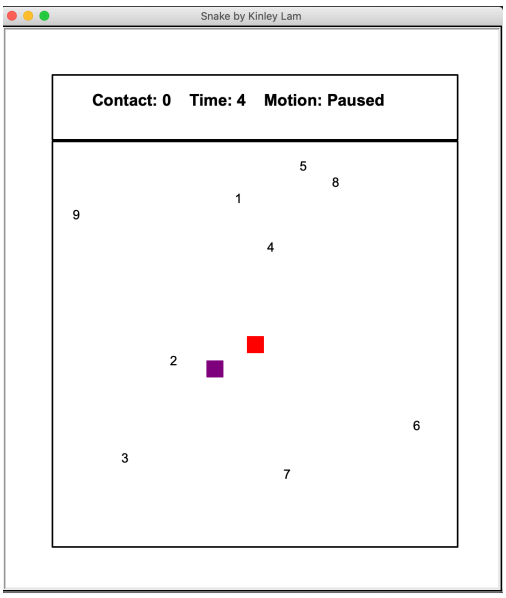
OVERVIEW

In this assignment, you are going to design and develop a Snake game. The game is composed of 3 objects: a snake, a monster and food items represented by a set of numbers from 1 to 9. In the figure shown above, the snake is represented by a sequence of squares where its head and its body are displayed in red and black colors respectively, while the monster by a purple square. The numbers are food items to be consumed by the snake.

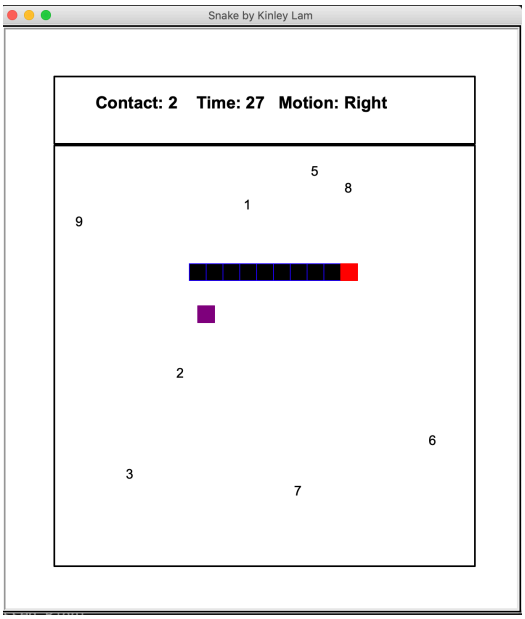
The goal of the game is to maneuver the snake within the game area in four directions (up, down, left and right), trying to consume all the food items while avoiding head-on collision with the monster. As each food item is consumed, the snake grows with its body lengthened in size equal to the value of the number being passed. While directing the movement of the snake you should avoid contact with the monster. Furthermore the monster is also programmed to be motioned in the direction towards the head of the snake at a variable speed.

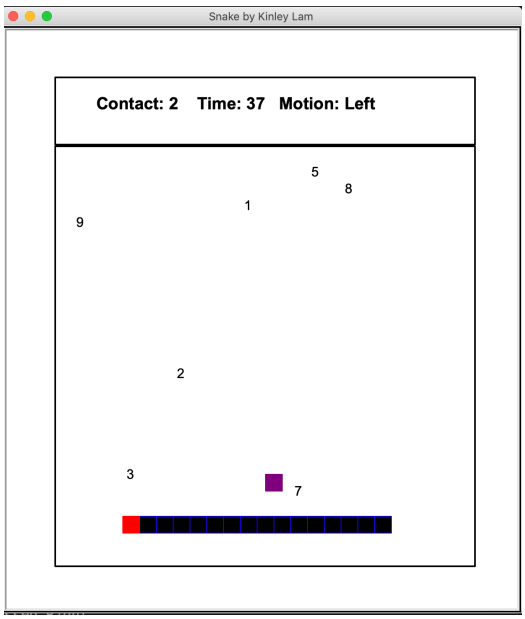
The following screens show the first display of the game (left) and the start of the game (right) after a mouse click:



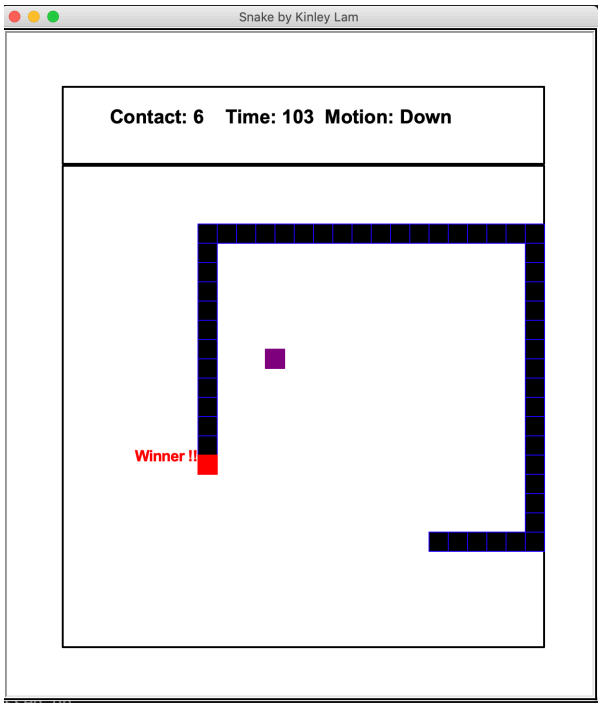


The following 2 screens show the snake with its extended tail after consumption of food items (4 & 6), while the monster actively chasing after the snake:

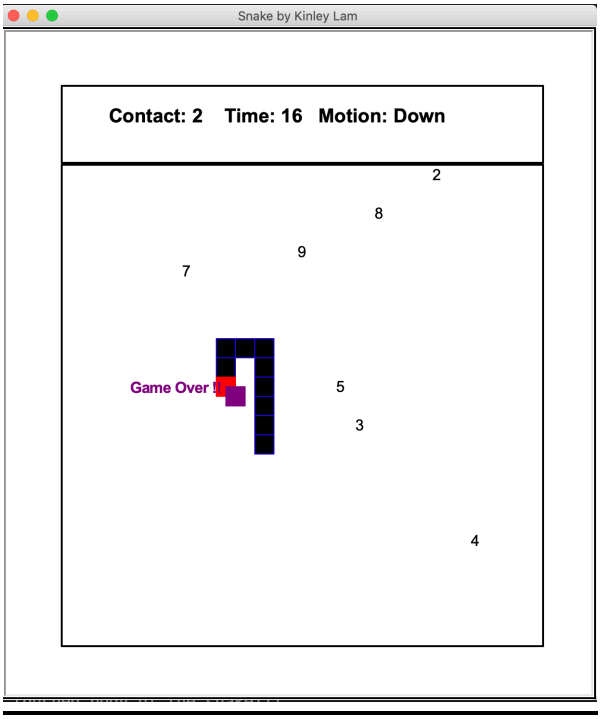




The following screen shows the ending of the game after the snake has consumed all the food; shown on the status area are the number of contacts with the monster and the total elapsed game time in seconds:

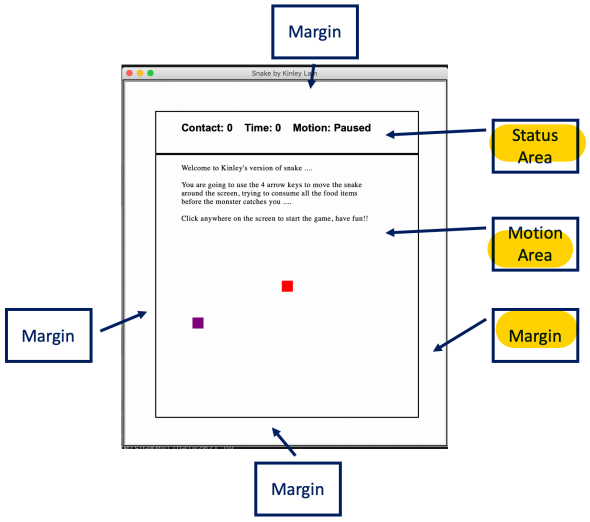


The following screen shows the ending of the game where the snake collided head-on with the monster and where some food items left unconsumed:



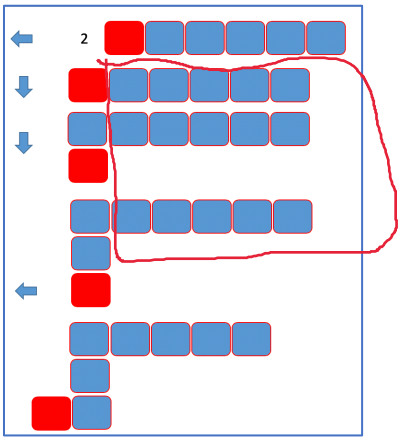
SCOPE

- Design the snake game using the standard module "turtle", including the following components:
  a. Game Area (status and motion area)
  b. A Snake
  c. A Monster
  d. Food Items
  e. Game Status
  f. Controls
  g. Motion
- Game Area



  a. The game area is composed of an upper area for statuses and a motion area where the snake and monster are moved around, surrounded by a fixed margin along the four sides, with the following dimensions:
     i. Upper status area = 500 (w) x 80 (h)
     ii. Lower motion area = 500 (w) x 500 (w)
     iii. Margins = 80 pixels around
  b. Draw a border for both the status area and motion area.
- Food Items
  a. Represented as numbers from 1 to 9, displayed within the motion area in random locations. These numbers will be kept visible all time until they are consumed by the snake. When the head of the snake crosses one of these numbers, the number being crossed is considered consumed and it will be removed from the game area permanently. So, any one food item can be consumed once.
- Snake
  a. The snake is composed of a head with a tail which extends as it consumes any food items.
  b. Use only simple, built-in shape "square" for both head and tail, default size.

c. Use different colors for the head (ex: red) and the tail (ex: black with blue border color); choose a border color for the tail so that the length of the tail can be counted easily.
d. The tail extends as the snake moves, not at the point when the food item being consumed; in other words, at the moment the snake crosses a food item, the snake doesn't change in size; as the snake moves the tail extends in the direction of the movement as if the end of the tail sticks onto the screen. The tail extension ends when the length of the snake has grown in size equal to the value of the number being crossed. The following figure shows the sequence of moves of the snake crossing a food item.

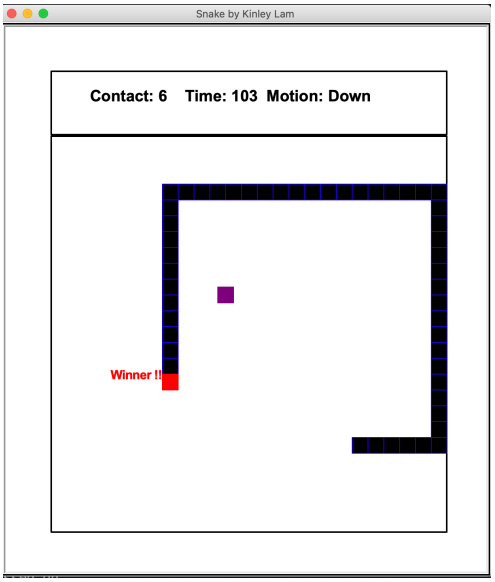


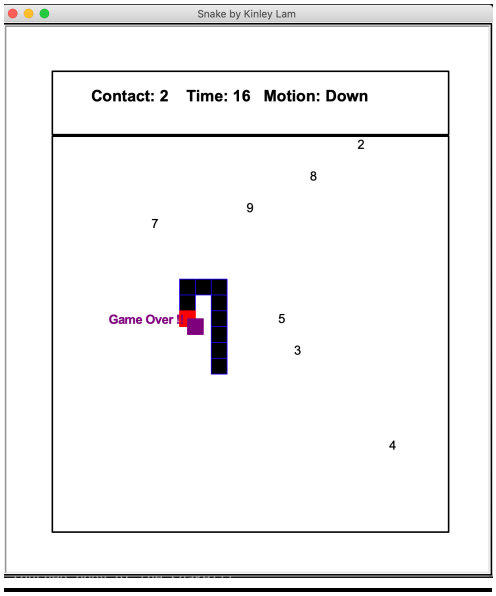
e. As the tail is being extended, the movement of the snake will slow down. See "Timer" below.
f. At the start of the game, the length of the tail is set to 5. One square shape counts as one unit length, so the tail will be composed of 5 square shapes when fully extended.

- Monster
  a. A fixed size object to be programmed to move towards the snake, trying to make a head-on collision.
  b. On startup, place the monster on a random position with a fair distance from the snake.
  c. The monster should move at a random rate, a rate that is slightly faster or slower than that of the snake. See "Timer" below.
  d. Use only simple, built-in shape "square" for the monster, default size.
  e. Use a different color such as purple.
- Game Status
  a. Show the motion of the snake (Left, Right, Up, Down, Paused), in other words, the last motion key pressed (including the space bar), regardless of whether the snake is in motion or being blocked.
  b. Show the total count of body contact of the snake with the monster. The count should be based on the motion of the monster timer. Each time the monster is re-positioned, it should then check if it overlaps with any part of the snake.
  c. Keep track of the total elapsed game time in seconds. The time counter starts as soon as the game starts and will stop only when game is over. In other words, the counter will not be stopped when the snake is being paused.

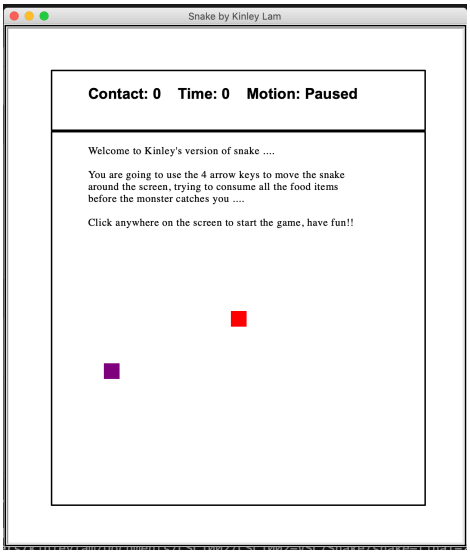
- Motion via Timer
    a. "Timer" controls the frequency that a specific event to take place at a regular interval, in this case the event is the movement of either the snake or the monster.
    b. Use separate timers to manually refresh the movement of both snake and the monster
        i. Turn off the built-in automatic screen refresh
    c. Keep the timer rate no faster than 0.2 second.
    d. On each timer event, always advance the snake or the monster in a distance equivalent to the length of the turtle shape (square). If the square's dimension is 20x20 (pixels), then your logic should advance the turtle object 20 pixels at a time.
    e. Both snake and monster move in four directions, left, right, up or down, NOT diagonally.
    f. Design the timers in such a way:
        i. the monster should move in a random time range slightly above or lower than that of the snake, while snake always move at a fixed rate.
        ii. when the snake crosses a food item (a number) slow down its movement by increasing its timer rate until its tail is fully extended, that is, the snake will motion slower while the tail being extended
        iii. furthermore, you don't want the snake to move too fast that the monster will never catch up with the snake, or vice versa. That is, you don't want the monster moves so quickly that it always catch the snake before it has a chance to consume all the food items.
- Controls
    a. Use the four arrow keys (Up, Down, Left, Right) to maneuver the snake in Up, Down, Left and Right motion respectively.
    b. The motion will continue in the direction of the last arrow key pressed. For an example, If Left key is pressed, the snake will continuously move in the left direction until a different arrow key is pressed.
    c. Use "Space Bar" to toggle (pause and un-pause) snake motion (note: monster never pause). While in motion, pressing the spacebar will pause the snake (not the monster). While paused, pressing the space bar the snake will resume motion in the direction of the last arrow key pressed. Furthermore, while paused, pressing any of the four arrow keys will un-pause the snake motion and move in the direction of the arrow key being pressed.
    d. Snake cannot move beyond the motion area; when the head of the snake is moved against any of the 4 sides, the snake will be stopped and it will remain blocked until its heading is changed away from the edges.
    e. Snake can cross its body, left to right, up to down or vice versa.

- Game Termination
    a. The game ends when the snake consumes all the food items and its body is fully extended, in this case “Winner” or the monster caught the snake, in this case “Game Over”.



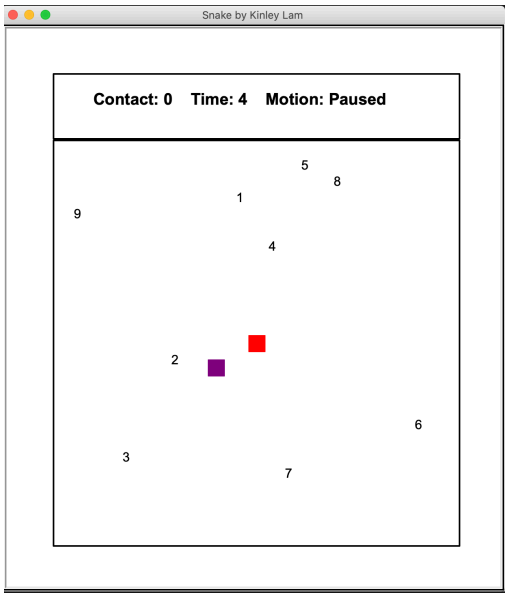


- Game Startup
    a. On startup, show (1) the game area with a brief introduction to describe the game objective and control, (2) the snake (in red) positioned at center and (3) the monster (purple) at a random position far enough from the snake.



;

b. User mouse-click anywhere on the screen to start the game, all the food items will be shown subsequently, user then moves the snake around using the 4 arrow keys



- Coding Styles
  a. Ensure that your program follows the proper layout structure as discussed in class.
  b. You might declare many global variables used for this assignment, ensure that a consistent naming convention is in place to differentiate various variable scopes.

NOTE:

- Keep your entire source code in ONE SINGLE file.
- Use only standard python modules
- In your design stick ONLY to functions, in other words, no class objects of your own.

STARTUP OPTIONS

Not applicable

SKILLS

In this assignment, you will be trained on the use of the followings:

- Use built-in turtle module to design the snake program as per scope
- Use standard objects (strings, numbers & lists)
- GUI interaction
- Variable Scope
- Functions for program structure and decomposition

DELIVERABLES

1. Design documentation (A1_School_StudentID_Design.doc/pdf)
2. Program source code (A1_School_StudentID_Source.py)

where School is SSE, SME, HSS, FE or LHS and StudentID is your 9-digit student ID.

Zip all files above in a single file (A1_School_StudentID.zip) and submit the zip file by due date to the corresponding assignment folder under “Assignment (submission)”

For instances, a SME student with student ID “119010001”:

- A2_SME_119010001.zip:
    - A2_SME_119010001_Design.doc/pdf
    - A2_SME_119010001_Source.py

5% will be deducted if any files are incorrectly named!!!

For the design document kindly refer to section “Design Documentation” for details.

DESIGN DOCUMENTATION

For the design document provide write-up for the following sections:

1. Design
    a. Overview
    b. Data Model
        i. describe core data objects called Data Model (such as list, string, dictionary and so on) that you used to develop your program for the following items:
            1. snake including the tail
            2. food items
    c. Program Structure (your thoughts and overall approach)
        i. describe the breakdown of your logic into various functions and how these functions are organized (basically the structure of your program)
    d. Processing Logic
        i. Describe the logic used to motion the snake and monster.
        ii. Describe the logic used to expand the snake tail
        iii. Describe the logic used to detect body contact between the snake and the monster
2. Function Specifications
    a. Describe usage of all your own defined functions, including details of parameter(s) and output if any.
3. Output
    a. Show samples of output (including status) from your program, including
        i. Winner
        ii. Game over
        iii. 2 others showing various stages of the game:
            1. With 0 food item consumed
            2. With 3 food items consumed

TIPS & HINTS

- Follow the layout structure as mentioned in class (import, declarations, functions, main process).
- Clearly name and comment your global variables.
- Refer to python website for program styles and naming convention (PEP 8).
- Use Turtle() as objects for the snake, monster and all food items.
- Use the shape() function to set the shape for your objects, or pass the shape as string to the Turtle(). Note: use simple, built-in shape such as "square" for your snake and monster objects. Complex shapes will slow down the screen refresh!!!!
- Use stamp() to make copy of the turtle shape at its current position; use clearstamp(idx) to remove a specific stamp copy or clearstamps() to clear one or more stamp copies.
- Use "stampItems" of the Turtle object to return the list of turtle stamps
- Use write() to display a text on the screen.
- Remember to set the pen in "up" position to avoid line drawing.
- Use ontimer() to motion your snake and monster.
- Use Screen() to configure the game area and use onclick() to capture mouse-click event.
- Use tracer(0) to disable auto screen refresh and call update() to manually refresh the game area.
- Refer to https://docs.python.org/3/library/turtle.html for more information on Turtle Graphics.

SAMPLE OUTPUT

Refer to the Overview session.

MARKING CRITERIA

- Coding Styles – overall program structure including layout, comments, white spaces, naming convention, variables, indentation, functions with appropriate parameters and return.
- Design Documentation
- Program Correctness – whether or the program works 100% as per Scope.
- User Interaction – how informative and accurate information is exchanged between your program and the player.
- Readability counts – programs that are well structured and easy-to-follow using functions to breakdown complex problems into smaller cleaner generalized functions are preferred over a function embracing a complex logic with nested conditions and sub-functions! In other words, a design with clean architecture with high readability is the predilection for the course objectives over efficiency.
- KISS approach – Keep It Simple and Straightforward.
- Balance approach – you are not required to come up with a very optimized solution. However, take a balance between readability and efficiency with good use of program constructs.

ITEMS	PERCENTAGE	REMARKS
DESIGN DOC	10%-15%	
CODING STYLES	20%-25%	0% IF PROGRAM DOESN'T RUN
USER INTERFACE	15%-20%	0% IF PROGRAM DOESN'T RUN
FUNCTIONALITY	>40%	REFER TO SCOPE

DUE DATE

April 25th, 2021, 11:59:59PM

APPENDIX - TEMPLATE

Design Doc

OVERVIEW

- Write one or two paragraphs to describe in high-level what your program does. See section Overview.

DATA MODEL

- Describe the type(s) of data that you used to model your core objects in your program. In this case, it will be the snake, food items and monster, plus any other objects that you think of essential to mention.

PROGRAM STRUCTURE

- Describe the structure of your program, specifically how you organized your thoughts in terms of functions and how these functions are organized. In general, you might breakdown your logic into many functions organized by functional components, each of which performs specific role.
- For each component, describe the role and list out the function(s) in high level description, leave the details to be included in section Functional Specifications.

PROCESSING LOGIC (SPECIFIC)

- Main processing logic - describe how you piece together various components and data model to implement your program for the following items:
    - the logic used to motion the snake and monster.
    - the logic used to expand the snake tail
    - the logic used to detect body contact between the snake and the monster

FUNCTIONAL SPEC

- Describe usage of all of your own defined functions including an overview description, detailed description of parameters, as well as output if any.

SAMPLE OUTPUT

- Include a few samples of outputs from your program.

OTHER IDEAS:

If you feel that you have the passion for programming, here's list of other ideas that you could further enhance your program by refactoring your existing logic.

1. Implement multiple monsters; start with one monster initially and then another one on every fixed time interval, up to some numbers, say 3 (be cautious not to have too many monsters suddenly). You need to ensure that use a different timer logic to for each monster chasing after the snake. Otherwise all monsters will merge into each other eventually and moved altogether.
2. Randomly hide and unhide any food items.
3. Randomly shift any food items; shift food items in random direction (left, right, up or down), not too much a shift, yet enough to confuse the player
4. Do not allow the snake to cross its body; block its movement
5. Implement surprise items; you can have reward or penalty items on screen for snake to consume:
    a. Reward: Increase game time limit (20 seconds)
    b. Reward: Freeze motion of all monsters for a few seconds (10 seconds)
    c. Penalty: generate another food item
    d. Penalty: Hide all food items for a few seconds (10 seconds)
6. Add sound effect (motion, body contact, food consumption, surprise items and so on).
7. Set aside the top portion of the game area for better status display (with larger text size and different colors)
8. Set a time limit; player must finish the game within the time limit
9. Add prompt for a new game

Further challenges: After you submit your assignment, leave the program alone and wait until the end of the school year, around end July, then start the development. You will see the value of refactoring and decomposition. Share me your experiences on your development.

NOTE:

- Please DO NOT IMPLEMENT ANY OF THESE FEATURES with your assignment, do it separately.
- No extra credit will be given
- Remember to show your game objectives, or include a readme.txt file
- Share your program to me by email if you want
- If you come up with other ideas share them with me as well
- Enjoy programming