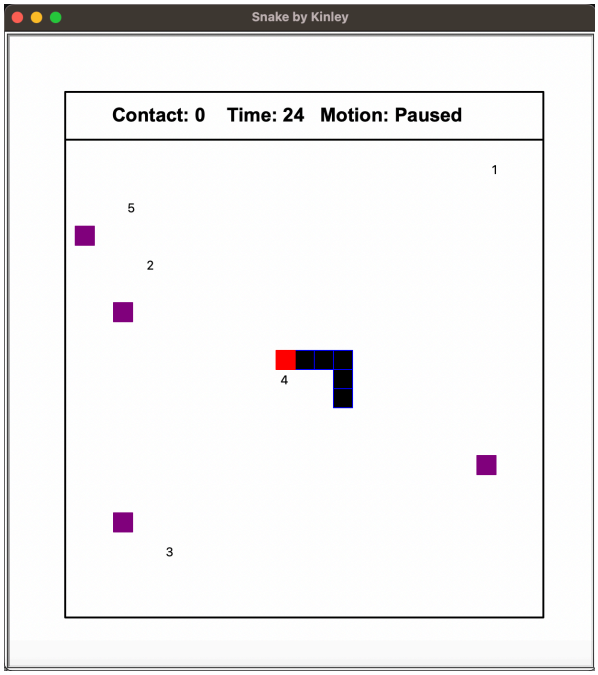
Snake – 2024



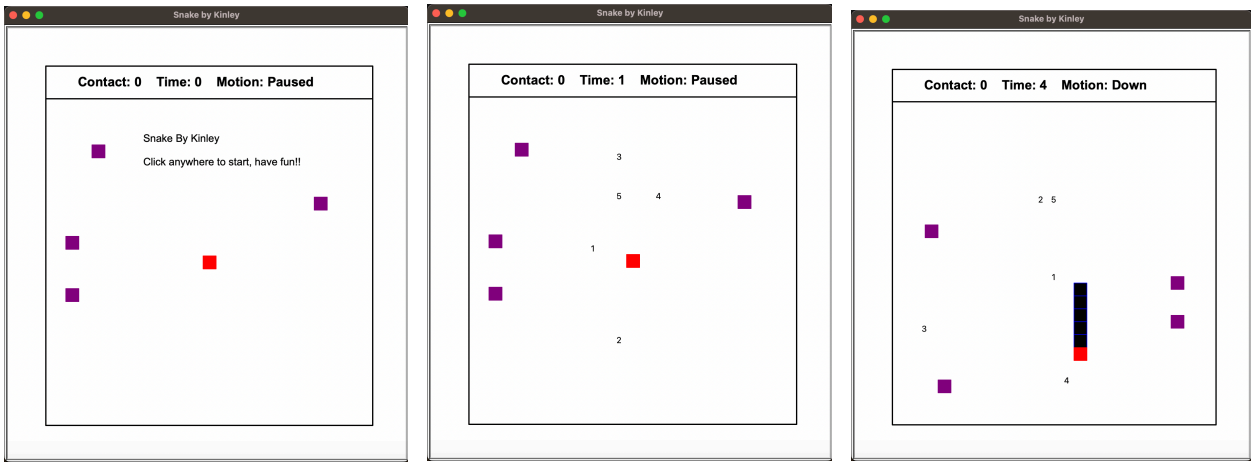
OVERVIEW

In this assignment, you will design and create a Snake game. The game consists of three elements: a snake, a few monsters, and a few food items represented by numbers between 1 and 9. In the daigram shown above, the snake is depicted as a series of squares with its head in red and its tail in black, while the monsters are shown as purple squares. The numbers represent the food items that the snake needs to consume.

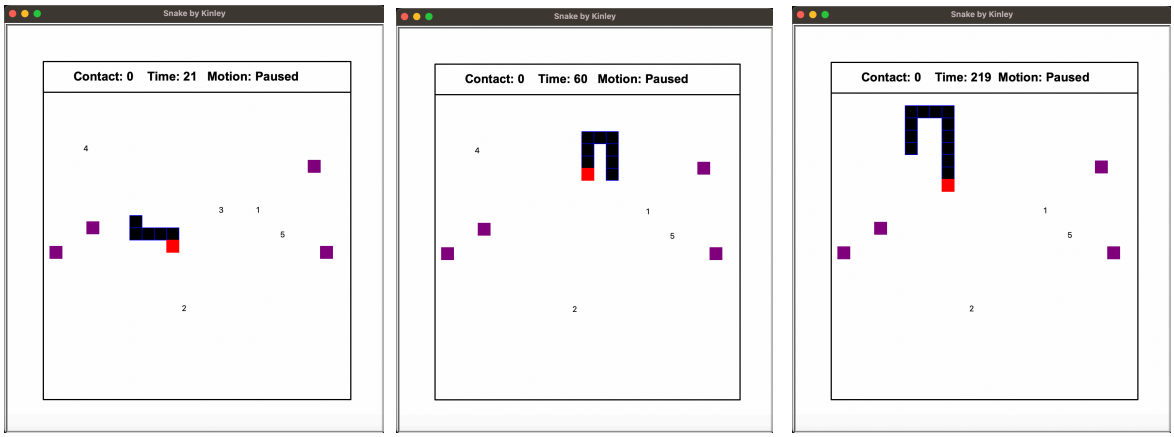
The goal of the game is to maneuver the snake within the game area in four directions (up, down, left and right), aiming to consume all the food items while avoiding a head-on collision with the monster. Each time a food item is consumed, the tail of the snake grows in length by the value of the number crossed. While directing the movement of the snake, you should avoid coming into contact with any monsters. Furthermore, the monster is also programmed to move towards the snake's head at a variable speed.

Attention: Refer to the section Scope for specific requirements pertaining to this assignment, such as the quantity and the values of the food items, the expected number of monsters to be deployed on the game, and so on. All the diagrams shown throughout this document are used for illustration purposes, thus the exact color, the size, and the number of food items and monsters might not match 100% as specified in the scope section.

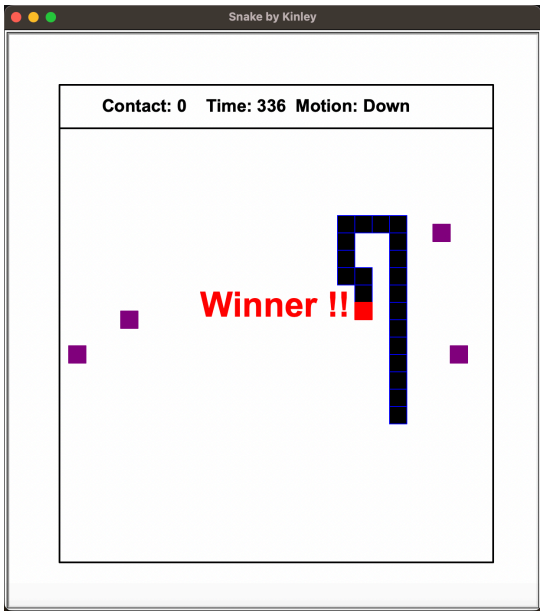
The following diagrams show the initial interface of the game (left), the start of the game (middle) after a mouse click, and the snake (right) with its tail after movement.



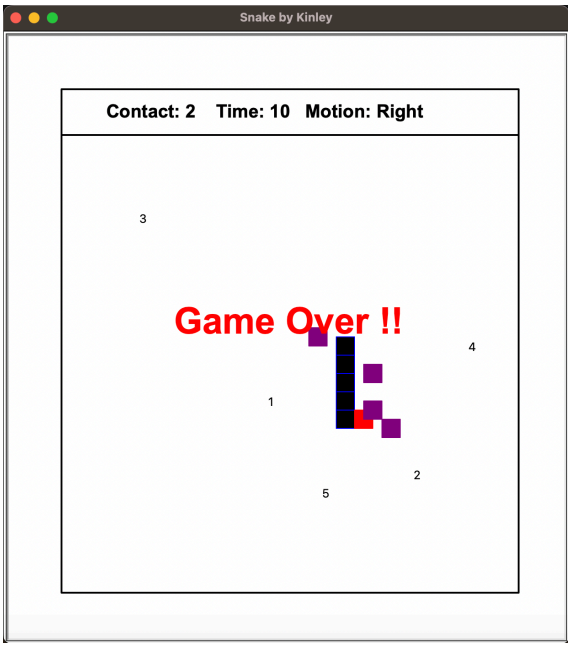
The diagrams below display the initial size of the snake (left diagram) and its fully extended tail after consuming two food items: 3 (middle) and 4 (right). The middle diagram illustrates the complete length of the snake's tail following the consumption of food item 3, while the right diagram depicts the situation after consuming food item 4.



The following diagram displays the ending of the game once the snake has consumed all the food, including its fully extended tail. The status area shows the number of encounters with the monster and the total game time in seconds:



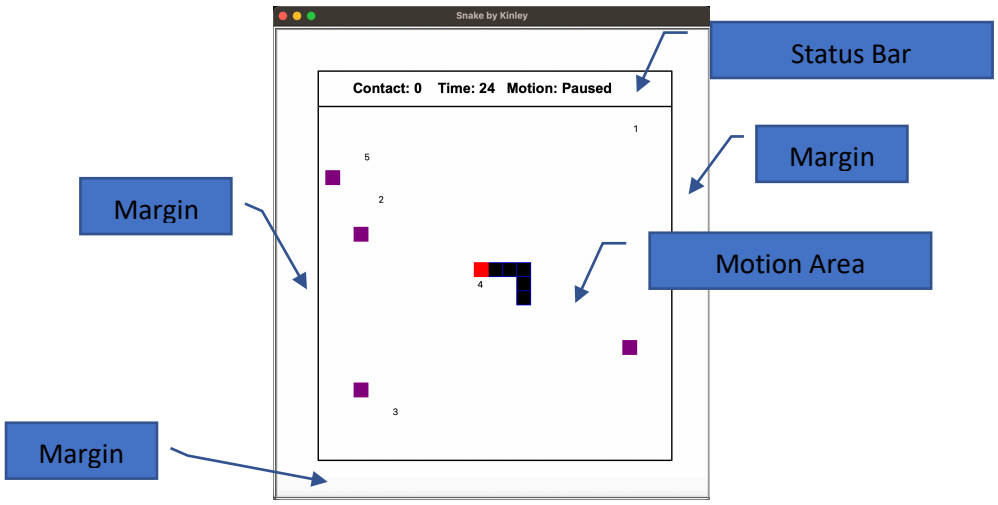
The following diagram shows another ending of the game, depicting the snake colliding directly with the monster while some food items remain unconsumed:



SCOPE

- Design the snake game using the standard module “turtle”, including the following components:
    a. Game Area (status, motion area and margin)
    b. Food Items
    c. Snake
    d. Monsters
    e. Game Status
    f. Motion vis Timer (Snake & Monsters)
    g. Controls
    h. Game Termination
    i. Game Startup

- Game Area



    a. The game area is composed of a status bar in the upper area and a motion area where the snake and monster(s) are moved around, surrounded by a fixed margin along the four sides, with the following dimensions in pixels:
        i. Upper status area = around 500 (w) x 60 (h)
        ii. Lower motion area = around 500 (w) x 500 (w)
        iii. Margins = around 30 pixels
    b. Draw a border for both the status area and motion area.

    Note: the dimension of the motion area is chosen in multiple of standard turtle shape (default 20 pixels)

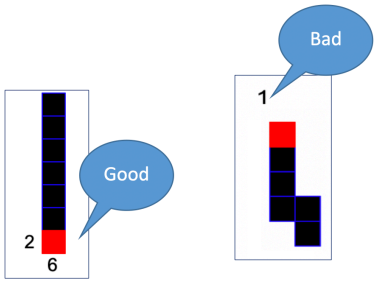
- Food Items

    c. Display 5 food items from 1 to 5 (1,2,3,4 & 5) within the motion area in random locations. These numbers will be kept visible all the 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.

    d. During the game, randomly shift the food items by an appropriate distance, say 40 pixels. Use an appropriate random timer rate, say a few seconds between 5 and 10, then randomly pick one or more unconsumed food items, and then shift the positions in any left, right, up or down directions.

    e. Font Size – choose an appropriate font size such that the number is fully enclosed within the snake's head such as size 18, Arial font in bold.

    f. Alignment – place the food items closely aligned to the snake's head, see diagram below.



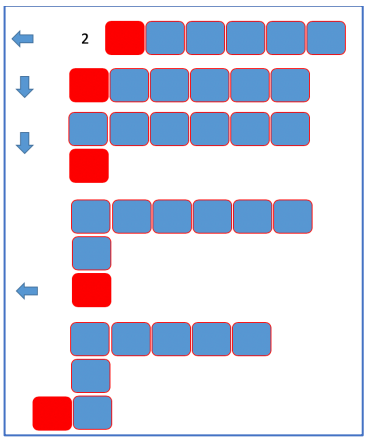
- Snake

    a. The snake is composed of a head with a tail which extends as the snake consumes any food items.

    b. Use only a simple, built-in shape "square" for both head and tail, and the default size.

    c. Use different colors for the head (ex: red) and the tail (ex: black with blue outline color); choose outline 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 is being consumed; in other words, at the moment the snake crosses a food item, the snake's tail doesn't change; as the snake moves the tail extends in the direction of the movement. 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 diagram shows the sequence of moves of the snake crossing a food item.

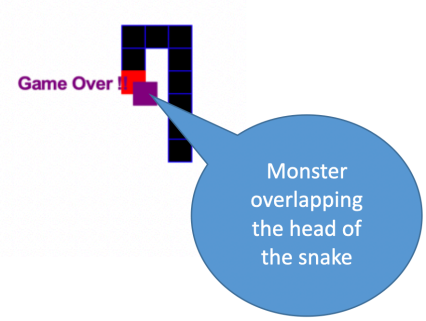


e. As the tail is being extended, the movement of the snake will slow down. See "Timer" below.

f. At the start of the game, the length of the tail is set to 5. One square shape counts as one unit of length, so the tail will be composed of 5 square shapes when fully extended.

- Monsters

a. Deploy 4 monsters at the start of the game in random locations within the motion area, each with a fair distance from the snake's head. The locations should be chosen such that the snake's head should overlap the shape of the monster when collided, see the diagram below.



b. Each monster is a square-shaped turtle object programmed to move toward the snake, trying to make a head-on collision.

c. The monsters 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 a simple, built-in shape "square" for the monster, with the default size of 20 pixels.

e. Choose a different color such as purple.

- Game Status

a. Show the motion key last pressed (Left, Right, Up, Down, or Paused).

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 the game is over. In other words, the counter will not be stopped when the snake is paused by the spacebar or stopped when trying to go beyond the boundaries.

- Motion via Timer

a. "Timer" controls the frequency that a specific event takes 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 the snake and the monster

i. Turn off the built-in automatic screen refresh, if needed.

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 the snake always moves at a constant 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 is 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 monsters to move so quickly that they always catch the snake's head 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 example, If the Left key is pressed, the snake will continuously move in the left direction until a different arrow key is pressed or the spacebar is pressed.

c. Use "Space Bar" to toggle (pause and un-pause) snake motion (note: monsters never pause). While in motion, pressing the spacebar will pause the snake (not the monster). While paused,

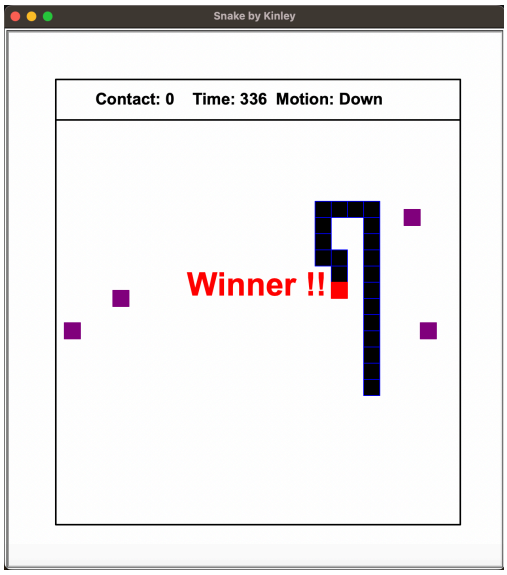
pressing the space bar the snake will resume motion in the direction of the last arrow key pressed. Furthermore, while paused, pressing any of the four arrow keys will un-pause the snake motion and move in the direction of the arrow key being pressed.

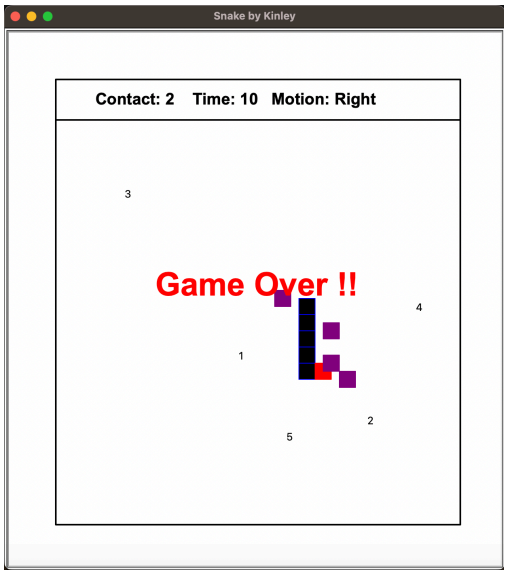
d. Blocking: The snake is unable to move beyond the boundary of the playing area; if the snake's head is directed towards any of the 4 sides, it will encounter a blockage and will stay blocked until its direction is altered away from the obstructed side.

e. The snake can move across its body from left to right, up to down, or vice versa. For instance, if the snake is currently moving to the right, pressing the left key will make it move to the left, resulting in its head (red) moving over its tail.

- Game Termination

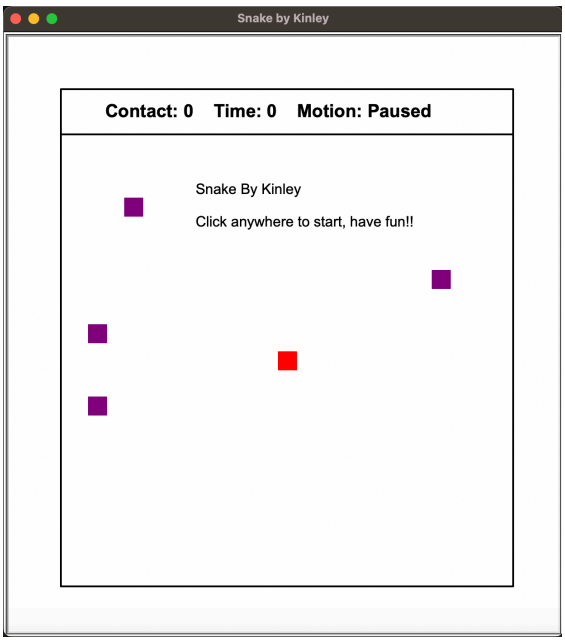
a. The game ends when the snake consumes all the food items and its body is fully extended. In this scenario, a brief message “Winner !!” will be displayed. Alternatively, if one of the monsters catches the snake, a short text "Game Over !!” will appear. Position the text in the center of the motion area as depicted in the diagrams:





- Game Startup

b. On startup, show (1) the game area with a short message “Click anywhere to start, have fun!!!!”, (2) the snake (in red) positioned at the center of the motion area and (3) position the monsters (purple) at random positions far enough from the snake’s head.



c. User mouse-clicks anywhere on the screen to start the game, all the food items will be shown subsequently, the 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, and 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 python modules as specified in the “Permitted Modules” section.
- In your design stick ONLY to functions, in other words, no class objects of your own.
    - Furthermore, the lines of code containing the sub-function(s) defined within another function will be counted as part of the parent function.

STARTUP OPTIONS

Refer to Scope.

SKILLS

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

- Problem Decomposition, Clean Code, Top-Down Design based on functional specification
- Functions (with parameters and return) for program structure and logic decomposition
- Standard objects (strings, numbers & lists)
- Variable Scope (Global vs Local, Constants)
- GUI-based programming: mouse click event, callback, asynchronous-style programming
- Turtle Graphics - a basic graphics drawing tool as the building block for creating the visual representation of the snake game.

PERMITTED MODULES

Only the following python modules are allowed to be used:

- random, turtle, time, functools.

DELIVERABLES

1. Program source code (A3_School_StudentID.py)

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

Submit the plain (do not compress your file) python file by due date to the corresponding assignment folder under "Assignment (submission).

For instances, a SME student with student ID "119010001" will name the python file as follows:

- A3_SME_119010001.py:

5% will be deducted if file is incorrectly named!!!

TIPS & HINTS

- Follow the layout structure as mentioned in class (import, declarations, functions, main process).
- Be consistent with naming convention of variable scope
- Refer to python website for program styles and naming convention (PEP 8).
- Use Turtle() as objects for the snake, monster and all food items. Remember to set the pen in "up" position to avoid line drawing.
- 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 write() to display a text on the screen via turtle object.
- Use Screen() to configure the overall size of the game area
- Use tracer(0) to disable auto screen refresh and call update() to manually refresh the game area.
- Use stamp() to copy image 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 ontimer() to separately motion your snake and monster.
- Use distance() to determine if two squares are overlapped.
- Use towards() to determine the angle between two turtle objects.
- Refer to https://docs.python.org/3/library/turtle.html for more information on Turtle Graphics.

SAMPLE OUTPUT

Refer to the Overview section.

MARKING CRITERIA

- Coding Styles – overall program structure including layout, comments, white spaces, naming convention, variables, indentation, functions with appropriate parameters and return.
- Design Documentation if required
- 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. The logic in each function should be kept simple and short, and it should be designed to perform a single task and be generalized with parameters as needed.
- KISS approach – Keep It Simple and Straightforward.
- Balance approach – you are not required to come up a very optimized solution. However, take a balance between readability and efficiency with good use of program constructs.

ITEMS	PERCENTAGE	REMARKS
CODING STYLES	20%	0% IF PROGRAM DOESN'T RUN
USER INTERFACE	20%	0% IF PROGRAM DOESN'T RUN
FUNCTIONALITY	60%	REFER TO SCOPE

DUE DATE

April 28th, 2024, 11:59:59PM

OTHER IDEAS:

Please DO NOT IMPLEMENT ANY OF THESE FEATURES with your assignment except for those items mentioned in the Scope section.

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. Deploy multiple monsters one by one; start with one monster initially and then a new one on every fixed time interval.
2. Set a game time limit; player must finish the game within the set time limit
3. Randomly hide and unhide any food items on random time interval.
4. Deploy additional food items based on game time.
5. 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
6. Do not allow the snake to cross its body; block its movement
7. 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)
8. Add sound effect (motion, body contact, food consumption, surprise items and so on).

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:

- 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 come up with other ideas share them with me as well
- Enjoy programming