Snake Documentation

Jak Waterfall

N0906911

Contents

[Contents 2](#_Toc57657633)

[Aims 3](#_Toc57657634)

[Analysis 3](#_Toc57657635)

[Menu 3](#_Toc57657636)

[Snake 5](#_Toc57657637)

[Snake Class 5](#_Toc57657638)

[Food Class 6](#_Toc57657639)

[Element Class 6](#_Toc57657640)

[Design 7](#_Toc57657641)

[Menu.py Flow Chart 7](#_Toc57657642)

[Snake.py and Input Flow Charts 8](#_Toc57657643)

[Class Diagram 10](#_Toc57657644)

[Testing 11](#_Toc57657645)

[Critique 12](#_Toc57657646)

Aims

* Made for entertainment purposes.
* To build upon the classic game of snake from mobile.
* To store all your high scores for the game.
* To present 3 different difficulty mode based on width of the snake.

Analysis

My project is based on the old mobile game snake made famous on the Nokia 3310. I am making this game with the purpose to entertain with a new spin on an already established game. My game is made up of two files; the snake file will hold all the code for the snake game and the menu file will hold all the code for the menu. The menu file is the entrance into the application.

Menu

* Imports tkinter to use for GUI.
* Imports pickle for high score file system.
* Imports snake to play game.
* Instantiates all the variables and tkinter objects for the menu screen.
* startGame()
  + Starts snake game.
  + When snake game ends the score is passed to the showAddScoresScreen() function.
* showAddScoresScreen()
  + Shows the add score screen and widgets.
  + Allows the user to input their name.
  + Adds their name and score to the High Scores using addScore() function.
* addScore()
  + Checks name is legal.
  + Gets 3 score lists (small, normal, large) using getScoresFromFile() function.
  + Attaches or appends new score to correct list.
  + Adds scores back to file using appendScoresToFile() function.
  + Returns back to title screen using backToTitleScreen() function.
* getScoresFromFile ()
  + Opens file “highscores.dat”.
  + If no file present it creates it and appends 3 Null elements.
  + Returns 3 lists of high scores in the corresponding sizes (small, normal, large).
* appendScoresToFile()
  + Reverse sorts each score list to go from highest to lowest.
  + Times each list down so that the scores do not overflow the text widget they are displayed on.
  + Writes new score lists to file.
* backToTitleScreen()
  + Brings back the title screen and uses a key to determine which screen widgets to remove.
  + If the key “high” is given the high score screen widget will be removed.
  + If the key “add” is given the add score screen widget will be removed.
    - Text is removed from all text boxes.
  + Title screen widgets added to screen.
* showHighScoreScreen()
  + Shows the high score screen and widgets.
  + Gets 3 score lists (small, normal, large) using getScoresFromFile() function.
  + Appends scores to text box widget.

Snake

* Imports random to randomise food placement.
* Imports time to force a tick rate (the speed that the code is run) for game loop.
* Instantiate variables, snake and food objects.
* Bind arrow keys to snake arrow key functions.
* Game Loop (while the game is not over):
  + Snake moves.
  + Game over flag is modified if the snake hits itself.
  + Snake is drawn to canvas.
  + Tick rate is modified if the food has been eaten.
  + Food is drawn to canvas.
  + Score is drawn to canvas.
  + The process sleeps for the time indicated by tickRate.

Snake Class

* Instantiate variables.
* Add two elements to the body using addBody() function.
* Score()
  + Property getter for score.
* Bodys()
  + Property getter for the list of elements that makes up the snake’s body.
* addScore()
  + Increments the score by one.
* addBody()
  + Adds an element to the body (increase the snake’s size).
* Draw()
  + Draws the snake to screen.
  + Makes a colour gradient across the snake by adding 100 to the hexadecimal RGB colour value with each loop.
  + Ignores the tail over the snake so that the illusion of a one tile snake can be seen (for the snake to work; the body must be two elements wide see: snake.tick).
* Tick()
  + Updates the snake’s position by removing the tail and adding it to where the head should be.
  + Sets the flag “has moved” to force only one input per tick to stop button mashing.
  + Check if the snake has hit a wall using wallCheck() function.
* wallCheck()
  + Checks if the snake has hit the wall and moves the snake to the corresponding position on the over side.
  + If the snake hits the left wall, its head is moved to the right-side wall and vice versa (same with top and bottom).
* hitSelfCheck()
  + Checks if snake has hit itself and returns flag to end the game.
  + Ignores tail for collisions as the tail is invisible to the player.
* leftKey()
  + Turns snake left if moving on vertical plane.
* rightKey()
  + Turns snake right if moving on vertical plane.
* upKey()
  + Turns snake up if moving on horizontal plane.
* downKey()
  + Turns snake down if moving on horizontal plane.
    - All key functions return if a movement key has already been pressed to avoid button mashing.
    - Flag set to avoid button mashing.
    - Snake can only move from vertical to horizontal plane or vice versa to stop snake from attempting to move over itself.
* spaceKey()
  + Space keyboard Key Debug: To add Elements to the snake’s body.

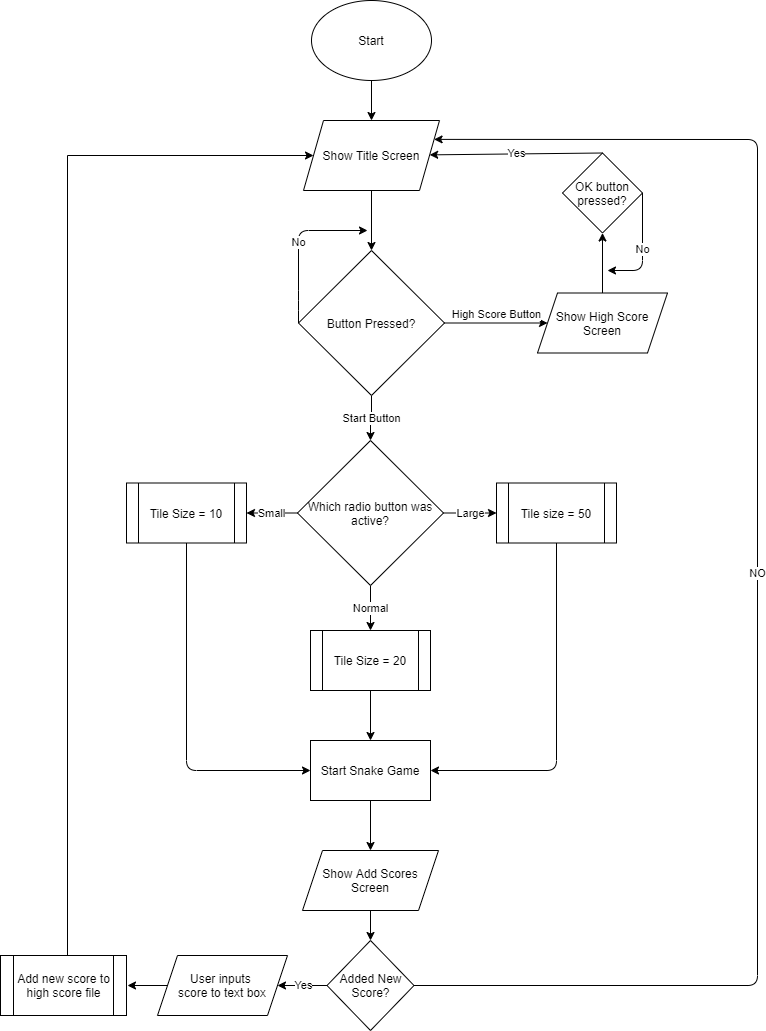
Food Class

* Instantiate variables.
* Adds food to game using addFood() function.
* addFood()
  + Adds food to a random location on screen within the tile bounds and not within the snake.
  + Recursion Error except statement to catch if the snake has filled the screen with the snake and ends the game (filling the screen with the snake is seemingly impossible by a human player because the snake will be moving too fast).
* drawFood()
  + Draws food to the screen.
* foodCheck()
  + Checks if snake ate the food and speeds up the snake, adds a new food tile, increments the score by one and increase the size of the snake if it did.

Element Class

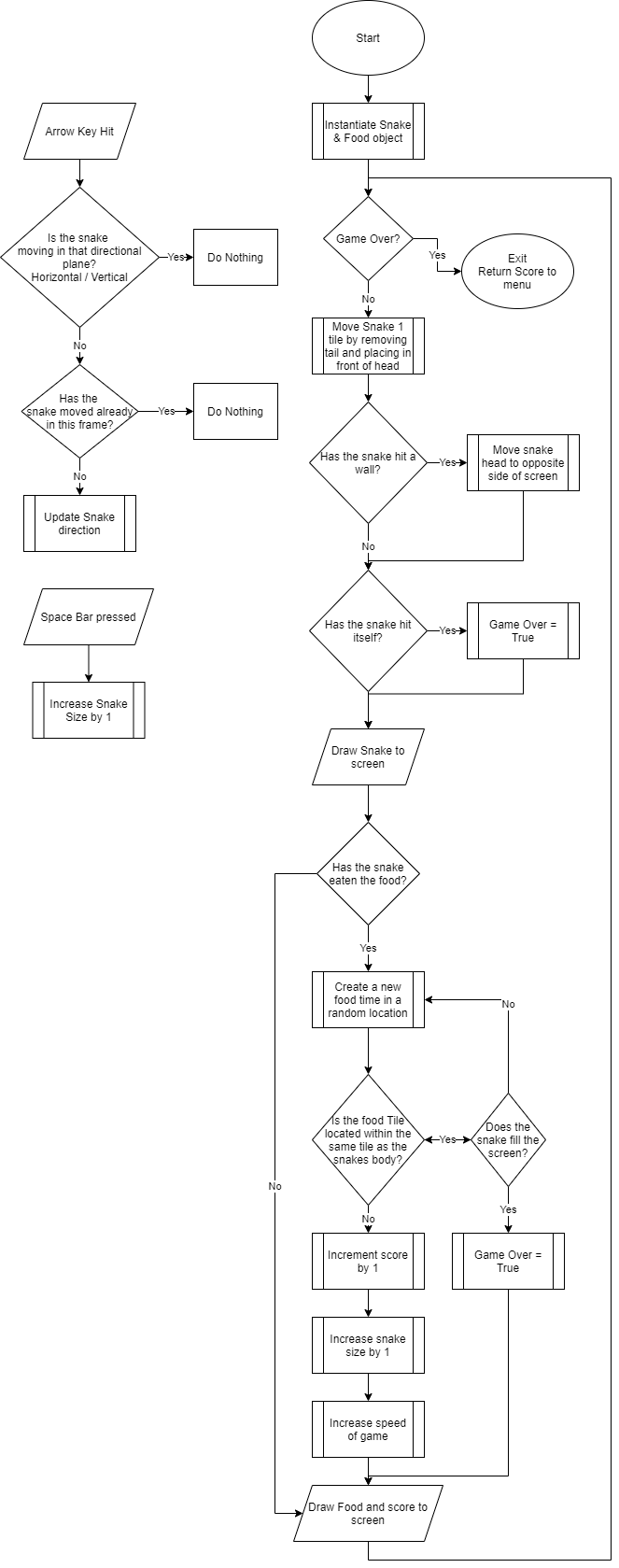
* Element data structure for snake and food body.
* Holds the x and y coordinates for the elements of the snake and food.

Design

Menu.py Flow Chart

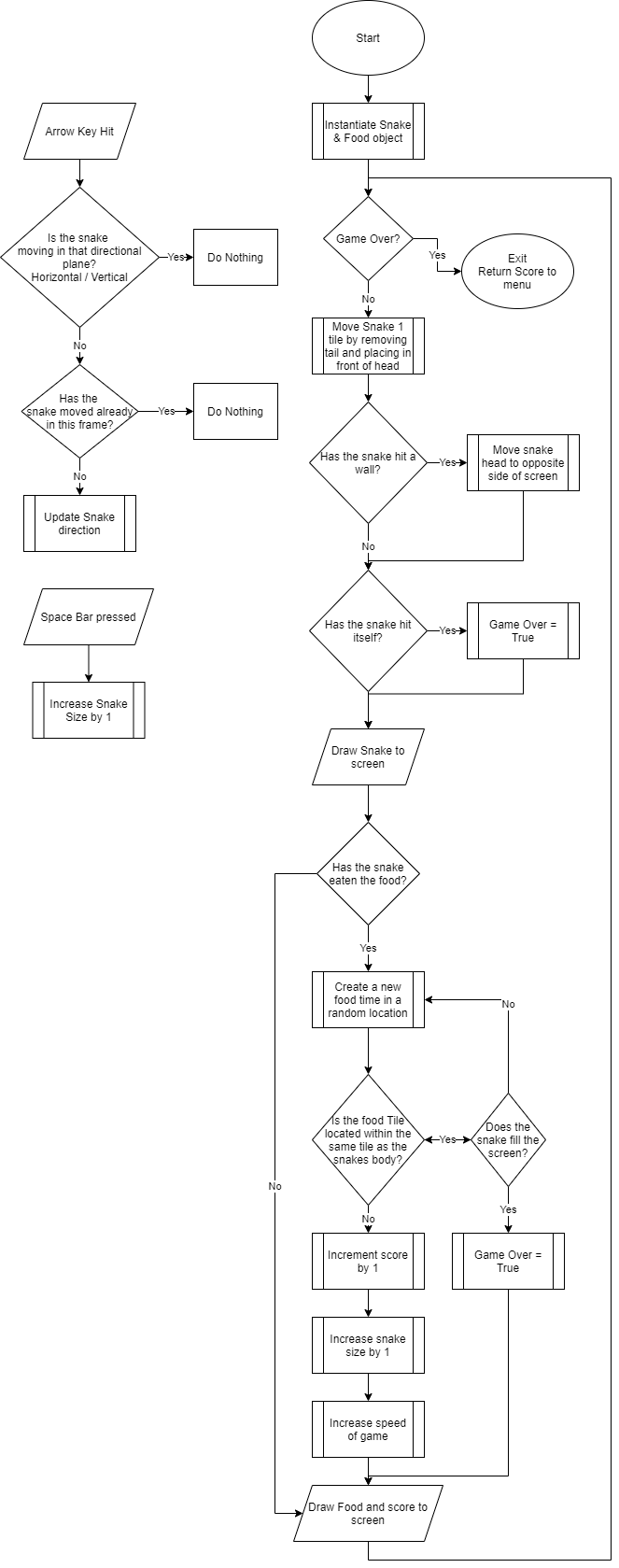
Menu.py Flow Chart

Snake.py and Input Flow Charts

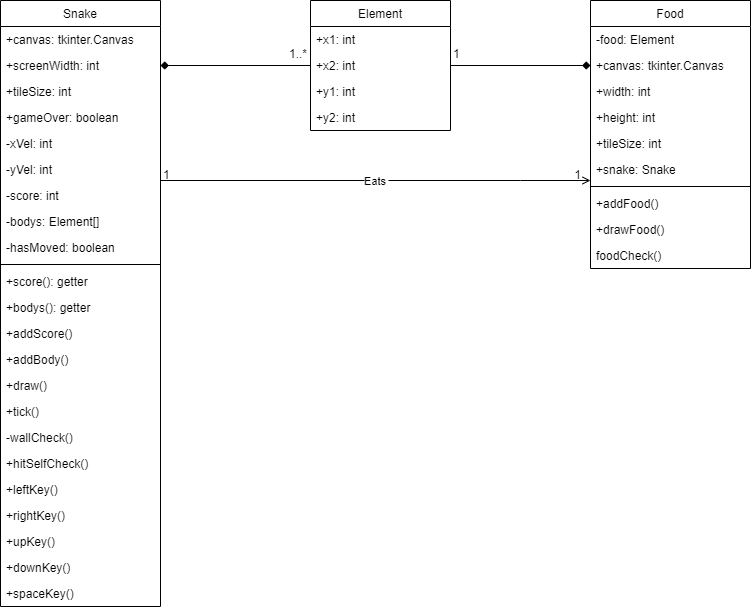


Snake.py flow chart

Input flow charts



Class Diagram



Snake Game Class Diagram

Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Name of test | Description | Expected results | Did it pass? |
| Snake eating food. | Does the snake eat the food when it enters the same tile? | The Food will be eaten, and a new food will be placed in a random area.  The snake will increase in size.  The speed of the snake increases. | True |
| Movement. | Does the snake move by the tail following the head? | The snake should move by moving the last tail element to the front of the head. (giving the illusion of the snake moving by one tile) | True |
| Direction. | Does the snake change direction when the corresponding arrow key is pressed? | The snake should change direction when the arrow keys are hit.  The arrow keys should not be able to be button mashed to get an unexpected result. | True |
| Wall Collision. | Does the snake hitting one of the walls work correctly? | The snakes head should move to the opposite end of the screen when it goes passes the wall threshold to cause the snake to wrap arrowed and stay in the play area. | True |
| Tail Collision. | Does the snake hitting its own tail work correctly? | The game should end upon the head of the snake colliding with any of the tail elements. | True |
| Score Counter. | Does the score counter work correctly? | The score counter should increment by one when the snake eats the food. | True |
| Save High Scores to File | Does the high scores of all the games get saved to a separate file? | All the scores from each game played should be saved to an external file ready to be called upon later. | True |
| Show high scores on screen | Does the high score screen show all the scores from past games? | The high score screen should show all scores from past games; each within their separate sizes. | True |

Critique

If I did the project again, I would probably spend more time on the customisation. For example, I would allow the player to choose if hitting the walls killed the snake and maybe include a variety of maps which would add obstacles that the snake had to avoid to not cause a game over. I would add some variety to the food objects; some would cause the snake to slow down, speed up greatly or bad food to avoid which would lower your score.