Snake Game

# Design Document

### By Mal (Daniel) Lambert for CSC221 Final Project

## **Introduction**

**Project Functionality**

My snake game provides three difficulty options to choose from once the player first opens the game. After selection, the game loop is simple: the player navigates a snake around a limited arena, eats food to grow the snake, and avoids navigating into the arena’s walls or the snake’s own growing tail. The snake moves faster as it grows. If (when) the player loses, they’re brought to a game-over screen that presents the reason they lost, their score, and the option to play again or quit.

**Design Process**

I started the project by writing my first draft of pseudocode for how I expected the game to work, then researched how others had built this kind of game. After studying Geeksforgeeks.org’s snake game in particular, I reordered my pseudocode and started coding my own version in VS Code. Through testing and debugging I got it to a complete state, and after adjusting the parameters and difficulty to my liking, wrote the last draft of my pseudocode for others to write their own version of my game if they so like.

## **Project Development**

**Pseudocode**

(Words in #hashes# refer to variables)

Import libraries:

pygame, random

Assign global variables:

- Make colors (#gray#, #purple#, #blue#, #blackish#, #white#), assign them appropriate RGB values

- Make #arena\_width#, assign it 825

- Make #arena\_height#, assign it 525

Define main():

- Initialize pygame

- Initialize pygame.display.set\_mode() with (#arena\_width#, #arena\_height#), assign to #window#

- Initialize pygame.display.set\_caption() and set caption to "[Your Name]'s Snake Game"

- Initialize pygame.time.Clock(), assign to #time\_var#

- Fill #window# with #blackish#

- Begin difficulty selection screen by calling pygame.font.SysFont("font", size, bold=True).render("difficulty selection: e, m, h", True, #white#), assign to #difficulty\_font#

- Create the pygame surface for that message by calling #difficulty\_font#.get\_rect(center=(#arena\_width# / 2, #arena\_height# / 2.25)), assign to #difficulty\_surface#

- Display all that by calling #window#.blit(#difficulty\_font#, #difficulty\_surface#) and then pygame.display.flip()

- Make a new variable, #select\_screen#, and assign it True

- Start a while-loop that runs as long as #select\_screen# is True:

- Use for-loop for each event in pygame.event.get():

- If event.type is pygame.QUIT, call pygame.quit() and then quit()

- If event.type is pygame.KEYDOWN, then check for if event.key is pygame.K\_e, pygame.K\_m, or pygame.K\_h -- for each, assign a #difficulty# of 1, 2, or 3, and set #select\_screen to False

- Starting assignment for the snake, make a #head# list, assign it [75, 255]

- Make a #body# tuple, assign it [[75, 255], [60,255]]

- Make a #direction# variable, assign it "right" string

- Make a #change\_dir# variable, assign it #direction#

- Begin food assignments by making a #food\_loc# variable; assign it random tuple coordinates with [random.randint(0, (arena\_width // 15 - 15)) \* 15, random.randint(repeat for height)]

- Make a #food\_ate# boolean, assign it True

- Begin game loop with a while-loop set to True:

- Fill #window# with #gray#

- Define player movement with a for-loop for each event in pygame.event.get():

- If event.type is pygame.QUIT, call pygame.quit() and then quit()

- If event.type is pygame.KEYDOWN, check for whether event.key is pygame.K\_LEFT, pygame.K\_UP, pygame.K\_RIGHT, or pygame.K\_DOWN; if K\_LEFT, assign "left" to #change\_dir#, repeat for each key

- Disallow player from turning snake directly into its body by checking each input: if #change\_dir# is assigned "left" AND #direction# is NOT assigned "right", assign "left" to #direction#; repeat for "up", "right", and "down"

- Move snake by checking if #direction" is assigned "left"; if so, change first item in #head# tuple to itself minus 15 (-= 15)

- If #direction# is assigned "up", change second item in #head# tuple to -= 15

- If #direction# is assigned "right", change first item in #head# tuple to itself plus 15 (+= 15)

- If #direction# is assigned "down", change second item in #head# tuple to += 15

- Insert a tail segment under the new location of the head through #body#.insert(0, list(head)), assigning it the first location in the #body# tuple

- Check if snake ate food by checking if first items in #head# and #food\_loc# tuples are the same, AND if the second items in #head# and #fooc\_loc# tuples are the same; if so, set #food\_ate# to False; otherwise, call #body#.pop() to remove farthest tail segment

- If #food\_ate# is False, then reassign #food\_loc# random tuple coordinate with same code as originally, then set #food\_ate# to True

- Redraw the background with #window#.fill(gray)

- Draw the snake in the arena; run a for-loop by checking for every loc in #body#, call pygame.draw.rect(#window#, #blue#, pygame.Rec(loc[0], loc[1], 15, 15))

- Draw the food in the arena by calling pygame.draw.rect(#window#, #blue#, pygame.Rect(for-loop-coordinate[0], for-loop-coordinate[1], 15, 15))

- Update player's score before by calling len(#body#) and assigning it to #body\_segments#, then call int(#body\_segments# - 2) and assign it to #score#

- Check for game over by determining if the first item in the #head# tuple is either less than 0 or greater than #arena\_width#, OR if the second item in #head# tuple is either less than 0 or greater than #arena\_height#; if so, call #game\_over#() with the parameters "You ran into the wall.", #window#, and #score#

- Run a for-loop by checking for each segment in #body#[1:], ignoring the first segment; if the first items in #head# and segment tuples are the same, AND the second items in #head# and segment tuples are the same, then call game\_over() with the parameters "You ran into the your tail.", #window#, and #score#

- Refresh the display by calling pygame.display.update()

- Update the difficulty of the snake with three if's; if #difficulty# is set to 1, then assign (#body\_segments# / 6) + 10 to a new #speed# variable

- If #difficulty# is 2, then make #speed# (#body\_segments# / 4) + 15)

- If 3, then (#body\_segments# / 3) + 20

- Update the game's running speed by setting this new variable to the frame rate through calling #time\_var#.tick(#speed#)

Define game\_over(reason, #window#, #score#):

- Redraw background with #window#.fill(blackish)

- Begin game over screen by writing the game over message, score, and offer to play again through calling pygame.font.SysFont("font", size).render(f"{reason} Score: " + str(score) +". Play again? y/n", True, #white#), assign to #g\_o\_font#

- Create the pygame surface for that message by calling #g\_o\_font#.get\_rect(center=(#arena\_width# / 2, #arena\_height# / 2.25)), assign to #g\_o\_surface#

- Display all that by calling #window#.blit(#g\_o\_font#, #g\_o\_surface#) and then pygame.display.flip()

- Allow playing again through a while-loop set to True:

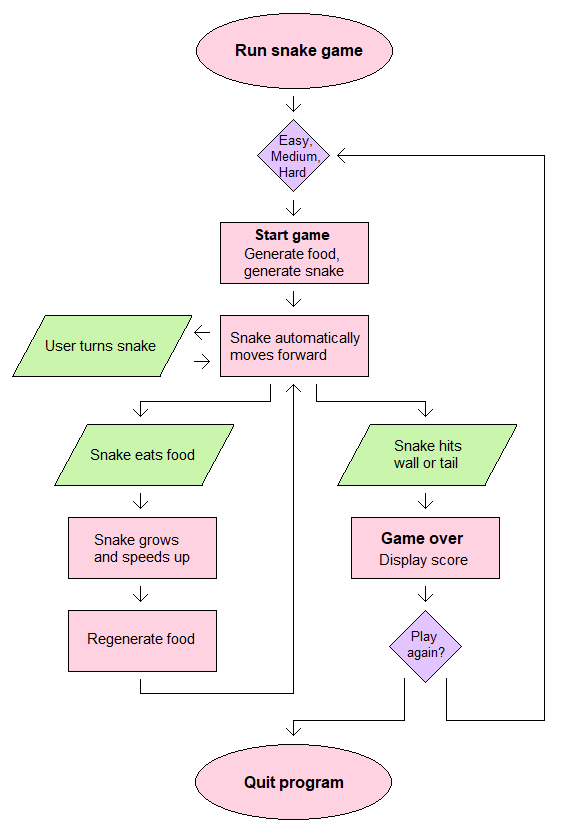
- Use for-loop for each event in pygame.event.get():

- If event.type is pygame.QUIT, call pygame.quit() and then quit()

- If event.type is pygame.KEYDOWN, then check for if event.key is pygame.K\_n or pygame.K\_y -- if K\_n, then quit game by calling pygame.quit() and quit(); if K\_y, then call main()

Begin game by calling main()

**Flowchart**



**Requirements**

**• Includes user interaction (e.g., command-line inputs, GUI, keyboard movement,**

**etc.).**

Accomplished

**• Performs some processing (e.g., conditional logic, loops, function calls, math**

**operations).**

Accomplished

**• Generates cleanly formatted output based on input/processing.**

Accomplished

**• Implements at least one custom function/method/module.**

Accomplished

**• Use an imported Python library.**

Accomplished