Smart Snake

Team Smart Snake Project Documentation

Benjamin Axline (U19649803), Eric Davis (U28393843), Albert Zhao (U68071332)

Github Repo Link:

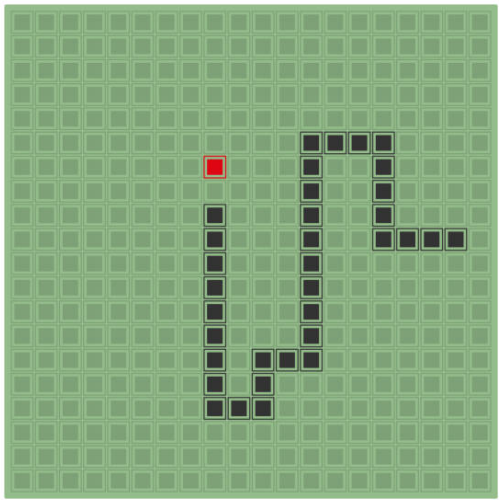
https://github.com/benaxline/EC327_FinalProject_SmartSnake

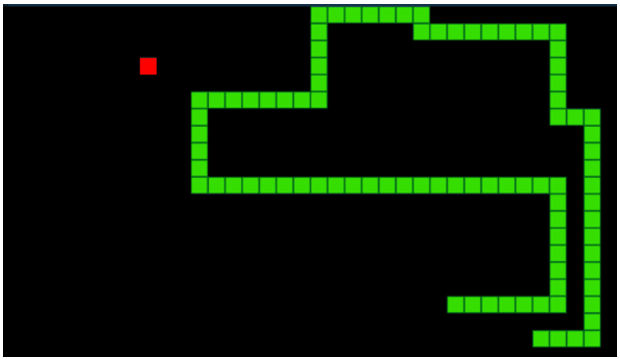
Youtube Project Demo Link:

https://www.youtube.com/watch?v=b1D-0l0GVpE

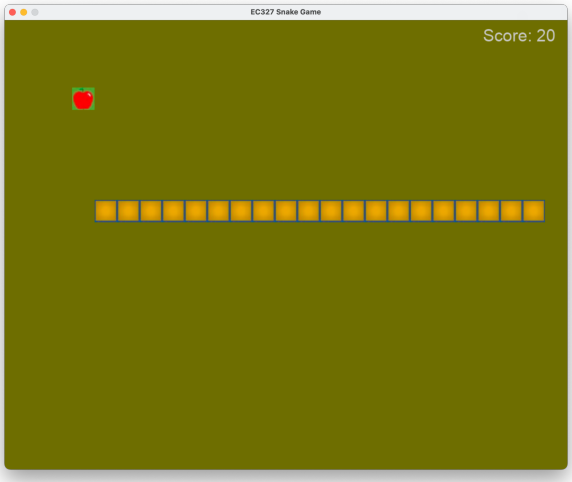
Table of Contents:

Project Overview:





A traditional snake game typically involves controlling a snake-like character that moves around a screen, trying to eat food items while avoiding running into walls or its own body. The snake grows longer as it consumes more food, and the game becomes more difficult as the snake gets longer and has to navigate tighter spaces. The game's goal is to score as many points as possible by eating as much food as possible without dying. The game ends when the snake hits a wall or itself.



The Smart Snake is like the traditional snake game, but with a twist. Instead of only including the generic snake game, the Smart Snake game includes different modes for the user to select amongst. These modes are normal, trivia and math. If the user chooses trivia or math, the game plays out a little differently. Instead of starting off at zero and increasing whenever an apple is hit, the snake will instead start off with 20 blocks. In these modes, each time the snake touches an apple, the game will prompt either a math or trivia question, depending on the mode the user selected on the welcome page. If the user answers the question correctly, the length of the snake will become 1 block shorter. On the contrary, if the user answers the question incorrectly, the snake will grow 1 block longer. **The player wins the game if the length of the snake becomes 0.**

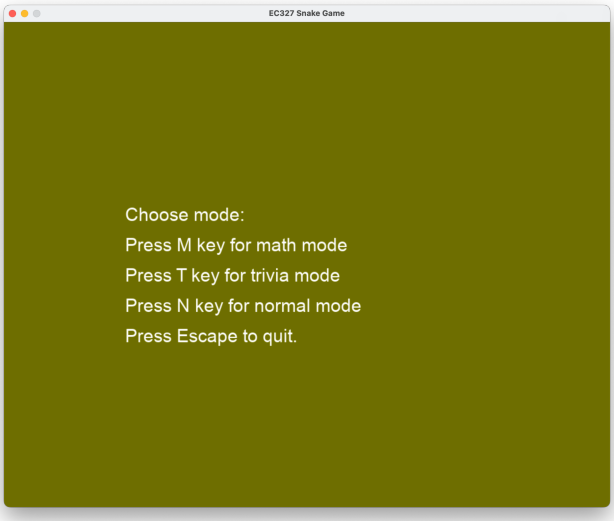
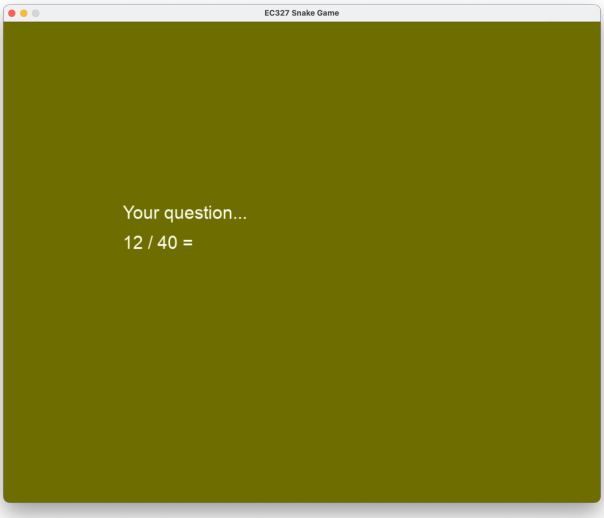


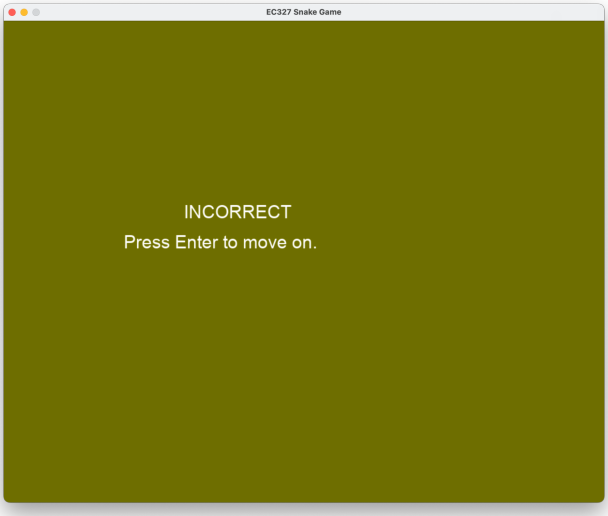
Figure 3: Welcome page, game mode selection window

The main menu is where the user can select the mode is chosen can be seen above. Listed are the math, trivia and normal modes. Once the specific key is pressed, the game will commence in that specific game mode.

Math Mode:

In math mode, the snake will start off with 20 blocks. The graphics look like a traditional snake game. However, when the snake touches or consumes an apple, the game will prompt a mathematical question:

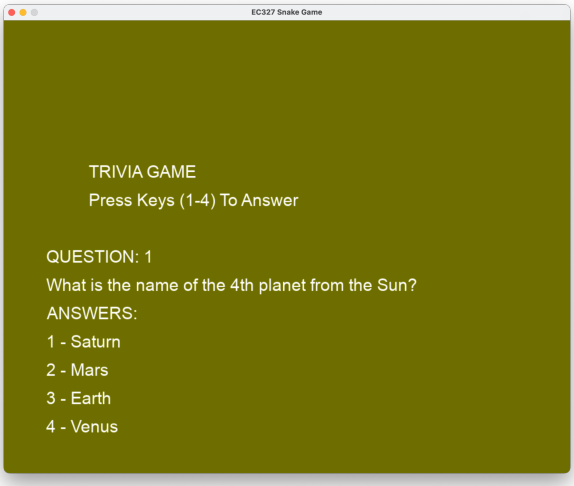




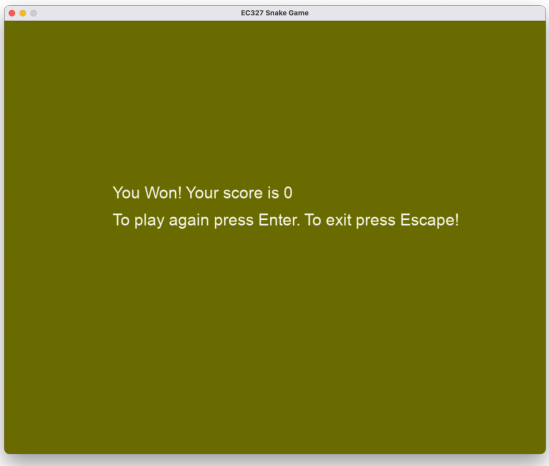
As described earlier, if the player answered the question correctly, the snake length will be reduced by 1 block. If the player answers the question incorrectly, the snake length will be increased by 1 block.

Trivia Mode:

Similar to math mode, the snake will start with 20 blocks. But instead of the player answering math questions, the player will instead answer trivia questions.



If the player is smart enough and successfully answers enough questions to shorten the snake length to 0 while making sure that the snake doesn't collapse into the wall or into itself, the game will display a victory message to the player.



Normal Mode:

The normal mode plays just like a traditional snake game! Try to eat as many apples as you can, but be careful not to run into the walls or your own tail. Every time the snake eats an apple, its body will grow longer, making it harder to maneuver. The game ends if the snake runs into a wall or its own tail.

Targeted Audience:

We all know that people hate doing math or trivia questions. We hope that Smart Snake will make math and trivia questions more enjoyable. Research has also shown that people tend to be more engaged in learning when it involves some sort of "game" aspect. By integrating math and trivia questions into a traditional snake game, we hope that people will start liking those questions more and become an absolute academic weapon.

Challenges:

- The first challenge we encountered was that the buttons don't really work in python. It is really hard to create a button instance. We constantly ran into the program where every time we created a button and clicked on the button, the program skips over everything and immediately shows the game over.
    - We solved this issue by just overlaying the information over the same canvas/window, and pausing the game in the background.
- Another challenge we encountered was that we were trying to use a secondary pop-up window. We eventually decided not to use a pop-up window because pygame actually doesn't support a secondary pop-up window. We would have to use another python module to achieve what we wanted.
    - We resolved this issue by overlaying the questions over the same canvas/window, and pausing the game in the background.
- The biggest challenge we faced is probably learning python in a short period of time. Python is also an object-oriented programming language so it is not too hard to learn. We just had to learn all the syntax differences between python and C++.

Technical Documentation:

Right from the start, our team knew that we wanted to use python and pygame to create the Smart Snake. Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows our team to create fully featured games and multimedia programs in the python language. Pygame is also highly portable and runs on nearly every platform and operating system.

To import pygame into our computer system:

```
1  import pygame
2  from pygame.locals import *
```

Those two lines add the pygame module into the program.

Game.py

The main components (canvas and game window) of the game are not in main.py, but rather, in game.py.
Here is the code snippet of the default constructor for class Game:

```
24  class Game:
25      def __init__(self):
26          pygame.init()
27          pygame.display.set_caption("EC327 Snake Game")
28          self.surface = pygame.display.set_mode((1000, 800))  # set mode: setting the window size of the game
29          self.snake = Snake(self.surface)
30          self.snake.draw()
31          self.apple = Apple(self.surface)
32          self.apple.draw()
33          self.snake_zero = False
34          self.off_grid = False
35          self.math = Math()
36          self.pause = False
37          self.running = False
38          self.mode = ''
39          self.expression = ""
40          self.math_answer = 0
41          self.num1 = 0
42          self.num2 = 0
43          self.op = ''
44          self.current = 0
45          self.scored = False
46          self.failed = False
47          self.trivia = Trivia("trivia_data.txt")
48          self.result = 0
49          self.count = 0
50
```

The Class game handles all the display of the main menu, math questions, and trivia questions.

- Displaying math problems:

```
71      def displayMathProblem(self):
```

- Displaying trivia questions:

```
193     def displayTriviaProblem(self):
```

- Displaying score:

```
412     def display_score(self):
```

- Displaying game over page:

```
417     def show_game_over(self):
```

- Displaying game won page:

```
427     def show_game_won(self):
```

- Displaying snake moving off-grid message:

```
437        def show_off_grid(self):
```

The Game class also handles the component used to generate math questions:

```
494     def math_exp(self):
495         num1 = random.randint(1,100)
496         num2 = random.randint(1,100)
497         num = random.randint(1,4)
498         if num == 1:
499             operator = '+'
500             self.math_answer = num1 + num2
501         elif num == 2:
502             operator = '-'
503             self.math_answer = num1 - num2
504         elif num == 3:
505             operator = '*'
506             self.math_answer = num1 * num2
507         elif num == 4:
508             operator = '/'
509             self.math_answer = num1/num2
510         expression = str(num1) + " " + operator + " " + str(num2)
511         return expression
```

Math.py:

Math class generates math questions. In addition, it also checks if the user answers the question correctly.

```
17      def generate_operator(self):
18          #determine which operator to use
19          num = random.randint(1,3)
20          if num == 1:
21              operator = '+'
22          elif num == 2:
23              operator = '-'
24          elif num == 3:
25              operator = '*'
26          # elif num == 4:
27          #     operator = '/'
28          return operator
```

We used a random number generator from 1 to 3 to decide which type of math operation the question should ask:

1 - addition

2 - subtraction

3 - multiplication

We did not include division because the questions became too complicated for players to mimic the correct answer.

```
41      def solve(self):
42          first = self.first_num
43          second = self.second_num
44          if self.operator == '+':
45              answer = first + second
46          elif self.operator == '-':
47              answer = first - second
48          elif self.operator == '*':
49              answer = first * second
50          elif self.operator == '/':
51              answer = first / second
52          return answer
```

Then, the solve membered function produces the result of the math game. The answer is then returned to Game class to compare with the player's answer input.

Trivia.py:

The Trivia class handles all the operations related to trivia questions. Before starting, we obtained 40 trivia questions and organized them into a single txt file:

trivia.txt:

```
1  Which fast food restaurant has the most locations in the world?
2  Jack In The Box
3  Chipotle
4  Subway
5  McDonald's
6  3
7  Which US city is the sunniest major city?
8  Phoenix
9  Miami
10 San Francisco
11 Austin
12 1
13 Which planet has the most moons in the solar system?
14 Uranus
15 Saturn
16 Neptune
17 Jupiter
18 4
```

Displaying questions are handled by the show_question membered function:

```
39      def show_question(self):
40          print_text(font1, 10, 5, "TRIVIA GAME")
41          print_text(font2, 190, 500-20, "Press Keys (1-4) To Answer", purple)
42          print_text(font2, 530, 5, "SCORE", purple)
43          print_text(font2, 550, 25, str(self.score), purple)
44
45          #get correct answer out of data (first)
46          self.correct = int(self.data[self.current+5])
47
48          #display question
49          question = self.current // 6 + 1
50          print_text(font1, 5, 80, "QUESTION " + str(question))
51          print_text(font2, 20, 120, self.data[self.current], yellow)
52
53          #respond to correct answer
54          if self.scored:
55              self.colors = [white,white,white,white]
56              self.colors[self.correct-1] = green
57              print_text(font1, 230, 380, "CORRECT!", green)
58              print_text(font2, 170, 420, "Press Enter For Next Question", green)
59              sys.exit()
60
61          elif self.failed:
62              self.colors = [white,white,white,white]
63              self.colors[self.wronganswer-1] = red
64              self.colors[self.correct-1] = green
65              print_text(font1, 220, 380, "INCORRECT!", red)
66              print_text(font2, 170, 420, "Press Enter For Next Question", red)
67
68          #display answers
69          print_text(font1, 5, 170, "ANSWERS")
70          print_text(font2, 20, 210, "1 - " + self.data[self.current+1], self.colors[0])
71          print_text(font2, 20, 240, "2 - " + self.data[self.current+2], self.colors[1])
72          print_text(font2, 20, 270, "3 - " + self.data[self.current+3], self.colors[2])
73          print_text(font2, 20, 300, "4 - " + self.data[self.current+4], self.colors[3])
```

That is the basic overview of our code structure.