**grafik, ekran görüntüsü, simge, sembol, grafik tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu Snake Game Coding in Python**

Aysu Genç, Mert Özkumanlar, Yasin Berk Şimşek

Cracow University of Technology

**Abstract**

This report investigates the development of the classic Snake game using Python's Tkinter library. The primary focus is on the essential of the code, including game initialization, event handling, movement logic, collision detection, and rendering. The objective is to explain the methodology of the snake game code, discuss the core functionalities, and evaluate the game's performance and potential improvements.

**Introduction**

The Snake game is a well-known and simple arcade game that has been implemented across various platforms. The game is basically moving a line (snake) to collect dots (food) while avoiding collisions with the walls or its own body. Python was used to code this game in this project. Python is a flexible and widely-used programming language.

The Snake Game’s history begins in the 1970s. The earliest version, "Blockade," was released in 1976 by Gremlin Industries. Back then, it was a multiplayer arcade game where players controlled lines which grew as they are moved. Aim was to avoid collisions with walls or other lines. The simple concept was the reason for the game’s popularity.

The game got into mainstream culture in the late 1990s when Nokia included Snake on its mobile phones in 1997. It became an addictive game which captivated millions worldwide. Over the years, The Snake Game has been remade across various platforms, including PCs, gaming consoles, smartphones, and web browsers. Despite these adaptations, the core mechanic of navigating a growing line to consume food while avoiding collisions remained unchanged, showing this game is timeless.

In this project, Tkinter was used for the graphics. Tkinter is a standard GUI (Graphical User Interface) toolkit in Python, to put the codes in practice. The aim of this project is to explore the implementation details, focusing on the game logic and the integration of the Tkinter library to handle graphical rendering and user interactions.

**Methodology**

The snake game was encoded by adhering to a methodology. Below are the followed steps.

**Game Initialization**

The first thing done is to import modules that we need. In this project, Tkinter is used for the graphics. Random is needed for random functions. After they were imported, the necessary constants such as the grid size (*rows, cols*) and tile dimensions (*tile\_size*) were defined. The window dimensions are calculated based on these constants. The Tile class is introduced to represent the snake and food positions within the game grid. The size of snake and food were arranged.

**Creating the Game Window**

A Tkinter window is created to display the game. The window is centered on the screen by an additional code. And a canvas is added for drawing the game elements.

**Game Variables and Initial State**

The game is initialized with a *snake\_head, food\_tile* (they are defined in terms of *tile\_size*), *velocity\_x,* *velocity\_y*(which are zero*), game\_over*, and *score*. For *game\_over*, it is *False*. And *score* starts from 0.

**Event Handling**

A function *on\_key\_release* is defined to handle the keyboard input for changing the snake's direction, ensuring that the snake does not reverse into itself. To do so, *Up, Down, Left, Right* were defined using *if* and *elif* cases and using velocities in both axis. Also, system *return*s if *game\_over*.

*velocity\_x, velocity\_y* and *game\_over* are the *global* variables.

**Movement and Enlargement of the Snake**

The *move* function updates the snake's position based on its velocity. It also handles collision detection with the walls, the snake's own body, and the food. If snake touches the window or itself, it yields *return*, and *game\_over* is *True*.

The food must pop up somewhere else, randomly, after it was eaten. So it is encoded with *random.randint*, which yields random integers to be the new coordinates of the food.

In the snake game, snake enlarges as it eats. For each food eaten, snakes tail enlarges one *tile\_size*.

**Rendering the Game**

The *render\_game* function is responsible for rendering the game elements on the canvas, including the snake, the food, score and a game over message. For snake and food, *canvas.create\_rectangle* was used. For score and game over text, *canvas.create\_text* was used. When a new frame is drawen, the previous frame must be deleted. For this, *canvas.delete('all')* isapplied after *update\_snake\_position()*.

One can select different colors for elements by taking a look at canvas library.

The *render\_game* function is called initially, and the *on\_key\_release* function is bound to the keyboard events.

For the velocity of the game overall, *window.after(x, render\_game).* Here, x is frames per second.

Direction of the snake changes when releasing the arrow keys was encoded. Finally, *window.mainloop()* was used for our game window to stay open.

ekran görüntüsü, ekran, görüntüleme, metin, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 1. A Capture From The Snake Game

**Discussion**

The implementation of the Snake game using Python and Tkinter highlights several important aspects: event handling, game logic, and rendering. The code provided efficiently accomplishes these tasks; however, potential issues should be investigated to improve the gameplay experience. One notable feature is the ability for the game to restart, allowing players to easily start a new game without closing the application. Additionally, the game's speed increases as the player's score rises, adding a dynamic challenge and keeping the gameplay engaging.

There are some issues to discuss about the Snake Game. Firstly, the arrow keys and movement of the snake. The current implementation binds the arrow keys to change the snake's direction. If the keys are released too quickly, the direction changes might not register correctly. Besides that, simultaneous key presses can result in conflicting directions.

Secondly, the snake game ends when the snake collides with the window boundaries. High-speed movement or small window sizes can cause issues. Similarly, self-collisions are detected by iterating through the snake's body tiles, which works well for smaller games but might become inefficient for larger games or higher speeds. In this project, 10 frames per second was applied without yielding any problem.

Thirdly, the rendering process, while functional, could also see enhancements. The game runs at a fixed frame rate of 10 frames per second, which might be too slow or too fast depending on the hardware and user preference. Allowing dynamic adjustment of the frame rate based on game difficulty or user settings could improve the experience. Additionally, the entire canvas is cleared and redrawn every frame. In this project, since this is a basic game, it is not a problem. However, this approach might lead to performance issues for larger games or higher resolutions. Optimizing the rendering process by only updating the parts of the canvas that change could be better for more complicated projects. One can also add sound effects and more textures on graphics and even difficulty levels by adding obstacles to make the game more appealing.

**Conclusion**

The Snake Game project using Python and Tkinter is a great way to boost basic Python skills. In this project, you get to understand fundamental game development concepts like starting up game components, handling user inputs, implementing game logic, creating graphics, and being methodical. Building an interactive game from scratch is a hands-on experience that's really valuable for learning.

One of the cool features in this game is the restart option. By pressing a key, players can quickly reset and start over without needing to close and reopen the application. This makes the game more user-friendly. Another fun feature is that the game gets harder as you play. As your score goes up, the snake gets faster, which makes the game more challenging and exciting.

This project, while simple, is a perfect starting point for anyone new to game development with Python. Future versions could focus on making the game even better, like improving the user experience, optimizing performance, and adding new features. For example, adding sound effects, better graphics, and different difficulty levels with obstacles would make the game more engaging and fun.

In summary, the Snake Game project is a practical and educational way to show how to use Python and Tkinter for game development. By building on this foundation, developers can create more complex and fun games that players will enjoy. This project shows the potential of Python and Tkinter for making interactive applications and is a great stepping stone to more advanced game development projects.