Snake Game

Abhyud Krishna#1, Bandi Amith Sreenivasa Reddy#2, Shreehari SM #3, Chandrakiran Kopparappu#4

*#Amrita School of Computing, Amrita Vishwa Vidyapeetham*

*Bengaluru, Karnataka, India*

1bl.en.u4aie23103@bl.students.amrita.edu

2bl.en.u4aie23104@bl.students.amrita.edu

3bl.en.u4aie23128@bl.students.amrita.edu

4bl.en.u4aie23141@bl.students.amrita.edu

***Abstract*— (have to fill)**

***Keywords*— nand2tetris, , jack and hack language, VM Translator, Jack compiler.**

# INTRODUCTION

Nand2Tetris is an educational project that guides learners through the process of building a computer from the ground up, starting with basic logic gates and culminating in a fully functional computer system capable of running programs written in a high-level language. The Snake game is a timeless classic that has entertained players for decades. Originating from the era of arcade gaming, it has transcended generations and platforms to become a staple in the world of casual gaming. In this project, we present our implementation of the Snake game developed as part of the Nand to Tetris course. The objective of the game is simple yet addictive: control a snake as it moves around a grid, consuming food pellets to grow longer while avoiding collisions with itself and the walls. With each pellet consumed, the snake's length increases, making maneuvering more challenging. The game continues until the snake collides with itself or the boundaries of the grid. The game runs on the VM computer emulator or, tools provided by the Nand2Tetris course to test and execute programs on the custom-built hardware platform.

# DESIGN AND IMPLEMENTATION

*A. Design*

The design of the Snake game follows established rules and gameplay mechanics:

* Game Rules: In the Snake game, players take control of a snake represented by a series of connected segments on a grid-based playing field. The primary objective is to guide the snake to consume food pellets scattered throughout the grid. Each food pellet consumed increases the length of the snake, posing a greater challenge as the game progresses. However, players must navigate the snake carefully to avoid collisions with both itself and the boundaries of the playing field. If the snake collides with any part of its own body or reaches the edge of the grid, the game ends immediately, resulting in defeat for the player. Therefore, players must balance the pursuit of food pellets with strategic movement to ensure the snake's survival and maximize their score.
* Gameplay Mechanics: Gameplay mechanics are designed to offer dynamic and responsive controls, allowing players to direct the snake's movement using arrow keys or designated controls. The snake moves continuously in the direction indicated by the player's input, with no option to stop or reverse its movement. Randomly appearing food pellets provide opportunities for score accumulation, with each pellet consumed contributing to the player's overall score. However, collision detection ensures that the game ends abruptly if the snake collides with itself or the boundaries of the playing field, adding an element of risk and challenge to the gameplay experience.
* User Interface: The user interface is carefully crafted to provide clear and intuitive feedback to the player. A grid-based playing field prominently displays the snake, food pellets, and grid boundaries, facilitating easy navigation and visualization of gameplay elements. Instructions and feedback messages offer guidance and assistance to the player, providing notifications of score updates, collisions, and game over conditions. Visual cues differentiate between the snake, food pellets, and grid cells, enhancing the clarity and readability of the game interface. End-game notifications provide closure to the player's gaming session, summarizing their performance and outcomes achieved throughout the gameplay session.

*B. Implementation*

The design and implementation of the snake game involve several key components, each responsible for specific aspects of the game's functionality. This section elaborates on the main components and their interactions.

1. Overall architecture: The overall architecture of the Snake game is structured around a set of interconnected classes, each serving a specific role in managing the game's behavior and presentation. At the core of this architecture lies the SnakeGame class, which orchestrates the overall game flow and coordinates interactions between different components. The Snake class represents the player-controlled snake entity, responsible for its movement, growth, and collision detection. Complementing the Snake class is the SnakeGrid class, which defines the game grid and handles rendering operations to display the snake and other game elements. Additionally, the Random class manages random number generation for various game mechanics, such as determining the initial snake position and placing food pellets. Together, these classes form the backbone of the game's architecture, providing a robust foundation for implementing and extending the Snake game's features and functionality.
2. Game Logic: The game logic of the Snake game encompasses a series of rules and mechanics governing the behavior of the snake, interactions with the game environment, and progression through the game levels. At its core, the Snake class manages the movement of the snake, including updating its position based on user input and collision detection with walls and itself. The SnakeGame class oversees the overall game state, including the snake's length, score, and level progression. It also handles the placement of food pellets within the game grid and triggers events such as the snake's growth upon consuming food. Additionally, the SnakeGrid class defines the boundaries of the game grid and facilitates collision detection between the snake and the grid boundaries. By encapsulating these rules and mechanics within dedicated classes, the game logic ensures consistent and engaging gameplay, challenging players to navigate the snake effectively while avoiding collisions and strategically consuming food to advance through the game levels.
3. Graphics Rendering: Graphics rendering in the Snake game involves the visual representation of various game elements on the screen to provide players with a clear and intuitive interface. The SnakeGrid class manages the rendering of the game grid, including the grid lines and boundaries, to provide a spatial context for the snake's movement. Additionally, the Snake class is responsible for rendering the snake itself, dynamically updating its position and appearance as it moves across the grid. Each segment of the snake is drawn individually, creating a smooth and fluid animation effect as the snake slithers through the game environment. Furthermore, the Snake Game class coordinates the rendering of additional visual elements, such as food pellets and status indicators, to convey essential information to the player. By combining these rendering components, the game presents players with a visually engaging experience, enhancing immersion and facilitating intuitive interaction with the game mechanics.
4. User Interface: The user interface component facilitates player input and feedback, allowing users to control the snake's movement and receive information about the game state. Keyboard inputs are processed to change the snake's direction and perform other game actions, such as pausing or quitting the game. Feedback messages and status updates are displayed to inform players of their score, level, and current game status, enhancing the overall gameplay experience.

# CODE STRUCTURE

The code structure of the snake game is meticulously organized into various classes, each encapsulating specific functionality. This modular design facilitates ease of maintenance, readability, and extensibility. Below is a detailed breakdown of the key components and their interactions.

1. *Overall Architecture*
2. Main Class (Main.jack): Serving as the entry point of the game, the Main class initializes essential game components and manages the main game loop. It orchestrates the setup of the initial game state, including the grid, view, and event handling.
3. SnakeGame Class (SnakeGame.jack): The SnakeGame class encapsulates the core game logic and flow. It handles various aspects of gameplay, such as controlling the snake's movement, managing collisions, updating the game state, and handling user input.
4. SnakeGrid Class (SnakeGrid.jack): Responsible for managing the game grid, the SnakeGrid class handles the rendering and state management of the grid cells. It facilitates the visualization of the snake's movement and other game elements on the grid.
5. Snake Class (Snake.jack): The Snake class represents the player-controlled snake entity within the game. It manages the snake's position, length, movement direction, and collision detection. Additionally, it handles the snake's growth and movement mechanics.
6. RandSeed Class (RandSeed.jack): The RandSeed class provides functionality for generating a random seed value based on user input. This seed is used to initialize the game's pseudo-random number generator, ensuring unpredictable gameplay elements such as food pellet placement.
7. Random Class (Random.jack): Similar to the RandSeed class, the Random class is responsible for generating random numbers used in various game mechanics, such as determining the snake's initial direction and positioning food pellets on the grid.
8. *Game Logic*

The game logic is implemented primarily in the SnakeGame, Snake, and SnakeGrid classes, involving several key functions:

1. Grid Initialization: The grid is initialized with a specified width and height, divided into 4x4 pixel blocks. The SnakeGrid class handles the creation of the grid, ensuring it is ready for gameplay by drawing an initial dividing line and preparing the cell states.
2. Food Placement: The SnakeGrid class manages the random placement of food pellets on the grid. The placeFood method ensures that food is placed in an unoccupied cell by checking the grid for the presence of the snake.
3. Snake Movement and Growth: The Snake class controls the movement of the snake. The tryMove method determines the snake's new position based on its current direction. If the snake eats a food pellet, the grow method increases the snake's length and places new food on the grid.
4. Collision Detection: The Snake class also handles collision detection. It checks if the snake's new position collides with itself or the boundaries of the grid. If a collision is detected, the game is halted, and a crash message is displayed.
5. Direction Control: The game captures user input to change the snake's direction. The SnakeGame class interprets keyboard inputs and updates the snake's direction accordingly, ensuring smooth and responsive gameplay.
6. Status Updates: The SnakeGrid class provides methods to update and display the game's status line. It shows the current level, score, snake length, and whether the game is paused. This keeps the player informed of their progress and the game state.
7. *Graphics Rendering*

The graphical rendering in the Snake game is managed by the SnakeGrid class. This class ensures the game state is visually represented on the screen, providing clear feedback to the player. Key functions include:

1. Drawing the Grid: The SnakeGrid class initializes the game grid and displays the horizontal line that separates the game area from the status line. The grid is updated to show the snake's position, food pellets, and status information.
2. Updating the Display: The display is updated continuously to reflect the current game state. This includes drawing the snake's movement, placing food pellets, and updating the score, level, and length of the snake. The drawSnakeBit method marks the snake's position on the grid, while clearSnakeBit removes the tail of the snake as it moves.
3. Handling Game Over: When the game ends (either by the snake crashing into itself or into the wall), the drawCrashed method displays a "*CRASHED*" message on the status line. If the player successfully pauses or exits the game, corresponding messages ("[PAUSED]" or "[DONE]") are shown to indicate the game state.
4. *User Interface*

The user interface (UI) allows players to interact with the game using keyboard inputs. Key aspects of the UI include:

1. Handling Keyboard Input: The game responds to keyboard inputs to control the snake's direction. Players use specific keys to move the snake up, down, left, or right. The input handling ensures smooth and responsive gameplay.
2. DisplayingStatusInformation**:** The game provides real-time feedback through the status line at the bottom of the screen. It displays the current level, score, snake length, and whether the game is paused or crashed. This information helps players keep track of their progress and game state.
3. Game Controls and Messages**:** Instructions for controlling the snake and game status messages (such as "PAUSED", "DONE", or "*CRASHED*") are displayed to keep the player informed. These messages ensure that the player knows the current state of the game and can make informed decisions during gameplay.
4. *Error Handling and Edge Cases*

The Snake game includes error handling to manage unexpected inputs or situations, ensuring robust performance. Edge cases, such as the snake moving out of bounds or colliding with itself, are appropriately handled.

1. Invalid Input Handling: The game gracefully manages invalid inputs, such as attempting to move the snake in the opposite direction of its current movement (which would cause an immediate collision with itself). The game ensures that the snake can only move in valid directions relative to its current heading.
2. Edge Case Management: Specific edge cases, like the snake colliding with the walls or itself, trigger appropriate game state updates. For instance, a collision results in the game ending with a "CRASHED" message displayed to the player. Additionally, when the snake successfully eats food, it grows in length, and new food is placed in an unoccupied cell, ensuring continuous gameplay without errors.

*E. Testing and Debugging*

Extensive testing and debugging were conducted to ensure the Snake game operates correctly under various scenarios. Test cases included checking grid initialization, snake movement, food placement, and user interface responses.

1. Unit Testing: Individual functions were tested to verify their correctness, particularly the game logic in the SnakeGrid and Snake classes. This included ensuring proper food placement, snake movement, collision detection, and grid updates.
2. Integration Testing: The game was tested as a whole to ensure that all components work together seamlessly. This involved testing the interactions between the Snake, SnakeGrid, and SnakeGame classes to ensure consistent and error-free gameplay.
3. User Testing: User feedback was incorporated to improve the user interface and overall gameplay experience. This included refining controls, adjusting difficulty levels, and ensuring that the game responded accurately and intuitively to player actions.

# FILE STRUCTURE

The code is organized into multiple files, each serving a specific purpose. This modular approach enhances readability and maintainability. In this section, we provide a summary of the key files in the Snake game implementation, detailing the purpose and responsibilities of each file.

1. *Main.jack file*

This file defines the Main class with the main function that serves as the entry point for the Snake game. The main function initializes the game components by creating an instance of SnakeGame and starts the game by calling the run method of SnakeGame. This setup ensures that the game environment is properly initialized and the main game loop is initiated.

1. *SnakeGame.jack file*

The SnakeGame class serves as the main controller for the Snake game. It manages the game state, handles user input, and updates the game board accordingly. The class initializes the game grid, places the snake and food, and runs the main game loop. It also handles the game's timing and calls methods to update the game state and render the graphics.

1. *Snake.jack file*

The Snake class represents the snake in the game. It manages the snake's position, length, and movement. The class includes methods for moving the snake, growing it when food is eaten, and checking for collisions with the walls or itself. The Snake class also keeps a history of positions to manage the snake's body segments.

1. *SnakeGrid,jack file*

The SnakeGrid class represents the game grid. It initializes the grid, places food, and checks the occupancy of cells. The class includes methods for drawing the grid, placing and drawing food, and marking cells as occupied by the snake. It also handles drawing the status line and game over messages.

1. *Random.jack file*

The Random class provides utility functions for generating random numbers, which are used to place food on the grid. It includes methods for generating random numbers within a specified range and managing the random seed.

1. *RandSeed.jack file*

The RandSeed class initializes and manages the random seed used by the Random class. It sets the seed value to ensure that the random number generation is predictable and repeatable, which is useful for debugging and testing.

# RESULTS

The Snake game successfully implements the core mechanics of the classic snake game and provides an engaging and interactive experience for players. The game demonstrates the feasibility of developing complex applications using Jack and Hack Assembly Language and reinforces key concepts from the Nand2Tetris curriculum. The implementation showcases the effective use of object-oriented design principles and the integration of graphical and input-handling components, resulting in a functional and enjoyable game.

A screenshot of a computer

Description automatically generated

Fig.1

A screenshot of a computer

Description automatically generated

Fig.2

A screenshot of a computer

Description automatically generated

Fig.3

A screenshot of a computer screen

Description automatically generated

Fig.4

A screenshot of a computer game

Description automatically generated

Fig.5

VI. FUTURE IMPROVEMENTS

Future enhancements for the Snake game could include implementing various additional features to enhance gameplay and user experience. These could involve introducing different difficulty levels to cater to players of varying skill levels, incorporating a timer to add a sense of urgency

# CONCLUSION

In conclusion, this paper has provided a comprehensive overview of the design, implementation, and testing of the Snake game using the Hack assembly language and Jack programming language. The Snake game serves as an excellent demonstration of applying theoretical concepts learned in the Nand2Tetris course to practical software development. By exploring the code structure, game logic, graphics rendering, and user interface aspects, this paper showcases how the Snake game project encapsulates key computer science principles, including modular design, algorithmic logic, and user interaction.

# ACKNOWLEDGMENT

The authors sincerely thank Mr. Niranjan DK for his invaluable guidance and support during the preparation of this IEEE report. His expertise and encouragement played a pivotal role in shaping our understanding of the complex topics discussed herein. His feedback and suggestions greatly enriched the content and quality of this report.

# REFERENCES

1. Nisan, N., & Schocken, S. (2005). The Elements of Computing Systems: Building a Modern Computer from First Principles. MIT Press.
2. IEEE Standards Association. (2020). IEEE Editorial Style Manual.
3. https://www.nand2tetris.org/
4. <https://github.com/idelvall/nand-mines>