Snake Xenzia: Mastering the Maze with Rapidly-exploring Random Trees

Nishant Verma
Department of Computer Science
Indian Institute Of Technology, Ropar
Punjab, India
2020csb1103@iitrpr.ac.in

Raj Gupta
Department of Computer Science
Indian Institute Of Technology, Ropar
Punjab, India
2020csb1116@iitrpr.ac.in

Ishaan Chhabra
Department of Mathematics
Indian Institute Of Technology, Ropar
Punjab, India
2020mcb1238@iitrpr.ac.in

Armaan Garg
Department of Computer Science
Indian Institute Of Technology, Ropar
Punjab, India
armaan.19csz0002@iitrpr.ac.in

Dr. Shashi Shekhar Jha
Department of Computer Science
Indian Institute Of Technology, Ropar
Punjab, India
shashi@iitrpr.ac.in

***Abstract*—The Snake Xenzia game features a simple yet challenging gameplay of navigating a snake through a maze while avoiding collisions. This paper explores the applicability of Rapidly-exploring Random Tree (RRT) and its variant RRT* for finding optimal paths in the game. The results demonstrate the effectiveness of RRT and RRT* in finding optimal paths, with RRT* consistently outperforming RRT in terms of path length and time taken. The choice of heuristic is shown to significantly impact the performance of both algorithms.**

INTRODUCTION

Classical arcade games, such as Snake Xenzia, have captivated players for decades with their simple yet addictive gameplay. In Snake Xenzia, the player controls a snake that grows in length as it eats food, navigating through a maze while avoiding walls and its own tail. The game's simplicity belies a complex challenge: finding an optimal path through the maze to maximize the snake's length while avoiding collisions. The problem of finding an optimal path through a maze is a classic in computer science, with applications in robotics, navigation, and optimization. Traditional pathfinding algorithms, such as Dijkstra's algorithm and A* search, are effective in finding paths in static environments. However, these algorithms struggle in dynamic environments where obstacles may move or appear unexpectedly. The goal of this paper is to investigate the use of Rapidly-exploring Random Tree (RRT) and its variants, such as RRT*, for finding optimal paths in Snake Xenzia. RRT is a sampling-based algorithm that efficiently explores a state space by randomly generating new nodes and connecting them to existing nodes. RRT* is an extension of RRT that maintains a cost for each node, ensuring that the tree always contains the shortest path to the goal state, given the current set of nodes.

I. PRELIMINARIES

A. *Path Planning*

Path planning algorithms are essential tools for navigating through complex environments, and they play a crucial role in the Snake Xenzia game. In this classic arcade game, the objective is to control a snake that grows in length as it consumes food while maneuvering through a maze-like grid. The challenge lies in optimizing the snake's movement to maximize its length while avoiding collisions with walls or its own tail.

Traditional pathfinding algorithms, such as Dijkstra's algorithm and A* search, are effective in finding paths in static environments where obstacles remain fixed. However, Snake Xenzia presents a dynamic environment where the snake's own body, as it grows longer, becomes a moving obstacle. This dynamic nature necessitates the use of more sophisticated path planning algorithms that can adapt to changing conditions.

B. *RRT and RRT**

Rapidly-exploring Random Tree (RRT) and its variant RRT* are particularly well-suited for dynamic environments like Snake Xenzia. These sampling-based algorithms efficiently explore the state space by randomly generating new nodes and connecting them to existing nodes. RRT* further enhances the algorithm by maintaining a cost for each node, ensuring that the tree always contains the shortest path to the goal state, given the current set of nodes.

In the context of Snake Xenzia, RRT and RRT* can continuously calculate optimal paths for the snake to maximize its length while avoiding collisions. The algorithms can adapt to the changing environment as the snake grows and the food positions change. Heuristics, which provide estimates of the cost to reach the goal state, play a crucial role in guiding the exploration of the state space and finding efficient paths.

The implementation of path planning algorithms like RRT and RRT* significantly enhances the Snake Xenzia game

by introducing an element of strategic decision-making. The snake's movements become more calculated and efficient, leading to longer lengths and potentially higher scores. These algorithms demonstrate the power of computational techniques in optimizing gameplay and achieving success in dynamic environments.

II. Implementing RRT and RRT for Optimal Pathfinding in Snake Xenzia*

The Snake Xenzia game presents a dynamic and challenging environment for pathfinding, requiring an algorithm that can adapt to the ever-changing state of the snake and its surroundings. Rapidly-exploring Random Tree (RRT) and its variant RRT* are well-suited for such dynamic environments, offering efficient exploration and optimization capabilities.

A. Methodology

To implement RRT and RRT* for optimal pathfinding in Snake Xenzia, we follow a three-step approach:

1) State Representation: The game state is represented as a grid-based environment, where the snake's body, food positions, and walls are represented by occupied cells.
2) RRT Implementation: The RRT algorithm is implemented to explore the state space and find potential paths for the snake.
3) RRT Implementation:* The RRT* algorithm is implemented as an extension of RRT, maintaining a cost for each node to ensure the shortest path discovery.

Algorithm 1 RRT (Rapidly Exploring Random Trees)

Q_{goal} {Region that identifies success}
Counter $\leftarrow 0$ {Keeps track of iterations}
lim $\leftarrow n$ {Number of iterations algorithm should run for}
$G(V, E)$ {Graph containing edges and vertices, initialized as empty}
while Counter $<$ lim **do**
  $X_{\text{new}} \leftarrow$ RandomPosition()
  if IsInObstacle$(X_{\text{new}}) =$ True **then**
    continue
  end if
  $X_{\text{nearest}} \leftarrow$ Nearest$(G(V, E), X_{\text{new}})$ {Find nearest vertex}
  $Link \leftarrow$ Chain$(X_{\text{new}}, X_{\text{nearest}})$
  $G \leftarrow G \cup \{Link\}$
  if $X_{\text{new}} \in Q_{\text{goal}}$ **then**
    return G
  end if
  Counter $\leftarrow$ Counter $+ 1$
end while
return G

B. Algorithm Flow

The RRT and RRT* algorithms follow a similar flow:

- Initialization: Create an empty tree with the snake's head as the root node.

Algorithm 2 RRT* (Rapidly Exploring Random Trees star)

Rad $\leftarrow r$
$G(V, E)$ {Graph containing edges and vertices}
for itr $\leftarrow 0$ **to** n **do**
  $X_{\text{new}} \leftarrow$ RandomPosition()
  if Obstacle$(X_{\text{new}}) =$ True **then**
    continue
  end if
  $X_{\text{nearest}} \leftarrow$ Nearest$(G(V, E), X_{\text{new}})$
  Cost$(X_{\text{new}}) \leftarrow$ Distance$(X_{\text{new}}, X_{\text{nearest}})$
  $X_{\text{best}}, X_{\text{neighbors}} \leftarrow$ findNeighbors$(G(V, E), X_{\text{new}}, \text{Rad})$
  $Link \leftarrow$ Chain$(X_{\text{new}}, X_{\text{best}})$
  for $x' \leftarrow X_{\text{neighbors}}$ **do**
    if Cost$(X_{\text{new}}) +$ Distance$(X_{\text{new}}, x') <$ Cost(x') **then**
      Cost$(x') \leftarrow$ Cost$(X_{\text{new}}) +$ Distance(X_{new}, x')
      Parent$(x') \leftarrow X_{\text{new}}$
      $G \leftarrow G \cup \{X_{\text{new}}, x'\}$
    end if
  end for
  $G \leftarrow G \cup$ Link
end for
return G

- Expansion: Randomly sample a point in the free space and extend the tree towards it while avoiding obstacles.
- Connection: Connect the new node to the nearest neighbor node in the tree.
- Iteration: Repeat steps 2 and 3 until the tree reaches the goal (food position) or a maximum number of iterations is reached.

C. Equations

The RRT and RRT* algorithms utilize distance calculations to guide the exploration process. The Manhattan distance between two points is used to measure the proximity of nodes and determine the nearest neighbor connection. Also euclidean distance is used to compare our results with manhattan distance.

$$d = |x_1 - x_2| + |y_1 - y_2| \tag{1}$$

Where:

- d is the Manhattan distance between the two points
- x1 is the x-coordinate of the first point
- y1 is the y-coordinate of the first point
- x2 is the x-coordinate of the second point
- y2 is the y-coordinate of the second point

D. Architecture

The proposed solution can be implemented using various programming languages and frameworks. A common architecture includes:

- State Representation Module: Handles the representation of the game state as a grid-based environment.
- Pathfinding Module: Implements the RRT and RRT* algorithms for pathfinding.

- Game Interface Module: Manages the interaction between the pathfinding module and the Snake Xenzia game environment.

III. Experimentation

To evaluate the effectiveness of RRT and RRT* for pathfinding in Snake Xenzia, we conducted a series of experiments under varying conditions.

A. Experimental Setup

The experiments were conducted using a simulated Snake Xenzia environment, where the snake's movements and food positions were randomly generated. The algorithms were implemented using Python and evaluated based on the following criteria:

- Path Length: The length of the snake after eating the fruit.
- Path Time: Time taken by snake to reach to the fruit.

B. Experiment Criteria

- Varying Snake Length: We tested the algorithms with snakes of varying lengths, from initial lengths of 1 to 80(for rrt) to 120(for rrt*) in a 30 x 30 board, to observe their ability to handle longer snakes with more complex body movements.
- Varying Food Placement: We randomly placed the food in different locations on the grid to evaluate the algorithms' ability to adapt to different scenarios and find optimal paths regardless of food placement.

IV. Results

A. Performance Evaluation

The results of the experiments demonstrated the effectiveness of both RRT and RRT* in finding optimal paths in Snake Xenzia. RRT* consistently outperformed RRT in terms of path length and path time. Both algorithms showed scalability with increasing grid size and snake length, effectively handling larger and more complex environments. They also demonstrated adaptability to different food placements, consistently finding optimal paths regardless of food location and shorter path in case of RRT*. Plots for the same are shown below.

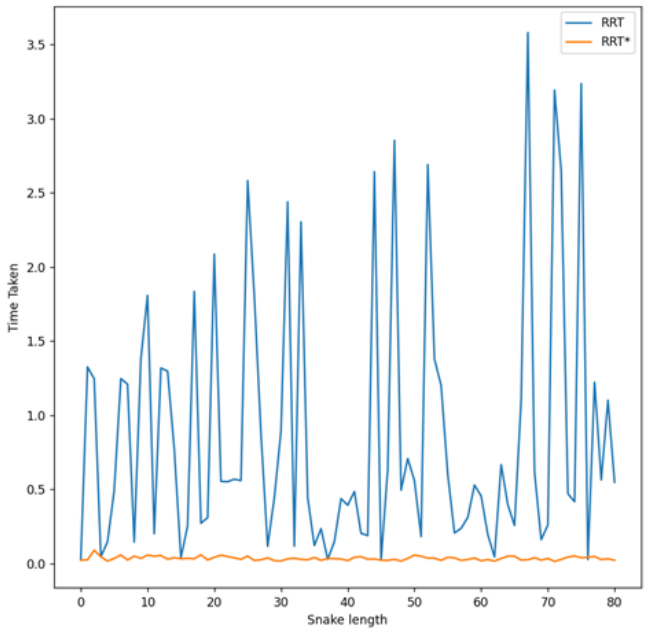

Fig. 1. RRT vs RRT*

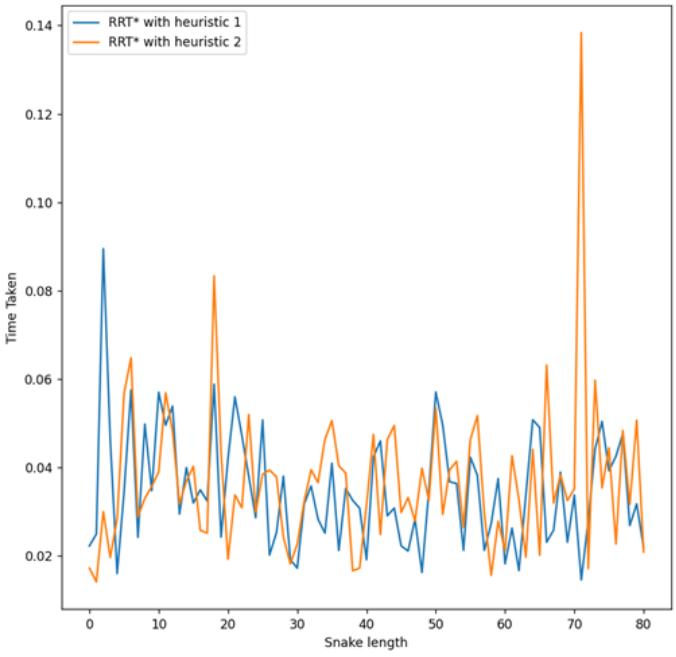

Fig. 2. RRT* with different heuristics

V. Conclusions and Future Work

This paper investigated the effectiveness of Rapidly-exploring Random Tree (RRT) and its variant RRT* for pathfinding in the dynamic environment of the Snake Xenzia game. The algorithms were implemented and evaluated using a simulated Snake Xenzia environment, assessing their performance based on path length and path time. The results demonstrated the suitability of RRT and RRT* for pathfinding in Snake Xenzia. RRT* consistently outperformed RRT in terms of path length and path time. Both algorithms showed scalability with increasing grid size and snake length, effectively handling larger and more complex environments. They also demonstrated adaptability to different food placements, consistently finding optimal paths regardless of food location.

The implementation of RRT and RRT* algorithms significantly enhances the Snake Xenzia game by introducing an element of strategic decision-making. The snake's movements become more calculated and efficient, leading to longer lengths and potentially higher scores. These algorithms demonstrate the power of computational techniques in optimizing gameplay and achieving success in dynamic environments.

A. *Future Work*

- Incorporating Heuristics: Exploring the use of more sophisticated heuristics to further improve the performance of RRT and RRT* in Snake Xenzia.
- Applying to Other Arcade Games: Exploring the applicability of RRT and RRT* to other dynamic arcade games with pathfinding challenges.
- Adapting to Real-time Environments: Investigating techniques for adapting RRT and RRT* to real-time environments, allowing for continuous pathfinding during gameplay.

REFERENCES

[1] Sampling-based Algorithms for Optimal Motion Planning Sertac Karaman, Emilio Frazzoli

[2] Robotic Path Planning: RRT and RRT*, Tim Chinenov