CSE 202 Project: Mario Kart Problem Formulation

Group Members: Akshay Gopalkrishnan, Shresth Grover, Isheta Bansal, Ronit Chougule, Balaaditya Mukundan

Introduction

Mario Kart is a popular video game that combines exciting racing mechanics with strategic decision-making on dynamic tracks. In this project, we model a Mario Kart race as a computational problem by representing the track as a weighted graph. Each section of the track is assigned a terrain type, influencing kart speed and control. Using graph algorithms to find the shortest path, we aim to determine the fastest route around the track while accounting for boosts, obstacles, and terrain variations. We hope our approach will provide a simplified yet engaging way to analyze the challenges and strategies of Mario Kart racing.

3D Grid Representation as a Graph

1. Grid Representation

The 3D environment of Mario Kart is discretized into a grid where each cell is defined by coordinates (x,y,z), representing:

- x: Length of the track (forward direction)
- y: Width of the track (left/right direction)
- z: Height/depth

Set of Grid Cells

The grid is bounded by maximum dimensions X,Y,Z, defining the set of cells: $G = \{(x,y,z)|0<x<X,0<y<Y,0<z<Z\}$

2. Terrain Type Function

Each grid cell is assigned a terrain type via a function: T: G->{Road, Grass, Boost, Obstacle ,Special,OutOfbounds}

Terrain Probability Distribution

3. Speed Modifier Function

Traversal cost is defined by: T (Road,Gras,Boost,Obstacle,Out of bound) -> R

- T(road) = 1 (base unit time)
- T(grass) = 2 (slower speed)
- T(boost) = 0.5 (faster speed)
- T(obstacle) = 10 incurs a penalty

- T(out of bound) = 10 incurs a penalty

Tile Representation:

In **Mario Kart**, the track is represented as a **graph** where each **tile** (or segment of the track) is a **node**, and the connections between adjacent tiles are **edges**. Since movement is possible in both directions along the track an **undirected graph** is used.

- **Nodes (Vertices):** Each track segment or tile. The node can store properties such as the position and tile type.
- **Edges:** Connections between 3D adjacent tiles, allowing movement. Since edges are **undirected**, movement between adjacent tiles is **bidirectional**.

Adjacency List Representation

Each tile (node) maintains a list of adjacent tiles.

Here's how it's structured:

```
track_graph = {
    1: [2, 5],   # Tile 1 is connected to Tiles 2 and 5
    2: [1, 3, 6], # Tile 2 is connected to Tiles 1, 3, and 6
    3: [2, 4, 7], # Tile 3 is connected to Tiles 2, 4, and 7
    4: [3, 8],    # Tile 4 is connected to Tiles 3 and 8
    ...
}
```

This allows for efficient traversal and decision making.

Special/Probabilistic Tiles

- In our Mario Kart graph formulation, each vertice has different properties that will change the speed the player can go at with these tiles. As a result, the properties of a tile will affect the outgoing edges from this tile. To determine the edge weights outgoing from these tiles, we will use an estimated **tile per second** metric, which represents how many tiles per second a player is moving in Mario Kart.
- For tiles like a speed up zone, a player will have a faster **tile per second** speed, so in our graph we will have different edges to represent these speeds. We want our edge weights to have low cost for high speeds and higher cost for low speeds such that when applying an algorithm like Dijkstra's we can find a path with the minimum sum of edge weights. Therefore, for edge

weights, we can use **1 / (tile per second)**, that way faster speeds have smaller weights and slower have larger edge weights.

- Below we have defined the speeds for each of the vertex types:
    - Regular tile: 1 tile per second, so these vertices will have edge weights of 1.
    - Grass tile: 0.5 tiles per second, edge weights of 2
    - Speed boost tiles: 2 tiles per second, so these vertices will have edge weights of 0.5
    - Out of bounds tiles: There are cases where a player can go off the course, in which case they will go out of bounds and be placed on the nearest land tile. In this case, any edge incoming or outgoing of an out of bounds tile will have an edge weight of five.
    - Item tile: At an item tile, there is a chance of either getting a mushroom speed up, a coin, or a non-speed boost item. A mushroom speed boost will allow for a speed of 3 tiles per second, and a coin will provide a speed boost of 1.5 tiles per second. Therefore, the expected speed at an item tile will be (3 + 1.5 + 1) / 3 = 1.833, so these tiles will have outgoing edge weights of approximately 0.55

Objective Function

The objective function quantifies the total time to complete a lap by summing the edge weights along the chosen path. These edge weights are derived from the speed (tiles per second) of the player on each tile type, converted into traversal time per edge. This ensures that faster speeds correspond to smaller edge weights (lower traversal time), aligning with the goal of minimizing total time.

Function:

For Path P, $P = [v_0, v_1, v_2, \ldots\ldots, v_k]$ where $v_0 = v_k = S$:

$$Total\_Time(P) = \sum_{i=0}^{k-1} T(v_i)$$

where T is the traversal cost of the tile.

Optimization Criteria:

A path P_1 is better than P_2 if:

$$Total\_Time(P_1) < Total\_Time(P_2)$$

Example:
For Path P, $P = [Road, Road, Boost, Grass, Road]$, the total cost is given by:

$$Total\ Cost(P) = 1 + 1 + 0.5 + 2 = 4.5\ units$$

Output Format of the Algorithm

The output of the algorithm is a sequence of tiles that represent the fastest possible lap path from the starting position back to the same position while minimizing the total traversal time. This sequence of tiles is represented as an ordered list of grid coordinates (x, y, z) which define the optimal path through the 3D track environment.

The final output is a list of tiles in the order they should be traversed to complete a lap in the shortest possible time. Each tile in the path is represented as a coordinate in the 3D grid:

$$P = [(x_0, y_0, z_0), (x_1, y_1, z_1), \dots, (x_k, y_k, z_k)]$$

where (x_0, y_0, z_0) and (x_k, y_k, z_k) are the starting and ending positions, respectively.

Along with the path, the output includes the traversal time for each tile based on its terrain properties. This ensures that we can validate the computed path against the objective function. The traversal times are represented as:

$$T(P) = [T_0, T_1, \dots, T_k]$$

where T_i is the time cost associated with moving from tile i to tile $i + 1$. The total time required to complete the lap is computed as:

$$Total\_Time(P) = \sum_{i=0}^{k-1} T(v_i)$$

This value provides the key metric for comparing different paths. The output must form a valid cycle, meaning the final position must match the initial starting position: $P_0 = P_k$