Source Code (Github Repo): https://github.com/n1k1trh/CEPWA2_2025.git
P5js simulator (for easier access):
https://editor.p5js.org/GMR_da_Pro/full/YBFgp5wKb

Overview:
This Project in P5JS is an interactive pathfinding game, simulating a grid of nodes where the aim is for the player to reach the end node (bottom-rightmost node) through a series of selecting its starting and target nodes using the Dijkstra's shortest path algorithm. The user has to avoid falling balls which have the ability to kill the user, while managing its limited energy which can be gained back by eating up the falling balls.

How the game works:
Grid contains a 12 row by 16 column display of nodes. Player is the red ball that starts at the top left-most node. Initially, the target node is the bottom rightmost node. However, using left-clicks the player can choose where his red ball is to start at and through right-clicks the player selects where his red ball is to end up at. The process of moving from the start node and end node selected by the player is determined through the Dijkstra Algorithm (which is inevitably the shortest path). If the player is able to reach the bottom rightmost node, the player wins.

p.s. User is unable to choose the bottom-rightmost node as his start node as that would be a cheat code and would be unfair. (learnt it the hard way when I gave it to my friend and he used that sneaky method haha)

As the user (determined by the red ball) moves, it loses its kinetic energy through frictional force, and when the kinetic energy left is no more, the user remains stationary and dies off. To combat this, the user must press the shift button and click on one of the falling balls from the top of the screen to gain back their lost energy. However, these falling balls also have the potential to automatically kill the user if they come into contact with it. The score at the top is based on the time it takes for the user to win - the lower the score the better it is for the user. The game also has 3 different difficulty levels (easy, medium and hard) which the user can choose initially at the homepage which is ranked based on the drop rate of the balls (how many of the balls fall from the top), the speed of the balls falling down which determines the reaction speed required from the user such that they don't die and finally the player speed which is the speed the user holds to travel between 2 distinct nodes, equal from each other. After a loss, the game goes to a screen which displays that you lost and an option to replay the game and after a win, the game returns to the original homepage to choose the difficulty. The homepage also contains information on the instructions for the user to read through.

Science Concepts Applied:

- Pathfinding in changing environments
- Brownian motion of particles which affects position of particles within a fixed radius
- Energy usage, conversion for ball bouncing and user moving
- For all of these concepts, you may refer to the main reflection for a further detailed explanation of each of these.

Cool Stuff I added:

1. The ability to see in real time the shortest-path, and how it continuously alters based on the varying position of the particle due to its brownian motion affecting it
2. It's shortest path being highlighted in yellow and once it's done, being replaced by in red
3. Adding a soundtrack whenever the ball bounces by importing it from the web to animate the sounds → **TAKE NOTE IN INSTRUCTIONS**
4. And much much more (read through main reflections) :)

Instructions/Steps needed for it to run:

- Download the mp3 track from the zip file
- Upload it to the p5js sketch file and rename it as 'bounce.mp3'
- If the sound is too loud, on line 110:
- if (bounceSound){ //&& //!bounceSound.isPlaying()) {
- U may change this line to
- if (bounceSound && !bounceSound.isPlaying()) {
- This such that the explosion sound is only played once per the ball's lifespan
- Right-click for target node
- Left-click for start node
- Shift + Click for eating up balls
- *I WAS UNABLE TO ADD THE SOUNDTRACK INTO THE ZIP FILE, HENCE I HAVE ADDED IT INTO THE GITHUB REP*

Main Reflection Qns:

Q1. This project combines the basic idea of graph theory, pathfinding algorithms which particles use and physics-based motion generation to develop an interactive & visual simulation. Do take note that the Physics concepts used in this coded game are simplified and abridged for computational and game clarity reasons rather than exact realism. The concepts which I have applied that are related to Physics are listed as such down below:

Brownian Motion: The Nodes in the main graph are subjected to positional jitter (within a specified radius, so they don't go wham :)) by applying random displacement to a frame. This estimate does not directly take into account the random probability distribution of the particles in the real world being driven by molecular collisions, but is effectively able to capture the natural variation of their positions. From a game dev point of view, it resembles how biological systems within our own bodies interact and how the "object" which could be blood is transported around these continuously-moving organs.

Player Movement and Steering: The "object" moves along the determined shortest path utilising a steering action. This steering force is calculated by the difference between the desired velocity (to the next node) and the current one in order to achieve smooth acceleration and deceleration. Fun fact: After much research, I found out that this technique is a typical trick in autonomous agent simulations as it is able to reflect complex movements without much computational power needed.

Collision and Edge Breaking: The detection of collisions consists of checking the Euclidean distance between circular drop particles and node locations line segments (the pathways) that are the edges in this graph. When edges are deleted, the object is unable to traverse them, having them redo the shortest distance algorithm to the desired node (I used dijkstra for this).

Energy System: Players have energy which is lost through frictional resistance while moving and can be regained by clicking on the falling balls.

Q2. I faced a couple of challenges while playing this game, which I will list below as well as how I managed to overcome these difficulties and fix them. **Dynamic Graph:** Since the falling balls are able to delete the edges, the graph structure will constantly change. This means that the shortest path between the object and the end node will also change, meaning that the recalculation of the shortest path must be done extremely frequently to make the game accurate, which will use a lot of computational power. Hence, I used a Set as a p.q. (priorityqueue) -> time complexity is O(vsq + e) where e is the edges and v is the number of nodes. This isn't the most efficient but is still able to work given that there are only 192 nodes in the game.

Input Handling: I enabled multiple modes to click (left-click the node to make it the starting node, right-click for the end node, and shift-click to gobble up the moving ball) which was quite complex to handle. However, changing variable names to simplify stuff made it much easier for myself to understand aswell as commenting on the necessary lines.

Smooth Movement along Discrete Graph Nodes: The object had to fluidly move between the graphs. Using the steering mechanism I had to constantly update the path coordinates of the object and fine-tune it.

Q3. For this project, the performance calculations were extremely necessary to ensure that the object does not lag (aka game-play is smooth). For the Dijkstra computation for the path, I used adjlists (adjacency lists) and terminated each algorithmic run as soon as the object hit the end node which was selected to reduce further processing. The computations for the Physics Formulae were also simplified (especially during the ball bouncing on the nodes, and for the energy lost thru friction) to simple vector arithmetic which helped to avoid doing integration or collision resolution methods. In order to save the memory, I also removed the falling balls from their arrays as soon as the exited the screen as they were redundant.

Q4. Some creative extensions were as such: the dynamic edge breaking of the graph (which resulted in its topology to be altered, adding complexity to the pathfinding), altering the difficulty levels (easy, med, hard, with diff drop rates of the balls, speed of the falling balls, and energy lost to make it adjacent to its level) and enhancing the visual aspect of the game (through the coloured stroke and the highlighted yellow showing its current path to it turning bright glowing red when the path has finished). Furthermore, the imported sound per ball bounce was also creative.

Q5. The grid is modeled as a network of *nodes* (points) connected by *edges* (lines), forming what is known as a *graph* in mathematics. Each node represents a possible position, and each edge represents a direct connection or path between two positions. **Shortest Path Algorithm:** The game brings Dijkstra's algorithm to life, showing how computers can systematically find the quickest or least costly route between two points in a network. As the player moves or as the grid changes (when paths are destroyed by falling balls), the algorithm recalculates the shortest path in real time. This dynamic visualization helps users grasp how algorithms adapt to changes in a system, and how weighted graphs (where connections can have different "costs" or lengths) are navigated efficiently. It reinforces the practical value of shortest path algorithms in fields like navigation, logistics, and network design.

Physics-Based Motion & Steering: The player's movement is not just a simple jump from node to node. Instead, it uses concepts from physics, such as velocity, acceleration, and friction, to create smooth, natural motion. This approach, inspired by how real objects move, helps illustrate principles like inertia (an object in motion stays in motion) and energy loss (due to friction). By visualizing these forces, players can intuitively understand how motion is governed by physical laws, and how these laws are simulated in computer graphics and robotics.

Collision Detection and Geometric Reasoning: The interactions between falling balls (circles) and the grid's paths (lines) introduce the basics of collision detection—a key topic in both physics and computational geometry. Detecting when a moving circle touches a line segment is a classic problem, relevant to everything from video game development to robotics and engineering simulations. By observing how collisions are handled (balls bouncing off nodes, paths being broken), users gain insight into how computers use geometry to simulate and predict physical interactions in a virtual environment1.

Dynamic Systems and Real-Time Simulation: Because the grid changes as balls fall and break connections, the visualization demonstrates how systems can evolve over time and how algorithms must adapt to these changes. This mirrors real-world scenarios, such as traffic networks where roads can be blocked or new routes opened, requiring constant recalculation of optimal paths.

Weekly Reflections:

W1:

This was super fun looking at all of their projects. One of the project i liked the most was shresth's typing game. The way he managed to replicate the moving clouds was super good and the background looked qutie good. One thing i can learn from this is to not be too stuck on certain elements like having rgb colours as my background but rather expand my reach and see how i can incorporate more natural or human-like elements into my game. I can incorporate this new-found knowledge into my wa2 by no.1 having a similar background -> since we r using physics i can now animate the background and make it related to like a solar system using high graphics or something like that insteasd making the background just blakc or white.

W2:

1. My code is stored locally on the school's laptop, hence, I am unable to send it to you.2. As of now, I'm probably at a 4; meaning that I fully understand what is going on in class and it's quite familiar to me (the concepts). The only new thing that I learnt is the Physics formulae but that too isn't that confusing, given that we have the softcopy version of the textbook and also have ChatGPT to help us J3. As of now, I don't really have a full-on version of the code, but my idea as of now is to make a simulator of the Milky Way Galaxy with the Moon and the Sun and the Earth all orbiting around.

W3:

(sorry for the late submission, as i had a viral bacteria infection that lasted 1 week)Yes, i have posted my project on the teams channel. I commented on Chen Weilian and Shaurya's project. Some cool things i learnt were as such: Using a set time step ensures that updates like movement and collisions happen at a consistent rate, regardless of the computer's speed. This keeps the simulation stable and predictable, avoiding issues where faster or slower frame rates affect object behavior.Some projects use random seeds to introduce controlled randomness. This means the simulation appears unpredictable (like random particle positions), but can still be repeated exactly—making testing easier.Visual effects, like arrows, are often used to represent forces, helping users clearly see directions or paths of movement.

Acknowledgements:

1. **https://formatter.org/ for formatting code accordingly**
2. **https://www.perplexity.ai/search/how-do-i-change-it-such-that-w-pcODIZjqSDyir_p037klhQ for help on applying Gravitational Force to the ball and small amendments here and there**
3. **https://pixabay.com/sound-effects/search/explosion/ for the mp3 audio soundtrack**
4. **https://canbayar91.medium.com/game-mechanics-2-path-finding-ab4f55c1d580 for pathfinding algorithms**
5. **My own Competitive Programming Knowledge + Physics Formulae from OneNote of CEP Y3 2025**