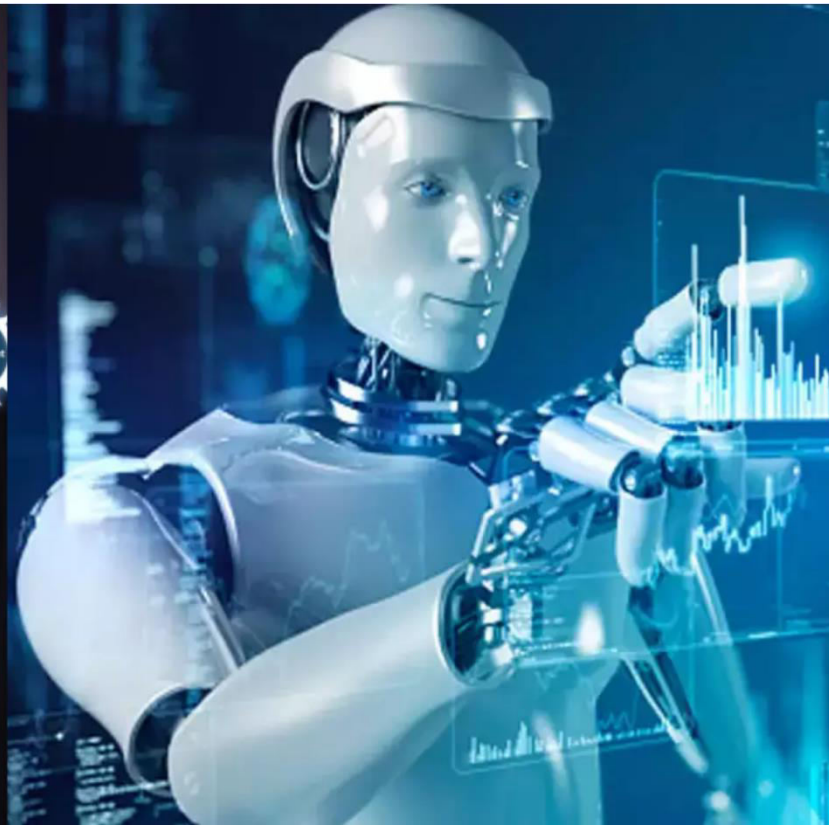
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 14, Issue 12, December 2025

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 12, December 2025

|DOI: 10.15680/IJIRSET.2025.1412074|

2D Super Mario Game using Python

Sagar Shetty A, Dr. Sanjay K S

P.G. Student, Department of Master of Computer Application, PES Institute of Technology and Management, Shivamogga, Karnataka, India

Associate Professor and Head, Department of MCA, Department of Master of Computer Application, PES Institute of Technology and Management, Shivamogga, Karnataka, India

ABSTRACT: The 2D platform game has taken steady steps in the evolution of accessible programming frameworks and game engines. The Mario-style side-scrolling game development approach structured in this paper uses Python and Pygame toward modularity, physics correctness, and responsive game behavior, featuring tile-based map loading, collision management with enemy AI pattern support integrated into interactive player mechanics for compelling gameplay experience. Proposed design measures include integrating a camera scrolling system with an event-handling module based on an object-oriented architecture to optimize performance at different levels of complexities; meanwhile rendering synchronization among movement animations coupled with real-time input handling inside the main loop for keeping movements seamless across screens. Sprite-based animation that supports physics rules combined with map-driven world building demonstrates scalable infrastructure for 2D game development as this project does. approach also highlights the importance of modular programming, reusable components and clear system flow, making the design suitable for academic learning, prototyping and further expansion into advanced game systems.

KEYWORDS: 2D platformer, Pygame, Camera scrolling, Collision detection, Game loop, Enemy AI, Sprite animation, Tile-based mapping.

I. INTRODUCTION

Because of their straightforward design and difficult gameplay elements classic platform games still have an impact on contemporary game development. It takes knowledge of physics simulation animation object interaction and real-time input processing to replicate such experiences. Because traditional game design frequently relied on tightly coupled code and manual object placement updates and scalability were challenging. These days level-editing tools and modular design enable the creation of more complex and engaging game environments. This project uses Python and Pygame to create a 2D side-scroller with a Mario theme. Using an object-oriented methodology the system incorporates character movement hit-box management collision response and scripted enemy behaviors into tile-based maps made with third-party editors. In contrast to basic prototypes this implementation places a strong emphasis on a structured game loop that frequently updates physics animations and events to maintain fluid and responsive gameplay. The primary goals are to guarantee fluid character control precise platform and object interaction detection and background element synchronization via camera scrolling. Sprite-based animation scenes improve the visual experience and Pygames event management system allows for real-time player input. This game is a useful illustration of how to create an interactive digital environment by utilizing computer science concepts like modularity state management inheritance and encapsulation. Students researchers or developers who want to investigate 2D game mechanics and expand them into more complex gameplay scenarios can adapt the system thanks to this method.

II. LITERATURE SURVEY

Character movement patterns and basic grid-based collision rules were the main focus of early 2D game development research which mainly relied on human behavior scripting [1]. Subsequent studies added structured physics layers to enhance consistency and realism when interacting with platforms [2]. In order to streamline map creation and facilitate modular level design authors in [3] investigated tile-based world construction which greatly shortened the development time for side-scrolling environments. By separating player logic enemy scripts and rendering modules using object-oriented design principles work in [4] expanded these concepts and improved game systems maintainability. The significance of camera scrolling was highlighted in [5] allowing for expansive game

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 12, December 2025

|DOI: 10.15680/IJIRSET.2025.1412074|

worlds without limiting gameplay to a static screen. Sprite sequencing was used by other researchers to improve visual fidelity by incorporating animation management frameworks [6]. Real-time event handling was introduced in studies in [7] to support dynamic interactions like environmental triggers damage systems and power-ups. Performance optimization for Python-based games was covered in research in [8] emphasizing the value of effective loops and batch rendering operations. The use of collision masks and bounding boxes for high-accuracy hit detection in platform games was covered in more detail in [9] and [10]. Full gameplay systems combining physics artificial intelligence and animation control were demonstrated in recent works such as [11] which served as inspiration for organized implementations in small-scale academic gaming projects.

III. PROPOSED METHODOLOGY

The game system is based on a structured game loop that updates physics animations input and interactions in real time to keep the gameplay flowing. The world objects enemies and player character are all implemented as modular classes with states behavior functions and sprite animations. Because level maps are made externally with a TMX map editor tile arrangement can be quickly changed without changing the source code. During runtime these maps are loaded and transformed into world objects like enemies platforms tubes and decorative elements. Rectangular bounding boxes are used for collision detection which enables precise player enemy and world tile interaction. By anchoring the viewport around the players position while preserving screen boundaries the camera module guarantees fluid horizontal scrolling. Simple rule-based logic is used to control enemy behavior allowing for movement interaction and defeat mechanics akin to those found in classic Mario games. Event triggers that are tracked inside the loop are used to handle power-ups score increases and coin animations. In order to ensure visual clarity rendering is done in layers: background platforms enemies player and UI elements. The technique combines timelines for object destruction sound playback and event sequencing to produce a cohesive and engaging gaming experience. This method places a strong emphasis on modular separation scalability and clarity across all major components.

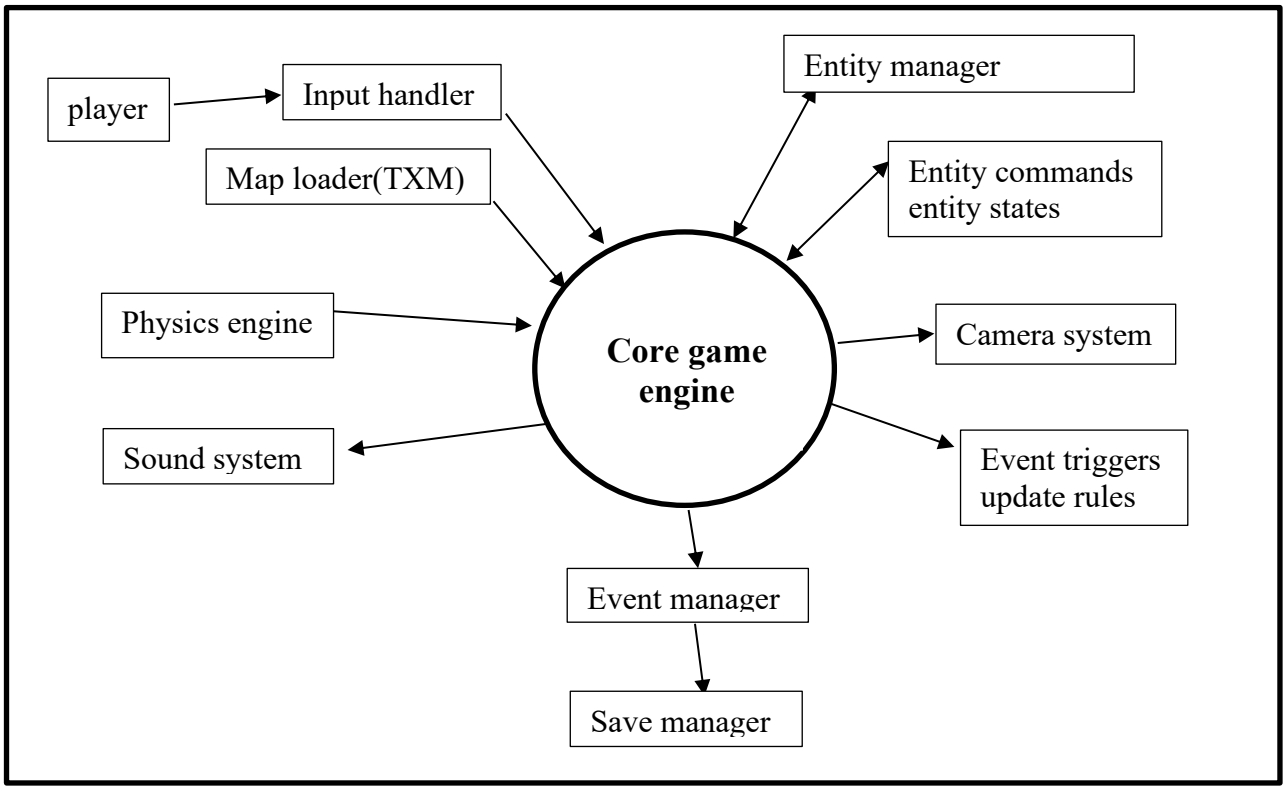


Fig. 1 Data Flow Diagram

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 12, December 2025

|DOI: 10.15680/IJIRSET.2025.1412074|

The transformation of player actions into visible on-screen gameplay follows a structured flow. Starting the game and interacting with the Main Menu is the first step in the process. The Core Game Engine receives inputs like launching a level managing the character and navigating menus. Map loading entity behavior physics updates and event responses are all coordinated by the engine which serves as the central processing system. Platforms objects and enemy locations within the game world are created by the Core using level data that is retrieved from stored TMX files. The Input Module continuously transmits keyboard commands to the Player Controller during gameplay which calculates movement and initiates interactions jumps and attacks. By contrasting player and enemy hit-boxes with map tiles the Physics Engine analyzes collisions. The Entity Manager handles power-up events environmental triggers and enemy logic. Every time the game loop is repeated the Rendering Module draws updated frames to the screen while the Camera System modifies the viewport in accordance with player movement. Animations and sound effects operate simultaneously in response to triggered events.

Algorithms Used: To guarantee predictable platforming behavior the Mario-style game employs deterministic rule-based algorithms for movement collision and enemy AI. It uses tile-indexed collision checks tuned physics multipliers and basic state machines for enemies based on position and collisions in place of complex neural nets. This design provides developers with easy debugging responsive controls and efficient execution.

1. Tilemap Loading & Layer Parsing: The game reads a Tiled *.tmx* map and loads each tile layer into two complementary structures: a 2-D collision grid and flat render lists. The loader iterates over visible layers, extracts tile images and tile IDs, and creates Platform or BGObject instances depending on layer type. Foreground tiles become collision platforms placed into the map[x][y] array so collision lookups use integer indices, while background tiles are stored in *obj_bg* for rendering only. Question-blocks (ID 22) load as a tuple of animated frames to let the platform switch images when activated. After tiles are placed, special objects (tubes, flags, spawn points) are appended and collision-only tiles are added for tubes.
2. Player Physics & Movement: Player motion blends simple Euler integration with tuned multipliers for jump control and fall behavior. Inputs set target acceleration: left/right keys adjust x_vel incrementally, while x_vel decays when no direction is pressed. Jumping sets y_vel negative and marks already_jumped to avoid repeated boosts. Gravity is applied each frame; while rising the engine uses a lower gravity multiplier if the jump button remains pressed (allowing longer jumps), and a higher fall multiplier when falling so descents feel snappy. Horizontal velocity is clamped to different caps for normal and “fast” running, and tiny residual velocities are zeroed to avoid jitter.
3. Camera & Culling: In order to avoid displaying empty space outside of the level the camera employs a constrained centering algorithm which attempts to center the viewport on the player while clamping to map boundaries. To keep the player close to the center of the screen the cameras complex_camera calculates an x offset and clamps x between 0 and the negative of (map width − window width). To maintain the viewport at ground-level depth Y is made simpler. By applying camera offsets to object rectangles the cameras apply function converts world coordinates into screen coordinates. Obj_bg mobs obj tubes and dynamic arrays (projectiles debris) are all iterated during rendering core . get_map() . get_camera() are used for each object. Apply (self) to sketch in the new position.
4. Entity AI & Collision Handling: Enemies are controlled by simple deterministic AI: each mob has a direction flag and a fixed horizontal speed which it reverses upon colliding with a platform or world boundary. With minor modifications for shell or death mechanics Goombas and Koopas both use the same basic entity behavior. When an entity moves it requests a small neighborhood of tiles from the map (a few tiles surrounding their grid coordinate) and only checks rectangle collisions against these neighbors. This collision detection is local. Bounding-box checks are used to handle player-enemy interactions: unless the player is invincible if the players feet overlap an enemy while descending the enemy dies and the player bounces otherwise the player takes damage. Similar neighborhood checks are carried out by projectiles (fireballs) which explode when tiles collide. This local tile-indexed collision results in consistent interactions that are simple to analyze and test while reducing the number of per-frame checks.
5. Event & State Management: A central Event object controls game events like death level completion and power-up transitions. When a player dies the event sets a delay flips their sprite to dead plays the death sound and updates timers. After the delay is over the map either resets or changes to a game-over state. When a team wins flag animation logic and a timed score-countdown are triggered with the remaining time gradually converting to points. For some systems state flags like in_event halt routine updates (e. g. A g. temporarily turn off AI and camera scrolling) to make scripted animations behave consistently.. Power-level changes (small ⇄ big) trigger level-up and level-down animations with separate timers and temporary invulnerability handled by unkillable flags.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 12, December 2025

|DOI: 10.15680/IJIRSET.2025.1412074|

Testing:
The development of the Mario-style 2D platform game depends heavily on software testing to make sure that every feature operates as intended and reflects the typical behavior of a traditional side-scrolling setting. Validating gameplay mechanics performance stability user control responsiveness visual consistency and collision interaction accuracy were the main goals of testing. Due to the games heavy reliance on real-time physics animation timing input handling and event sequencing each modules dependability and consistency were thoroughly examined. This ensures that players will be able to play smoothly and have confidence in the systems predictable response.

1. Unit Testing Each component was tested independently including enemy states animation switching collision handlers and player movement logic. To verify that physics updates sprite modifications and object interactions performed as intended each function was tested under normal conditions edge cases and invalid scenarios. Additionally unit tests confirmed that minor mistakes like missing assets or incorrect coordinates were handled gracefully without interfering with gameplay.

2. Integration testing ensured that the main game modules Input Processing, Physics Engine, Map Loader, Event Manager, and Rendering System worked together without conflict. This step confirmed that position updates flowed smoothly into collision checks, enemy behaviour responded accurately to the physics engine, and the camera system correctly tracked player motion. Integration tests also validated transitions between menus, loading screens, and levels, ensuring that the overall experience remained seamless.

3. In order to assess the games responsiveness visual elements clarity and ease of control a group of users were asked to participate in acceptance testing. Test players evaluated how natural the camera followed movement whether enemy encounters behaved as expected and how intuitive jumping running and interactions felt. According to feedback both novice and seasoned players found the layout and flow to be sufficiently straightforward indicating that the system is both user-friendly and captivating.

Test Cases:

Test Case ID	Description	Input	Expected Output	Result
TC-01	Validate game launch process	Click game executable	Game window opens, main menu displayed	Pass
TC-02	Handle invalid user input	Press unsupported key repeatedly	Game ignores invalid input without crashing	Pass
TC-03	Verify map loading	Select level 1-1	Tmx map loads correctly with all tiles visible	Pass
TC-04	Enemy AI movement	Enemy on flat platform	Enemy walks in set direction and reveres at obstacles	Pass
TC-05	Player platform collision accuracy	Move and jump onto platform	Player lands correctly and does not fall through tiles	Pass
TC-06	Fireball interaction	Player shoots fireball	Fireball moves, explodes on collision and removes enemy	Pass
TC-07	Camera scrolling	Move player across long map	Camera follows smoothly while maintaining boundaries	Pass

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

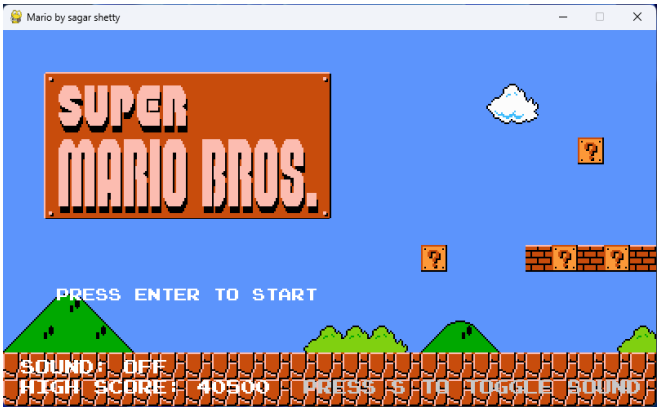
Volume 14, Issue 12, December 2025

|DOI: 10.15680/IJIRSET.2025.1412074|

TC-08	Player death event	Player collides with enemy incorrectly	Life decreases, death animation triggers and respawn begins	Pass

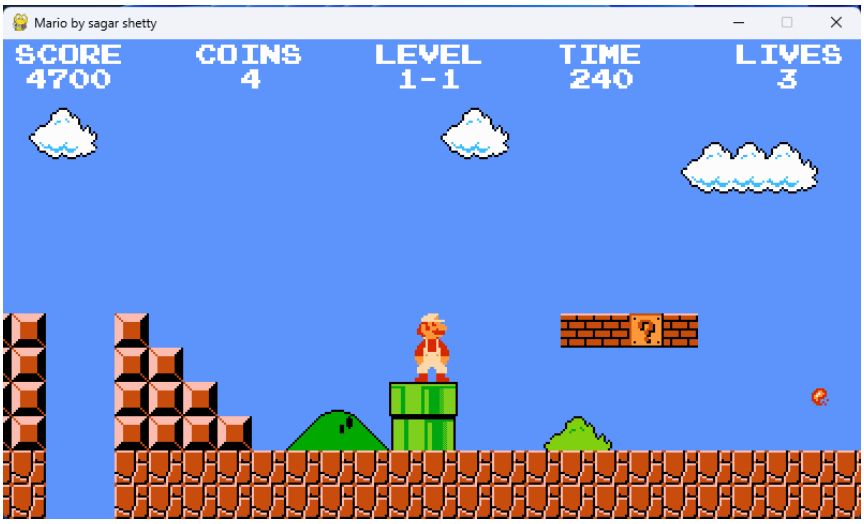
IV. EXPERIMENTAL RESULTS

The performance and functionality of the created Mario-style 2D platform game are highlighted by the experimental results. through hands-on examples of its key characteristics. Players can access essential elements like the Main Menu simple layout of the game interface. The gameplay outcomes demonstrate the Player Control Modules dependable operation. Level Loader and Gameplay Screen in the clear and



MAIN MENU

RESULT



SHOOTING MARIO

The system loads important components like the platforms enemies background tiles interactive blocks and map layout when the player enters a level. Stable frame timings responsive physics and precise sprite animations are all visible in the rendered environment. Additionally the Player Status Panel is updated in real time and displays the current world information score coins collected and remaining lives. Experiments verified that the player platforms question blocks adversaries and projectiles all handled collisions in a predictable manner. Power-up reactions fireball interactions and enemy movement patterns all matched the intended mechanics. Player movement was reliably tracked by the camera-

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 12, December 2025

|DOI: 10.15680/IJIRSET.2025.1412074|

scrolling system without any lag or jitter. These results show that the game successfully creates a consistent visually cohesive and engaging gaming experience in every scenario that was tested.

V. CONCLUSION

This study highlights how modular design tile-based mapping and object-oriented programming contribute to an effective and scalable game system by demonstrating the methodical creation of a Mario-style 2D platform game using Python and Pygame. The techniques used replace ad hoc scripting with a methodical approach that distinguishes between event processing animation control movement logic and collision handling. The systems dependability across levels with numerous enemies obstacles and interactive objects was verified through testing. The responsive physics engine rule-based enemy AI and scrolling camera all contribute to a reliable and entertaining gaming experience. Additionally without requiring significant code restructuring the architecture allows for the addition of new worlds character abilities or enemy types. All things considered the game achieves its goal of offering a fundamental platform appropriate for scholarly study and additional experimentation in 2D game development.

REFERENCES

[1] Kent, Steven L. The Ultimate History of Video Games: From Pong to Pokémon and Beyond... The Story Behind the Craze that Touched Our Lives and Changed the World. Three Rivers Press, 2001.
[2] Kohler, Chris. Power-Up: How Japanese Video Games Gave the World an Extra Life. Brady Games, 2004.
[3] Sheff, David. Game Over: How Nintendo Conquered the World. Vintage Books, 1994.
[4] Wolf, Mark J. P. The Medium of the Video Game. University of Texas Press, 2002.
[5] A. Reynolds, T. Bishop, “Foundations of 2D Game Physics and Collision Response.”
[6] Plunkett, Luke. "The Creation of Super Mario Bros." Kotaku, 18 Nov. 2015, www.kotaku.com/the creation-of-super-mario-bros-1743315637.
[7] J. Carmichael, S. Patel, “Tile-Based World Construction Using TMX and Tiled Map Editor.”
[8] K. Lin & R. Hughes, “Sprite Animation Techniques for Platformer Games.”
[9] Pygame Official Documentation | https://www.pygame.org/docs/ -This is the official documentation for Pygame. It covers everything from initializing the screen to handling sprites, sounds, and events.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details