Science, Engineering & Education, 10, (1), 2025, 11-18

Game Development in Python Through the Use of Matrices

Sava Statkov[1], Georgi Hristov[1,2*]

[1] Second English Language School "Thomas Jefferson ",
26 Trayanova Vrata St., Sofia 1408, Strelbishte Residential Complex,
[2]Sofia University "St. Kliment Ohridski", 15 Tsar Osvoboditel Blvd., Sofia 1504
Received 13 September 2025, Accepted 31 October 2025
DOI: 10.59957/see.v10.i1.2025.2

ABSTRACT

Matrices are a fundamental mathematical tool that play a central role in both theoretical research and practical programming. This report demonstrates their application through the development of a Python-based game, where every element of gameplay - from movement and collision detection to health tracking and win/loss conditions - is governed by operations on a two-dimensional matrix. The results show that a simple grid structure can successfully model an interactive environment, highlighting the flexibility of matrices beyond traditional scientific and numerical tasks. This project not only illustrates the connection between mathematics and programming but also emphasizes the educational value of using matrices as a foundation for developing logical thinking and problem-solving skills.

Keywords: matrices, Python, game development, multidimensional lists, NumPy.

INTRODUCTION

In the modern programming world, the choice of language and concepts for software development is crucial. Python is one of the most flexible and widely used programming languages, with applications in scientific computing, data processing, and video game development. It features syntax close to natural language and numerous libraries that facilitate coding and provide efficient structures for working with matrices and large datasets.

Matrices, as mathematical objects, are used in various fields of science and technology. In programming, they play a role in organizing and manipulating data, making their use important for modern programmers.

The topic of matrices in Python is relevant because they are a fundamental component of many algorithms and systems that the language supports. The report also includes a demonstration of a game developed using matrices, which are traversed via different functions.

EXPERIMENTAL

Methods

The basics of the logic behind working with matrices were reviewed from a mathematical perspective and applied to machine logic using the Python programming language. For the development of the game, the integrated development environment (IDE) PyCharm was used.

**Correspondence to:*Correspondence to Georgi Hristov, Second English Language School "Thomas Jefferson " Trayanova Vrata St., Sofia 1408, Strelbishte Residential Complex, and Sofia University "St. Kliment Ohridski", 15 Tsar Osvoboditel Blvd., Sofia 1504, e-mail: georgi.hristov@2els.com*

Development stages

The development process began with defining the concept and goal of the project: to design a simple but functional game that demonstrates how matrices can serve as the foundation of gameplay. The game world was modelled as a 6×6 two-dimensional grid, where each cell represented a wall, a collectible, a hazard, or the player's position.

Once the structure was defined, the functionality was implemented through Python functions that rendered the grid, located the player, processed movement, and managed interactions. These functions operated within a main loop that continuously updated the game state until a win or loss condition was reached.

The implementation was tested through repeated play sessions to verify correct boundary handling, collision detection, and updating of health and collectibles. This ensured the game worked reliably as a complete demonstration of matrix-based logic.

Foundation

What exactly are matrices in Python [1]? A matrix is a collection of numbers arranged in a rectangular array in rows and columns shown in Fig. 1. In engineering, physics, statistics, and graphics, matrices are widely used to represent image rotations and other types of transformations.

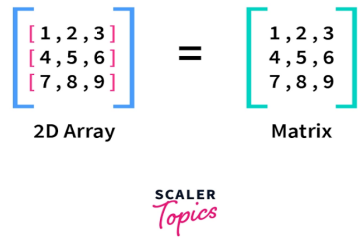

Fig. 1. Multidimensional array and Matrix.

Scientific computing and numerical methods with NumPy

NumPy is a powerful Python library that provides efficient tools for working with multidimensional arrays and numerical computations [2]. It is a core component in scientific research and data analysis, often used in linear algebra, statistics, and information processing. One of its most important functionalities is working with matrices - a special type of two-dimensional array on which basic mathematical operations can be performed. While it was not required for the implementation of this project, libraries like NumPy are widely used in more advanced game development tasks

```python
import numpy as np

# Create two matrices
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])

# Addition of matrices
result_add = matrix_a + matrix_b
print("Addition of the two matrices:\n", result_add)

# Subtraction of matrices
result_sub = matrix_a - matrix_b
print("Subtraction of the two matrices:\n", result_sub)

# Element-wise multiplication of matrices
result_mul = matrix_a * matrix_b
print("Multiplication of the two matrices:\n", result_mul)

# Matrix multiplication (dot product)
result_dot = np.dot(matrix_a, matrix_b)
print("Dot product of the two matrices:\n", result_dot)
```

```
Addition of the two matrices:
 [[ 6  8]
 [10 12]]
Subtraction of the two matrices:
 [[-4 -4]
 [-4 -4]]
Multiplication of the two matrices:
 [[ 5 12]
 [21 32]]
Dot product of the two matrices:
 [[19 22]
 [43 50]]
```

Fig. 2. Creating and operating with multidimensional arrays.

```python
import numpy as np

# Inverse of a matrix
matrix_c = np.array([[1, 2], [3, 4]])
matrix_inv = np.linalg.inv(matrix_c)
print("The inverse matrix is:\n", matrix_inv, "\n")

# Determinant of a matrix
matrix_det = np.linalg.det(matrix_c)
print("The determinant of the matrix is:", matrix_det, "\n")

# Transposing a matrix
transposed_matrix = matrix_c.T
print("The transposed matrix is:\n", transposed_matrix)
```

```
The inverse matrix is:
 [[-2.   1. ]
 [ 1.5 -0.5]]

The determinant of the matrix is: -2

The transposed matrix is:
 [[1 3]
 [2 4]]
```

Fig. 3. Inversion, determinant calculation, and transposition of a matrix.

such as pathfinding, physics simulation, and handling large-scale grid systems.

Creating matrices and performing operations on them

There are many operations we can perform with multidimensional arrays shown in Fig. 2.

Matrix inversion and determinant calculation operations are often used in linear algebra which the NumPy library also provides Fig. 3. A matrix can also be transposed (rows and columns swapped) using the transpose () function or the T attribute.

Matrices in databases and large dataset processing

Multidimensional arrays (matrices) play a key role in database management and data processing. Pandas is primarily used for data analysis, but its tabular structures parallel the logic of game grids [3]. In game development, such libraries can support tasks like analysing player data, generating procedural content, or storing level layouts.

Introduction to the game's matrix-based core

The developed game is entirely based on a two-dimensional matrix structure, which functions as the complete model of the game's world [4]. Every element of the gameplay - player position, obstacles, collectibles, hazards - is stored inside this matrix. This means that the matrix itself is the game, and all mechanics are

Fig. 4. The powerful Pandas and NumPy libraries.

simply mathematical operations that read and update specific coordinates within it.

A matrix in programming can be visualized as a grid with rows and columns. Each element in the matrix can be accessed using two indices:

- ❖ The row index (i) - counting from the top down.
- ❖ The column index (j) - counting from left to right.

For example, in a 6×6 matrix, the top-left cell is (0,0), and the bottom-right cell is (5,5).

The actual matrix that defines the initial layout of the game is in Fig. 5.

Here, each value has a precise meaning:

- ❖ 1 - Wall (cannot be passed through).
- ❖ 2 - Player's current position.
- ❖ 3 - Collectible point (worth progress toward winning).

❖ ‘G’ - Hazard (reduces health by 50 when stepped on).

❖ 0 - Empty space (safe to move into).

At the start, the player is at coordinates (1,1), where row index 1 is the second row from the top, and column index 1 is the second column from the left.

Mathematical concept of movement in the grid

Player movement is a matter of changing coordinates in this matrix [5]. If the current player position is (i, j), then a move in one of the four cardinal directions changes the coordinates as follows [6]:

❖ Up (W) → (i-1, j)

❖ Down (S) → (i+1, j)

❖ Left (A) → (i, j-1)

❖ Right (D) → (i, j+1)

This works because:

❖ Moving up means going to the previous row (i-1) without changing the column.

❖ Moving down means going to the next row (i+1).

❖ Moving left means staying on the same row but going to the previous column (j-1).

❖ Moving right means staying on the same row but going to the next column (j+1).

Before committing the move, the game checks what is at the target coordinates:

❖ If the value is 1 (wall), movement is blocked.

❖ If it is 3 (collectible), the cell is replaced with 2 (player).

❖ If it is ‘G’ (hazard), the player moves there, but health decreases by 50.

❖ If it is 0, the move simply changes the position.

Finding the player’s coordinates in the matrix

At the start of each move, the program locates the player’s position shown in Fig. 6.

Explanation of the enumerate function [7]:

❖ enumerate(grid) iterates over each row, returning both the row index i and the row’s contents.

❖ enumerate(row) iterates over each cell in the row, returning the column index j and the cell value.

❖ Once cell == 2 (player’s position) is found, the function returns the coordinates (i, j).

Rendering the matrix to the console

Since the game is text-based, the matrix is rendered as a visual grid in the console so the player can see their position and surroundings which is achieved with the following code in Fig. 7.

Explanation:

❖ Clearing the screen – os. system(cls) updates the console dynamically after moving.

❖ Symbol mapping – The dictionary symbols convert matrix values into readable symbols.

❖ Iterating rows – Each row is processed and converted into a line of symbols.

❖ Health display – Player’s health is shown after the grid.

```
grid = [
    [1, 1, 1, 1, 1, 1],
    [1, 2, 3, 0, 0, 1],
    [1, 0, 3, 'G', 0, 1],
    [1, 3, 0, 3, 3, 1],
    [1, 'G', 3, 0, 0, 1],
    [1, 1, 1, 1, 1, 1]
]
```

Fig. 5. The initial game matrix.

```
def find_player():
    for i, row in enumerate(grid):
        for j, cell in enumerate(row):
            if cell == 2:
                return i, j
    return None
```

Fig. 6. Finding coordinates.

```
def print_grid():
    os.system('cls' if os.name == 'nt' else 'clear')
    symbols = {0: '-', 1: '|', 2: 'P', 3: ':', 'G': 'G'}

    for i, row in enumerate(grid):
        padding = " " if i > 0 else ""
        print(padding + " ".join(symbols[cell] for cell in row))

    print(f"\nHealth: {health}")
```

Fig. 7. Rendering and visuals.

```
moves = {
    'w': (-1, 0),  # Up    → decrease row index
    's': (1, 0),   # Down  → increase row index
    'a': (0, -1),  # Left  → decrease column index
    'd': (0, 1)    # Right → increase column index
}
```

Fig. 8. Directions and key binds.

Movement implementation in the matrix

The move_player() function is the core mechanic that governs how the player navigates the grid-based world. Its logic revolves around updating the two-dimensional matrix that represents the game map, where each cell may contain a wall, a collectible, a hazard, or empty space i.e., a discrete grid world state representation used widely in the literature [8, 9]

When the player inputs a movement command (W, A, S, or D), the function applies the corresponding direction vector to the player's current coordinates to calculate a new target position. The consequences of stepping onto that position - whether the move is blocked, a collectible is gathered, or health is reduced by a hazard - follow the interaction rules already outlined in the Collision and Interaction Logic section.

Finally, the matrix is updated by setting the player's old cell to 0 (empty) and marking the new coordinates with 2 (player). This ensures the grid always reflects the current state of the game after each valid move [10].

Direction to coordinate mapping

The game uses a dictionary shown in Fig. 8 to define how each keypress changes the player's coordinates.

Collision and interaction logic

Once the candidate coordinates are calculated, the function inspects the target cell in the matrix and checks what will happen which is achieved by the code written in Fig. 9.

Explanation of how the player interacts with the environment [11]:

- ❖ Checks if the target cell is not a wall (1) before moving.
- ❖ If it is a collectible (3), decreases the counter of remaining items.
- ❖ If it is a hazard ('G'), reduces player health by 50.
- ❖ Updates the matrix by clearing the old position and setting the new one as the player (2).

Game loop and execution flow

The central control structure of the game is the main game loop, which repeatedly updates

```
if grid[new_x][new_y] != 1:
    if grid[new_x][new_y] == 3:
        global dots_remaining
        dots_remaining -= 1
    elif grid[new_x][new_y] == 'G':
        health -= 50
        print("-50 HP")

    grid[x][y] = 0
    grid[new_x][new_y] = 2
```

Fig. 9. Interactions of the player with the environment.

```
health = 100
dots_remaining = sum(row.count(3) for row in grid)

while dots_remaining > 0 and health > 0:
    print_grid()  # Display the current matrix
    move = input("Move (WASD): ").strip().lower()
    if move in moves:
        move_player(move)  # Process movement
        dots_remaining = sum(row.count(3) for row in grid)

print_grid()
if health <= 0:
    print("Game Over! You lost all your health!")
else:
    print(f"Congratulations! You collected all the dots with {health} HP left!")
```

Fig. 10. Implementation of the Game Loop.

the game state until a win or loss condition is reached. This loop integrates all the functions described in the previous sections - rendering the grid, receiving player input, moving the player, and checking the status of the game The use of such a continuous input-update-render pattern is a well-established game design construct, ensuring that the gameplay feels dynamic and responsive [12].

The purpose of the game loop is to ensure that the player experiences the game as an ongoing interactive process rather than a sequence of independent function calls shown in Fig. 10. Its primary responsibilities are:

❖ Display the current state of the matrix in the console.

❖ Request input from the player (W, A, S, or D).

❖ Interpret and process this input into a change of position.

❖ Update the game state, including the number of collectibles and the player's health.

❖ Check whether the game should continue or if a win/loss condition has been met.

Explanation

❖ Initialization: The variables health and dots_remaining track the current status of the player and the game.

❖ Condition Check: The while loop continues as long as the player still has health and there are collectibles left.

❖ Game State Update: After each valid move, the function recalculates how many collectibles remain by scanning the matrix.

❖ Termination: The loop ends once one of the two end conditions is reached. The program then outputs a victory or defeat message.

The ultimate goal of the game is to collect all the available collectibles on the map while avoiding hazards and keeping the player's

```
print_grid()
if health <= 0:
    print("Game Over! You lost all your health!")
else:
    print(f"Congratulations! You collected all the dots with {health} HP left!")
```

Fig. 11. Win | Loss conditions.

health above zero. These end conditions define the outcome of the game and ensure that every play session has a clear resolution. In code this is achieved by making two conditions shown in Fig. 11.

Win condition

The player wins the game when all collectibles have been gathered.

❖ Collectibles are represented in the matrix with the value 3.

❖ Each time the player moves onto a collectible cell, it is replaced with the player's symbol, and the collectible counter decreases.

❖ The win condition is satisfied when this counter reaches zero.

Loss condition

The player loses the game when their health is fully depleted.

❖ Hazards are represented by the symbol 'G'.

❖ Stepping on such a cell reduces the player's health by 50 points.

❖ Once the health variable reaches zero or less, the game loop ends and displays a "Game Over" message.

RESULTS AND DISCISSION

Results

The game successfully demonstrates the application of matrices as the central structure of a complete interactive system. Each mechanic - from movement to collision detection and win/loss conditions - is implemented through simple coordinate manipulations within the matrix. This confirms that matrices are not only theoretical tools but also highly effective for modelling and managing dynamic environments.

Possible improvements

Although the game already demonstrates the use of matrices in Python, several enhancements could expand its functionality and educational value.

❖ Larger Grids: Increasing the matrix size (e.g., to 10×10 or 20×20) would create more complex mazes and richer gameplay.

❖ New Hazards and Power-Ups: Introducing additional cell types such as traps, healing items, or temporary boosts that would diversify player choices [13].

❖ Multiple Levels: Linking several matrices into progressive levels would extend the game's scope while reusing the same core logic.

❖ Graphical Interface: Replacing the console with a graphical display (using Pygame, Tkinter or other GUI libraries) would improve usability while retaining the matrix as the underlying model [14].

❖ Save/Load Functionality: Saving the current matrix, health, and collectibles to a file would enable longer and more flexible play sessions.

CONCLUSIONS

This report demonstrated how matrices, a fundamental mathematical structure, can be applied in programming not only for abstract problems but also for building interactive systems. Through the development of a Python-based game, it was shown that movement, collision detection, health management, and victory conditions can all be modelled within

a two-dimensional grid. The project highlights both the educational value and the practical versatility of matrices, proving that they are not only theoretical constructs but also powerful tools for real-world applications.

Acknowledgment

The author would like to thank his teacher, Georgi Hristov, for the guidance and support throughout the preparation of this report and acknowledges the University of Chemical Technology and Metallurgy for its role in fostering research initiatives.

REFERENCES

1. K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition," 2015.
2. H.V. Henderson, "Numerical Computing with NumPy," McGill University, 2007. https://www.cs.mcgill.ca/~hv/articles/Numerical/numpy.pdf., Accessed 17.08. 2025.
3. Pandas Developers, "Intro to Data Structures," pandas.pydata.org. https://pandas.pydata.org/docs/user_guide/dsintro.html., Accessed 21.08.2025.
4. Arcade Academy, "Array Backed Grids," Arcade Academy. https://learn.arcade.academy/en/latest/chapters/28_array_backed_grids/array_backed_grids.html., Accessed 17.08.2025.
5. Photon GameDev, "Basic Grid-Based Movement," WordPress, 2013. https://photongamedev.wordpress.com/2013/07/26/basic-grid-based-movement/, Accessed 20.08.2025.
6. Stack Overflow Community, "How to move along a 2D grid in Python," Stack Exchange Inc., 2019. https://stackoverflow.com/questions/59108529/how-to-move-along-a-2d-grid-in-python., Accessed 20.08.2025
7. Real Python, "Python Enumerate," Real Python, 2020. https://realpython.com/python-enumerate/., Accessed 20.08.2025.
8. R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, 2nd ed. ed., Cambridge, MA: MIT Press, 2018, p. 145-160.
9. M.O. Riedl, A Python Engine for Teaching Artificial Intelligence in Games (GAIGE), 2015. https://arxiv.org/abs/1511.07714
10. M.-A. Côté, TextWorld: A Learning Environment for Text-Based Games, 2018. https://arxiv.org/abs/1806.11532
11. A.K.T. Varga, A Simplified Pursuit-Evasion Game with Reinforcement Learning, Periodica Polytechnica Electrical Engineering and Computer Science, 65, 2, 2021.
12. Game Programming Patterns, "Game Loop Pattern," gameprogrammingpatterns.com, 2014, https://gameprogrammingpatterns.com/game-loop.html, Accessed 23.08.2025.
13. M. Khalil, M. Ebner, M. Kopp, Analysing Gamification Elements in Educational Environments Using an Existing Gamification Taxonomy, 2020, https://arxiv.org/abs/2008.05473
14. W. McGugan, Beginning Game Development with Python and Pygame: From Novice to Professional, Berkeley, CA, 2007, 23-309.