PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA UNIVERSITY EL-OUED
DEMOCRATIC AND POPULAR ALGERIAN REPUBLIC
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
ECHAHID HAMMA LAKHDAR UNIVERSITY OF EL-OUED
FACULTY OF TECHNOLOGY

# Master's dissertation

**In order to obtain a diploma of an Academic Master**

**Department: Electrical Engineering**

**Specialty: Telecommunication Systems**

**Theme**

## Development of a simple arabic word game with smart hints using python/pygame

**Prepared by:**
Nour Elhouda Gharib
Rayane Ghrissi

**Supervised by:**
Nour El imane Bellili

| | | | |
|---|---|---|---|
| **President** | **GHENDIR SAID** | **M.C.A** | **El-Oued University** |
| **Examiner** | **TEDJANI AMINA** | **M.A.A** | **El-Oued University** |
| **Supervisor** | **BELLILI NOUR ELIMANE** | **M.C.A** | **El-Oued University** |

**University year:** 2025/ 2026


بسم الله الرحمن الرحيم

# إهداء

الحمد لله حباً وشكراً، الذي يسّر البدايات وبلّغنا النهايات، والقائل في محكم تنزيله:

(وَآخِرُ دَعْوَاهُمْ أَنِ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِين)

أهدي ثمرة هذا التخرج

إلى نفسي الطموحة: التي بدأت بالوعد وانتهت بالنجاح

إلى والدي العزيز: مَن أحمل اسمه فخراً، ومَن مهد لي طريق العلم بصبره؛ ها قد وفيت بوعدي

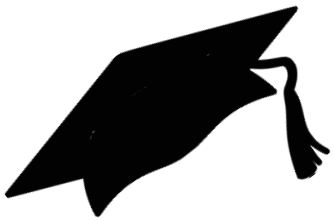
إلى والدتي الغالية: الجسر الصاعد بي إلى الجنة، مَن علمتني الأخلاق وساندتني في كل لحظات ضعفي

إلى زوجي ورفيق دربي: سندي ومُلهمي الذي زرع الإصرار في داخلي وكان الظل لهذا النجاح

إلى ملاكي الصغير "ريحان": مَن كانت براءتها تمنحني القوة وتنسيني سهر الليالي.

أخيراً مَنْ قال أنا لها "نالها" وأنا لها إن أبت رغماً عنها أتيتُ بها، ما كنت لأفعل لولا توفيق من الله، ها هو اليوم العظيم هنا، اليوم الذي أجريت سنوات الدراسة الشاقة حالمةً فيها حتى توالت بمنّه وكرمه لفرحة التمام، الحمد لله الذي به خيراً وأملاً واغرقنا سروراً وفرحاً ينسيني مشقتي

**Nour Elhouda Gharib**



# إهداء

بسم الله الرحمن الرحيم

**فَرِحِينَ بِمَا أتاهُمُ اللَّهُ مِنْ فَضْلِهِ**

هذه اللحظة لا تورث ولا تشترى، لقد دفعت ثمنها عمري وشبابي، سهري، خوفي، ودموعي، هي لهفة الوصول، ودمعة الحزن، فرحة العمر هذه هي النهاية السعيدة لبدايات أعظم.

أهدي تخرجي إلى من كل ما أملك، إلى العمود الثابت في عثرات أيامي، إلى من في الحياة حياة، إلى من بذلت جهد السنين من أجلي "أمي".

كما أهدي تخرجي إلى من وقفوا بجانبي في السراء والضراء، ولكل روح عانقتني بالحب والدعوات. شكراً لكل من كان في طريقي، استندت عليه أو مر عابراً، شكراً لكل من علمني كلمة أو مهارة

.

**Rayane Ghrissi**

# شكر وتقدير

كل الشكر والتقدير للأستاذة نور الإيمان بليلي إشرافها الكريم على هذا المشروع، وعلى ما قدمته من نصائح علمية قيمة ومساندة مستمرة كان لها الفضل الكبير في تجاوز الصعوبات وإنجاز هذا البحث

## Abstract

This dissertation aims to develop an interactive learning environment to enhance technical terminology among students through the design and implementation of a serious game called TechnoQuest. The study begins by establishing a theoretical foundation in Python and Object-Oriented Programming (OOP) principles as essential tools for system construction. It then explores the concept of Serious Games and their role in interactive learning using the Pygame library. The work concludes with the practical implementation phase, describing the game architecture and the multi-level hint system, while addressing technical challenges such as Arabic language support and data management via JSON.

**Keywords:** Python, Pygame Library, Serious Games, Educational Word Games, Interactive Learning, Object-Oriented Programming (OOP), Intelligent Hint System

## Résumé

Ce mémoire vise à développer un environnement d'apprentissage interactif pour renforcer la terminologie technique chez les étudiants à travers la conception et la mise en œuvre d'un jeu sérieux nommé TechnoQuest. L'étude commence par l'établissement d'une base théorique sur le langage Python et les principes de la Programmation Orientée Objet (POO) en tant qu'outils essentiels à la construction du système. Elle explore ensuite le concept des Jeux Sérieux et leur rôle dans l'apprentissage interactif en utilisant la bibliothèque Pygame. Le travail se conclut par la phase d'application décrivant l'architecture du jeu et le système d'indices progressifs, tout en traitant les défis techniques tels que le support de la langue arabe et la gestion des données via JSON.

**Mots-clés :** Python, Bibliothèque Pygame, Jeux sérieux, Jeux de mots éducatifs, Apprentissage interactif, Programmation Orientée Objet, Système d'indices intelligent

## الملخص

تهدف هذه المذكرة إلى تطوير بيئة تعليمية تفاعلية لتعزيز المصطلحات التقنية لدى الطلاب، وذلك من خلال تصميم وتنفيذ لعبة جادة تدعى Techno Quest. تبدأ الدراسة بتأسيس قاعدة نظرية حول لغة Python ومبادئ البرمجة كائنية التوجه (OOP) كأدوات أساسية لبناء النظام، ثم تنتقل لاستكشاف مفهوم الألعاب الجادة ودورها في التعلم التفاعلي باستخدام مكتبة Pygame. ويختتم العمل بالجانب التطبيقي الذي يصف هندسة اللعبة ونظام التلميحات المتدرج، مع معالجة التحديات التقنية كدعم اللغة العربية وإدارة البيانات عبر JSON.

**الكلمات المفتاحية:** بايثون، مكتبة بايجيم، الألعاب الجادة، ألعاب الكلمات التعليمية، التعلم التفاعلي، البرمجة كائنية التوجه، نظام التلميحات الذكي

# List of Figures

Figure1.1: Python 3.12 Setup Interface with Default Options. ..... 8
Figure1.2: Python 3.12 IDLE Console Interface. ..... 8
Figure1.3: PyCharm IDE Logo ..... 9
Figure1.4: Visual Studio Code (VS Code) ..... 10
Figure1.5: High-level file handling in Python ..... 10
Figure1.6: Low-level file handling in C++ ..... 11
Figure1.7: Pillars of OOP ..... 11
Figure1.8: Abstract Class Example (Animal and Dog) ..... 12
Figure1.9: Encapsulation Example (Animal and Dog) ..... 13
Figure 1.10: Inheritance Example (Animal and Dog Classes) ..... 13
Figure 1.11: Polymorphism Example (Animal, Dog and Cat) ..... 14
Figure1.12: NumPy logo ..... 17
Figure 1.13NumPy library example ..... 18
Figure1.14: Panda's logo ..... 18
Figure1.15 :Pandas library example ..... 19
Figure1.16: Matplotlib logo ..... 19
Figure1.17: Matplotlib library example ..... 20
Figure 1.18: Data graph representation ..... 20
Figure2.1: Implementation of a Memory Puzzle Game using Pygame [55] ..... 31
Figure2.3: Implementation of a Wormy Game using Pygame[55] ..... 32
Figure2.4: Player navigates through the map [63] ..... 35
Figure2.5: Quizzes for testing the player's knowledge during the game[63] ..... 36
Figure2.6: A 9x9 Sudoku puzzle in its initial state, illustrating the grid with some pre-filled numbers and empty spaces that the player needs to fill [64]. ..... 36
Figure2.7: Implementation of a Hangman Game using Pygame[50] ..... 39
Figure2.8: Implementation of a Wordle Game using Pygame [50] ..... 40
Figure2.9: Levels of the Hint System in Sudoku Puzzles [86] ..... 43
Figure 3.1: TechnoQuest Architecture Modules Layout in PyCharm IDE. ..... 50
Figure 3.2: TechnoQuest Use Case Diagram ..... 51
Figure 3.3: TechnoQuest Class Diagram ..... 52
Figure 3.4: TechnoQuest Sequence Diagram ..... 53
Figure 3.5: The Main Menu Interface of TechnoQuest. ..... 54
Figure 3.6: The Gameplay Screen Interface of TechnoQuest. ..... 55
Figure 3.7: The Win Screen Interface of TechnoQuest. ..... 56
Figure 3.8: Arabic support and correct answer validation. ..... 63
Figure 3.9: Initial score state before hint deductions. ..... 63
Figure 3.10: The First-Level Hint Activation Interface. ..... 64
Figure 3.11: The Second-Level Hint Activation Interface. ..... 64
Figure 3.12: The Thrid-Level Hint Activation Interface. ..... 65
Figure 3.13: "Easy" level unlocked. ..... 65

# List of Figures

Figure 3.14: "Easy" and "Medium" levels unlocked. ..... 65
Figure 3.15: All difficulty levels unlocked. ..... 66

## List of Table

Table 1.1: Key Milestones in Python’s Evolutionary Timeline ..... 6
Table 3.1: TechnoQuest System Test Cases..... 61

# Table of contents

**إهداء** ..... **3**

**شكر وتقدير** ..... **5**

**Abstract** ..... **6**

**Résumé** ..... **6**

**الملخص** ..... **6**

**List of Figures** ..... **7**

**List of Table** ..... **9**

**Table of contents** ..... **10**

**Nomenclature** ..... **14**

**General Introduction** ..... **1**

**Chapter 1: Programming Foundations**

Introduction ..... 5

1.1 Overview and Historical Evolution of Python ..... 6

1.1.1 The Genesis of Python ..... 6

1.1.2 Python 2.x vs. Python 3.x ..... 7

1.1.3 Design Principles ..... 7

1.2 Environment Setup ..... 7

1.2.1 Python Installation ..... 7

1.2.2 Python Development Environments ..... 9

1.2.3Python High Low Language ..... 10

1.3 Object-Oriented Programming (OOP) Paradigms ..... 11

1.3.1 Abstraction ..... 12

1.3.2 Encapsulation ..... 12

1.3.3 Inheritance ..... 13

1.3.4 Polymorphism ..... 14

1.4 Python Libraries and Ecosystems ..... 14

1.4.1 Modules ..... 15

1.4.2 Packages ..... 16

1.4.3 Many libraries ..... 17

Conclusion ..... 21

**Chapter 2: Specialized Domain**

Introduction ..... 23
2.1 Computer Games ..... 24
2.1.1 Definition of Computer Games ..... 24
2.1.2 Components of Computer Games ..... 24
2.1.3 Educational Benefits of Computer Games ..... 25
2.2 Serious games ..... 25
2.2.1 Definition of Serious Games ..... 25
2.2.2 Applications of Serious Games in Education ..... 26
2.3 Word Games ..... 26
2.3.1 Definition of Word Games ..... 26
2.3.2 Cognitive and Linguistic Benefits of Word Games ..... 26
2.4 The Pygame Library ..... 27
2.4.1 Introduction to Pygame ..... 27
2.4.2 Installing Pygame ..... 27
2.4.3 Basic Window Creation in Pygame ..... 28
2.4.4 Structure of the Game Loop in Pygame ..... 28
2.4.5 Event Handling in Pygame ..... 29
2.5 Game Development with Pygame ..... 30
2.5.1 Game Development Process Using Pygame ..... 30
2.5.2 Classification of Games Developed via Pygame ..... 30
2.5.3 Advantages and Constraints of Pygame in Development ..... 33
2.6 Serious Game with Pygame ..... 33
2.6.1 Framework for Developing Serious Games in Pygame ..... 33
2.6.2 Integrating Educational Design Principles ..... 34
2.6.3 Review of Existing Serious Games Developed with Pygame ..... 35
2.7 Word Games with Pygame ..... 37
2.7.1 Design Considerations for Pygame-Based Word Games ..... 37
2.7.2 Game Logic, String Manipulation, and Mechanics ..... 37
2.7.3 User Interface (UI) and Visual Design ..... 38
2.7.4 Examples of Word Games Developed with Pygame ..... 38
2.8 Intelligent Hint Systems in Educational Games ..... 40
2.8.1 Concept of Hint Systems ..... 40
2.8.2 Adaptive and Intelligent Hints ..... 41

2.8.3 Role of Hints in Improving Learning ..... 41
2.8.4 Implementation of Hint Systems in Game Development Environments ..... 42
2.8.5 Hint Systems in Games ..... 43
Conclusion ..... 44

**Chapter 3: Engineering and Implementation**

Introduction ..... 46
3.1Project Overview "TechnoQuest" ..... 47
3.1.1 Game Concept ..... 47
3.1.2 Educational Goals ..... 47
3.1.3 Target Users ..... 47
3.1.4 Gameplay Structure & Mechanism ..... 48
3.1.5 Functional Requirements ..... 48
3.2 System Design ..... 49
3.2.1 Architecture ..... 49
3.2.2 UML Overview ..... 50
3.3 User Interface ..... 54
3.3.1 Main Menu Interface ..... 54
3.3.2 Gameplay Screen ..... 54
3.3.3 Win Screen ..... 55
3.4 Implementation ..... 56
3.4.1 Development Environment and Tools ..... 56
3.4.2 Arabic Text Rendering and Logic ..... 56
3.4.3 Data Persistence ..... 57
3.4.4 Intelligent Hint System ..... 58
3.5 Algorithms and pseudo-code ..... 58
3.5.1 Grid Positioning and Centering Algorithm ..... 58
3.5.2 Word Validation and Cell Locking Algorithm ..... 59
3.5.3 Progression and Level Unlocking Algorithm ..... 60
3.5.4 Scaffolding Hint Algorithm ..... 61
3.6 Testing and Evalution ..... 61
3.6.1 Test Cases ..... 61
3.6.2 Performance ..... 66
3.7 Discussion ..... 66
3.7.1 Strengths ..... 66

3.7.2 Limitations....67
3.8 Future Improvements....67
Conclusion....68
**General Conclusion....69**
**References....71**

## Nomenclature

**API** : Application Programming Interface

**CPU** : Central Processing Unit

**CWI** : Centrum Wiskunde & Informatica

**FPS** : Frames Per Second

**GPU** : Graphics Processing Unit

**GUI** : Graphical User Interface

**IDE** : Integrated Development Environment

**IDLE** : Integrated Development and Learning Environment

**JSON** : JavaScript Object Notation

**OOP** : Object-Oriented Programming

**PEP** : Python Enhancement Proposals

**OOP** : Object-Oriented Programming

**RGB** : Red, Green, Blue

**RTL** : Right-to-Left

**UI** : User Interface

**UML** : Unified Modeling Language

**VS Code** : Visual Studio Code

**WTI** : Word-to-Text Integration

# *General Introduction*

In the modern computing era, Python has emerged as a preeminent force, renowned for its exceptional balance between computational power and a design philosophy that prioritizes human readability [1]. The transition to Python 3.x further modernized its capabilities, making it an ideal choice for developing sophisticated digital solutions [1]. Within the realm of software engineering, Object-Oriented Programming (OOP) stands as a fundamental paradigm, enabling developers to achieve modularity and scalability through concepts such as abstraction, inheritance, and encapsulation [2].

Despite rapid technological advancement, a significant challenge remains in bridging the gap between complex technical terminology and effective learner comprehension. Traditional educational methods often struggle to create an engaging environment, leading to a disconnect between theoretical knowledge and its practical application [3]. This issue is particularly evident in software environments that lack native support for right-to-left (RTL) languages like Arabic, which complicates the development of interactive educational tools tailored for diverse linguistic backgrounds [4].

As a solution to this problem, this research presents "Techno Quest", a serious educational word game designed to simplify technical terminology. Built on the principle that digital games possess immense potential to enhance literacy and cognitive development, this project merges gaming engagement with instructional precision [3]. The game features an "Intelligent Hint System" that employs a pedagogical hierarchy ranging from initial letter cues to semantic explanations to reinforce long-term memory and conceptual understanding [5].

To implement this system, the Pygame library was utilized due to its robust capabilities in managing 2D game engines, real-time event handling, and the core "Game Loop" [6]. The development process adhered to software engineering standards, utilizing UML diagrams (such as Class and State diagrams) to ensure organized code architecture and efficient management of in-game objects [2], [7]. Furthermore, custom algorithmic solutions were implemented to handle Arabic text reshaping and RTL orientation, ensuring a seamless bilingual user experience [7].

This work aims to demonstrate the efficacy of integrating software engineering principles with educational media. To achieve this, the dissertation is structured as follows:

**Chapter 1**: Programming Foundations

This chapter transitions from the general concepts of the Python environment to advanced levels of Object-Oriented Programming (OOP). It is an essential component to demonstrate the mastery of the correct technical tools and the construction of a robust, scalable software system.

**Chapter 2**: Specialized Domain

The research moves from general programming to the specialization of "Serious Games" and the Pygame library. This highlights the academic and methodological aspects of the dissertation ’s field, illustrating how games can be employed as effective educational tools.

**Chapter 3**: Engineering and Implementation

This chapter represents the "heart of the dissertation" as it explains how programming concepts were integrated with graphical tools to produce the "TechnoQuest" project. It focuses on innovative technical solutions such as the smart hint system, Arabic language support, and database management.

# *Chapter 1: Programming Foundations*

## Introduction

Python stands as a preeminent force in the modern computing era, renowned for its exceptional balance between power and simplicity [8], [9]. This exploration begins with an overview and historical evolution, tracing the genesis of Python back to its creation by Guido van Rossum, who envisioned a language that prioritizes human readability [8]. A defining moment in this journey is the transition between Python 2.x and Python 3.x, a move that refined the language's internal consistency and modernized its capabilities for the digital age [9]. At the heart of this success are Python's core design principles, which advocate for explicit, simple, and "Pythonic" solutions to complex problems [10].

To translate these principles into reality, a proper environment setup and configuration is vital. This involves a seamless Python installation and effective path management to ensure the interpreter is accessible across various operating systems [11]. Furthermore, developers rely on diverse development ecosystems, including advanced IDEs and virtual environments, to maintain project integrity and workflow efficiency [11], [12]. Architecturally, Python is built upon object-oriented programming (OOP) paradigms. By mastering abstraction, encapsulation, inheritance and code reusability, along with polymorphism and dynamic typing, developers can achieve a sophisticated technical implementation that is both flexible and scalable [10], [13]. Finally, the true strength of the language is magnified by its extensive libraries and ecosystems. This includes the robust standard library architecture, streamlined package management via pip, and a vast array of application-specific libraries that empower domains such as artificial intelligence and web development [12].

## 1.1 Overview and Historical Evolution of Python

### 1.1.1 The Genesis of Python

The origins of Python date back to the late 1980s, specifically in 1989, when the Dutch programmer Guido van Rossum began developing it at the Centrum Wiskunde & Informatica (CWI) in the Netherlands [8]. According to the historical records of the language's development, Python was originally conceived as a "hobby" programming project to keep van Rossum occupied during the Christmas week. The primary technical motivation was to create a successor to the ABC language which focused on ease of use but lacked the extensibility needed for system administration [8].

From its inception, the language was designed to emphasize code readability and simplicity, allowing programmers to express concepts in fewer lines of code than many other major languages [9]. Furthermore, the name "Python" was inspired by the British comedy series "Monty Python's Flying Circus," reflecting the creator's desire for a unique and slightly mysterious identity for his new project [8].

Table 1.1: Key Milestones in Python's Evolutionary Timeline

| Release Year | Version | Technical Evolution | Source |
|---|---|---|---|
| **1991** | 0.9.0 | Inception: Introduced classes with inheritance, exception handling, and core functions. | [8],[10] |
| **2000** | 2.0 | Maturation: Added list comprehensions and cycle-detecting garbage collection. | [8] |
| **2004** | PEP 20 | Philosophy: Formalized The Zen of Python (readability and simplicity). | [9] |
| **2008** | 3.0 | Refinement: Unified string/Unicode types; removed redundant features (Incompatible with 2.x). | [8],[10] |
| **2020-Present** | 3.10 + | Modernization: Structural Pattern Matching and optimized Python interpreter performance. | [10] |

### 1.1.2 Python 2.x vs. Python 3.x

When we look at the history of Python, we find a major turning point between Python 2 and Python 3. In 2008, Python 3 was introduced as a fresh start for the language. The creator, Guido van Rossum, decided to make it incompatible with older versions to clean up the language and fix old design mistakes [11]. This meant that programmers couldn't simply run their old Python 2 code on the new version without making changes.

In the beginning, many people continued using Python 2.7 because many important tools and libraries were not yet ready for Python 3. It took a few years for these libraries to be updated. Once they were ready, Python 3 became the standard choice for both students and professional developers [9]. A very important reason for this shift was that Python 3 handles different languages (like Arabic) much better thanks to its improved Unicode support. Since official support for Python 2 ended in 2020, Python 3 is now the only version used for new projects and research [11].

#### 1.1.3 Design Principles

Python’s design philosophy is formally defined by a set of 19 guidelines known as the "Zen of Python" (PEP 20). Created by Tim Peters in 1999, these principles were intended to protect the language’s unique identity as it grew in popularity among developers from different backgrounds. Unlike many other languages that prioritize execution speed at the cost of complexity, Python’s design prioritizes readability and simplicity [11].

The core principles, such as "Beautiful is better than ugly" and "Simple is better than complex," encourage developers to write Pythonic code. These guidelines are not strict rules but a philosophical guide that helps in making design decisions and keeping the code clean. These design choices are built directly into the language and can be accessed by anyone via the `import this` command. By focusing on high-level abstraction, Python allows developers to solve complex problems with less code, making it an ideal choice for modern software engineering [9].

## 1.2 Environment Setup

#### 1.2.1 Python Installation

The installation of Python represents a fundamental step in software development and data analysis workflows. The official distribution is available through the Python Software Foundation website (www.python.org), where the latest stable release (e.g., Python 3.x) can be downloaded according to the target operating system. On Windows systems, enabling the

"Add Python to PATH" option during installation is essential to ensure that the Python interpreter is accessible from the command-line interface. The installation process is followed by executing the python --version command to verify successful configuration. Additionally, testing the installation through the IDLE environment confirms that the interactive shell is properly configured and ready for script execution and educational use [11].


Figure1.1: Python 3.12 Setup Interface with Default Options.


Figure1.2: Python 3.12 IDLE Console Interface.

### 1.2.2 Python Development Environments

An Integrated Development Environment (IDE) is a software suite that consolidates the fundamental tools required for software construction and testing. It typically integrates a source code editor, an interpreter, and a debugger into a single graphical user interface (GUI) [12].

The design of these environments focuses on improving developer productivity by automating repetitive tasks and providing advanced project management tools [11].

Python offers a diverse range of environments tailored to different project requirements, most notably:

**1. IDLE:** The official environment bundled with Python, designed by Guido van Rossum in 1998. It is built on simplicity, providing an interactive shell that allows beginners to test code directly without complex configurations.

**Official Website:** https://python.org/idle [11]

**2.PyCharm**: Launched by JetBrains as a professional IDE focused on intelligent code analysis in 2010. It provides advanced tools for refactoring and managing large-scale projects that require high-precision error checking.

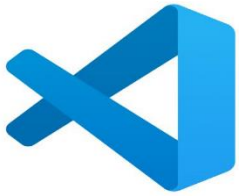
Official Website: https://jetbrains.com/pycharm [12]


[12]

Figure1.3: PyCharm IDE Logo

**3. VS Code:** Released by Microsoft as a lightweight, customizable code editor in 2015. Its technical core relies on Extensions, allowing users to transform it into a full IDE tailored to specific programming needs.

**Official Website**: https://code.visualstudio.com [12]



[11]

Figure1.4: Visual Studio Code (VS Code)

### 1.2.3Python High Low Language

Python is classified as a high-level programming language. This classification is attributed to the level of abstraction it provides by supporting a rich set of data types and operations that exceed those provided directly by the Central Processing Unit (CPU) [13]. The fundamental distinction between Python and low-level languages commonly known as machine languages lies in the fact that the latter are restricted to the primitive instructions supported by the processor, whereas high-level languages aim to simplify the development process [13],[11].

The difference in abstraction is clearly visible when comparing how both languages handle tasks, such as reading a file. In Python, the code is concise and focuses directly on the programming logic:

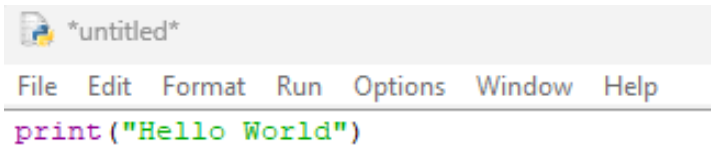



Figure1.5: High-level file handling in Python

In contrast, a lower-level approach (as seen in C++) requires managing precise technical details such as file pointers, buffer sizes, and manual file closing:

```c
#include <stdio.h>

int main() {
    printf("Hello World");
    return 0;
}
```

Figure1.6: Low-level file handling in C++

Since computers can only execute machine language, programs written in Python must undergo an automated translation process using "interpreters" to be converted into an executable format [13].

## 1.3 Object-Oriented Programming (OOP) Paradigms

The Object-Oriented Programming (OOP) paradigm is based on representing software system components as "objects." Each object possesses independent state information represented internally, along with a set of supported behaviours that allow interaction with other components [14]. The structural strength of this model lies in the ability to customize and adapt software by extending existing code rather than modifying it in place [8]. This paradigm is built upon the following core pillars:

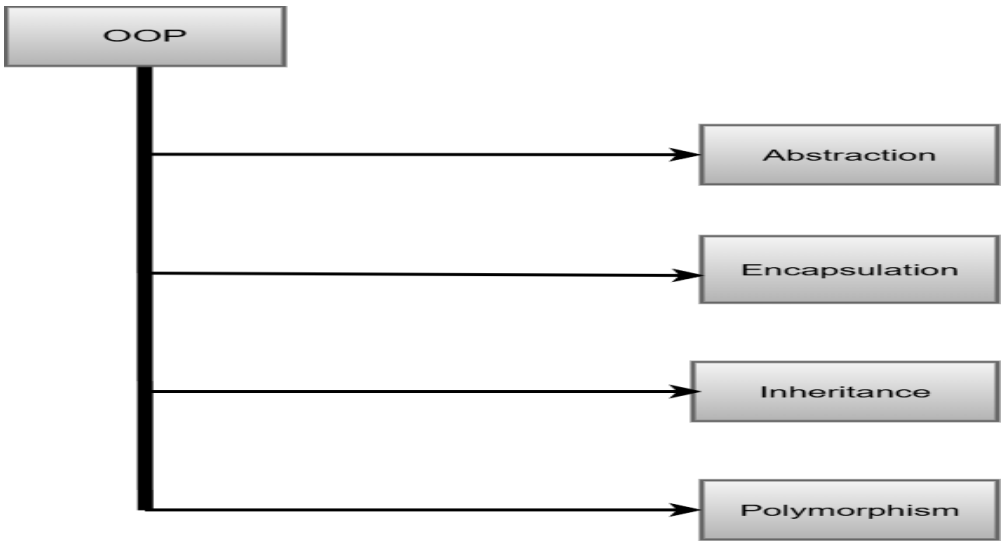



Figure1.7: Pillars of OOP

### 1.3.1 Abstraction

Abstraction focuses on providing simple interfaces that deal with essential ideas while hiding complex internal details. It allows the user to focus on "what" an object does rather than "how" it performs its tasks [8], [11].

Example:

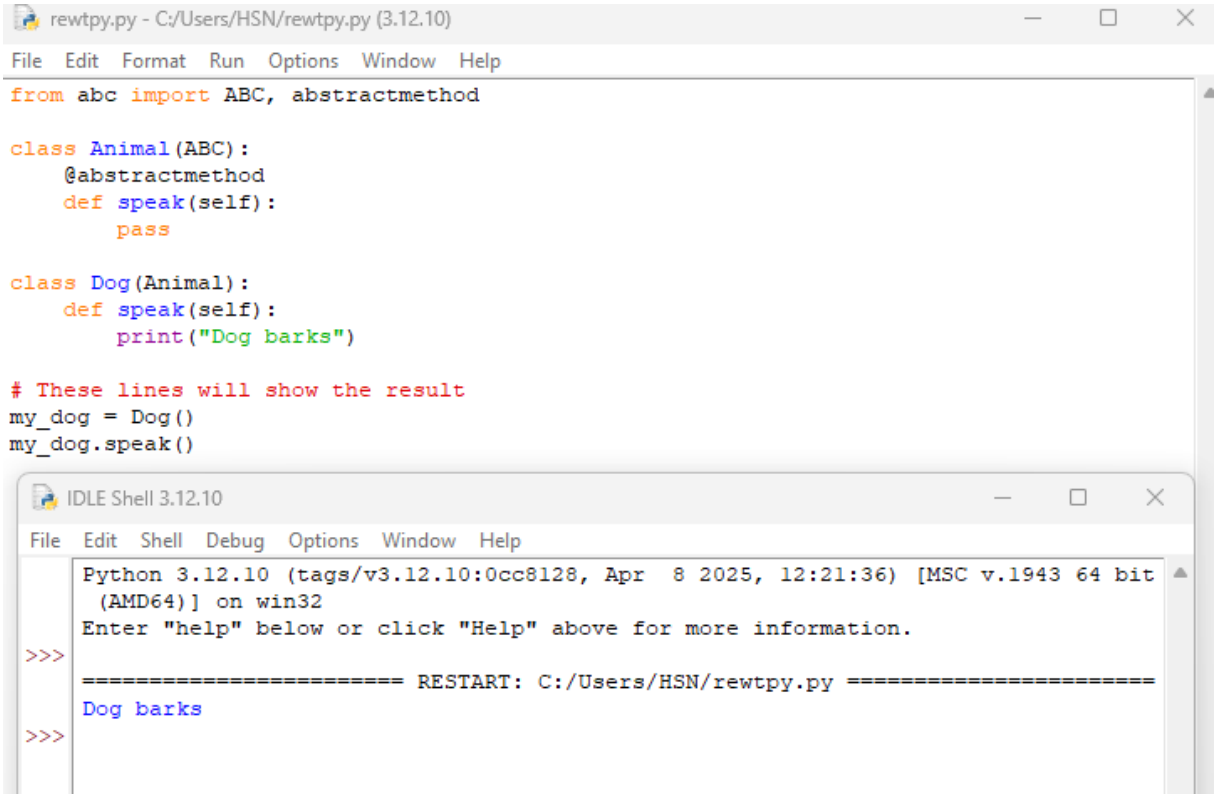



Figure1.8: Abstract Class Example (Animal and Dog)

### 1.3.2 Encapsulation

Encapsulation is defined as the process of combining data (attributes) and the functions (methods) that operate on that data into a single unit (the object). This mechanism allows for "information hiding" by concealing the internal implementation details from the external user [11].

Example:


Figure1.9: Encapsulation Example (Animal and Dog)

### 1.3.3 Inheritance

Inheritance is a mechanism that allows a new class (subclass) to acquire the properties and behaviours of an existing class (superclass). It is used to avoid code duplication and promote reusability by allowing software customization through extension [14], [11]. Example:

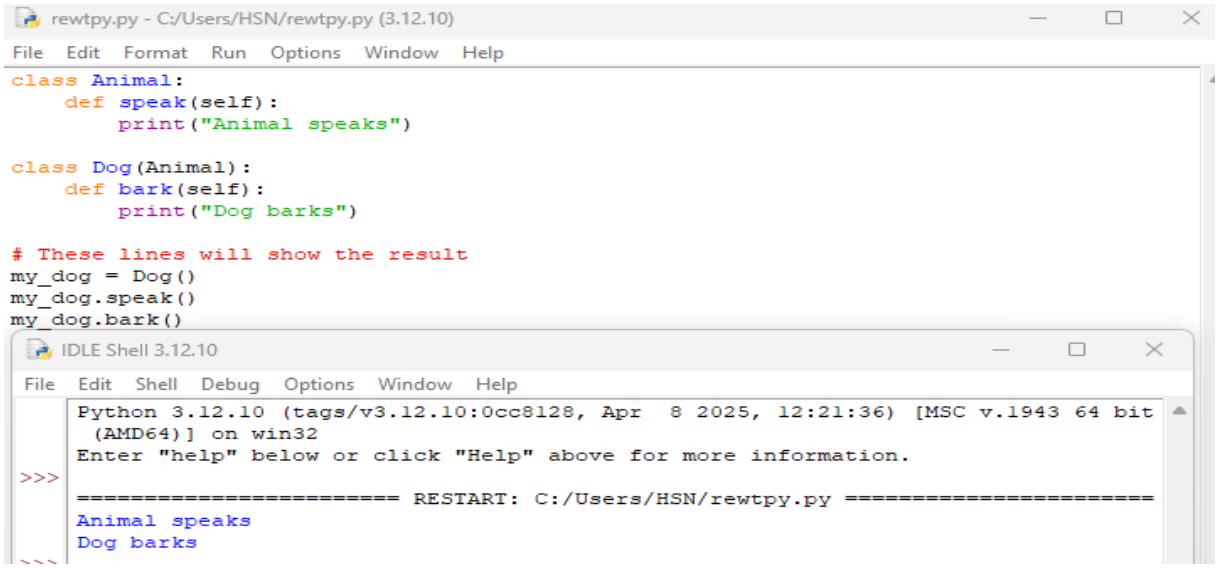



Figure 1.10: Inheritance Example (Animal and Dog Classes)

### 1.3.4 Polymorphism

Polymorphism reflects the ability of different objects to respond to the same method call in different ways depending on their type. This provides the system with high flexibility in handling multiple data types through a unified interface [2].

Example:

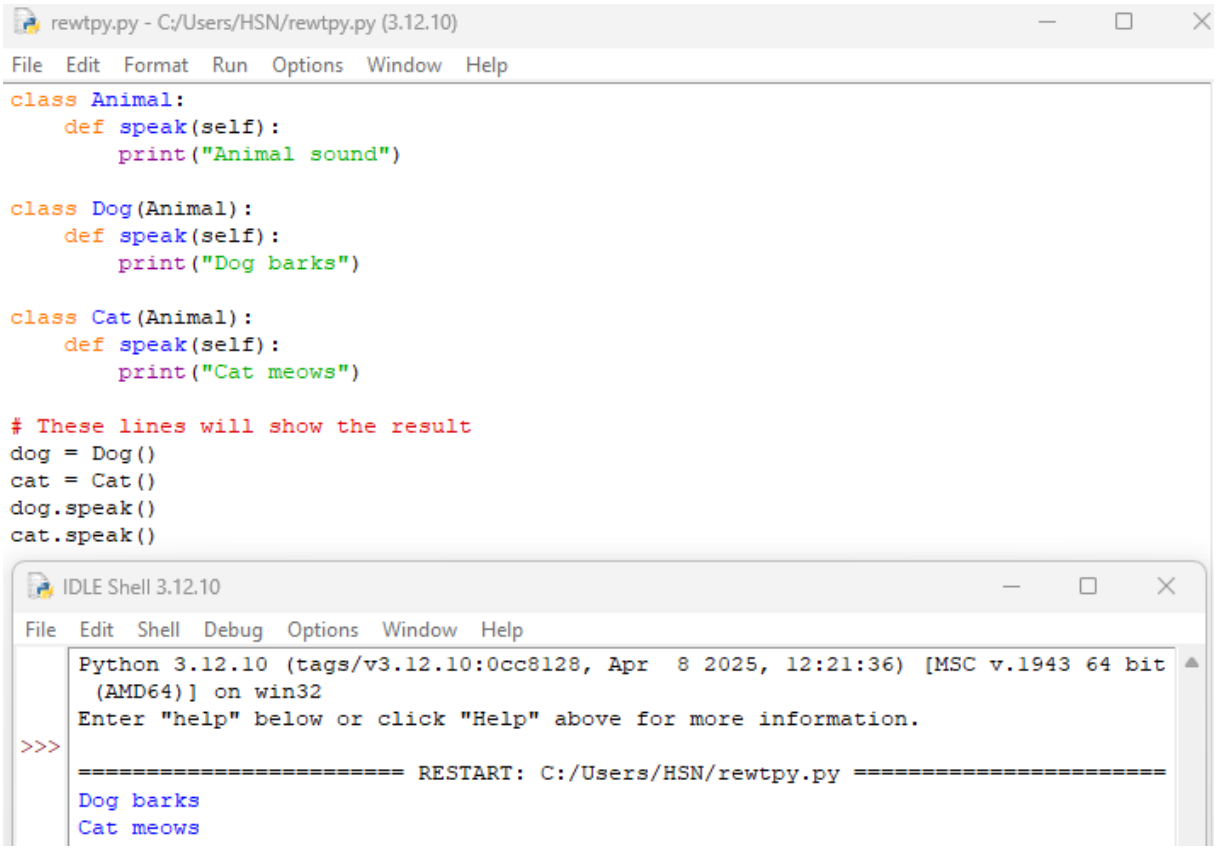



Figure 1.11: Polymorphism Example (Animal, Dog and Cat)

## 1.4 Python Libraries and Ecosystems

Before using Python libraries, it was necessary to manage and install software packages efficiently. For this purpose, the PIP tool was used, which is the official package manager for the Python language [13]. This tool allows developers to download, install, update, and manage thousands of libraries available within the Python ecosystem [11].

The package installation process in Python is organized, as PIP connects to the global package repository, retrieves the required files, and installs them in the correct system path [11]. The role of this tool is not limited to initial installation only, but also extends to version management to ensure project compatibility, removing unnecessary packages, and generating requirement files that guarantee the project can run efficiently on other devices [13][11].

Examples of installing Python libraries using PIP:

```
python -m pip install pygame
python -m pip install numpy
python -m pip install flask
```

### 1.4.1 Modules

Modules in Python are used to organize code into separate files so it becomes easier to reuse and manage. A module is just a file with the extension .py that contains Python code, and it can be used in another program by importing it. The import statement is used to access the content of a module in the current program [8], [11]. Some common modules in Python are

, random, os, time, sys, and json. The math module is used for mathematical operations like square roots. The random module is used to generate random numbers. The os module is used to interact with the operating system. The time module is used to work with time and dates. The sys module gives system functions, and json is used to handle **JSON** data.

**Examples:**

**math:**

- import math
- print(math.sqrt(25))

**random:**

- import random
- print (random.randint(1, 10))

**os:**

- import os
- print(os.getcwd())

**time:**

- import time
- print(time.ctime())

**sys:**

- import sys
- print(sys.version)

**JSON:**

```
import json
data = '{"name": "Ali"}'
print(json.loads(data))
```

A module can be imported in a simple way like this:

```
import module
```

After importing, functions are called like this:

```
module.function()
```

For example:

```
import math
print(math.sqrt(25))
```

When we write import math, Python loads the module so we can use its functions in the program [11].

### 1.4.2 Packages

In Python, a package is a directory that contains a collection of modules organized in a hierarchical structure.

Packages are useful for organizing large projects into smaller, reusable components.

Packages are imported using dotted notation, for example:

```
import package.module
```

It is also possible to import specific elements using the from statement.

Example:

```
import pygame
pygame.init()
screen = pygame.display.set_mode ((400, 300))
pygame.display.set_caption ("Pygame Example")
```

In this example, Pygame is a Python package library used for game development. It contains different modules for graphics, sound, animation, and event handling.

This form of import reflects a directory structure containing Python files, where each folder represents part of the package hierarchy.

The import path must exist in Python's search path (sys.path) so that the interpreter can locate the package.

Before Python 3.3, every package directory needed a file named _init_.py to initialize the package during import.

In modern Python versions, packages can also be created without this file; these are known as Namespace Packages [8].

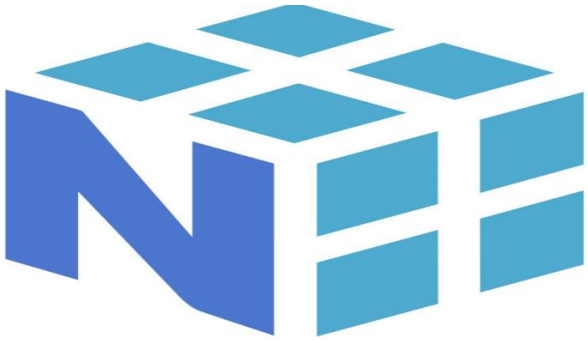
### 1.4.3 Many libraries

Python libraries are considered essential toolboxes containing pre-written code and functions that enable developers and researchers to execute tasks efficiently without starting from scratch. [8]

These libraries aim to simplify critical processes, including data analysis, visualization, unstructured data retrieval from the web, image processing, and building machine learning models. [8], Key Libraries and Functional Roles:

#### 1.4.1.1 Data Science and Analysis:

**NumPy:**



[8]

Figure1.12: NumPy logo

Used for numerical computations and handling multi-dimensional arrays (ndarray). It is more efficient than standard Python lists in terms of speed and memory management. Developed by Travis Oliphant in 2005. Official website: https://numpy.org [14]

**Code Example:**

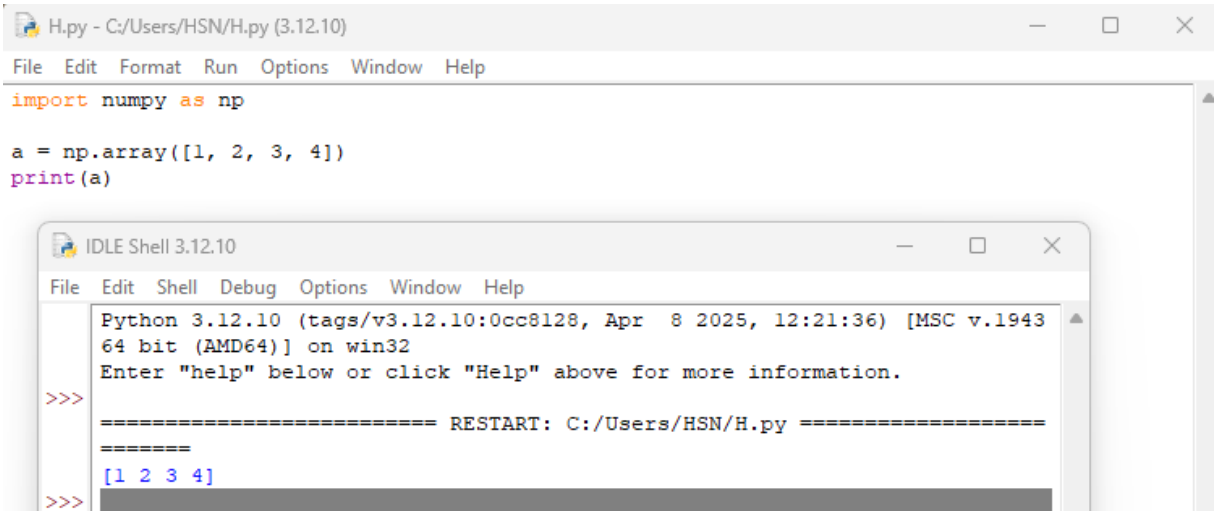



Figure 1.13NumPy library example

**Pandas:**


[15]

Figure1.14: Panda's logo

Simplifies working with structured data through DataFrames. It is utilized in recommendation systems by major companies like Netflix and Amazon to evaluate user behaviour. Developed by Wes McKinney in 2008. Official website: https://pandas.pydata.org [14]

**Code Example:**

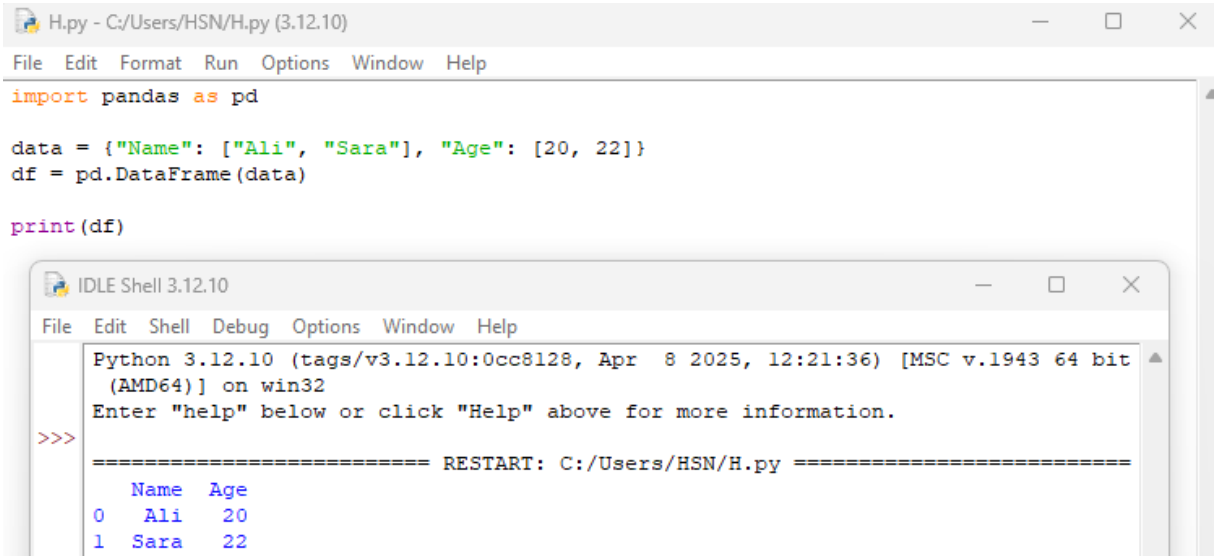



Figure1.15 :Pandas library example

**Matplotlib:**

[16]

Figure1.16: Matplotlib logo

The reference library for creating basic plots and charts. This helps decision-makers understand patterns and trends that may not be apparent in text-based data. Developed by John D. Hunter in 2003. Official website: https://matplotlib.org [15]

**Code Example:**

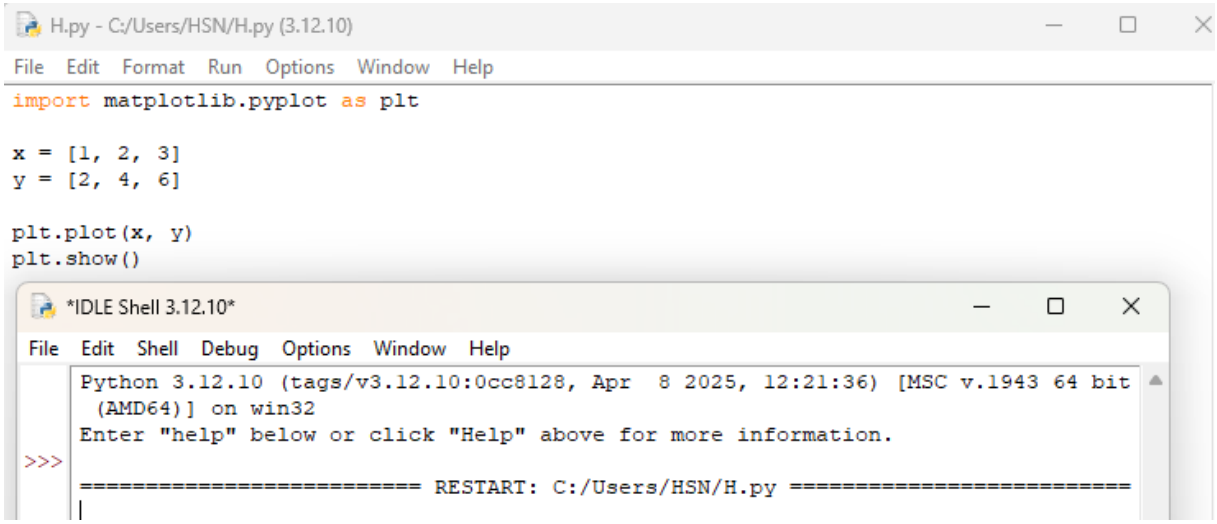



Figure1.17: Matplotlib library example

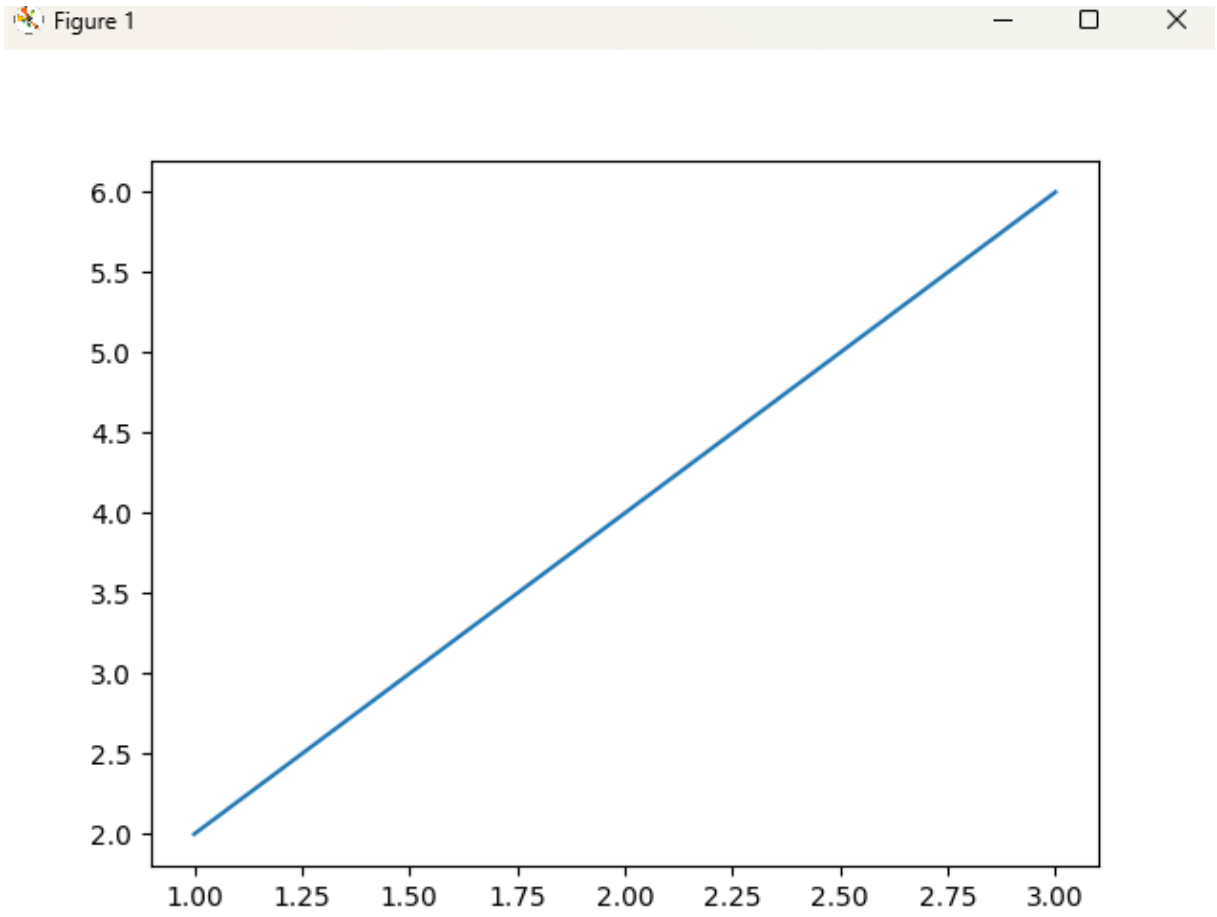



Figure 1.18: Data graph representation

## Conclusion

In conclusion, the comprehensive framework of Python extending from its historical foundations to its advanced modular capabilities establishes it as a vital tool for contemporary software engineering [17], [19]. The strategic evolution of the language, from its genesis through the pivotal transition from Python 2.x to Python 3.x, demonstrates a strong commitment to both innovation and developer productivity [18]. By integrating proper environment configuration with the rigorous application of object-oriented design principles, programmers can develop systems that are both efficient and maintainable [11], [21].

Moreover, the synergy between Python's foundational design principles and its expansive library ecosystem ensures that the language remains adaptable in a rapidly evolving technological landscape [19], [20]. As package management via pip continues to simplify the integration of external modules, Python maintains its position as a leading platform for innovation across multiple domains [22]. Ultimately, mastering these components enables developers to bridge the gap between theoretical concepts and the development of impactful real-world applications [17], [20].

# *Chapter 2: Specialized Domain*

## Introduction

The process of developing computer games is an engineering system that requires a robust link between software components and internal game logic. To achieve this balance, the Pygame library was adopted. Pygame is a set of Python modules designed for writing video games, providing an Application Programming Interface (API) for handling graphics and sound without the need to manage low-level programming complexities [23].

The structural organization of the game system, as described by McGugan, relies on a precise arrangement of components that separates external event processing from the internal engine that manages operations [24]. At the heart of this architecture lies the "Game Loop" concept a continuous loop consisting of three essential stages: handling user events, updating the game state, and drawing the elements onto the screen [23].

This architectural separation between input/output interfaces and game logic provides the necessary flexibility to build complex interactive systems [24]. Through this structure, the game ensures consistent interaction and provides immediate feedback to the learner via the intelligent hint system, the technical implementation of which will be detailed in the following chapters.

## 2.1 Computer Games

### 2.1.1 Definition of Computer Games

Computer games, often referred to as video games, are interactive digital systems in which players engage with rule-based environments to achieve specific objectives, often within a meaningful fictional context . Over the last five decades, videogames have evolved into a major component of popular culture and a multi-billion-dollar industry, encompassing a wide variety of forms, from simple mobile games to expansive online virtual worlds. Defining computer games has been a persistent challenge in game studies, with scholars debating their essential properties and boundaries. While Bergonse emphasizes the interaction between player, machine, and fictional context [25], Bogost highlights the importance of procedural rules in understanding games [26]. Deterding emphasizes player agency and how games are experienced [27], whereas Tavinor discusses their cultural and artistic significance [28]. This diversity of definitions reflects the complexity of the medium and underscores the importance of understanding computer games both as designed systems and as cultural artifacts

### 2.1.2 Components of Computer Games

Computer games are structured around a set of core components that define how players interact with the system and experience gameplay. These components are primarily governed by rules, which distinguish games from free play by constraining player actions, organizing interactions, and establishing clear goals and outcomes [29]. In digital environments, these rules are implemented through game mechanics that shape player experience, regulate levels of challenge, and support cognitive engagement [30].

In addition to mechanics, narrative and aesthetics are essential elements that contribute to the overall player experience. Narrative elements provide meaningful contexts and models of behaviour, while aesthetics enhance realism and immersion. Together, these elements significantly influence player engagement and intrinsic motivation [30].

Another key component is the goal structure, which provides direction and purpose for the player. Clearly defined goals guide player actions and contribute to meaningful gameplay experiences. Alongside goals, feedback systems are essential, as they inform players about their progress and performance, helping to maintain engagement and regulate difficulty [31].

Furthermore, player interaction mechanisms determine how users engage with the game system, linking player input with system responses. Digital games can thus be understood as interactive systems where rules, mechanics, and player actions are interconnected [32].

Finally, the game environment provides the context in which gameplay occurs, integrating visual, narrative, and interactive elements into a coherent experience. Together, these components rules, goals, feedback, interaction, and environment form the core architecture of computer games.

### 2.1.3 Educational Benefits of Computer Games

Digital games are no longer viewed merely as entertainment tools but are increasingly recognized as learning environments built upon sound pedagogical and cognitive principles [33]. According to Gee, high-quality games function as external simulations of complex problem spaces that require players to adopt specific identities and perspectives in order to achieve goals, thereby fostering situated learning and deep conceptual understanding [34]. From a cognitive perspective, recent empirical evidence indicates that even a single session of engaging with complex video games can significantly enhance cognitive functions, including executive control and reaction time [35].

Research utilizing functional Near-Infrared Spectroscopy (fNIRS) has demonstrated that video gaming activates the prefrontal cortex, particularly regions associated with decision-making and information processing, suggesting measurable improvements in cognitive efficiency [35]. Furthermore, digital games distribute cognitive processes between the player and the interactive system, allowing learners to engage with complex structures through human computer interaction, which would otherwise be difficult to understand through traditional instructional methods alone [34].

These findings collectively highlight the potential of digital games as effective tools for cognitive development, learning enhancement, and skill acquisition in both formal and informal educational contexts.

## 2.2 Serious games

### 2.2.1 Definition of Serious Games

Serious Games are structured systems where players engage in challenges governed by rules, resulting in measurable outcomes [36]. Unlike entertainment-focused games, serious games use game elements to achieve educational, training, or behavioural goals [37]. Digitally, they allow learners to control actions, explore outcomes, and practice decision-making in a risk-free environment [38].

The main difference lies in purpose: while commercial games aim for leisure, serious games redesign learning tasks to enhance engagement and effectiveness [2]. They serve key

educational functions: acquiring new knowledge, practicing skills, fostering creativity, and preparing for future learning [38]. Interactive feedback loops allow players to see the results of their actions and reflect on performance [40]. These features make serious games effective for experiential learning, motivation, and skill development [41].

### 2.2.2 Applications of Serious Games in Education

Serious games have been widely applied in education to enhance learning outcomes. Research indicates that these games can improve students' knowledge acquisition and understanding of educational content. Additionally, serious games are effective in increasing motivation and engagement, as game elements such as challenges, immediate feedback, and rewards make learning more interactive and appealing. Beyond knowledge and motivation, serious games also contribute to the development of cognitive, behavioral, and affective skills, providing learners with opportunities to practice problem-solving, critical thinking, and decision-making in an interactive environment. These findings demonstrate that serious games offer a valuable approach to education by combining learning with engaging gameplay[42].

## 2.3 Word Games

### 2.3.1 Definition of Word Games

Word games are defined as rule-based interactive systems where language and vocabulary serve as the primary mechanics of the experience; their design aims to create a balance between cognitive challenge and entertainment from a perspective that prioritizes the user experience [45]. In an educational context, these games are described as "digital learning environments" that enhance the player's ability regarding Word-to-Text Integration (WTI), thereby improving reading comprehension through gameplay mechanics [43]. Furthermore, the definition extends to include design aspects that ensure "accessibility," where game interfaces are adapted to ensure that all users can effectively interact with linguistic challenges [44].

### 2.3.2 Cognitive and Linguistic Benefits of Word Games

Word games provide significant cognitive and linguistic benefits by reinforcing the mental structures underlying language acquisition. From a cognitive perspective, these games enhance memory processes by strengthening semantic and orthographic connections within the mental lexicon, which in turn improves processing efficiency, reduces decision-making time, and increases retrieval fluency [46].

In addition, activities such as crossword puzzles and other word-based challenges contribute to lowering the cognitive load associated with learning new or technical vocabulary. This reduction allows learners to allocate more cognitive resources to higher-order thinking skills, including reasoning, analysis, and problem-solving [47].

From a linguistic perspective, word games function as active learning tools that promote meaningful engagement with vocabulary, leading to improved acquisition and long-term retention. This sustained exposure to lexical items plays a crucial role in the development of the four fundamental language skills: listening, speaking, reading, and writing.

Moreover, digital game-based learning (DGBL) platforms, such as Kahoot and Wordwall, further enhance these outcomes by increasing learner motivation, encouraging repeated practice, and creating an interactive learning environment that supports improved academic performance and communication skills [48]

## 2.4 The Pygame Library

### 2.4.1 Introduction to Pygame

Pygame is a collection of Python modules designed for the development of video games and multimedia applications. It is considered an extension library that enhances the capabilities of the Python programming language by providing built-in functionalities for graphics rendering, audio processing, and user input management. Moreover, Pygame is built on top of the Simple DirectMedia Layer (SDL), which abstracts low-level hardware operations and allows developers to focus mainly on game logic and design rather than system-level multimedia handling. According to Sweigart, Pygame enables programmers to develop interactive games efficiently using Python [49]. In addition, the official Pygame documentation describes it as a high-level wrapper over SDL that provides a simplified and user-friendly interface for game development tasks [50].

### 2.4.2 Installing Pygame

Pygame requires Python to function, and it can be downloaded from the official website (python.org) if it is not already installed. It is recommended to use the latest version of Python, as it generally provides better performance and improved features compared to older versions. It is important to note that Pygame no longer supports Python 2.

The recommended method for installing Pygame is through the pip package manager, which is included by default in recent versions of Python. The following command is used to install Pygame in the user directory:

– Python3 -m pip install -U pygame --user

To verify that the installation is successful, a built-in example can be executed using the following command:

– Python3 -m pygame.examples.aliens

If the example runs correctly, this confirms that Pygame has been successfully installed and is ready for use [51][50].

### 2.4.3 Basic Window Creation in Pygame

Basic window creation in Pygame is a fundamental step in developing any graphical application or game. The display module is responsible for managing all visual output and must be initialized before use. This is done using pygame.display.init(), which activates the display system. In most cases, this step is automatically handled when calling pygame.init() [52].

After initialization, the main game window is created using pygame.display.set_mode(), which defines the size of the display surface where all graphics will be rendered. To provide a meaningful identity to the application, the window title can be set using pygame.display.set_caption (), which assigns a name that appears on the window frame [52].

During program execution, the display must be continuously refreshed to reflect changes on the screen. This is achieved using pygame. display. update (), which can update either the full screen or specific regions for better performance [52].

In addition, Pygame provides pygame.display.set_caption () to retrieve the current window title, ensuring that the application can verify or use the existing caption when needed.

Finally, the display module can be checked using pygame.display.get_init () and properly closed using pygame. display. quit () when the program terminates [52]. These functions together form the complete lifecycle of window management in Pygame.

### 2.4.4 Structure of the Game Loop in Pygame

The game loop is the core structure of any Pygame application. Once the game is initialized, it is placed inside a continuous loop that runs until the player chooses to exit. This loop ensures that the game continuously updates, processes input, and maintains real-time interaction with the user.

At the beginning of each iteration (frame), the program checks for Pygame events such as keyboard input, mouse clicks, joystick movement, window resizing, or a request to close

the window. In this example, the primary event handled is the user attempting to quit the game by closing the window. When this event occurs, the loop terminates, which ends the program execution. [53]

After handling events, the program redraws the screen by re-blitting the background and updating all visual elements. Although this step may seem unnecessary in static programs where nothing changes on screen, it is essential in dynamic games where objects move or change state, ensuring that the display always reflects the current game state. [54]

Finally, the display is updated to render all changes on the screen, completing one frame of the game loop before the next iteration begins using pygame. display. update (). [53]

### 2.4.5 Event Handling in Pygame

Pygame manages all event communication through an event queue. This queue stores all input and system events generated during program execution. The event system depends on the initialization of the display module; if the display is not properly initialized, event processing may not function correctly. Since the event queue has a limited capacity, events may be lost if it becomes full. Therefore, developers must continuously process events in every frame using functions such as pygame.event.get() or pygame.event.pump() to ensure that user inputs are not ignored.

Failure to handle events regularly may cause the operating system to consider the application unresponsive.

Pygame also provides additional tools such as pygame.event.set_blocked() to control which events are allowed in the queue and improve performance and efficiency (1).

The following code illustrates a typical event handling loop in Pygame:

```
for event in pygame.event.get ():  # event handling loop
    if event.type == QUIT or (event.type == KEYUP and
event.key == K_ESCAPE):
        pygame.quit()
        sys.exit()
    elif event.type == MOUSEMOTION :
        mousex, mousey = event.pos
  elif event.type == MOUSEBUTTONUP :
        mousex, mousey = event.pos
        mouseClicked = True
```

This loop processes all events generated since the last frame of the game loop. It iterates over the list of pygame.Event objects returned by pygame.event.get(), ensuring real-time interaction between the user and the game. The event handling loop is executed inside the main game loop and is responsible for detecting keyboard input, mouse movement, and window actions such as quitting the game (2).

## 2.5 Game Development with Pygame

### 2.5.1 Game Development Process Using Pygame

The game development process using Pygame follows a loop-based architecture that combines initialization, event handling, game state updates, and rendering in a continuous cycle. This structure ensures real-time interaction and smooth execution in 2D game applications Pygame.

The process begins with library initialization and creation of the game window, followed by the implementation of the main game loop. This loop continuously processes user inputs, updates game logic, and redraws the screen until a termination condition is reached.

In many Pygame-based educational and prototype games, entities are commonly represented using dictionaries rather than object-oriented classes. For instance, in Squirrel Eat Squirrel, player and enemy objects are stored with attributes such as position coordinates (x, y) and rectangular data for rendering and collision detection. This simplifies game state management while maintaining flexibility [55].

A key concept is the distinction between game world coordinates and pixel coordinates, where the first defines logical positioning while the second controls screen rendering. This separation supports camera movement and larger game worlds.

Additionally, Pygame follows an event-driven approach where user inputs are captured and processed in real time. Some educational game designs also integrate level progression and scoring systems to enhance engagement and motivation [56].

### 2.5.2 Classification of Games Developed via Pygame

Pygame supports the development of various categories of 2D games, particularly in educational and prototype-based environments. These games can be classified according to their mechanics, including puzzle-based games and arcade-style games, which are commonly used to introduce fundamental concepts in game design and programming Pygame.

One major category is memory-based puzzle games, where the player is required to match identical elements hidden behind covered tiles. In Memory Puzzle, all icons are initially hidden, and the player uncovers two at a time to find matching pairs. If the selected icons match, they remain visible; otherwise, they are hidden again. This type of game improves memory and concentration skills [55].

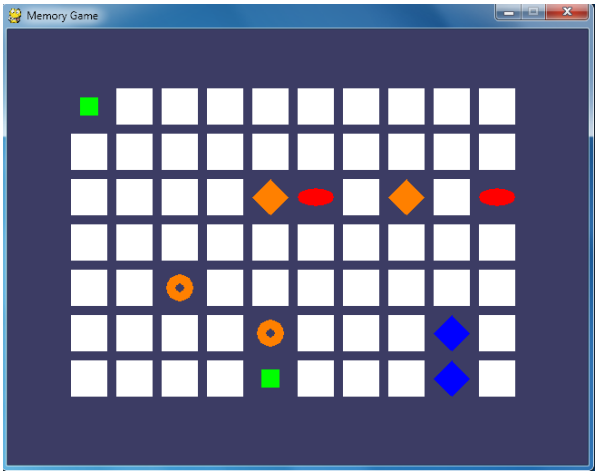
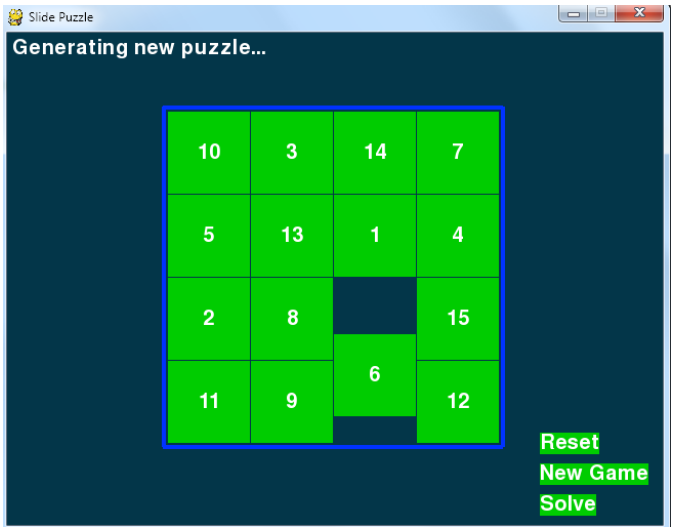



Figure2.1: Implementation of a Memory Puzzle Game using Pygame [55]

Another category is sliding puzzle games, which involve rearranging tiles into a specific order within a grid structure. In Slide Puzzle, the board consists of a 4×4 grid with numbered tiles and one empty space. The player must slide tiles strategically to restore the original sequence.




This genre emphasizes logical thinking and spatial reasoning [55].

Figure2.2: Implementation of a Slide Puzzle Game using Pygame [55]

A third category is arcade-style games, particularly snake-like mechanics. In Wormy, the player controls a continuously moving worm that cannot stop or slow down, but can only change direction. The objective is to eat randomly appearing apples, which increases the worm's length after each consumption. The game ends if the worm collides with itself or the boundaries of the screen. This type of game focuses on reflexes and real-time decision-making [55].

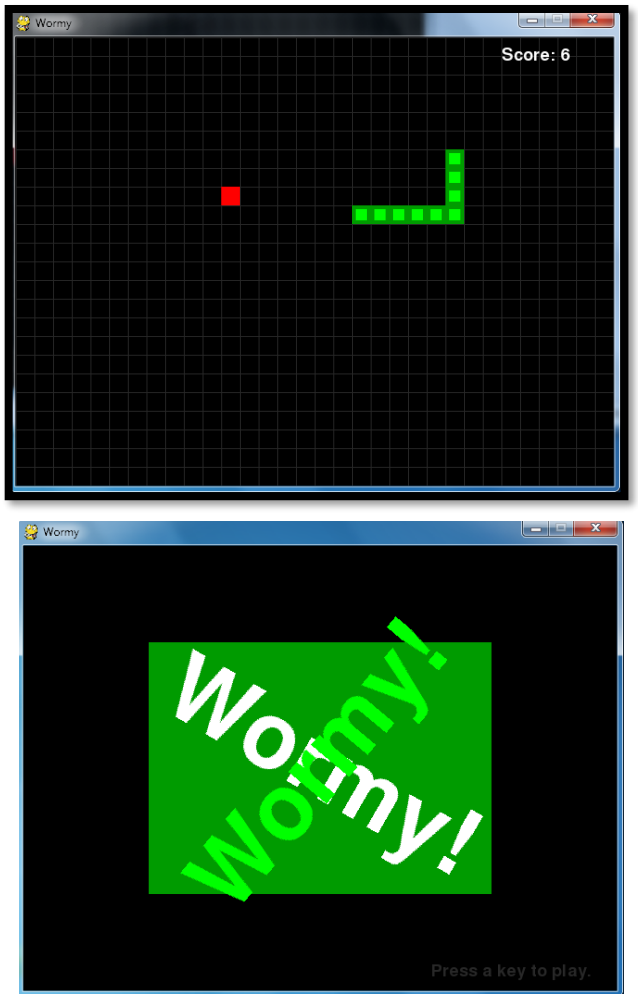



Figure2.3: Implementation of a Wormy Game using Pygame[55]

Overall, these examples demonstrate how Pygame can be used to implement different types of interactive games ranging from cognitive puzzles to reflex-based arcade games.

### 2.5.3 Advantages and Constraints of Pygame in Development

The integration of Python with the Pygame library reflects a modern approach in game development, where high-level scripting is used for game logic while relying on optimized C-based libraries for performance and system-level operations.

Pygame provides an effective environment for learning and developing interactive applications, as it reinforces fundamental programming concepts such as variables, loops, and functions through immediate visual feedback, which enhances experimentation and understanding of code behaviour [57].

In addition, Pygame is built on top of the Simple DirectMedia Layer (SDL), which enables access to low-level multimedia functions and supports hardware acceleration, leading to improved rendering performance depending on system capabilities and optimization techniques [80]. Another important advantage is its cross-platform portability, as applications developed using Pygame can run on Windows, Linux, and macOS with minimal or no modifications, which increases development flexibility and reduces deployment effort.

Furthermore, Pygame offers a flexible and low-level structure that does not enforce a rigid architecture, allowing developers full control over game design and logic, although this requires stronger programming skills and careful system organization .

However, despite these advantages**,** Pygame presents several constraints. Performance limitations may appear as game complexity increases, particularly due to the computational ceiling of Python in real-time processing scenarios.

Moreover, Pygame is a library rather than a complete game engine, and therefore lacks built-in advanced features such as physics engines, AI systems, and animation frameworks, requiring developers to implement or integrate additional components manually [58]. In addition, rendering performance may vary depending on hardware support, and inefficient use of graphics operations, such as full-screen scrolling, can significantly affect execution speed, especially when hardware acceleration is not properly utilized.

## 2.6 Serious Game with Pygame

### 2.6.1 Framework for Developing Serious Games in Pygame

The development of serious games is guided by structured frameworks that combine educational theories with game design principles to ensure both learning effectiveness and player engagement. These frameworks focus on aligning instructional goals with gameplay elements, allowing educational content to be meaningfully embedded within interactive experiences.

The first component of such frameworks is the definition of clear learning objectives. These objectives ensure that the game is designed to achieve specific educational outcomes and support different cognitive levels as defined in Bloom's revised taxonomy, including remembering, understanding, applying, and analysing knowledge [60]. In the context of serious game development, learning objectives serve as the foundation for designing tasks, challenges, and feedback systems that promote progressive learning.

The second component is the MDA (Mechanics, Dynamics, Aesthetics) framework, which provides a formal structure for analysing and designing games. Mechanics refer to the rules and interactive elements implemented in the game, dynamics describe the behaviour that emerges from player interaction, and aesthetics represent the emotional and experiential response of the player. This model helps designers balance educational content with engaging gameplay experiences [59].

By combining learning objectives with the MDA framework, designers can create serious games that effectively integrate pedagogy and gameplay. These frameworks are conceptual models rather than platform-dependent solutions and can therefore be implemented using game development tools such as Pygame, which provides functionalities for rendering graphics, handling user input, and managing game loops, making it suitable for building interactive educational games.

### 2.6.2 Integrating Educational Design Principles

The successful development of Serious Games relies on a systematic integration of didactic and psychological principles to ensure that the gaming environment functions as an effective learning tool . Didactic design principles focus on learning-promoting elements such as the clear definition of learning objectives, the integration of real-world problem-solving situations, and the inclusion of feedback mechanisms [61], [62]. These elements ensure that the game is not merely entertaining but achieves a sustainable learning effect by allowing players to acquire desired knowledge and skills through direct interaction with the system [61]. Psychologically, the design must influence player behaviour and motivation by creating emotionally appealing game worlds and applying motivational reinforcement mechanisms [61].

A core pedagogical requirement is the principle of "learning through play," where action-oriented learning precedes the introduction of formal theories, thereby creating a relaxed and motivated atmosphere [61]. This is supported by the "Game Loop" or repetitive patterns that allow learners to practice, reinforce, and internalize knowledge until it becomes

routinized as mental scripts [61]. Furthermore, the difficulty level must be dynamically adjusted to match the player's progress a concept rooted in "Mastery-Learning" and "Scaffolding" which helps maintain a state of flow and optimal balance between challenges and individual abilities [61], [62]. Ultimately, the integration of these principles facilitates a didactic-immersive design where learning is perceived as a natural process rather than a strenuous task [61].

### 2.6.3 Review of Existing Serious Games Developed with Pygame

The Pygame library has been utilized in the development of educational and prototype-level serious games due to its flexibility in handling 2D graphics and its efficiency in implementing interactive learning environments. This section highlights representative examples that illustrate the integration of pedagogical objectives with game mechanics.

The first example is PY-RATE ADVENTURES, a 2D platform-based serious game designed to introduce students to fundamental programming concepts using Python. Developed within the Pygame framework, the game integrates educational content such as variables, loops, and conditional statements into interactive gameplay challenges. According to [63], the game was designed following the Educational Game Design Framework (EGDF) and evaluated using the MEEGA+ model, demonstrating its effectiveness in improving short-term learning outcomes and enhancing engagement among novice programmers

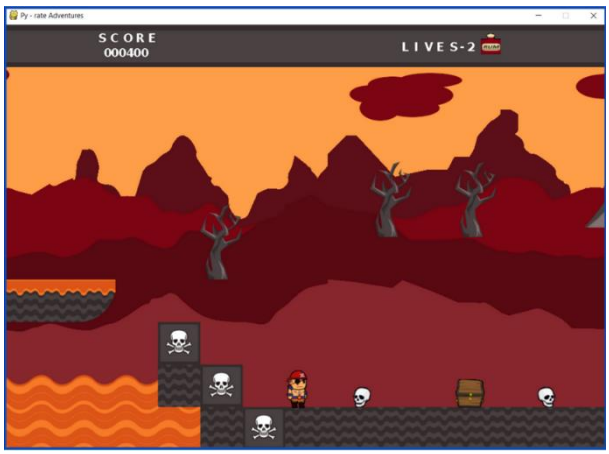



Figure2.4: Player navigates through the map [63]


Figure2.5: Quizzes for testing the player's knowledge during the game[63] .

The second example involves the use of Sudoku solver implementations as interactive educational tools for teaching algorithmic problem-solving. While Sudoku itself is a logic-based puzzle, its implementation within Pygame environments enables the visualization of complex computational strategies. As discussed in [64], various algorithms such as backtracking, constraint propagation, and metaheuristic approaches like Ant Colony Optimization (ACO) can be integrated and compared within such systems. When combined with user interaction and real-time feedback, these implementations evolve into serious game-like environments that facilitate experiential learning and deeper understanding of algorithm efficiency.

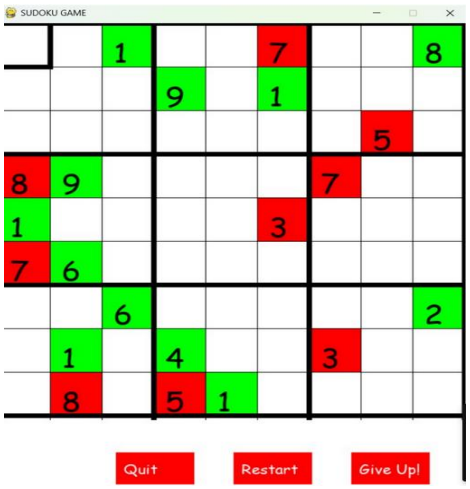



Figure2.6: A 9x9 Sudoku puzzle in its initial state, illustrating the grid with some pre-filled numbers and empty spaces that the player needs to fill [64].

Overall, these examples demonstrate that Pygame-based serious games are particularly effective in computer science education, where abstract concepts can be transformed into interactive and engaging learning experiences.

## 2.7 Word Games with Pygame

### 2.7.1 Design Considerations for Pygame-Based Word Games

The development of educational word games using the Pygame library requires a strategic integration of interface design and software architecture to ensure pedagogical effectiveness. Drawing from the "Lens of the Interface" framework [65], the design must prioritize the seamless translation of physical actions into virtual outcomes. In Pygame, this begins with the Physical Input → World loop, where keyboard events captured via pygame.event.get () must be mapped to precise character rendering on the game grid [66]. Efficient event handling ensures that the player's cognitive load is focused on linguistic problem-solving rather than technical navigation.

Furthermore, the Virtual Interface → World transition is critical in serious games. When a player interacts with a "Smart Hint System," the underlying game state must update immediately to provide scaffolded feedback. This is implemented in Pygame by manipulating Surface objects and Rect objects to highlight specific letter slots or display definitions [66]. According to Schell, the World → Virtual Interface loop must also be addressed; changes in the game state, such as a decreasing timer or a depletion of "health," must be visually manifested through dynamic HUD (Heads-Up Display) elements [65]. These elements are typically rendered using the pygame.font module to ensure high legibility and contrast, which are essential for maintaining user engagement [66].

Finally, the Virtual Interface → Physical Output consideration dictates the aesthetic and sensory feedback provided to the learner [65]. Effective word game design utilizes color-coded feedback and auditory cues to signal success or error. By leveraging Pygame's ability to handle frame-rate independent updates and pixel-perfect collision detection, developers can create a responsive environment where the interface acts as a transparent membrane between the student and the educational content [67].

### 2.7.2 Game Logic, String Manipulation, and Mechanics

The integration of game logic, string manipulation, and core mechanics forms the fundamental architecture of interactive software developed with Python and Pygame. Game logic serves as the primary controller, utilizing continuous game loops and Boolean states to

process user input and update the game's internal status in real-time . For educational or text-based games, string manipulation is a critical mechanic; it involves using methods such as `lower ()` for case-insensitive input comparison and join () to display updated game states, like revealed letters in a word puzzle [68]. Furthermore, the mechanics extend to spatial and temporal management, where Cartesian coordinate systems (X, Y) define object positions, and collision detection algorithms determine interactions between sprites . These elements are synchronized through timing constants and frame rate controllers (FPS) to ensure consistent gameplay and maintain the software's pedagogical or entertainment objectives [69].

### 2.7.3 User Interface (UI) and Visual Design

The User Interface (UI) in Pygame-based word games functions as a vital bridge that translates internal mechanics into an interactive player experience. From a technical perspective, the foundation of this interface is the Surface object, specifically the display surface initialized via pygame.display.set_mode (), which serves as the primary canvas for all graphical elements [70]. Since word games rely heavily on legibility, the rendering of text is handled through specialized font objects where developers transform strings into pixel-based surfaces, often applying anti-aliasing to smooth character edges. Beyond mere technical rendering, the visual design must incorporate high contrast between text and backgrounds to mitigate eye strain during intense reading sessions, a core principle of player-centric design [71]. Feedback is further reinforced through the RGB colour system, where specific colour shifts such as green for correct words or red for errors provide immediate cues that guide the player's progress. Spatial organization is efficiently managed using pygame.Rect objects, which allow for the precise mathematical cantering of characters within tiles and the creation of structured grids. Ultimately, the integration of these tools with design aesthetics ensures that the interface is not only functionally robust but also emotionally engaging, aligning with the "Aesthetics" layer of the formal MDA framework [72].

### 2.7.4 Examples of Word Games Developed with Pygame

The versatility of Pygame allows for the development of a wide range of word-based games, spanning from simple text-driven logic to more advanced graphical interfaces. A primary example is Hangman, which serves as a foundational project for introducing concepts such as string manipulation, conditional logic, and basic game state management [73]. The core gameplay of Hangman revolves around guessing a hidden word letter by letter within a limited number of attempts. Each incorrect guess results in drawing a part of a gallows, and the player loses if the drawing is completed before the word is revealed.

Initially implemented as a command-line interface (CLI) application, Hangman can be effectively adapted into a Pygame-based version by incorporating graphical components such as the gallows, dynamic text rendering, and interactive letter selection. Its implementation relies heavily on string operations such as .lower() and .upper() for input normalization, as well as .split() for managing word datasets [74].

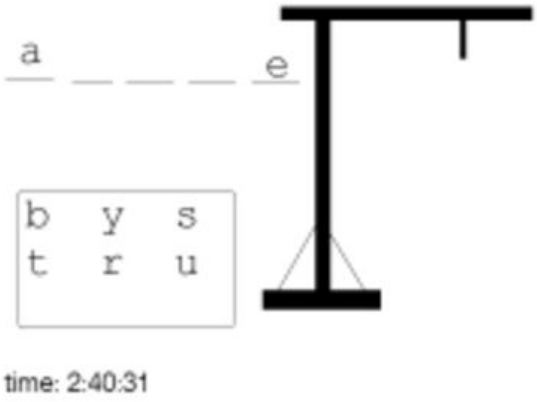



Figure2.7: Implementation of a Hangman Game using Pygame[50]

Beyond introductory projects, more advanced word games have been inspired by modern applications such as Wordle. In this game, players are tasked with guessing a secret five-letter word within six attempts. After each guess, the game provides real-time feedback by changing the color of the tiles: green indicates the correct letter in the correct position, yellow means the letter is in the word but in the wrong spot, and gray shows that the letter is not in the word at all. Although originally developed as a web-based game, Wordle-like mechanics are frequently reimplemented in Pygame environments to demonstrate grid-based user interface design and real-time feedback systems.

These systems typically rely on position-based string comparison and constraint satisfaction logic to provide color-coded responses to user input [75]. Such implementations highlight the use of two-dimensional lists and coordinate systems in Pygame for managing game boards and tile-based interactions, thereby reinforcing key programming concepts related to data structures and spatial representation [76].


Figure2.8: Implementation of a Wordle Game using Pygame [50]

Collectively, these examples demonstrate not only the flexibility of Pygame but also its effectiveness in supporting both procedural and data-driven game logic. Word-based games, in particular, provide a practical framework for applying algorithmic thinking, as they involve string processing, pattern recognition, and state management. Moreover, they hold significant value in educational contexts, contributing to vocabulary development, cognitive engagement, and problem-solving skills through interactive and visually structured gameplay.

## 2.8 Intelligent Hint Systems in Educational Games

### 2.8.1 Concept of Hint Systems

In puzzle-based games, embedded hint systems are essential mechanisms designed to regulate the challenge level and enhance the overall player experience. The primary objective of a hint system is to manage the delicate balance between game difficulty and player satisfaction, ensuring that the “Game Flow” remains uninterrupted by excessive frustration [78]. Effectively designed hints help players navigate obstacles while preserving their sense of achievement; a hint that is too direct may diminish the reward of solving the puzzle, whereas a lack of guidance can lead to disengagement [78].

Traditionally, several hint mechanisms have been implemented in the gaming industry, each with specific conceptual frameworks and limitations:

Hint on Request: Players manually trigger assistance, which can sometimes break immersion or lead to over-reliance [79].

Hint on Delay: Guidance is provided after a fixed period of inactivity, but it cannot distinguish between a player who is "stuck" and one who is intentionally thinking deeply.

Hint on Failure: Assistance appears after a set number of incorrect attempts, which may inadvertently highlight the player's struggle and increase negative emotions.

Hint on Loading: Often used during transitions, these are usually context-independent and may not address the specific challenge the player is currently facing.

While these traditional hint mechanisms are widely used, they are generally static or rule-based, lacking awareness of the player's real-time cognitive or emotional states.

### 2.8.2 Adaptive and Intelligent Hints

Adaptive Hint Systems represent a paradigm shift toward personalized and context-aware assistance. The core of this concept lies in the system's ability to dynamically adjust its intervention based on the player's real-time cognitive and emotional state [78]. This intelligence is typically built upon two technological pillars:

Multimodal Sensing and Detection: Intelligent systems utilize multimodal data such as facial expressions, head movements, and gaze patterns to detect a "stuck state" [79]. By employing deep learning architectures like Long Short-Term Memory (LSTM) networks, the system can differentiate between productive concentration and genuine frustration, ensuring the hint is delivered at the most opportune moment [79].

Contextual and Semantic Content Delivery: Intelligent hints focus on what to show as much as when to show it. Semantic and concise clues guide the player's attention toward a solution without revealing it entirely, maintaining immersion and encouraging "Aha!" moments

By integrating these adaptive layers, hint systems evolve from simple help menus into proactive assistants that optimize the learning curve and emotional journey of the player [79].

### 2.8.3 Role of Hints in Improving Learning

Hints are not merely tools for overcoming technical obstacles; they function as a fundamental scaffolding mechanism that enhances educational outcomes and cognitive development. According to Choi et al. (2023), the effectiveness of hints in a learning environment is significantly amplified when combined with reflection prompts, serving several key roles [80]:

Encouraging Reflective Thinking: Rather than providing a passive solution, modern hint systems utilize reflection prompts to encourage learners to think deeply about “why” a certain solution works. This process transforms a hint from a direct answer into a pedagogical tool that fosters conceptual understanding [80].

Enhancing Academic Performance: Empirical evidence suggests that learners who engage with hints structured around reflective feedback show a measurable improvement in their subsequent performance. These hints bridge the knowledge gap and support the learner in navigating complex problem-solving scenarios [80].

Increasing Perceived Learning and Self-Efficacy: The strategic use of hints contributes to a higher level of perceived learning. When learners receive incremental, reflective support, it builds their confidence and motivation, reinforcing their sense of agency within the digital learning environment [80].

Supporting Self-Regulated Learning: Hints serve as a form of “intelligent scaffolding” that helps learners monitor their own progress. By providing just-in-time feedback, these systems allow learners to self-correct and develop independent problem-solving strategies over time [80].

### 2.8.4 Implementation of Hint Systems in Game Development Environments

The technical implementation of hint systems within game development frameworks necessitates a robust architecture that balances logical triggers with user experience. In lightweight environments such as Pygame, this implementation is primarily driven by the game loop and manual state management. Developers must explicitly track player behaviour through variables such as elapsed time or failed attempts to determine when hints should be triggered. According to Sweigart, managing game states involves updating variables continuously within the loop to control game events [81], [82]. In this context, a hint system can be implemented as a conditional module where predefined states activate response renders using the `pygame.font` module [81].

Beyond basic frameworks, advanced engines like Unity adopt a component-based architecture, allowing hints to be triggered through spatial colliders or scripted behaviours. However, the design of these triggers is critical; poorly timed hints can disrupt the player's "flow state" and negatively impact learning performance [83]. To mitigate this, modern implementations increasingly rely on data-driven approaches that support multiple delivery modes: on-demand, automatic, and adaptive [84]. These systems often incorporate

"scaffolding" strategies, providing vague nudges initially and progressing to direct solutions only as a last resort to preserve the educational challenge [85].

Furthermore, the integration of Artificial Intelligence has evolved these mechanisms into Intelligent Tutoring Systems (ITS). These platforms provide dynamic adaptation and real-time feedback, moving beyond static triggers to analyse the learner's specific needs [86]. Research indicates that such intelligent systems, when implementing "sub step-based" tutoring, can achieve effectiveness levels comparable to human tutoring by guiding the student through the process rather than just providing the final answer [85]. Thus, the implementation of a hint system in an educational game should not be viewed merely as a help feature, but as a sophisticated pedagogical tool that mimics human-like intervention through algorithmic logic.

### 2.8.5 Hint Systems in Games

Hints are a fundamental educational tool that supports the learner's cognitive progress without directly providing the answer. In Sudoku, hints appear as initial numerical entries distributed within the grid; these entries are mathematically designed to guide the player toward a unique solution while controlling the puzzle's difficulty level.[86]

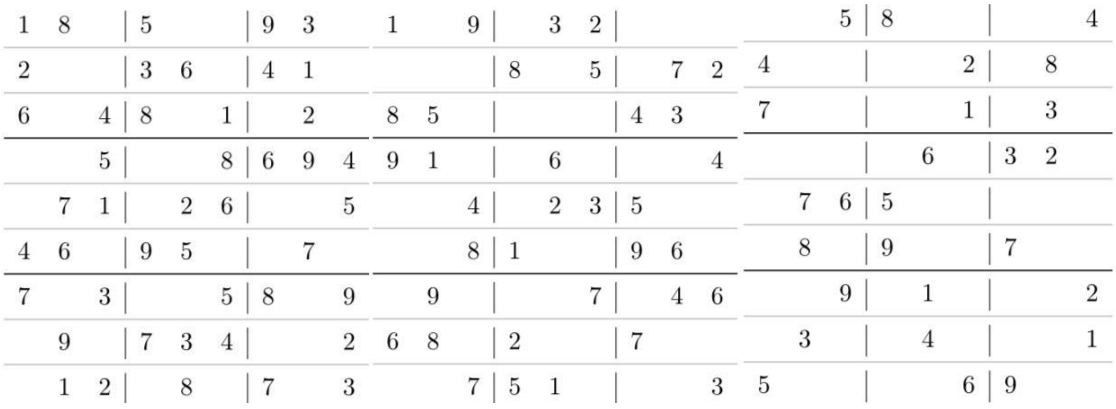


Figure2.9: Levels of the Hint System in Sudoku Puzzles [86]

Similarly, in Reversegam (also known as Othello), hints are utilized through a "Hints Mode" that provides a visual representation of legal moves on the board using small dots, thereby teaching the player strategic rules through active practice [87].

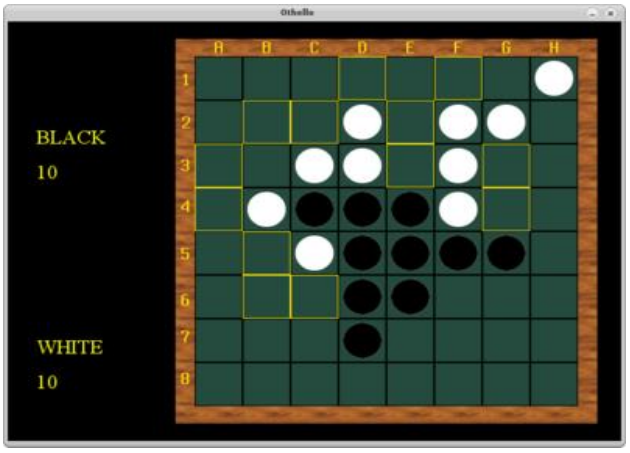



Figure2.10: Legal Move Hints in Othello [88]

Furthermore, hints serve as the primary gameplay mechanic in other titles, such as Bagels, where the system provides descriptive word-based clues to narrow down the correct numerical sequence, or in Hangman, which relies on the progressive revelation of correct letters to help the player deduce the hidden word [87]. Thus, hints transform from simple aids into effective pedagogical instruments that enhance logical reasoning and deduction skills.

## Conclusion

In conclusion, this chapter has established the theoretical and technical foundations necessary for the development of serious games. The exploration of the Pygame framework demonstrates its effectiveness as a flexible environment for building interactive systems through the management of the Game Loop and real-time event handling [23]. Furthermore, the structural analysis of game components reveals that the separation between core game logic and input/output interfaces provides the essential architectural framework for integrating complex features [24].

Building upon these technical principles, the requirements of the "Techno Quest" project introduce additional challenges that extend beyond general game development, specifically the integration of an Intelligent Hint System and the adaptation of the library to support the Arabic language (in terms of character reshaping and right-to-left orientation). This conceptual framework serves as a blueprint for the next chapter, where the focus will shift from theoretical descriptions to the practical software engineering and algorithmic implementation required to handle Arabic text and adaptive pedagogical scaffolding.

# *Chapter 3: Engineering and Implementation*

## Introduction

This chapter represents the practical core of this research, marking the transition from theoretical and analytical frameworks to the technical realization of the "TechnoQuest" project. The fundamental objective of this chapter is to provide a comprehensive and in-depth vision of the game's engineering, focusing on the delicate balance between pedagogical requirements and programming constraints.

The journey begins with defining the overall system architecture based on the principle of modularity, which ensures the independence of software units and facilitates future development. Furthermore, this chapter highlights the selected technical tools, primarily the Python language and the Pygame library, justifying these choices based on their efficiency in handling Arabic language challenges and User Interface (UI) requirements.

Additionally, the algorithmic logic of the intelligent three-tier hint system is detailed, illustrating how this system integrates with JSON databases to ensure a seamless flow of information. Through the use of Unified Modeling Language (UML) diagrams, we document the dynamic interactions between system components, providing a clear technical roadmap of the implementation process.

## 3.1Project Overview "TechnoQuest"

### 3.1.1 Game Concept

Techno Quest is an educational intellectual game within the puzzle genre, based on the classic crossword format but specialized in computer science and modern technologies. The game aims to deliver technical educational content through an engaging entertainment medium, where players interact with a grid of empty cells to be filled with precise programming and technical terminology. To realize this interactive experience, the game is being developed using the Python programming language and the Pygame library for both graphical user interface design and game logic. The core gameplay mechanic relies on providing "Technical Clues" that describe the required term, requiring the player to deduce the correct word.

What distinguishes Techno Quest is its intersection system, where shared letters assist the player, enhancing logical deduction skills and transforming the learning process into a challenging, interactive experience.

### 3.1.2 Educational Goals

Techno Quest goes beyond being a mere entertainment tool, its primary objective is to bridge the gap between theoretical memorization and practical cognitive application within the technical field. The project aims to achieve several pedagogical goals, most notably reinforcing core technical terminology for students through consistent practice. Rather than relying on traditional rote learning, the game emphasizes interactive learning via a hint system that encourages players to analyse definitions and establish logical links between concepts.

Furthermore, the game aims to expand the players' technical vocabulary (in both Arabic and English), assisting beginners and university students in grasping scientific material more fluidly. It also seeks to develop their technical problem-solving skills through the deductive reasoning required to navigate and complete the crossword grid.

### 3.1.3 Target Users

Techno Quest is designed to reach a broad audience of tech enthusiasts, focusing primarily on university students, particularly those in Computer Science, Information Technology, and Telecommunications. The game aims to help these students review their academic concepts in a nontraditional manner, free from the pressure of formal examinations. Furthermore, the game targets beginners and hobbyists in the tech world who wish to build a solid linguistic and cognitive foundation. It provides an accessible and enjoyable entry point

for understanding complex terminology. Thanks to its simple interface and progressive difficulty levels, the game serves as an ideal educational tool for anyone seeking to enhance their digital literacy and develop the skills needed to link technical definitions with their correct terms.

### 3.1.4 Gameplay Structure & Mechanism

Techno Quest utilizes a consistent grid architecture to ensure a uniform and organized user experience. All levels (Easy, Intermediate, and Hard) are designed to contain the exact same number of words and the same number of intersections between horizontal and vertical entries. The core philosophy behind the difficulty progression lies in the "cognitive depth" and complexity of the terminology rather than the physical size of the grid. The levels are structured as follows:

**Easy Level**: Includes common and fundamental technical terms, where the clues are direct and easy to deduce.

**Intermediate Level**: Raises the challenge to include programming and structural concepts that require prior knowledge of computer science basics.

**Hard Level**: Focuses on highly specialized and deep technical terminology, featuring more complex clues that require precise technical analysis.

In terms of gameplay mechanisms, the system allows free navigation within the grid, offering a "Progressive Hint System" to balance the term's difficulty with the player's progress. Maintaining a constant grid structure across all levels ensures that "technical knowledge" is the primary and sole metric for the player's advancement and mastery of the game.

### 3.1.5 Functional Requirements

The core functions of the Techno Quest system are governed by a set of programmatic rules that ensure smooth interaction and a fair educational experience. These requirements include:

- **Progressive Hint System**: The system provides three levels of assistance for each word, revealed sequentially:

  Level 1: Provides a the first letter of the word.

  Level 2: Provides a conceptual definition or illustrative examples of the term.

  Level 3: Reveals the complete term in English.

- **Scoring System**: The system employs a precise "Reward and Penalty" mechanism; the player is awarded 100 points for every correct answer, while 50 points are deducted for each requested hint level.

**Auto-Navigation Mechanism**: To enhance user experience, the cursor automatically moves to the next cell upon typing a letter, ensuring a seamless typing flow without the need for manual navigation.

- Real-time Validation and Feedback:

Correct Entry: The word is highlighted in Green and becomes permanently fixed in the **grid**, meaning it cannot be deleted or modified.

Incorrect Entry: If the entered word is incorrect, the system automatically clears the input as soon as the word attempt is completed, prompting the player to try again.

- **Data Management**: Saving and retrieving player progress and scores using (JSON) files to ensure gameplay persistence and continuity.

## 3.2 System Design

### 3.2.1 Architecture

The architectural design of "Techno Quest" is built upon the Modular Architecture principle, where the project is subdivided into independent modules that exchange data seamlessly. This functional separation ensures system efficiency, maintainability, and scalability. The core architecture consists of the following components:

- **Core Logic Module (main.py):** Serves as the central nervous system, managing the Game Loop, processing keyboard events, and coordinating between modules to execute word validation logic.
- **Grid Management Module (grid.py):** Acts as the engineering core of the application. It is responsible for mapping textual data onto a 2D coordinate system, managing cell coordinates, and handling complex intersections to ensure words share the correct characters. It also tracks cell states (empty, occupied, or active).
- **User Interface Module (ui.py):** Dedicated to translating programmatic logic into visual elements. It handles grid rendering, hint displays, and font processing, while ensuring visual compatibility for Right-to-Left (RTL) text direction.
- **Data Layer Module (words.py):** Functions as a structured database containing the technical dictionary, associated hints, and the spatial coordinates for each word within the grid.

❖ **Sub-systems Module (hint_system.py):** A specialized module for managing the arithmetic operations related to reward points and the hint system's deduction logic

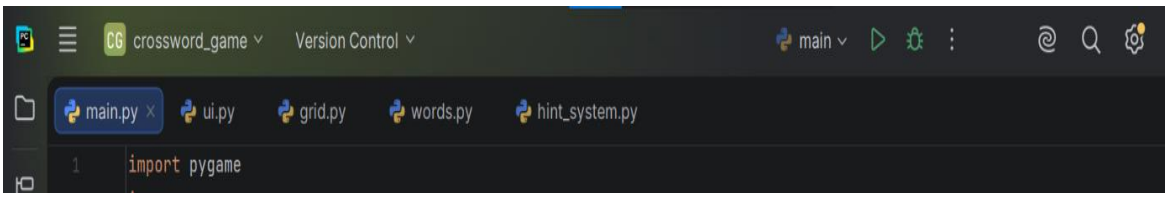



Figure 3.1: TechnoQuest Architecture Modules Layout in PyCharm IDE.

### 3.2.2 UML Overview

The architectural design of Techno Quest is documented using Unified Modelling Language (UML) to provide a clear blueprint of the system's functionality and its underlying object-oriented structure.

- **Use Case Diagram**

The Use Case Diagram illustrates the functional requirements and the interactions between the Player and the Techno Quest System:

- **Level Management:** The player can select between Easy, Medium, and Hard levels, with the system enforcing a progression lock where higher levels are only accessible after completing the preceding ones.
- **Gameplay Interaction**: Core actions include entering letters to solve the grid, which triggers automatic progress saving and the unlocking of subsequent challenges.
- **Strategic Hint System:** The player has access to three distinct hint types to assist with difficult terms, though each request carries a strategic cost of 50 points.

Accountability and Control: Players can view their current score, reset their overall game progress, or exit the application.

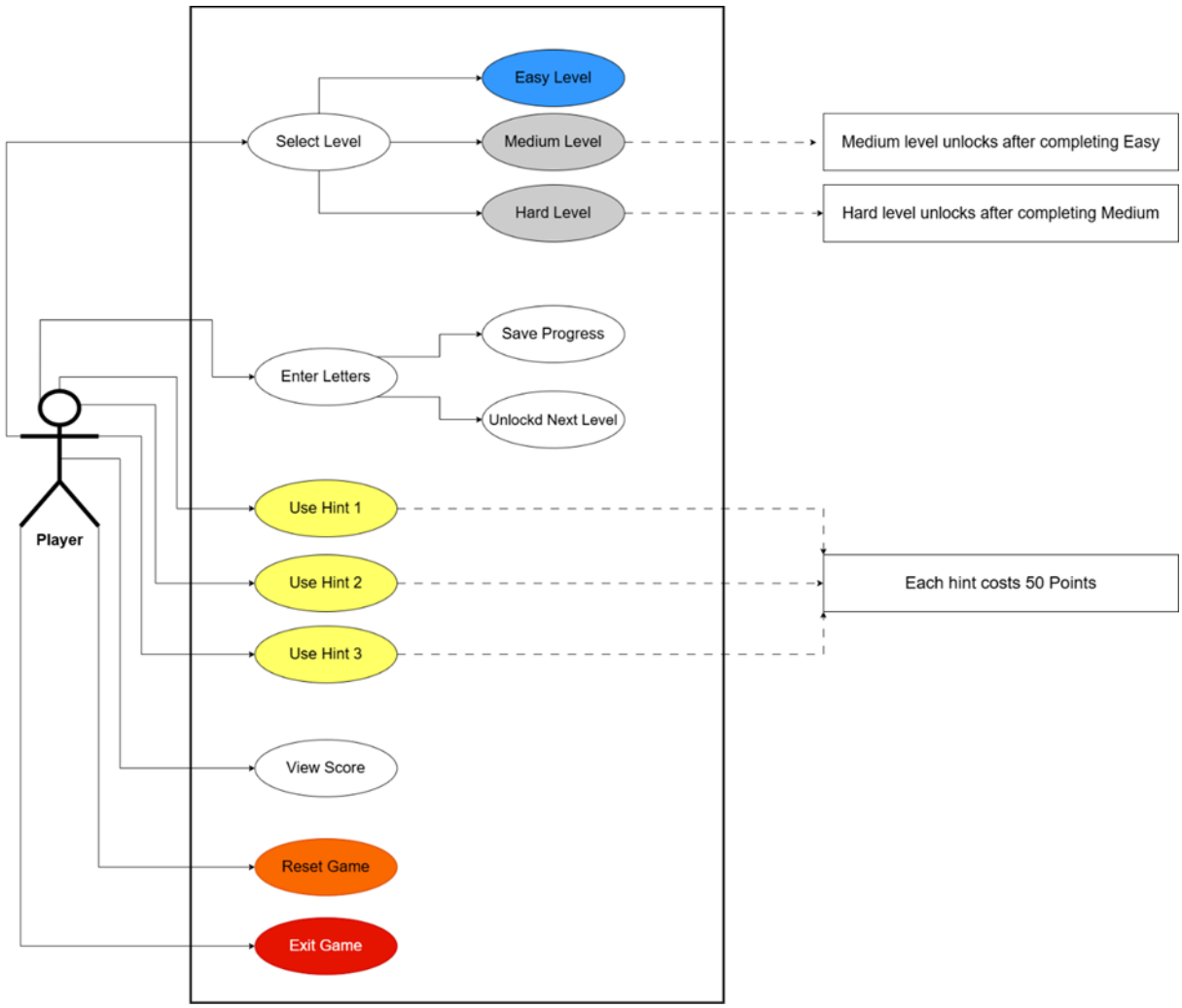



Figure 3.2: TechnoQuest Use Case Diagram

- **Class Diagram**

The Class Diagram defines the static structure of the system, showcasing how different modules interact to maintain the game state and user interface:

- **Game Manager**: Acts as the central controller, managing total points, level unlocking logic, and the main game loop.
- **Bimodule**: A specialized module dedicated to rendering the interface, including score displays, sidebars, and handling the specific formatting required for Arabic text.
- **Word Object**: Encapsulates the data for each technical term, including the target word, its associated technical hint, and the collection of cells it occupies.
- **Cell**: Represents the smallest unit of the grid, storing individual character data, grid coordinates, and selection states.
- **Hint Module**: Manages the logic for revealing hints and communicating point deductions back to the Game Manager.

- **Data Handler:** Handles all input/output operations, ensuring that user progress and scores are persistently stored in a designated JSON file path.

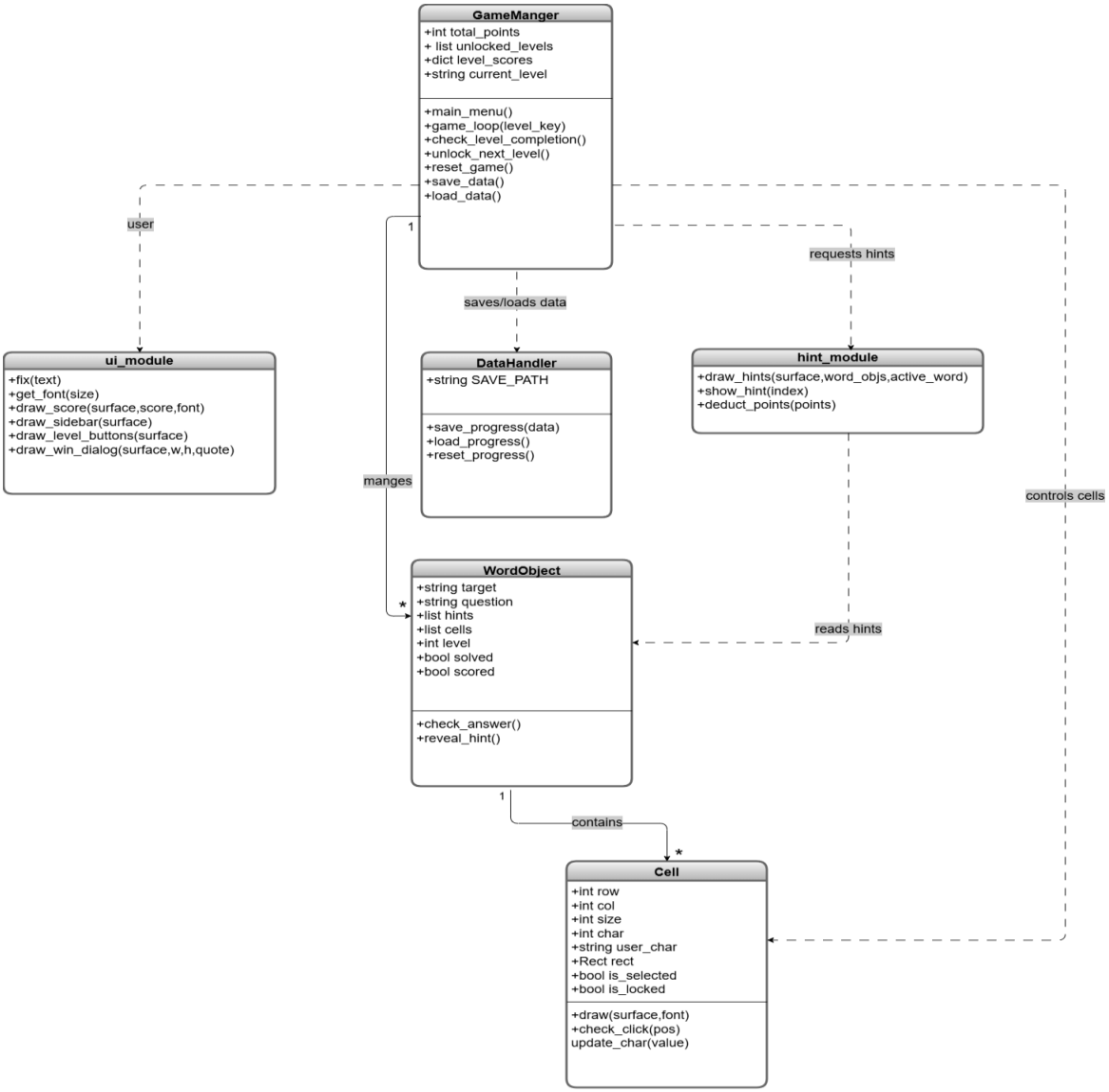



Figure 3.3: TechnoQuest Class Diagram

**Sequence Diagram**

The Sequence Diagram describes the step-by-step process of how the system handles a player's input and validates their answers:

Input Initiation: The process begins when the Player performs a "Key Press" interaction with the UI / Cell component.

Character Update and Rendering: The UI sends an "Update Char" request to the Game Logic, which then returns a "Render Text" instruction back to the UI to display the character on the screen.

Word Validation: Once a word is entered, the UI triggers the "Validate Word" process. The Game Logic communicates with the Data Store to "Check Target Word" and receives a "Return Result".

Conditional Outcomes:

[Correct Answer]: If the answer is valid, the Game Logic instructs the UI to "Lock Cells" and "Update Score," while simultaneously sending a "Save Progress" command to the Data Store.

[Wrong or Full Input]: If the input is incorrect or the word grid is full but invalid, the Game Logic triggers a "Clear Cells" command to reset the specific input area.

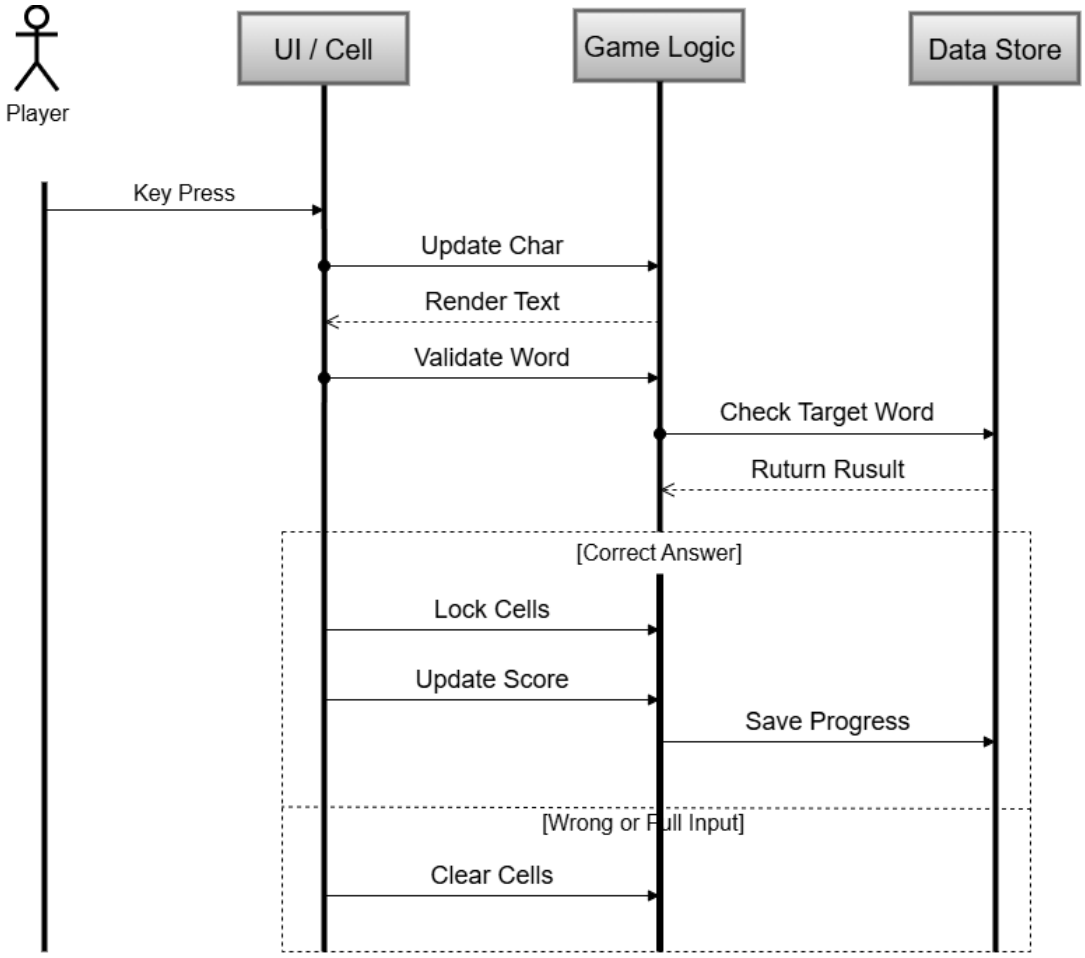



Figure 3.4: TechnoQuest Sequence Diagram

## 3.3 User Interface

### 3.3.1 Main Menu Interface

The Main Menu of TechnoQuest serves as the primary gateway connecting the player to the various components of the game. It is designed following a user centric approach aimed at minimizing cognitive load, ensuring that the learner's focus remains directed toward the educational objectives.

This interface features a dynamic Level Selection system that reflects the outputs of the Progression Algorithm; levels are enabled or locked based on real-time data retrieved from the player's JSON progress files. Additionally, the menu provides essential administrative functions, such as the "Reset Progress" option, which offers full flexibility in managing performance records, alongside the "Exit" command.

From a design perspective, the graphical elements have been optimized for visual balance, with strategic placement of interactive buttons such as the repositioned navigation controls to facilitate seamless transitions between selecting an educational path and engaging with the crossword grid logic


Figure 3.5: The Main Menu Interface of TechnoQuest.

### 3.3.2 Gameplay Screen

The graphical interface of TechnoQuest is built upon a functional partitioning strategy designed to ensure a clear and efficient user experience, dividing the screen into three primary interactive zones that support the learning path. At the top left, the "Status Area" features the

total score panel, utilized with bold typography and high-contrast colors to provide immediate feedback and enhance learner motivation.

The center of the screen is occupied by the "Core Gameplay Area" which hosts the crossword grid , its geometric layout is engineered to facilitate Arabic character input and manage programmatic intersections effectively.

On the right side, the "Question and Control Panel" displays the list of technical terms alongside interactive yellow hint icons, serving as the gateway to the Smart Hint System. This design is finalized with the "Return to Main Menu" button, strategically positioned at the bottom of the panel in a distinct orange hue to improve navigation flow and minimize visual distraction for the user.

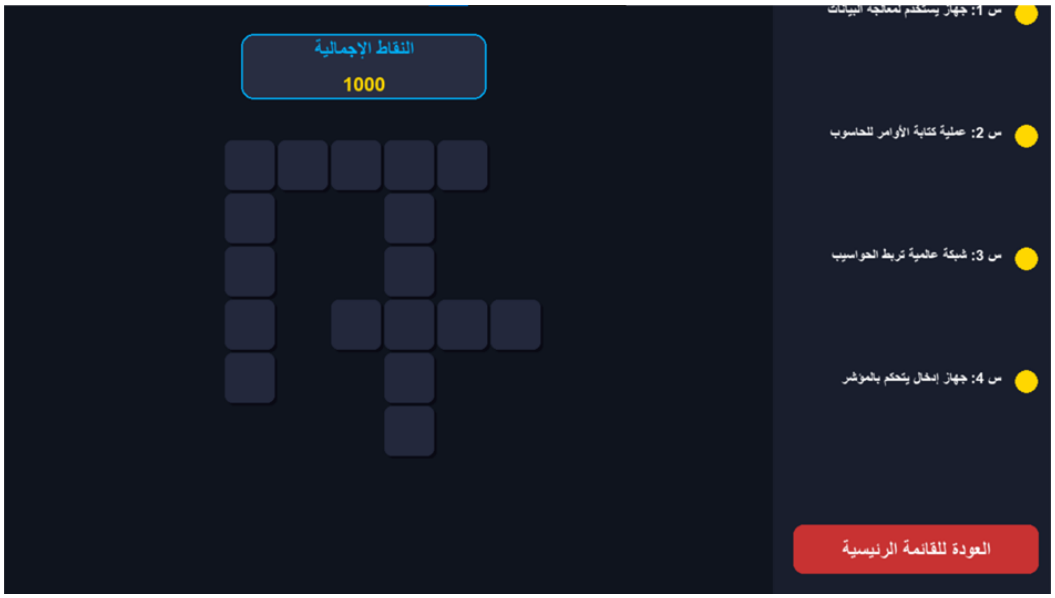



Figure 3.6: The Gameplay Screen Interface of TechnoQuest.

### 3.3.3 Win Screen

The Win Screen serves as the concluding component for each level, providing immediate positive reinforcement once all crossword entries are successfully validated. A central pop up window featuring a gradient background displays motivational messages, establishing a rewarding milestone before the next challenge. This interface includes a prominent interactive button to encourage continued engagement with the instructional content. Programmatically, the activation of this screen is tied to the validation algorithm, which triggers the win event when user inputs align with the database, thereby reinforcing the student's sense of technical and cognitive achievement.

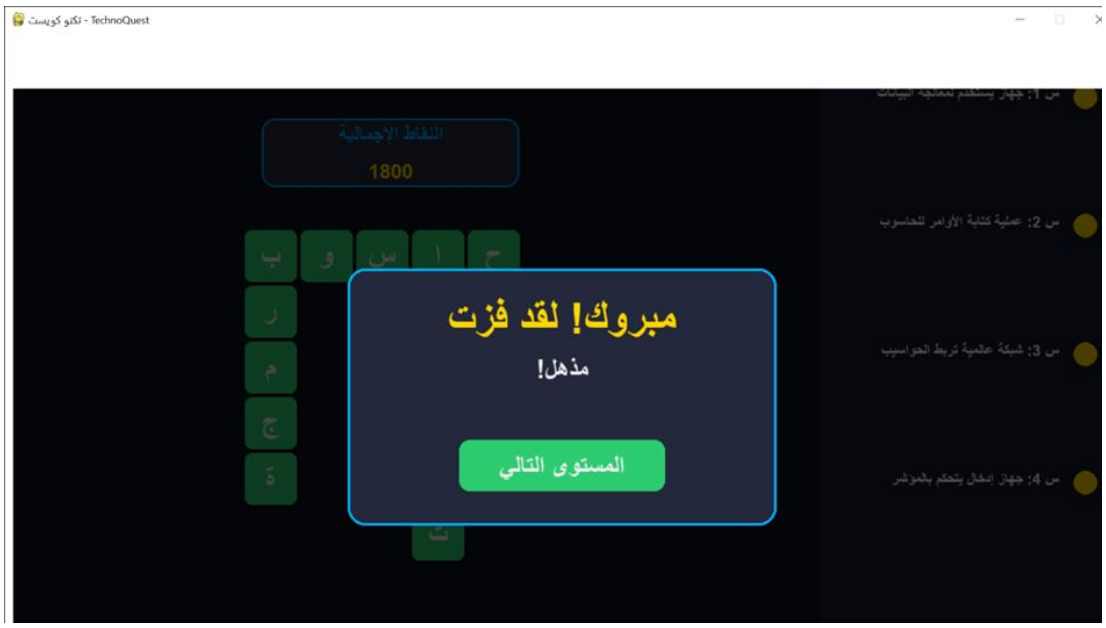



Figure 3.7: The Win Screen Interface of TechnoQuest.

## 3.4 Implementation

### 3.4.1 Development Environment and Tools

The technical implementation of the TechnoQuest system is built upon a specialized software stack designed for high-performance data handling and real-time graphical rendering.

Python 3.12.0 (*python.org* (*https://www.python.org/*)) serves as the core programming language, facilitating robust dictionary-based data modeling and modular software architecture.

The graphical interface and interaction logic are powered by Pygame 2.6.1 (*pygame.org* (*https://www.pygame.org/*)), which leverages the SDL 2.28.4 framework (*libsdl.org* (*https://www.libsdl.org/*)) to provide efficient event handling and spatial rendering of the word grid using (X, Y) coordinate systems.

Furthermore, the development workflow was optimized within the PyCharm IDE (*jetbrains.com/pycharm* (*https://www.jetbrains.com/pycharm*)), utilizing a dedicated virtual environment ( *.venv*) to maintain dependency integrity and ensure system stability.

### 3.4.2 Arabic Text Rendering and Logic

The implementation of Arabic script within the Pygame environment presents a significant technical hurdle, as the framework lacks native support for Right-to-Left (RTL)

orientation and dynamic letter shaping. To overcome this in TechnoQuest, a dual-library processing pipeline was integrated:

**arabic_reshaper Library**: This tool transforms isolated Arabic characters into their context-specific glyphs (initial, medial, or final forms), ensuring correct visual connectivity between letters.

**python-bidi Library**: This library applies the bidirectional algorithm required to reorder the character sequence for RTL display, aligning with standard Arabic orthography.

**Font Integration (Arial):** To ensure maximum compatibility and stable rendering across different Windows-based environments, the project utilizes the Arial typeface via the `pygame.font.SysFont` interface. Arial was selected as the primary rendering engine because it contains comprehensive support for Arabic Unicode glyphs and maintains consistent character spacing within the game's grid system.

This computational sequence reshaping, reordering, and font mapping is executed prior to the rendering phase. This allows the game engine to display technical terminology accurately within the grid cells while maintaining the integrity of the underlying coordinate system (X, Y) and interaction logic.

### 3.4.3 Data Persistence

The TechnoQuest game utilizes a flexible storage system to ensure the permanent preservation of player progress and score balance, allowing the game state to be retrieved even after the program is closed. The JSON (JavaScript Object Notation) format was chosen as the ideal choice for storing this data, as it provides a balance between human readability and programming efficiency when dealing with complex data structures such as dictionaries and lists in Python. The data is organized within a local file named `save_data.json` with a hierarchical structure that includes key-value pairs: points to track the score balance, unlocked to record the levels that have been unlocked, and a sub-dictionary called saves that accurately stores the content of the word grid by saving the character entered by the user along with its associated cell coordinates.

From a programming perspective, this system is implemented through three pivotal functions that ensure data integrity and a seamless user experience. First, the Save function converts programming objects into JSON strings while enabling the ensure_ascii=False property to ensure full compatibility with Arabic characters and prevent them from being encoded incorrectly, in addition to using indentation (indent=4) for technical organization. Second, the Load function runs at the start of each level to verify the file's existence and

restore previous progress, preventing the player from losing their effort. Finally, the Reset function provides a mechanism to clear storage records and return to the default state. To enhance system stability, all these operations are embedded within error-handling blocks (Try...Except) to avoid any sudden game crashes in the event of file loss or external corruption.

### 3.4.4 Intelligent Hint System

The hint system in TechnoQuest is built upon a "Scaffolding" strategy, progressively guiding the player toward the correct solution through three escalating tiers of assistance.

**Level 1** provides a "Guidance Cue" by showing the player the first letter of the word, requiring them to manually input it into the correct cell rather than revealing it automatically, which encourages active interaction.

**Level 2** focuses on the "Conceptual Dimension," offering a detailed explanation of the technical term, its practical uses, and real-world examples to deepen academic understanding.

Finally, **Level 3** offers a "Linguistic Hint" by providing the equivalent term in English, enabling students to bridge the gap between Arabic terminology and global standards in technology. To maintain the game's challenge and encourage independent thinking, the system incorporates an economic mechanism that deducts 50 points for each hint requested. This system is programmatically managed using Python and JSON to ensure that the state of unlocked hints is preserved and remains persistent across different play sessions.

## 3.5 Algorithms and pseudo-code

This section describes the logical backbone of "TechnoQuest." The following algorithms ensure the game functions correctly, handles user input efficiently, and maintains the educational challenge.

### 3.5.1 Grid Positioning and Centering Algorithm

In a crossword game, the grid's size varies by level. This algorithm calculates the necessary offsets `off_x` and `off_y` to ensure that the game board is always rendered in the center of the screen, providing a consistent user experience regardless of the word layout.

**Pseudo-code:**

```
ALGORITHM GridCentering
INPUT: word_data_list, cell_size, screen_width, screen_height
OUTPUT: offset_x, offset_y
BEGIN
```

```
    Initialize empty lists: all_rows, all_columns
      FOR each word_entry IN word_data_list:
       FOR each character_index IN word_length:
          Calculate row (r) and column (c) based on direction
          APPEND r TO all_rows
          APPEND c TO all_columns
    // Determine the boundaries of the grid
    min_r = MIN(all_rows), max_r = MAX(all_rows)
    min_c = MIN(all_columns), max_c = MAX(all_columns
    // Calculate total grid dimensions in pixels
    grid_width = (max_c - min_c + 1) * cell_size
    grid_height = (max_r - min_r + 1) * cell_size
    // Calculate offsets to center the grid on the screen
    offset_x = (screen_width - grid_width) / 2 - (min_c * cell_size)
    offset_y = (screen_height - grid_height) / 2 - (min_r * cell_size)
      RETURN offset_x, offset_y
END
```

### 3.5.2 Word Validation and Cell Locking Algorithm

This algorithm manages the core gameplay loop. It monitors the user's input in real-time, compares it with the target word, and handles the reward system (scoring) and state management (locking cells).

**Pseudo-code:**

```
ALGORITHM ValidateWordEntry
INPUT: word_objects, current_points
OUTPUT: updated_points, cell_states
BEGIN
    FOR each object IN word_objs:
        // Concatenate user characters into a single string
        user_string = Concatenate(cell.user_char FOR cell
IN object.cells)
        // Check if the user's input matches the target
word
        IF user_string EQUALS object.target_word THEN
            FOR each cell IN object.cells:
```

```
                SET cell.is_locked = TRUE // Prevent
further editing
            // Award points only once per word
            IF object.scored IS FALSE THEN
                current_points += 100
                object.is_scored = TRUE
        // If the word is full but incorrect, clear the
cells
        ELSE IF LENGTH(user_string) EQUALS
LENGTH(object.target_word) THEN
            FOR each cell IN object.cells:
                IF cell.is_locked IS FALSE THEN
                    cell.user_char = "" // Reset incorrect
input
    END FOR
END
```

### 3.5.3 Progression and Level Unlocking Algorithm

This algorithm manages the transition between game levels. It ensures that the player follows a structured learning path by unlocking the next difficulty level only upon the successful completion of the current one.

**Pseudo-code:**

```
ALGORITHM LevelProgression
INPUT: current_level, unlocked_levels_list, levels_order
OUTPUT: next_action
BEGIN
  game_result = ExecuteGameLoop(current_level)
    IF game_result IS "WON" THEN
    // Find the position of the current level in the sequence
    current_index = FindIndex(current_level, levels_order)
    // Check if there is a next level to unlock
    IF current_index < (LENGTH(levels_order) - 1) THEN
      next_level = levels_order[current_index + 1]
      IF next_level NOT IN unlocked_levels_list THEN
        APPEND next_level TO unlocked_levels_list
```

```
            SaveDataToFile() // Persist progress to JSON
        RETURN StartLevel(next_level)
    RETURN GoToMainMenu
END
```

### 3.5.4 Scaffolding Hint Algorithm

To assist players without making the game too easy, this algorithm provides a "Hint System." It manages the exchange of game currency points for progressive clues lamps.

**Pseudo-code:**

```
ALGORITHM ProcessHintRequest
INPUT: mouse_click_position, hint_buttons, total_points
OUTPUT: updated_points, updated_hint_level
BEGIN
    FOR each button IN hint_buttons:
// Check if the player clicked a hint and has enough points
        IF mouse_click_position COLLIDES WITH button AND
total_points >= 50 THEN
            IF current_word.hint_level <
total_available_hints THEN
                total_points -= 50
                current_word.hint_level += 1
                SaveState() // Auto-save after using points
    END FOR
END
```

## 3.6 Testing and Evalution

This section details the verification process used to ensure that TechnoQuest functions correctly from both a technical and pedagogical perspective.

### 3.6.1 Test Cases

To verify the system's reliability, a series of Black-Box tests were conducted. These tests focus on the inputs and outputs without delving into the internal code structure.

Table 3.1: TechnoQuest System Test Cases

| Feature under test | Action/input | Expected result | Status |
|---|---|---|---|
| **Arabic language support** | Displaying technical terms in Arabic (e.g., حاسوب). | Characters appear correctly reshaped and connected (RTL) without fragmentation. | pass |
| **Correct word entry** | Typing the correct word matching the definition. | Cells turn Green, characters are locked (cannot be edited), and 100 points are added to the score. | pass |
| **Incorect word entry** | Typing characters that do not match the target word. | The Auto-Clear feature is triggered, emptying the cells for a new attempt. | pass |
| **Hint level 1(first letter)** | Clicking the Hint button for the first time. | The First Letter of the word is revealed in the grid, and 50 points are deducted. | pass |
| **Hint level 2(definition)** | Clicking the Hint button for the second time. | A Technical Definition or example explaining the term is displayed, and 50 points are deducted. | pass |
| **Hint level 3(translation)** | Clicking the Hint button for the third time. | The English Translation of the word is shown to assist the player, and 50 points are deducted. | pass |
| **Level unlocking** | Successfully completing all words in the current grid. | The "Win Screen" appears, and the button to transition to the next level is activated. | pass |

The following figures illustrate some of the test cases described in Table 3.1, showing the game's interface during correct and incorrect interactions.

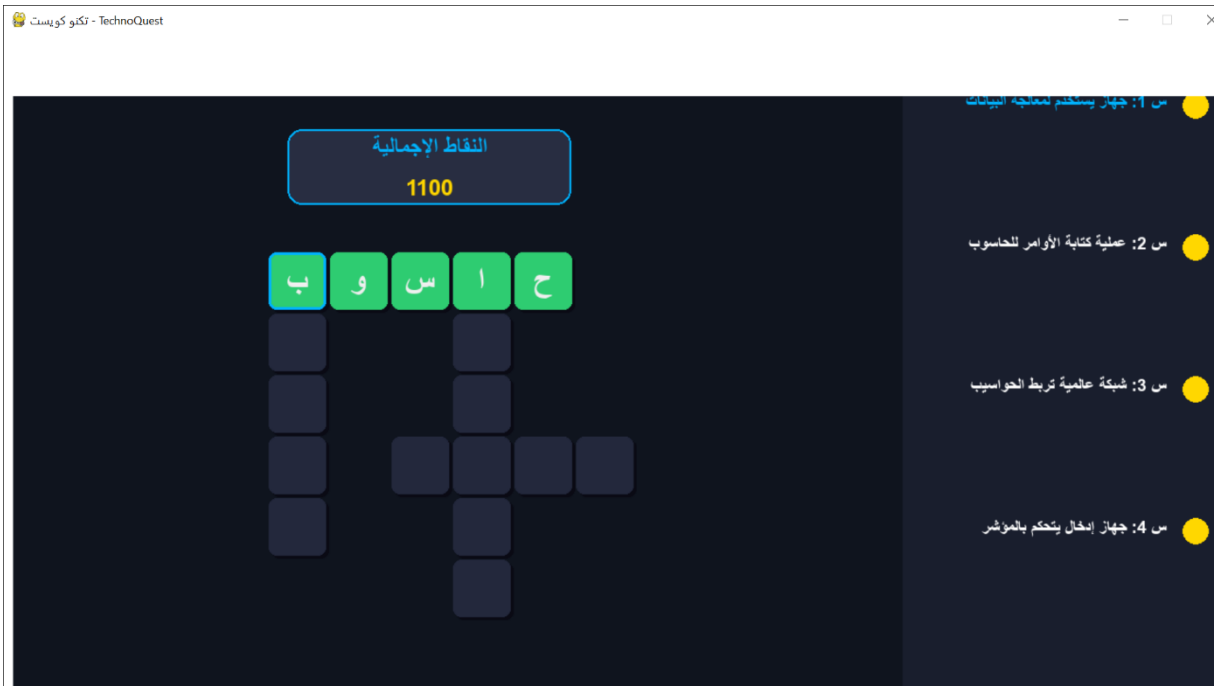



Figure 3.8: Arabic support and correct answer validation.

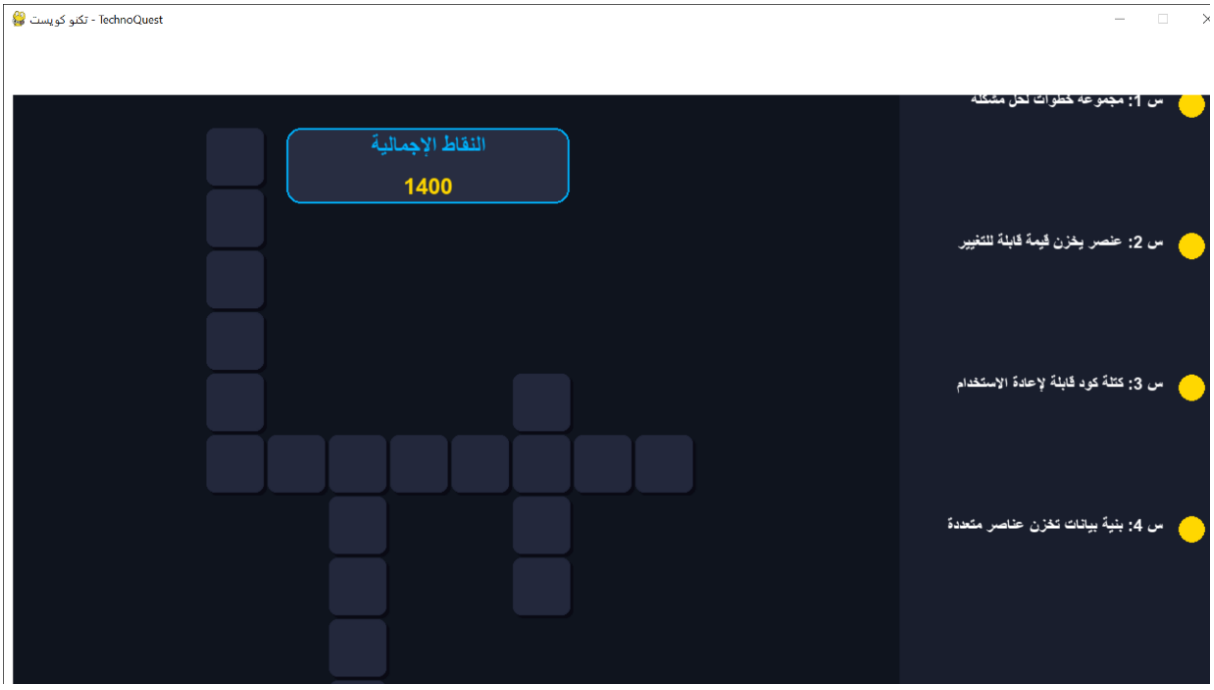



Figure 3.9: Initial score state before hint deductions.

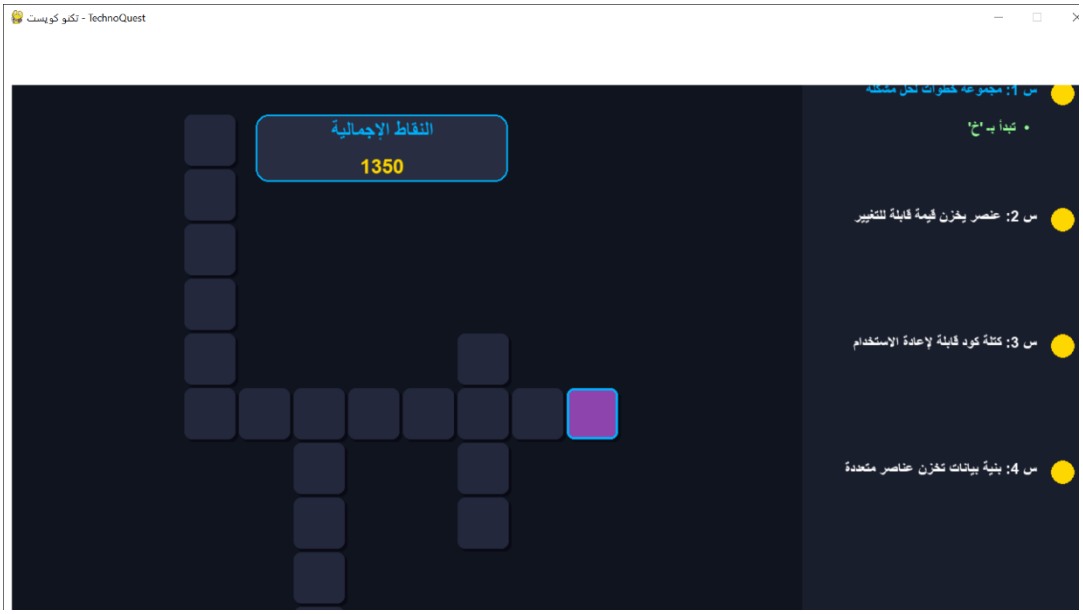



Figure 3.10: The First-Level Hint Activation Interface.

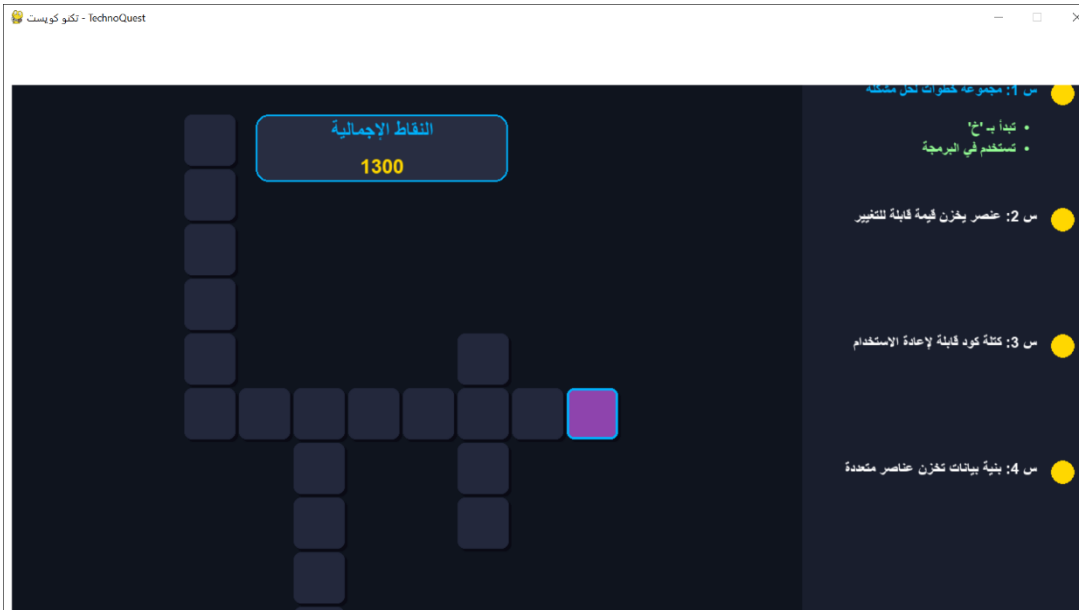



Figure 3.11: The Second-Level Hint Activation Interface.

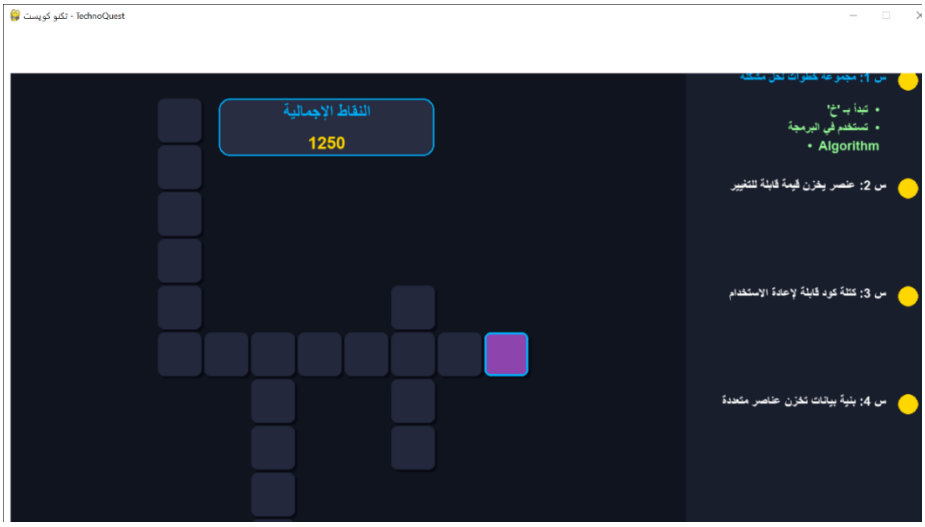



Figure 3.12: The Thrid-Level Hint Activation Interface.


Figure 3.13: "Easy" level unlocked.


Figure 3.14: "Easy" and "Medium" levels unlocked.


Figure 3.15: All difficulty levels unlocked.

### 3.6.2 Performance

The performance of TechnoQuest was evaluated based on resource efficiency and responsiveness to ensure a smooth educational experience. The results are summarized as follows:

**Responsiveness**: The game exhibits high responsiveness; the time elapsed between user input (keyboard entry or mouse click) and the system's reaction (word validation or hint display) is near-instantaneous (less than 100ms).

**Resource Efficiency**: Developed using Python and Pygame, the application maintains a low memory footprint, ensuring compatibility with standard educational computers without requiring high-end hardware.

**Stability**: During extended testing sessions across multiple levels, the system remained stable with no crashes or memory leaks, even when handling complex Arabic text rendering.

**UI Fluidity**: The transition between different game states (Main Menu, Gameplay, and Win Screen) is seamless, maintaining a consistent frame rate (FPS) throughout the session.

## 3.7 Discussion

This section evaluates the "TechnoQuest" system from both pedagogical and technical perspectives, analysing its current effectiveness while acknowledging the challenges encountered during the development phase.

#### 3.7.1 Strengths

The system demonstrates several key advantages that enhance its value as an educational tool:

- **Smart Hint System**: This feature serves as a significant educational pillar, providing progressive scaffolding. It encourages students to use deduction and comprehension rather than random guessing, thereby fostering independent learning.
- **Arabic Language Integration**: A major technical achievement of this project is the successful handling of Arabic text and dynamic character linking within the Pygame environment, addressing a notable gap in Arabic-centric technical educational software.
- **Gamified Learning**: By transforming dry programming terminology into an interactive crossword challenge, the system effectively bridges the gap between theoretical knowledge and engaging practice.

### 3.7.2 Limitations

Despite its functional success, certain limitations persist in the current version:

**Database Scope**: The current technical vocabulary is relatively limited, which may affect the long-term engagement of more advanced learners.

**Static Logic**: The hint system relies on pre-defined static strings within the database. It currently lacks adaptive Artificial Intelligence (AI) capable of generating hints tailored to an individual's learning pace.

**Single-Player Constraint**: The project is currently restricted to a single-player mode, missing out on the collaborative and competitive dynamics found in multiplayer educational environments.

## 3.8 Future Improvements

To evolve "TechnoQuest" into a more comprehensive platform, the following enhancements are proposed for future development:

- **AI Integration**: Incorporating Machine Learning algorithms to generate dynamic, adaptive hints that respond to the user's performance level.
- **Expanded Vocabulary**: Broadening the database to include diverse computer science domains, ensuring the content remains fresh and challenging.
- **Multiplayer and Collaborative Modes**: Developing network capabilities to allow students to compete or collaborate in real-time, enhancing motivation through social learning.
- **UI/UX Optimization**: Refining the graphical interface with advanced animations and interactive sound effects to create a more immersive user experience.

## Conclusion

With the completion of this chapter, the foundational building blocks of the "TechnoQuest" technical product have been established, transforming initial concepts into a tangible interactive educational environment. The design and implementation phase has demonstrated the capacity of the adopted architecture to accommodate the complexities associated with educational games, particularly regarding the execution of the multi-level hint system and the effective management of bilingual (Arabic and English) data.

The initial results of the implementation process indicate that the choice of Object-Oriented Programming (OOP) was decisive in organizing the source code and facilitating the management of in-game objects, such as the grid, words, and interfaces. However, this work opens the door to fundamental enhancements; technically, there is potential to integrate Artificial Intelligence algorithms to make hints more adaptive to the user's learning style, and functionally, the scope can be expanded to include multiplayer modes. Based on what has been achieved, this research concludes by presenting this prototype as an educational tool that merges the engagement of gaming with the precision of technical knowledge, laying a solid cornerstone for the development of future, more intelligent educational applications.

# *General Conclusion*

In conclusion, this research has shown that the Python programming language represents one of the most important modern tools in software engineering due to its balance between simplicity, efficiency, and flexibility. The study demonstrated that the continuous evolution of Python, especially in its recent versions, has significantly enhanced developers' productivity and expanded its applications in educational and interactive systems.

Through the study of the Pygame library, it was found that it provides a suitable environment for developing educational and serious games, thanks to its integrated tools for handling graphics, sound, and event management. This allows developers to focus on game logic and educational objectives without dealing with the complexity of low-level programming.

The practical application of this study culminated in the development of "TechnoQuest", an educational serious game designed to bridge the learning gap in technical terminology. The study led to several key findings, including:

The importance of pedagogical design in serious games: The success of TechnoQuest demonstrated that technical implementation must be integrated with psychological and educational principles to ensure learning while maintaining interaction and motivation.

Overcoming Technical Constraints for Arabic Support: The project successfully addressed the limitation of Pygame regarding Right-to-Left (RTL) text rendering, providing a viable software solution for Arabic educational game development.

The Effectiveness of Scaffolding through a Smart Hint System: Implementing a structured, pedagogical Smart Hint System proved effective in reducing cognitive load on students and encouraging deductive thinking rather than random guessing.

The technical flexibility offered by Python and Pygame in building maintainable and scalable applications, while enabling the application of Object-Oriented Programming (OOP) concepts to organize and improve code structure.

Finally, this work highlights the importance of using modern programming technologies in the development of interactive educational tools that combine learning and entertainment. It also opens the way for future research aimed at incorporating advanced artificial intelligence techniques to create more adaptive educational games that personalize the learning experience based on each student's capabilities.

# *References*

[1] M. Lutz, Learning Python: Powerful Object-Oriented Programming, 4th ed. Sebastopol, CA: O'Reilly Media, 2009.

[2] W. McGugan, Beginning Game Development with Python and Pygame: From Novice to Professional. Berkeley, CA: A press, 2007.

[3] J. P. Gee, What Video Games Have to Teach Us About Learning and Literacy. New York: Palgrave Macmillan, 2003.

[4] Python Software Foundation, "Executive Summary," Python.org. [Online]. Available : https://www.python.org/doc/essays/blurb (https://www.python.org/doc/essays/blurb/)/.

[5] K. Becker, Choosing and Using Digital Games in the Classroom: A Practical Guide. Switzerland: Springer International Publishing, 2016.

[6] A. Sweigart, Making Games with Python & Pygame. CreateSpace, 2012.

[7] Pygame Community, "Pygame Documentation," Pygame.org. [Online]. Available: https://www.pygame.org/docs (https://www.pygame.org/docs/)/.

[8] M. Lutz, Learning Python, 5th ed., O'Reilly Media, 2013.

[9] D. Beazley, Python Essential Reference, 4th ed., Addison-Wesley, 2009.

[10] L. Ramalho, Fluent Python, 2nd ed., O'Reilly Media, 2022.

[11] E. Matthes, Python Crash Course, 2nd ed., No Starch Press, 2019.

[12] A. Sweigart, Automate the Boring Stuff with Python, 2nd ed., No Starch Press, 2019.

[13] M. H. Goldwasser and D. Letscher, Object-Oriented Programming in Python, Pearson, 2014.

[14] M. H. Goldwasser and D. Letscher, Object-Oriented Programming in Python, Pearson, 2014.

[15] D. Phillips, Python 3 Object Oriented Programming, Packt Publishing, 2010.

[16] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems 32 (NeurIPS 2019), 2019, pp. 8024–8035. [Online]. Available: PyTorch

[17] M. Lutz, Learning Python, 5th ed., O'Reilly Media, 2013.

[18] D. Beazley, Python Essential Reference, 4th ed., Addison-Wesley, 2009.

[19] L. Ramalho, Fluent Python, 2nd ed., O'Reilly Media, 2022.

[20] A. Sweigart, Automate the Boring Stuff with Python, 2nd ed., No Starch Press, 2019.

[21] D. Phillips, Python Object-Oriented Programming, 3rd ed., Packt Publishing, 2018.

[22] D. Beazley and B. K. Jones, Python Cookbook, 3rd ed., O'Reilly Media, 2013.

[23] A. Sweigart, Making Games with Python & Pygame. San Francisco, CA: Invent with Python, 2012.

[24] W. McGugan, Beginning Game Development with Python and Pygame: From Novice to Professional. Berkeley, CA: Apress, 2007.

[25] BERGONSE, R. (2017). FIFTY YEARS ON: THE DEFINITION OF GAMES REVISITED. THE COMPUTER GAMES JOURNAL, 6(3), 239–253.

[26] BOGOST, I. (2009). UNIT OPERATIONS: AN APPROACH TO VIDEOGAME CRITICISM. MIT PRESS.

[27] DETERDING, S. (2013). THE LENS OF GAME STUDIES. IN THE GAME STUDIES READER. MIT PRESS.

[28] TAVINOR, G. (2008). THE ART OF VIDEOGAMES. WILEY-BLACKWELL.

[29] SALEN, K., & ZIMMERMAN, E. (2004). RULES OF PLAY: GAME DESIGN FUNDAMENTALS. MIT PRESS.

[30] ALEXIOU, A., & SCHIPPERS, M. (2018). DIGITAL GAME ELEMENTS, USER EXPERIENCE AND LEARNING. EDUCATION AND INFORMATION TECHNOLOGIES.

[31] CLARK, D. B., ET AL. (2023). ACADEMICALLY MEANINGFUL PLAY: DESIGNING DIGITAL GAMES FOR LEARNING. COMPUTERS & EDUCATION.

[32] PÖTZSCH, H., ET AL. (2023). DIGITAL GAMES AS MEDIA FOR TEACHING AND LEARNING.

[33] J. P. GEE, "WHAT VIDEO GAMES HAVE TO TEACH US ABOUT LEARNING AND LITERACY," ACM COMPUTERS IN ENTERTAINMENT, VOL. 1, NO. 1, 2003.

[34] J. P. GEE, "ARE VIDEO GAMES GOOD FOR LEARNING?" KEYNOTE SPEECH, ITU-KONFERANSEN, UNIVERSITY OF WISCONSIN–MADISON, 2006.

[35] C. LI, X. GUO, J. WANG, AND S. WANG, "SINGLE VIDEO GAMES IMPROVE COGNITIVE FUNCTIONING IN COLLEGE STUDENTS: EVIDENCE FROM BEHAVIORAL AND FNIRS ASSESSMENTS," FRONTIERS IN PSYCHIATRY, VOL. 16, ART. NO. 1640142, 2025.

[36] LAAMARTI, F., EID, M., & EL SADDIK, A. (2014). AN OVERVIEW OF SERIOUS GAMES. INTERNATIONAL JOURNAL OF COMPUTER GAMES TECHNOLOGY, 2014, 1–11. HTTPS://DOI.ORG/10.1155/2014/358152

[37] MICHAEL, D., & CHEN, S. (2006). SERIOUS GAMES: GAMES THAT EDUCATE, TRAIN, AND INFORM. THOMSON COURSE TECHNOLOGY.

[38] PLASS, J. L., HOMER, B. D., & KINZER, C. K. (2020). FOUNDATIONS OF GAME-BASED LEARNING. EDUCATIONAL PSYCHOLOGIST, 55(4), 1–16.

[39] GARRIS, R., AHLERS, R., & DRISKELL, J. E. (2002). GAMES, MOTIVATION, AND LEARNING: A RESEARCH AND PRACTICE MODEL. SIMULATION & GAMING, 33(4), 441–467.

[40] WOUTERS, P., VAN NIMWEGEN, C., VAN OOSTENDORP, H., & VAN DER SPEK, E. D. (2013). A META-ANALYSIS OF THE COGNITIVE AND MOTIVATIONAL EFFECTS OF SERIOUS GAMES. JOURNAL OF EDUCATIONAL PSYCHOLOGY, 105(2), 249–265.

[41] CONNOLLY, T., BOYLE, E., MACARTHUR, E., HAINEY, T., & BOYLE, J. (2012). A SYSTEMATIC LITERATURE REVIEW OF EMPIRICAL EVIDENCE ON COMPUTER GAMES AND SERIOUS GAMES. COMPUTERS & EDUCATION, 59(2), 661–686.

[42] R. V. Mulder, P. van der Zande, and J. de Greef, "Digital Games for Learning: A Systematic Review on Word-to-Text Integration," The Computer Games Journal, vol. 9, no. 1, pp. 25–45, Mar. 2020.

[43] M. A. Garcia-Ruiz and A. Alvarez, "Current Trends in Game Accessibility: A Review of the Special Issue," The Computer Games Journal, vol. 7, no. 2, pp. 53–57, Jun. 2018.

[44] T. Marsh, "Serious Games: Analysis and Design from a Phenomenological Perspective," The Computer Games Journal, vol. 5, no. 1, pp. 5–15, Apr. 2016.

[45] K. THANASUAN AND S. T. MUELLER, "IMPROVING LEXICAL MEMORY ACCESS AND DECISION-MAKING PROCESSES USING COGNITIVE WORD GAMES," DEPT. COGN. LEARN. SCI., MICHIGAN TECHNOLOGICAL UNIV., HOUGHTON, MI, USA, 2014.

[46] W. KHAEWRATANA, "WORD GAMES FOR EDUCATION: INVESTIGATING THE EFFECTIVENESS OF ADDING ELABORATION TASKS TO CROSSWORDS FOR LEARNING TECHNICAL VOCABULARY," PH.D. DISSERTATION, DEPT. COGN. LEARN. SCI., MICHIGAN TECHNOLOGICAL UNIV., HOUGHTON, MI, USA, 2022.

[47] B. BOUZAIANE AND A. YOUZBASHI, "THE ROLE OF DIGITAL-GAME BASED LANGUAGE LEARNING IN EFL VOCABULARY LEARNING AND RETENTION:

A CASE STUDY AT A HIGHER EDUCATIONAL INSTITUTE IN OMAN," J. LANG. TEACH. RES., VOL. 15, NO. 5, PP. 1660-1669, SEPT. 2024.

[48] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME, SAN FRANCISCO, CA, USA: NO STARCH PRESS, 2012.

[49] PYGAME COMMUNITY, "PYGAME DOCUMENTATION," 2025. [ONLINE]. AVAILABLE: HTTPS://WWW.PYGAME.ORG/DOCS/ (ACCESSED: APR. 7, 2026).

[50] PYTHON SOFTWARE FOUNDATION, "PYTHON DOCUMENTATION," 2025. [ONLINE]. AVAILABLE: HTTPS://WWW.PYTHON.ORG/ (ACCESSED: APR. 7, 2026).

[51] PYGAME, "DISPLAY MODULE DOCUMENTATION," 2025. [ONLINE]. AVAILABLE: HTTPS://WWW.PYGAME.ORG/DOCS/REF/DISPLAY.HTML (ACCESSED: APR. 7, 2026).

[52] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME, 2012.

[53] PYGAME COMMUNITY, "PYGAME DOCUMENTATION," HTTPS://WWW.PYGAME.ORG/DOCS/, ACCESSED 2026.

[54] PYGAME COMMUNITY, "EVENT HANDLING — PYGAME.EVENT MODULE," PYGAME DOCUMENTATION, N.D. ACCESSED: APRIL 7, 2026. AVAILABLE: HTTPS://WWW.PYGAME.ORG/DOCS/REF/EVENT.HTML

[55] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME, AL SWEIGART, 2012.

[56] K. G. NALBANT, M. D. KAYGUSUZ, AND D. TUNC, "EDUCATIONAL GAME DESIGN FOR KIDS: A CASE STUDY WITH PYTHON MAZE GAME," WSEAS TRANSACTIONS ON ADVANCES IN ENGINEERING EDUCATION, VOL. 22, 2025, DOI: 10.37394/232010.2025.22.11.

[57] E. MATTHES, PYTHON CRASH COURSE: A HANDS-ON, PROJECT-BASED INTRODUCTION TO PROGRAMMING, 3RD ED., NO STARCH PRESS, 2022.

[58] A. SWEIGART, MAKING GAMES WITH PYTHON AND PYGAME, CREATESPACE INDEPENDENT PUBLISHING PLATFORM, 2012.

[59] R. HUNICKE, M. LEBLANC, AND R. ZUBEK, "MDA: A FORMAL APPROACH TO GAME DESIGN AND GAME RESEARCH," 2004.

[60] L. W. ANDERSON ET AL., A TAXONOMY FOR LEARNING, TEACHING, AND ASSESSING, 2001.

[61] W. BECKER AND M. METZ, "DIDACTIC AND PSYCHOLOGICAL PRINCIPLES OF SUCCESSFUL DESIGN OF SERIOUS GAMES," IN PROCEEDINGS OF THE 18TH EUROPEAN CONFERENCE ON GAMES BASED LEARNING (ECGBL 2024), 2024, PP. 1075-1081.

[62] R. DÖRNER, S. GÖBEL, W. EFFELSBERG, AND J. WIEMEYER, EDS., SERIOUS GAMES: FOUNDATIONS, CONCEPTS AND PRACTICE. WIESBADEN: SPRINGER, 2018.

[63] X. AUTHOR AND Y. AUTHOR, "PY-RATE ADVENTURES: A 2D PLATFORM SERIOUS GAME FOR LEARNING THE BASIC CONCEPTS OF PROGRAMMING WITH PYTHON," SIMULATION & GAMING, VOL. 49, NO. 6, PP. 751-767, 2018.

[64] Y. MANYAM, S. SUNDARAM, AND A. SAHAY, "SUDOKU SOLVER ALGORITHMS: A COMPARATIVE ANALYSIS," IN PROCEEDINGS OF THE 2024 FIRST INTERNATIONAL CONFERENCE FOR WOMEN IN COMPUTING (INCOWOCO), BENGALURU, INDIA, 2024, PP. 1-6, DOI: 10.1109/INCOWOCO64194.2024.10863160.

[65] J. SCHELL, THE ART OF GAME DESIGN: A BOOK OF LENSES, 2ND ED. BOCA RATON, FL: CRC PRESS, 2014.

[66] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME. CREATESPACE INDEPENDENT PUBLISHING PLATFORM, 2012.

[67] PYGAME COMMUNITY, "PYGAME DOCUMENTATION," PYGAME.ORG, 2024. [ONLINE]. AVAILABLE: HTTPS://WWW.PYGAME.ORG/DOCS/. (HTTPS://WWW.PYGAME.ORG/DOCS/)

[68] A. SWEIGART, INVENT YOUR OWN COMPUTER GAMES WITH PYTHON, 3RD ED. SAN FRANCISCO, CA: CREATIVE COMMONS, 2015.

[69] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME, 1ST ED. SAN FRANCISCO, CA: CREATIVE COMMONS, 2012.

[70] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME. 1ST ED. [ONLINE]. AVAILABLE: HTTP://INVENTWITHPYTHON.COM/PYGAME, (HTTP://INVENTWITHPYTHON.COM/PYGAME) 2012.

[71] T. FULLERTON, GAME DESIGN WORKSHOP: A PLAYCENTRIC APPROACH TO CREATING INNOVATIVE GAMES, 4TH ED. BOCA RATON, FL: CRC PRESS, 2019.

[72] R. HUNICKE, M. LEBLANC, AND R. ZUBEK, "MDA: A FORMAL APPROACH TO GAME DESIGN AND GAME RESEARCH," IN PROCEEDINGS OF THE CHALLENGES IN GAME AI WORKSHOP, NINETEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2004.

[73] A. SWEIGART, INVENT YOUR OWN COMPUTER GAMES WITH PYTHON, 3RD ED. SAN FRANCISCO, CA: INVENTWITHPYTHON.COM, 2015, PP. 98–101.

[74] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME. SAN FRANCISCO, CA: INVENTWITHPYTHON.COM, 2012, PP. 33–51.

[75] Z. ZHANG, S. WANG, AND D. SUN, "A STUDY OF WORDLE GAMES BASED ON MATHEMATICAL MODELS," IN 2024 IEEE 2ND INTERNATIONAL CONFERENCE ON CONTROL, ELECTRONICS AND COMPUTER TECHNOLOGY (ICCECT), 2024, PP. 1–6.

[76] A. KULIKOV AND P. PEVZNER, LEARNING ALGORITHMS THROUGH PROGRAMMING AND PUZZLE SOLVING. SAN DIEGO, CA: ACTIVE LEARNING TECHNOLOGIES, 2019, PP. 117–120.

[77] Pygame Community, "Pygame Logo," Pygame.org. [Online]. Available: https://www.pygame.org/docs/_static/pygame_tiny.png. [Accessed: May 14, 2026].

[78] HAO HE, YULIN ZHU, AND WEI CAI. "AN ADAPTIVE HINT SYSTEM FOR PUZZLE GAMES: A MULTIMODAL-BASED APPROACH." IN 2022 IEEE GAMES, ENTERTAINMENT, MEDIA CONFERENCE (GEM). IEEE, 2022.

[79]CHOI, H., JOVANOVIC, J., POQUET, O., BROOKS, C., JOKSIMOVIĆ, S., & WILLIAMS, J. J. (2023). THE BENEFIT OF REFLECTION PROMPTS FOR ENCOURAGING LEARNING WITH HINTS IN AN ONLINE PROGRAMMING COURSE. THE INTERNET AND HIGHER EDUCATION, 58, 100903. HTTPS://DOI.ORG/10.1016/J.IHEDUC.2023.100903

[80] A. SWEIGART, MAKING GAMES WITH PYTHON & PYGAME. SAN FRANCISCO, CA: INVENT WITH PYTHON, 2012.

[81] A. SWEIGART, INVENT YOUR OWN COMPUTER GAMES WITH PYTHON, 3RD ED. SAN FRANCISCO, CA: NO STARCH PRESS, 2015.

[82] E. O'ROURKE, C. BALLWEBER, AND Z. POPOVIĆ, "HINT SYSTEMS MAY NEGATIVELY IMPACT PERFORMANCE IN EDUCATIONAL GAMES," IN PROC. FIRST ACM CONF. LEARNING @ SCALE (L@S '14), 2014, PP. 1–10.

[83] H. WAUCK AND W. T. FU, "A DATA-DRIVEN, MULTIDIMENSIONAL APPROACH TO HINT DESIGN IN VIDEO GAMES," IN PROC. 22ND INT. CONF. INTELLIGENT USER INTERFACES (IUI ’17), 2017, PP. 1–12.

[84] E. SHIRLEY, "WHY GAME DESIGNERS MAKE HINT SYSTEMS IN GAMES," MASTER'S THESIS, BREDA UNIV. APPLIED SCIENCES, 2024.

[85] D. GOMES, "A COMPREHENSIVE STUDY OF ADVANCEMENTS IN INTELLIGENT TUTORING SYSTEMS THROUGH ARTIFICIAL INTELLIGENT EDUCATION PLATFORMS," IN ADVANCEMENTS IN INTELLIGENT TUTORING SYSTEMS, 2024.

[86] T. Ates and F. Cavdur, "Sudoku puzzle generation using mathematical programming and heuristics: Puzzle construction and game development," Expert Systems with Applications, vol. 282, p. 127710, 2025.

[87] A. Sweigart, Invent Your Own Computer Games with Python, 4th ed. San Francisco, CA: No Starch Press, 2017.

[88] Pygame Community, "Pygame Logo," Pygame.org. [Online]. Available: https://www.pygame.org/docs/_static/pygame_tiny.png. [Accessed: May 14, 2026].