

COMP1002 – Mini-Project Report Template

(PolyU and department of computing logo here)

Project Title: Link Link Game

Team Members:

WANG Yuqi - [REDACTED]

WANG Zihua - [REDACTED]

YANG Jinkun – [REDACTED]

Course Title: Computational Thinking and Problem Solving

Instructor: Dr. Muhammad Tayyab

Department: Computing

Institution: The Hong Kong Polytechnic University

Submission Date: 2023/11/27

Table of Contents

(use the MS word auto generate feature to get table of contents)

- Introduction
- Objectives
- Project Design
- Implementation Details
- Testing and Results
- Conclusions
- References
- Appendices

Introduction

This report comprehensively explores the technical intricacies and developmental processes of a minigame named "Link Link Game." The game's codebase comprises four primary components: the Model Module, the View Module, the Control Module, and the Main Module, collectively forming the Model-View-Control (MVC) architecture. This document will outline the challenges encountered, the strategic structuring of our codebase, and detailed implementations of each module and component.

Objectives

The problem statement delineates the following tasks:

1. Design and display the game board layout on the console.
2. Develop a function for generating a random game board.
3. Create a mechanism for player selection.
4. Construct game logic to identify successful pairings and determine game completion.
5. Conduct comprehensive testing to ensure functionality and address edge cases.
6. Develop a Graphical User Interface (GUI).

Our project is guided by four distinct objectives:

1. Develop a module to manage all game logic.
2. Construct an interactive module for the GUI.
3. Design graphical elements for the GUI, such as various board entities.
4. Perform final testing to confirm the program's reliability and proper operation.

To achieve these goals efficiently and within the stipulated time frame, we allocated five days: the first for coding the logic module, the second and third days for developing the GUI, and the final day for thorough testing.

Project Design

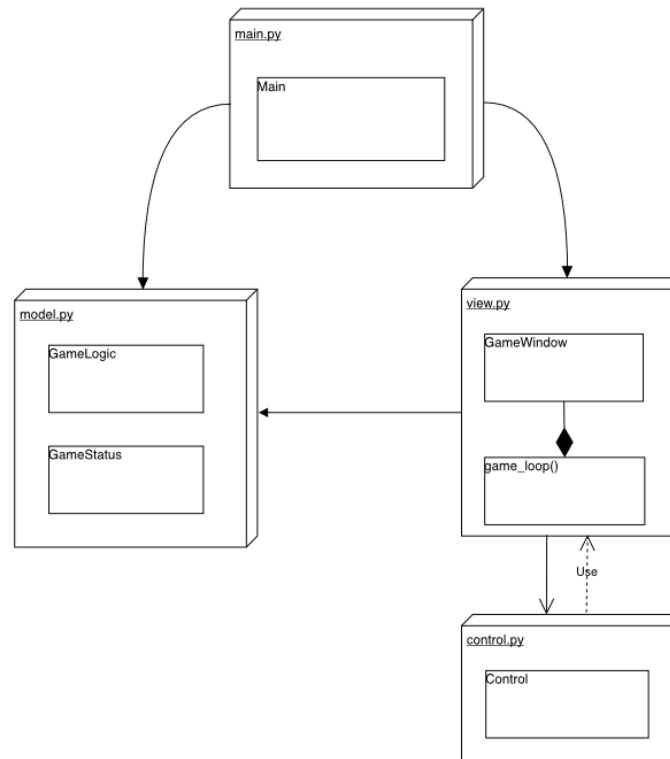


Figure 1: Class Hierarchy Diagram of the Project

This figure illustrates the class hierarchy within our project. The Main Module initiates a `GameLogic` object and a `GameWindow` object derived from the Model and View Modules, respectively. This setup guarantees the initiation of a new game following the conclusion of the preceding one.

The `GameLogic` Object is tasked with managing all logical elements of the game, including:

- Initialization of the game board.
- Ensuring a solvable configuration of the current board.
- Algorithmic validation of element pairings.
- Determination of game completion, marked by the absence of remaining elements.

The `GameWindow` Class is responsible for rendering the Graphical User Interface (GUI) and continuously updating it through a `game_loop()`. It operates as an observer of the `GameLogic` object, maintaining an updated awareness of the game board's status.

Located at the bottom right, the Control Module is utilized exclusively by the View Module. This module's primary function is to convert raw user inputs, such as cursor coordinates, into forms that are compatible with other modules. Additionally, it handles the error correction of any unforeseen user inputs.

Overall Design Approach

Our design approach draws inspiration from the Model-View-Control (MVC) architecture, a prevalent framework in game development. However, due to the integral nature of PyGame's game loop with its GUI rendering, we opted against a strict segregation into distinct View and Control Modules as traditionally seen in MVC models, to avoid adding undue complexity. Furthermore, we implemented specific modifications to the conventional MVC architecture to more effectively cater to the unique requirements of our project ("Model-View-Controller," 2023).

Implementation Details

Programming Constructs & Control Structure

Due to the hierarchical and divisibility of the task, a wide variety of both programming constructs were used. We implemented 4 different modules, classes within the modules, and numerous methods within the classes. Enum classes, dictionaries, and double-ended queue (deque) were also utilized for optimizing performance and organizing the code (*Collections — Container Datatypes*, n.d.).

To ensure maximum readability and conciseness of our code, a wide variety of different control structures were also used, such as loops, conditional statements, function calls, and jump statements.

Modules and Methods

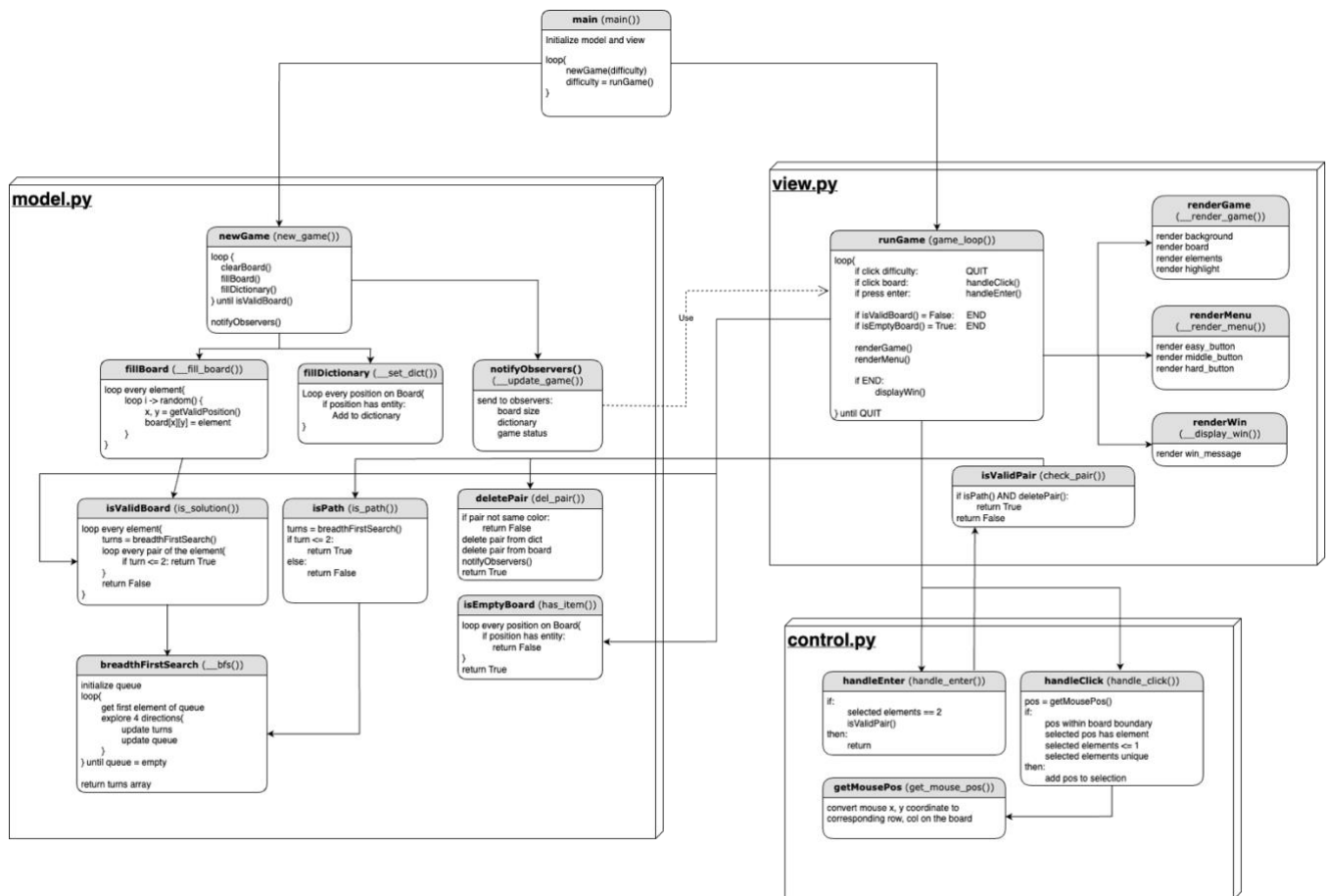


Figure 2: UML Class Diagram of the Project

Figure 2 presents an elaborated version of Figure 1, delineating all principal methods in each module. The arrows symbolize dependency relationships, indicating scenarios where one method's functionality is contingent on another's to execute its tasks effectively.

Model Module (model.py)

Functionality: This module includes a network of interrelated components (`new_game()`) and various individual methods that support the view module's operations. The Breadth-first Search (BFS) algorithm is central to this module, enabling verification of element pair connectivity within two turns and assessing the viability of subsequent moves to avert stalemate situations (Cormen, 2009, pp. 594–595).

Breadth-first Search Adaption: A tailored version of the Breadth-first Search (BFS) algorithm that diverges from the conventional pathfinding approach was created by focusing on minimizing turns rather than distance. Traditional BFS computes the shortest-path matrix, advancing one tile at a time. In contrast, our adapted BFS constructs a minimum-turn matrix, allowing traversal in a

single direction until an obstacle—a game element or a visited tile—is encountered. This specialized adaptation is particularly suited to our game's logic as it aligns with the primary goal.

Optimization: To enhance the efficiency of pairing identical elements on the game board, we implemented an optimization strategy that leverages the speed of hash tables (dictionaries) for indexing. This approach significantly reduces the time complexity of indexing from $O(n^2)$ to $O(n^2/m)$, where n represents the board width, and m denotes the number of element classes. By organizing the coordinates of identical elements within a hash table, we can access and compare these elements with greater speed and less computational effort.

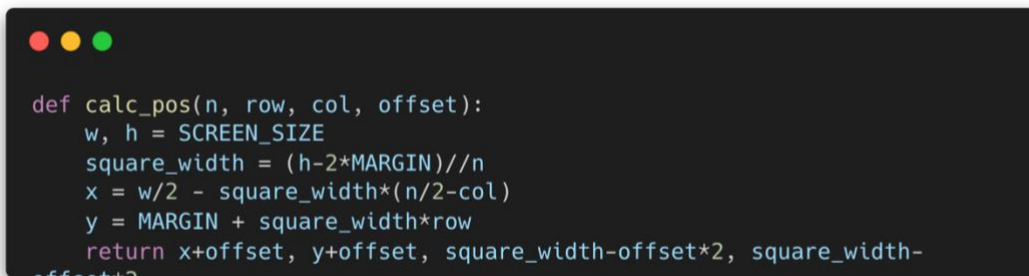
In addition to the hash table optimization, we considered further refining the process by precomputing a matrix that stores cumulative sums of elements from the first tile to the i -th tile in each row. This preprocessing could potentially accelerate the validation of element pairs by providing a quick reference to the number of elements along a path. However, our testing revealed that the performance gains from this preprocessing were outweighed by the overhead it introduced, leading us to forgo this enhancement for the current project scope where $n = 20$.

Furthermore, we substituted Python's native list data structures with NumPy arrays. This is because NumPy arrays offer a more compact memory footprint and are designed for high-performance computation, particularly when handling large arrays of data (Riturajsaha, 2020) (veyak, 2023).

View Module (view.py)

Functionality: The View Module is structured around a principal loop (`game_loop()`) which operates at a consistent 60 frames per second (FPS). Each iteration of the loop is responsible for the following checks:

- **Input Recognition:** Detects keyboard and mouse inputs and conveys this information to the Control Module for processing.
- **Game Termination Validation:** Evaluates whether the game has concluded by verifying if the game board is vacant or if no further moves are possible. Upon game completion, a victory message is displayed.
- **Difficulty Adjustment Monitoring:** Observes any changes in difficulty level made by the user, terminating the loop, reverting to the Main Module, and initiating a game board reset if a change is detected.



```
def calc_pos(n, row, col, offset):  
    w, h = SCREEN_SIZE  
    square_width = (h-2*MARGIN)//n  
    x = w/2 - square_width*(n/2-col)  
    y = MARGIN + square_width*row  
    return x+offset, y+offset, square_width-offset*2, square_width-  
    offset*2
```

Figure 3

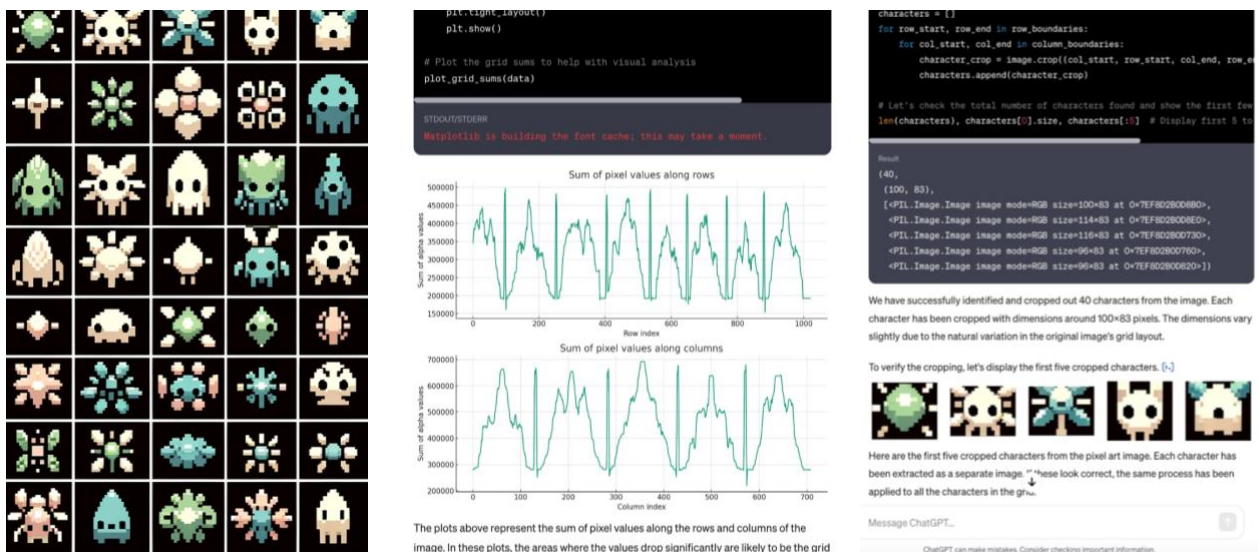
Challenges in Development

One of the significant challenges encountered during the development of this module was the precise calculation of coordinates for rendering shapes and images. As Figure 3 illustrates, this task requires meticulous consideration of multiple variables, where slight miscalculations can result in substantial misalignment within the user interface.

Libraries and Tools Used

This project incorporated several libraries and tools to enhance functionality and performance:

- **PyGame:** Employed for GUI rendering and graphics handling.
- **Deque from Collections:** Integrated for implementing the breadth-first search algorithm.
- **NumPy:** Selected for its array manipulation capabilities and its contribution to performance optimization.
- **Secrets:** Utilized for the generation of true random numbers
- **Enum, auto from Enum:** Applied for the definition and enumeration of constants.



For the creation of visually engaging game elements, the DALL·E 3 model was leveraged. Due to the incompatibility of the model's output image with direct code integration (shown in the left), we instructed ChatGPT to crop out the individual pixel arts by analyzing the aggregate pixel values across the image's rows and columns. This process allowed for the precise extraction and utilization of visual assets within the game.

[illegible]

Conclusions

The culmination of our project aligns with our predefined objectives. Utilizing object-oriented programming techniques, we successfully developed model, view, and control modules that synergistically interact, fulfilling our initial two objectives. Leveraging sophisticated AI tools enabled the creation of a visually striking graphical user interface (GUI), achieving our third objective. Comprehensive testing, including both unit and system tests, fortified the robustness of our code, meeting the fourth goal.

This endeavor significantly enhanced our proficiency in object-oriented programming and provided a foundational understanding of GUI operations. The implementation of the MVC architecture and exploration of observer patterns enriched our software design knowledge. Our Python programming skills, particularly in handling iterables, have evolved substantially, moving beyond traditional looping methods, and embracing more innovative approaches, thereby refining our codebase's clarity and efficiency.

Potential enhancements for this project are manifold. Firstly, augmenting user interaction with visual and auditory feedback during gameplay could notably enhance the user experience. Secondly, addressing the minor lags experienced during element pairing, potentially through the adoption of multithreading, could allow simultaneous UI updates and logical processing. Lastly, the game's current architecture, which heavily taxes a single CPU core at higher difficulty settings (due to larger game board and more elements); This is primarily due to high number of high-resolution screen elements, and could be optimized either through GPU acceleration or by transitioning to actual pixel art which are low resolution.

In terms of its potential as a commercial product, the game's core mechanics present inherent limitations. A substantial redesign of these mechanics would be imperative to transition this project into a viable commercial product.

References

List all the resources used during the development, following a consistent citation style. APA is one option to do citation, however you can follow whichever you find easy to work with.

collections—Container datatypes. (n.d.). Python Documentation. Retrieved November 28, 2023, from <https://docs.python.org/3/library/collections.html>

Cormen, T. H. (Ed.). (2009). *Introduction to algorithms* (3rd ed). MIT Press.

Model–view–controller. (2023). In *Wikipedia*.

<https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1181180757>

Riturajsaha. (2020, February 25). Why is Numpy faster in Python? *GeeksforGeeks*.

<https://www.geeksforgeeks.org/why-numpy-is-faster-in-python/>

veyak. (2023, June 13). Numpy vs Traditional Python Lists: A Performance Showdown. *Medium*.

<https://medium.com/@vakgul/numpy-vs-traditional-python-lists-a-performance-showdown-1e8bebc55933>

Appendices

A - User Manual: Step-by-step instructions on how to use/ interact with the project.

1. Ensure you have Python 3 installed
2. In terminal, type “pip3 install pygame”
3. In terminal, navigate to the project folder
4. Run “python3 main.py”

B - Code Listings: Important code snippets or a link to where the full code can be found (e.g., a GitHub repository). Making the code closed or open source is up to you.

<https://github.com/Cylrx/LinkLinkGame>

C - Testing Evidence: Screenshots of progress, tests, output results, or tables documenting test cases.

Note to Students:

- **Language:** Maintain a formal but accessible tone appropriate for a technical document.
- **Consistency:** Adopt a uniform style for headings, subheadings, and fonts throughout the report.
- **Proofreading:** Review the report for clarity and check for typographical errors before submission.
- **Submission Format:** Submit the report in both PDF and DOCX format.
- **Backup:** Always have a backup of your report files and any related project files.