# 

# 

# **Botany Downs Secondary College**

**Internal Assessment**

**Level 3**

**91906- Use Complex Techniques to Develop a Computer Program-6 credits**

**91907- Use complex processes to develop a digital technologies outcome** **-6 credits**

|  |  |  |
| --- | --- | --- |
| **Achieved** | **Merit** | **Excellence** |
| • Use complex programming techniques to develop a computer program. | • Use complex programming techniques to develop an informed computer program. | • Use complex programming techniques to develop a refined computer program.  **Student Declaration:**  I hereby declare that I have completed the assessment for 91906-907, independently and to the best of my abilities. This assessment represents my own work and is based on my own research, practice and understanding of the subject matter.  I confirm that all sources used in this assessment, including but not limited to books, articles, online resources, and any other references, have been appropriately cited and acknowledged according to the prescribed referencing style.  I further affirm that I have not engaged in any form of academic dishonesty, such as plagiarism or unauthorized collaboration, in the completion of this assessment. The ideas, arguments and content presented in this assessment are my own and have not been copied or reproduced from any other source.  I understand that any act of academic misconduct or violation of the academic integrity may result in disciplinary actions, which could include penalties such as grade reduction, course failure or other consequences as determined by the institution.  I take full responsibility for the authenticity and originality of the assessment and acknowledge that my work will be subject to scrutiny and evaluation by my instructors or assessors.  Signed : Lucas Yu Date: 21/7/25  Signed:  Date: |

**Introduction to chosen task:** (your task should explain the following items. Background Information:

* Objectives
* Significance

)

My inquiry is to develop a memory flashcard game using Python and Tkinter that challenges users to match pairs of hidden cards. The goal is to support memory improvement through an engaging and interactive learning tool, while also demonstrating complex programming techniques like GUI design, object-oriented programming, and state management. The program is mostly for people who have dementia or struggle with memory issues but anyone can use it.

**Methodology**: (Describe the methodology or approach used to tackle the problem. Explain briefly the algorithms, techniques or tools you will employ in the project. This section should provide a clear understanding of how the project will be implemented.)

To develop my memory flashcard game, I plan to use Python and the Tkinter library to create a graphical user interface (GUI) where users can flip and match flashcards. The game will feature three difficulty levels which is going to be easy, medium and hard, with different grid sizes and card pairs. My approach involves using object-oriented programming which will help keep the code organised and maintainable. This makes the game easier to build, fix, and change later for each iteration.

I plan to use lists to store the layout of the cards in a grid format. Each card will have a hidden icon or colour that will only be revealed when clicked. To make the game different each time, the cards will be shuffled randomly using Python’s random module. When a user clicks on a card, I will use Tkinter’s event binding to detect the click and run a function that flips the card. If two selected cards match, they will stay face up. If they do not match, they will flip back after a short delay using Tkinter’s after() which I learnt from W3schools.

* (Python Random module generates random numbers in Python. It introduce randomness into programs. It offers functions that support randomization operations, making it easier to work with unpredictable or varied outputs in different programming scenarios.) – copied from W3schools

To manage my workflow and ensure that I stay on track, I will use Microsoft Planner to break the project into smaller tasks such as planning, designing, coding, testing, and refining. I will use Miro to create flowcharts and visual diagrams to help plan and visualise the program. For writing and testing my code, I will use Visual Studio Code as my main development app. I will be doing testing of each version of the program after completion, and I will collect feedback from stakeholders to improve functionality and usability throughout each iteration.

**Software Requirements**(List down the system requirements and dependencies necessary to run the project, include the version of python used and any external libraries or specific hardware requirements.)

**System Requirements for the Memory Game Application**

* **Software Requirements:**

Visual Studio Code version 1.102 (recommended for the best compatibility)

**Python** with standard libraries:

* + - **Tkinter** – for the graphical user interface
    - **random** – for shuffling the cards
* **Operating System:**

Tested on Windows

Compatibility with other operating systems (e.g., macOS) is unknown since I have never owned or used a macOS device

* **Hardware Requirements:**

A computer capable of running Python

A graphical display (screen resolution of 800x600 pixels or higher recommended)

**List of Complex Techniques being used**:

* **Object-Oriented Programming:**  
  Using classes to organise the game, card data, and GUI elements separately.
* **Event-Driven Programming:**  
  Handling user interactions through Tkinter’s event binding.
* **Randomisation:**  
  Shuffling the cards at the start of each game using Python’s random module to ensure different gameplay each time.
* **Management:**  
  Keeping track of flipped cards, matched pairs, and game progress using variables and lists.
* **Tkinter’s GUI:**  
  Creating a graphical interface with custom icons and coloured cards.
* **Controlled Delay:**  
  Managing timing for flipping cards back if they don’t match, providing a smooth gameplay for the users.
* **Multi-level Design:**  
  Implementing different difficulty levels (e.g. Easy, Medium and Hard) with different grid sizes and card quantities.

List of Complex Processes being used:

* **Using project management techniques**I used Microsoft Planner to organise my tasks into stages such as planning, designing, coding, and testing. I also used GitHub for saving and versioning my code and Miro for visual planning through flowcharts and diagrams.
* **Decomposing the digital outcome into smaller components**I broke down my memory game into separate parts, including the GUI setup, card logic, randomisation, and match-checking functions. This helped make the development process easier to manage and test.
* **Trialling components of the outcome**I trialled parts of the program individually — for example, testing card flipping, matching logic, and the delay timing before combining them into the full game.
* **Testing that the outcome functions as intended**I tested the game to make sure it displays correctly, flips only two cards at a time, matches pairs properly, and resets cards when they don’t match.
* **Addressing relevant implications**I considered functionality, usability, accessibility, ethical design, and end-user requirements to ensure the game is easy to use, safe, and suitable for all audiences.
* **Using information from testing to improve functionality**Based on feedback and my own testing, I improved timing delays, card visibility, and interface spacing to make gameplay smoother and clearer for users.

**Planning Requirements: (How will you plan your application design. This should include a pseudocode, flowchart, project management tools, saving tools and drawing tools.)**

**To plan my Memory game GUI, I plan to use a range of tools to help organise my ideas, design the app structure, and manage the project effectively. I used Microsoft planner to keep track of my tasks and progress. Each task was organised into lists such as “To Do,” “In Progress,” and “Done,” which helped me manage time and stay on track. I wrote pseudocode to outline the logic of the app before starting to code, which helped me understand the flow and structure of functions. To design the flow of the app, I created a flowchart and UML Diagram using Miro, which clearly shows how the user interacts with the interface and how data moves through the system. I used Visual Studio Code for writing and testing my code. For saving and backing up my files, I will save the version of my code on my GitHub.**

**I made my Flowchart using Miro:**

**A diagram of a diagram

AI-generated content may be incorrect.**

**Pseudocode:**

**START PROGRAM**

**OPEN main game window**

**CREATE two game modes:**

**- Easy (4x4 grid)**

**- Medium (6x6 grid)**

**- Hard (8x8 grid)**

**FOR each mode:**

**CREATE list of card pairs (e.g. AABBCC...)**

**SHUFFLE the list randomly**

**SPLIT the list into a 2D grid called answer\_grid**

**CREATE a board of the same size, filled with "." to show face-down cards**

**SET moves to 0**

**SET previous selection to [invalid position]**

**DISPLAY the game board with all cards face-down**

**WHILE game is not complete:**

**WAIT for player to click on a card**

**IF card is already flipped:**

**IGNORE the click**

**ELSE IF this is the first card clicked:**

**FLIP the card and remember its position**

**ELSE IF this is the second card clicked:**

**FLIP the card**

**INCREMENT move counter**

**IF both cards match:**

**KEEP them flipped**

**RESET previous selection**

**ELSE:**

**WAIT 1 second (using after())**

**FLIP both cards back**

**RESET previous selection**

**REDRAW the game board**

**IF all cards are matched:**

**DISPLAY "You win!"**

**END PROGRAM**

**Relevant Implications: (list the implications that are only relevant to your program development. Note: How these has been applied in developing your outcome should be written at the end.)**

|  |  |  |
| --- | --- | --- |
| **Relevant Implication** | **Describe** | **Explain** |
| Functionality | Functionality is when the program works correctly and does what it is supposed to do. | I will ensure that the program functions properly by continuously testing each part of the game during development. This includes checking that cards flip correctly, card matches are detected accurately, and unmatched cards flip back after a short delay. I will also test all difficulty levels to make sure the grid sizes and card layouts display as intended. Through testing and trialling , I will fix errors to make sure the game runs smoothly without crashes or glitches. |
| Usability | |  | | --- | | Usability is about how easy and clear the program is for users to interact with. |  |  | | --- | |  | | The user interacts with the game through a simple graphical interface made with Tkinter. They click on buttons that represent the cards to flip and match pairs. The layout is arranged in a clear grid so players can easily see which cards are face up or face down. I will make sure that the buttons respond quickly when clicked and that visual feedback such as cards revealing their values is clear and easy to understand. This ensures the game is simple and enjoyable for all users to interact with. |
| Accessibility | |  | | --- | |  |  |  | | --- | | Accessibility is about designing the program so that people with different abilities or needs can use it easily. It focuses on making sure that users with visual, motor, or hearing difficulties can still interact with and enjoy the game without barriers. | | The game both use both colours and icons to identify cards, thus helping users who are colour-blind. The layout will use large clickable areas and simple visuals, making it easier for users with vision problems to interact. |
| Data Security Privacy | Data security privacy is about protecting user data and making sure private information is not misused or stored unnecessarily. | The game does not collect any personal data. The game will not require any logins or storage of user information, ensuring complete privacy and safety for all players. |
| Ethical | |  | | --- | |  |  |  | | --- | | Ethical implications is about involving making sure that the program is appropriate, fair, and used in a responsible way. | | The game is designed to support learning and memory development. It contains no violent, harmful, or addictive content, making it suitable for educational or casual use by a wide range of users. |
| End user consideration | End user consideration is about keeping the intended audience in mind during development. | The game is aimed at students and casual users or people struggling with dementia or other memory issues, so the design will be clean, fun, and age-appropriate. It also includes an easy level for beginners and a harder level for those who want more of a challenge. |

**Links to Project Management:**

|  |  |
| --- | --- |
| **Project Management Tool** | **Link** |
| **Microsoft Planner** | <https://planner.cloud.microsoft/webui/plan/zzU_-rwOjE2a-zh7I0I7ScgABgIc/view/board?tid=d92c92d5-676a-4f74-b9fe-02f7ae57db25> |
| **Github** | <https://github.com/LucasYu33/Flash-cards> |
| **Miro** | <https://miro.com/app/board/uXjVJZ7c9fI=/> |

**(Screenshot of Task Decomposition from Trello)**

A screenshot of a computer

AI-generated content may be incorrect.

**Iteration1: Explain what actions will happen in your first iteration.**

**(**This should include the UML diagram with classes and inheritance if any, wireframe of your GUI, code structure, classes used, functions used. Highlight the key sections of the code that are of significance.**)**

This memory game code creates a simple card-matching game using Python’s Tkinter GUI. It first starts a 4x4 grid of buttons, each representing a hidden card with a letter (A to H, each appearing twice). When the player clicks a button, the corresponding card is revealed by changing the button’s text. The game keeps track of the two most recently flipped cards using the flipped and flipped\_values lists. If the two revealed cards match, they stay visible and are added to the matched list if not, they are hidden again after a short delay using after(). The on\_card\_click function handles each click, and check\_match determines whether the two flipped cards are a pair. The cards are shuffled at the beginning to ensure randomness, and the buttons are created and placed in a 4x4 grid using a loop. The game continues until all pairs are matched. Thus, this is my first iteration of my Python program.

**UML Diagram: A screenshot of a computer screen

AI-generated content may be incorrect.**

**Table of objects required in the program (**Add more rows if required.**)**

|  |  |  |
| --- | --- | --- |
| **Objects/Variables/Storage structures such as lists/dictionaries/CSV/JSON** | **Datatype** | **Purpose and its relevance to the outcome development** |
| **window** | **Tkinter object** | **Creates the main game window where all buttons and text are displayed.** |
| **cards** | **list** | **Stores all card values (e.g. ['A', 'A', 'B', 'B', ...]) used for matching; shuffled each time the game starts for randomness.** |
| **buttons** | **list** | **Holds all button objects that represent the cards on the GUI grid.** |
| **flipped** | **list** | |  | | --- | |  |  |  | | --- | | **Keeps track of the buttons that the user has flipped during a turn.** | |
| **flipped\_values** | **list** | **Stores the actual values (letters) of the flipped cards to check if they match.** |
| **matched** | |  | | --- | | **list** |  |  | | --- | |  | | |  | | --- | | **Contains buttons that have already been matched so they stay revealed.** |  |  | | --- | |  | |
| |  | | --- | | **on\_card\_click(index)** |  |  | | --- | |  | | **function** | **Handles what happens when a user clicks a card (shows value, adds to flipped list, checks for matches).** |
| **check\_match()** | **function** | **Compares the two flipped cards and decides whether to keep them shown or flip them back.** |
| **make\_button(index)** | **function** | |  | | --- | |  |  |  | | --- | | **Creates individual card buttons, assigns them commands, and arranges them in a 4x4 grid layout.** | |
| **random.shuffle()** | **Function from random module** | **Randomises the card order to make each game unique.** |
| **window.after()** | **Tkinter method** | **Adds a small delay (e.g. 1000 ms) before flipping unmatched cards back to improve gameplay experience.** |

**Screenshot of Version 1 Flowchart:**

**Iteration 1: Component Testing Table: Include Screenshots and give reason which component is selected and why?**

**Valid Boundary Testing:** (This is an example) Copy this table as you require below. Also create a table in this style with Invalid Boundary Testing.

| **Test Case Information** | Test flipping a card |
| --- | --- |
| **Test Type** | Valid |
| **Component Being Tested** | on\_card\_click() function |
| **Test Data Entered** | User clicks on a card button |
| **Why This Data Was Used** | To check that a card reveals its letter when clicked |
| **Expected Result** | Card shows the correct letter (e.g. “A”) |
| **Actual Result** | Accepted the password |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | No fix needed |
| **Screenshot** |  |

| **Test Case Information** | Test flipping two cards that match |
| --- | --- |
| **Test Type** | Valid |
| **Component Being Tested** | |  | | --- | |  |  |  | | --- | | check\_match() function | |
| **Test Data Entered** | |  | | --- | |  |  |  | | --- | | Click two cards with same letter (e.g. A and A) | |
| **Why This Data Was Used** | To confirm that matched cards stay revealed |
| **Expected Result** | Both cards stay visible after delay |
| **Actual Result** | Both cards remained visible |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | No fix needed |
| **Screenshot** |  |

| **Test Case Information** | Test flipping two cards that don’t match |
| --- | --- |
| **Test Type** | Valid Boundary |
| **Component Being Tested** | check\_match() function + window.after() delay |
| **Test Data Entered** | Click two different cards (e.g. A and B) |
| **Why This Data Was Used** | To verify that unmatched cards flip back after delay |
| **Expected Result** | Cards flip back to blank after ≈1 second |
| **Actual Result** | Works as expected after delay |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | No fix needed |
| **Screenshot** |  |

| Test flipping two cards that don’t match |
| --- |
| Valid Boundary |
| check\_match() function + window.after() delay |
| Click two different cards (e.g. A and B) |
| To verify that unmatched cards flip back after delay |
| Cards flip back to blank after ≈1 second |
| Works as expected after delay |
| Pass |
| No fix needed |
| A screenshot of a white square with black letters  AI-generated content may be incorrect.A white square object with black lines  AI-generated content may be incorrect. |

| **Test Case Information** | **Test clicking matched card again** |
| --- | --- |
| **Test Type** | **Invalid** |
| **Component Being Tested** | **on\_card\_click() condition** |
| **Test Data Entered** | **Click on an already matched card** |
| **Why This Data Was Used** | **To check that matched cards cannot be re-clicked** |
| **Expected Result** | **Nothing should happen when clicked** |
| **Actual Result** | **No change occurred on click** |
| **Pass/Fail** | **Pass** |
| **How the Error Was Fixed** | **No fix needed** |
| **Screenshot** |  |

| **Test Case Information** | **Test clicking same card twice in a row** |
| --- | --- |
| **Test Type** | **Invalid** |
| **Component Being Tested** | **on\_card\_click() logic** |
| **Test Data Entered** | **Click one card twice quickly** |
| **Why This Data Was Used** | **To ensure the same card isn’t counted as a pair** |
| **Expected Result** | **Only one flip should count** |
| **Actual Result** | **The same card was added twice to the flipped list, causing an index error during check\_match()** |
| **Pass/Fail** | **Fail** |
| **How the Error Was Fixed** | |  | | --- | |  |  |  | | --- | | **I fixed the issue by adding a line to ignore clicks if the same button is clicked twice in a row: python if len(flipped) == 1 and flipped[0] == button: return This stops the same card being re-selected before the second click.** | |
| **Screenshot** |  |

**Feedback on Version 1: include a testing video here.**

|  |  |
| --- | --- |
| **Stakeholder** | **Feedback** |
| **SH1  Alan Yang** | Very simplistic and easy to use |
| **SH2  Zane Mansoor** | Needs more colouring and aesthetics |
| **Teacher’s Feedback** | Very basic and needs more aesthetics but since its iteration 1 it should be ok for now |

**Updated Trello Screenshot after Version1**

**A diagram of a diagram

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Summary of Feedback and intended changes to make in Version 2:**

**From the feedback I collected in Version 1, I learned that while the main functionality of my memory game works correctly, the program is currently too basic in terms of visual design and aesthetics. Both of my stakeholders (Alan Yang and Zane Mansoor) mentioned that the game was easy to use but needed more colour and improved aesthetics to make it more appealing. My teacher also said and agreed, noting that while it functions well for iteration 1, the current program lacks any aesthetics.**

**For Version 2, I plan to improve on the graphical interface and visual appeal of the game. Specifically, I will:**

* **Add background colours to the window and buttons to make the game more vibrant and colourful.**
* **Increase the button size and spacing slightly to improve visibility and usability.**
* **Add a title label at the top of the window so users know what the game is.**
* **Include comments in the code for readability and easier maintenance.**
* **Add difficulties for the program**

**These changes will make the game more engaging and user-friendly while keeping the interface simple and accessible. These qualities also aligns with the feedback about improving aesthetics without overcomplicating the gameplay.**

**Iteration2: Explain what actions will happen in your second iteration.**

**(**This should include the GUI Wireframe,code structure, classes used, functions used. Highlight the key sections of the code that are of significance. Include the Task screenshot from Trello**)**

**Actions in Iteration 2**

* **Improved GUI design (To enhance visual appeal and make the game more engaging.)**
* **Difficulty levels added (To increase replayability and let users choose challenge level.)**
* **Dynamic grid layout (Automatically adjusts the game board size based on difficulty.)**
* **Restart functionality (Players can restart the game without restarting the app.)**
* **Move counter (Tracks how many moves the player has made.)**
* **Win message (A clear visual message when all matches are found.)**

**Code Structure**

|  |  |  |
| --- | --- | --- |
| **Section** | |  | | --- | |  |   **Description** |
| Global variables | Hold game state: cards, flips, matches, lock, etc. |
| Difficulty configuration | DIFFICULTIES dictionary maps to grid sizes |
| UI setup | tkinter window, labels, buttons, dropdown |
| Dynamic board creation | start\_game() builds the board based on selected difficulty |
| Card click logic | on\_card\_click() flips cards, stores values |
| Matching logic | check\_match() checks for a match after 2 flips |
| Win condition | show\_win\_message() displays a win message |
| Restart logic | Resets the board without quitting the app |

**Functions used**

|  |  |
| --- | --- |
| **Function name** | **Purpose** |
| start\_game(difficulty) | Sets up a new board based on selected difficulty |
| on\_card\_click(index) | Handles card click, reveals value, and triggers match check |
| check\_match() | Compares flipped cards, updates matched list, resets unmatched |
| show\_win\_message() | Displays a win label when game is completed |
| window.mainloop() | Starts the tkinter GUI event loop |

**A screenshot of a computer

AI-generated content may be incorrect.**

**Screenshot of V2 Flowchart:**

**Iteration 2: Component Testing Table: Include Screenshots: Include Screenshots and give reason which component is selected and why? Add more tables for other cases such as invalid cases.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Information** | |  | | --- | |  |  |  | | --- | | Matching two identical cards | |
| **Test Type** | |  | | --- | |  |  |  | | --- | | Valid | |
| **Component Being Tested** | check\_match() function |
| **Test Data Entered** | Clicked two matching cards (e.g., ‘A’ and ‘A’) |
| **Why This Data Was Used** | To verify correct behavior when cards match |
| **Expected Result** | Both cards remain visible and background changes to green |
| **Actual Result** | Exactly as expected |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | No error present |
| **Screenshot** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case Information** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Flipping two different cards | | |
| **Test Type** | |  | | --- | |  |  |  | | --- | | Valid | |
| **Component Being Tested** | check\_match() function |
| **Test Data Entered** | Clicked two different cards (e.g., ‘A’ and ‘B’) |
| **Why This Data Was Used** | |  | | --- | |  |  |  | | --- | | To confirm cards flip back if they don’t match | |
| **Expected Result** | Both cards flip back after a short delay |
| **Actual Result** | |  | | --- | |  |  |  | | --- | | Cards flipped back after 800ms | |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | |  | | --- | |  |  |  | | --- | | Working correctly | |
| **Screenshot** |  |

|  |  |  |
| --- | --- | --- |
| **Test Case Information** | |  | | --- | |  |   Difficulty levels |
| **Test Type** | |  | | --- | |  |  |  | | --- | | Valid | |
| **Component Being Tested** | start\_game(difficulty) + difficulty menu |
| **Test Data Entered** | |  | | --- | |  |  |  | | --- | | Selected “easy” difficulty | |
| **Why This Data Was Used** | |  | | --- | |  |  |  | | --- | | To verify 4x4 grid is displayed (16 cards) | |
| **Expected Result** | |  | | --- | |  |  |  | | --- | | 4 rows × 4 columns of cards | |
| **Actual Result** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Correct grid appears | | |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | No error | | |
| **Screenshot** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case Information** | |  | | --- | |  | |  |  |  | | --- | | Restart button | |
| **Test Type** | |  | | --- | |  |  |  | | --- | | Valid | |
| **Component Being Tested** | Restart button + start\_game() reset |
| **Test Data Entered** | |  | | --- | |  |  |  | | --- | | Clicked the Restart button after some moves | |
| **Why This Data Was Used** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | To ensure the game resets correctly | | |
| **Expected Result** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | All cards hidden, move counter reset, new shuffle | | |
| **Actual Result** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | All conditions met | | |
| **Pass/Fail** | Pass |
| **How the Error Was Fixed** | |  | | --- | |  |  |  | | --- | | Code resets flipped, matched, move and more | |
| **Screenshot** | A screenshot of a game  AI-generated content may be incorrect. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case Information** | |  | | --- | |  | |  |  |  | | --- | | Prevent Double-clicking same card | |
| **Test Type** | |  | | --- | |  |  |  | | --- | | Inalid | |
| **Component Being Tested** | |  | | --- | |  |  |  | | --- | | on\_card\_click() function | |
| **Test Data Entered** | |  | | --- | |  |  |  | | --- | | Clicked the same card twice rapidly | |
| **Why This Data Was Used** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | To make sure users can't match a card to itself | | |
| **Expected Result** | |  | | --- | |  |  |  | | --- | | Second click should be ignored (no match attempted) | |
| **Actual Result** | |  | | --- | |  |  |  | | --- | | The card was flipped twice and incorrectly matched itself | |
| **Pass/Fail** | Fail |
| **How the Error Was Fixed** | |  | | --- | |  |  |  | | --- | | Added a condition inside on\_card\_click() where line checks if the clicked button is already flipped or matched, and if so, it exits the function to prevent further action on that button. | |
| **Screenshot** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case Information** | |  | | --- | |  | |  |  |  | | --- | | Prevent clicking extra cards while matching | |
| **Test Type** | |  | | --- | |  |  |  | | --- | | Invalid | |
| **Component Being Tested** | |  | | --- | |  |  |  | | --- | | on\_card\_click() function | |
| **Test Data Entered** | |  | | --- | |  |  |  | | --- | | Clicked a third card before the first two finished comparing | |
| **Why This Data Was Used** | |  | | --- | |  |  |  | | --- | | To check if the game correctly blocks interaction during matching delay | |
| **Expected Result** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Third card should not respond during delay | | |
| **Actual Result** | |  | | --- | |  |  |  | | --- | | Third card flipped while two were already flipped which caused a bug | |
| **Pass/Fail** | Fail |
| **How the Error Was Fixed** | |  | | --- | |  |  |  | | --- | | The bug was caused by not blocking input while checking for a match. It was fixed by adding: if lock: return at the beginning of on\_card\_click() to prevent clicks during the check. | |
| **Screenshot** |  |

**Feedback on Version 2: include a testing video here.**

|  |  |
| --- | --- |
| **Stakeholder** | **Feedback** |
| **SH1**  **Alan Yang** | The new colour scheme looks much better and makes the game more fun to play. However, the Hard mode feels a bit too hard, maybe reduce the grid or add a timer to make it more manageable. |
| **SH2  Zane mansoor** | I like how the difficulty selector works and the buttons are more spaced out and I also liked the move counter. One issue is that the win message is easy to miss try making it more noticeable with a sound or popup. |
| **Teacher’s Feedback** | improved the visuals and added useful features like difficulty and restart, a lot better than iteration 1. Try adding a text box where the user can input their names and more. |

**Update the Trello and place a screenshot showing completion of above task.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Summary of Feedback and intended changes to make in Version 3:**

* **Alan Yang said the updated visuals are much better and make the game more enjoyable. However, he found Hard mode a bit too difficult, and suggested making it more manageable, possibly by adding a timer or progress indicator.**
* **Zane Mansoor liked the improved spacing and the working difficulty selector. But he felt that the win message was easy to miss and suggested adding a popup or sound effect to make it more rewarding.**
* **The teacher liked the improvements from Iteration 1, especially the added difficulty levels and restart button. she recommended adding a textbox for players to enter their names, to personalize the game and improve the programs functionality and usability.**

**Intended Changes for Version 3:**

1. **Add a Timer Feature**
   * **Display a count-up timer so players can see how long they’ve taken.**
   * **Possibly use time to give feedback like "Great job!" or "Try faster next time!" after winning.**
2. **Add a Name Input Box**
   * **Let the user enter their name before starting the game.**
   * **Show the name in the move counter or win message to personalize the experience.**
3. **Polish the UI Further**
   * **Ensure all buttons, labels, and game elements fit well across all grid sizes.**
4. **Change Hard mode’s grid size from 10x10 to 8x8 to make it easier for users**
   * **Fix any minor bugs or lag when flipping cards or restarting.**

**Iteration3: Explain what actions will happen in your 3rd iteration.**

**(**This should include the GUI Wireframe, code structure, classes used, functions used. Highlight the key sections of the code that are of significance.**)**

**Actions in Iteration 3**

* **Added player name input (To personalize the game and display player-specific stats.)**
* **Implemented a count-up timer (To track time taken and increase player engagement.)**
* **Dynamic button sizing and precise grid placement (To fit different grid sizes neatly without gaps.)**
* **Improved UI layout with frames and consistent styling (For cleaner, more organized interface.)**
* **Lock mechanism to prevent extra clicks during card check (To avoid game glitches.)**
* **Win message popup showing player name, moves, and time (Clear feedback on game completion.)**
* **Restart button to reset the game quickly (Improves usability and replayability.)**

|  |  |
| --- | --- |
| **Section** | **Description** |
| Global variables | Store game state: buttons, flipped cards, matched cards, moves, lock, timer status, player name |
| Difficulty configuration | DIFFICULTIES defines grid sizes for Easy, Medium, Hard |
| UI setup | Tkinter window, title label, input box, dropdown, timer/moves labels, frames |
| Game board creation | start\_game() creates cards and buttons dynamically based on difficulty and player input |
| Card flip handling | flip(i) reveals card, manages flipped list, and prevents invalid clicks |
| Match checking | check() compares flipped cards, updates matched list, manages UI feedback |
| Timer update | |  | | --- | |  |  |  | | --- | | update\_timer() refreshes elapsed time display every second | |
| Win condition | win() shows popup with player name, moves, and time |
| Restart functionality | |  | | --- | |  |  |  | | --- | | Calls start\_game() to reset the game state and UI | |

**Functions used**

|  |  |
| --- | --- |
| **Function name** | **Purpose** |
| start\_game() | Initializes the game board, resets variables, starts the timer, and builds UI dynamically |
| flip(i) | Handles flipping cards, reveals symbol, controls game input locking |
| win() | Stops the timer and displays a popup congratulating the player with stats |
| check() | |  | | --- | |  |  |  | | --- | | Checks if two flipped cards match, updates matched list and UI, increments move count | |
| update\_timer() | |  | | --- | |  |  |  | | --- | | Continuously updates the on-screen timer while the game is running | |
| window.mainloop() | |  | | --- | |  |  |  | | --- | | Runs the Tkinter event loop to keep the GUI responsive and interactive | |

**Screenshot of V3 Flowchart:**

**Iteration 3: Component Testing Table: Include Screenshots : Include Screenshots and give reason which component is selected and why? Add more tables for other cases such as invalid cases.**

| **Test Case Information** |  |
| --- | --- |
| **Test Type** |  |
| **Component Being Tested** |  |
| **Test Data Entered** |  |
| **Why This Data Was Used** |  |
| **Expected Result** |  |
| **Actual Result** |  |
| **Pass/Fail** |  |
| **How the Error Was Fixed** |  |
| **Screenshot** |  |

**Feedback on Version 3: include a testing video here.**

|  |  |
| --- | --- |
| **Stakeholder** | **Feedback** |
| **SH1** |  |
| **SH2** |  |
| **Teacher’s Feedback** |  |

**Summary of Feedback and intended changes to make in Version 4:**

**A comparative chart of Version GUI showing iterative improvement**

|  |  |
| --- | --- |
| **Version1 Description** | **GUI Design** |
|  |  |
| **Version2 Description and Feedback** | **Gui Design** |
|  |  |
| **Version 3 Description and Feedback** | **GUI Design** |
|  |  |

**Discuss how you addressed the Relevant Implications you described and explained earlier. Please provide screenshots where you applied the implication.**

|  |  |
| --- | --- |
| **Relevant Implication** | **How I applied this in the development of my outcome.** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Final Trello Board showing all the Tasks Done :**

**Fitness for Purpose:**

**(Discusses how the information from planning, testing and trialling of components assisted in the development of a high-quality outcome. Include how this can be further developed and implemented in the future.)**

**Make sure you tick all the boxes here.**

|  |  |
| --- | --- |
| **Requirement** | **Status**  **√** |
| Introduction to the project |  |
| End users |  |
| Project Management Tools such as Trello, Github, Draw.IO used and updated from version to version. |  |
| Relevant implications- Described and Explained |  |
| Software requirements listed |  |
| Program design requirements such as selection, sequence and iteration control structures |  |
| Flowcharts |  |
| Defined Classes and created obje |  |
| GUI wireframes for all versions including annotations |  |
| Coding conventions followed |  |
| Reads from or writes to files or other persistent storage used |  |
| Defined classes and created objects |  |
| Defined and used custom data types |  |
| Used complex data structures such as Queues |  |
| Trello updated frequently. |  |
| Links for Trello, Github and Draw.IO given |  |
| Comments written to describe the code |  |
| Annotated screenshots or screencast videos showing the testing procedures |  |
| Annotated screenshots or screencast videos demonstrating the program is functioning |  |
| Documented all the testing procedures using tables provided for each version. |  |
| Feedback from end users/stakeholders documented correctly. |  |
| Addressed relevant Implications with screenshots |  |
| Summary of fitness for purpose |  |
| Final Update of Trello board with all tasks done. |  |