Coursework Coversheet

| Module: | computer science | Term: | second |
| --- | --- | --- | --- |
| Date: | 07/05/2025 | Weighting: |  |
| Student First Name: | mohamed seif el islam | | |
| Student Last name: | arar | | |
| Course Tutor: | claudia papi | | |
| Class: | 1 | | |

Marks Obtained (out of 100):

Grade:

*Declaration that the work submitted is my own. I:*

* declare that this submission is entirely written in my own words and no part has been

generated by AI software.

* declare that this submission has been written with contributions from AI software.

I acknowledge that-

* used AI for suggestions, generate ideas or understand core concepts as a preparatory activity
* used AI to write, rephrase, or paraphrase part of the essay
* used QuillBot, Grammarly or other software to review language

At the end of the reference section, list AI tools used and explain how much (in percentage) they have contributed. Failure to acknowledge use of generative AI tools (such as ChatGPT and Bard) is considered a violation of EF’s plagiarism standards.

**Final Project Report: Number Guessing Game**

### Problem Definition and Analysis

The project addresses a simple yet engaging problem: creating an interactive number guessing game using Python's Tkinter library for graphical user interfaces. The main goal is to develop a user-friendly application where players attempt to guess a randomly generated number between 1 and 100 within a limited number of attempts. This choice was ideal for the final project because it aligns with the Python skills gained during the course, particularly in GUI development, event-driven programming, and condition handling.

The project targets beginner users or learners who are exploring number logic and pattern recognition. The interface was kept intuitive to ensure accessibility across different age groups and technical abilities. The development was done using Python because of its readability, built-in GUI support through Tkinter, and extensive documentation. The program flow includes random number generation, user input validation, conditional feedback, and game resetting. This logical sequence ensures clarity and allows for easy debugging and enhancement.

Research involved reviewing Python's Tkinter documentation and example applications for layout design and input validation strategies. The problem was decomposed into modules: input handling, feedback generation, and UI updates, all dependent on the game's current state. The clear objective was to deliver a fun, functional, and interactive learning experience using core Python principles.

### Documented Design

#### System Overview

The Number Guessing Game follows a **top-down design** approach, starting with the main GUI structure and breaking it into smaller modules:

* Game initialization
* User input collection
* Guess checking logic
* Feedback output
* Game reset

Below is a simplified **system flowchart** of the main process:

Start → Generate Random Number → Wait for User Input

→ Check if input is valid

→ If correct → Show success message

→ If incorrect → Reduce attempt count

→ If attempts left = 0 → Show failure message

→ Else → Display "Too low"/"Too high" feedback

→ Repeat or Reset Game

#### Main Modules

1. **\_\_init\_\_ and UI Setup:**
   * Initializes the game window using Tkinter.
   * Sets up the labels, entry box, and button widgets.
   * Generates the secret number and sets the attempt count.
2. **check\_guess():**
   * Triggered when the user presses the "Submit Guess" button.
   * Validates user input (must be an integer between 1 and 100).
   * Compares the guess to the secret number.
   * Updates feedback and attempts left accordingly.
   * Displays a message box when the game ends (win or lose).
3. **reset\_game():**
   * Resets the game state to allow replaying.
   * Clears the input field and feedback.
   * Generates a new secret number and resets attempts.

#### Data Items

| **Variable** | **Type** | **Description** |
| --- | --- | --- |
| secret\_number | int | Random number between 1 and 100 |
| attempts\_left | int | Number of guesses remaining |
| entry.get() | str | Input from user (converted to int) |
| feedback\_label | string | Text feedback on each guess |

#### Algorithms

The core logic revolves around a simple condition-based flow:

if guess == secret\_number:

# Correct guess

elif guess < secret\_number:

# Too low

else:

# Too high

Attempts are reduced using:

self.attempts\_left -= 1

Validation checks are in place to handle incorrect inputs using a try-except block:

try:

guess = int(self.entry.get())

except ValueError:

self.feedback\_label.config(text="Invalid input! Please enter a number.")

#### Human–Computer Interface

The user interface is designed with clarity and simplicity in mind:

* **Title label** clearly explains what the user should do.
* **Entry field** is centered for aesthetic and input focus.
* **Feedback label** displays dynamic responses (e.g., “Too high!”).
* **Attempts label** shows how many tries are left.
* **Message boxes** provide end-of-game alerts (win/lose).

#### Tools and Libraries

* **Tkinter**: Built-in Python library used for creating the GUI.
* **Random**: Used to generate the secret number between 1 and 100.
* **Messagebox**: For displaying win/loss messages in pop-ups.

#### Sample GUI Layout

+-------------------------------------------+

| Guess a number between 1 and 100 |

| |

| [ 45 ] |

| |

| [ Submit Guess ] |

| |

| Too low! Try again. |

| Attempts Left: 5 |

+-------------------------------------------+

#### Error Handling

* Handles non-integer inputs gracefully with user-friendly messages.
* Prevents crashing on empty or invalid input.
* Feedback ensures users are guided toward valid interactions.

### Testing and Evaluation

Testing was done manually through a series of representative test cases designed to verify the core functionality of the number guessing game. These include valid guesses, invalid inputs, boundary conditions (1 and 100), win and loss scenarios, and repeated playthroughs.

#### Sample Tests:

| **Test Case** | **Input** | **Expected Result** |
| --- | --- | --- |
| Correct guess | Secret = 50, guess = 50 | Victory message, game resets |
| Too low guess | Secret = 60, guess = 40 | "Too low!" message |
| Too high guess | Secret = 30, guess = 90 | "Too high!" message |
| Out of range | guess = 105 | "Enter number between 1 and 100" |
| Invalid input | guess = "abc" | "Invalid input!" message |
| Exhaust all attempts | 7 wrong guesses | Game Over message, game resets |

These tests ensure that the user interface responds correctly in various situations and provides appropriate feedback.

#### Evaluation

**Project Objectives Recap:**

1. Create an interactive number guessing game using Python and Tkinter.
2. Implement real-time input validation and user feedback.
3. Allow replayability and user-friendly design.

**Achievement:**

* ✅ All objectives were fully met.
* ✅ Game was functional, engaging, and stable.
* ✅ GUI was responsive and error-handling was effective.

**Potential Improvements:**

* Add difficulty levels (e.g., more range or fewer attempts).
* Track scores across games.
* Add sound or animations to enhance experience.

**User Feedback:**

Testers appreciated the clean design and instant feedback. One suggestion was to allow users to set their number range.