Programming Languages Term Project Report
Eiffel: Matt Keefe, Matt Jolie, Paul O'Sullivan

The Snowman common program is a graphical version of the classic game hangman, where the user enters letters through the input/output window and the game responds by updating a graphical snowman drawing based on the correctness of the guesses. The program selects a random word from a dictionary file, tracks correct and incorrect guesses, and displays the partially or fully completed snowman as the game progresses. The program utilizes Eiffel's object-oriented features with classes for managing the game window (MAIN_WINDOW), the drawing (EV_DRAWING_AREA), and the application flow (EV_APPLICATION). It also leverages data structures like a HASH_TABLE for storing the word list and the RANDOM class for generating random indices. While Eiffel's built-in graphical library made it easy to connect the graphical objects to the game loop so that each incorrect guess triggered a new drawing step, the complexity of the graphics was limited to simple polygons. This worked fine for a simple graphical output like a snowman, but it would definitely be difficult to incorporate more complicated graphics with an Eiffel program. Another challenge with implementing the common program with Eiffel were the debugging/error messages, which were often less clear than error messages in other languages and required more research to solve the issues. Overall, Eiffel's features allowed for a clear and reliable implementation of game logic and graphics, though debugging and customization posed some challenges.

Our creative program is a corporate structure simulator that provides a graphical interface for managing employees within a company. The application allows users to add, remove, and update employees, as well as view the organizational hierarchy tree and search for specific individuals. The program features an user interface that leverages Eiffel's object-oriented capabilities to interact with various graphical components like buttons (EV_BUTTON), text labels (EV_LABEL), and drop-down menus (EV_COMBO_BOX). Eiffel's graphical library, Vision2, facilitates seamless integration of the GUI components with our code, enabling a smooth user experience when performing actions like adding an employee or displaying the company tree. The user can input employee details (name and office number), and use dropdown boxes for reporting structures and work modes (remote, hybrid, or in-person), which are then processed and reflected in the company's organizational structure. However, as with our common program, debugging was challenging due to the less detailed error messages. Implementing more stylized/advanced GUI features also presented difficulties due to the limitations of Eiffel's graphical libraries, which are better suited for simpler interfaces. Nonetheless, Eiffel's strong typing and contract-based design ensured that our program is robust and reliable even if it wasn't eye-popping in terms of graphics.

Works Cited

file Flat contracts. (2020, November 20).
https://www.eiffel.org/files/doc/static/24.05/libraries/base/file_flatshort.html
This source provides documentation for the FILE class, detailing its methods for handling sequential files in Eiffel. These methods, such as make_open_read and read_line, were essential in implementing the dictionary file reading functionality in the common program. The documentation offered clear examples and explanations used for file operations within the program.

Void-safety: Background, definition, and tools. (n.d.). Types as “attached” or “detachable.”
https://www.eiffel.org/doc/eiffel/Void-safety-_Background%2C_definition%2C_and_tools#Types_as_%22attached%22_or_%22detachable%22
This source explains the concepts of "attached" and "detachable" types in Eiffel, which are crucial for ensuring void safety. These principles align with Eiffel’s Design By Contract methodology, ensuring that preconditions are met before executing features. In the common program, this documentation was used to properly declare and check that random_word was not void before proceeding with game logic. This ensured that the game operated reliably, preventing runtime errors related to void references. It was also used in the creative program multiple times to ensure that user input and drop-down boxes were not void for the same reasons.

Seiler, M. (2007, May 7). Random numbers. https://www.eiffel.org/article/random_numbers
This article by Martin Seiler provides an overview of the RANDOM class in Eiffel, explaining its use in generating random numbers and the importance of seeding for variability. It also discusses the command-query separation principle, which is a foundational concept in Eiffel. Example code from this resource was used in the common program to implement the generate_random_word feature, ensuring a random word is selected from the dictionary for each game session. This source also introduced the TIME class as a necessary component for generating a random seed, leading to the class being added as a library for the common program.

ev_combo_box Chart. (n.d.).
https://www.eiffel.org/files/doc/static/24.05/libraries/vision2/ev_combo_box_chart.html
This documentation outlines the EV_COMBO_BOX class and its methods for creating and managing drop-down menus in Eiffel. In the creative program, these methods were used to implement features such as job_title_dropdown and work_mode_dropdown in the show_add_employee_interface. These drop-down menus allow users to select job titles and work modes, enhancing the program’s usability by providing pre-defined options.

Ev_button Chart (n.d.).
https://www.eiffel.org/files/doc/static/24.05/libraries/vision2/ev_button_chart.html
This documentation provides details on the EV_BUTTON class and its methods for creating and managing buttons in Eiffel applications. In the Company Management Program, EV_BUTTON was used extensively to create interactive elements like add_employee_button, remove_employee_button, and quit_button. These buttons allow users to navigate through the application without needing command line input, improving interactivity and user experience.