FINAL PROGRAM DESIGN DOCUMENT

SOFTWARE DEVELOPMENT COMP2604

VLADIMIR BREABIN C21345416

RUAN MULLIGAN C21331716

TU821/2 GROUP B

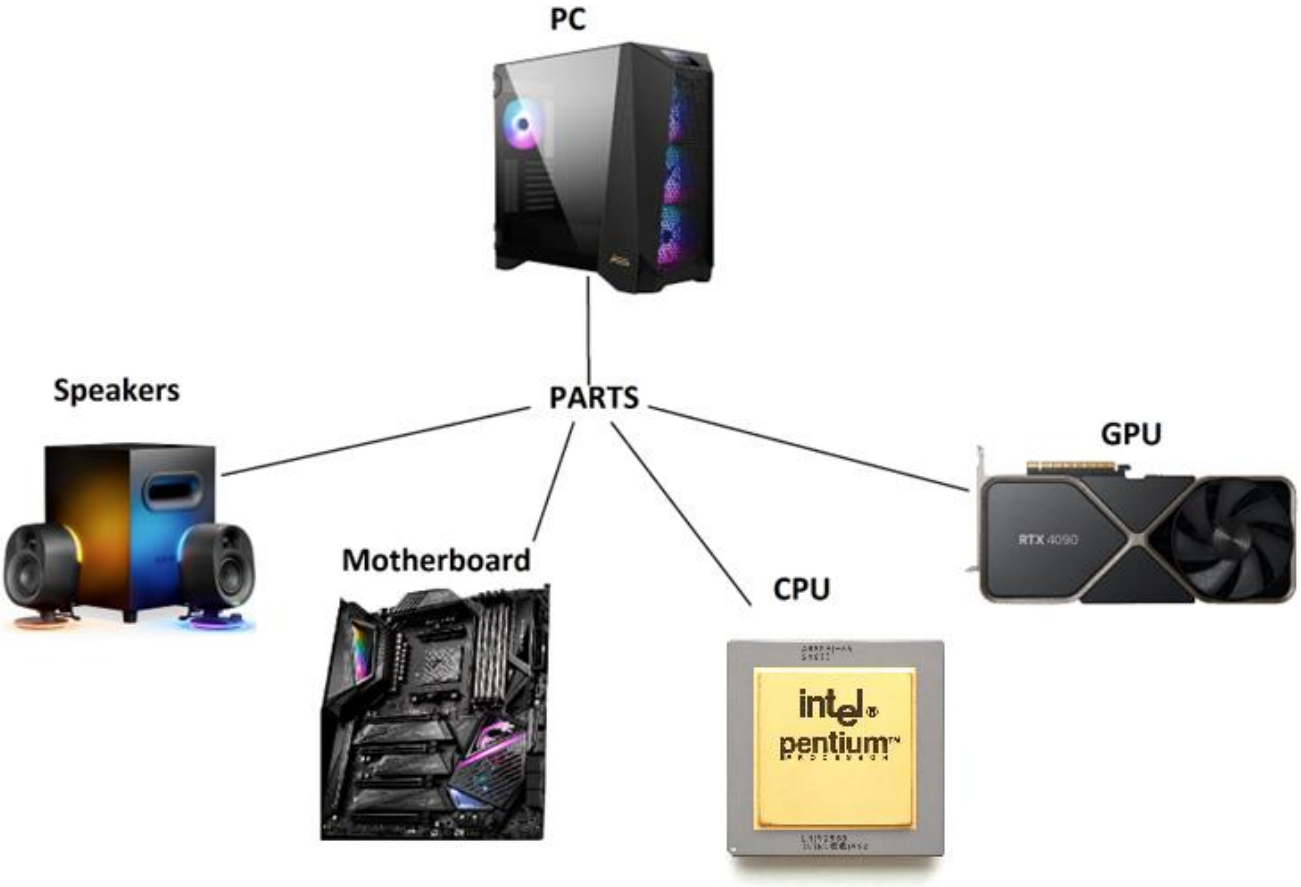
Table of Contents

Introduction:

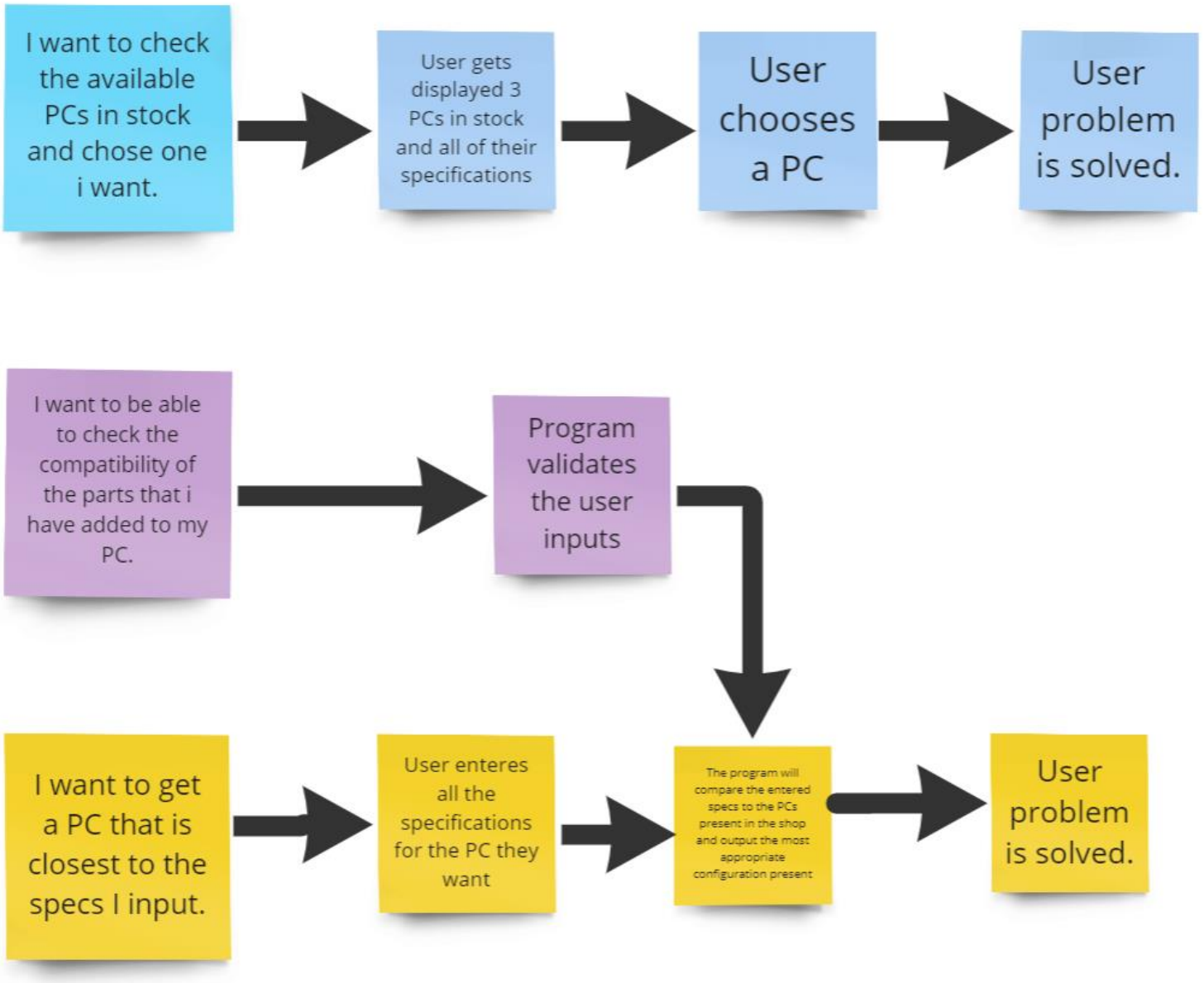
Our application is a system for managing and building PCs. It allows users to create new PCs by selecting and adding different PC parts such as CPUs, GPUs, and motherboards, and to save and load these configurations to and from files. Users can also view the specifications and details of each individual part, as well as the total power rating and compatibility of the entire PC configuration.

To use the application, users can start by selecting a PC part they wish to add to their configuration, such as a CPU or GPU. They can then specify the details of this part, such as its clock speed or number of cores, and add it to their PC configuration. They can repeat this process for all the other parts until they have completed their configuration.

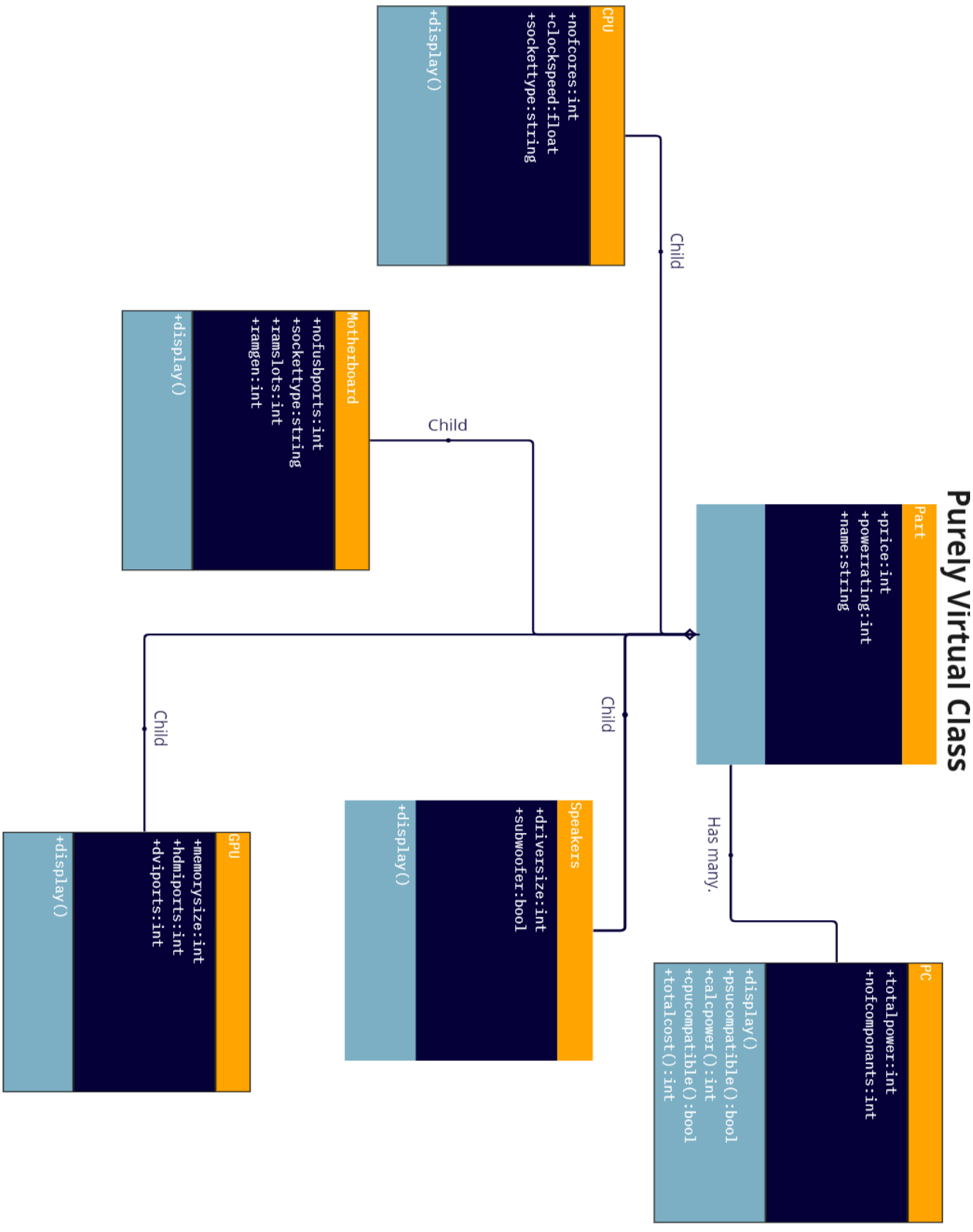
Once the user has finished creating their configuration, they can save it to a file so that it can be easily reloaded later. They can also load previously saved configurations from files, allowing them to easily switch between different PC setups. Our application provides a user-friendly interface for building and managing PC configurations and can be used by both novice and advanced users to create custom PC setups for a variety of purposes, such as gaming or work.



User's Journey Map:



Class Hierarchy:



Functional Description of File Handling:

The save function allows for the saving of a PC object to a file by writing its member variables to an output stream. This function takes an output stream as input and writes the name, price, clock speed, TDP, and number of cores of the CPU, GPU, and RAM objects, as well as the storage capacity of the Storage object to the output stream.

The load function allows for the loading of a PC object from a file by reading its member variables from an input stream. This function takes an input stream as input and reads the name, price, clock speed, TDP, and number of cores of the CPU, GPU, and RAM objects, as well as the storage capacity of the Storage object from the input stream. It then creates a new PC object with these member variables and returns it.

Functional requirement 1:

Requirement description: Save a PC object to a file.

Input: A PC object

System processing: The system will save the PC object to a file with a user-specified filename.

Output: The saved PC object is written to the file, and the system displays a message indicating that the save was successful.

Functional requirement 2:

Requirement description: Load a PC object from a file.

Input: A filename for a saved PC object

System processing: The system will read the file and create a new PC object with the details from the file.

Output: The system displays a message indicating that the load was successful, and the new PC object is returned.

Functional requirement 3:

Requirement description: Handle errors when loading a PC object from a file.

Input: An invalid filename for a saved PC object

System processing: The system will attempt to read the file, but if the file does not exist or is not a valid PC object, it will display an appropriate error message and return a null PC object.

Output: The system displays an appropriate error message and returns a null PC object.

Evaluation:

Test case 1:

To save a PC object to a file when the file does not already exist, it would take the following steps:

1. Create a PC object with details such as a name, price, CPU, GPU, and RAM.
2. Call the SavePC() function with a new file name.
3. Check if the file has been created with the expected details by opening and reading the file.

Expected outcome:

The PC object is saved to a new file with the expected details and the system displays a message indicating that the save was successful.

Test case 2:

The method of loading a PC object from a file involves:

1. Create a PC object with details such as a name, price, CPU, GPU, and RAM.
2. Call the SavePC() function with a new file name to save the object to a file.
3. Create a new, empty PC object.
4. Call the LoadPC() function with the filename of the saved PC object.
5. Check that the loaded PC object has the expected details.

Expected outcome:

The loaded PC object has the expected details, and the system displays a message indicating that the load was successful.

Test case 3:

Error handling when loading a PC object from a file:

1. Attempt to load a PC object from a non-existent file.
2. Check that the system displays an appropriate error message and returns a null PC object.

Expected outcome:

The system displays an appropriate error message and returns a null PC object when attempting to load a PC object from a non-existent file.

Conclusions and future work:

In the end we have managed to construct an efficient C++ application with an easy-to-follow user interface and file handling algorithm. Error prevention functions were especially useful in preventing logic/computation errors of data when creating computer parts for a desirable PC for the user.

The future iteration of this application should –

Use more memory from stack – we have tried implementing it in the PC class, but it became complicated very quickly because of the SavePC() and LoadPC() functions were creating computational errors as they were implemented based on the parts <vector> data storage method instead of new parts method.

Have a bigger PC library size – more PCs constructed would allow for a better test of the sorting algorithms implemented as a part of the previous assignments.

Self-reflection:

Overall, this project has been interesting to work on – especially seeing how it evolved throughout the semester. It has significantly improved our teamwork and problem-solving skills.