Design Document (Cheat engine for games)

Stefan-Nikola

2024

Contents

Chapter 1

Introduction

The idea of my project is a simple cheat engine! I'll be developing a user-friendly tool for single-player gamers to modify their game's memory in real-time. With a simple interface, users can select their game, scan its memory, edit values like health or currency, and save changes.

1.1 Scope

The scope of the project is to have a simple cheat engine that allows gamers to modify memory of their favorite single player games with ease. It should contain only the most crucial elements such as process handling and memory scanning/manipulation.

Chapter 2

Requirements

2.1 Functional requirements

1. The program shall have a user-friendly graphical interface for ease of use.
2. The program shall detect and allow the user to select the game process to modify.
3. The program shall include a feature to scan the memory of the selected game process.
4. The program shall enable users to edit memory values related to in-game attributes.

2.2 Non-functional requirements

1. The cheat engine shall be responsive and have low latency, ensuring smooth interaction with the user interface and quick memory scanning.
2. It shall be optimized to minimize CPU and memory usage, allowing it to run efficiently on a variety of hardware configurations.
3. The cheat engine shall be stable and reliable, with robust error handling to prevent crashes or data corruption.

4. The cheat engine shall be intuitive and easy to use, with a clean and organized user interface that minimizes user confusion.

5. The cheat engine shall be scalable, allowing for future enhancements and updates to be easily integrated into the existing framework.

6. The cheat engine shall be accompanied by comprehensive documentation, including user guides, technical specifications, and release notes.

Chapter 3

Design details

1. Application:

- For UI, I'll utilize a library such as IMGUI and I'll include elements for process selection, memory scanning, memory editing and savin changes.
- For the application, I'll use C++ to handle all the logic.

2. More details

- I'll implement a feature that enumerates running processes and retrieves their IDs and displays their names in UI.
- For memory scanning I'll implement platform specific APIs (either WindowsAPI or LinuxAPI) to read the memory of the selected game and display the memory addresses in the UI.
- I'll implement UI controls for functionality that edits the value of the selected memory address.
- I'll implement a save change UI element to save all the changes I've made to the memory address.
- I'll implement error handling to avoid crashes and catch runtime errors.
- I'll provide documentation that explains all the code and how everything works, together with examples and API references.

Chapter 4

User interface design

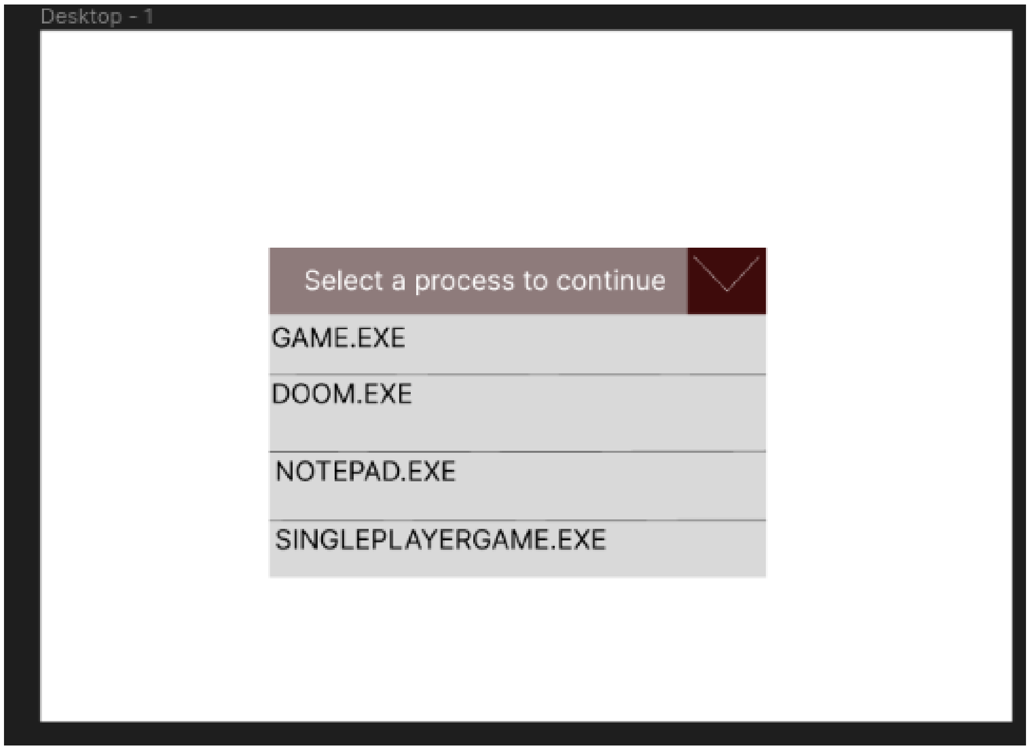

Figure 4.1: Screen after initial startup
After startup, Screen will prompt the user to select the process he want to modify the memory on.

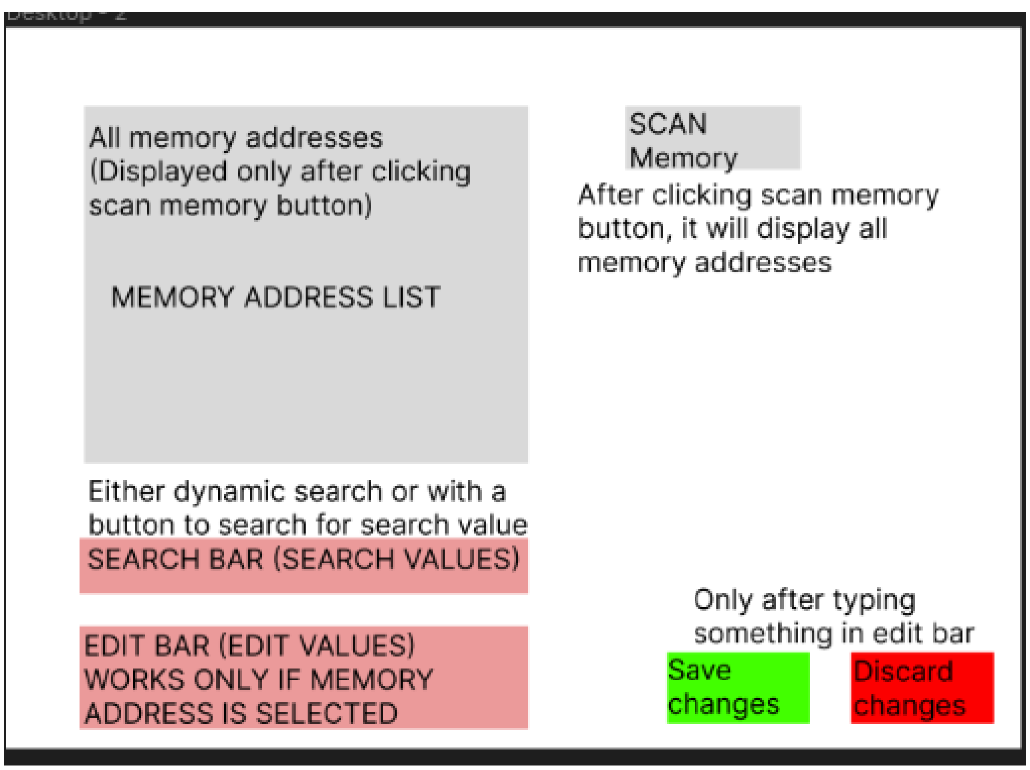
Desktop - 2
All memory addresses (Displayed only after clicking scan memory button)
MEMORY ADDRESS LIST
SCAN Memory
After clicking scan memory button, it will display all memory addresses
Either dynamic search or with a button to search for search value
SEARCH BAR (SEARCH VALUES)
EDIT BAR (EDIT VALUES) WORKS ONLY IF MEMORY ADDRESS IS SELECTED
Only after typing something in edit bar
Save changes
Discard changes

Figure 4.2: Application overview
After selecting the process, user can click on scan memory to display memory addresses of the process. He can search for specific values in the search bar. He can edit the value if he clicks on one of the addresses and can either save or discard the changes.

Chapter 5

Testing

For testing I'll conduct the following testing methods:

1. Unit testing
2. Manual testing
3. User acceptance testing(UAT)