Visualization of Dynamic Memory in C++ Applications

Filip Salén

Computer Game Programming, bachelor's level
2019

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Visualization of Dynamic Memory in C++ Applications

Author: Filip Salén
Supervisor: Patrik Holmlund
filip.salen@gmail.com

July 9, 2019
Department of Computer Science, Electrical and Space Engineering
Abstract

Optimized memory management is important for a demanding application to achieve high-performance results. But un-optimized or unwanted memory behaviors can usually be difficult to detect without an overview of how memory is structured during run-time.

This thesis explores real-time visualization of dynamic memory allocation for applications using the memory pool allocation technique. In the thesis, technical and graphical challenges are described together with their solutions and the design choices that were made.

The final result is a program that can visualize dynamic memory management in real time, with the focus on displaying a detailed and comprehensive memory overview that preserves the level of detail over time.

Sammanfattning

Optimerad minneshantering är viktig för att uppnå hög prestanda i en krävande applikation. Men att upptäcka icke optimerade eller oönskade minnesbeteenden kan vara svårt utan en visuell översikt över hur minnet är strukturerat under programmets exekvering.

Denna avhandling undersöker hur dynamisk minnesallokering kan visualiseras i realtid för applikationer som använder allokeringstekniken minnespool. I avhandlingen beskrivs tekniska och grafiska utmaningar tillsammans med deras lösningar och de designval som gjordes.

Slutresultatet är ett program som kan visualisera dynamisk minneshantering i realtid och som fokuserar på att visa en detaljerad och omfattande minnesöversikt som bevarar detaljeringsgraden över tiden.
# Contents

Abstract ............................................................. i
Sammanfattning .................................................... i
List of Figures ......................................................... iv
Abbreviations and Terms ............................................. v

## 1 Introduction

1.1 Goal and Purpose .............................................. 2
1.2 Gold Town Games ............................................. 2
1.3 Background ..................................................... 2
  1.3.1 Graphical User Interface .................................. 2
  1.3.2 Dynamic Memory .......................................... 3
  1.3.3 Memory Pools ............................................ 3
  1.3.4 Threading ................................................ 4
1.4 Related Work .................................................. 4

## 2 Design and Implementation

2.1 Approach ....................................................... 6
  2.1.1 Graphical User Interface Library ..................... 6
  2.1.2 Inter-Process Communication ......................... 6
  2.1.3 Message construction ................................ 7
2.2 Design .......................................................... 8
  2.2.1 Application .............................................. 8
  2.2.2 Design iteration ....................................... 9
  2.2.3 Color Selection ......................................... 9

## 3 Result

3.1 Menu Bar ...................................................... 12
3.2 Simulation Menu Bar ....................................... 12
3.3 Memory Pool Visualization View ......................... 13
  3.3.1 Memory Pool Detail View ............................... 13
  3.3.2 Memory Pool Selection View ......................... 13
3.4 Activity Log Table .......................................... 14
  3.4.1 Activity Log Icons .................................... 14
3.5 Information View ........................................... 15
3.6 Memory History Chart ..................................... 15
4 Discussion
  4.1 Limitations ....................................................... 17
  4.2 Social, Ethical, and Environmental Considerations ........ 18
  4.3 Future Work ..................................................... 18

5 Conclusion ............................................................ 19

Bibliography ........................................................... 20
List of Figures

2.1 First main view sketch ........................................... 10
2.2 Last main view sketch ........................................... 10
2.3 Chart view sketch ............................................... 11
3.1 Application overview ........................................... 13
3.2 Activity log icons ............................................... 14
3.3 Chart view with allocated memory focus .................... 15
3.4 Chart view with available memory focus .................... 16
Abbreviations and Terms

- **AAA**: Classification for video games produced by major studios, usually with high budgets.
- **C++**: Object-oriented low-level programming language.
- **CPU**: Central Processing Unit, carries out the instructions of a computer program.
- **GUI**: Graphical User Interface, graphical interface that allows the users to interact with electronic devices through input.
- **Heap**: Memory location where dynamically allocated memory resides.
- **Pointers**: In C++, pointers store a memory address referring to another object in memory.
- **Process**: Instance of a computer program that is being executed.
- **RAM**: Random Access Memory, computer data storage that stores currently used data.
- **Thread**: Component of a process which allows for parallel execution.
- **UDP/TCP**: Protocols that determine how traffic will be sent over a network.
Chapter 1

Introduction

Computer software and mobile applications have been increasing in performance over the last decades for instance due to better and faster hardware[3]. This has made it possible to run today’s demanding applications on our devices. To be able to get the most out of an application, it is important to have it well optimized. Especially for time-limited, computational heavy or traffic heavy applications. One important aspect when it comes to performance optimization of modern software is their ability to manage dynamic memory in an efficient way.

Dynamic memory can be used by programmers to have more control over memory lifetime than what would be possible with local or global memory. A block of dynamic memory can be requested from the operating system by the default memory allocator. But this process can be slow because of the large overhead and the cost of performing a context-switch. Instead, a custom memory allocator can be used that only handle application specific cases in an optimized way.

It is important to understand an application’s needs to allocated and managed dynamic memory to be able to improve memory efficiency and run-time speed. Many memory logging and analyzing programs do already exist which are good at finding incorrect behaviors like memory-leaks and dangling pointers. But it can be difficult to get an overall understanding of when, where and how dynamic memory is allocated and deallocated. The ability to visualize dynamic memory can help in addressing these factors and make it easier for the developer to find inefficient and unwanted memory behaviors.

Gold Town Games[5], a medium sized game development company in Skellefteå, develop their two game titles World Football Manager and World Hockey Manager, as well as a match simulating software. They want to visualize how dynamic memory is managed in their software to be able to see if any improvements are necessary or need priority.

This thesis is based on a project done for Gold Town Games with the goal to provide a tool that can visualize dynamic memory in real-time. The project has been done in collaboration with Martin Sperens, a student at Luleå University of Technology, who has created a custom dynamic memory management library in C++ that uses fixed-size memory pools to optimize memory allocation speed. The library communicates information to the visualization tool, where it is stored and visualized to give an overview to the user of how, where and when dynamic memory is allocated and managed.
1.1 Goal and Purpose

The goal of this project was to create a dynamic memory visualization application for Gold Town Games and their game series World Hockey Manager and World Football Manager, but which also could be used in their future projects. The application must be compatible with both Windows and Linux and be able to visualize memory pools and their contents in real-time which includes pool creation and destruction and memory segment allocation and deallocation.

The main purpose of the application was to provide a visual overview representation to the user of computer programs dynamic memory usage during run-time. That, to give the user the ability to see when, where and how dynamic memory is allocated, deallocated and managed to be able to find faulty behaviors, see memory fragmentation, find memory leaks and to later improve memory management of the program.

1.2 Gold Town Games

Gold Town Games[5] is a game developer company located in Skellefteå, Sweden. It was founded in 2015 and has currently 20 employees. Gold Town Games ambitions are to make social, competitive and intuitive sport manager games, foremost on the mobile platform. They currently have two released game titles, World Football Manager and World Hockey Manager. Both games let you play as a manager of a sports team. You can exercise and develop your team by signing new player contracts, build up your arena and face other teams worldwide in ranked tournaments. The sports matches are simulated in their match simulation software that takes all team and player stats into account. Both games are available at Google Play and App Store. Since 2016 the company is listed on the Swedish stock exchange.

1.3 Background

1.3.1 Graphical User Interface

A graphical user interface or GUI allows users to interact with electronic devices through visual indicators and graphical icons. Its roots come from early radar technology after the second world war[1]. The Canadian navy wanted to create a system where data from radars could be shared between many war-ships. A trackball that could control a cursor on a screen was developed. The cursor could select a location on the screen that was conveyed to a processor.

Today a GUI can be found in most electronic devices. Everywhere from in our computers, televisions and mobile phones to microwaves, thermometer and guitar tuners. Graphical User Interfaces are also present within video games but are then usually referenced as a head-up display(HUD).
But creating a good GUI can be a complex task with lots of aspects to take into account. Some decisive ones that need to be considered are layout, color combination, interaction capabilities and ease of use and understanding. It is also a question of personal preferences. Although most GUI devices differ in style and area of use, they all share the general goal to create an interface that is easier to understand and use than a text-based or signal-based counterpart.

1.3.2 Dynamic Memory

Dynamic memory is a type of computer memory together with local and global memory. It allows for more control as it can be acquired and returned in any order without any relation to other memory. Dynamic memory allocators provide dynamic memory blocks and manage memory on the heap. Every time a memory block is needed or not needed anymore a request is sent to the operating system where most perform a context-switch and handles the request. A context-switch is a costly process that allows the operating system to switch from user mode to kernel mode which is needed to handle memory.

In C++, memory is allocated with \texttt{malloc()} or the \texttt{new} operator which enables a memory block on the heap to be used by the software. To deallocate a memory block \texttt{free()} or the \texttt{delete} operator is uses which return the block to its unused state. When many memory allocations and deallocations are made there is a risk for the heap to develop holes of available memory between occupied memory. This phenomenon is called memory fragmentation and can cause wasted available memory because allocations will not fit in all small and separate free blocks.

By default, memory is managed by the operating system’s default allocator. It can be slow since a context-switch might be performed every memory request and it also needs to accept allocations of any size which requires a lot of management overhead. Because of this, it is common for time-restrained applications to use custom dynamic memory managers to increase run-time speed.

1.3.3 Memory Pools

To address the costly process of the default memory allocator, a custom allocator can be used. An example of an allocation technique is to use so-called memory pools. The memory pool allocator is known for having good performance in applications with predefined memory limitations and semi-predictable behavior. An example of such time-limited applications is video games.

The advantage of this technique is that it allocates huge memory blocks, so-called pools, during application start-up. The memory pool can then be filled and emptied during run-time without the need to cause a context-switch. If an application’s maximum memory needs are known prior to
its execution, the memory pool allocator can allocate precisely the needed amount, which in turn decreases overhead size.

Ben Kenwright’s paper[7] describes how the memory pool technique can be implemented without overhead nor loops which make for fast memory allocations and deallocations.

### 1.3.4 Threading

Threading[2] allows a process to be executed more or less in parallel depending on the system. A single-core processor generally implements multithreading using time slicing. This allows the CPU to switch between software threads rapidly to make the appearance of parallelism. A multi-core processor allows for multiple threads to be executed in parallel.

The ability to use threads can allow for better and more efficient applications. A program with many resembling tasks can be optimized by executing them all in parallel. Furthermore, threads can be used together with blocking function calls that is encountered in socket or pipe programming. If the main thread would happen to get blocked, it will cause the application to freeze until the block is released. If a separate thread would handle the blocking function call instead, the main thread can continue to run which allows for a still usable application.

### 1.4 Related Work

For a long time, there has been an interest in collecting information about how software performs their work in order to be able to improve performance and behaviors[12]. Information has previously mostly been logged and displayed in its raw text format. This might be due to, among other things, limited computational and graphical capabilities[3].

Existing programs that log information about dynamic memory allocations and behaviors include Valgrind[16] and Microsoft’s DebugDiag[10]. Valgrind and DebugDiag among other memory logging software are great at finding faulty memory behaviors in a software, such as memory leaks or dangling pointers. But a log usually contains a lot of text and can be hard to read and understand. Future more, it can be especially difficult to find correct but unwanted memory behaviors and to get an overall understanding of where dynamic memory is allocated and deallocated, when reading a log file.

Valgrind has an extension called Massif[15] that can visualize how much memory different resources have allocated. It can also show how much memory the application has allocated and how much memory that is wasted in order to keep the heap 8 or 16 bytes aligned. This helps the developer to discover memory demanding resources and ineffective allocation patterns so that the application can be improved.

For applications that are memory demanding and time-restrained, for example AAA games, a custom memory manager are usually used to acquire
more memory control and to be able to optimize memory usage. As most applications behave differently, custom memory managers are adjusted to handle just certain actual cases as optimized as possible. Tools visualizing dynamic memory must also be customized to specific memory managers to be able to visualize memory in a desirable way and to show application relevant information. Since each solution is more or less unique and because companies have their own tools, there are not many memory visualization applications available that visualize dynamic memory.

A few programs have been made that visualizes how dynamic memory is allocated and managed during application run-time. GCspy\[13\] render color-coded tiles representing 128Kb memory blocks to visualize how memory is allocated in snapshots. To get a historical overview of the memory activity, a historical graph is utilized. Sergio Moreta’s and Alexandru Telea’s paper\[11\] is a good example of how dynamic memory can be visualized over time. It displays memory allocations as color-coded rectangles in a chart scaled by time and memory addresses. Features like a memory occupancy bar and a cushion cursors to check correlations between different charts are also present.
Chapter 2

Design and Implementation

2.1 Approach

It is always important to start a project with proper research, especially for shorter projects when there is no time to restart to achieve sought after results. Even though this project is about visualizing dynamic memory and memory pools, it is important to understand the basics of these concepts to be able to create a visual representation in the most fitting way.

The technical challenges of the project were to, first of all, find a GUI library to use for the creation of the application. Then a way of communication between the memory library and the visualization application needed to be established. Also what kind of information that would need to be sent and how to structure the message.

2.1.1 Graphical User Interface Library

The requirements for the GUI library were that it must support cross-platform development and the creating of graphics and charts.

ImGui[4] and Nuklear[9] are two GUI libraries of the immediate mode type. This means that their event processing is directly controlled by the user. They are more direct and less complex to use but also have the drawbacks of the need to recalculate and redraw every frame. Therefore they are great for simpler applications or as a debug GUI for games or graphics applications. But not well suited for bigger or more complex interfaces.

Qt[14] and wxWidgets[17] are types of GUI libraries that are suitable for both large and small applications and have support for both graphics and charts. The decision to use Qt as a GUI library for this project was made due to their detailed documentation and wide range of example interfaces. Furthermore, Qt has its own platform-independent implementation of threads, a signal/receiver system for easy communication between threads and the support for dock widgets.

2.1.2 Inter-Process Communication

The applications must access data from the memory library in order to be able to visualize dynamic memory. It would be a great advantage if a similar communication method could be implemented for both the Windows
and Linux versions. This, so that the rest of the application could remain unaffected and therefore be implemented independently. Mujtaba Khambatti goes through different types of communication methods in his paper[8] and discuss the subject further.

Socket communication is a variant that can be used. It has the advantage of allowing the application to run on a separate system and sending information over a network. A TCP or UDP protocol can be used to send messages depending on how important it is for all messages to reach the recipient. But since network communication was not a requirement, the method was not chosen because sockets have more packeting overhead than other methods to be able to send data over networks. For network communication, one must also take into account that messages can be read by others. If the message contains secret or sensitive data it can be encrypted to make the access more difficult, but there is always a risk that the encryption can get hacked.

Shared memory is another way of process communication. It works by having a block of memory in the RAM that can be accessed by multiple processes. This type of communication is usually faster compared to sockets or below mentioned named pipes, mostly due to the avoidance of copying data to and from message buffers. But to know where memory is free to be written to and when the memory can be read requires some kind of management which can be a relatively complicated task to design and create.

Named pipes are another example of a communication method that can be implemented on both Windows and Linux, but on the Linux platform it is mostly referred to as a FIFO(first-in, first-out queue). A named pipe appears as a file that other processes can connect to for reading and/or writing access. It has a lot of similarities to sockets but requires less overhead because the messages are not sent over a network. Both named pipes and sockets use some blocking functions which require the data communication to be handled on a separate thread to prevent the application from freezing. This way of communication was chosen for this project due to it being easy to implement and work with, and that named pipes have less overhead and no need for encryption in relation to sockets.

2.1.3 Message construction

It was decided that it would be better for the messages to carry much information about a memory event instead of risking that the user would have to use another program to get a certain event parameter. All message types that were necessary to convey different event from the memory library was produced and resulted in:

- Memory pool creation
- Memory pool destruction
- Memory allocation in pool
- Memory deallocation in pool
- Huge memory allocation outside pool
• Huge memory deallocation outside pool

In addition to memory allocation and deallocation within memory pools, a message was needed for huge allocations and deallocations outside of memory pools. The reason behind the message is when a memory allocation exceeds the maximum sized of memory pools, then the allocation is simply done without a containing pool.

The messages are structured to contain information such as a prefix that describes the message type, starting memory address, size, event time, pool index, a trace-back call stack with function name, line number and file path, and more. But since a trace-back call stack can become very long, mostly depending on the length of the file path, a limit was set to allow for a maximum message size of 2048 bytes, to avoid slow message handling. Since the communication does not take place over the network no encryption is used. Which also saves time and allows for the messages to be logged and legible.

2.2 Design

It can be difficult to design a GUI for more complicated programs, where there is a lot of information, inputs or dependencies. In this application, the main goal is to visualize memory pools and their contents. The user should be able to see when and where memory pools are created and destroyed and when and where memory is allocated and deallocated inside a memory pool. The main visualizing challenges were to:

• Create a GUI that is easy to understand and use even though much information might be present on the screen at the same time.
• Create a good overview so that there is minimal need to use the scroll wheel or the mouse to navigate around. If more detailed information is required it should be just one or a few actions away.
• Create a visualization that does not lose precision or detail over time or at least loses as little as possible.

2.2.1 Application

Visualizing change over time means another dimension to take into account, more than only memory location. It gives a good historical view of how memory has been allocated and can give the user a better understanding of the reason behind certain behaviors. But since time is a constantly growing factor, it can result in stretched visualization views and graphs. This leads to the risk of losing accuracy and details from an overview perspective. Therefore the difficult choice was made to not use a time dimension in the visualization view. Instead, the visualization would focus on the present and have an overall detailed overview.

Three views were chosen as the main representation of the application overview. A memory pool visualization view that also acts as the main view.
Chapter 2. Design and Implementation

It is divided into a detailed view the can display one memory pool and its contents and an overview that shows small representations of memory pools the can be selected to display in the detailed view. An activity log view that logs all events in a table with an event icon and minor information. This view gives the user access to all events in occurrence order so that no events will be missed, and offers simple historical capability. The last view is an information view that can display further information about selected objects.

To give the user more options, a chart displaying available memory in correlation to occupied memory over time was implemented. The chart improves historical analyzing capabilities over time but in trade of losing details over time.

After an analysis of the application and its design, some features felt deficient or difficult and required improvement. First of all, much information was present on the screen and it could be hard to understand how an event in the log list relates to its graphical representation in the visualization view. Therefore, a linkage was established between the graphical view and the log list to be able to highlight both representations when one of them gets selected. Secondly, if many events occurred in a close time frame it could be hard to take in the information. An event queue system was created to address this where the user had the ability to choose to simulate in real-time or to manually step through each event.

2.2.2 Design iteration

In the early stage of the project, a graphical sketch was designed to act as a reference and assist during the development, as shown in figure 2.1. The sketch was updated throughout the project as a more detailed design was established, see figure 2.2 and 2.3.

2.2.3 Color Selection

Colors and color combinations are important to have in mind during the design process of an application. Not only because color can make it easier to understand a GUI and can give the user a better experience. But also to ensure that as many people as possible can use the application under the same conditions. Therefore color combinations were chosen that would not affect people with color blindness.

According to the National Health Institute of USA[6], 8% percent of men and 0.5% percent of women with Northern European ancestry are red-green colorblind, which is also the most common type of color blindness. On a much smaller scale but not to be overlooked is yellow-blue color blindness. Altogether, it constitutes to a relatively large proportion of the population that should be taken into consideration to enable a wider user base.

A green-blue color combination was selected to be applied throughout the application, where blue represents occupied memory or memory allocations and where green represents free memory or memory deallocations.
FIGURE 2.1: First sketch of the main view.

FIGURE 2.2: Updated sketch of the main view later into the project.
FIGURE 2.3: A sketch of how a chart representation could look like, displaying occupied and available memory.

The colors black and gray were also used in the application to symbolize active and inactive events or objects.
Chapter 3

Result

The final result is an application that can visualize computer program’s dynamic memory in real time. This application is designed to visualize memory pools and their contents with the focus to provide an informative overview that does preserve the level of detail over time.

When a new simulation starts, a named pipe connection is established between the application and the memory library. The application manages and displays information in different views from incoming messages.

An image 3.1 was captured of the application’s main overview, which shows how a running simulation can look like. Throughout the application, the color blue represents allocation or occupied memory and green represents deallocation or available memory. The different parts of the application are explained below.

3.1 Menu Bar

The menu bar has three main tabs: file, view, and help. Under the file tab, a new simulation can be started and closed, the application can also be shut down from there. The view and help tab contains toggles for the activity log, information view, and a help window.

3.2 Simulation Menu Bar

The simulation menu bar allows the user to change simulation mode, either to simulate real-time or to manually step through the events. The real-time simulation mode process incoming events directly while manual step mode add incoming events to a queue for manual processing. In the manual step mode, the event queue size is displayed and a step forward button to process the next event. The simulation mode can be changed at any time. If the mode is changed from manual step mode to simulate real-time mode, the application will process all queued events to catch up to the current time.
Chapter 3. Result

3.3 Memory Pool Visualization View

The memory pool visualization view consists of two sections, a detailed memory pool view and a memory pool selection view. Both views display memory pool information in real-time as new events are processed.

3.3.1 Memory Pool Detail View

The memory pool detail view displays one memory pool in more detail where green represents available memory and blue allocated memory. Allocated memory can be selected and is displayed in a light blue color, more information about the selected allocation is then displayed in the information view.

3.3.2 Memory Pool Selection View

The memory pool selection view displays a table of all allocated memory pools. Each pool is represented by an index, pool size and a small pool overview. If a pool is selected it will be displayed in the detail view and more information about the pool will be displayed in the information view. If the index text is black the pool is still active. But if the index text is gray the pool and all contents have been deallocated, but is still displayed how it looked like right before the deallocation.
3.4 Activity Log Table

The activity log table logs all memory events. Each row represents one event with some basic information. The activity log displays: event type, size in bytes, pool index, start address and event time. An event type is represented by an icon and description: Allocation, Deallocation, Huge allocation, Huge deallocation, Pool created or Pool deleted. The icons will update if a later event changes the state of a previous one. E.g. when an allocation also has been deallocated.

When memory is deallocated or a pool destroyed the corresponding event will appear with gray text, also the allocation or pool creation event will be changed to display with gray text. When an event is selected from the activity log more detailed information can be viewed about the event in the information view.

3.4.1 Activity Log Icons

1. Creation of a memory pool.
2. Deallocation of a memory pool and all its containing memory allocations.
3. Memory allocation in a pool.
4. Memory deallocation in a pool.
5. Memory allocation outside of a pool because of a huge allocation size.
6. Memory deallocation of a huge memory size outside of a memory pool.
7. The creation of a memory pool that also has been destroyed.
8. A memory allocation that has been deallocated.
9. A memory allocation that has been deallocated because of the destruction of its memory pool.
10. The allocation of memory with a huge size that also has been deallocated.

![All the icons that can represent an event in the activity log.](image)
3.5 Information View

The information view(7) simply displays information about a selection in the activity log or in the memory pool visualization view. The information that is displayed are: memory address, allocation size, pool index or containing pool index, allocation or deallocation time, lifetime and a back traceable allocation call log and deallocation call log. An element in the call log is made up from a function name, file path and the line number in the file. Memory pools block size is displayed which represents the spacing between the start address of new allocation locations in a memory pool. A deallocation type is also displayed for deallocated memory which shows if the memory has been deallocated from a deallocation or from the destruction of its pool.

3.6 Memory History Chart

The memory history chart displays the amount of allocated and the available memory in memory pools over time. The focus can be changed by clicking the change focus button to scale the y-axis(size) to either focus on allocated memory or available memory. See figure 3.3 and 3.4.

![Memory History Chart](image)

**Figure 3.3:** The chart view set to focus on allocated memory.
FIGURE 3.4: The chart view set to focus on available memory.
Chapter 4

Discussion

Designing a good GUI is a difficult task and even though the main goal for the project was reached and additional features added, there are still many features that can be improved and added. Furthermore, the goal could have been achieved in many different ways, and it is hard to know if one would be better than another as they all have different advantages and flaws.

The biggest design decision was to choose if time would be considered or not in the visualization view. For this project, time was not used to not lose details in the overview. But if time would have been taken into account, it would have resulted in a completely different application. In that case, it would give a better historical memory representation but also require good zoom functionality to make small details apparent during an extended simulation.

A GUI application will never really be completely finished, improvements or functionality can always be added or the appearance can be improved. This shows in today’s popular applications that are constantly updating and changing.

4.1 Limitations

The biggest limitation during the project was the time available. With 8 weeks from the start to the end of the project, the research period was kept short in order to have time to implement the visualization application. The focus was to first to be able to reach the main goal which was to visualize dynamic memory. Other features were later added in a prioritized order as the time allowed for.

It was known already from the start that the application would not be able to get tested against Gold Town Games software during the project time span. Instead, it would be tested and used at a later stage when Gold Town Games saw it fit. This conceded that the application was only tested against a setup test program.
4.2 Social, Ethical, and Environmental Considerations

During this project, the choice was made that communication with the memory library would only take place locally. This also minimizes the risk that the messages can be read by unauthorized persons. Therefore, no encryption of the messages is needed and there is no risk of the messages being hacked.

It was considered important that as many people as possible would be able to use the application and its features. Due to a relatively large proportion of the population being color-blind and find it difficult to use applications with unsuitable colors, the design was made to counter this. Colors like blue and green were chosen that would not have any negative effects for general color blindness.

Since this application only is a visualization application that visualizes dynamic memory, there is no risk of ruining or affect the memory of the program whose memory is being visualized.

4.3 Future Work

In the early stages of the project a list of design features was developed and sorted after priority. Due to the time constraint, far from all were implemented. There is still room for improvements and new features for the application. For example, the ability to save a simulation session and load an old one, or to give the user the ability to step backward in a simulation in manual mode. To easier keep track of memory of interest, a favorite system could be used where favored events appear in a separate list. A database could be used to store data to allow for easy sorting of tables and lists but with the risk of slower performance. To improve the outline and visual effect of memory segment in the visualization view, a block shadow can be used as Sergio Moreta’s and Alexandru Telea’s describe in their paper[11]. Each memory segment could also change color shade depending on lifetime to show newly allocated areas.
Chapter 5

Conclusion

In conclusion, this project did reach its goal to create a cross-platform application that could visualize dynamic memory and memory pools in real-time. It is difficult to determine if this was the best approach and solution when it never could be tested against Gold Town Games’ software during the project period.

There are many ways to visualize dynamic memory and memory pools. In this project, the decision was made to focus on displaying a detailed memory overview to the user that does not lose detail over time.

In my own opinion, I think the resulting application for this project was successful. I also think that all technical and design decisions throughout the project were good and that I would have made the same choices in a post perspective. But the application could of course benefit from a lot of more features that there was no time to implement during this project.
Bibliography


