INF-1100: Introduction to programming and computer behavior

Assignment Template

Sera Madeleine Elstad

November 12th, 2021

1 Introduction

The fifth assignment in INF-1100 is to create a variation of the classic bouncing ball animation. Each ball in the animation, which should be represented as a separate data structure, should start at a random speed in the x direction in the top left corner of the screen. When the ball collides with the edge of the screen, it should bounce back and lose some speed in the process. After losing all speed, the ball should come to rest at the bottom of the screen for five seconds before being removed. To begin, some pre-code is provided. The pre-code contains three files that must be changed:

1. main.c
2. list.c
3. object.c

This rapport will primarily comment on the functions mentioned above, as well as the changes made to enable the bouncing ball animation.

2 Technical Background

2.1 SDL and Make-File

The makefile is a set of commands with variable names and targets that are used to create and remove object files with the commands make and make clean. The makefile is useful because it simplifies the compilation process by automatically determining which files need to be updated (1). SDL (Simple Direct Media Layer) is a cross-platform development library that provides low-level access to audio, keyboard, and graphics hardware, among other things (2).

2.2 Dynamic Memory allocation

Dynamic memory allocation is the proses of assigning the memory space during the execution time. It is used when the programmer does not know how much memory will be

required for the program ahead of time. The library functions are “malloc”, “calloc”, “realloc” and “free” are used to dynamically allocate memory. “Malloc” stands for memory allocation and the function reserve a block of memory (in heap) of the specified number of bytes. When executed, the program can request memory blocks of a particular size from the memory allocator, reserve this block by marking it as allocated, and return to this block using a pointer. The programmer can test whether the expression results in a NULL pointer to see if the memory can be allocated. Calloc stands for contiguous allocation. The malloc function allocates memory and leaves it uninitialized, whereas the calloc function allocates memory and initializes all bites to zero. The programmer can change the size of previously allocated memory with realloc. If a block is allocated with either malloc or calloc, it will not be freed until the programmer decide to explicitly deallocate it by calling the memory deallocater, free (3, pp. 614-618).

2.3 Pointers

A pointer is a variable that contains the address of other memory objects. The "*" character is used between the type definition and the variable name to declare a pointer. Pointers, like other variables, must be declared before any variable address can be stored. Variables in arrays and structures can both be accessed using pointers (3, pp. 563).

2.4 Linked lists and arrays

Linked lists and arrays are two different ways of organizing data in memory. The concept of linked lists combines the concepts of structures and dynamic memory allocation to create a data structure. Linked lists are similar to arrays in that they contain data that is best organized in a list format. Except for the last element, each element in an array has a next element that follows in memory, and it is likewise in a linked list. The biggest difference between a linked list and an array is that in a linked list the elements do not have to be adjacent in memory. Instead, each element in a linked list contains a pointer to the next element, allowing the next element to be placed anywhere in the relation. The pointer is used to reconstruct the sequential order. Linked lists are useful because their structure is dynamic and can be expanded or shrunk during execution. Because of their dynamic quality, they are appealing for use in situations where arrays would be inefficient. (3, pp. 618-629)

2.5 Structs

Structures is a user-defined data type that allows a programmer to group together variables of various data types under a single name. When a structure is declared with the keyword "struct," no memory is allocated. Some variables must be created in order to allocate memory of a specific struct type (3, pp. 628).

2.6 Fps locks – limiting the frames per second

To make sure that the ball has a smooth movement a fps lock, which limits the framed per second, can be implemented. If the program does not have a lock like fps, it will run as fast as the computer can compile it. This can lead to a less smooth movement. While using a fps lock the programmer can decide who many frames the program should have and thereby slow down or speed up the movement. This is done by defining who many ticks the programmer want, and time who much time the program needs to compile and run the function (4).

3 Design and Implementation

3.1 Implementation of the object interface

The object programming interface consist of a function that creates the object, a function that destroys the object, and a function that draws the object on the screen. All the functions accept a pointer to an object data structure as an input parameter. Create object also accept a pointer to the SDL screen and a variable telling the size of the model as an input parameter.

To be able to create an object a new object is made. This new object also needs some place to store all its information, and this is possible when there are allocated some memory for both the object and the model. Inside the object.h file there are a structure for the object, telling which variables the object needs to have. The variables are declared by using a pointer, which points from the object to the variable of interest, and a value are given. At the end the function returns a pointer to a new object data structure.

The destroy object function should free all memory allocated to represent the object. This is done by using free, which is a dynamic memory allocation function.

Draw object is the last function in this file. The draw object function must draw the object on the screen by calling DrawTriangle on each of the model triangles. Before this is done the scale, translation, etc., must be updated in each data structure. This is done by making a for loop that runs through all the tx, ty, rotation, etc. values for the given model and updates them. When that are done, the DrawTriangle function are called.

3.2 The list implementation

The list implementation consists of a few structs, ten functions and a call to the list.h file. This is what the functions do:

1. Returns a newly created, empty list
2. Frees the list and nodes, but not the items it holds
3. Adds an item first in the provided list
4. Adds an item last in the provided list
5. Removes an item from the provided list, only freeing the node
6. Return the number of items in the list
7. Return a newly created list iterator for the given list
8. Free the memory for the given list iterator
9. Move iterator to next item in list and return current
10. Let iterator point to first item in list again

The first function creates a new empty list and returns it. To begin, some memory must be allocated on the heap using malloc. An if-statement checks whether the expression results in a NULL pointer to see if memory can be allocated. The new list is set to be the head, and thus the first element in the list if memory allocation is possible. Because the list is supposed to be empty, the head is set to NULL (this is a representation of an empty list). The number of items in the new list is updated before it is returned; in this case, the number of items is zero.

Deallocating is used in the second function to free the list and its nodes, but not the items it holds. To begin, a while-loop is created, and the node is moved to the top of the list, the head. The while-loop will continue to run as long as there are nodes to process, so the node cannot be NULL. The node is updated to be the next node and then freed within the loop. The list is freed at the end of the function.

Some memory must be allocated before an item can be added in the list. Adding the item first in the provided list is accomplished by creating a node that contains the item, and then inserting the node into the first element of the list (head). It is also possible to add an item at the end of a provided list. This is accomplished by creating a node that contains the item and setting the next node to NULL. The last node in the list will be found as a result of this. The current node is then created and designated as the head. This node will traverse all the nodes in the linked list, beginning at the head. If the node is equal to NULL, it is the only node in the program, and the item is added to it. If not, the program checks to see if the next node is NULL, and if so, the item is added to that node. If none of the preceding conditions are met, a while-loop will run. The loop will run as long as the node do not equal NULL. This loop will continue until the last node is found, at which point the item will be added to the end of the list.

Removing an item from the provided list, only freeing the node, are done by using deallocation. First a node that start at the head are created. The program then checks to see if the node is empty; if it is, there is nothing to free. If the node holds the item, the head is set to be the next node, the current node is freed, and the number of items is updated. If none of the preceding conditions are met, a for-loop traversing all nodes to find the item is executed. When the item is found, an if-statement is executed inside the loop. When this occurs, the node is set to be the next node, the current node is freed, and the number of items is updated. To see who many items the list holds, the list size function can be used. The function returns the number of items is allocated somewhere on the heap.

List iterator returns a newly created list iterator for a given list. This is accomplished by allocating memory for the list iterator as well as the list itself. Then the iterator is set to run through every list, from the head. The function returns an iterator. The memory of a given list is freed by destroying the iterator. This is accomplished by constructing a while-loop that iterates through all the list elements from beginning to end, freeing memory allocated by the iterator.

The function list next is used to move the iterator to the next item in the list and return the current list. The current item is set to NULL, and an if-statement states that as long as the next list does not equal NULL, the current item is updated to be the next item in the list, and then the current item is returned. The final function allows the iterator to return to the first item. This is accomplished by setting next to the head list.

3.3 The algorithm for bouncing balls of screen edges

To be able to see the ball a window must be drawn, and this is done by using SDL functions. First the SDL_Surface, SDL_GetWindowSurface and the SDL_Event are declared in the beginning of the function.

After creating the window, the object is defined, and a new list are created to call for the function list_create. Then the number off objects in the list are define, and the balls are added to the list by a while-loop that creates the objects, add them first in the list, and accelerate them until the program have created as many balls as it is asked for. Before creating a while-loop an iterator is defined.

Inside the while-loop the screen is clear, this is done make sure that the image of the ball from the last time the loop ran are left on the screen. Then a for-loop is made. The for-loop uses an iterator to run through the linked list, and it will run as long as there are nodes in the list. Then a new while-loop are made to create some new balls. If this is not done, there will not be made any new balls after the first balls are removed from the screen. The while-loop making these balls are almost identical to the lats, the only difference is that this loop adds the new object to be the last node in the list.

To make the balls lose some speed when they are bouncing of the edges of the screen before they are coming to a rest at the bottom of the screen two new factors, speed and gravity, are declared. The speed has a negative value less than one because the ball should turn around and loose some speed when hitting the edge and by making a new factor this can be done easy. Then the ball position and the speed value are updated.

The boundary coordinates are zero, height and wight, and they are used to detect when the balls hit the boundary. To do this two if-statements are made, one for the x boundary and one for the y boundary. When the ball hit the boundary, and then enter the code statement, the coordinate is updated for either x or y. The speed on the other hand is when hitting the x boundary only updated for x and for both when hitting the y boundary. Updating both is who the ball finally comes to rest.

To end the while-statement SDL are used to update the window surface. This while-loop are from the last mandatory task, task three.

3.4 The algorithm for when the balls come to rest

The algorithm for when the balls some to rest consist of if-statements, that are placed inside the if-statement for speedy. If the speed in both the x- and the y-direction is close to zero a timer is started (SDL_GetTicks). Because the speed of the ball at rest can be both positive and negative, the absolute value of the speed is used. After the timer is started an if-statement saying that the time until the ball should be removed (ttl) are equal to timer plus 5000. When doing this the program starts counting to 5 seconds. While doing this another if-statement says that when ttl is less than or equal to the timer the list and object should be removed and then the object should be destroyed. Following the destruction of the object, the number of items is updated.

3.5 Algorithm for a smooth object movement

To be able to get a smooth movement a spf lock was implemented. This is done telling the program who many frames it should have. Then SDL_GetTicks are set in the begging and the end of the code part the programmer is interested in. When the time it takes to compile and run the program are known, the program calculates the time difference. After this, an if-statement are used to delay the program so the programmer gets the number of frames that is asked for. Without this lock the program would run as fast as the compiler could run it, and this would be different from computer to computer, and it could also make some parts of the movement faster than others.

4 Evaluation and discussion

The list implementation was tested by using make list and ./testlist. Doing this resulted in the message “Your list implementation looks OK!”. Then the rest of implementation was tested by printing the balls and testing if it works with different number of balls. All the balls are drawn on the screen and they start of at a random speed at the top of the screen. They are also removed from the screen 5 second after coming to rest at the bottom, as expected. Therefore, are the implementation most likely correct.

There are several ways to make this program work, and a lot of different features can be added to the program. During the development of the program there was some difficulties in getting the ball to stop, and not disappear out of boundary. If the speed is changed the ball will not stop, but make small jumps up and down in the x direction. If the gravity is changed the ball will stop for a second before slowly disappear outside the boundary. By testing some values were found, and they prevent this from happening.

The program leaves room for a lot of features. One feature that could have been added, if there was some more time, is the feature were a new ball are added when a specific key on the keyboard is pressed down. This can be done by using SDL and a loop that adds a new object to the list when the key is pressed down.

5 Conclusion

The task of this assignment was to implement a variant of the classic bouncing ball animation. To do these a linked list, that holds all the object data, had to be made and implemented so it could be used in the programming interface and in main. After completing all the code and necessary testing were done, the result is a variant of the classic bouncing ball program.

6 Acknowledgement

Additional participants contributing to the development of this assignment were Daniel Andreassen. For the for-loop in list_remove Johannes Figenschau contributed.

References

[1] Gnu operationg system. GNU. [Online]. Available from: https://www.gnu.org/software/make/ [Accessed November 2021].

[2] Simple directmedia layer. Libsdl. [Online]. Available from: https://www.libsdl.org/ [Accessed November 2021].

[3] Yale n patt, Sanjay j patel. Introduction to computing systems: from bits and gates to c and beyond. (3nd ed.). No place: McGraw-Hill Education; 2019.

[4] C++ [SDL] 05 Limiting the Frames Per Second. John Hammond. Available from: https://www.youtube.com/watch?v=Hze87w_WMIg. 22 April 2014.