Bubble Stack Breaker
Team No: 14
Team Name: EXCHANGE
4th year 1st semester
Department of Computer Science and Engineering
Begum Rokeya University, Rangpur

Technical Documentation

Introduction

Bubble Shooter is a simple interactive game. The goal of the game is to clear the playing field by forming groups of three or more like-colored marbles. The game ends when the balls reach the bottom line of the screen. The more balls destroyed in one shot, the more points scored. A player wins when there are no balls remaining on the playing field.

Table of Contents

Environment/Tools

We developed this game using the below tools:

1. OpenGL
2. C++
3. Code Blocks

OpenGL: Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.

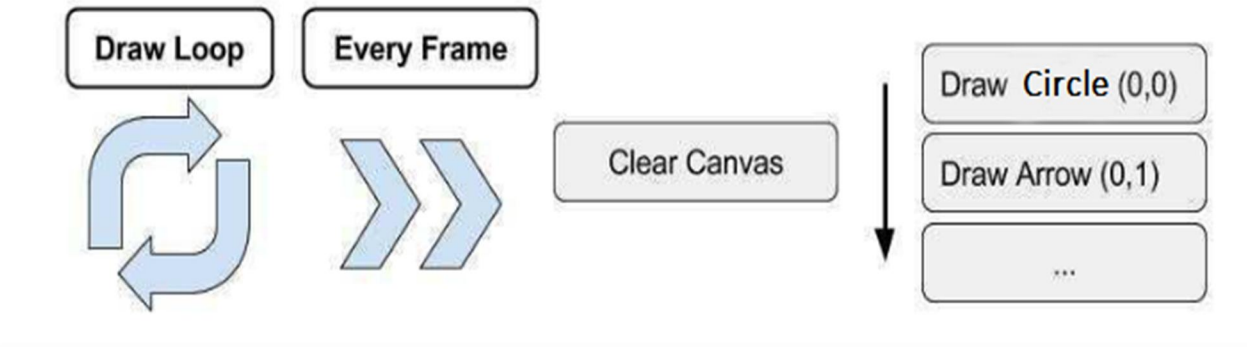
C++ : C++ (see plus plus") is a general-purpose programming language that was developed by Bjarne Stroustrup as an extension of the C language, or "C with Classes". It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, Microsoft, Intel, and IBM, so it is available on many platforms.

Code Blocks: Code::Blocks is a free, open-source cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C, C++, and Fortran. It has a custom build system and optional make support.

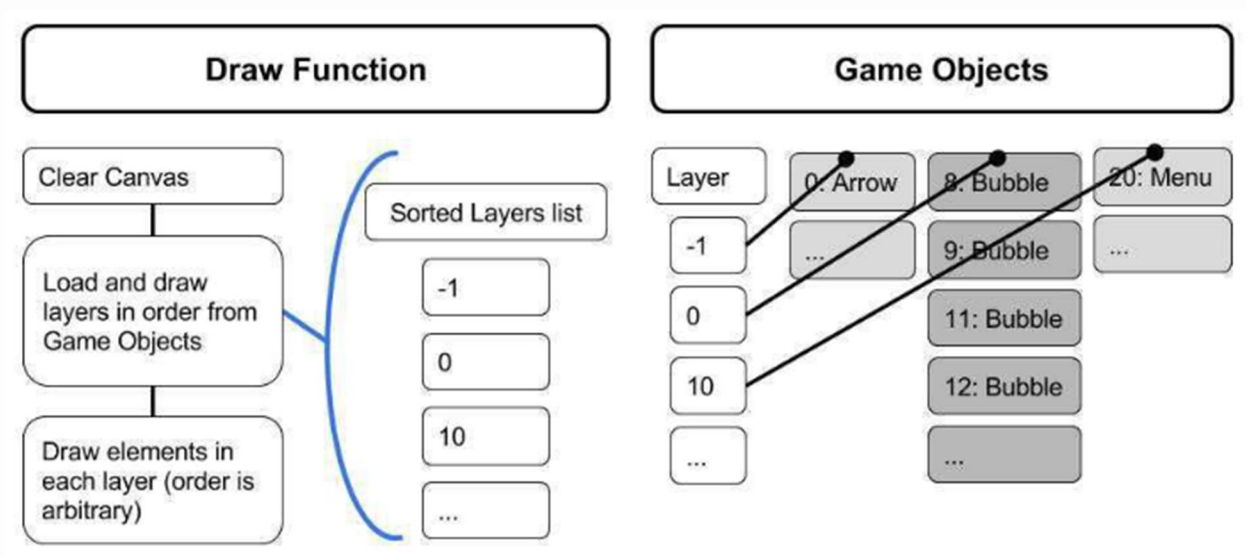
Development

Game Design

This game is made using OpenGL and C++ from scratch. The game uses a draw loop to redraw and update the game at a set interval. Each one of these draw loops represents a frame. The different elements of the game are added to a list and then drawn by the main draw loop; each element is responsible for drawing itself. This functionality along with some other core features are implemented in the **"int main(int argc, char *argv[])", "void display(void)", "void drawline(int x1,int y1)"** functions of the script **"bubbleShooter.cpp"**



This method had a few flaws because when we wanted to control when elements were drawn. If we wanted the arrow to appear behind the balls, we needed to add it before the bubbles. This was not practical so we used a layer system to achieve this. When adding elements, we specified a layer. Then we used this information to control the order in which elements were drawn. We used a nested hash table, the first level is the list of layers and the second level is the elements in each layer. We used a hash map to achieve fast insertion and deletion. All items, when added to the hash, are assigned an arbitrary id so they can be indexed.



Drawing elements is achieved through OpenGL. This allows for great flexibility and cross platform implementation that is easier than making the application in another platform.

Functions Layout

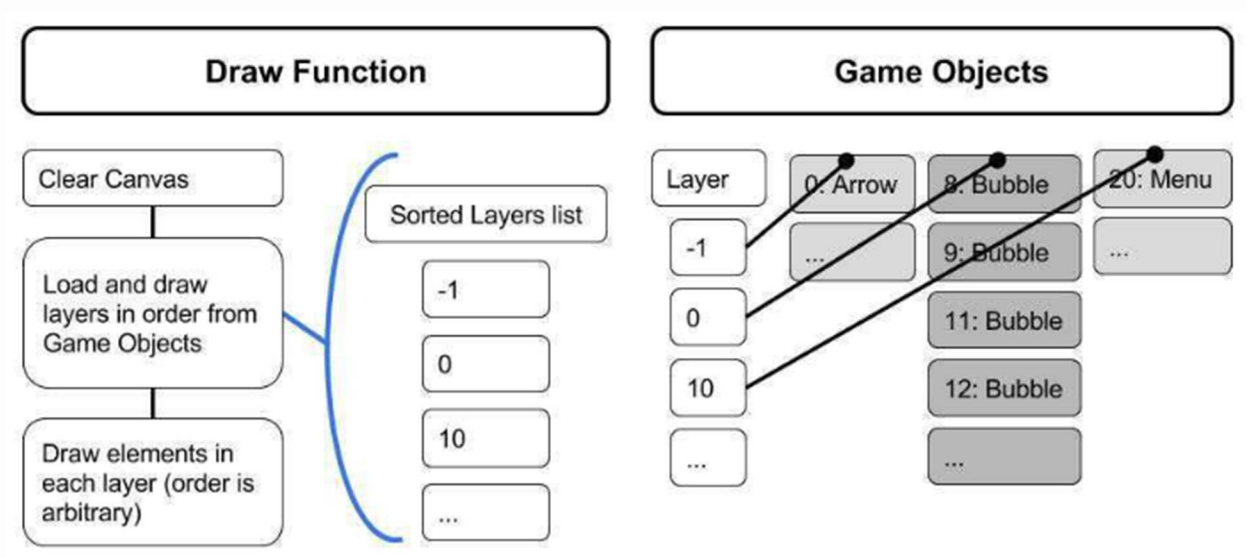
Three main objects are added to the object list and control the game.

- main() – The game start from this function.
- inits() - Primary setup is controlled by this function.
- display – This function control the display.
- circle() – All bubbles comes from this function.
- drawline() – The shooter line comes from this function.
- score() – Score is calculated by this function.
- dfs() – This function matches the same category bubbles and those path for taking action.
- gover() – This function define the result of the game.

These are all represented by same script and have additional functions to support them. One of the more prevalent objects in the project is ball. This is the colored bubble that can move across the screen and draw itself.

Shooter Design

The Shooter, Manager, Grid and Ball each have their own files that describe all the functions responsibilities. These functions are used to operate the game. The Manager controls the game progress. The shooter has a state machine to control when the ball can fire. It continuously updates the position of the arrow and, when allowed, fires a ball. The Manager controls when the shooter can fire. This is driven by events in the game. The shooter is the main way that a user interacts with the game. This interaction is controlled by either a mouse or by touching the screen.

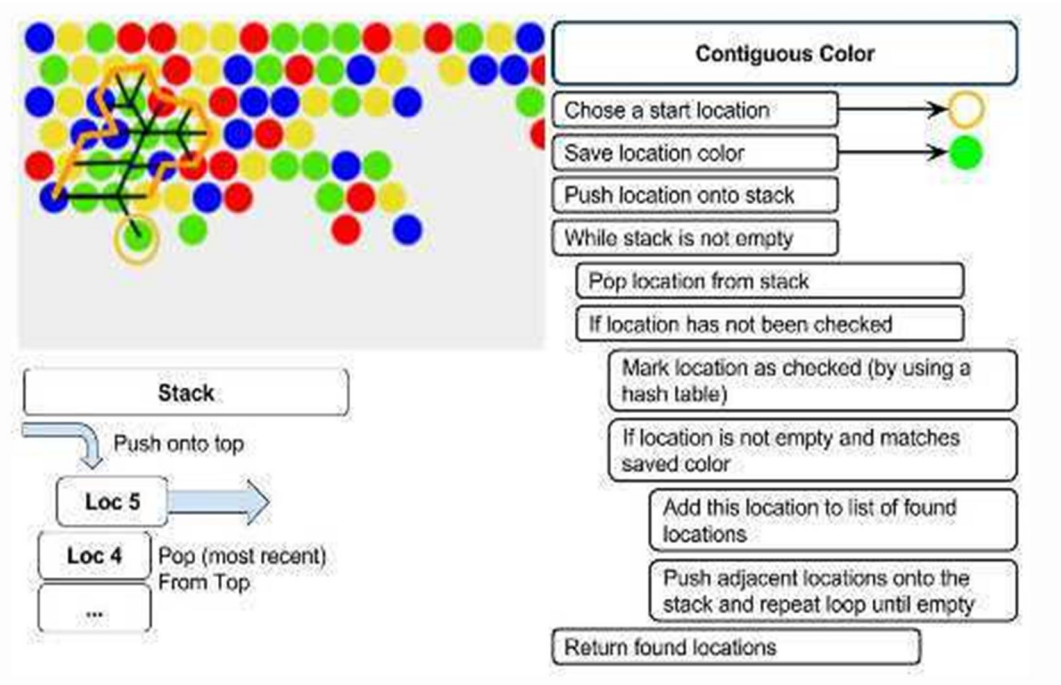


Grid Design

If the grid was a square grid, a simple solution would be to check the four locations that the ball is next to on the grid. The game uses a hex grid so this solution will not work. In order to resolve this for a hex grid, I found the location closest to the ball's current location; then I took this location and checked for collisions with the adjacent locations. This is only 7 collision checks per frame in the worst case.

Contiguous Color Identification Algorithm

When attaching bubbles to the grid, the game needs to detect for contiguous groups of bubbles. If there is a contiguous group of a specific color that at the location is larger than three, the bubbles need to be removed. In order to achieve this I used a flood algorithm that uses adjacency rules of a hex grid. I had a bit of difficulty implementing a flood algorithm. This is implemented in the **"void dfs(int i,int j,int c)"** methods in the **"bubbleShooter.cpp"** script.



Bubble Design

One of the most important elements in the game is the bubble (Also called a ball). This is a core element and is used directly or indirectly by all other elements in the game. The bubble is just a circle on the screen that has the ability to move. There is nothing very special about the bubble but this collection of functionally into **"void circle(double xc,double yc,int R,double r,double g,double b)"** of the **"bubbleShooter.cpp"** script to the

development of the game. The bubble is shared across multiple files and used to communicate between methods.

One example of this communication is between the Manager and shooter. The manager loads the shooter queue with bubbles. Then the shooter loads itself with bubbles from the queue.

Manager Design

The manager moderates all other interactions between elements within the game. As mentioned before, the manager controls the shooter directly. Additionally, the manager controls when new rows are added to the grid, decides when the player loses, and gives the player a score. This score is determined by user actions and when a fired shot collides with the grid.

The last major element the manager needed to moderate was the “Congrats, You Win”. This panel allowed the user to show the player”s score. This is done by the **“void score()”** functions of the **“bubbleShooter.cpp”** script.

Future Improvements

If we were to do this project again, we would have made a few changes. It's not perfect and we hope to use these lessons in the future. One of our major problems was to put more effort into design. We did not know what to expect and just hit the ground running. Even if these classes passed our initial requirements, the requirements changed over time because we did not do enough planning before we started working.

Additionally, we would have made the mouse controls better defined. The mouse is just a variable that is controlled by the **"void mouse()"** function. This lead to future problems with getting direct user input to activate or deactivate full screen. If we had implemented this better I could have saved many hours of troubleshooting.

In the future, we would like to add more game modes as this current version just supports one mode, lose after about 50 turns. We did not put much thought into balancing the game but it is still fun. Hopefully we will be able to use this experience to improve our future projects.