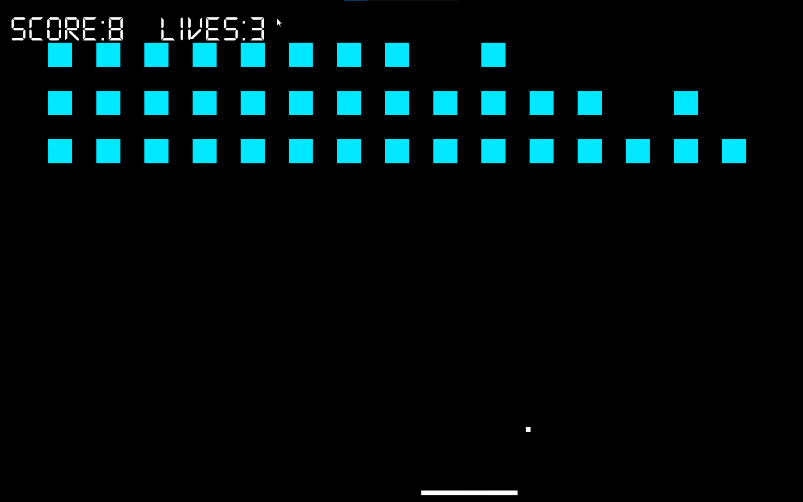
**CA 2 – Design Document**

***Luke Courtney***

**Part 1 – Overview**

As part of our second CA project, we were required to take a pre-existing pong game and improve it in some way. This was done in C++ using SFML. The unaltered game features a paddle along the bottom of the screen, and a ball that travels in a straight line until it bounces off the top or sides of the screen, or the paddle. If the ball hits the bottom of the screen a life lost. The user starts with 3 lives, but if all are lost, the game resets.

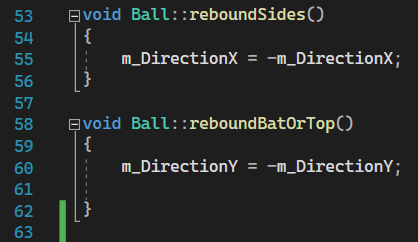
 The change that I made to the game was to turn it from a simple single player game of pong into something more akin to brick-breaker or breakout. This involved creating a grid of boxes along the top of the screen that disappear when hit by the ball. Each square destroyed grants the player a point, and when all boxes are destroyed the player wins. If the player fails to bounce the ball off the paddle 3 times, they lose all of their lives, and the game is over.

**Figure 1 - Game screen**

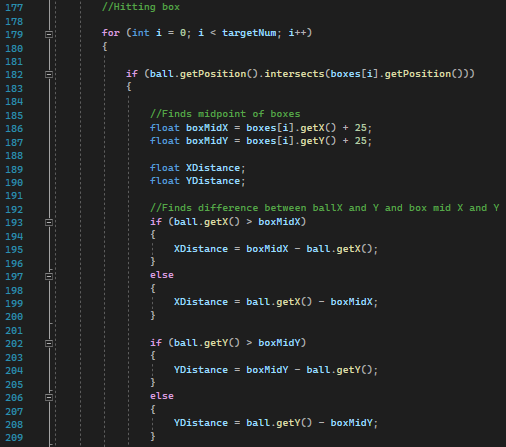
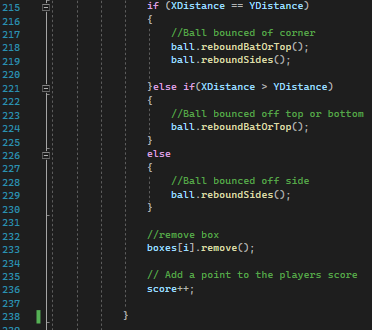
**Part 2 – Coding and Implementation**

The first change that I made to the game was creating an object for the ball to bounce off of. I created a simple 50x50 square that would appear in the top left corner of the screen. I then created an object array to store 45 of these squares and used a loop to draw them to the screen in three rows of 15. The process of having the ball bounce off of the squares proved challenging. It had to detect if it bounced off the top or bottom, or off the sides of a square and change its velocity accordingly. If it bounced off the top or bottom, it should flip its Y velocity, and if it bounced off the sides it should flip its X velocity. To do this I ran a check whenever the ball hit a square that checked if it was closer to the middle of square on the Y axis, or on the X axis. As the squares were 50x50 pixels, if the ball hit the top or bottom of the square then it would be 25 pixels away from the centre on the Y axis, and anywhere from 0 to 25 pixels away from the centre on the X axis. The check for the ball hitting the sides of a square is done the same way, but it is instead looking for the ball to be closer to the centre on the Y axis. This leaves only one literal edge-case where the ball hits the exact edge of the square. This was handled with a simple if statement checking if the Y distance is the same as the X distance. After which side of the square the ball hit was determined, the suitable velocity change would be activated using a function. If the ball hit the corner, the ball would have both its X and Y velocity inverted, though this is very rare in actual gameplay. Though this approach was successful in bouncing the ball in the correct direction, it did lead to an issue that took some time to work out.

Initial testing showed the ball to bounce correctly, though every now and then it would seem to pass through the box or bounce off in the wrong direction. I had encountered a similar issue during CA 1 while making changes to the Timber game where an object would collide with another every frame while they were in contact. In pong, this was causing the ball to “collide” with the box again even after it had changed directions. My first idea on how to fix this was to create a timer system that would not allow any collisions for a small amount of time after one had occurred. This would ideally give the ball enough time to get away from the square that it had just collided with but would not be so long as to stop the ball from colliding with a different square soon after. During the development of this system, I realized that I may have neglected to keep in mind the plan for the game further down the line. The plan involved having boxes disappear when they were hit, which would negate the need for any sort of system to avoid repeat collisions. As such, I removed what existed of the timer system and instead had the boxes set their size to 0x0 pixels on impact with the ball. This solved the problem with the collisions.

 After the targets were created and the problems with the collisions overcome, it was relatively plain sailing. I changed the scoring system to instead add a point whenever a box was destroyed, rather than whenever the ball bounced off the paddle. I then added a check for if the score was at 45, the max possible, and a check to see if the user had no lives remaining. If either occurred, the ball is frozen in the top right corner and the user is told if they have won or lost.

**Figure 2 - Code for bouncing ball**



**Figure 3 - Code for detecting ball bounce direction [2/2]**

**Figure 4 - Code for detecting ball bounce direction [1/2]**

**Part 3 – C++ and SFML**

During this project I learned an important lesson about creating a game using SFML. The issue that I encountered with the collisions solidified in my mind the concept that the code is run every frame and that I must be careful to avoid issues caused by a piece of the code being run several times in quick succession. The process was good practice programming in C++ as up until both this and CA 1, most of my experience in programming was using Java.

**Conclusion**

As part of CA 2 I created a version of the pong game similar to Breakout or Brick-Breaker where the user has to bounce a ball off a paddle to hit and destroy boxes at the top of the screen. The user wins when all boxes are destroyed but loses if the ball hits the bottom of the screen 3 times. This was done in C++ and SFML and proved to be a valuable learning experience with both.