Programming Principles Report: Breakout Project

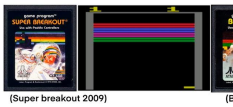
Background

Atari's Breakout is a 70's game that took inspiration from another, Pong, to build what is considered to be the first actual video game. Unlike games of its time, it was not mimicking real life but created a world that could only exist inside a screen. The gameplay is relatively simple, break the blocks with a ball by bouncing it on the sides and the paddle. You must break all the blocks to move on to the next level. Points, lives, levels, and player numbers vary depending on the version. Below is the breakdown of the differences and gameplay :

(Atari Breakout 2020)

Cool Math version of "Breakout", diffrent levels of bricks corrospond to diffrent colors, however dose not relate to how many hits to break but increasing point worth. Each round takes 468 points to move onto the next, 5 rounds in total. A counter for the points is in the top left a counter for lives is in the top middle, and Level counter in top right.

The game gave me an idea of a user friendly visual tructure, since the website Cool Math games is for children and one way of how to determine when the board has Ben cleared of blocks. While the touch screen part isn't as applicable, playing a few moments of the game I was able to see what I could add such as the start menu, the display of levels and points, along with the changing of speed depending on the angle hit and speed of the paddle on contact. When thinking about tha addition of power ups or negative bonus blocks a distinguishing characteristics could be white in this instance or some other deviation of the color palette.



(Super breakout 2009)

(Breakout 2009)

In both Atari 2600 versions Breakout and Super Breakout you have a point counter, and a life counter, the 1 on the far end is the player count and the option for 2 players. Both versions came out at the same year, however the Super is closer visually to the orignal in terms of block grouping.

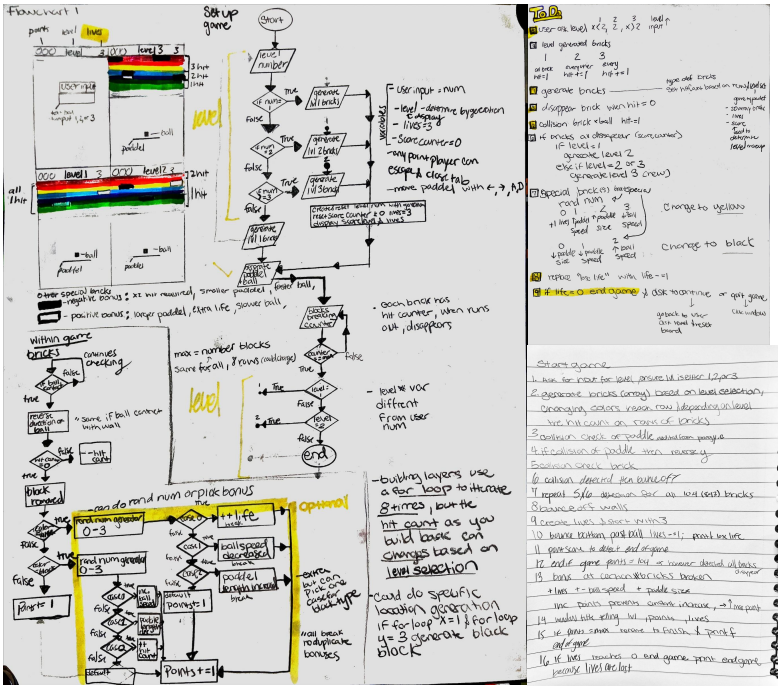
Since the colors change with each row is kept in modern day throughout more iterations, it has become the more recognizable version of Breakout. The flush bricks can be easily achieved in SDL2 and without risking bugs. Going with the point system to easily identify when to go to the next level or end the game, and visually displaying lives is critical to game functionality.

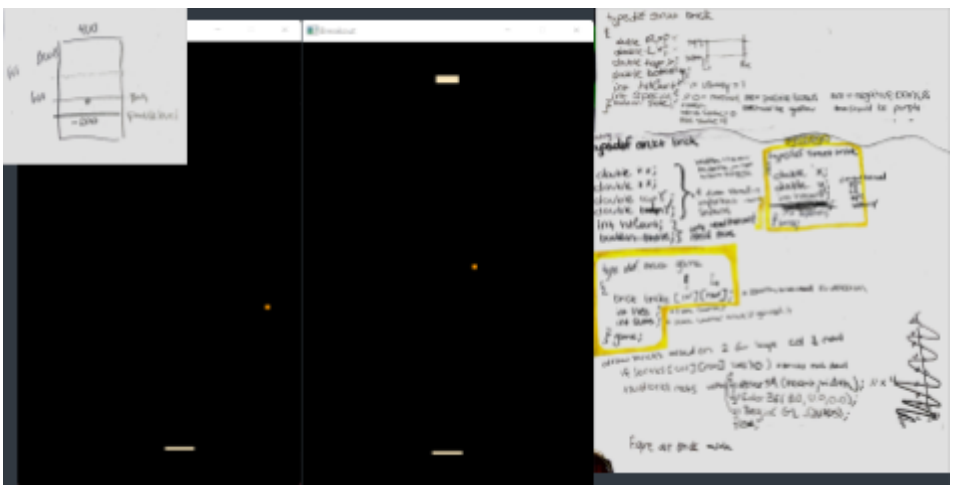
(Perry, Atari Breakout: The best video game of all time? 2022)

The arcade version of Atari's Breakout had layers colored as well, however in vertical pairs instead of each row. Players had to clear 2 screens in 3 lives, sililar to the requirements of 3 lives and including 3 levels for the brief. Each level has 448 points, the ball increases speed after a number of hits or levels, increasing the difficulty or decreasing the paddle size too. In the article " Atari Breakout: The Best Videogame of All Time?" author Perry points out that the game took inspiration from Pong, however was unique that it wasn't based on real life like most video game of the time. It has the block counter which is the 0 on the left side at the top, the right 1 at the top is the life you're on, left most 1 is the level you're on. There is also an option for two player mode, which I assume is the other set of zeros on the right side.

This first iteration of the game allows for a more steral look at the setup and final conclusions of the design of the game. The rows are always diffrent colors, challenges can be given based on points, and points are given based upon the row location of the ball. Numbers indicating life and points are the most consistent, while level count, two player option, title screens, and design of the bar varies between the three. However, by taking the minimalist requirements and common attributes, I can manipulate pongy.c, an incomplete moc of Pong, and slowly build to my version of the game Breakout.

I had decided to do this project since it would be something that would entertain me as a tester, challenge me with the number of bricks to create (preferably with an array or 2D array and plenty of for loops) and allow me to explore more with SDL. The more important part for me is that I would still want to work on the project despite its difficulties with a higher interest in it, which would keep me motivated. The Breakout project was something that felt fun and manageable to accomplish. When comparing and contrasting the differences in the Breakout version differences, I decided that being able to display levels, points, and live was critical, along with the structure of 8 rows and 13 columns for the brick board organisation. Changing the colour of each row helps identify bricks and is easier for the eyes than a solid mass of colour slowly broken down. I used the advantages of having the distinguishing factors between different rows to indicate a change in how many hits it requires to break the blocks as the levels increase. Features such as adding lives, increasing paddle size, or increasing/decreasing the speed of the ball were bonuses I added to help the player weren't in the original games. Instead, I took inspiration as to what would aid a player the most, given how difficult it is initially to see the paddle, and the original speed of the ball makes playing the game difficult when I utilised peers to test the difficulty and make alterations when needed. From the pongy.c exercise I built the code around, it mirrored the history of the original Breakout game that took inspiration from Pong, and I had kept to the essence of the Atari's version with bricks that were flush together, displaying lives, points, and the level at the top.





Through the use of flowcharting, I could understand what I needed to test thoroughly and what parts of the program to complete before being able to move on. For instance, through planning, I knew I needed to test my draw, the ball hit, and render functions with one brick in that order because of the chain reaction while still keeping in mind further to implement colour differences between rows with the array eventually initialised in main. Building code around the idea of if statements allow the detection and proper building of levels that nested for loops and will enable the transition between singular block builds and multi blocks and change variables, so the appearance of the bricks and behaviour alters between rows and columns such as colours, hit points, and locations.

With the constant revision of the program and to-do lists, I made small, manageable goals. For instance, resizing the window, getting the orientation of the paddle and ball, the collision of the paddle and the ball to work correctly, and working on function for future use in a 2D array but with one brick testing to make sure that my logic wasn't causing any issues with it. Things that I would write down would be my initial planning of the struct of the brick and revising it so that I could limit the number of variables to writing down my idea of how to accomplish a specific task with nested for loops or if statements and the desired effect that would happen in it.

Implementation

Lines 1-15: Include

In order to use SDL2 to create the window, Open GL to create the paddle, bricks, and the ball, my program to work on windows, and be able to use user input and printf, these are what I needed to include. There are also commented lines to what to put into clang or run the program depending on if you're on Windows or Linux, if the makefile isn't downloaded or, for whatever reason, doesn't work. Using Visual Studio for Microsoft, you can build the c file and run a local debugger window to play the game as I have done.

Lines 16-44:Structures

```c
typedef struct brick
{
    double x;
    double y;
    int color;//colors based on int to later assign color when drawing brick
    int hitCount;//usually 1, can change based on row, if it reaches 0 disappear
```

In order to have each part of the program to operate, I needed to store variables that could be grouped, which made multiples of, especially with bricks. However, I had to keep the variable count down as there will be over a hundred bricks to create

Lines 44-52:User Input

```c
int userInput()
{
    int choice;
    printf("Enter level choice (1,2,3): ");
    scanf_s("%d", &choice);
    printf("level : %d \n", choice);
    return choice;
}
```

A function that allows the user to select a choice and the level selection is returned and stored in the main's variable level.

Lines 52-77:Initialize Paddle, Ball, and Brick

```c
void initializeBrick(brick* sqr, double x, double y, int hit,int color)

    sqr->x = x;//middle of brick +-15
    sqr->y = y;//middle of brick +-5
    //10*30 allows for bricks to cover width of screen evenly
    sqr->hitCount = hit;//usually set to 1
    sqr->color = color;//num(usually 0 or 1)
```

By looking at Dr Eike's pongy setup for initiating these two items, I could model the InitializeBrick function properly. I had also included notes for myself when calculating Draw brick in the comments of initialising Brick. By removing the width and height of Initialize Brick comparatively to Paddle, I stored 208 fewer variables than if I had included it in the struct.

Lines 77-98: Update Paddle and BallXPaddle

```c
char ballXpaddle(ball* b, paddle* p) /* collision detection */
{   /* return if the ball has collided with the side of the paddle that faces the centre of the screen */
    if ((b->y + b->radius <= p->y + (p->height/2.0)) && (b->y - b->radius >= p->y - (p->height / 2.0)))//checks h
    {
        if ((b->x + b->radius >= p->x - (p->width/ 2.0)) && (b->x - b->radius <= p->x +(p->width / 2.0)))//checks
        {
            return 1;
        }
    }
    return 0;
}
```

Altered from Dr Eike's original pongy.c function for the collision detection to behave appropriately, and the Update Paddle allows the player to move with the reassignment of keys in the main. By troubleshooting this issue, I was also able to solve similar issues for brick.

Lines 98-111: BallXBrickTop

```c
char ballXbricktop(ball* b, brick* br) /* collision detection */
{   /* return if the ball has collided with the side of the brick  */
    if ((b->y + b->radius >= br->y) && (b->y - b->radius <= br->y + 9) )
    {
        if ((b->x + b->radius <= br->x+15&& (b->x - b->radius >= br->x - 15)))
        {
            return 1;
        }
    }

    return 0;
}
```

Using the x and y values of the brick, I was able to check the x by +- 15 or the width of the brick and y+9 to account for the radius of the ball and the height of the ball as a buffer as if I hadn't the ball would get trapped in the brick sometimes. This will return 0 or 1 in an if statement to determine if the ball needs to change direction in ball Brick.

Lines 111-159: Update Ball

```c
void updateBall(ball* b, double f, paddle* pl,brick*br, int w, int h,int* lives,int*go)
{
    /* collision detection & resolution with screen boundaries */
    if ((b->x - b->radius) < -1.0 * (double)(w / 2))
    {
        b->x = -1.0 * (double)(w / 2) + b->radius; /* ensure the ball does not go beyond the boundaries */
        b->speedX *= -1.0;
    }
    if ((b->x + b->radius) > (double)(w / 2))
    {
        b->x = (double)(w / 2) - b->radius; /* ensure the ball does not go beyond the boundaries */
        b->speedX *= -1.0;
    }
    if ((b->y - b->radius) < -1.0 * (double)(h / 2))//<= before, = was causing an error  to result in several lose lives
    {
        b->y = -1.0 * (double)(h / 2) + b->radius; /* ensure the ball does not go beyond the boundaries */
        b->speedY *= -1.0;
        *lives -= 1;//life deduction
        printf("Lose Life\n"); fflush(stdout);// hits and bounces off, sucessfully records only once without error can be replaced with deduction of life coun
        if (*lives <= 0)
        {
            printf("\n\n\nYou lost all your lives : (\n\n\n"); fflush(stdout);//tells user what caused the closing of the window
            *go = 0;//closes window if lives reaches 0
        }
    }
    if ((b->y + b->radius) > (double)(h / 2))
    {
        b->y = (double)(h / 2) - b->radius; /* ensure the ball does not go beyond the boundaries */
        b->speedY *= -1.0;
    }

    /* collision detection with paddles*/
    if ((ballXpaddle(b, pl)))
    {
        b->y += 6;
        b->speedY *= -1.0;
    }
    /*collision detection with brick*/
    if ((ballXbricktop(b, br)))
    {
        b->speedY *= -1.0;
        br->hitCount -= 1;
    }

    /* update position */
    b->x += f * b->speedX;
    b->y += f * b->speedY;
}
```

Altered from Dr Eike's pongy.c method to include lost life in lines 129-134, and it's recorded that lives have been lost. This is also where if all lives are deducted, then the window closes, ending the game. It was also the function I had first allowed detection for the brick collision. While it did work, doing it on a massive scale and calling this function for each brick resulted in the speeding up of the ball due to the updated position in lines 155-157, which led to the creation of another function BallXBrick.

Lines 159-197: BallBrick

```
159 void ballBrick(ball*b, brick*br, int *points,int *lives, paddle* p1)//contact with brick action
160 {
161     if ((ballXbricktop(b, br))&& br->hitCount>0)
162     {
163         b->y -= 6;//prevents weirdness of it going through paddle
164         b->speedY *= -1.0;//reverses ball
165         br->hitCount -= 1;//reduces hit count
166         *points += 1;//when ball broken adds points
167         if (*points == 5)
168         {
169             *points += 1;//points become 6 and range of 105
170             *lives += 1;
171             b->speedY *= 0.8;//dec. speed by 20%
172             p1->width += 5;//inc size of paddle
173             p1->height += 10;
174         }
175         if (*points == 30)
176         {
177             *points += 1;//points become 31 and range of 106
178             *lives += 1;
179             p1->height += 5;
180
181         }
182         if (*points == 50)
183         {
184             *points += 1;//points become 51 and range of 107
185             *lives += 1;
186             b->speedY *= 1.2;//increase speed by 20%
187         }
188         if (*points == 100)
189         {
190             *points += 1;//points become 101 and range of 108
191             *lives += 1;
192             b->speedY *= 0.8;//dec. speed by 20%
193             p1->height *= 1.5;//inc size of paddle
194         }
195     }
196
197 }
```

With the utilisation of the ballXbricktop function, I was able first to reverse the ball speed and adjust the y to remove and conflicts in the collision, delete a hit count so that I can use it to determine when to show the ball and when not to. In addition, depending on the points, bonuses are awarded to the player when breaking a certain amount of bricks and adding to the points, so it doesn't repeat until another brick is broken. Bonuses include lives added, width and height (visibility) of the paddle to increase, and the ball speed to be deduced or added upon. When determining when the user has eliminated all bricks, one thing to consider is the additional points. Hence, I needed to increase the max points with each bonus because of the additional point.

Lines 197-286: Draw Ball, Paddle, and Bricks render

```
void drawBricks(brick* br)
{
    GLint matrixmode = 0;
    glGetIntegerv(GL_MATRIX_MODE, &matrixmode); /* get current matrix mode */

    glMatrixMode(GL_MODELVIEW); /* set the modelview matrix */
    glPushMatrix(); /* store current modelview matrix */
    glTranslated(br->x, br->y, 0.0); /* move the brick to its correct position */

    glBegin(GL_QUADS); /* draw brick */
    switch (br->color)//checker for selection
    {
    case 0://normal bricks level 1
        glColor3f(0.933f, 0.573f, 0.298f);//orange but diffrent from balll
        break;
    case 1://bricks  level 1+2
        glColor3f(0.616f, 0.851f, 0.824f);
        break;
    case 2://bricks level 2+3
        glColor3f(0.984f, 0.91f, 0.737f);//yellow
        break;
    case 3://bricks level 3
        glColor3f(0.294f, 0.22f, 0.776f);//purple
        break;
    default://color not assigned
        glColor3f(1.0f, 1.0f, 1.0f);//orange
    }
    glVertex3d(-15, 5, 0.0);//calculate the 4 vertexes depending on width and height established
    glVertex3d(15, 5, 0.0);
    glVertex3d(15, -5, 0.0);
    glVertex3d(-15,-5, 0.0);
    glEnd();
    glPopMatrix(); /* restore previous modelview matrix */
    glMatrixMode(matrixmode); /* set the previous matrix mode */
}
```

While looking at the draw Paddle and drawBall I could model a similar drawBrick function. However, instead of a constant colour like these two in pongy.c, I used a switch case and the variable colour stored within a brick to determine what colour the brick would be. This allows different levels to have different colour bricks to be generated. I used OpenGL, similarly to the previously mentioned function and the commented size from the brick struct and set the coordinated to base my glVertex3D off of and properly size my bricks. Because each of the bricks would be the same size, I only needed the middle coordinates of the brick. To get the proper colours, I originally had the hex colours for my palette and used *Convert hex colour to GLSL VEC3* from *Create a gradient - coolors* to have the proper colours appear, which helped change the colours of the paddle and ball as well.

Lines 276-285 are from the pongy.c render slightly altered to render a brick, ball, and paddle. However, I couldn't use this function to properly render the 2D array of bricks so I had to render the bricks using a different method than within this tester part.

Lines 288-384: Variables Used

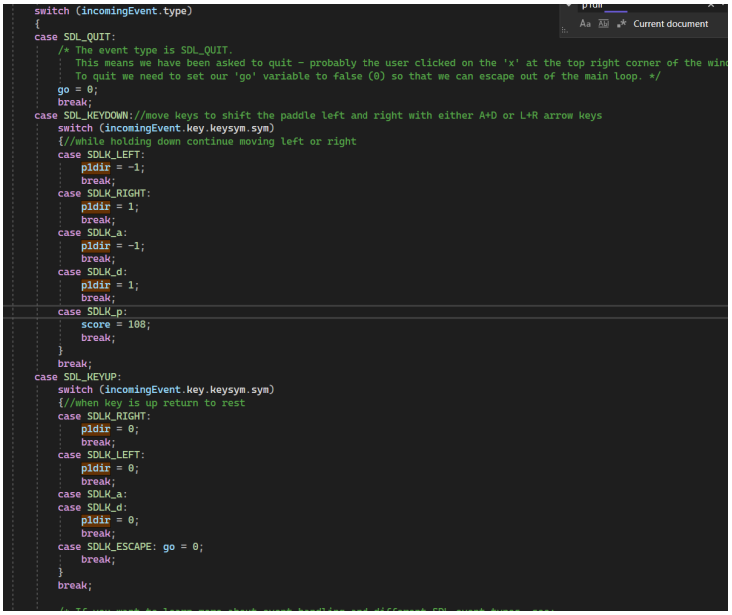
```
387     initializeBall(&myB, 0.0, -200.0, 6.0, 100.0, 100.0);
388     initializePaddle(&p1, 0.0, -250.0, 4, 40, 1, 100.0);//check top of the paddle
389
390     initializeBrick(&b1, 0,0, 1, 1);//one brick initalized as a starter needed for below
391     //was running into issues doing it straight from the array, unfortunetly
392
393         if (level == 3)
394         {
395             for (int i = 0; i < 8; i++) {//makes array of bricks
396
397                 for (int j = 0; j < 13; j++) {
398                     b1.x = (-180) + (j * 30);
399                     b1.y = (220) + (i * 10);
400                     b1.hitCount = i+1;//each level bricks inc 1 more hit needed
401                     b1.color = ((i % 2)+2);
402                     bricks[j][i] = b1;
403                 }
404             }
405         }
406         else if (level == 2)
407         {
408             for (int i = 0; i < 8; i++) {//makes array of bricks
409
410                 for (int j = 0; j < 13; j++) {
411                     b1.x = (-180) + (j * 30);
412                     b1.y = (220) + (i * 10);
413                     b1.hitCount = (int)(i/2)+1;//slow increase
414                     b1.color = ((i % 2)+1);//alternates colors
415                     bricks[j][i] = b1;
416                 }
417             }
418         }
419         else// level 1
420         {
421             for (int i = 0; i < 8; i++) {//makes array of bricks
422
423                 for (int j = 0; j < 13; j++) {
424                     b1.x = (-180) + (j * 30);
425                     b1.y = (220) + (i * 10);
426                     b1.hitCount = 1;
427                     b1.color = (i % 2);//alternates colors
428                     bricks[j][i] = b1;
429                 }
430             }
431         }
```

This is where I had initialised variables I would be using, such as starting lives being 3, level selected by user input, bricks, and window title. I had changed the window size so that it could be a side game while you work on something else and not cover the entire screen. Adding to this, I also needed to make sure the window width was changed so that it was easily divisible by 13 so the bricks would be flush with each other and to the sides of the screen. For initialising the bricks, I chose to work with a 2D array, and one of the issues I encountered was creating the 2D array in a way that I could alter. Eventually, I had done brick bricks[13][8] = {0}; and assigned the starter brick in the initialising phase to overcome one of these issues. Lines 318-384 is the creation of the window from pongy.c altered using the variables before it

Lines 385-431: Initializing Ball, Paddle, and Bricks

In a similar structure to initilizePaddle, I had initializedBrick however, trying to do this for all 104 bricks had resulted from what I had tried errors in rendering the bricks. So, I used the original teaser brick, adjusted the variables stored in it, and assigned it to brick [col][row] through nested for loops. By use of the level that was chosen before, three different levels generate with different increasing hit counts and appearances. Using the row or i variable in the nested for loop, I was able to increase the hit count either by every other or by each, depending on how I assigned the b1.hitcount. Adding to this, the colour switch I could use as well, along with i (row count) switches every other row colour. After all the alterations to b1, the initialising brick, I assigned it to the correct brick.

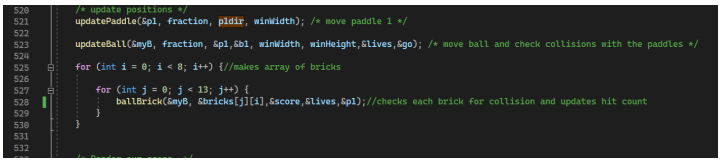
Lines 433-517:Key Events



```
switch (incomingEvent.type)
{
case SDL_QUIT:
    /* The event type is SDL_QUIT.
       This means we have been asked to quit - probably the user clicked on the 'x' at the top right corner of the wind
       To quit we need to set our 'go' variable to false (0) so that we can escape out of the main loop. */
    go = 0;
    break;
case SDL_KEYDOWN://move keys to shift the paddle left and right with either A+D or L+R arrow keys
    switch (incomingEvent.key.keysym.sym)
    {//while holding down continue moving left or right
    case SDLK_LEFT:
        pldir = -1;
        break;
    case SDLK_RIGHT:
        pldir = 1;
        break;
    case SDLK_a:
        pldir = -1;
        break;
    case SDLK_d:
        pldir = 1;
        break;
    case SDLK_p:
        score = 108;
        break;
    }
    break;
case SDL_KEYUP:
    switch (incomingEvent.key.keysym.sym)
    {//when key is up return to rest
    case SDLK_RIGHT:
        pldir = 0;
        break;
    case SDLK_LEFT:
        pldir = 0;
        break;
    case SDLK_a:
    case SDLK_d:
        pldir = 0;
        break;
    case SDLK_ESCAPE: go = 0;
        break;
    }
    break;
```

This is where we say that go = 1 or that the game is set up and the key events from pongy. c was altered so that the paddle would move side to side and move on the x-axis, not the y. If you click p you also automatically win the game, even if the bricks are not all broken. Given how long the game takes to complete, I wanted the player to have some sort of easier way to win or cheat at the game. This sparked me to allow the bricks to be hit several times despise initially being a mistake it allows for a different goal of the game to hit on the sides of the brick after making a ‘tunnel-like structure to more efficiently win properly instead of this shortcut.

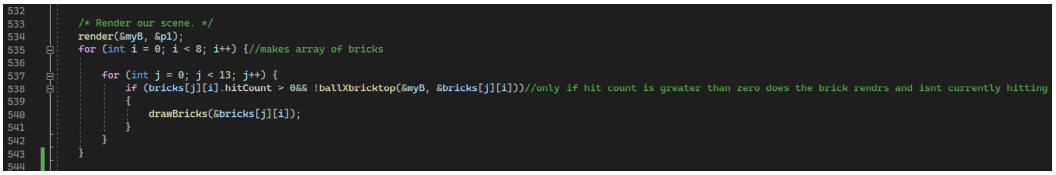
Lines 520-530: Update Ball and Bricks



```
    /* update positions */
    updatePaddle(&p1, fraction, pldir, winWidth); /* move paddle 1 */

    updateBall(&myB, fraction, &p1,&b1, winWidth, winHeight,&lives,&go); /* move ball and check collisions with the paddles */

    for (int i = 0; i < 8; i++) {//makes array of bricks

        for (int j = 0; j < 13; j++) {
            ballBrick(&myB, &bricks[j][i],&score,&lives,&p1);//checks each brick for collision and updates hit count
        }
    }
```

Using a nested for loop, I was able to utilise the collision detection between ball and bricks, and using updateBall would have sped up the ball and made the game inoperable. The ball will only hit the brick and detect the space as a brick if the hit count is greater than zero as an indicator for visibility else it treats it as an invisible brick and is later displayed as such.

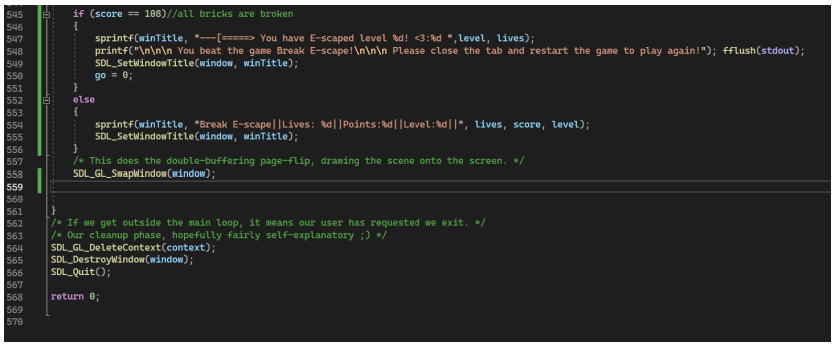
Lines 531-542: Render



```
    /* Render our scene. */
    render(&myB, &p1);
    for (int i = 0; i < 8; i++) {//makes array of bricks

        for (int j = 0; j < 13; j++) {
            if (bricks[j][i].hitCount > 0&& !ballXbricktop(&myB, &bricks[j][i]))//only if hit count is greater than zero does the brick rendrs and isnt currently hitting
            {
                drawBricks(&bricks[j][i]);
            }
        }
    }
```

A nested for loop is used again to access the 2D array of the bricks, and it will only render bricks that the hit count is greater than 0 and the ball is currently not touching, which would, on my computer, result in improper rendering. If those conditions are met, then drawing bricks is accomplished, else then the bricks are invisible.

Lines 544-555: Window Title

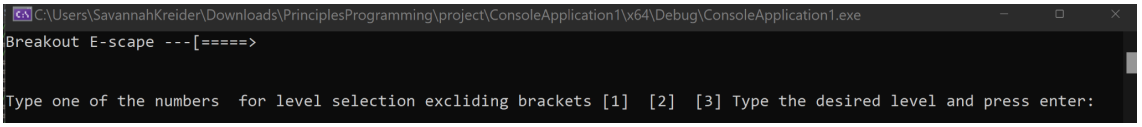


```
    if (score == 108)//all bricks are broken
    {
        sprintf(winTitle, "~~~[=====> You have E-scaped level %d! <3:%d ",level, lives);
        printf("\n\n\n You beat the game Break E-scape!\n\n\n Please close the tab and restart the game to play again!"); fflush(stdout);
        SDL_SetWindowTitle(window, winTitle);
        go = 0;
    }
    else
    {
        sprintf(winTitle, "Break E-scape||Lives: %d||Points:%d||Level:%d||", lives, score, level);
        SDL_SetWindowTitle(window, winTitle);
    }
    /* This does the double-buffering page-flip, drawing the scene onto the screen. */
    SDL_GL_SwapWindow(window);

}
/* If we get outside the main loop, it means our user has requested we exit. */
/* Our cleanup phase, hopefully fairly self-explanatory ;) */
SDL_GL_DeleteContext(context);
SDL_DestroyWindow(window);
SDL_Quit();

return 0;
```

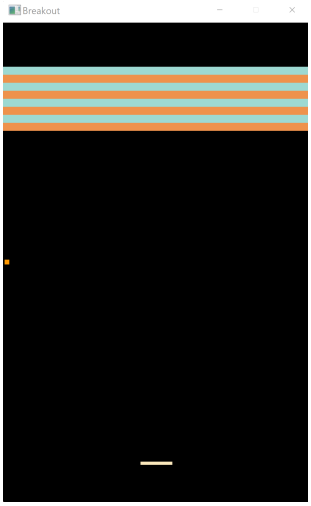
As I troubleshot for text display, I used a similar setup to loony. C and utilised the window title to be able to display the lives, level, and points. By making the window title a char array, I was able to use sprintf to convert the string to the array properly and be able to update easily. It also allows me to change the title when all points are awarded to the player and thus, the window is closed. A record is then displayed of the actions of the player such as losing lives, if they had beaten the game or if they had lost all their lives,

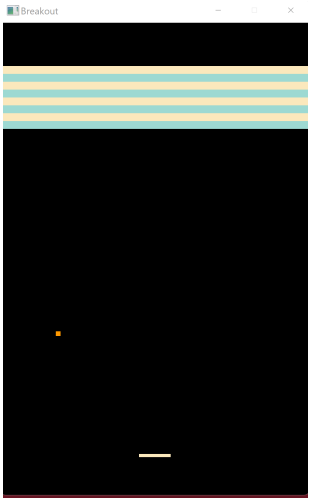
Result

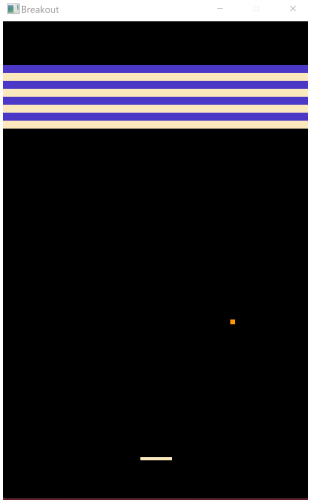
The user is prompted to select a level and is given directions on how to do so

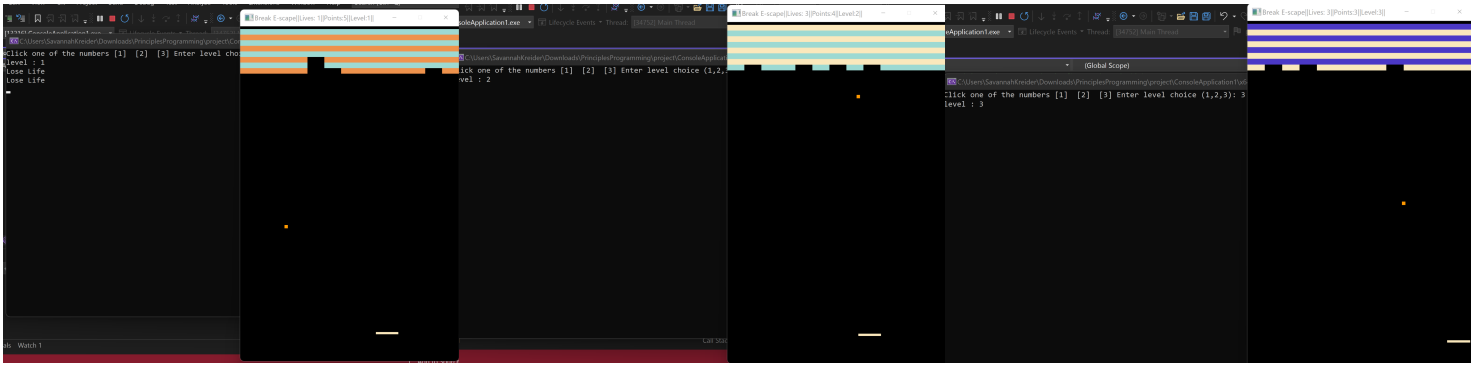


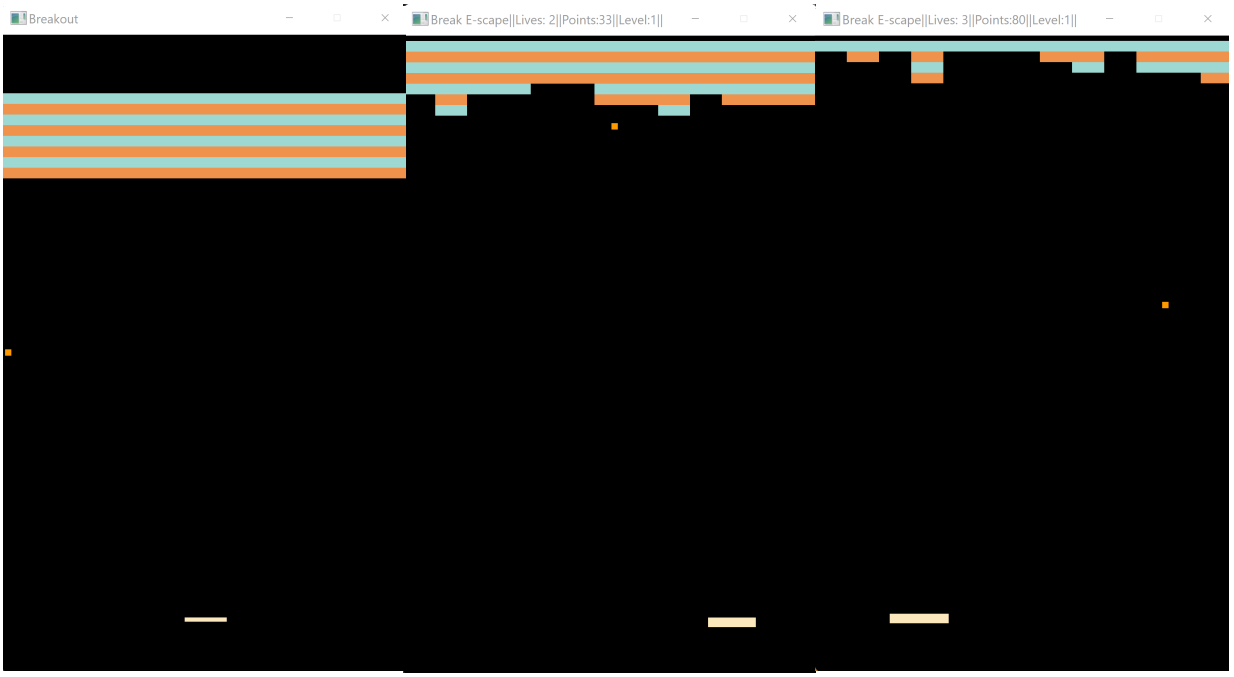
After the user has selected, there are 3 different levels that could generate level 1 with orange, level 2 with a light blue, and level 3 with a purple-blue that's every other. Difficulty in breaking the bricks goes as the following: level 1 all bricks take 1 hit, level 2 every other row the brick's hit count needed to break increases and level 3 every level the hit count increases.



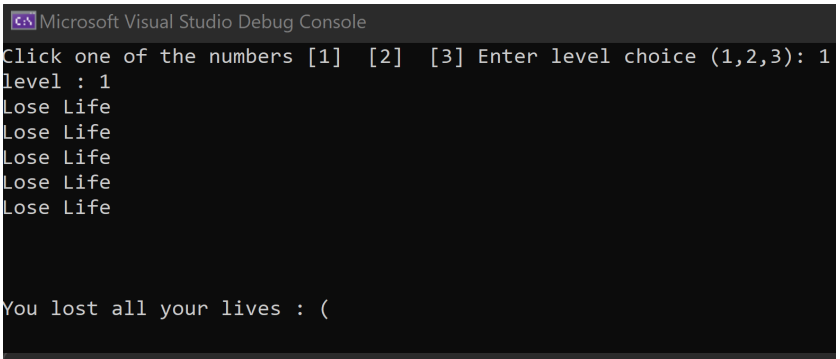




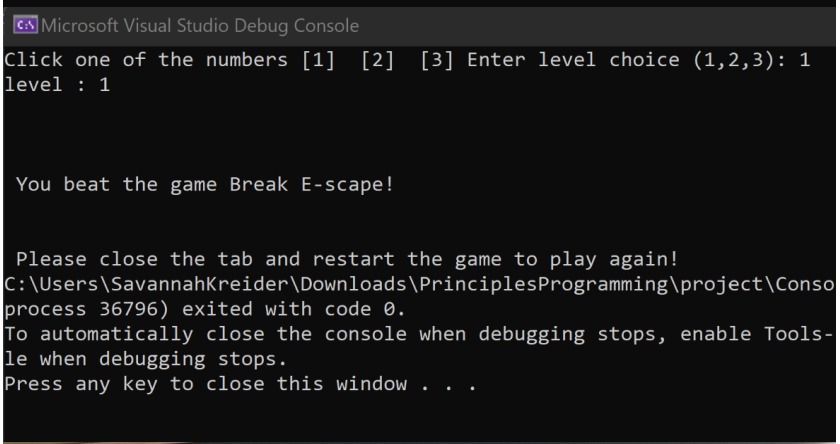




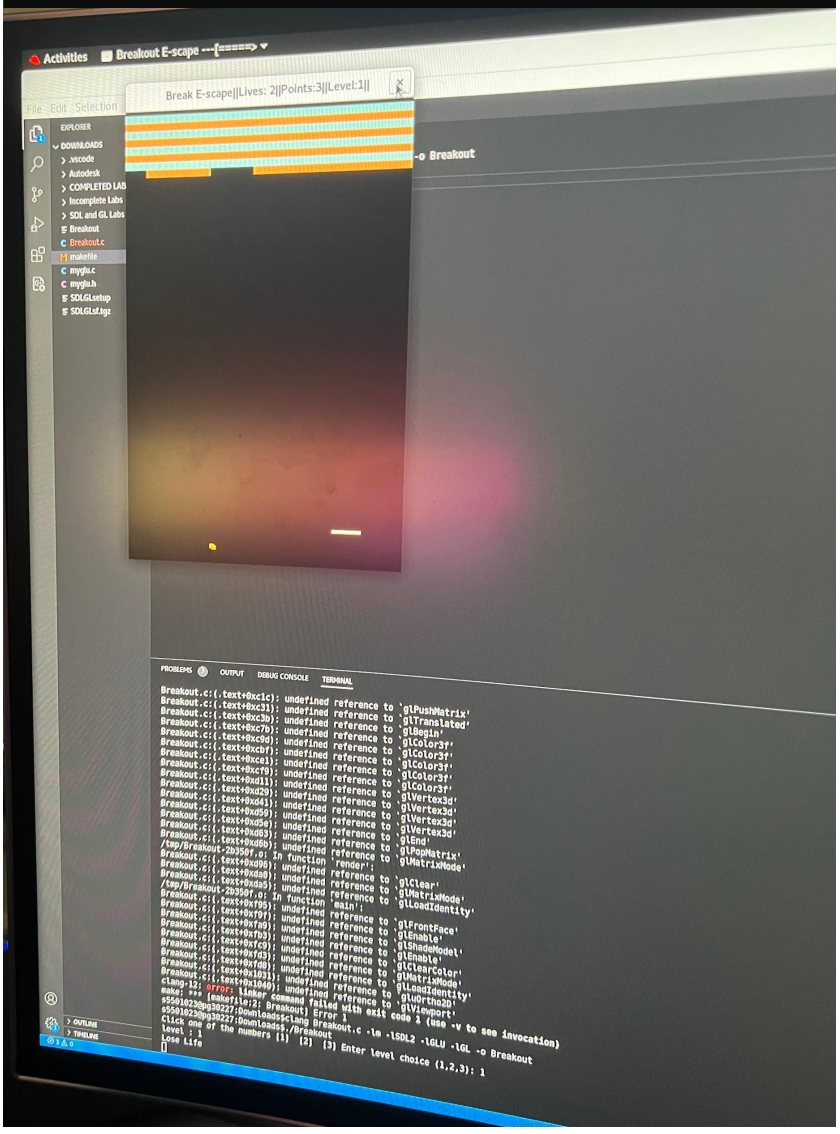
As points are awarded the title changes with it, and bonuses such as earning lives, increasing paddle with and height change along with it. Above is an example of how the title changes with each level as well.



Ending 1 is that the user has unsuccessfully broken all the bricks and that all the lives, including bonus lives awarded points, and the window is closed to display the # of lost lives and that it was unsuccessful because of the life count reaching 0.



Ending 2 is that either the player has clicked p, which I used for demo purposes due to how long a game can get, or that the player truthfully broke all the bricks and got all the game's bricks. THe successful message is sent with beating the game, and the user is instructed to close the tab and reselect a level.



Proof it works on Linux and not just windows

Project Reflection

I had broken down a lot over trying to get the array of bricks to work better, resulting in a lot of time spent panicking. Often times after these breakdowns, I was able to have some sort of success, like building a brick or something, only to encounter another part, such as the ball collision not working. With most of my program being built during the break, it was difficult to ask for help or know where to turn to for help. While there were some I could easily search up, such as why I was incorrectly initialising the array like array [][]; it needed ={0}; resulting in another workaround to properly set up bricks. I'm glad that I have gotten to this point, but if I were to expand on the program, I would have the levels back to back so that when you complete level 1 you move to level 2. Another way I would expand upon this, if I had the knowledge to do this, is to have a screen saying completed level held for a duration. I think a lot of things I had wanted to do initially were difficult due to my knowledge of SDL and OpenGL and their capabilities and trying to teach the basics to myself first. As a side game that one could have on a computer while you work, it accomplishes this objective, with the first level taking more than 15 minutes to complete, but I needed to include some "holes" because of how long it gets exponentially. It's not designed for a quick game and may lack in grabbing attention. This was something I had observed when I let my friends play the game. It was challenging and they grew frustrated with it. If I had to make the game more than just my intended side window game I would enlarge everything especially on the lab monitors which are so big compared to my laptop screen.My struggle to grasp concepts like dereferencing was improved and learned when to use ->, *, and & in the proper circumstances really helped in mastering that fairly quickly when analysing pongy.c and loony.c. For a lot of the project tI felt lost and panicked that my lack of knowledge despite researching I couldn't implement till I had really played around with values in pongy.c and trial and error in methods I had brainstormed in getting the brick to work properly which had resulted in me spiralling till I was in a better head space I'm very happy, even if the breakthrough don't seem like much on the documentation, I was able to come up with solutions and tried my best. I'm not the best programmer but I feel less insecure about my knowledge after this knowing what I had learned from it was things I struggled with during lessons.

References.

1. Atari Breakout (2020) Coolmath Games. Available at: https://www.coolmathgames.com/0-atari-breakout (Accessed: December 17, 2022)
2. Super breakout (2009) Atari 2600 VCS Super Breakout : scans, dump, download, screenshots, ads, videos, catalog, instructions, roms. Available at: http://www.atarimania.com/game-atari-2600-vcs-super-breakout_7848.html (Accessed: December 20, 2022).
3. Breakout (2009) Atari 2600 VCS Breakout : scans, dump, download, screenshots, ads, videos, catalog, instructions, roms. Available at: http://www.atarimania.com/game-atari-2600-vcs-breakout_18135.html (Accessed: December 20, 2022).
4. Eike Anderson (2022) ponty.c (Available on lab 9 on Brightspace)
5. Eike Anderson (2022) pongy.c (Available on lab 9 on Brightspace)
6. Eike Anderson (2022) loony.c (Available on lab 11 on Brightspace)
7. Perry, T.S. (2022) Atari Breakout: The best video game of all time?, IEEE Spectrum. IEEE Spectrum. Available at: https://spectrum.ieee.org/atari-breakout (Accessed: December 20, 2022).
8. Realities, P. *2D Shoot 'Em Up Tutorial*, *SDL2 - creating a 2D shoot 'em up*. Available at: https://www.parallelrealities.co.uk/tutorials/shooter/shooter11.php (Accessed: January 3, 2023).
9. *Convert hex color to GLSL VEC3*. Available at: https://airtightinteractive.com/util/hex-to-glsl/ (Accessed: January 3, 2023).
10. *Create a gradient - coolors* (no date) *Coolors.co*. Available at: https://coolors.co/gradient-maker/9dd9d2-ee924c?position=53%2C53&opacity=100%2C100&type=linear&rotation=90 (Accessed: January 16, 2023).

Other research sources

(That I can recall helped the most)

-*Breakout* (no date) *LearnOpenGL*. Available at: https://learnopengl.com/In-Practice/2D-Game/Breakout (Accessed: January 16, 2023).

-*C Arrays* (no date) *C arrays*. Available at: https://www.w3schools.com/c/c_arrays.php (Accessed: January 16, 2023).

-*Getting started with SDL and C* (2021) *Learn C Games Programming Blog*. Available at: https://learncgames.com/tutorials/getting-started-with-sdl-and-c/ (Accessed: January 16, 2023).

-Holtkamp, C. (no date) *SDL2-C*, *Acry's Coding Site*. Available at: https://acry.github.io/SDL2-C.html (Accessed: January 16, 2023).

-Jamal JenkinsJamal Jenkins 111 bronze badge and Nir CohenNir Cohen 122 bronze badges (1969) *How do I pass 2D struct arrays to different functions [SDL2/opengl breakout project]*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/70491755/how-do-i-pass-2d-struct-arrays-to-different-functions-sdl2-opengl-breakout-proj (Accessed: January 16, 2023).

-*Lazy Foo' productions* (no date) *Lazy Foo' Productions - Beginning Game Programming v2.0*. Available at: https://lazyfoo.net/tutorials/SDL/ (Accessed: January 16, 2023).

-Realities, P. (no date) *create a 2D shoot em up*, *SDL2 - creating a 2D shoot 'em up*. Available at: https://www.parallelrealities.co.uk/tutorials/shooter/shooter13.php (Accessed: January 16, 2023).

-Rembound (2020) *The breakout tutorial with C++ and SDL 2.0*, *Rembound*. Rembound. Available at: https://rembound.com/articles/the-breakout-tutorial-with-cpp-and-sdl-2 (Accessed: January 16, 2023).

-*Scanf() and fscanf() in C* (2022) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/scanf-and-fscanf-in-c/?ref=rp (Accessed: January 16, 2023).

-*SDL Library in C/C++ with examples* (2022) *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/sdl-library-in-c-c-with-examples/ (Accessed: January 16, 2023).

-*C Dereference Pointer - javatpoint* (no date) *www.javatpoint.com*. Available at: https://www.javatpoint.com/c-dereference-pointer (Accessed: January 16, 2023).

-*What is scanf_s in C?* (no date) *Educative*. Available at: https://www.educative.io/answers/what-is-scanfs-in-c (Accessed: January 16, 2023).

Self Care Notes / Reflection

(kinda personal but I am planning on spending next week getting some help with my changes in behaviour)

I think with how much stress I put myself under it had resulted in a panic attack in Asda suddenly with the combination of not sleeping and stuff . I really need to focus on myself during projects and not to freak out. A lot of that anxiety probably could have been avoided if I had asked and reached out for help but felt like I couldn't in combination with other projects due before this it wasn't something that I was monitoring. I really had beaten up on myself when I couldn't get simple things such as initialising an array without values to assign which caused a detriment to my morale in completing the game. When I did have success I couldn't sleep, again, another thing that I need to make a priority in uni.