

Breakout Returns

Source Code Preview

Being my first project, I had some fun getting a solid framework down. I went much more object oriented with this project, learned a lot, and still use this as a fundamental basis for my personal, and academic, projects.

Source Files

- main.cpp
- Audio.cpp
- Audio.h
- Button.cpp
- Button.h
- Display.cpp
- Display.h
- Engine.cpp
- Engine.h
- Font.cpp
- Font.h
- Game.cpp
- Game.h
- Gem.cpp
- Gem.h
- Global.cpp
- Global.h
- Input.cpp
- Input.h
- Message.cpp
- Message.h
- Timer.cpp
- Timer.h

Responsibilities:

- Programming
- Music Development
- Design
- Graphic Art Development
- Documentation

Source File: main.cpp

```
#include <SDL.h>
#include "Global.h"

int main(int argc, char* args[])
{
    // Initialize Message Log System
    Message::Init();
    Message::logMessage("Welcome to Breakout Returns Log Messaging System!");

    Message::logMessage("Initializing Engine Components...");
    gp_Engine = new Engine();           // Create New Engine Object
    gp_Engine->Init();                  // Initialize Engine Components
    Message::logMessage("Success!");

    Message::logMessage("Starting Breakout Returns! Have Fun!");
    // Begin Game Loop
    while(gp_Engine->m_Done != true)
    {
        SDL_FillRect(sp_Screen, 0, 0); // Clear Screen

        gp_Engine->Update();           // Update Display

        SDL_Flip(sp_Screen);          // Show Display
    }

    Message::logMessage("Thank You For Playing Breakout Returns!");
    // Clean Engine Object
    gp_Engine = NULL;
    // Close Down SDL
    TTF_Quit();
    SDL_Quit();
    // Time To Leave. Mission Complete.
    Return 0;
}
```

Source File: Game.cpp

Description: This is the core of Breakout Returns. Managing the loading of windows and each action that window produces each logical cycle.

```
void Game::Update()
{
    int x = 20;
    int y = 10;

    switch(GAMEWINDOW)
    {
    case 0: // Logo Screen
        // Load Initial Menu Music
        gp_Audio->PlayLevelMusic(0, -1);
        // If Logo is not faded, Fade It
        if(LogoFaded == false)
            gp_Display->FadeIn(sp_Screen, sp_LogoScreen, 1);
        // Keep the surface on the screen
        gp_Display->PlaceSurface(0, 0, sp_LogoScreen, sp_Screen);

        GameDelay = SDL_GetTicks();
        // If 4 seconds have elapsed, move on to the game menu
        if(GameDelay > 4000)
        {
            gp_Display->CrossWinFade(sp_Screen, sp_LogoScreen, sp_LoadScreen, 1);
            GAMEWINDOW = 1;
        }
        break;
    case 1: // Main Load Screen
        // If the Cursor is not shown, Show It
        if(Cursor == false)
            Cursor = true;
        // Keep the load image on the screen
        gp_Display->PlaceSurface(0, 0, sp_LoadScreen, sp_Screen);
        break;
    case 2: // Game Window
        // Place the levels background
        UpdateGameLevel(gp_Input->GameLevel);
        // Display the users lives
        gp_Display->DisplayLives();
        // Display the paddle
        gp_Input->DisplayPaddle();
        // Start the block position at 0
        block = 0;

        // Update Blocks
        for(int i = 0; i < 11; i++)
        {
            for(int j = 0; j < 11; j++)
            {
                gp_Display->PlaceSurface(x, y, sp_Tiles, sp_Screen,
                &BlockClip[GameBoard[i][j]]);
                gp_Display->Blocks[block].xLoc = i;
                gp_Display->Blocks[block].yLoc = j;
                gp_Display->Blocks[block].X = x;
                gp_Display->Blocks[block].Y = y;
                block++; x += 50;
            }
            x = 20; y += 20;
        }
        // Update Block Fonts
        gp_Display->UpdateFont();
        // Is the game complete?
        CheckLevelCompletion();
        break;
    }
```

```

case 3: // Score Window
    // Load the high scores
    LoadHighScores();
    // Keep the score image on the screen
    gp_Display->PlaceSurface(0, 0, sp_ScoreScreen, sp_Screen);
    // Update the scores (if needed)
    UpdateScreenScore();
    break;
case 4: // Option Window
    // Keep the option image on the screen
    gp_Display->PlaceSurface(0, 0, sp_OptionScreen, sp_Screen);
    break;
case 5: // Credit Window
    // Keep the credit image on the screen
    gp_Display->PlaceSurface(0, 0, sp_CreditScreen, sp_Screen);
    break;
case 6: // Continue Window
    // Update the current level
    UpdateGameLevel(gp_Input->GameLevel);
    // Keep the continue image on the screen
    gp_Display->PlaceSurface(145, 180, sp_ContinueScreen, sp_Screen);
    break;
case 7: // High Score Input Window
    // Keep the score input image on the screen
    gp_Display->PlaceSurface(0, 0, sp_ScoreInputScreen, sp_Screen);
    // Update the block score fonts
    UpdateScreenFonts();
    break;
case 8: // Game Over Window
    // Update the current level
    UpdateGameLevel(gp_Input->GameLevel);
    // Display Lives
    gp_Display->DisplayLives();
    // Display Paddle
    gp_Input->DisplayPaddle();
    // Keep the game over screen on the window
    gp_Display->PlaceSurface(145, 180, sp_GameOverScreen, sp_Screen);
    break;
case 9: // Select Mode Window
    // Keep the load screen on the window
    gp_Display->PlaceSurface(0, 0, sp_LoadScreen, sp_Screen);
    // Dim the alpha
    SDL_SetAlpha(sp_LoadScreen, SDL_SRCALPHA, 100);
    // Keep the select image on the window
    gp_Display->PlaceSurface(238, 190, sp_SelectMode, sp_Screen);
    break;
case 10: // Practice Mode Selection Window
    // Keep the practice image on the window
    gp_Display->PlaceSurface(0, 0, sp_PracticeScreen, sp_Screen);
    break;
case 11: // Challenge Mode Prepare Window
    // Keep the challenge image on the window
    gp_Display->PlaceSurface(0, 0, sp_ChallengeScreen, sp_Screen);
    break;
}
}

```

Source File: Input.cpp**Description:** This handles the laser weapon power up. A maximum of 40 streams can be active at a time

```
void Input::UpdateLiveFire(bool fire)
{
    // If the power up has been activated
    if(gp_Gem->PowerUpActive[9] == true)
    {
        // Has the player fired a laser?
        if(fire == true)
        {
            // If so, Fire away!
            FireForEffect = rand()%40;
            gp_Display->Bullets[FireForEffect].Active = true;
            gp_Display->Bullets[FireForEffect].gotX = true;
        }
        for(int i = 0; i != 40; i++)
        {
            // If any lasers are active, Update them
            if(gp_Display->Bullets[i].Active == true)
            {
                // Start Location from Paddle
                if(gp_Display->Bullets[i].gotX == true)
                {
                    gp_Display->Bullets[i].X = gp_Display->PaddleOne.X+50;
                    gp_Display->Bullets[i].Y = gp_Display->PaddleOne.Y;
                    gp_Display->Bullets[i].gotX = false;
                }
                // Place Graphic
                gp_Display->PlaceSurface(gp_Display->Bullets[i].X, gp_Display->Bullets[i].Y, sp_Bullet, sp_Screen);

                // Continue Movement Upward
                gp_Display->Bullets[i].Y -= 5;
            }
            // If the game is started, Cycle through each block to check for collision
            if(GameStarted == true){
                for(int j = 0; j < 121; j++)
                {
                    for(int p = 0; p != 40; p++)
                    {
                        b_Left    = gp_Display->Bullets[p].X;
                        b_Right   = gp_Display->Bullets[p].X + 5;
                        b_Top     = gp_Display->Bullets[p].Y;
                        b_Bottom  = gp_Display->Bullets[p].Y + 15;

                        if(b_Bottom >= gp_Display->Blocks[i].Y && b_Right >=
gp_Display->Blocks[j].X && b_Left <= gp_Display->Blocks[j].X + gp_Display->Blocks[j].WIDTH && b_Top
<= gp_Display->Blocks[j].Y + gp_Display->Blocks[j].HEIGHT)
                        {
                            if(gp_Display->Blocks[j].alive == true)
                            {
                                GameBoard[gp_Display->Blocks[j].xLoc][gp_Display->Blocks[j].yLoc] = -1;
                                gp_Display->Blocks[j].alive = false;
                                gp_Display->Blocks[j].ShowGem = true;
                                gp_Display->Bullets[p].Active = false;
                                gp_Display->Blocks[j].Tick += 1;
                                gp_Game->TotalScore += gp_Display->Blocks[j].PointValue;
                                gp_Display->UpdateScores(gp_Display->Blocks[j].Color);

                                gp_Display->Bullets[p].X = 295;
                                gp_Display->Bullets[p].Y = 460;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Source File: Input.cpp**Description:** Collision Detection between the ball, paddle, and blocks.

```
bool Input::CollisionDetect()
{
    // Get the Updated Coords for the Game Ball
    b_Left   = gp_Display->GameBall.X;
    b_Right  = gp_Display->GameBall.X + gp_Display->GameBall.WIDTH;
    b_Top    = gp_Display->GameBall.Y;
    b_Bottom = gp_Display->GameBall.Y + gp_Display->GameBall.HEIGHT;
    // Get the Updated Coords for the Game Paddle
    p_Left   = gp_Display->PaddleOne.X;
    p_Right  = gp_Display->PaddleOne.X + gp_Display->PaddleOne.WIDTH;
    p_Top    = gp_Display->PaddleOne.Y;
    p_Bottom = gp_Display->PaddleOne.Y + gp_Display->PaddleOne.HEIGHT;

    // Get the Coords for the Game Blocks
    for(int i = 0; i < 124; i++)
    {
        gp_Display->Blocks[i].Bl_Left   = gp_Display->Blocks[i].X;
        gp_Display->Blocks[i].Bl_Right  = gp_Display->Blocks[i].X + gp_Display->
>Blocks[i].WIDTH;
        gp_Display->Blocks[i].Bl_Top    = gp_Display->Blocks[i].Y;
        gp_Display->Blocks[i].Bl_Bottom = gp_Display->Blocks[i].Y + gp_Display->
>Blocks[i].HEIGHT;
    }
    // Check each block for collision with the game Ball
    for(int i = 0; i < 121; i++)
    {
        // If a Block is Hit
        if(b_Top <= gp_Display->Blocks[i].Bl_Bottom && b_Right > gp_Display->Blocks[i].Bl_Left
&& b_Left < gp_Display->Blocks[i].Bl_Right && b_Bottom >= gp_Display->Blocks[i].Bl_Top)
        {
            // And the Block is Alive
            if(gp_Display->Blocks[i].alive == true)
            {
                // Does the player have the Bomb Power Up?
                if(gp_Display->GameBall.BombActive == true)
                {
                    // If so, we need to randomly destroy several blocks
                    for(int i = 0; i != 10; i++)
                    {
                        bomb = rand()%121;
                        if(gp_Display->Blocks[bomb].alive == true)
                        {
                            if(bomb < 121)
                            {
                                Mix_PlayChannel(-1, mp_Explosion, 0);
                                GameBoard[gp_Display->
>Blocks[bomb].xLoc][gp_Display->Blocks[bomb].yLoc] = -1;
                                gp_Display->Blocks[bomb].alive = false;
                                gp_Display->Blocks[bomb].Tick += 1;
                                gp_Display->Blocks[bomb].ShowGem = true;
                                gp_Game->TotalScore += gp_Display->
>Blocks[i].PointValue;
                                gp_Display->UpdateScores(gp_Display->
>Blocks[i].Color);
                            }
                        }
                    }
                    gp_Display->GameBall.BombActive = false;
                }
            }
        }
    }
    // Play Sound
    Mix_PlayChannel(-1, mp_BallHit, 0);
    // Turn the block off from the main game board
    GameBoard[gp_Display->Blocks[i].xLoc][gp_Display->Blocks[i].yLoc] = -1;
```

```

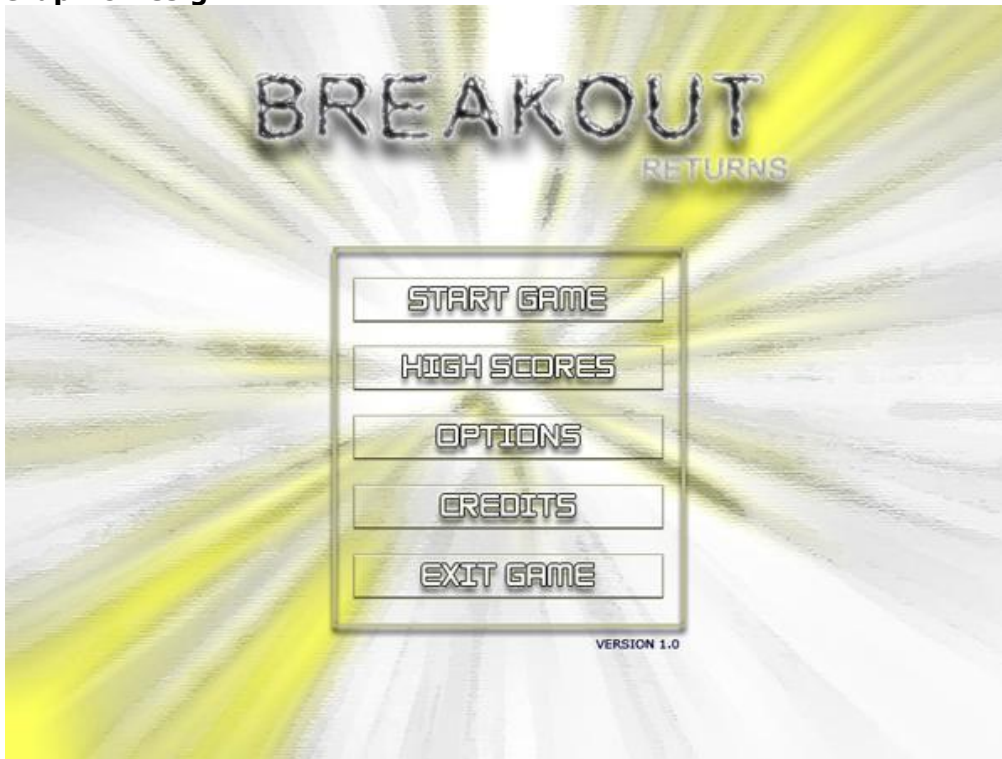
        // Turn this block off, we hit it!
        gp_Display->Blocks[i].alive = false;
        // If this block had a gem, it will now be shown
        gp_Display->Blocks[i].ShowGem = true;
        // Increment the destroyed count
        gp_Display->Blocks[i].Tick += 1;
        // Add points to Total Score
        gp_Game->TotalScore += gp_Display->Blocks[i].PointValue;
        // Update scores for block color destroyed
        gp_Display->UpdateScores(gp_Display->Blocks[i].Color);
        return true;
    }
}

// If the ball hit's the paddle, react to specific movement of the paddle
if(b_Bottom >= p_Top && b_Right >= p_Left && b_Left <= p_Right && b_Top <= p_Bottom)
{
    if(gp_Display->PaddleOne.Direction == 0) // Stationary
    {
        Mix_PlayChannel(-1, mp_BallHitPad, 0);
        gp_Display->GameBall.yVel -= gp_Display->GameBall.yVel-gp_Display-
>GameBall.MaxAccel;
        gp_Display->GameBall.xVel += gp_Display->GameBall.xVel;
        if(b_Right <= p_Left+3)
            gp_Display->GameBall.xVel -= gp_Display->GameBall.xVel-2+gp_Display-
>GameBall.MaxAccel;
        if(b_Left >= p_Right-3)
            gp_Display->GameBall.xVel -= gp_Display->GameBall.xVel+2+gp_Display-
>GameBall.MaxAccel;
    }
    if(gp_Display->PaddleOne.Direction == 1) // Left
    {
        Mix_PlayChannel(-1, mp_BallHitPad, 0);
        gp_Display->GameBall.yVel -= gp_Display->GameBall.yVel-gp_Display-
>GameBall.MaxAccel;
        gp_Display->GameBall.xVel -= 3;
        gp_Display->GameBall.yVel = gp_Display->GameBall.yVel-((gp_Display-
>PaddleOne.xVel+20+gp_Display->GameBall.MaxAccel)/2);
    }
    if(gp_Display->PaddleOne.Direction == 2) // Right
    {
        Mix_PlayChannel(-1, mp_BallHitPad, 0);
        gp_Display->GameBall.yVel -= gp_Display->GameBall.yVel;
        gp_Display->GameBall.xVel -= 3+gp_Display->GameBall.MaxAccel;
        gp_Display->GameBall.yVel = gp_Display->GameBall.yVel-(gp_Display-
>PaddleOne.xVel/2+gp_Display->GameBall.MaxAccel);
    }
}
// If power up 2 is turned off (speed up ball), slow the ball down
if(gp_Gem->PowerUpActive[2] == false)
    gp_Display->GameBall.MaxAccel = 0;

return false;
}

```

Graphic Design



BREAKOUT

RETURNS

NEW HIGH SCORE!!!

ENTER INITIALS

PXL

| | | |
|------|-----|--------------|
| 4000 | ACE | BEST #1 GAME |
| 3000 | NEO | BEST #2 GAME |
| 2822 | CDE | BEST #3 GAME |
| 2000 | GLO | BEST #4 GAME |
| 179 | | BEST #5 GAME |

A B C D E F G

H I J K L M N

O P Q R S T U

V W X Y Z ↵

▶ DONE

Breakout Returns

New Game

BLOCKS DESTROYED

| | |
|--|----|
| | 11 |
| | 23 |
| | 22 |
| | 17 |
| | 29 |
| | 15 |
| | 26 |
| | 24 |
| | 1 |

1454

Exit

BREAKOUT

RETURNS

POWER UP GUIDE



LOSE 1 LIFE



PRODUCES SPEED BURSTS UPON IMPACT WITH THE PADDLE



SHRINK PADDLE



GROW PADDLE



WILL FREEZE BALL IN PLACE, THEN FIRE IT IN AN UNKNOWN DIRECTION



GAIN 1 LIFE



COVERS GAMEPLAY AREA WITH A FOG OF CLOUDS



RANDOMLY DESTROYS UP TO 10 BLOCKS



ALLOWS PADDLE TO FIRE A LASER



PLUS 30 POINTS



MINUS 50 POINTS



PLUS 10 SECONDS



MINUS 15 SECONDS

CHALLENGE
MODE SPECIFIC