```c
// on Linux compile with:   clang pongy.c -lm -lSDL2 -lGLU -lGL -o pongy
// on Windows compile with: clang pongy.c -l SDL2 -l SDL2main -l Shell32 -l glu32 -l opengl32 -o pongy.exe
-Xlinker /subsystem:console
#define SDL_MAIN_HANDLED
// This is the main SDL include file
#include <SDL2/SDL.h>
#ifdef _WIN32
#include <windows.h>
#endif
// include OpenGL
#include <GL/gl.h>
#include <GL/glu.h>
// include relevant C standard libraries
#include <stdio.h>
#include <stdlib.h>

/* data structures */
typedef struct paddle
{
   double x;
   double y;
   double width;
   double height;
   char side; /* left: 1; right -1 */
   double speed;
} paddle;

typedef struct ball
{
   double x;
   double y;
   double radius;
   double speedX;
   double speedY;
} ball;

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 disappears, changes on ballxBrick
   //-1 is a negative bonus, 0 normal, 1 is positive bonus
}brick;
int userInput()
{
   int choice;
   printf("Type the desired level and press enter: ");
   scanf("%d", &choice);
   //scanf_s is what worked on my machine but is causing issues here
   printf("level : %d \n", choice);
   return choice;
}
void initializePaddle(paddle* p, double x, double y, double w, double h, char sd, double sp)
{
   p->x = x;
   p->y = y;
   p->width = w;
   p->height = h;
   p->side = sd;
   p->speed = sp;
}
void initializeBall(ball* b, double x, double y, double r, double sx, double sy)
{
   b->x = x;
   b->y = y;
   b->radius = r;
   b->speedX = sx;
   b->speedY = sy;
}
```

```
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)
}

void updatePaddle(paddle* p, double f, int d, int w)
{

   p->x+=p->speed*f*(double)d; /* calculate next position */

   /* ensure the paddle does not go beyond the boundaries */
   if(p->x+(p->width/2.0)>=(double)(w/2)) p->x=(double)(w/2)-(p->width/2.0);
   if(p->x-(p->width/2.0)<=(double)(w/-2)) p->x=(double)(w/-2)+(p->width/2.0);
}

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)+2) && (b->y - b->radius >= p->y - (p->height /
2.0)-2))//checks height of paddle
   {
       if ((b->x + b->radius +5>= p->x - (p->width/ 2.0)) && (b->x - b->radius -5<= p->x +(p->width /
2.0)))//checks height of paddle
       {
           return 1;
       }
   }
   return 0;
}

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;
}
void updateBall(ball* b, double f, paddle* p1,brick*br, int w, int h,int* lives,int*go)
{
   /* collision detection & resolution with scene 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 counter
       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, p1)))
    {
        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;
}
void ballBrick(ball*b, brick*br, int *points,int *lives, paddle* p1)//contact with brick action
{
    if ((ballXbricktop(b, br))&& br->hitCount>0)
    {
        b->y -= 6;//prevents weirdness of it going through paddle
        b->speedY *= -1.0;//reverses ball
        br->hitCount -= 1;//reduces hit count
        *points += 1;//when ball broken adds points
        if (*points == 5)
        {
            *points += 1;//points become 6 and range of 105
            *lives += 1;
            b->speedY *= 0.8;//dec. speed by 20%
            p1->width += 5;//inc size of paddle
            p1->height += 10;
        }
        if (*points == 30)
        {
            *points += 1;//points become 31 and range of 106
            *lives += 1;
            p1->height += 5;

        }
        if (*points == 50)
        {
            *points += 1;//points become 51 and range of 107
            *lives += 1;
            b->speedY *= 2;//increase speed by 20%
        }
        if (*points == 100)
        {
            *points += 1;//points become 101 and range of 108
            *lives += 1;
            b->speedY *= 0.8;//dec. speed by 20%
            p1->height *= 1.5;//inc size of paddle
        }
    }

}

void drawBall(ball* b)
{
    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(b->x, b->y, 0.0); /* move the ball to its correct position */

   glBegin(GL_QUADS); /* draw ball */
   glColor3f(1.0f, 0.6f, 0.0f);//orange ball
   glVertex3d(b->radius / -2.0, b->radius / 2.0, 0.0);//creates square ball based on radius with 4 vertexes
   glVertex3d(b->radius / 2.0, b->radius / 2.0, 0.0);
   glVertex3d(b->radius / 2.0, b->radius / -2.0, 0.0);
   glVertex3d(b->radius / -2.0, b->radius / -2.0, 0.0);
   glEnd();

   glPopMatrix(); /* restore previous modelview matrix */
   glMatrixMode(matrixmode); /* set the previous matrix mode */
}

void drawPaddle(paddle* p)
{
   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(p->x, p->y, 0.0); /* move the paddle to its correct position */

   glBegin(GL_QUADS); /* draw paddle */
   glColor3f(0.984f, 0.91f, 0.737f);//orange paddle
   glVertex3d(p->height / -2.0, p->width / 2.0, 0.0);//calculate the 4 vertexes depending on width and height
established
   glVertex3d(p->height / 2.0, p->width / 2.0, 0.0);
   glVertex3d(p->height / 2.0, p->width / -2.0, 0.0);
   glVertex3d(p->height / -2.0, p->width / -2.0, 0.0);
   glEnd();
   glPopMatrix(); /* restore previous modelview matrix */
   glMatrixMode(matrixmode); /* set the previous matrix mode */
}
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 */
}

void render(ball* b, paddle* p1)
{
   /* Start by clearing the framebuffer (what was drawn before) */
   glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

   /* Set the scene transformations */
   glMatrixMode(GL_MODELVIEW); /* set the modelview matrix */
   glLoadIdentity(); /* Set it to the identity (no transformations) */

   /* draw the objects */
   drawPaddle(p1);
   drawBall(b);
}

int main(int argc, char* argv[])
{
   int winPosX = 150;
   int winPosY = 50;
   int winWidth = 390;//divisible by 13, each brick width 30
   int winHeight = 600;

   int go;
   int lives = 3;//--life if touches bottom
   int p1dir = 0;
   int score = 0;//tracks points
   int level;
   printf("Breakout E-scape ---[=====>\n\n\n"); fflush(stdout);//asks for user to type 1,2,or 3 to select a
level
   printf("Type one of the numbers  for level selection excliding brackets [1]  [2]  [3]
");fflush(stdout);//asks for user to type 1,2,or 3 to select a level
   level = userInput();
   Uint32 timer;

   paddle p1;
   ball myB;
   brick b1;
   brick bricks[13][8] = {0};
   char winTitle[62] = "Breakout E-scape ---[=====>";


   /* This is our initialisation phase

      SDL_Init is the main initialisation function for SDL
      It takes a 'flag' parameter which we use to tell SDL what systems we are going to use
      Here, we want to initialise everything, so we give it the flag for this.
      This function also returns an error value if something goes wrong,
      so we can put this straight in an 'if' statement to check and exit if need be */
   if (SDL_Init(SDL_INIT_VIDEO) != 0) /* alternative: SDL_INIT_EVERYTHING */
   {
       /* Something went very wrong in the initialisation, all we can do is exit */
       perror("Whoops! Something went very wrong, cannot initialise SDL :( - ");
       perror(SDL_GetError());
       return -1;
   }

   /* Now we have got SDL initialised, we are ready to create an OpenGL window! */
   SDL_Window* window = SDL_CreateWindow(winTitle,  /* The first parameter is the window title */
       winPosX, winPosY,
       winWidth, winHeight,
       SDL_WINDOW_OPENGL | SDL_WINDOW_SHOWN); /* ensure that OpenGL gets enabled here */
   /* The last parameter lets us specify a number of options.
      Here, we tell SDL that we want the window to be shown and that it can be resized.
      You can learn more about SDL_CreateWindow here:
https://wiki.libsdl.org/SDL_CreateWindow?highlight=%28\bCategoryVideo\b%29|%28CategoryEnum%29|%28CategoryStruct
%29
      The flags you can pass in for the last parameter are listed here: https://wiki.libsdl.org/SDL_WindowFlags
```

```
      The SDL_CreateWindow function returns an SDL_Window.
      This is a structure which contains all the data about our window (size, position, etc).
      We will also need this when we want to draw things to the window.
      This is therefore quite important we do not lose it! */


   if (window == NULL)
   {
       printf("Window could not be created! SDL Error: %s\n", SDL_GetError());
       exit(EXIT_FAILURE); /* crash out if there has been an error */
   }

   /* Use OpenGL 1.5 compatibility */
   SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 1);
   SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 5);
   SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK, SDL_GL_CONTEXT_PROFILE_COMPATIBILITY);
   SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16); /* set up Z-Buffer */
   SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1); /* enable double buffering */


   /* SDL_GLContext is the OpenGL rendering context - this is the equivalent to the SDL_Renderer when drawing
pixels to the window */
   SDL_GLContext context = SDL_GL_CreateContext(window);
   if (context == NULL)
   {
       printf("OpenGL Rendering Context could not be created! SDL Error: %s\n", SDL_GetError());
       exit(EXIT_FAILURE); /* crash out if there has been an error */
   }

   /* Set up the parts of the scene that will stay the same for every frame. */

   glFrontFace(GL_CCW);     /* Enforce counter clockwise face ordering (to determine front and back side) */
   glEnable(GL_NORMALIZE);
   glShadeModel(GL_FLAT); /* enable flat shading - Gouraud shading would be GL_SMOOTH */
   glEnable(GL_DEPTH_TEST);

   /* Set the clear (background) colour */
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

   /* Set up the camera/viewing volume (projection matrix) */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-1.0 * (GLdouble)(winWidth / 2), (GLdouble)(winWidth / 2), -1.0 * (GLdouble)(winHeight / 2),
(GLdouble)(winHeight / 2));

   glViewport(0, 0, winWidth, winHeight);

   /* initialize the timer */
   timer = SDL_GetTicks();

   /* initialize objects */

   initializeBall(&myB, 0.0, -200.0, 6.0, 100.0, 100.0);
   initializePaddle(&p1, 0.0, -250.0, 4, 40, 1, 100.0);//check top of the paddle

   initializeBrick(&b1, 0,0, 1, 1);//one brick initalized as a starter needed for below
   //was running into issues doing it straight from the array, unfortunetly

       if (level == 3)
       {
           for (int i = 0; i < 8; i++) {//makes array of bricks

               for (int j = 0; j < 13; j++) {
                   b1.x = (-180) + (j * 30);
                   b1.y = (220) + (i * 10);
                   b1.hitCount = i+1;//each level bricks inc 1 more hit needed
                   b1.color = ((i % 2)+2);
                   bricks[j][i] = b1;
               }
           }
```

```
       }
       else if (level == 2)
       {
           for (int i = 0; i < 8; i++) {//makes array of bricks

               for (int j = 0; j < 13; j++) {
                   b1.x = (-180) + (j * 30);
                   b1.y = (220) + (i * 10);
                   b1.hitCount = (int)(i/2)+1;//slow increase
                   b1.color = ((i % 2)+1);//alternates colors
                   bricks[j][i] = b1;
               }
           }
       }
       else// level 1
       {
           for (int i = 0; i < 8; i++) {//makes array of bricks

               for (int j = 0; j < 13; j++) {
                   b1.x = (-180) + (j * 30);
                   b1.y = (220) + (i * 10);
                   b1.hitCount = 1;
                   b1.color = (i % 2);//alternates colors
                   bricks[j][i] = b1;
               }
           }
       }


   /* We are now preparing for our main loop.
      This loop will keep going round until we exit from our program by changing the int 'go' to the value
false (0).
      This loop is an important concept and forms the basis of most SDL programs you will be writing.
      Within this loop we generally do the following things:
        * Check for input from the user (and do something about it!)
        * Update our graphics (if necessary, e.g. for animation)
        * Draw our graphics
   */
   go = 1;
   while (go)
   {
       /* Here we are going to check for any input events.
          Basically when you press the keyboard or move the mouse, the parameters are stored as something
called an 'event' or 'message'.
          SDL has a queue of events. We need to check for each event and then do something about it (called
'event handling').
          The SDL_Event is the data type for the event. */
       SDL_Event incomingEvent;

       double fraction = 0.0;

       /* SDL_PollEvent will check if there is an event in the queue - this is the program's 'message pump'.
          If there is nothing in the queue it will not sit and wait around for an event to come along (there
are functions which do this,
          and that can be useful too!). Instead for an empty queue it will simply return 'false' (0).
          If there is an event, the function will return 'true' (!=0) and it will fill the 'incomingEvent' we
have given it as a parameter with the event data */
       while (SDL_PollEvent(&incomingEvent))
       {
           /* If we get in here, we have an event and need to figure out what to do with it.
              For now, we will just use a switch based on the event's type */
           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 window.
                  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:
                p1dir = -1;
                break;
            case SDLK_RIGHT:
                p1dir = 1;
                break;
            case SDLK_a:
                p1dir = -1;
                break;
            case SDLK_d:
                p1dir = 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:
                p1dir = 0;
                break;
            case SDLK_LEFT:
                p1dir = 0;
                break;
            case SDLK_a:
            case SDLK_d:
                p1dir = 0;
                break;
            case SDLK_ESCAPE: go = 0;
                break;
            }
            break;

            /* If you want to learn more about event handling and different SDL event types, see:
              https://wiki.libsdl.org/SDL_Event
              and also: https://wiki.libsdl.org/SDL_EventType */
        }
    }

    /* update timer */
    {
        Uint32 old = timer;
        timer = SDL_GetTicks();
        fraction = (double)(timer - old) / 1000.0; /* calculate the frametime by finding the difference in
ms from the last update/frame and divide by 1000 to get to the fraction of a second */
    }

    /* update positions */
    updatePaddle(&p1, fraction, p1dir, 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
        }
    }


    /* 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]);
               }
           }
       }

       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!\n\n\n "); 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;
}
```

MAKE FILE

```
Breakout: Breakout.c
   clang Breakout.c -lm -lSDL2 -lGLU -lGL -o Breakout

   ./Breakout
```