20.9.2022

Blocker Game with Arduino Uno

Final Project for Microcontroller and Applications

Gurleen Kour

HOCHSCHULE BREMEN

CONTENT

SHORT SUMMARY

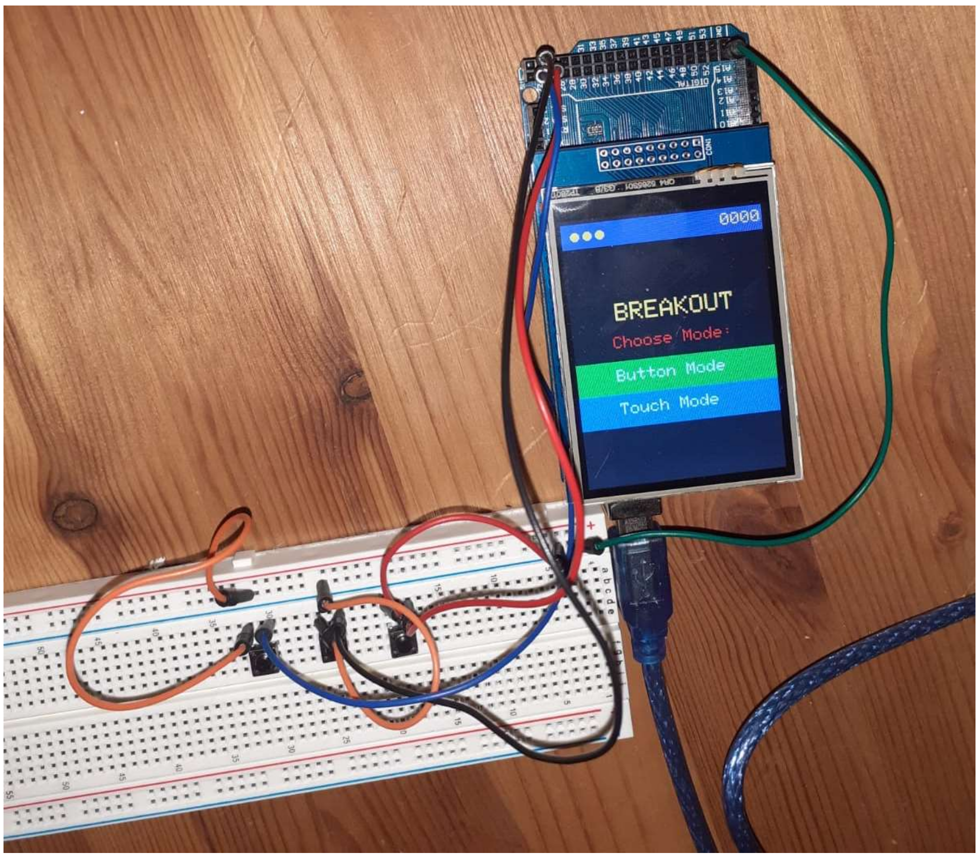
This Breakout game has 16 levels with different wall patterns of bricks. The wall patterns for each level can have up to eight rows, with each two rows a different colour and a different point gain per brick hit. The player bounces the ball on the paddle by moving the paddle with the buttons (button mode) or by pressing the right or left side of the screen (touch mode). By pressing the home button, the game is reset, and the home screen appears.

The objective of the game is to knock down as many bricks as possible by using the walls and/or the paddle below to ricochet the ball against the bricks and eliminate them. If the player's paddle misses the ball's rebound, they will lose a life. If all lives are lost, the game is lost.

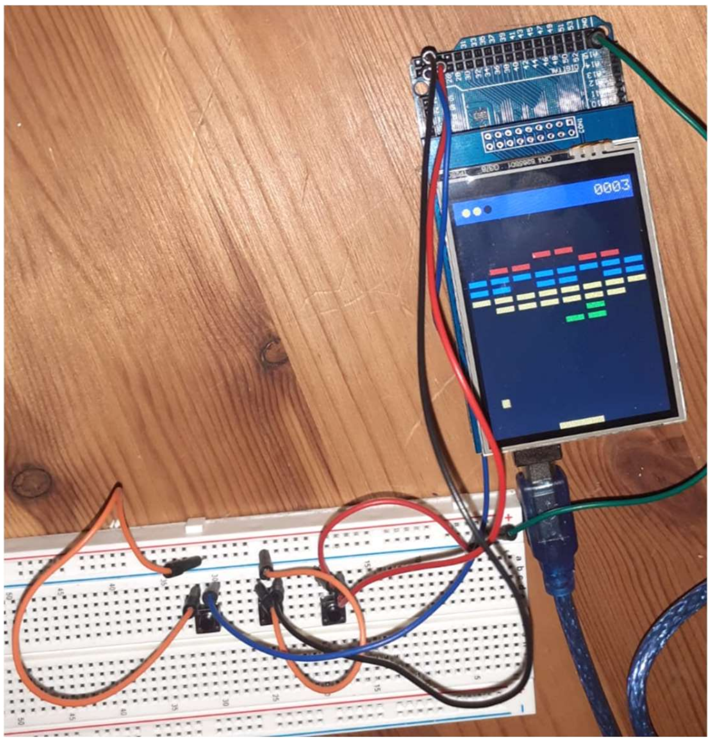
Row	Points gained per brick hit
First 2 rows	1
3rd and 4th row	3
5th and 6th row	5
Last 2 rows	7

PHOTOS OF THE PROJECT

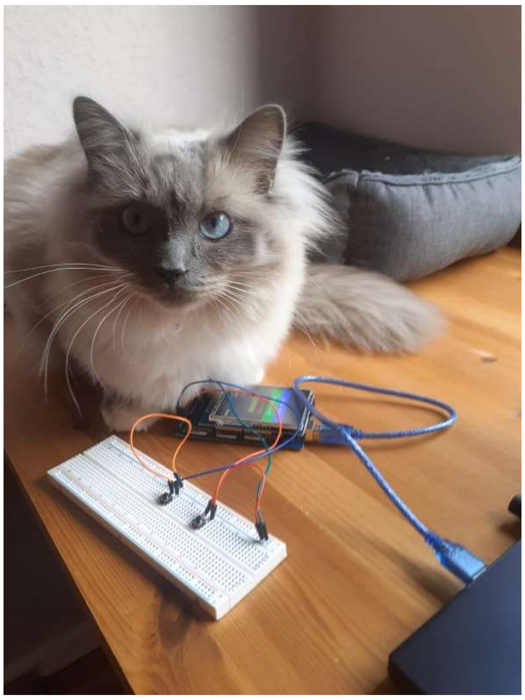
Home screen

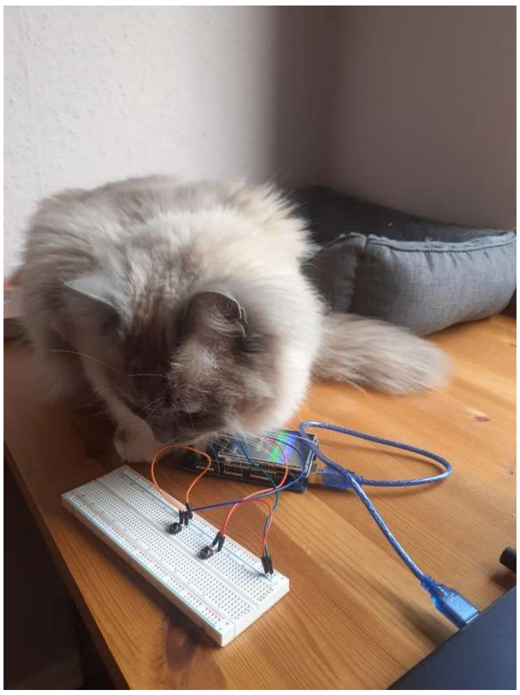


First Level



My assistant (he tried to chew the jumper cables)





HARDWARE

ESSENTIALS

Arduino Mega 2560 Board

2,8" TFT Touchscreen Display compatible with Arduino Mega 2560, 240x320 pixel resolution

3 buttons (right, left and home)

OPTIONAL

Optional materials will be required for Nice-To-Have Features.

Loudspeaker

External battery box

Case (to be 3D printed)

TARGET SPECIFICATION

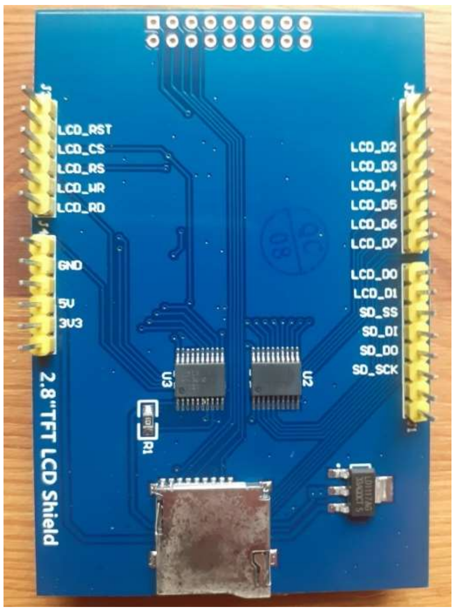
Feature	Priority	Status
There are at least three levels.	Must	✓
Each level has a unique wall pattern which can have up to eight rows and eight columns.	Must	✓
Each two rows of bricks in every wall pattern have a different colour. The colours are the same for every level of the game.	Must	✓
For every brick destroyed the player gains a number of points which depends on the brick's colour. The colour-points gained pairings are the same for every level.	Must	✓
The paddle will be moved with two buttons: one will move the paddle forward, and one will move the paddle backward.	Must	✓
The paddle will also be able to be controlled by touch. A touch on the right side of the screen will move the paddle forward, and a touch on the left side of the screen will move the paddle backwards.	Nice to have	✓
The paddle size and ball size vary by level.	Nice to have	✓

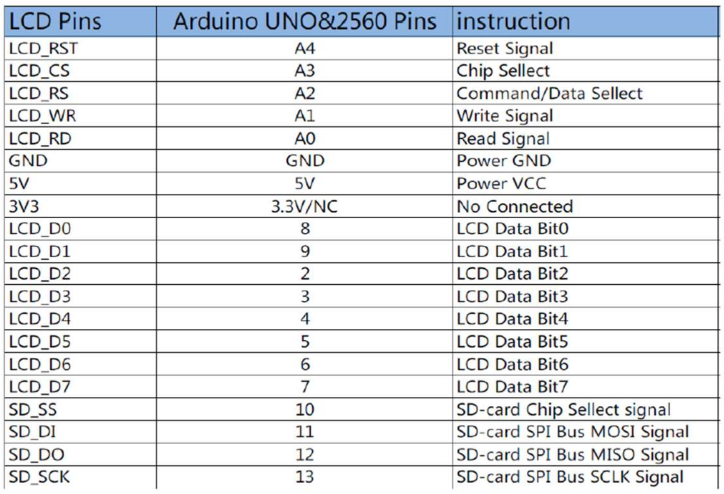
There are lives. If the player's paddle misses the ball's rebound, they will lose a life.	Must	✓
If all lives are lost, the player loses the game.	Must	✓
When all bricks of a level are destroyed, the level is completed.	Must	✓
When a level is completed, the next level appears on the display and is playable.	Must	✓
If all levels are completed, the player wins the game.	Must	✓
A “Game lost” screen appears on the display when the game is lost.	Must	✓
A “Game won” screen appears on the display when the game is won.	Must	✓
The total ball speed increases each time it hits a brick.	Nice to have	✘
Console will function with external battery.	Nice to have	✘
A loudspeaker will be built in to play music while playing.	Nice to have	✘
Console will be contained in a plastic 3D printed case.	Nice to have	✘

ASSEMBLY

The shield is assembled on the board with the below described connections.

Pin correspondence between LCD and Arduino





LCD Pins	Arduino UNO&2560 Pins	instruction
LCD_RST	A4	Reset Signal
LCD_CS	A3	Chip Sellect
LCD_RS	A2	Command/Data Sellect
LCD_WR	A1	Write Signal
LCD_RD	A0	Read Signal
GND	GND	Power GND
5V	5V	Power VCC
3V3	3.3V/NC	No Connected
LCD_D0	8	LCD Data Bit0
LCD_D1	9	LCD Data Bit1
LCD_D2	2	LCD Data Bit2
LCD_D3	3	LCD Data Bit3
LCD_D4	4	LCD Data Bit4
LCD_D5	5	LCD Data Bit5
LCD_D6	6	LCD Data Bit6
LCD_D7	7	LCD Data Bit7
SD_SS	10	SD-card Chip Sellect signal
SD_DI	11	SD-card SPI Bus MOSI Signal
SD_DO	12	SD-card SPI Bus MISO Signal
SD_SCK	13	SD-card SPI Bus SCLK Signal

Buttons connections with Arduino

Buttons	Left Button	Home Button	Right Button
Pins	22	23	24

SOFTWARE

THE DISPLAY DRIVERS

ELEGOO, the manufacturer of the display used, delivered some excellent drivers for the display available which are based off the Adafruit ILI9341 library. Initially I tried to use the Adafruit drivers and adapt them for my own display because they have more functions, which I thought could be helpful. I couldn't adapt them though, so in the end I decided to use the ELEGOO drivers.

RESOURCES

I used some code I found online, which can be described as "spaghetti code", that is, very difficult to understand and extend.

From it I took:

- Some of the data structures
- Most game attributes for each level, corresponding to the data structure LevelConstants
  (I.e. ball_size, player_width, gamefield_top, rows, columns, brick_gap, lives, wall[8] – wall pattern)
- The colours and general graphic style

The code taken over was heavily refactored to increase code readability and improved (for example unnecessary variables were removed, methods were optimized according to clean code techniques, and many bugs were detected and removed). The game logic was entirely written by me.

These statements can easily be confirmed by looking at the source code online (see sources).

THE ARDUINO SKETCH

The Arduino Sketch is divided into the following sections:

VARIABLES DEFINITIONS

Here the variables and objects necessary for the game are defined. Constant variables are in a separate .cpp file.

SETUP

In the setup() method, game objects are initialized and the newGame() method is called.

LOOP

In the loop, first is the user input evaluated, then the positions of all game objects updated. After that, the level completion is evaluated by checking if all bricks were removed, in which case the game will progress to the next level. Last but not least, the number of remaining lives is checked. If it is zero, the game will display a “You Lost” message and return to the home screen.

GAME LOGIC

This section contains the following methods:

- newGame()
- updateLives()
- updateBallPosition()
- updatePlayerPosition()
- checkBallCollisions() and its helper methods checkBrickCollision(), checkCornerCollision(), checkBorderCollision(), checkPaddleCollision()
- ballHitsBottom() and its helper method resetBall()
- noBricks()

newGame()

This method sets up the home screen, where the player chooses between two game modes:

- Button Mode: player moves the paddle with buttons
- Touch Mode: player moves the paddle by touching the right or left side of the screen

After the player makes a choice, the first level is painted on the screen.

GUI

This section contains methods to paint the game objects on the screen:

- drawText()
- clearDialog()
- touchToStart()
- showGameOver()
- showGameWon()
- updateScore()
- setupWall() and its helper methods isBrickIn(), setBrick() unsetBrick()
- drawBrick()
- hitBrick() and its helper method removeBrick()
- drawPlayer()
- removeOldPlayer()
- drawBall()
- removeOldBall()
- cleanGamefieldBottom()

INPUT HANDLING

Contains the method readTouch, which handles the input generated from the touchscreen.

SETUP

Contains two methods, initTft(), which adjusts the display's settings, and setupState(), which initializes the variable game_state.

game_State contains the information on the current status of the game.

SKETCH

This Sketch uses display drivers provided by ELEGOO and the library InputDebounce. The software is not finalized. There are a couple of bugs which will be solved until the presentation.

```
/*
 * @Author Gurleen Kour
 */

#include <InputDebounce.h>
#include "Elegoo_GFX.h"    // Core graphics library
#include "Elegoo_TFTLCD.h" // Hardware-specific library
#include "TouchScreen.h"
#include "colours.h"
```

```
#include "data_containers.h"
#include "pin_definitions.h"
#include "game_constants.cpp"

// Buttons
InputDebounce button_left;
int button_left_on;
InputDebounce button_home;
int button_home_on;
InputDebounce button_right;
int button_right_on;

unsigned long now;

Elegoo_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
TouchScreen touchscreen = TouchScreen(XP, YP, XM, YM, 300);
TFT_Size tft_size;
static LevelConstants* current_level_constants;
static game_state_type game_state;

int level;
long millisLastBallMove = 0;
int16_t millisMoveBall = 20;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  initTft(tft);
  tft_size = {0, 0, tft.width(), tft.height()};
  current_level_constants = &all_levels_constants[0];
  button_left.setup(BUTTON_LEFT, BUTTON_DEBOUNCE_DELAY,
InputDebounce::PIM_INT_PULL_UP_RES);
  button_home.setup(BUTTON_HOME, BUTTON_DEBOUNCE_DELAY,
InputDebounce::PIM_INT_PULL_UP_RES);
  button_right.setup(BUTTON_RIGHT, BUTTON_DEBOUNCE_DELAY,
InputDebounce::PIM_INT_PULL_UP_RES);
  newGame(current_level_constants, &game_state, tft);
}

void loop(void) {

  // Button Mode
  if(game_state.game_mode == 1){

    button_left_on = button_left.process(now);   // poll button state, return
continuous on-time [ms] if pressed (debounced)
    button_home_on = button_home.process(now);
    button_right_on = button_right.process(now);

    if(button_home_on) {
      // reset game
      game_state.score = 0;
      level = 0;
      newGame(current_level_constants, &game_state, tft);
    } else if (button_left_on){
      updatePlayerPosition(-1, &game_state);
    } else if (button_right_on){
      updatePlayerPosition(1, &game_state);
    }
  }

  // Touch Mode
  if(game_state.game_mode == 2){
    int16_t x_screen;
    int16_t y_screen;
    bool screen_touched = readTouch(current_level_constants, &game_state,
&x_screen, &y_screen);

    if(screen_touched){
      if(x_screen > tft.width() / 2){
        updatePlayerPosition(-1, &game_state);
      } else {
        updatePlayerPosition(1, &game_state);
      }
    }
  }

  drawPlayer(current_level_constants, &game_state);
  removeOldPlayer(current_level_constants, &game_state);

  updateBallPosition(&game_state);
  checkBallCollisions(current_level_constants, &game_state, game_state.x_ball,
game_state.y_ball);

  // draw ball in new position
  removeOldBall(game_state.x_ball, game_state.y_ball, game_state.x_ball_old,
           game_state.y_ball_old, current_level_constants->ball_size);
  drawBall(game_state.x_ball, game_state.y_ball,
          current_level_constants->ball_size);

  //increaseBallSpeed();

  // if no bricks go to next level
  if (noBricks(current_level_constants, &game_state) && level < LEVELS_NUMBER)
{
    level++;
    newGame(&all_levels_constants[level], &game_state, tft);
  } else if ( game_state.lives_left <= 0) {
    showGameOver(current_level_constants, &game_state);
    game_state.score = 0;
    level = 0;
    delay(1000);
    newGame(current_level_constants, &game_state, tft);
  }
}

//  * * * * * * * * * * * * * * *
//  * GAME LOGIC                *
//  * * * * * * * * * * * * * * *

void newGame(LevelConstants* newGame, game_state_type*
game_state,  Elegoo_TFTLCD &tft) {
  current_level_constants = newGame;
  setupState(current_level_constants, game_state, tft);

  touchToStart(current_level_constants, game_state);

  clearDialog(tft_size);
  updateLives(current_level_constants->lives, game_state->lives_left);
  updateScore(game_state->score);
  setupWall(current_level_constants, game_state);
}

void updateLives(int lives, int lives_left) {

  for (int i = 0; i < lives; i++) {
    tft.fillCircle((1 + i) * 15, 15, 5, BLACK);
  }

  for (int i = 0; i < lives_left; i++) {
    tft.fillCircle((1 + i) * 15, 15, 5, YELLOW);
  }
}

void updateBallPosition(game_state_type* game_state){
  // If not delayed, the loop executes very often and the ball moves way too
quickly
  if((millis() - millisLastBallMove) >= millisMoveBall){
    game_state->x_ball += game_state->ball_speed_x;
    game_state->y_ball += game_state->ball_speed_y;
    millisLastBallMove = millis();
  }
}

void updatePlayerPosition(int16_t direction, game_state_type* game_state) {
  // Move player in direction touched
  if (direction < 0) {
    game_state->x_player -= 10;
  } else {
    game_state->x_player += 10;
  }

  // Player has reached the edge
  if (game_state->x_player >= tft.width() - current_level_constants-
>player_width)
    game_state->x_player = tft.width() - current_level_constants-
>player_width;
  if (game_state->x_player < 0)
    game_state->x_player = 0;
}

void checkBrickCollision(LevelConstants* current_level_constants,
game_state_type* game_state, uint16_t x_ball, uint16_t y_ball) {
  int x1 = x_ball + current_level_constants->ball_size; // x of left corners
of ball
  int y1 = y_ball + current_level_constants->ball_size; // y of bottom corners
of ball

  int collisions = 0;
  collisions += checkCornerCollision(current_level_constants, game_state,
x_ball, y_ball);   // top left corner of ball
  collisions += checkCornerCollision(current_level_constants, game_state, x1,
y1);           // bottom right corner of ball
  collisions += checkCornerCollision(current_level_constants, game_state,
x_ball, y1);       // bottom left corner of ball
  collisions += checkCornerCollision(current_level_constants, game_state, x1,
y_ball);       // top right corner of ball

  if (collisions > 0 ) {
    game_state->ball_speed_y = (-1 * game_state->ball_speed_y);

    if ((((x_ball % game_state->brick_width) == 0)  && ( game_state-
>ball_speed_x < 0 ))     // ball hits right edge of brick
        || (((x1 % game_state->brick_width) == 0)  && ( game_state-
>ball_speed_x > 0 )) ) {  // ball hits left edge of brick
      game_state->ball_speed_x = (-1 * game_state->ball_speed_x);
    }
  }
}

int checkCornerCollision(LevelConstants*
current_level_constants,  game_state_type* game_state, uint16_t x_ball,
uint16_t y_ball) {
  if ((y_ball > game_state->wall_top) && (y_ball < game_state->wall_bottom))
{              // check if ball is in wall area

    // Determine which brick should be in the position of the ball
    int row = ( y_ball - game_state->wall_top) / game_state->brick_height;
    int column = (x_ball / game_state->brick_width);
    if (isBrickIn(game_state->wallState, column, row)) {
      hitBrick(game_state, column, row);
      return true;
    }
  }
  return false;
}

void checkBorderCollision(LevelConstants*
current_level_constants,  game_state_type* game_state, uint16_t x_ball,
uint16_t y_ball) {
  // check wall collision
  if (x_ball + current_level_constants->ball_size >=  tft.width())
{         // ball hits right wall
    game_state->ball_speed_x = -abs(game_state->ball_speed_x);
  }
  if (x_ball <= 0)
{                                                         // ball hits left
wall
    game_state->ball_speed_x = abs(game_state->ball_speed_x);
  }
  if (y_ball <= SCORE_SIZE )
{                                               // ball hits top border of
gamefield
    game_state->ball_speed_y = -abs(game_state->ball_speed_y);
  }
  if((y_ball + current_level_constants->ball_size)  >=  game_state-
>gamefield_bottom){  // ball hits bottom border of gamefield
    ballHitsBottom(current_level_constants, game_state);
  }
}

void checkPaddleCollision(LevelConstants* current_level_constants,
game_state_type* game_state, uint16_t x_ball, uint16_t y_ball){
  // if guard - helps avoiding nested ifs :D
  if((y_ball + current_level_constants->ball_size) <= (game_state-
>gamefield_bottom - player_height)
  || (y_ball + current_level_constants->ball_size) >= game_state-
>gamefield_bottom
  || (x_ball + current_level_constants->ball_size) <= game_state->x_player
  || x_ball >= (game_state->x_player + current_level_constants-
>player_width)){
    return;
  }

  game_state->ball_speed_y = -abs(game_state->ball_speed_y);

}
```

```
void checkBallCollisions(LevelConstants* current_level_constants,
game_state_type* game_state, uint16_t x_ball, uint16_t y_ball) {
  checkBrickCollision(current_level_constants, game_state, x_ball, y_ball);
  checkBorderCollision(current_level_constants, game_state, x_ball, y_ball);
  checkPaddleCollision(current_level_constants, game_state, x_ball, y_ball);
}

void ballHitsBottom(LevelConstants* current_level_constants, game_state_type*
game_state) {
  game_state->lives_left--;
  updateLives(current_level_constants->lives, game_state->lives_left);
  delay(500);
  resetBall(current_level_constants, game_state);
  cleanGamefieldBottom(current_level_constants);
}

boolean noBricks(LevelConstants* current_level_constants, game_state_type *
game_state) {
  for (int row = 0; row < current_level_constants->rows ; row++) {
    if (game_state->wallState[row])
      return false;
  }
  return true;
}

void resetBall(LevelConstants* current_level_constants, game_state_type*
game_state){
  game_state->x_ball_old = game_state->x_ball;
  game_state->x_ball = 50;
  game_state->y_ball_old = game_state->y_ball;
  game_state->y_ball = 200;
  game_state->ball_speed_x = initial_ball_speed_x;
  game_state->ball_speed_y = initial_ball_speed_y;
}

//  * * * * * * * * * * * * * * *
//  * GUI                       *
//  * * * * * * * * * * * * * * *

void drawText(const uint16_t x, const uint16_t y, const char* string, const
uint16_t fontsize,
                     const uint16_t colour) {
  tft.setTextSize(fontsize);
  tft.setCursor(x, y);
  tft.setTextColor(colour);
  tft.print(string);
}

//Clear the screen to the default backgrounds
void clearDialog(TFT_Size tft_size) {
  tft.fillRect(tft_size.x, tft_size.y, tft_size.width,
tft_size.height,  BLACK);
  tft.fillRect(tft_size.x, tft_size.y, tft_size.width, SCORE_SIZE,
PRIMARY_DARK_COLOR);
}

void touchToStart(LevelConstants* current_level_constants, game_state_type*
game_state) {
  // setup screen
  clearDialog(tft_size);
  updateLives(current_level_constants->lives, game_state->lives_left);
  updateScore(game_state->score);

  drawText(0, 100, "   BREAKOUT", 3, YELLOW);
  drawText(0, 140, "    Choose Mode:", 2, RED);
  tft.fillRect(0, 170, tft.width(), 40, GREEN);
  drawText(0, 180, "    Button Mode", 2, WHITE);
  tft.fillRect(0, 210, tft.width(), 40, BLUE);
  drawText(0, 220, "    Touch Mode", 2, WHITE);

  // get touch y
  int16_t x_screen;
  int16_t y_screen;
  while (!readTouch(current_level_constants, game_state, &x_screen,
&y_screen)) {}

  // select correct mode
  if(y_screen > 170 && y_screen < 210){
    game_state->game_mode = 1;
  } else if(y_screen > 210 && y_screen < 250){
    game_state->game_mode = 2;
  }
}

void showGameOver(LevelConstants* current_level_constants, game_state_type*
game_state) {
  tft.fillScreen(BLACK);
  drawText(30, 30, "GAME OVER", 3, YELLOW);
  drawText(40, 60, "YOU LOST", 3, YELLOW);
}

void showGameWon(LevelConstants* current_level_constants, game_state_type*
game_state) {
  tft.fillScreen(BLACK);
  drawText(30, 30, "WELL DONE!", 3, YELLOW);
  drawText(40, 60, "YOU WON", 3, YELLOW);
}


void updateScore(int score) {
  char buffer[5];
  snprintf(buffer, sizeof(buffer), scoreFormat, score);
  tft.fillRect(tft.width() - 50, 0, 100, SCORE_SIZE, PRIMARY_DARK_COLOR);
  drawText(tft.width() - 50, 6, buffer, 2, YELLOW);
}

void setupWall(LevelConstants* current_level_constants, game_state_type *
game_state) {

  int colors[] = {RED, RED, BLUE, BLUE,  YELLOW, YELLOW, GREEN, GREEN};

  game_state->wall_top = current_level_constants->gamefield_top + 40;
  game_state->wall_bottom = game_state->wall_top + current_level_constants-
>rows * game_state->brick_height;

  for (int row = 0; row < current_level_constants->rows; row++) {
    for (int column = 0; column < current_level_constants->columns; column++)
{
      if (isBrickIn(current_level_constants->wall, column, row)) {
        setBrick(game_state->wallState, column, row);
        drawBrick(game_state, column, row, colors[row]);
      }
    }
  }
}

boolean isBrickIn(int wall[], uint8_t column, uint8_t row) {
  return wall[row] &  BIT_MASK[column];
}

void setBrick(int wall[], uint8_t column, uint8_t row) {
  wall[row] = wall[row] | BIT_MASK[column];
}

void unsetBrick(int wall[], uint8_t column, uint8_t row) {
  wall[row] = wall[row] & ~BIT_MASK[column];
}

void drawBrick(game_state_type* game_state, int xBrick, int yBrickRow,
uint16_t colour) {
  int16_t x = (game_state->brick_width * xBrick) + current_level_constants-
>brick_gap;
  int16_t y = game_state->wall_top + (game_state->brick_height * yBrickRow) +
current_level_constants->brick_gap;
  int16_t width = game_state->brick_width - current_level_constants->brick_gap
* 2;
  int16_t height = game_state->brick_height - current_level_constants-
>brick_gap * 2;
  tft.fillRect(x, y, width, height, colour);
}
```

```
void removeBrick(game_state_type* game_state, int xBrick, int yBrickRow) {
  int16_t x = (game_state->brick_width * xBrick);
  int16_t y = game_state->wall_top + (game_state->brick_height * yBrickRow);
  int16_t width = game_state->brick_width;
  int16_t height = game_state->brick_height;
  tft.fillRect(x, y, width, height, BLACK);
}

void hitBrick(game_state_type* game_state, int xBrick, int yBrickRow) {
  game_state->score += pointsForRow[yBrickRow];
  drawBrick(game_state, xBrick, yBrickRow, WHITE);
  delay(16);
  drawBrick(game_state, xBrick, yBrickRow, BLUE);
  delay(8);
  removeBrick(game_state, xBrick, yBrickRow);
  unsetBrick(game_state->wallState, xBrick, yBrickRow);
  updateScore(game_state->score);
}

void drawPlayer(LevelConstants* current_level_constants, game_state_type*
game_state) {
  int16_t x = game_state->x_player;
  int16_t y = game_state->gamefield_bottom - player_height;
  int16_t width = current_level_constants->player_width;
  int16_t height = player_height;
  tft.fillRect(x, y, width, height, YELLOW);
}

void removeOldPlayer(LevelConstants* current_level_constants, game_state_type*
game_state) {
  if (game_state->x_player != game_state->x_player_old) {
    int16_t x = game_state->x_player;
    int16_t y = game_state->gamefield_bottom - player_height;
    int16_t width = abs(game_state->x_player - game_state->x_player_old);
    int16_t height = player_height;

    if (game_state->x_player < game_state->x_player_old) {
      x = game_state->x_player + current_level_constants->player_width;
    } else {
      x = game_state->x_player_old;
    }

    tft.fillRect(x, y, width, height, BLACK);
  }

  // store old position to remove old pixels
  game_state->x_player_old = game_state->x_player;
}
```

```
void drawBall(int x, int y, int ball_size) {
  tft.fillRect(x , y, ball_size, ball_size, YELLOW);
}

void removeOldBall(int x, int y, int xold, int yold, int ball_size){
  if (xold <= x && yold <= y) {
    tft.fillRect(xold , yold, ball_size, y - yold, BLACK);
    tft.fillRect(xold , yold, x - xold, ball_size, BLACK);
  } else if (xold >= x && yold >= y) {
    tft.fillRect(x + ball_size , yold, xold - x, ball_size, BLACK);
    tft.fillRect(xold , y + ball_size, ball_size, yold - y, BLACK);
  } else if (xold <= x && yold >= y) {
    tft.fillRect(xold , yold, x - xold, ball_size, BLACK);
    tft.fillRect(xold , y + ball_size, ball_size, yold - y, BLACK);
  } else if (xold >= x && yold <= y) {
    tft.fillRect(xold , yold, ball_size, y - yold, BLACK);
    tft.fillRect(x + ball_size, yold, xold - x, ball_size, BLACK);
  }

  // store old position to remove old pixels
  game_state.x_ball_old = game_state.x_ball;
  game_state.y_ball_old = game_state.y_ball;
}

void cleanGamefieldBottom(LevelConstants* current_level_constants){ // WORK IN
PROGRESS - DOESN'T WORK YET
  int16_t x = 0;
  int16_t y = tft.height() - current_level_constants->ball_size + 5;
  int16_t width = tft.width();
  int16_t height = current_level_constants->ball_size + 5;
  tft.fillRect(x, y, width, height,BLACK);
}

//  * * * * * * * * * * * * * * *
//  * INPUT HANDLING            *
//  * * * * * * * * * * * * * * *

bool readTouch(LevelConstants* current_level_constants, game_state_type*
game_state, int16_t* x_screen, int16_t* y_screen) {
  TSPoint tp = touchscreen.getPoint();   //tp.x, tp.y are ADC values

  // if sharing pins, you'll need to fix the directions of the touchscreen
pins
  pinMode(XM, OUTPUT);
  pinMode(YP, OUTPUT);
  // we have some minimum pressure we consider 'valid'
  // pressure of 0 means no pressing!

  if (tp.z > MINPRESSURE && tp.z < MAXPRESSURE) {
    *x_screen = map(tp.x, TOUCHSCREEN_RIGHT, TOUCHSCREEN_LEFT, 0,
tft.width());
    *y_screen = map(tp.y, TOUCHSCREEN_TOP, TOUCHSCREEN_BOTTOM, 320, 0);
    return true;
  }
  return false;
}



//  * * * * * * * * * * * * * * *
//  * GAME SETUP                *
//  * * * * * * * * * * * * * * *

void initTft(Elegoo_TFTLCD& tft) {
    Serial.begin(9600);

    tft.reset();

    uint16_t identifier = tft.readID();
    if (identifier == 0x0101)
    {
        identifier = 0x9341;
        Serial.println(F("Found 0x9341 LCD driver"));
    }
    else if (identifier == 0x1111)
    {
        identifier = 0x9328;
        Serial.println(F("Found 0x9328 LCD driver"));
    }
    else {
        Serial.print(F("Unknown LCD driver chip: "));
        Serial.println(identifier, HEX);
        identifier = 0x9328;

    }
    tft.begin(identifier);
    tft.setRotation(0);
}


void setupState(LevelConstants* current_level_constants, game_state_type*
game_state, Elegoo_TFTLCD& tft) {
    game_state->gamefield_bottom = tft.height();
    game_state->brick_width = tft.width() / current_level_constants-
>columns;  // it might not work
    game_state->brick_height = tft.height() / 24;  // it might not work

    for (int row = 0; row < current_level_constants->rows; row++) {
        game_state->wallState[row] = 0;
    }

    game_state->x_player = tft.width() / 2 - current_level_constants-
>player_width / 2;
    game_state->lives_left = current_level_constants->lives;
    game_state->x_ball = 50;
    game_state->x_ball_old = 50;
    game_state->y_ball = 200;
    game_state->y_ball_old = 200;
    game_state->ball_speed_x = initial_ball_speed_x;
    game_state->ball_speed_y = initial_ball_speed_y;
}
```

THE C++ SOURCE FILES

pin_definition.cpp

```
#ifndef DEFINITIONS
#define DEFINITIONS

// Buttons
#define BUTTON_LEFT 22
#define BUTTON_HOME 23
#define BUTTON_RIGHT 24

// Control pins for the LCD
#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0
#define LCD_RESET A4 // Can alternately just connect to Arduino's reset pin

#define LOWFLASH (defined(__AVR_ATmega328P__) && defined(MCUFRIEND_KBV_H_))

// Touch screen calibration

#define XP 8
#define XM A2
#define YP A3
#define YM 9 //240x320 ID=0x9341

#endif
```

colours.h

```
#ifndef COLOURS
#define COLOURS

// Colors with 16-bit values
#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF
#define PRIMARY_COLOR 0x4A11
#define PRIMARY_LIGHT_COLOR 0x7A17
#define PRIMARY_DARK_COLOR 0x4016
#define PRIMARY_TEXT_COLOR 0x7FFF

#endif
```

game_constants.cpp

```
#ifndef GAME_CONSTANTS
#define GAME_CONSTANTS

#include "data_containers.h"

static const int16_t TOUCHSCREEN_LEFT = 122;
static const int16_t TOUCHSCREEN_RIGHT = 929;
static const int16_t TOUCHSCREEN_TOP = 77;
static const int16_t TOUCHSCREEN_BOTTOM = 884;

// Touch screen pressure threshold
static const int16_t MINPRESSURE = 40;
static const int16_t MAXPRESSURE = 1000;

static const int16_t BUTTON_DEBOUNCE_DELAY = 20;   // [ms]

static char scoreFormat[] = "%04d";

static const uint8_t BIT_MASK[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
0x80};
static uint8_t pointsForRow[] = {7, 7, 5, 5, 3, 3 , 1, 1};

static const int16_t player_height = 8;
static const int16_t initial_ball_speed_x = 1;
static const int16_t initial_ball_speed_y = -1;

static LevelConstants all_levels_constants[LEVELS_NUMBER] =
  // ball_size, player_width, player_height, gamefield_top, rows, columns,
brick_gap, lives, wall[8], initial_ball_speed_x, initial_ball_speed_y
{
  { 10, 60, 40, 8, 8, 3, 3,  {0x18, 0x66, 0xFF, 0xDB, 0xFF, 0x7E, 0x24, 0x3C}},
  { 10, 50, 40, 8, 8, 3, 3,  {0xFF, 0x99, 0xFF, 0xE7, 0xBD, 0xDB, 0xE7, 0xFF}},
  { 10, 50, 40, 8, 8, 3, 3,  {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55}},
  { 8,  50, 40, 8, 8, 3, 3,  {0xFF, 0xC3, 0xC3, 0xC3, 0xC3, 0xC3, 0xC3, 0xFF}},
  { 10, 40, 40, 8, 8, 3, 3,  {0xFF, 0xAA, 0xAA, 0xFF, 0xFF, 0xAA, 0xAA, 0xFF}},
  { 10, 40, 40, 8, 8, 3, 3,  {0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA}},
  { 12, 64, 60, 4, 2, 3, 4,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 12, 60, 60, 5, 3, 3, 4,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 10, 56, 30, 6, 4, 3, 4,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 10, 52, 30, 7, 5, 3, 4,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 8,  48, 30, 8, 6, 3, 3,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 8,  44, 30, 8, 7, 3, 3,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 8,  40, 30, 8, 8, 3, 3,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 8,  36, 40, 8, 8, 3, 3,  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}},
  { 8,  36, 40, 8, 8, 3, 3,  {0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA}}
};

#endif
```

data_containers.h

```
#ifndef DATA_CONTAINERS
#define DATA_CONTAINERS

#define LEVELS_NUMBER 16
#define SCORE_SIZE 30

#include <stdint.h>

typedef struct TFT_Size {
  int16_t x, y, width, height;
} TFT_Size;

// Contains current data of the level played
typedef struct game_state_type {
  int game_mode;   // 1 - button mode, 2 - touch mode
  uint16_t x_ball;
  uint16_t y_ball;
  uint16_t x_ball_old;
  uint16_t y_ball_old;
  int ball_speed_x;
  int ball_speed_y;
  int x_player;
  int x_player_old;
  int wallState[8];
  int score;
  int lives_left;
  int gamefield_bottom;
  int wall_top;
  int wall_bottom;
  int brick_height;
  int brick_width;
};

typedef struct LevelConstants {
  int ball_size;
  int player_width;
  int gamefield_top;
  int rows;
  int columns;
  int brick_gap;
  int lives;
  int wall[LEVELS_NUMBER];
} LevelConstants;

#endif
```

PROBLEMS

A known problem of the Arduino IDE is that it often rebuilds the sketch instead of making a clean build. This mainly created problems when I decided to split the sketch into multiple files for better code readability and because it makes working with the code easier, because I got error messages saying that I defined a method multiple times. This is due to the Arduino IDE using cached libraries and cached core build.

I really wanted to find a solution, because in my opinion, the sketch is too cluttered and most code in it should have been split into classes. As I couldn't find a solution, I had to settle and write all necessary methods in the .ino file.

POSSIBLE TEST CASE

The home screen is visible. The player presses chooses if he wants to play in touch mode or button mode. He then starts the game by pressing any button (button mode) or touching the screen (touch mode). The first level appears, which has rows with different number of bricks. The player moves the paddle by either pressing the right and left buttons (button mode) or by touching the right or left side of the screen (touch mode). The player can bounce the ball with the paddle and try to hit bricks. If he can't bounce the ball and it falls to the bottom of the screen, he loses one life. When he hits a brick, he gains points. When no bricks are left, the next level appears. If the player loses all his lives, he loses the game and a "You Lost" screen appears. If he wins all levels, a "You Won" screen appears. In both cases, the player can return to the home screen by pressing the home button.

SOURCES

Display Drivers – ELEGOO

Online Blocker Code - https://create.arduino.cc/projecthub/javagoza/arduino-touch-breakout-game-eda113?ref=tag&ref_id=games&offset=66