Breakout tutorial (Beginner)

Target: 10x20 PixelGrid (Adafruit_NeoPixel + PixelGridCore) on Arduino Controls:

- LED data: pin 2
- Buttons (pins 3,4,9,10): serve/restart only
- Joystick left/right (pins 6,7): move paddle (auto-repeat while held)
- Joystick up/down (pins 5,8): unused

Features:

- Bricks start filled in rows 0..7 only
- Every 15 seconds: bricks shift down by 1, new row generated at top (single uniform colour per row, wheel cycles)
- No drawn walls
- One miss = game over
- Score shown on 6-digit panel (LCD_Panel digits)

What you will build

A working Breakout game split into small files:

- `Breakout.ino` (main loop)
- `Pins.h` (all pins, sizes, constants)
- `Input.h/.cpp` (debounce + joystick repeat + “serve pressed”)
- `Game.h/.cpp` (all state + physics + brick logic)
- `Render.h/.cpp` (all drawing + score digits)

You can still compile it as one “sketch” because Arduino compiles all `.ino/.cpp` in the folder together.

---

1) Create the folder

In the Games folder:

- Create a new folder called `Breakout`.
- In the same folder, add the files:

```
Breakout.ino
Pins.h
Input.h
Input.cpp
Game.h
Game.cpp
Render.h
Render.cpp
```

---

2) Pins.h (hardware + constants)

Create `Pins.h` and add:

```
#pragma once
#include <Arduino.h>

#define PIXEL_BUFFER_SIZE 300
#define PIN_LED 2

#define PIN_BTN1 3
#define PIN_BTN2 4
#define PIN_BTN3 9
#define PIN_BTN4 10

#define PIN_JOY_UP    5
#define PIN_JOY_LEFT  6
#define PIN_JOY_RIGHT 7
#define PIN_JOY_DOWN  8

static const uint8_t PREVIEW_ROWS = 0;
static const uint8_t PLAY_H = 20;
static const uint8_t W = 10;
static const uint8_t MATRIX_ROWS = PREVIEW_ROWS + PLAY_H;

static const uint16_t DEBOUNCE_MS = 18;
static const uint16_t FRAME_MS = 16;

static const uint8_t PADDLE_Y = PLAY_H - 1;
static const uint8_t PADDLE_W = 3;

static const uint16_t BALL_STEP_MS = 110;
static const uint16_t BALL_STEP_MIN_MS = 55;
static const uint16_t BALL_SPEEDUP_EVERY = 10;

static const uint16_t BRICK_DROP_MS = 15000;

static const uint8_t BRICK_TOP = 0;
static const uint8_t BRICK_BOTTOM = PLAY_H - 2;
static const uint8_t BRICK_H = (uint8_t)(BRICK_BOTTOM + 1);

static const uint8_t INITIAL_FILLED_ROWS = 8;

static const uint8_t COLOR_STEP = 9;

static const uint16_t MOVE_REPEAT_START_MS = 160;
static const uint16_t MOVE_REPEAT_MS = 65;
```

What this file does:

- It centralizes everything you might want to tweak later without hunting through the game.

3) Input module (debounce + joystick repeat)

Input.h

Create `Input.h`:

```
#pragma once
#include <Arduino.h>
#include "Pins.h"

struct Btn {
  uint8_t pin = 0;
  bool stable = false;
  bool prevStable = false;
  bool lastRaw = false;
  uint32_t lastChange = 0;

  void begin(uint8_t p);
  void update();
  bool pressedEdge() const;
  bool releasedEdge() const;
  void latch();
};

extern Btn btn1, btn2, btn3, btn4;
extern Btn joyL, joyR, joyU_unused, joyD_unused;

extern uint32_t tMoveL;
extern uint32_t tMoveR;
extern bool moveLRepeating;
extern bool moveRRepeating;

void Input_begin();
void Input_update();
void Input_latch();

bool Input_servePressedEdge();
int8_t Input_paddleStepFromJoystickRepeat(uint32_t now);
```

Input.cpp

Create `Input.cpp`:

```
#include "Input.h"

Btn btn1, btn2, btn3, btn4;
Btn joyL, joyR, joyU_unused, joyD_unused;

uint32_t tMoveL = 0;
```

```
uint32_t tMoveR = 0;
bool moveLRepeating = false;
bool moveRRepeating = false;

void Btn::begin(uint8_t p) {
  pin = p;
  pinMode(pin, INPUT_PULLUP);
  stable = false;
  prevStable = false;
  lastRaw = false;
  lastChange = millis();
}

void Btn::update() {
  bool raw = (digitalRead(pin) == LOW);
  uint32_t now = millis();
  if (raw != lastRaw) {
    lastRaw = raw;
    lastChange = now;
    return;
  }
  if (now - lastChange >= DEBOUNCE_MS) stable = raw;
}

bool Btn::pressedEdge() const { return stable && !prevStable; }
bool Btn::releasedEdge() const { return !stable && prevStable; }
void Btn::latch() { prevStable = stable; }

void Input_begin() {
  btn1.begin(PIN_BTN1);
  btn2.begin(PIN_BTN2);
  btn3.begin(PIN_BTN3);
  btn4.begin(PIN_BTN4);

  joyU_unused.begin(PIN_JOY_UP);
  joyL.begin(PIN_JOY_LEFT);
  joyR.begin(PIN_JOY_RIGHT);
  joyD_unused.begin(PIN_JOY_DOWN);

  uint32_t now = millis();
  tMoveL = tMoveR = now;
  moveLRepeating = moveRRepeating = false;
}

void Input_update() {
  btn1.update(); btn2.update(); btn3.update(); btn4.update();
  joyL.update(); joyR.update();
  joyU_unused.update(); joyD_unused.update();
}

void Input_latch() {
  btn1.latch(); btn2.latch(); btn3.latch(); btn4.latch();
  joyL.latch(); joyR.latch(); joyU_unused.latch(); joyD_unused.latch();
}
```

```
bool Input_servePressedEdge() {
  return btn1.pressedEdge() || btn2.pressedEdge() || btn3.pressedEdge() ||
btn4.pressedEdge();
}

int8_t Input_paddleStepFromJoystickRepeat(uint32_t now) {
  // LEFT
  if (joyL.stable) {
    if (joyL.pressedEdge()) {
      tMoveL = now;
      moveLRepeating = false;
      return -1;
    } else {
      uint16_t waitMs = moveLRepeating ? MOVE_REPEAT_MS : MOVE_REPEAT_START_MS;
      if (now - tMoveL >= waitMs) {
        tMoveL = now;
        moveLRepeating = true;
        return -1;
      }
    }
  } else if (joyL.releasedEdge()) {
    moveLRepeating = false;
  }

  // RIGHT
  if (joyR.stable) {
    if (joyR.pressedEdge()) {
      tMoveR = now;
      moveRRepeating = false;
      return +1;
    } else {
      uint16_t waitMs = moveRRepeating ? MOVE_REPEAT_MS : MOVE_REPEAT_START_MS;
      if (now - tMoveR >= waitMs) {
        tMoveR = now;
        moveRRepeating = true;
        return +1;
      }
    }
  } else if (joyR.releasedEdge()) {
    moveRRepeating = false;
  }

  return 0;
}
```

What you learned here:

- Debounce makes buttons stable
- `pressedEdge()` gives you “just pressed this frame”
- Repeat logic turns “hold left” into repeated paddle steps

4) Game module (state + rules)

Game.h

Create `Game.h`:

```
#pragma once
#include <Arduino.h>
#include "Pins.h"

extern uint32_t bricksGrid[BRICK_H][W];

extern int8_t paddleX;
extern bool ballStuck;
extern int8_t ballX, ballY;
extern int8_t ballVX, ballVY;

extern uint32_t score;
extern bool gameOver;

extern uint16_t ballStepMs;
extern uint32_t tBall;
extern uint32_t tBrickDrop;
extern uint16_t bricksHit;

extern uint8_t wheelPos;

void Game_reset();
void Game_movePaddle(int8_t dx);
void Game_serveBall();

void Game_brickDropTick();
void Game_stepBallOnce();

bool Game_isOver();
```

Game.cpp

Create `Game.cpp`:

```
#include "Game.h"
#include "Render.h"

uint32_t bricksGrid[BRICK_H][W];

int8_t paddleX = 3;
bool ballStuck = true;
int8_t ballX = 0;
int8_t ballY = 0;
int8_t ballVX = 1;
```

```
int8_t ballVY = -1;

uint32_t score = 0;
bool gameOver = false;

uint16_t ballStepMs = BALL_STEP_MS;
uint32_t tBall = 0;
uint32_t tBrickDrop = 0;
uint16_t bricksHit = 0;

uint8_t wheelPos = 0;

static void clearBricks() {
  for (uint8_t y = 0; y < BRICK_H; ++y)
    for (uint8_t x = 0; x < W; ++x)
      bricksGrid[y][x] = 0;
}

static void generateBrickRowAt(uint8_t row, uint32_t c) {
  if (row >= BRICK_H) return;
  for (uint8_t x = 0; x < W; ++x) bricksGrid[row][x] = c;
}

static void fillInitialBricks() {
  clearBricks();
  for (uint8_t y = 0; y < INITIAL_FILLED_ROWS; ++y) {
    uint32_t c = Render_wheelColor(wheelPos);
    wheelPos = (uint8_t)(wheelPos + COLOR_STEP);
    generateBrickRowAt(y, c);
  }
}

static void clampPaddle() {
  if (paddleX < 0) paddleX = 0;
  if (paddleX > (int8_t)(W - PADDLE_W)) paddleX = (int8_t)(W - PADDLE_W);
}

static void resetBallOnPaddle() {
  ballStuck = true;
  ballVX = (random(0, 2) == 0) ? -1 : 1;
  ballVY = -1;

  ballX = (int8_t)(paddleX + (PADDLE_W / 2));
  ballY = (int8_t)(PADDLE_Y - 1);

  tBall = millis();
}

static bool hitBrickAt(int8_t x, int8_t y) {
  if (x < 0 || x >= (int8_t)W) return false;
  if (y < (int8_t)BRICK_TOP || y > (int8_t)BRICK_BOTTOM) return false;

  uint32_t v = bricksGrid[(uint8_t)y][(uint8_t)x];
  if (v == 0) return false;
```

```
  bricksGrid[(uint8_t)y][(uint8_t)x] = 0;

  score += 10UL;
  bricksHit++;

  if (bricksHit % BALL_SPEEDUP_EVERY == 0) {
    if (ballStepMs > BALL_STEP_MIN_MS) {
      ballStepMs = (uint16_t)max((int)BALL_STEP_MIN_MS, (int)ballStepMs - 6);
    }
  }

  Render_updateScoreDigits(score);
  return true;
}

void Game_movePaddle(int8_t dx) {
  paddleX += dx;
  clampPaddle();
  if (ballStuck) {
    ballX = (int8_t)(paddleX + (PADDLE_W / 2));
    ballY = (int8_t)(PADDLE_Y - 1);
  }
}

void Game_serveBall() {
  if (!ballStuck) return;
  ballStuck = false;
  tBall = millis();
}

void Game_reset() {
  score = 0;
  gameOver = false;

  paddleX = (W - PADDLE_W) / 2;

  ballStepMs = BALL_STEP_MS;
  bricksHit = 0;

  wheelPos = 0;
  fillInitialBricks();

  Render_updateScoreDigits(score);
  resetBallOnPaddle();

  tBrickDrop = millis();
}

void Game_brickDropTick() {
  for (uint8_t x = 0; x < W; ++x) {
    if (bricksGrid[BRICK_BOTTOM][x] != 0) {
      gameOver = true;
      return;
```

```
    }
  }

  for (int8_t y = (int8_t)BRICK_BOTTOM; y > (int8_t)BRICK_TOP; --y) {
    for (uint8_t x = 0; x < W; ++x) {
      bricksGrid[(uint8_t)y][x] = bricksGrid[(uint8_t)(y - 1)][x];
    }
  }

  uint32_t c = Render_wheelColor(wheelPos);
  wheelPos = (uint8_t)(wheelPos + COLOR_STEP);
  generateBrickRowAt(BRICK_TOP, c);
}

void Game_stepBallOnce() {
  if (ballStuck) return;

  int8_t nx = (int8_t)(ballX + ballVX);
  int8_t ny = (int8_t)(ballY + ballVY);

  // side bounce
  if (nx < 0) { nx = 0; ballVX = 1; }
  else if (nx >= (int8_t)W) { nx = (int8_t)(W - 1); ballVX = -1; }

  // top bounce
  if (ny < 0) { ny = 0; ballVY = 1; }

  // brick collision
  if (hitBrickAt(nx, ny)) {
    ballVY = (int8_t)(-ballVY);
    ny = (int8_t)(ballY + ballVY);
  } else {
    if (hitBrickAt(nx, ballY)) {
      ballVX = (int8_t)(-ballVX);
      nx = (int8_t)(ballX + ballVX);
    } else if (hitBrickAt(ballX, ny)) {
      ballVY = (int8_t)(-ballVY);
      ny = (int8_t)(ballY + ballVY);
    }
  }

  // paddle collision
  if (ny == (int8_t)PADDLE_Y && ballVY > 0) {
    if (nx >= paddleX && nx < paddleX + (int8_t)PADDLE_W) {
      ballVY = -1;
      ny = (int8_t)(PADDLE_Y - 1);

      int8_t center = (int8_t)(paddleX + (PADDLE_W / 2));
      if (nx < center) ballVX = -1;
      else if (nx > center) ballVX = 1;
      else {
        if (ballVX == 0) ballVX = (random(0, 2) == 0) ? -1 : 1;
      }
    }
```

```
    }

    // miss
    if (ny > (int8_t)PADDLE_Y) {
      gameOver = true;
      return;
    }

    ballX = nx;
    ballY = ny;
  }

  bool Game_isOver() { return gameOver; }
```

---

5) Render module (draw everything)

Render.h

Create `Render.h`:

```
#pragma once
#include <Arduino.h>
#include <Adafruit_NeoPixel.h>
#include <PixelGridCore.h>
#include "Pins.h"

extern Adafruit_NeoPixel strip;
extern Pixel_Grid* pixelGrid;
extern LCD_Panel* lcdPanel;

extern uint32_t PLAY_BG_COLOR_U32;
extern uint32_t PADDLE_COLOR_U32;
extern uint32_t BALL_COLOR_U32;

void Render_begin();
void Render_updateScoreDigits(uint32_t s);
uint32_t Render_wheelColor(uint8_t pos);

void Render_renderFrame();
```

Render.cpp

Create `Render.cpp`:

```
#include "Render.h"
#include "Game.h"

Adafruit_NeoPixel strip(PIXEL_BUFFER_SIZE, PIN_LED, NEO_GRB + NEO_KHZ800);
```

```
Pixel_Grid* pixelGrid = nullptr;
LCD_Panel* lcdPanel = nullptr;

uint32_t PLAY_BG_COLOR_U32;
uint32_t PADDLE_COLOR_U32;
uint32_t BALL_COLOR_U32;

static inline uint16_t playRowToPixelRow(uint8_t logicalRow) {
  return (uint16_t)(MATRIX_ROWS - 1 - (PREVIEW_ROWS + logicalRow));
}

void Render_updateScoreDigits(uint32_t s) {
  char tmp[7];
  char out[6];
  snprintf(tmp, sizeof(tmp), "%6lu", (unsigned long)s);
  for (uint8_t i = 0; i < 6; ++i) out[i] = tmp[i];
  lcdPanel->changeCharArray(out);
}

uint32_t Render_wheelColor(uint8_t pos) {
  pos = (uint8_t)(255 - pos);
  if (pos < 85) return strip.Color((uint8_t)(255 - pos * 3), 0, (uint8_t)(pos *
3));
  if (pos < 170) {
    pos = (uint8_t)(pos - 85);
    return strip.Color(0, (uint8_t)(pos * 3), (uint8_t)(255 - pos * 3));
  }
  pos = (uint8_t)(pos - 170);
  return strip.Color((uint8_t)(pos * 3), (uint8_t)(255 - pos * 3), 0);
}

static void drawBackground() {
  for (uint8_t pr = 0; pr < PLAY_H; ++pr) {
    uint16_t r = playRowToPixelRow(pr);
    for (uint8_t x = 0; x < W; ++x) pixelGrid->setGridCellColour(r, x,
PLAY_BG_COLOR_U32);
  }
}

static void drawBricks() {
  for (uint8_t y = BRICK_TOP; y <= BRICK_BOTTOM; ++y) {
    uint16_t r = playRowToPixelRow(y);
    for (uint8_t x = 0; x < W; ++x) {
      uint32_t c = bricksGrid[y][x];
      if (c) pixelGrid->setGridCellColour(r, x, c);
    }
  }
}

static void drawPaddle() {
  uint16_t r = playRowToPixelRow(PADDLE_Y);
  for (uint8_t i = 0; i < PADDLE_W; ++i) {
    uint8_t x = (uint8_t)(paddleX + i);
    if (x < W) pixelGrid->setGridCellColour(r, x, PADDLE_COLOR_U32);
```

```
  }
}

static void drawBall() {
  if (gameOver) return;
  if (ballX < 0 || ballX >= (int8_t)W) return;
  if (ballY < 0 || ballY >= (int8_t)PLAY_H) return;
  uint16_t r = playRowToPixelRow((uint8_t)ballY);
  pixelGrid->setGridCellColour(r, (uint16_t)ballX, BALL_COLOR_U32);
}

static void showAll() {
  lcdPanel->render();
  pixelGrid->render();
  strip.show();
}

void Render_renderFrame() {
  if (gameOver) {
    uint32_t c = strip.Color(30, 0, 0);
    for (uint8_t pr = 0; pr < PLAY_H; ++pr) {
      uint16_t r = playRowToPixelRow(pr);
      for (uint8_t x = 0; x < W; ++x) pixelGrid->setGridCellColour(r, x, c);
    }
    showAll();
    return;
  }

  drawBackground();
  drawBricks();
  drawPaddle();
  drawBall();
  showAll();
}

void Render_begin() {
  strip.begin();
  strip.show();

  PLAY_BG_COLOR_U32 = strip.Color(6, 6, 12);
  PADDLE_COLOR_U32  = strip.Color(220, 220, 220);
  BALL_COLOR_U32    = strip.Color(255, 255, 255);

  pixelGrid = new Pixel_Grid(&strip, 0, MATRIX_ROWS, W);
  lcdPanel  = new LCD_Panel(&strip, 214, 6, strip.Color(255, 255, 255));

  Render_updateScoreDigits(0);
  lcdPanel->render();
  drawBackground();
  pixelGrid->render();
  strip.show();
}
```

6) Breakout.ino (wires modules together)

Create `Breakout.ino`:

```
#include <Arduino.h>
#include <Adafruit_NeoPixel.h>
#include <PixelGridCore.h>

#include "Pins.h"
#include "Input.h"
#include "Game.h"
#include "Render.h"

void setup() {
  randomSeed(analogRead(A0));
  Serial.begin(115200);

  Render_begin();
  Input_begin();
  Game_reset();
}

void loop() {
  static uint32_t tFrame = 0;
  uint32_t now = millis();
  if (now - tFrame < FRAME_MS) return;
  tFrame = now;

  Input_update();

  bool servePressed = Input_servePressedEdge();

  if (Game_isOver()) {
    if (servePressed) Game_reset();
    Render_renderFrame();
    Input_latch();
    return;
  }

  if (now - tBrickDrop >= BRICK_DROP_MS) {
    tBrickDrop = now;
    Game_brickDropTick();
    if (Game_isOver()) {
      Render_renderFrame();
      Input_latch();
      return;
    }
  }

  int8_t step = Input_paddleStepFromJoystickRepeat(now);
  if (step != 0) Game_movePaddle(step);
```

```
  if (servePressed) Game_serveBall();

  if (!ballStuck) {
    if (now - tBall >= ballStepMs) {
      tBall = now;
      Game_stepBallOnce();
    }
  } else {
    ballX = (int8_t)(paddleX + (PADDLE_W / 2));
    ballY = (int8_t)(PADDLE_Y - 1);
  }

  Render_renderFrame();
  Input_latch();
}
```

Run it:

- Upload to Arduino
- Press any button to serve
- Use joystick left/right to move paddle
- Press button while game over to restart
