How to Make a Video Game in Java or Kotlin (2D Basics)

Game development involves several core concepts, regardless of the language:

- **Game Loop:** The heart of your game, constantly updating game logic and rendering graphics.
- **Game Objects:** Representing characters, enemies, items, and the environment.
- **Graphics Rendering:** Drawing shapes, images (sprites), and text to the screen.
- **Input Handling:** Responding to player actions (keyboard, mouse, touch).
- **Collision Detection:** Determining when game objects interact (e.g., player hitting an enemy).
- **Game State Management:** Keeping track of scores, levels, lives, etc.

Prerequisites

Before diving into game development, ensure you have a solid understanding of:

- **Basic Syntax:** Variables, data types, control flow (if/else, loops).
- **Object-Oriented Programming (OOP):** Classes, objects, inheritance, encapsulation, polymorphism. This is crucial for organizing your game code.
- **Collections:** Lists, Maps, Sets for managing groups of objects.

Tools You'll Need

- **Java Development Kit (JDK):** For Java.

- **IntelliJ IDEA or Android Studio:** Recommended IDEs (Integrated Development Environments) for both Java and Kotlin. They provide excellent code completion, debugging, and project management.

Approach 1: Using Standard Libraries (Java AWT/Swing or Kotlin/JavaFX)

This approach is good for understanding the fundamentals without relying on complex frameworks.

Java (using Swing/AWT for graphics):

1. **Set Up Your Project:**
    - Create a new Java project in IntelliJ IDEA.
    - You'll typically have a `JFrame` (the window), a `JPanel` (where you draw), and your game logic.
2. **The Game Loop:**
    - Implement a `Thread` or `Timer` that repeatedly calls an `update()` method (for game logic) and a `repaint()` or `draw()` method (for rendering).
    - The `update()` method will handle movement, physics, and game state changes.
    - The `paintComponent(Graphics g)` method of your `JPanel` is where you'll draw your game elements.
3. <!-- end list -->
4. Java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleGame extends JPanel implements ActionListener {

    private Timer timer;
    private int playerX = 50;
    private int playerY = 50;

    public SimpleGame() {
        setPreferredSize(new Dimension(800, 600));
        setBackground(Color.BLACK);
        setFocusable(true); // Allow JPanel to receive keyboard input

        timer = new Timer(15, this); // Fires every 15ms (approx 60 FPS)
        timer.start();
    }
```

```
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        // Drawing the player (a red rectangle)
        g.setColor(Color.RED);
        g.fillRect(playerX, playerY, 30, 30);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // This method is called by the timer
        updateGame();
        repaint(); // Request a redraw
    }

    private void updateGame() {
        // Simple player movement (e.g., move right)
        playerX += 2;
        if (playerX > getWidth()) {
            playerX = -30; // Wrap around
        }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Simple Java Game");
        SimpleGame gamePanel = new SimpleGame();
        frame.add(gamePanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.pack(); // Size the frame to the panel's preferred size
        frame.setLocationRelativeTo(null); // Center on screen
        frame.setVisible(true);
    }
}
```

5.
6.
7. **Input Handling:** Implement `KeyListener` for keyboard input or `MouseListener` for mouse input.
8. **Game Objects:** Create classes for your game elements (e.g., `Player`, `Enemy`, `Bullet`) with properties like position, speed, and methods for drawing and updating.

Kotlin (using JavaFX for graphics):

Kotlin can use Java libraries, including JavaFX for game development. The structure would be similar to Java, but with Kotlin's more concise syntax.

1. **Set Up Your Project:**
    - Create a new Kotlin project in IntelliJ IDEA.
    - Add JavaFX dependencies to your `build.gradle.kts` file.
2. **Game Loop and Drawing:**
    - You'll extend `Application` from JavaFX and use a `Canvas` for drawing.
    - `AnimationTimer` is commonly used for the game loop.
3. <!-- end list -->
4. Kotlin

```
import javafx.animation.AnimationTimer
import javafx.application.Application
import javafx.scene.Group
import javafx.scene.Scene
import javafx.scene.canvas.Canvas
import javafx.scene.canvas.GraphicsContext
import javafx.scene.paint.Color
import javafx.stage.Stage

class SimpleKotlinGame : Application() {

    private var playerX = 50.0
    private var playerY = 50.0

    override fun start(primaryStage: Stage) {
        primaryStage.title = "Simple Kotlin Game"

        val root = Group()
        val canvas = Canvas(800.0, 600.0)
        root.children.add(canvas)
        primaryStage.scene = Scene(root, Color.BLACK)

        val gc = canvas.graphicsContext2D

        object : AnimationTimer() {
            override fun handle(now: Long) {
                updateGame()
                drawGame(gc)
            }
        }.start()

        primaryStage.show()
```

```
    }

    private fun updateGame() {
        playerX += 2.0
        if (playerX > 800) {
            playerX = -30.0
        }
    }

    private fun drawGame(gc: GraphicsContext) {
        gc.clearRect(0.0, 0.0, 800.0, 600.0) // Clear the canvas
        gc.fill = Color.RED
        gc.fillRect(playerX, playerY, 30.0, 30.0)
    }
}

fun main() {
    Application.launch(SimpleKotlinGame::class.java)
}
```

5.
6.

Approach 2: Using Game Frameworks/Libraries

For more complex games, using a dedicated game framework is highly recommended. They handle many low-level details, allowing you to focus on game logic.

- **libGDX (Java/Kotlin):** A powerful, cross-platform game development framework. It supports 2D and 3D graphics, input, audio, physics, and more. It's a popular choice for indie developers and even some commercial games.
    - **Why use it?** Handles rendering, input, asset management, and cross-platform deployment (desktop, Android, iOS, HTML5).
    - **Learning Curve:** Moderate. It provides a lot of flexibility but requires learning its API.
- **KorGE (Kotlin):** A modern, multi-platform 2D game engine built specifically for Kotlin. It's designed to be easy to use and provides a complete set of tools for 2D game development.
    - **Why use it?** Kotlin-native, simplifies common game development tasks, multi-platform.
    - **Learning Curve:** Generally considered easier to get started with than libGDX for 2D games.

General Steps when using a framework:

1. **Choose your framework:** libGDX or KorGE.
2. **Set up your project:** Frameworks usually have setup tools or templates.
3. **Learn the framework's API:** How to draw, handle input, play sounds, etc.
4. **Implement your game logic:** Use the framework's tools to build your game world, characters, and rules.

Next Steps for Deeper Learning:

- **Official Documentation:** Both Java and Kotlin have excellent official documentation.
- **Online Tutorials:**
    - **Java Game Development:** Search for "Java Swing game tutorial" or "libGDX Java tutorial."
    - **Kotlin Game Development:** Search for "KorGE Kotlin tutorial" or "libGDX Kotlin tutorial." Many resources for Java will also apply to Kotlin due to their interoperability.
    - **YouTube Channels:** Many channels offer full game development series (e.g., Kenny Yip Coding for Java Swing games).
- **Books:** Look for books on Java game programming or game development with libGDX/KorGE.
- **Open-Source Games:** Study the code of simple open-source games to see how others have implemented features.

Websites to Practice Coding Through Games

Practicing coding through games is a highly engaging way to solidify your skills. Here are some excellent websites:

1. **CodinGame:** (codingame.com)
    - **Languages:** Supports over 25 programming languages, including Java and Kotlin (though Kotlin might have fewer challenges).
    - **Concept:** You solve coding puzzles and challenges that are presented as mini-games. You write code to control a character or entity within a game world to achieve objectives. They also have competitive programming challenges.
    - **Why it's great:** Highly interactive, competitive element, and a wide variety of problems.
2. **CodeCombat:** (codecombat.com)
    - **Languages:** Primarily Python and JavaScript, but the concepts are transferable.
    - **Concept:** Learn to code by playing a real game. You write code to control a hero character through levels, collecting gems, battling ogres, and solving puzzles.
    - **Why it's great:** Excellent for absolute beginners, makes learning feel like playing.
3. **CheckiO:** (checkio.org)
    - **Languages:** Python and TypeScript (JavaScript).

    - **Concept:** Explore islands and solve coding challenges in a game-like environment. You can see other users' solutions, which is great for learning different approaches.
    - **Why it's great:** Focuses on logical problem-solving and offers community solutions.
4. **Flexbox Froggy / Flexbox Defense:** (flexboxfroggy.com, flexboxdefense.com)
    - **Languages:** CSS (specifically Flexbox).
    - **Concept:** These are fantastic for learning CSS layout. In Flexbox Froggy, you move a frog to its lily pad using Flexbox properties. In Flexbox Defense, you position towers to defend against enemies.
    - **Why it's great:** Specific to web layout, highly visual, and very effective for mastering Flexbox.
5. **Untrusted:** (alex.nisnevich.com/untrusted/)
    - **Languages:** JavaScript.
    - **Concept:** A meta-game where you navigate levels by rewriting JavaScript code that controls the environment. You literally change the game's code to progress.
    - **Why it's great:** Unique approach that forces you to understand how code affects game logic.
6. **Elevator Saga:** (play.elevatorsaga.com)
    - **Languages:** JavaScript.
    - **Concept:** You program the movement of elevators to efficiently transport people in a building. The goal is to optimize your code for speed and efficiency.
    - **Why it's great:** Focuses on algorithms and optimization in a practical, simulation-like setting.
7. **CodeWars:** (codewars.com)
    - **Languages:** Wide variety, including Java and Kotlin.
    - **Concept:** Solve "katas" (coding challenges) and compare your solutions with others. While not a "game" in the traditional sense, it has a martial arts theme and a ranking system that gamifies the learning process.
    - **Why it's great:** Extensive library of challenges, community-driven, and great for improving problem-solving and code elegance.

Remember, consistency is key! Start with simple projects, gradually increase complexity, and don't be afraid to experiment and break things. Happy coding and game making!