Game Programming Fundamentals

Your Journey into Interactive Entertainment Begins Here

Welcome to Week 1 of Game Programming! This course will guide you through the exciting world of game development, from core concepts to hands-on coding. Whether you're dreaming of creating the next viral indie hit or just curious about how your favorite games work, you're in the right place. Over the next two lectures, we'll explore what game programming is, how professional development teams create games, and you'll start building your very first 2D shooter game. No prior experience needed — just bring curiosity and enthusiasm!

What is Game Programming?

Definition

Game programming is the practice of writing computer code that brings video games to life. It's the technical engine behind every jump, explosion, and victory screen you've ever seen. Game programmers take the creative vision of game designers and turn it into working software that players can interact with. Think of it like this: if a game designer is the architect who draws the blueprint, a game programmer is the builder who constructs the actual building.

What Game Programmers Actually Do

- Write the code that controls player movement and actions
- Create systems that make enemies behave intelligently
- Build the physics engine so objects fall, bounce, and collide realistically
- Manage graphics, sound, and user interfaces
- Fix bugs and optimize performance so games run smoothly
- Work with tools like Unity, Unreal Engine, or Godot

Why It Matters

Without programmers, game design remains just an idea. Programming transforms imagination into playable experiences. You're not just writing code — you're creating worlds where people spend hours having fun, experiencing stories, and connecting with others.

Building Your First Game: 2D Shooter

What We're Creating

A 2D Shooter is one of the best projects for beginners because it teaches you essential game programming concepts without overwhelming complexity. You'll create a game where the player controls a spaceship, shoots bullets at enemies, and tries to survive as long as possible. Classic, fun, and educational.

Core Mechanics You'll Learn

Player Control

Making your spaceship respond to keyboard input — movement, rotation, and shooting

Enemy AI

Programming enemies that spawn, move toward the player, and shoot back

Collision Detection

Detecting when bullets hit enemies or enemies hit the player

Score System

Tracking points, displaying them on screen, and saving high scores

Simple Code Example: Moving Your Spaceship

```
// Update player position based on keyboard input
if (Input.GetKey(KeyCode.Left)) {
    player.x -= 5;  // Move left by 5 pixels
}
if (Input.GetKey(KeyCode.Right)) {
    player.x += 5;  // Move right by 5 pixels
}
if (Input.GetKey(KeyCode.Space)) {
    FireBullet();  // Create a new bullet
}
```

This simple code checks if the player is pressing arrow keys, and if so, updates the spaceship's position. When Space is pressed, it fires a bullet. That's the foundation of player input!

Understanding the Game Loop

The Heartbeat of Every Game

Every single game running right now — whether it's on your phone, console, or PC — is constantly repeating a cycle called the game loop. This loop runs dozens of times per second (typically 60 times, called 60 FPS or frames per second). Understanding this loop is fundamental to game programming.

The Three Main Phases

01

Input

Check what the player is doing. Did they press a button? Move the mouse? Touch the screen? The game reads all these inputs.

02

Update

Calculate changes. Move objects, check collisions, update score, make enemies move. Everything that needs to change this frame gets updated based on the input and game rules.

03

Render

Draw everything on screen. Take all the updated positions and states, and display them as images, animations, and text. This is what the player sees.

Code Example: The Game Loop in Action

```
while (gameIsRunning) {
    // INPUT
    HandlePlayerInput();

    // UPDATE
    UpdatePlayerPosition();
    UpdateEnemyPosition();
    CheckCollisions();
    UpdateScore();

    // RENDER
    DrawPlayer();
    DrawEnemies();
    DrawBullets();
    DrawUI();
}
```

This loop repeats 60 times per second. Each repetition is called a "frame." This is why we measure game smoothness in FPS (frames per second) — more loops per second means smoother gameplay.

How Are Games Actually Made?

The Game Development Pipeline

Creating a professional game is a complex journey involving many stages and disciplines. From a tiny indie team of 2-3 people to massive studios with hundreds of employees, the basic process remains similar. Let's walk through how real games are made, from initial spark to launch day.

Conceptualization

A team discusses the core idea. What's the game about? What makes it fun? They create design documents, sketches, and prototypes to test if the basic idea works. This phase might last weeks or months.

Pre-Production

The team creates detailed plans. Artists create concept art, designers write detailed game rules, and programmers plan the technical architecture. Everyone needs to understand the vision before building starts.

Production

This is where the bulk of work happens. Programmers write thousands of lines of code, artists create 3D models and animations, sound designers compose music. The game starts taking shape over many months.

Testing & Polish

Quality assurance teams play the game constantly, finding bugs. Programmers fix issues, artists refine visuals, and the team optimizes performance. The game gets tested on different devices and systems.

Launch & Support

The game releases to the public. The team monitors for issues, releases updates and patches, and listens to player feedback. Many successful games continue evolving for years after launch.

Timeline Reality Check

A small indie game might take 1-2 years to create with a small team. A AAA (big-budget) game like Grand Theft Auto or The Legend of Zelda can take 5-8 years with teams of 150+ people. Even "simple" mobile games usually require 6-12 months of dedicated work.

The Game Designer's Vision

Definition: What's a Game Designer?

A game designer is the creative architect who decides what a game is, how it plays, and how fun it should be. While programmers write the code that makes games work, designers figure out what should happen in those games. They're the "what" while programmers are the "how."

Core Responsibilities

Game Rules & Mechanics

Design the core systems. How do players win? What can they do? What are the limitations? How do difficulty and progression work?

Level Design

Create individual game levels or worlds. Where do enemies spawn? Where are pickups? What's the visual layout? Level designers think about player flow and challenge.

Player Experience

Think about the player's journey. How should they feel? When should they feel accomplished, frustrated, or surprised? Design creates emotional moments.

Documentation

Write design documents that explain everything to the team. Programmers, artists, and sound designers all need to understand the designer's vision.

A Simple Example: Design Decision for Your Shooter

Design Decision: "Enemies spawn every 3 seconds. Each enemy moves toward the player and shoots once every 2 seconds. The player starts with 100 health points. Collecting a blue star restores 25 health and increases score by 100."

Why this matters: The designer decided these specific numbers to balance difficulty. Too many enemies and the game is impossible. Too few and it's boring. The designer's job is finding that perfect balance, then telling programmers exactly what numbers to code.

The Game Development Team

Who Creates Games?

Game development is a collaborative effort involving many specialists. Each person brings unique skills to create experiences that are fun, beautiful, and technically sound. Let's explore the key roles you'll encounter in any game studio.

Programmers

Write all the code. Different programmers might specialize in graphics, physics, AI, or user interfaces. They implement the designer's vision into working software.

Artists

Create 2D and 3D visual assets. This includes character models, environments, textures, animations, and UI elements. They make the game beautiful.

Game Designers

Design game mechanics, levels, difficulty curves, and overall fun factor. They create the blueprint that everyone else follows.

Sound Designers

Create sound effects, music, and voiceovers. Audio is 50% of what makes games immersive, though many people underestimate its importance.

QA Testers

Play the game constantly, find bugs, and verify fixes work. They ensure the game is stable before release.

Project Manager

Keeps everything organized, manages timelines and budgets, and ensures communication flows smoothly between all departments.

How They Work Together

Imagine building your 2D shooter: The designer creates a level layout showing where platforms are, where enemies spawn, and what the visual theme is. Artists create the spaceship sprite and enemy graphics. Programmers write code that positions these graphics based on player input. Sound designers add laser fire sounds. QA plays and reports any glitches. Everyone's work interconnects to create the final game.

Game Development Costs & Limitations

Why Games Cost What They Do

Game development is expensive. A AAA console game can cost $50-150 million to develop. Even indie games typically cost $100,000-$500,000. Understanding these costs helps you appreciate the business side of gaming and make smart choices about your own game projects.

$50M

AAA Game Budget

Large-scale console games with teams of 100+ people, 5-8 year development cycles

$1-10M

Mid-Tier Game Budget

Quality indie games with 10-30 person teams, 2-3 year development

$10K-100K

Small Indie Budget

Solo developers or tiny teams creating games over 6-12 months

Where Does the Money Go?

Personnel Costs (60-70%)

Salaries for programmers, artists, designers, sound engineers, and managers. This is the biggest expense by far. Experienced developers in major studios earn $80K-$150K+ annually.

Software & Tools (10-15%)

Licenses for game engines, graphics software, audio tools, version control systems, and development software.

Hardware & Infrastructure (5-10%)

Computers, servers, networking equipment, and office space for the team.

Marketing & Publishing (10-15%)

Advertising, PR campaigns, convention booths, and distribution costs to get the game in front of players.

Real Constraints You'll Face

- **Time:** Scope creep is the #1 killer of indie games. What seems like a small feature can add months of work. Start small. Your first game should be finishable in 3-6 months.
- **Team Size:** More people doesn't always mean faster development. Communication overhead increases. Small, focused teams (2-5 people) often work better than large bureaucratic ones.
- **Technical Limitations:** Mobile games have CPU/memory constraints. Console games need to run on specific hardware. PC games must work across different configurations. These technical realities shape design decisions.
- **Storage & Performance:** High-quality 3D graphics require massive storage. Loading times matter. Optimization becomes crucial for player experience and can consume significant development time.
- **Testing Compatibility:** Games must work on different devices, screen sizes, operating systems, and hardware configurations. Testing across all these variations is expensive and time-consuming.

Your First Challenge: Writing Your First Code

Let's Build Something Together

Now that you understand the fundamentals, let's write actual game code. Here's a simple example that moves a player character and shows how real game programming works. Don't worry if this looks complex — we'll break it down piece by piece.

Complete Simple Game Example

```
// Define player properties
class Player {
    int x = 400;
    int y = 300;
    int speed = 5;
    int health = 100;

    void Update() {
        // Handle input
        if (Input.GetKey(KeyCode.Left)) {
            x -= speed;
        }
        if (Input.GetKey(KeyCode.Right)) {
            x += speed;
        }
        if (Input.GetKey(KeyCode.Up)) {
            y -= speed;
        }
        if (Input.GetKey(KeyCode.Down)) {
            y += speed;
        }

        // Keep player on screen
        x = Clamp(x, 0, 800);
        y = Clamp(y, 0, 600);
    }

    void TakeDamage(int damage) {
        health -= damage;
        if (health <= 0) {
            GameOver();
        }
    }
}

// Main game loop
while (gameRunning) {
    player.Update();      // Handle input and update
    DrawPlayer(player);   // Render the player
}
```

What Each Part Does

- **Class Definition:**

  We create a "Player" class that holds all information about the player — position (x, y), speed, and health.

- **Update Function:**

  This runs every frame. It checks keyboard input and changes the player's x and y position. The Clamp function keeps the player from moving off-screen.

- **TakeDamage Function:**

  When an enemy hits the player, this function reduces health. If health reaches zero, the game ends.

- **Main Game Loop:**

  Forever (while the game is running), update the player, then draw the player on screen. Simple but powerful!

Your Journey Starts Now

Key Takeaways from Week 1

1 Game programming brings creative visions to life through code.

You're not just writing instructions for computers — you're creating experiences that people enjoy and remember.

2 The game loop is the heart of every game.

Input → Update → Render. Understanding this cycle helps you think like a game programmer.

3 Game development is a team effort with many specialized roles.

Programmers, designers, artists, and sound engineers all contribute their expertise to create something greater than any individual could make alone.

4 Start small and focus on completing projects.

Your first game won't be perfect, and that's okay. Small, finished games teach you more than large, abandoned projects.

5 Game programming combines creativity and problem-solving.

You need both artistic vision and logical thinking. The best game programmers balance both sides of their brain.

What's Next?

In Week 1 Lecture 2, we'll dive deeper into your 2D shooter project. You'll set up your development environment, learn about the game engine we're using, and start writing code that actually makes things move on screen. We'll create the player spaceship, implement shooting mechanics, and build the foundation for enemies to attack.

Challenge for You: Before the next lecture, download and install a game engine (we recommend Unity or Godot — both are free). Explore the interface. Try one of their beginner tutorials. Familiarize yourself with the tools. This preparation will make next week much more productive and fun.

Welcome to game programming. You're about to create something amazing. 🚀