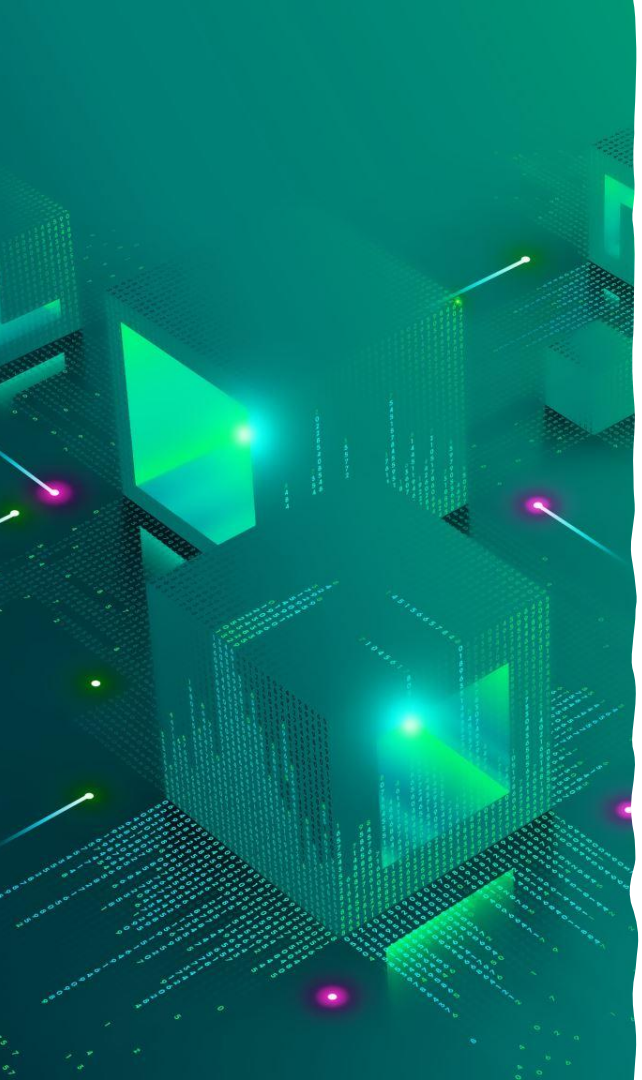
Introduction to Game Programming

Lecture by :
Engr: Fatima Jaffar

What is Game Programming?

- Definition: The art and science of creating interactive digital experiences
- Combines software development with creative design
- Requires understanding of programming, mathematics, physics, and game design
- Creates the logic and mechanics behind video games

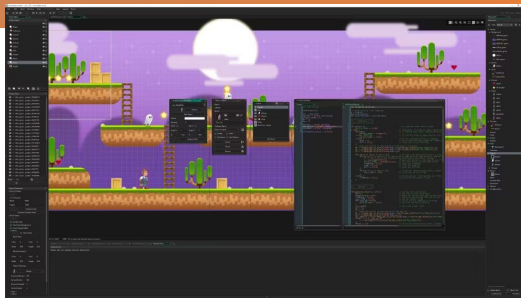


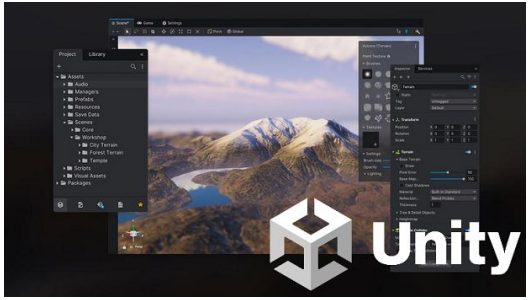
Programming Languages in Game Development

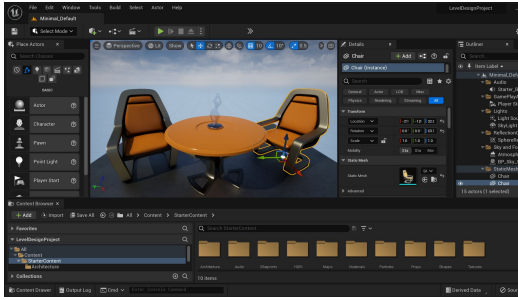
- **C++:** Industry standard for AAA games
- **C#**: Popular with Unity engine
- **Python**: Good for beginners and prototyping
- **Java**: Used in Android game development
- **Lua**: Common for game scripting

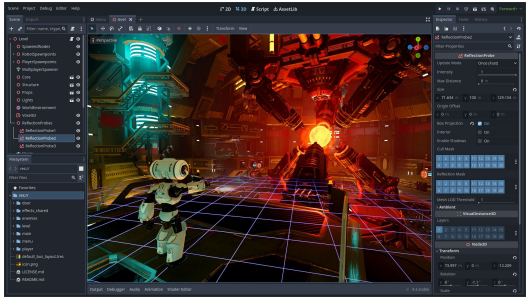
Game Engines Overview

- **Definition: Software frameworks for game development**
- **Popular engines:**
  - **Unity**
  - **Unreal Engine**
  - **Godot**
  - **GameMaker Studio**
- **Provides built-in tools and resources**









Core Programming Concepts

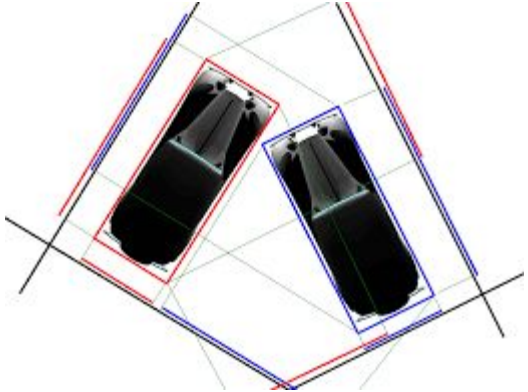
- Variables and data types
- Control structures
- Functions and methods
- Object-oriented programming
- Memory management
- Algorithm design

Game Loop Architecture

- Input processing
- Update game state
- Render graphics
- Maintain frame rate
- Core timing mechanisms
- State management

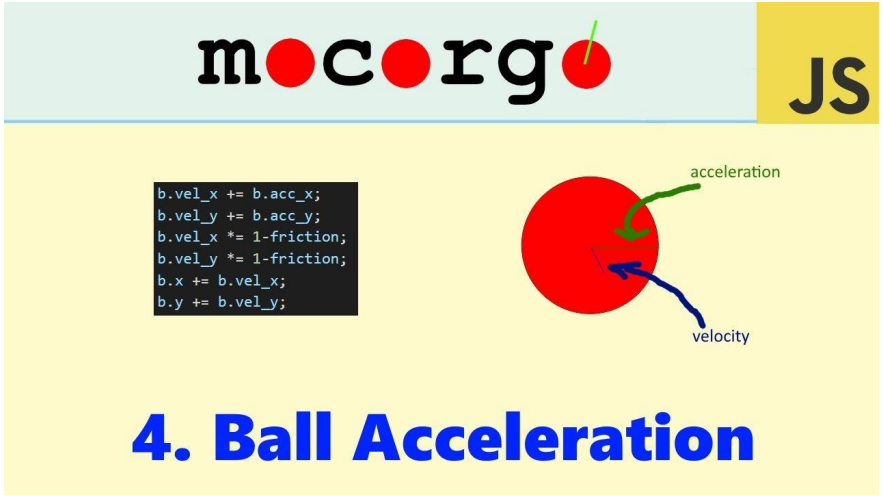
Physics in Games

- Collision detection



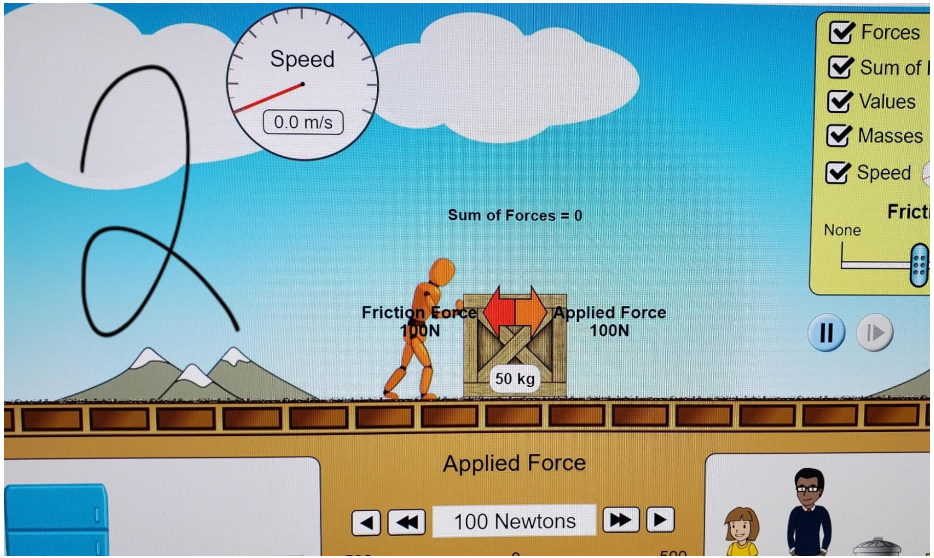
Physics in Games

- Collision detection
- Velocity and acceleration



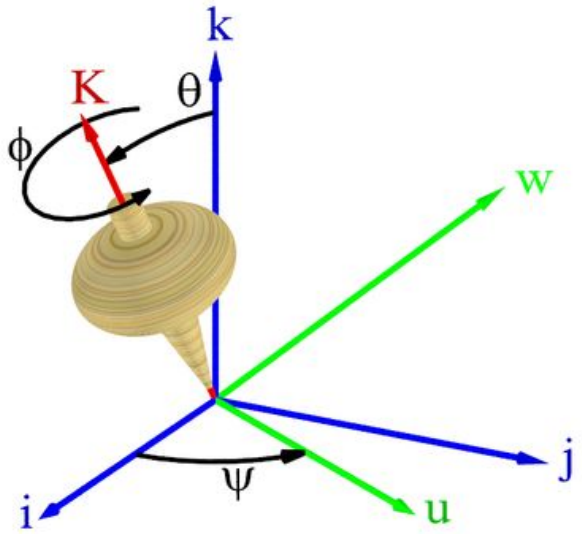
Physics in Games

- Collision detection
- Velocity and acceleration
- Force calculations



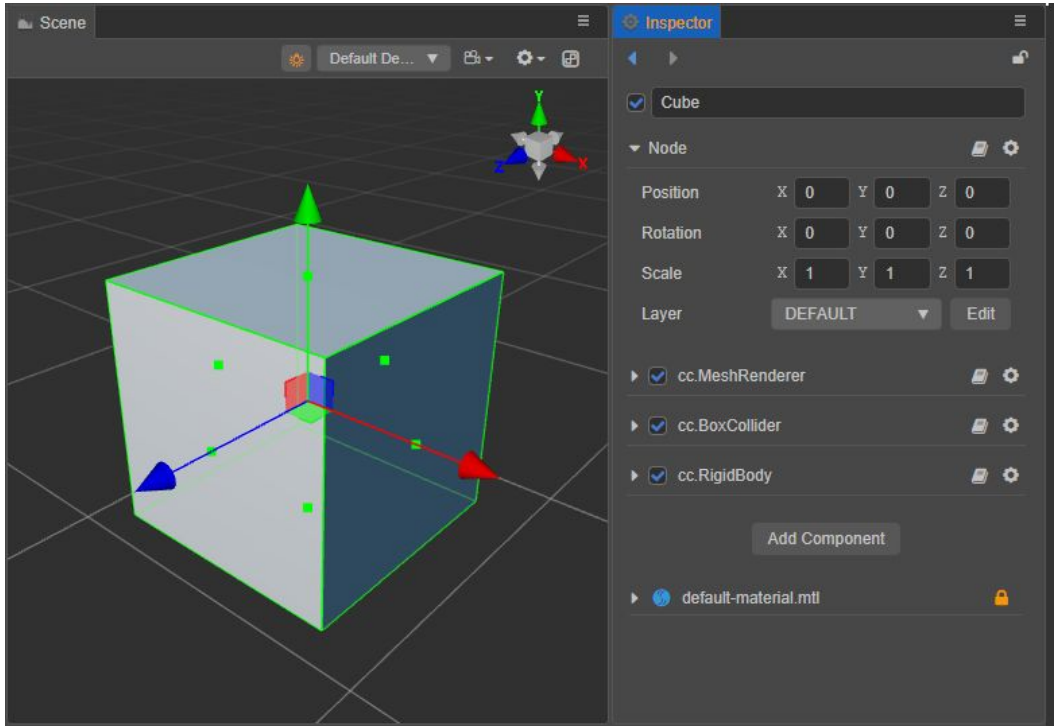
Physics in Games

- Collision detection
- Velocity and acceleration
- Force calculations
- Rigid body dynamics



Physics in Games

- Collision detection
- Velocity and acceleration
- Force calculations
- Rigid body dynamics
- Physics engines



Physics in Games

- Collision detection
- Velocity and acceleration
- Force calculations
- Rigid body dynamics
- Physics engines
- Implementation considerations

Graphics Programming Basics

- Rendering pipeline
- Sprites and textures
- 2D vs 3D graphics
- Shaders
- Animation systems
- Frame buffering

Input Handling

- Keyboard and mouse input
- Controller support
- Touch screen interactions
- Input mapping
- Event systems
- Input buffering

Audio Programming

Sound effects implementation

Background music

3D audio positioning

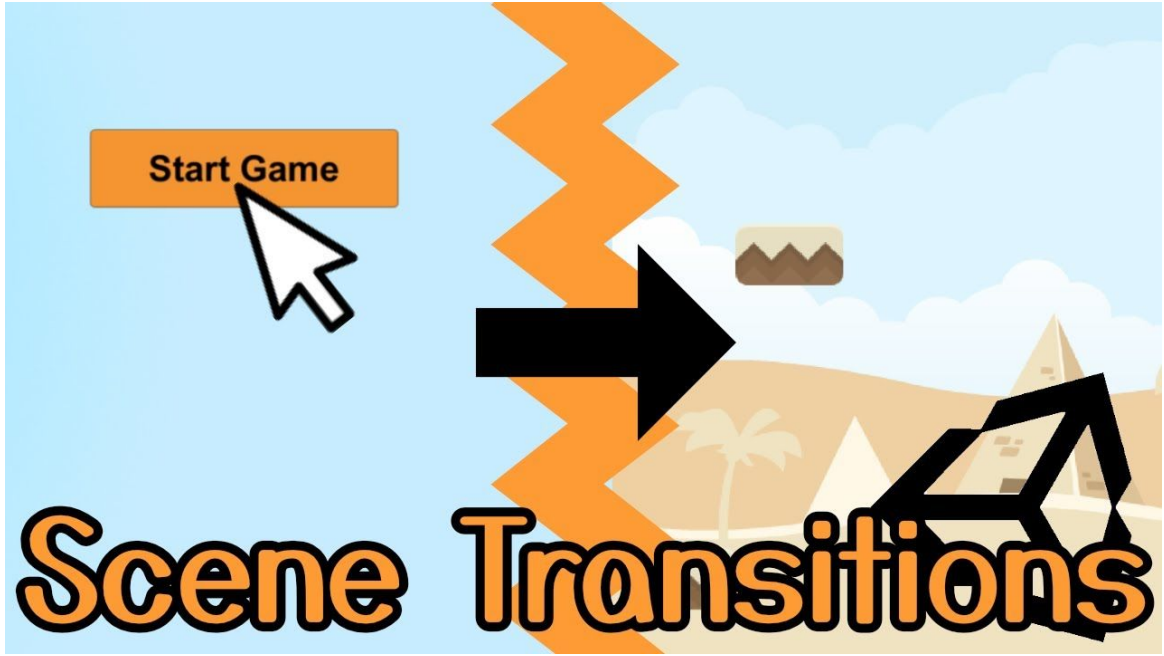
Audio formats

Mixing and effects

Resource management

Game State Management

- Scene transitions

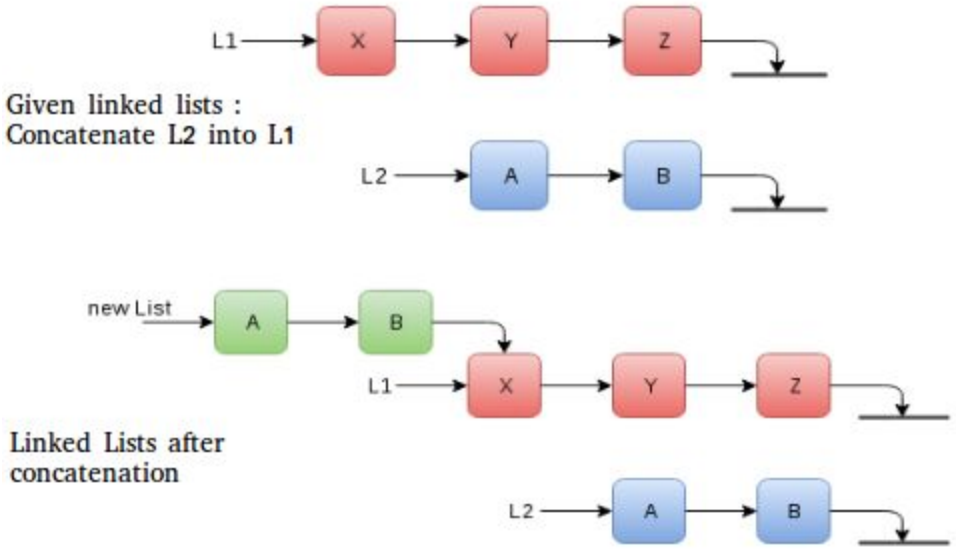


Game State Management

- Scene transitions
- Save/load systems

Game State Management

- Scene transitions
- Save/load systems
- Persistent data

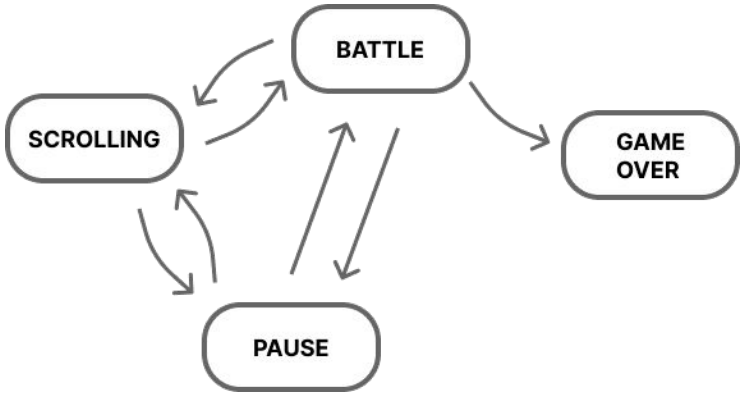


Game State Management

- Scene transitions
- Save/load systems
- Persistent data
- Memory management

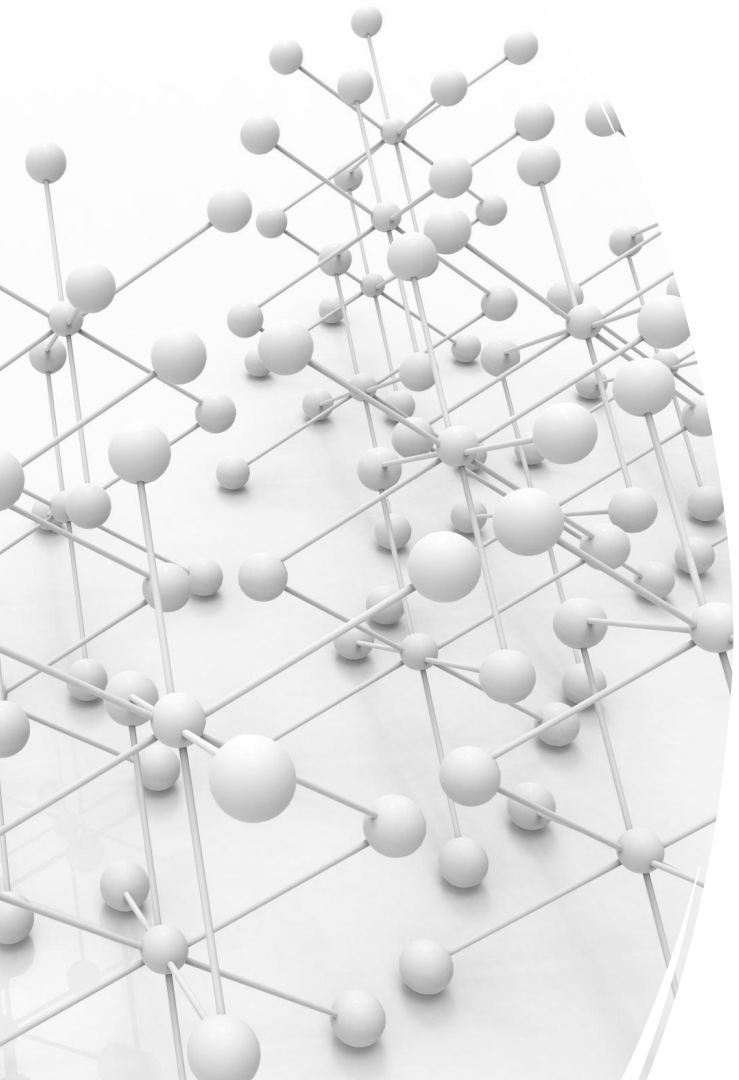
Game State Management

- Scene transitions
- Save/load systems
- Persistent data
- Memory management
- State patterns



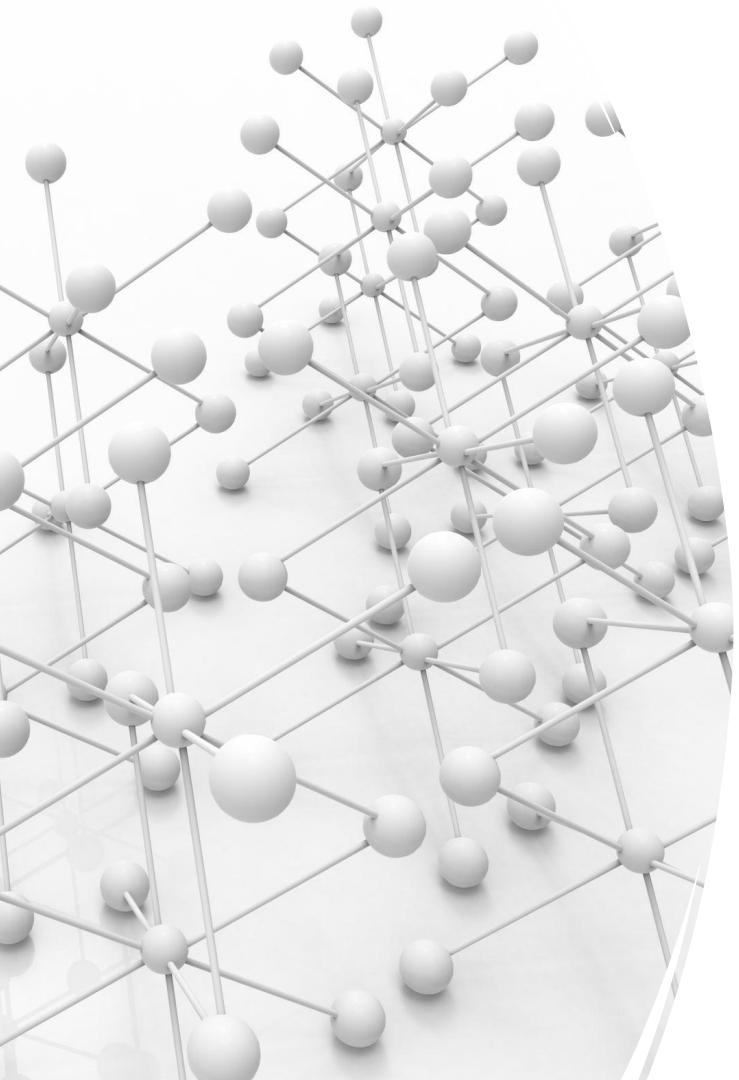
AI in Games

- Pathfinding algorithms



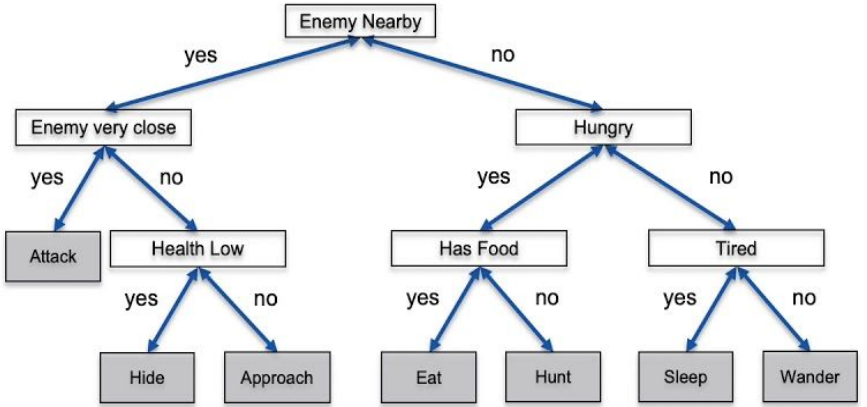
AI in Games

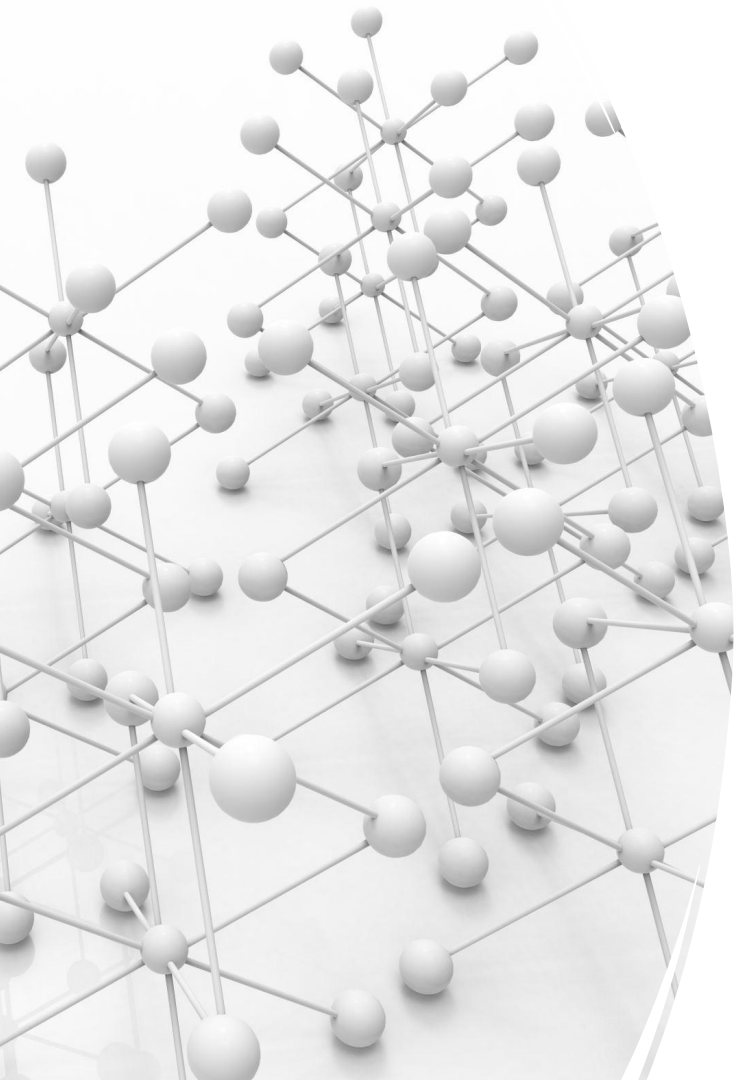
- Pathfinding algorithms
- Decision making



AI in Games

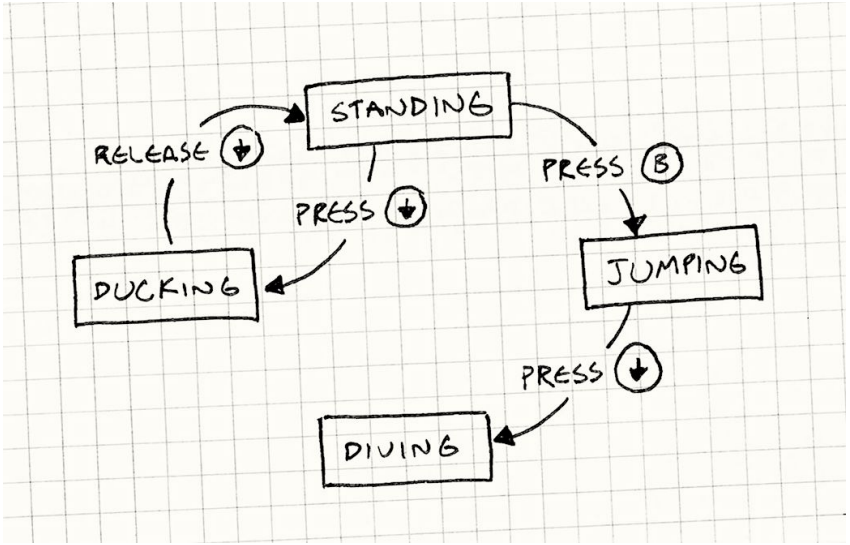
- Behavior trees





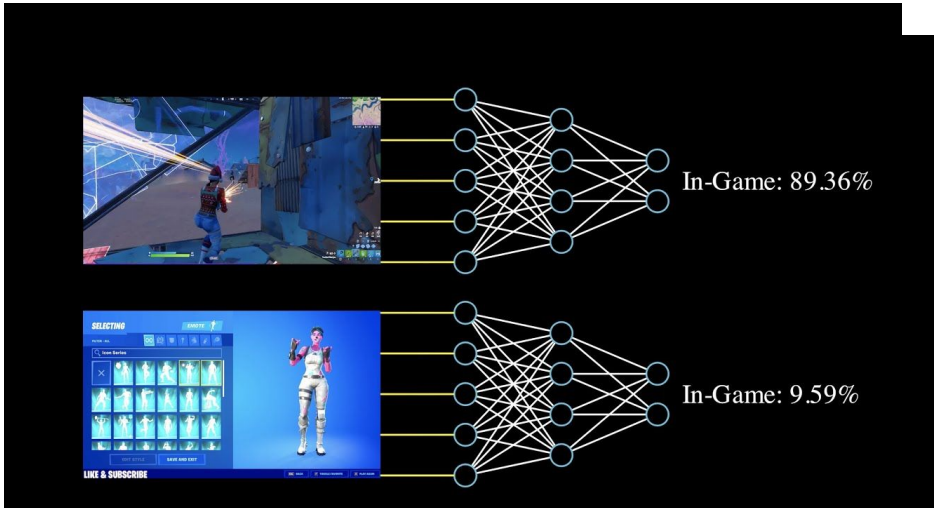
AI in Games

- State machines



AI in Games

- Neural networks



AI in Games

- NPC interactions

Optimization Techniques

Memory management

CPU optimization

GPU optimization

Load times

Asset streaming

Performance profiling

Networking Basics

Client-server architecture

Peer-to-peer networking

Latency handling

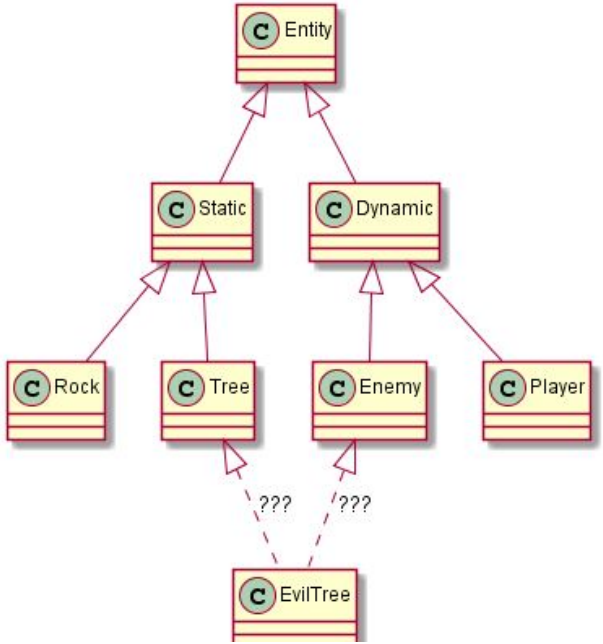
State synchronization

Multiplayer considerations

Security concerns

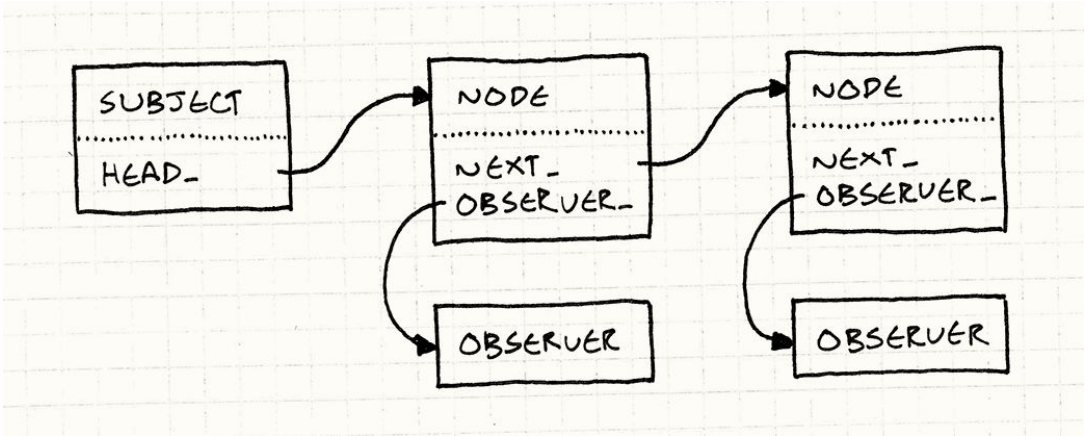
Game Design Patterns

- Component systems



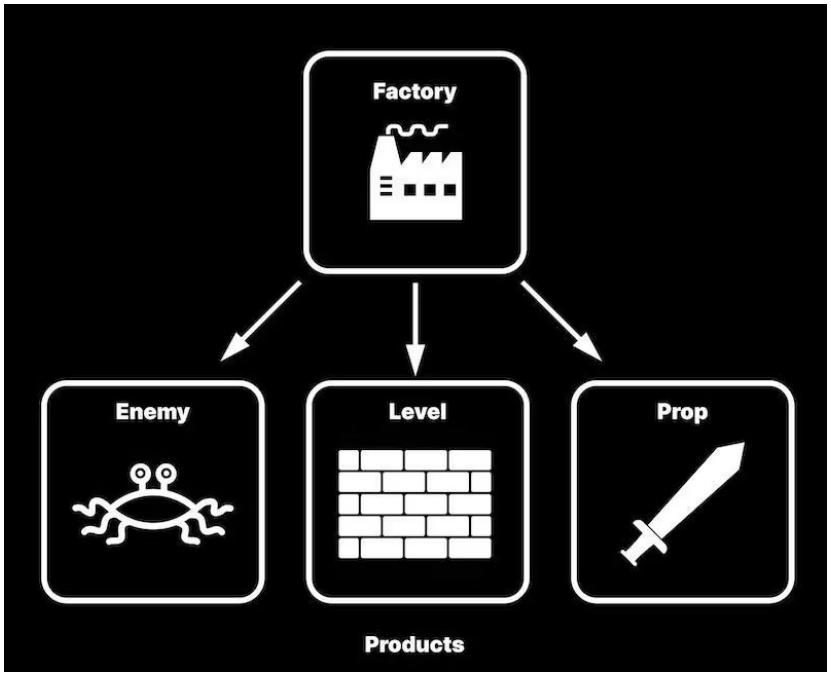
Game Design Patterns

- Component systems
- Observer pattern



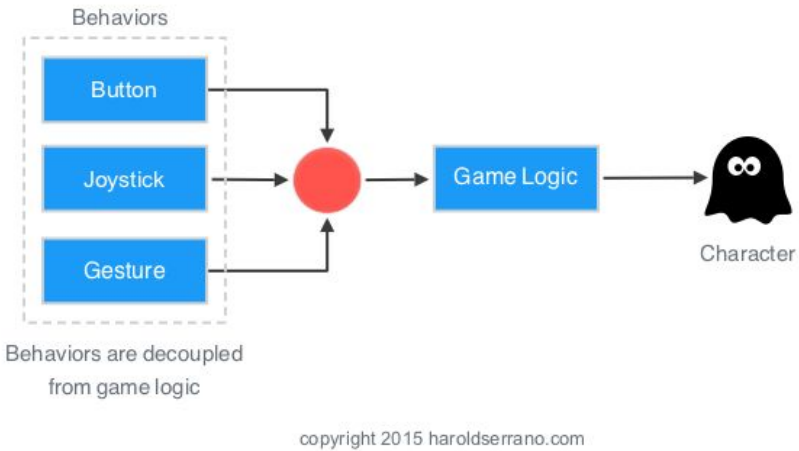
Game Design Patterns

- Component systems
- Observer pattern
- Factory pattern



Game Design Patterns

- Component systems
- Observer pattern
- Factory pattern
- Singleton pattern



Debug and Testing

- Debugging tools
- Unit testing
- Integration testing
- Playtesting
- Bug tracking
- Performance monitoring

Version Control

- Git basics
- Branching strategies
- Asset management
- Collaboration workflow
- Code reviews
- Release management

Development Tools

- IDEs
- Asset creation tools
- Version control systems
- Bug tracking software
- Profiling tools
- Documentation tools

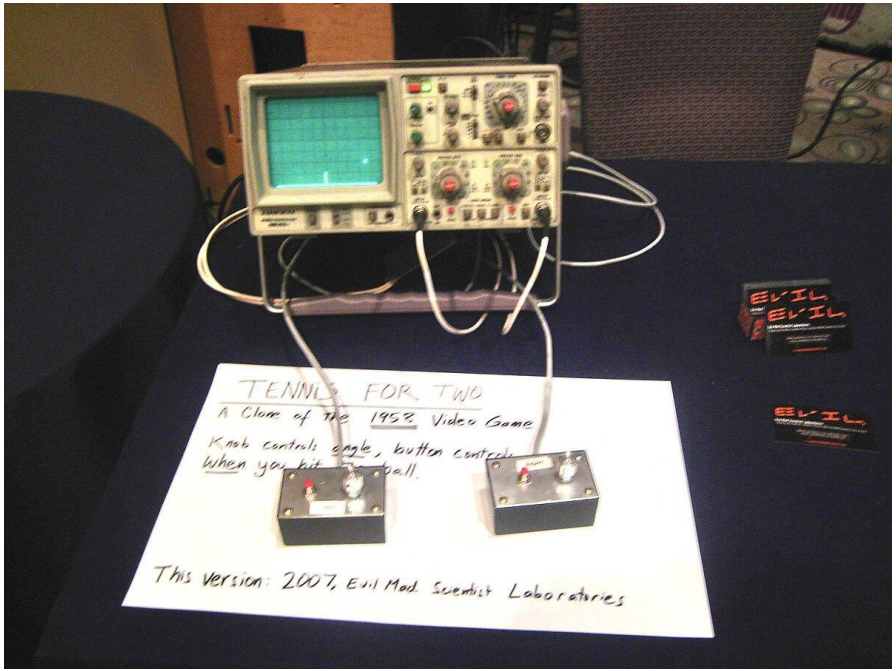
Getting Started

- Learning resources
- Practice projects
- Community involvement
- Portfolio building
- Industry trends
- Career paths

The Evolution of Game Programming: From Pixels to Virtual Worlds

The Dawn of Game Programming (1950s-1960s)

- First interactive computer game: Tennis for Two (1958)
- Created by William Higinbotham using an oscilloscope



The Dawn of Game Programming (1950s-1960s)

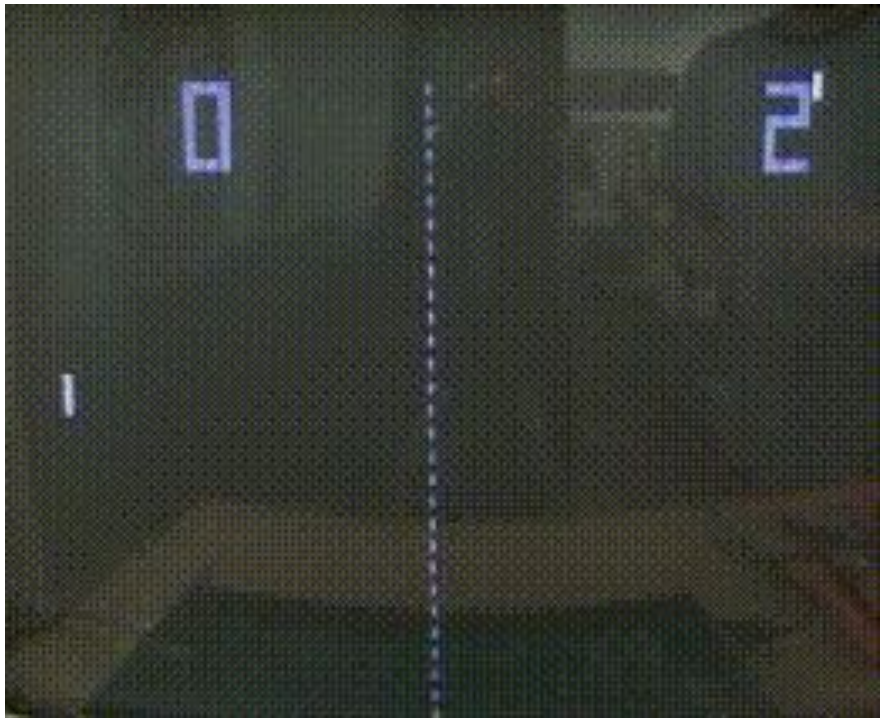
- Spacewar! (1962) - First digital computer game
- Early games ran on mainframe computers
- Programming languages: Assembly and early versions of FORTRAN

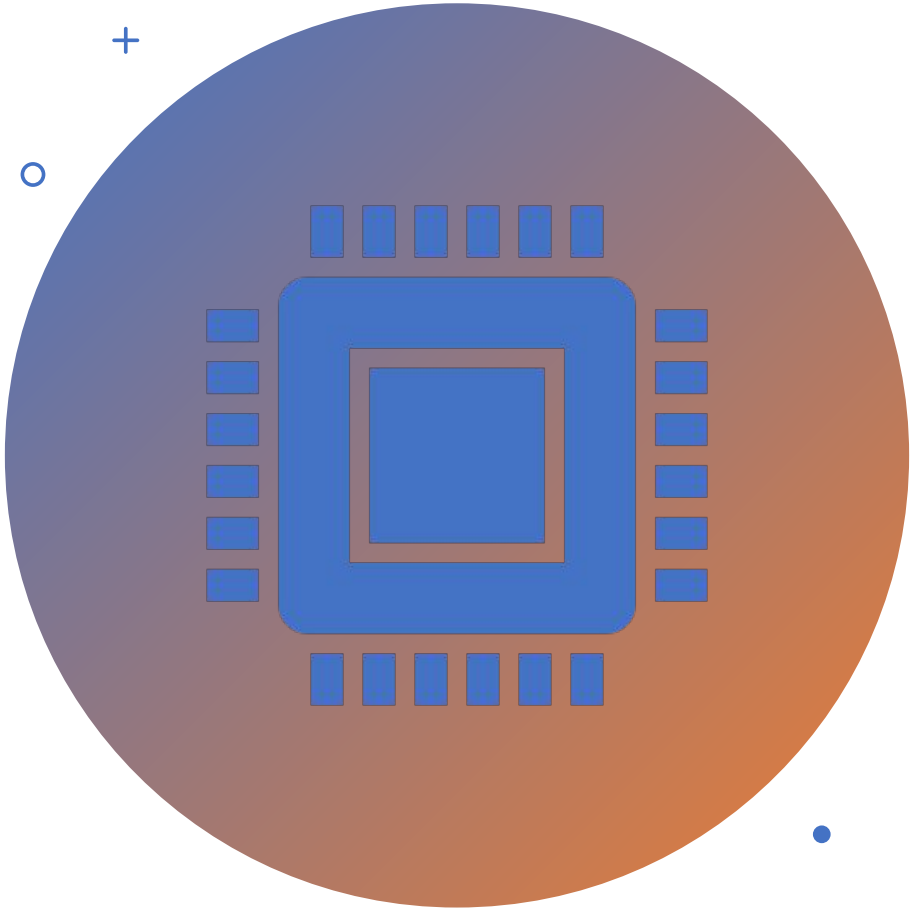
The Arcade Era (1970s)

- First arcade game: Computer Space (1971)

The Arcade Era (1970s)

- Pong (1972) revolutionizes gaming industry
- Rise of dedicated gaming hardware
- Introduction of microprocessors
- Programming primarily in Assembly language
- Focus on optimization due to hardware limitations





The Home Computer Revolution (1980s)

- Birth of personal computers (Apple II, Commodore 64)
- Introduction of BASIC programming language
- Rise of game development tools
- Emergence of game engines
- Sprite-based graphics programming
- Introduction of save game features

The 16-bit Era (Late 1980s-Early 1990s)

- Advanced graphics capabilities
- Introduction of C programming for games
- Structured programming approaches
- Development of level editors
- Sound programming advancement
- Rise of game development teams

The 3D Revolution (Mid 1990s)

- Introduction of 3D graphics programming
- OpenGL and DirectX APIs
- C++ becomes industry standard
- Physics engine development
- Advanced AI programming
- Modular programming approaches

Internet Gaming Era (Late 1990s-Early 2000s)

- Network programming for multiplayer games
- Client-server architecture
- Database integration
- Security considerations
- Emergence of MMORPGs
- Browser-based game development

Modern Game Development (2000s-2010s)

- Unity and Unreal Engine
- Mobile game development
- Cross-platform programming
- Middleware integration
- Asset pipeline management
- Version control systems

Current Trends (2010s-Present)

Cloud gaming infrastructure

Virtual and Augmented Reality

Machine Learning in games

Procedural content generation

Live service architecture

Advanced physics simulation

Future Directions

- Quantum computing applications
- AI-driven game development
- Blockchain gaming
- Advanced rendering techniques
- Cross-reality experiences
- Democratization of game development tools

Game Programming: The Digital Entertainment Industry

Industry Overview

- Global gaming market value: ($200) billion industry
- Major sectors: Mobile, Console, PC gaming
- Key players: Sony, Microsoft, Nintendo, Epic Games, EA
- Fastest growing entertainment sector
- Employs millions worldwide in various roles

Game Development Roles

- Game Programmers: Core mechanics, engine development
- Game Designers: Gameplay systems, level design
- Artists: 2D/3D art, animation, visual effects
- Sound Engineers: Music, sound effects
- Quality Assurance: Testing, bug reporting
- Producers: Project management, team coordination

Programming Languages in Gaming

- C++: Industry standard for AAA games
- C#: Unity engine development
- Python: Tools and prototyping
- Java: Android game development
- Lua: Scripting and modding
- JavaScript: Browser-based games

Game Engines

- Unity: Versatile, popular for indie development
- Unreal Engine: AAA standard, powerful graphics
- Godot: Open-source alternative
- Custom Engines: Built for specific needs
- Features: Physics, rendering, AI systems
- Market share and licensing models

Technical Considerations

- Performance optimization
- Memory management
- Frame rate and rendering
- Network architecture
- Cross-platform development
- Asset management systems

Current Industry Trends

- Cloud gaming services
- Virtual/Augmented Reality
- Cross-platform play
- Live service games
- Procedural generation
- AI-driven experiences

Career Pathways

- Entry-level positions: Junior programmer, QA
- Mid-level: Lead programmer, Engine specialist
- Senior roles: Technical director, CTO
- Indie development
- Specialized roles: AI, Graphics, Network programming

Market Analysis

- Revenue models: F2P, Premium, Subscription
- Platform market shares
- Regional markets: Asia, North America, Europe
- Growth projections
- Investment trends
- Market demographics

Future Outlook

- Emerging technologies: AI/ML integration
- New platforms and hardware
- Industry challenges and opportunities
- Skills demand forecast
- Environmental impact considerations
- Accessibility and inclusivity trends

Resources and Next Steps

- Industry certifications
- Online learning platforms
- Professional organizations
- Game jams and hackathons
- Networking opportunities
- Portfolio development strategies