Session One: Overview of the Game Development Process

DigiPen Institute of Technology
5001 150th Ave NE, Redmond, WA 98052
Phone: (425) 558-0299
www.digipen.edu
**INTRODUCTION:**

In partnership with Microsoft, DigiPen Institute of Technology is pleased to present a series of eight one-hour webcasts to introduce participants on how video games are produced. The main game development concepts will be demonstrated via the creation of a “top-down shooter” using C# 2005 Express Edition, Microsoft’s new lightweight integrated development environment designed for beginning programmers.

Participants are able to download C# 2005 Express Edition along with webcast tutorial materials to experience building the game themselves. It should be noted that the intent of this webcast series is to provide an overview of the game development process and is not designed to train participants in all aspects of C#.

**Webcast Sessions:**
- Session 1: Overview of Game Development Process
- Session 2: Basic Programming Concepts and Introduction to C#
- Session 3: Overview of Game Elements
- Session 4: Introduction to Sprites and Animation
- Session 5: Transformation and Collision of Sprites
- Session 6: Player Control of Sprites
- Session 7: Game Music and Sound Effects
- Session 8: Creating Sprite Behavior

The DigiPen Game Development Webcast Series continues through May 2005. Go to [http://www.microsoft.com/events/series/msdnvideodev.mspx](http://www.microsoft.com/events/series/msdnvideodev.mspx) to register or to access archives.

**ABOUT DIGIPEN INSTITUTE OF TECHNOLOGY:**

Based in Redmond, Washington, a major center of game development in the United States, DigiPen Institute of Technology is acknowledged as a leader in interactive entertainment technology education. Since 1994, DigiPen has been successfully educating students for careers in the video game industry as programmers, production artists and computer engineers.

**DigiPen offers the following degrees:**
- Master of Science in Computer Science
- Bachelor of Science in Real-time Interactive Entertainment
- Bachelor of Science in Computer Engineering
- Associate of Applied Arts in 3D Computer Animation
- Bachelor of Fine Arts in Production Animation.

For more details about DigiPen’s degree programs, visit [http://www.digipen.edu](http://www.digipen.edu).

In addition to this webcast series, DigiPen Institute of Technology also offers a number of outreach efforts to explore a career in video game development:

**DigiPen Summer Workshops:**
Series of introductory classes in game programming, 3D animation production and robotics. Information about the workshops is available at [http://workshops.digipen.edu](http://workshops.digipen.edu).

**DigiPen ProjectFUN® Distance Learning Initiative:**
DigiPen has launched a new online program taught live by DigiPen instructors. For details, go to [http://projectfun.digipen.edu](http://projectfun.digipen.edu).
1 Introduction to Game Web Cast Project

- Introduce the web cast topic and outline. The preparation for this section should start after the hours from 2 to 8 are prepared.
  - Introduction: covers what the first hour is about
  - C# overview: covers what the second hour is about
  - Game components: covers what the third hour is about
  - Building the demo: covers what the rest of the hours are about
- Second section of the first hour starts here; it is basically used to explain what a real-time interactive concurrent events application is.
- Play the demo game
  - Please note that the game is implemented with simplicity in mind in order to show clearly the game architecture and components
  - Try to show that the events of the game happen concurrently
  - Try to show that the objects are moved at real time using the keyboard interaction
- What we just saw is a real-time interactive concurrent events application simulating a starship fighting enemies in space.
- Why concurrent?
  - The starship, bullets, enemies, sound effect, music, and text exist and move at the same time as a series of coincident events
  - The list of coincident events:
    - Testing the keyboard for up, down, left, and right key strokes
    - Scrolling the background
    - New ship position calculation
    - Testing if the fire key is hit
    - Creating a bullet
    - Calculating the new bullet position
    - Testing for collision between each bullet with each enemy
    - Calculating the new enemy position
    - Playing a sound effect when an enemy intersects with a bullet
    - Playing the explosion animation when an enemy intersects with a bullet
    - Updating the texts
    - Playing the music
    - Etc.
- Why interactive?
  - The player decides when and where to move the ship
  - The player decides when and where to fire a bullet
  - The game AI decides when to attack
  - The number of bullets increases each time a bullet is fired
  - The score updates each time an enemy is hit
  - The enemy explodes when hit by a bullet
  - Etc.
- Why real-time?
  - When the ship moves, its new position is calculated at run time
  - The collision between the bullet and the enemy is detected at run time
  - The enemy’s new position happens at run time
  - The text updates at run time
  - Etc.

How do we write an application with concurrent events?
  - We need to be able to execute several instruction at the same time
  - It would be nice if we have a CPU dedicated for each event
  - Usually, we only have one CPU
  - But we need to execute in parallel several instructions
o If we divide the second into 60 pieces
  o Each piece would be 1/60 or 0.016 a second (this is also 16.66 milliseconds)
  o If during each 16.66 ms we update sequentially all game component
  o The game components would be updated 60 times a second
  o The player will get the illusion that the events are coincident or parallel – this means that the events are happening at the same time.
  o In reality the events are not parallel; they are pseudo-parallel
  o The events do not happen at the same time, they happen sequentially
  o Because the events are updated at a fixed interval of 60 times a second, the illusion of concurrent events is achieved
    o Each 16.66 ms duration is one game iteration
  o Since the iteration repeats as long as the game is playing, the concurrent events are controlled by repeating the game iteration through a game loop
  o The game loop also makes the series of events update as a motion picture or pictures in motion

• Game loop
  o The game loop iteration duration greatly affects the illusion of concurrent events
    o If the duration of the game iteration is long, lets say 0.1 s, then the simulation will feel slow
    o Why? Because the reaction to the events happens only 10 times in one second
  o Show the game at 10 fps, 30 fps, and 60fps
  o On the other hand if the duration of the game iteration is short, like 0.016 s, then the simulation feel smooth
    o Why? Because the reaction to the events happens 60 times in one second
  o Consequently, the duration of the game iteration is called the frame
    o Therefore, the game speed is measured by frames per second
    o For example, when we say a game speed is 60 frames per second or 60 f/s or 60 fps, then the game iteration duration is 16.66 ms

• How do we add interaction at real time with concurrent events?
  o During the game iteration, we:
    ▪ Detect and register the user input
    ▪ Execute the behavior of each object; usually the object behavior depends on:
      • Input from the keyboard
      • Input from other objects
      • Collision status
      • AI
      • Etc.
    ▪ Once all the objects are updated, their position and status at the current game loop is determined
    ▪ We render the objects
    ▪ What we just mentioned means that the objects are rendered as many times a second as the game speed
      ▪ For example in a 60 fps game, the objects are rendered 60 times a second
      ▪ During the beginning of the rendering during the game loop, a blank frame is prepared
      ▪ Then all the objects are rendered sequentially
      ▪ When an object o moves, its position at game loop n is slightly different than its previous position at game loop n-1; also, its position at game loop n+1 would be different than the position at game loop n
    ▪ Show example of the moving bullet at a low frame rate
- Being drawn several times consecutively, the sequence of different pictures and different positions provide the illusion of a motion picture
- As a matter of fact, movies at the movie theater play at 25 pictures per second

- Game components
  - Background: static object
  - Sprites: dynamic objects
  - Text
  - Sound
    - Sound effects
    - Music
  - Object behavior: specifies its interaction.

- Overview of the game flow

**Game Loop**

1. Initialize game
2. Initialize starting time
3. Read player input
4. Handling
   - Computing the new position for every object based on:
     - Behavior
     - Collision
     - Physics
     - Artificial Intelligence
5. Drawing
   - Rendering the image one object at a time to a buffer
6. Copy the buffer content to the Video Memory in order to display the frame
7. Check if elapsed time is less than 16.66 ms
   - Yes: Do nothing
   - No
8. Initialize game

---

DigiPen Game Development C# Webcast Series - Session One, Version 1.0
Copyright © 2005 DigiPen (USA) Corporation.