717310: Game Programming

Steffan Hooper

Wednesday, 29 July 2015

Overview

- Game Programming Tools and Libraries
- Game Structure
  - Middleware and Libraries
  - Game Engines
- The Game Loop
  - Timing
- Exercises

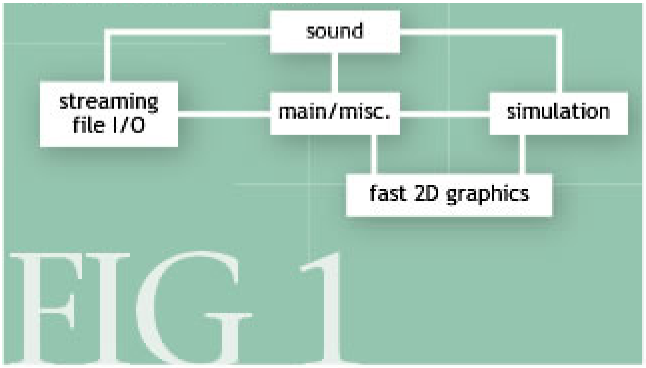
Game Programming Tools and Libraries

- Common Languages:
  - C++, Java, C#
- Software Engineering:
  - OOA:
    - Functional Requirements, Game Design Document, …
  - OOD:
    - Software Architecture, Technical Design Document, …
  - OOP: Implementation, …
  - Methodology: Agile…

Game Structure

- Game Structure
  - Modules:
    - Different responsibilities…
    - Contain algorithms…
  - Compare game complexity, over time:
    - Early 2D games…
    - Early 3D games…
    - Modern 3D games…
    - Modern 3D massively multiplayer games…

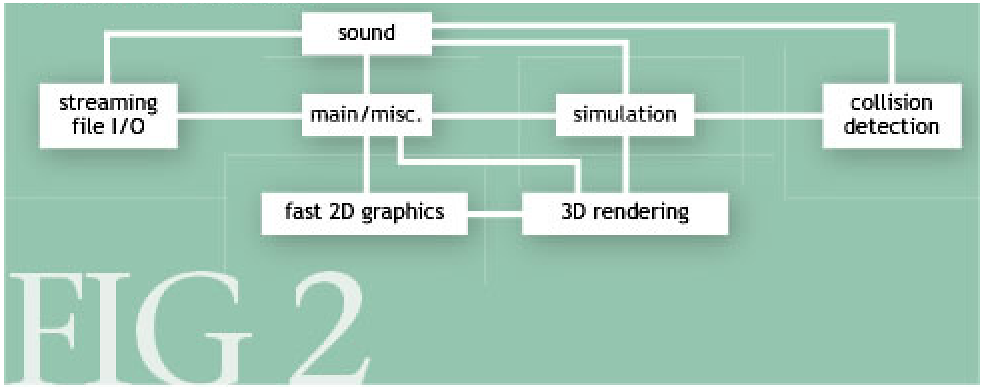
A 2D Game circa 1994:



http://queue.acm.org/detail.cfm?id=971590

Game Structure

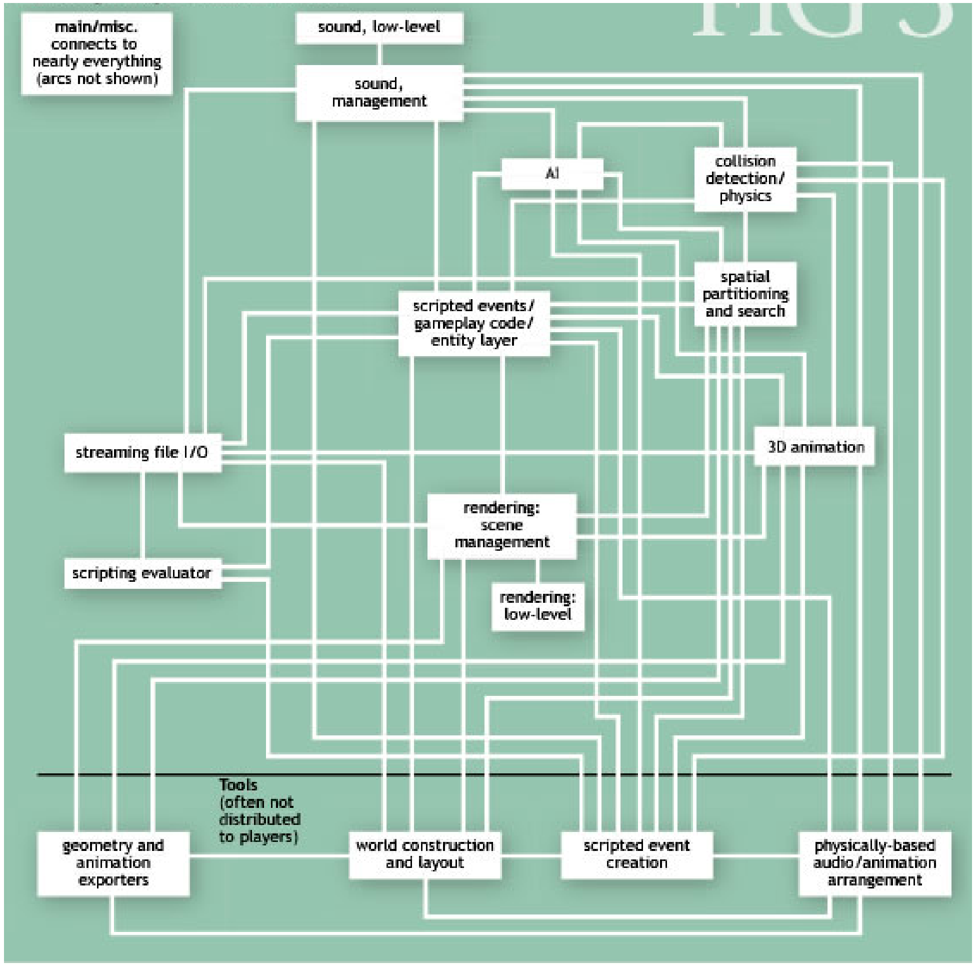
A 3D Game circa 1996:



http://queue.acm.org/detail.cfm?id=971590

Game Structure

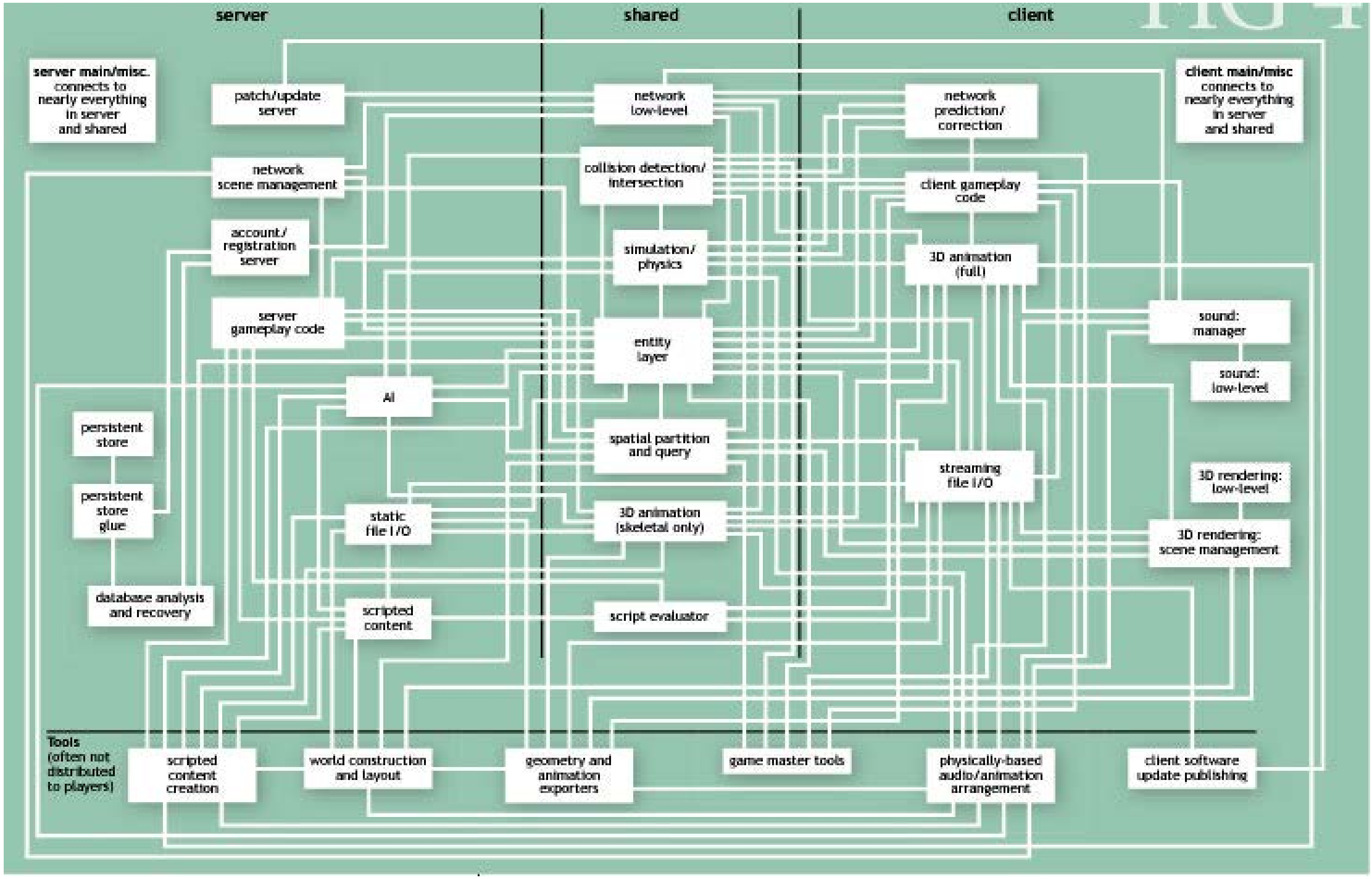
A 3D Single Player Game circa 2004:


main/misc. connects to nearly everything (arcs not shown)
sound, low-level
sound, management
AI
collision detection/ physics
spatial partitioning and search
scripted events/ gameplay code/ entity layer
streaming file I/O
3D animation
rendering: scene management
scripting evaluator
rendering: low-level
Tools (often not distributed to players)
geometry and animation exporters
world construction and layout
scripted event creation
physically-based audio/animation arrangement

http://queue.acm.org/detail.cfm?id=971590

Game Structure

A 3D Massively Multiplayer Game circa 2004:


server
shared
client
server main/misc. connects to nearly everything in server and shared
patch/update server
network scene management
account/ registration server
server gameplay code
AI
persistent store
persistent store glue
static file I/O
database analysis and recovery
scripted content
network low-level
collision detection/ intersection
simulation/ physics
entity layer
spatial partition and query
3D animation (skeletal only)
script evaluator
network prediction/ correction
client gameplay code
3D animation (full)
streaming file I/O
client main/misc connects to nearly everything in server and shared
sound: manager
sound: low-level
3D rendering: low-level
3D rendering: scene management
Tools (often not distributed to players)
scripted content creation
world construction and layout
geometry and animation exporters
game master tools
physically-based audio/animation arrangement
client software update publishing

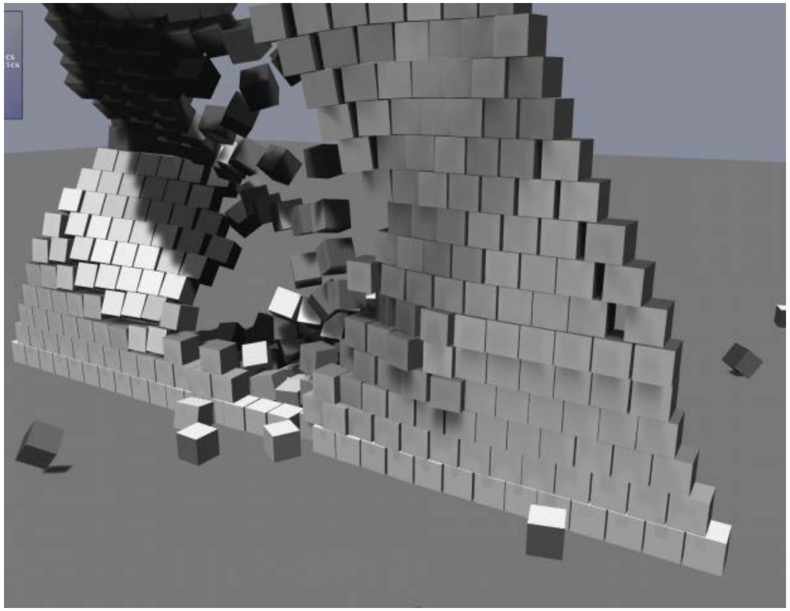
http://queue.acm.org/detail.cfm?id=971590

Game Structure

- Game Modules:
  - – Graphics / Rendering Engine
    - Scene Management
    - Models
    - Bones
    - Animation
    - Shaders
    - Particles

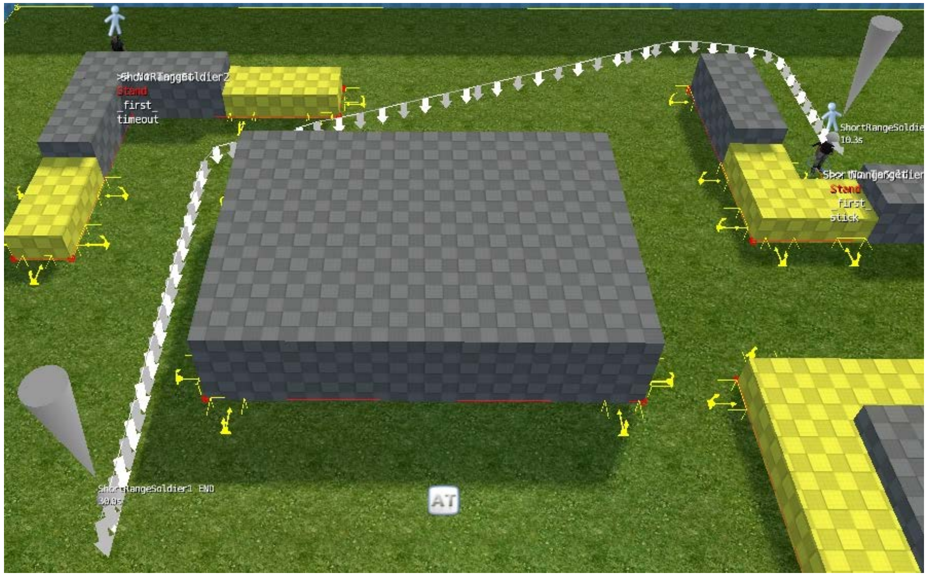
Game Structure

- Game Modules:
  - Physics Engine:
    - Collision Detection
    - Collision Response
    - Rigid Body
    - Joints
    - Mass/Spring
    - Particles:
      - Fluids, Smoke, Cloth, …



Game Structure

- Game Modules:
  - Artificial Intelligence:
    - Behaviour
    - Strategy
    - Path Planning
    - Learning
  - Scripting:
    - Data Driven



Game Structure

- Game Modules:
  - Networking:
    - Multiplayer
    - Client/Server
  - Streaming:
    - Video, Audio
    - Level Data
  - Memory Management
  - Process Management

Game Structure

- Game Modules:
  - Audio Engine:
    - Play Sound Effects
    - Play Music
    - 3D Sound
    - Occlusion
  - Events:
    - Event Driven Model

Game Programming Tools and Libraries

- Middleware:
  - Modular software that is integrated into a game to handle some specialised aspect…
- Libraries:
  - Graphics: DirectX, OpenGL
  - Physics: Box2D, PhysX, Havok, Bullet, ODE, PAL
  - Networking: RakNet, DemonWare, GameSpy
  - Artificial Intelligence: OpenSteer, xaitment
  - User Interface: Scaleform
  - Sound: Fmod, Wwise, Open AL

Game Programming Tools and Libraries

- Game Engine Overview:
  - Purpose: Reusable components across games…
    - “Recyclability”
  - Importance: Fit for purpose…
  - Architecture: Abstraction, Modularity…
  - Design:
    - Purpose… Particular Style of Game?
  - Data Pipelines:
    - Asset tool chain, Export Process, Editors, …

Game Programming Tools and Libraries

- Game Engine Features:
  - Engine Source Code available?
  - Live preview of executable on target platform…
  - Middleware integration…
  - Adaptability:
    - Resource management flexibility?
  - Support:
    - Access to development builds…
    - Developer support, forums, etc…

Game Programming Tools and Libraries

- Game Engine Categories:
  - High-fidelity:
    - Unreal Engine (Epic Games), CryEngine (Crytek), Source (Valve), idTech (id Software)
  - Mid-range:
    - Trinigy (Havok), Gamebryo (Emergent), Vicious Engine (Vicious Cycle Software), BlitzTech (Blitz Games Studios)
  - Casual:
    - Unity (Unity Technologies), Torque 3D (GarageGames), ShiVa (Stonetrip), Marmalade (Marmalade Technologies)

Game Programming Tools and Libraries

- Resource Management:
  - What are Resources?
  - Resources:
    - Assets: Sprite, Textures, Models, Animations, Sounds, Music, Movies, Level Data, …
    - Things that take up memory!
    - Have a location: file path…
    - Different platforms will have different formats!

Game Programming Tools and Libraries

- Resource Management:
  - Management?
    - Loading… from disk to memory.
    - Unloading… no longer needed in memory.
      - Limited memory platforms!
    - As the game needs…
      - Levels, Scenes, Models, Textures, etc…
    - Avoid loading the same resource twice!
    - Keep track of loaded resources…
    - Templatized (Generic) or Specific Design…

Game Programming Tools and Libraries

- Types of Input:
  - – Digital Button: On/Off
    - Up, Pressed, Released, Held
  - – Analogue: 0% to 100%
    - Sticks, Buttons
  - – Keyboard, Mouse, Gamepad Controller, Motion…
  - – Accelerometer…
  - – Gyroscope…
  - – Camera…

Game Programming Tools and Libraries

- Input Management:
  - Polling:
    - Repeatedly ask for current input state…

```
if (aKeyPressed == true) { /* … */ }
```

  - Event-Based Systems:
    - If an event occurs, then a function is called…
  - Buffered Input: Store all input, then process…
  - Key bindings…
    - Remapping… User configurable… HCI!

Game Programming Tools and Libraries

- Input Example, Microsoft’s XInput:
  - Using Xbox 360 Controllers in Windows…
    - With the DirectX API…
    - Retrieve the current state of a controller:

```
DWORD XInputGetState(DWORD dwUserIndex,
                           XINPUT_STATE* pState);
```

    - XINPUT_STATE Structure?

```
DWORD dwPacketNumber, XINPUT_GAMEPAD Gamepad
```

    - XINPUT_GAMEPAD Structure?

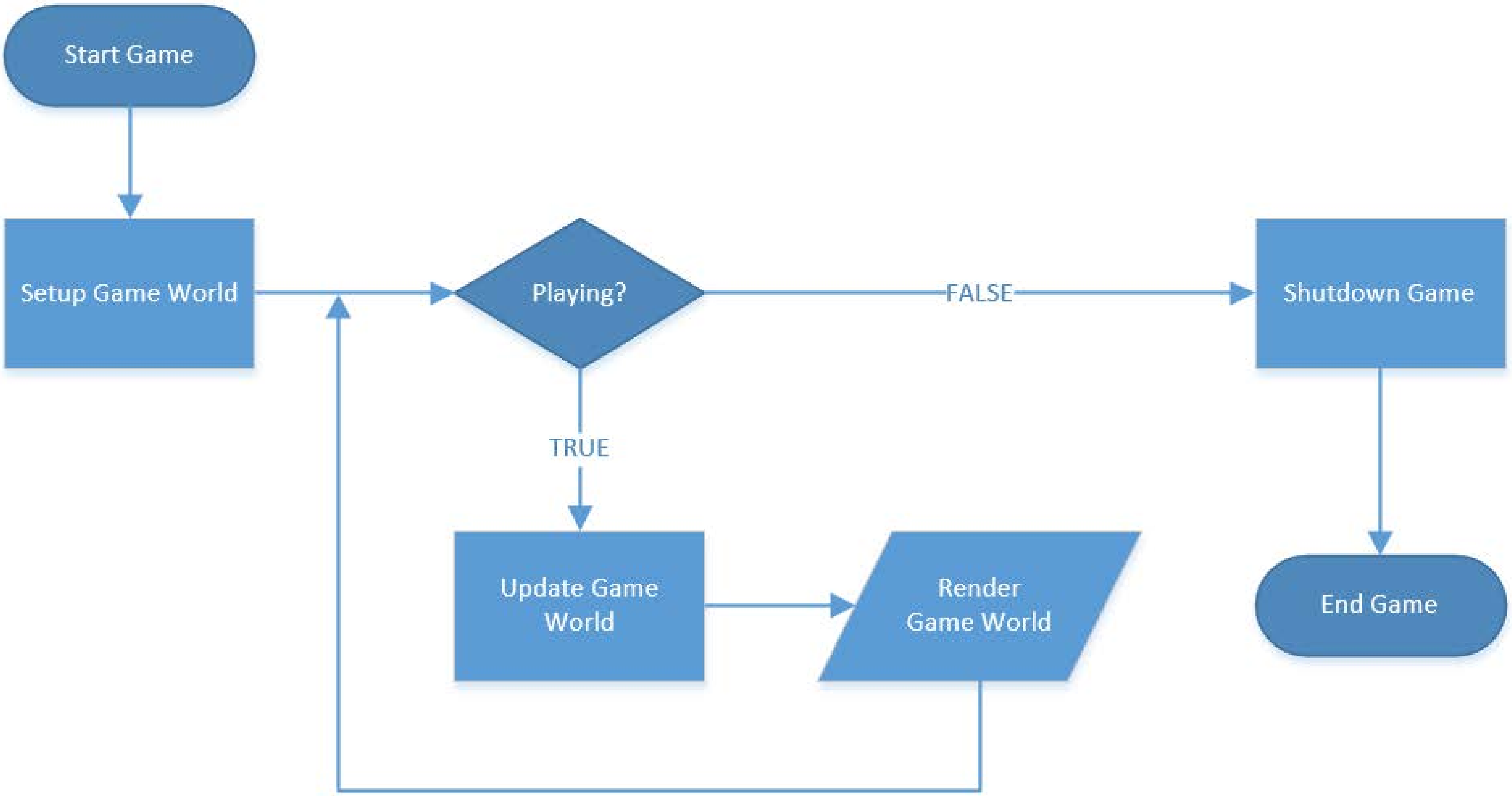
WORD wButtons (Bitmasking!), **BYTE bLeftTrigger, BYTE bRightTrigger, SHORT sThumbLX, SHORT sThumbLY, SHORT sThumbRX, SHORT sThumbRY**

Game Programming Tools

- The Game Loop
  - Interactive program: A Game!
  - Decoupling the progression of time from processor speed…
  - The game loop processes input…
    - Not blocking!
  - Processing moves the simulation forward in time.
    - By some amount of time…
  - Drawing renders the game.

Game Programming Tools

- The Game Loop



Game Programming Tools

- The Basic Game Loop

```
Initialisation(); // Setup the game world.
while (playing)   // The real-time loop.
{
    GetInput();   // Retrieve input.
    Process(dt);  // Moves the sim forward in time.
    Draw();       // Renders the updated sim.
}
Shutdown();       // Release resources.
```

Game Programming Tools

- The Game Loop continued…
  - Frames per Second (FPS):
    - The number of frames rendered, per real world second.
    - Common targets:
      - 30 FPS or 60 FPS.
  - If the game loop iterates too slowly…
    - The game becomes a slide show…
      - 1 FPS or 2 FPS…
  - If the FPS is high, then:
    - Smooth gameplay!

Game Programming Tools

- The Game Loop continued…
  - What makes frame rate?
    - How much “work” is done per frame…
      - The algorithms in the game logic!
    - Speed of the target platform… CPU, GPU…
  - Played an old game on a newer PC?
    - Old games may speed up on new hardware!!!
    - Making it impossible to play!!!
    - Old games: Fixed hardware targets…
      - Remember the TURBO button?

  - Solution: Run the game at a consistent speed!

Graphical Effects for Computer Games

- Timing
  - Target FPS?
    - 60 FPS: means 60 frames rendered per second…
    - Time per frame: 1/60 = 0.01666 seconds
      - 16 milliseconds per frame!
    - If all the game’s updating, and rendering can be done in less than 16 milliseconds…
      - Then the frame rate can be at least 60 FPS.
    - So… slow the game down…
      - Possibly… sleep the process!

Game Programming Tools

- The 60 FPS Game Loop

```
while (playing)
{
  float s = getCurrentTime(); // Seconds.
  Process();
  Draw();

  sleep(s + 0.016666 – getCurrentTime());
}
```

Graphical Effects for Computer Games

- Timing continued…
  - What if the loop iteration takes longer than 0.016666 seconds?
    - Do less work per frame!
    - Cut down on the algorithms…
      - Remove fancy graphics rendering…
      - Remove fancy physics calculations…
      - Remove fancy artificial intelligence calculations…
    - The poor game!
      - Where is the fun?

Graphical Effects for Computer Games

- Timing continued…
  - Solution: Time steps!
  - Variable Time step:
    - Time step based upon how much real world time passed since the last frame.
      - The longer the time, the bigger the time step.
    - Then update the simulation based upon the amount of time between frames:
      - Delta time!
    - However: Its non-deterministic…
      - Physics instability… networking… artificial intelligence…

Game Programming Tools

- The Variable Time Step Game Loop

```
float lastTime = getCurrentTime();
while (playing)
{
    float current = getCurrentTime();
    float delta = current – lastTime;
    Process(delta);
    Draw();

    lastTime = current;
}
```

Game Programming Tools

- Updating based upon time
  - Game Entities need to be updated with respect to time…
    - Position (2D coordinate)
    - Velocity (2D vector) (units per second).
    - Change the position per frame by the velocity…
      - Velocity vector is scaled by time!

```
void Enemy::Process(float dt)
{
    m_position += (m_velocity * dt);
}
```

Graphical Effects for Computer Games

- Timing
  - Better Solution: Fixed Time step
    - Stable for physics simulation
    - Series of fixed time steps…
    - At the start of a frame:
      - Calculate the “lag”: how much real time passed.
    - Inner loop inside the game loop to update the game world entities…
      - One fixed step, until caught up to the “lag”
    - Once caught up, draw the game again!

Game Programming Tools

- The Fixed Time Step Game Loop

```
float lastTime = getCurrentTime(); // in seconds...
float lag = 0.0f;
while (playing)
{
    float currentTime = getCurrentTime();
    float delta = currentTime – lastTime;
    lastTime = currentTime;
    lag += delta;
    ProcessInput();
    while (lag >= 0.016666)
    {
        Process(0.016666);
        lag -= 0.016666;
    }
    Draw();
}
```

Game Programming Tools

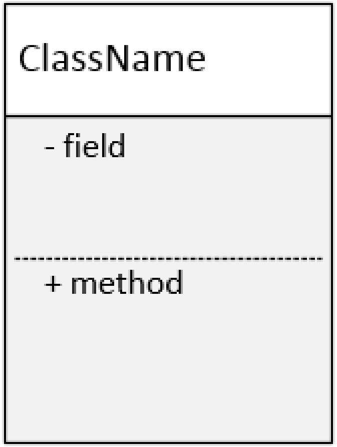
- The Update Method:
  - **`Process(dt)`** or **`Update(dt)`**
  - A chance to move the real-time simulation forward in time…
    - Game objects can simulate their behaviour…
  - Updates with respect to time…
    - How much time has elapsed between process calls?
    - Running the simulation takes time…
    - Rendering the simulation takes time…

Game Programming Tools

- UML:
  - Unified Modeling Language
    - Class Diagram
    - Use Case Diagram
    - Activity Diagram
    - Sequence Diagram
    - State Diagram
  - How familiar with these are you?

Game Programming Tools

- UML: Class Diagram
  - Class: Drawn as a box.
    - Three parts:
      - Class name
      - Member data (fields/properties)
      - Member functions (methods)
    - Visibility:
      - Public +
      - Private –
      - Protected #
    - Scope:
      - Class identifiers (static members) are underlined.



ClassName

- field

+ method

Game Programming Tools

- UML: Class Diagram
  - Relationships:
    - Links…
    - Association: ————
    - Aggregation: “Has a” part-whole relationship… ◇————
    - Composition: “Has a” life-cycle dependency… ◆————
    - Generalisation: “Is a” ————▷
      - Super (parent/base) / sub (child/derived) class relationship
    - Realisation: “Is a” implements an interface… - - - - - -▷
    - Dependency: “knows a”/”uses a” - - - - - - ->
  - Multiplicities… 0..1, 1, 0..*, 1..*, *

Exercises

- Week 2:
  - – Day 003.1 – Peer Critique: “Simple” Dice Game
  - – Day 003.2 – Peer Critique: Noughts and Crosses
  - – Day 003.3 – Space Invaders UML Class Diagram
  - – Day 003.4 – Weapon System Technical Design

Exercises

- Recommended Readings:
  - Blow, J. (2004). *Game Development: Harder than you think*. Retrieved from http://queue.acm.org/detail.cfm?id=971590
  - Stenerson, J. (2000). *A Case for Code Review*. Retrieved from http://www.gamasutra.com/view/feature/131847/a_case_for_code_review.php

Exercises

- Recommended Reference Books:
  - Madhav, S. (2013). *Game Programming Algorithms and Techniques: A Platform-Agnostic Approach*. Crawfordsville, IN: Addison-Wesley.
  - Thorn, A. (2011). *Game Engine Design and Implementation*. Sudbury, MA: Jones & Bartlett Learning.
  - Booch, G. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Upper Saddle River, NJ: Pearson Education, Inc.

Summary

- Game Programming Tools and Libraries
- Game Structure
  - Middleware and Libraries
  - Game Engines
- The Game Loop
  - Timing
- Exercises