Network Games Part III

Network Games Characteristics

- Multiple players
  - Cliques
- Multiple computers
  - Heterogeneous systems
- Span over a large area
- Each player has
  - A different agenda
  - A different style
- Players enter and leave the game
  - Timeout while in game

Implications

- Handling large volume of data
- Game accuracy
- Latency
- Fault tolerant
- Security
- Cheating detection

What Data to Send?

- Sending the entire game state (world state)
  - It is too expensive
  - Most of the data is the same
  - A player does not
    - see the whole world
    - does not care to see the world
    - Is not affected by many of the changes
  - Only local changes are unknown to the “world”
- Send what is necessary in order to keep the game going
  - User actions
  - NPC behaviour that is affected by user

Most game data is the same

- Game behaviour can be predictable
  - Maybe not to the user
- Internal game behaviour can be the same
  - Client actions
    - Game engine can simulate the expected client action
  - NPC behaviour can follow the same instruction code
- Mechanism
  - Use pseudo-random numbers from the same seed
  - Use the same decision making protocol

Problem – User’s Activity

- User’s activity does not “match” game capturing of user
  - Game guesses user actions
  - Users does not do what the game guessed
- Solution
  - Need to synchronize the user action with the game “guessing” actions
  - “Fix” the discrepancy in discreet way – smooth the synchronization action
- Other things
  - Errors in engine – game worlds are different on some game copies
  - Small errors can lead to a large change in behaviour (e.g., error in the random number gen.)
  - Errors in messages

Prediction

- Waiting for input is slow (show must go on)
  - Input from player
  - Input from other players (P2P, server)
- Clients predict next game “state”
  - Mainly motion prediction
- Discrepancy
  - Send corrections when things are wrong
  - Sends corrections periodically

Prediction of a client

Client Server (asynchronous)

- Prediction is carried out by server
- Light client
  - Server receives player’s action (delay)
  - Server sends an update to client
  - Server sends an update to all players
- Heavy client
  - Server receives client action (delay)
  - Server compares against prediction engine
  - Server sends a prediction update to all players (including client)

Peer to Peer

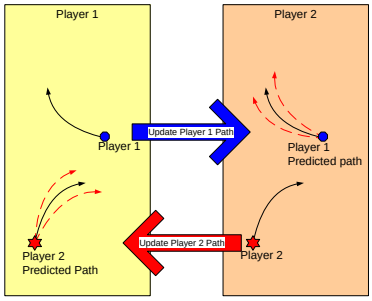
- Prediction is carried out by each client
- Client
  - receives player’s action
  - Client compares against prediction
  - Client sends an update to all player

Dead Reckoning - Prediction

- Each game entity simulates its own behviour
  - Position
  - Velocity, acceleration
  - Path taken
- Update behaviour
  - When user acts (own machine)
  - Receives an update from other machines

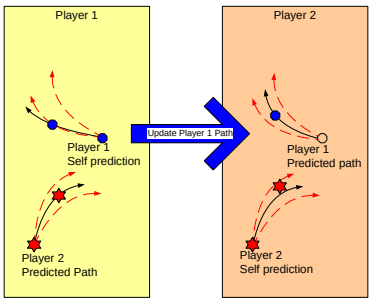
Dead Reckoning

Update path every 5 seconds



Dead Reckoning

Update – predicted path is incorrect

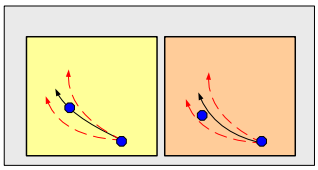


Dear Reckoning - Operation

- Receive latest update
  - Update copy of other players (e.g., vehicles)
  - Predicts location of other players
    - Using new data
- Update own location
  - Update true location and predicted location
  - Check if within prediction range
    - Send update location/path if needed
  - Sends update if timeout (predefined interval)

Dead Reckoning Requirments

- Create two copies of self
  - Accurate copy (current true location)
  - Prediction copy (location based on prediction)



Dead Reckoning - Latency

- Latency affects the prediction quality
  - Lager latency ➔ Worse prediction
- Prediction correction
  - Large discrepancies cause jumps
  - Visual jumps are noticeable and not pleasing
- Solution
  - If possible interpolate between correct path and current location
  - Converge to the correct path instead of an abrupt jump

Example

Sending Data

- Determine what data is needed
  - Create your own data structures
  - Create your own protocol (messages)
- Add some security features
  - Message sequence
    - Sequential
    - Sequence of random numbers

Creating a Protocol

- A message consists of
  - A message header
    - Standard – independent of message
      - "all game player should be able to understand it.
    - Predefined size
    - What should it include?
  - A message body
    - Content is message dependent
    - Variable size (can be fixed size)
  - Possible Issues
    - Security
    - Versions
    - Heterogeneous Hardware

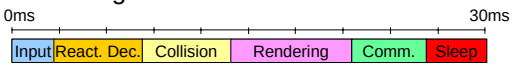
Message Security

- Encryption
- Random message sequence
  - Decide on a seed on the fly
- Self encryption
  - Random XOR
  - Arbitrary message content
- Checksums
  - Full
  - Partial

Player 1
True locati

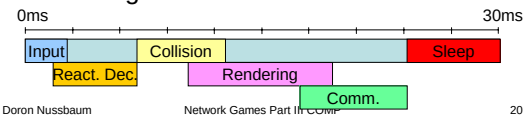
Handling Messages

- Game must address
  - Game play (decisions, collisions, behaviours)
  - I/O – graphics, sound,
  - Communication
- Assuming 30hz frames



Handling Messages

- Game must address
  - Game play (decisions, collisions, behaviours)
  - I/O – graphics, sound,
  - Communication
- Assuming 30hz frames



Threads

- Can is a path of execution of a code
  - Often is referred to as a “light process”
- A process is a “place holder for threads”
- Attributes
  - Has its own stack
  - Has its own “program”
  - Can share memory with other threads
- One main thread

Issues

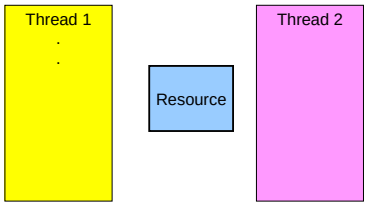
- Synchronization between threads
  - Race conditions
  - Priorities
  - Starvation
- Addressing it
  - Scheduling
  - coordination

Coordination

- Semaphore
  - Used for intercrosses communication
  - Usually when two or more resources are used
  - Attempts to book a resource (e.g., file, printer)
- Mutex
  - Used between threads
  - Used between processors
  - Usually to “lock” a single resource for a short time
- Critical area
  - Used within a single process
  - Used to communicate between threads

Coordination of Threads/Processes

- Acquire the “resource”
- Use it
- Release it



Common Causes for Blocks

- No release of a a sync object
  - No call to an unlock operation
  - Unlocking the wrong object
  - Missing an unlock operation due to changes in code path
- Resource Consumption
  - No clean up the object
  - Not calling DeleteCriticalSection or CloseHandle
  - Resource consumption
  - Keep initializing new synch objects
- Locking too early or unlocking too late
  - A good general rule is to lock for as short a time as possible and lock the required resources only.

Mutex

Critical Section

Creating Threads

Security

- Avoid acceptance of resent packets
- Server sends new random seed