Gameplay

Introduction

- What do we mean by gameplay?
  - Interaction between the player and the game
    - The distinguishing factor from non-interactive media like film and music
  - Sometimes used interchangeably with “game mechanics”
  - This is where fun lives
- Gameplay components
  - World representation
  - Behaviour simulation
    - Physics
    - AI
  - Camera

Levels of gameplay

- Second-to-second
  - Where is this missile moving next frame?
  - Has the enemy parried my attack?
  - “The simulation”
- Minute-to-minute
  - What is the players current objective?
  - “The mission script”
- Hour-to-hour
  - What skills have I unlocked?
  - Empire-building
  - “The meta game”

World Representation

- This is what changes as a result of player interaction
  - The AI also needs to keep track of what is going on in the game
- A very simple example:
  - We can represent a tic-tac-toe board as a two dimensional array of characters
- A slightly less simple example:
  - Pac Man consists (minimally) of the locations of Pac Man and all the ghosts, locations of the walls, and positions of the active pellets
- More complicated games typically do not have a fixed set of game entities
  - Need a dynamic data structure to manage entities

World Representation Requirements

Some important first questions:

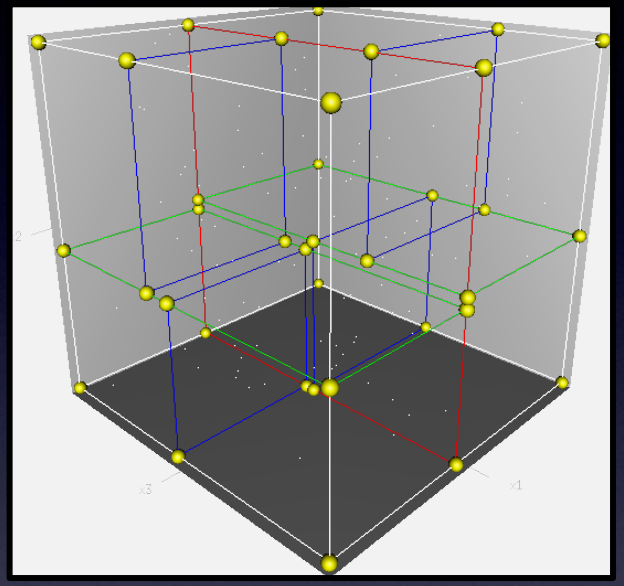
- How large is the world?
- How complex is the world?
- How far can you see?
- What operations will be performed?
  - Visibility
  - Audibility
  - Path finding
  - Proximity detection
  - Collision detection
  - Sending messages to groups of entities
  - Dynamically loading sections of world
    - If so, how fast can you travel?

World Representation: Lists

- Simplest approach: one big list
  - All search operations are pretty expensive
  - But all operations are about the same
    - i.e. no slower to search by name than by position
  - Storage space and algorithm complexity are low
  - Good for extremely simple games (< 100 entities)
- Can make it a little more useful with multiple lists
- In more complicated structures each world node will have a list of entities in that node

Spatial World Representation

- Spatial data structures
  - K-D trees
  - BSP trees
  - Grid
  - Graph
  - Whatever
- Dictionary
  - Spatial hashing
- Hybrid
  - Big games use multiple techniques at the same time
  - or different techniques for different kind of data
  - each optimised for the particular queries on that data



Sphere of Influence

- Rather than simulating thousands of entities in a large world, many games maintain a small bubble of activity around the player
  - Or around the camera
  - Could be somewhat off-centre
- Keeps the activity centered around the player
- The world outside the sphere is downgraded in fidelity, or shut off entirely
  - Typically multiple spheres for different types of entities
  - Typically tied to level-of-detail (LOD) systems
- Entities can be recycled as they leave the sphere
  - Strive to recycle objects that aren't visible
  - Or fade them in/out gently in the distance

Entity Behaviour

- We want our entities to do interesting things
- Two major strategies employed:
  - Scripted behaviour
    - as in acting, where an actor follows a script
    - good for drama
  - Simulated behaviour
    - let the rules of the world do their thing
    - good for novelty
- For example, consider the FPS cliché of the exploding barrel
  - How do we model this behaviour?

Scripted Behaviour

- Explicitly add individual behaviours to entities

```
function barrel::collide(hit_by)
    if hit_by.type == bullet
        damage += 10
        if damage >= 100
            PlayAnimation(exploding)
            PlaySound(exploding_barrel)
            DestroySelf()
        end
    end
end
```

Comments

- Simple to implement
- Good for one-off, unique game events
  - Cut-scene triggers
- Not flexible
- Misses out on emergent opportunities
  - No chain reaction explosions
  - Doesn't explode when hit by rockets
    - unless explicitly modified to do so
  - No splash damage
- Extending this model to complex interactions makes the code unwieldy
  - Numerous permutations have to be explicitly coded

Simulated Behaviour

- Define a few high-level rules that affects how objects behave
- Combine these rules in interesting ways when objects interact
- Properties of objects:
  - GivesDamage(radius, amount)
  - MaxDamageAbsorb(amount)
    - Object will “break” if it absorbs enough damage
  - BreakBehaviour(disappear | explode)
    - Disappear destroys entity
    - Explode destroys entity, and spawns a shock wave entity in its place

Entity Properties

- Entities in this example, and their properties:
  - Bullet
    - GivesDamage(0.01, 10)
    - MaxDamageAbsorb(0)
    - BreakBehaviour(disappear)
  - Barrel
    - MaxDamageAbsorb(20)
    - BreakBehaviour(explode)
  - Shockwave
    - GivesDamage(5.0, 50)

Explosion Behaviour

```
function entity::collide(hit_by)
    damage += hit_by.GivesDamage.amount
    if damage > MaxDamageAbsorb
        switch BreakBehaviour
            case disappear:
                DestroySelf()
            case explode:
                Spawn(shockwave, my position)
                DestroySelf()
        end
    end
end
```

Comments

- Observed behaviour the same as the first example
  - when a lone barrel is shot
- A lot of nice behaviour can emerge
  - Cascading barrel explosions
  - Non-bullet objects causing damage can be added easily
  - Splash damage
- Easy to add new properties and rules
  - A rocket is just a bullet with BreakBehaviour = explode
  - Different damage classes
    - e.g. electrical damage that only harms creatures, but doesn't affect inanimate objects
  - CanBurn, EmitsHeat properties with rules for objects bursting into flames
- It doesn't take many of these rules to create a very rich environment
  - Be careful about undesired emergent behaviour

Triggers

- Very common way to initiate entity behaviour
- Common types:
  - Volume
  - Surface
  - Time
- When the trigger condition is met (player occupies trigger volume, timer runs out, etc.):
  - Send event
  - Run script
  - Execute callback
- Triggers can be
  - One-shot
  - Edge-triggered
  - Continuous
- Games typically have a well developed trigger system available

AI

- Video games use a unique definition of AI
  - A lot of what games call “AI” isn't really intelligence at all, just gameplay
  - AI is the group of algorithms that control the objects in the game
  - It is the heart of the game, and often has the most influence on how much fun the game is
- Making the AI “smart” is not the hard part
  - The game is omniscient and omnipotent, so it can always kick your ass if it chooses to
  - The trick is in making AI that is challenging yet realistically flawed

State Machines

- State machines are used to control moderate to complex AI behaviour
- Often implemented in an ad-hoc manner with a big switch statement
  - Fine for relatively simple behaviour
- Commonly implemented in script
  - Still just a switch statement with lots of syntactic sugar
- Or you can build a graphical state machine editor
  - Supporting nested machines
  - With event handling
  - Transition scripts
  - Etc

Mapping Events to Behaviours

- The AI interprets a button press as an intention to perform a certain action
  - Call a function, run a script, set a variable
- Often this is a simple mapping, but it can become complex depending on the game
  - For example, some games have camera-relative controls
  - Fighting games require queueing of inputs for combos
- There are constraints on allowable behaviours
  - These constraints can be quite complex
    - Physical constraints
    - Logical constraints (rules)
      - E.g. conditions on state transitions

Physics

- What do we mean by physics
  - Rules by which objects move and react in the gameplay environment
    - Q: But isn’t this the same as AI?
    - A: To a large extent it is
- Physics doesn't necessarily imply a sophisticated rigid body dynamics system
  - Pong modelled ideal inelastic collisions pretty well
- In fact, “real physics” is usually just a tool in the box
- Game physics implementers have considerably more latitude to change the rules
  - A lot of physics can be “faked” without going to a dynamics engine
- How is physics used in a modern game?

Uses of Physics

- Collision detection
  - Detect interactions of entities in the environment, e.g. triggers
- Animation
  - Complex shapes and surfaces (chains, cloth, water)
  - Realistic environment interactions (bounce, tumble, roll, slide)
  - Reaction to forces (explosions, gravity, wind)
  - Augment “canned” animation with procedural animation
  - Hit reactions, “rag doll”
- Gameplay mechanics
  - Physics puzzles
  - Driving, flying
  - Damage calculation
- Sound triggering

AI Use of Physics

- Generally the AI keeps the physics system reigned in
  - Objects only go into “full simulation mode” under specific circumstances and often only for a limited period of time
- Example from Prototype and Cyberpunk 2077:
  - Traffic cars generally slide around the world "on rails"
  - If an object appears in the car's “visibility cone”, it comes to a gradual stop
  - Traffic cars in “rail mode” can impart forces on other objects (peds)
  - If the car collides with another car, the AI puts them into full simulation
    - AI computes an impact force based upon collision information, and tunables
    - Car is placed under control of the rigid body system, and allowed to bounce around until it comes to rest
    - Then the car is put to sleep (removed from rigid body system)
    - If it’s damaged it never returns to AI control

Interactions Between Objects

- So, some objects are under physics control, while other are under AI control
- What happens when they collide?
  - To the physics system, AI controlled objects don't follow the rules
    - Velocities, positions are under AI control
    - Properties like mass, and friction, and restitution may not be defined for AI controlled entities
- There needs to be a mechanism to compute plausible forces to pass to the physics system to apply to the simulated object
- Likewise, the AI controlled object will have some sort of collision response programmed into it
  - Play animation, move object, apply damage, trigger sounds, change entity state, etc.

Physics Hand-off

- When the AI places an object into full simulation, it sets up the initial conditions for the object's rigid body
  - Position, velocity, angular velocity
- In the simplest form, the AI-managed position and velocities are copied into the rigid body
- There may be considerable massaging of the conditions to make the response more interesting, or realistic-appearing
- Example from Hulk 2:
  - When Hulk elbows a car, angular velocity is carefully chosen to make it launch up into the air, tumble end-over-end (with variation), and land close behind him
  - Thrown objects are kept out of general physics simulation until they hit something

Tuning

- Physical simulations can produce a lot of emergent behaviour
- This can be good
  - Adds variety to gameplay and presentation
  - Players can discover or create situations that weren't envisioned by the designer
- This can be bad
  - Players can discover or create situations that weren't envisioned by the designer
    - Exploits, bugs, other bizarre behaviour
- Emergent systems are hard to tune!

Realism, Accuracy and Fun

- Realism is a powerful tool, but it is not the end goal for video games.
- There are differences between what people believe is realistic, and real-world behaviour.
  - “Real” realism is usually pretty boring
- Games provide a small amount of feedback and control compared to their real-life counterparts
- Physical simulation and hacks can live comfortably side-by-side.

Cameras

- AI camera models are usually motivated by:
  - Gameplay goals
    - Need to see player
    - Need to see important AI entities
    - Intuitive controls
  - Cinematic goals
    - Show what the player needs to see
    - Look cool
- Camera design is primarily an AI / gameplay issue
  - Not rendering!

Camera Models

- Simple camera models:
  - Fixed: the camera never moves
  - Tracking: the camera doesn't move, but points at an interesting object
  - Follow: the camera follows at a distance behind the target
- More sophisticated camera systems handle things like:
  - obstacle avoidance
  - framing
  - line of sight
- Instant replay camera:
  - Scripted camera animations
  - User controlled cameras
- Artists controlled cameras
  - Shot setup and animation done in 3D modeling/animation tool

Camera Models for Driving Games

- First-person
  - Glue camera to the bumper
  - Tune field-of-view to create enhanced sense of speed
    - More effective than tuning actual vehicle speed
- Third-person
  - Camera tracks behind the car at some distance
  - Perfect tracking doesn't look good
    - Car is locked to the centre of the screen
  - Add lag and anticipation to the camera movement
    - When braking, move camera closer
    - When accelerating, move further
    - Look into direction of turns
    - Lower/raise camera based on velocity of car
    - Don't spin camera right away if the car is spinning

Summary

- Gameplay is huge
  - Touches on every part of the system
  - Needs every trick from the bag
- Many areas weren't covered
  - Path finding
    - Will talk about it in the driving lecture
  - Enemy AI
    - Too wide and game-specific to cover here
  - AI animation control