Collision detection & physics

- What and why?
  - Want to model collision (and other physical effects) in games
  - E.g. bouncing balls, walls, sliding, gravity
- Modelling collision
  - Which collisions to check for?
  - How detect collisions?
  - How handle collisions?
    - Consequences in the game world of the collisions
- Book reference: chapters 11 and some of chapter 5

Collision detection goals

- A collision detection scheme for a particular game:
  - Discard as many objects as possible which cannot collide
    - Slow to examine every object in a large game world
  - Quickly decide whether objects collide
  - Use collision detection which is accurate enough
    - Trade-off between precision and speed
  - Does not annoy the player
    - Better to use smooth motion effects etc.

Basic collision detection

- Each game update, do:
  - For each object in the game world
    1. Update the object's location
    2. Check for collisions with other objects or the environment
    3. If a collision is found, revert the object to its previous location
- Do you understand the game world consequences of this?
  - Imagine two balls moving towards each other with the same speed; what would happen when they collide (with this basic scheme)?

Tile world example

- Setting: a 2D tile world game. The player can walk around and jump. If they touch evil monsters (sprites), they die...
- Assume that each sprite fits in one tile
- Then collision detection can be roughly:
  - Let (s.x, s.y) be the pixel coordinates of sprite s' bottom-left corner
  - boolean isCollision(Sprite s1, Sprite s2) {

```
return
    s1.x < s2.x + s2.width() && s2.x < s1.x + s1.width() &&
    s1.y < s2.y + s2.height() && s2.y < s1.y + s1.height();
}
```

Tile world example

- Setting: a 2D tile world game. The player can walk around and jump. But we want to forbid moving through walls, ceilings and floors...
- Essentially, every update we must check whether the player (a sprite) collides with some hard obstacle; and if so, change their position accordingly

Tile world example

- Detecting collisions with tiles
  - Assume that the player's size is one tile
  - The player can be in 1 up to 4 tiles at any time (we don't assume that movement is aligned with tiles – e.g. jumping/falling is allowed)
  - The player may move over many tiles between two updates
- Given the player's old and new position, we want to know if they hit any solid tile during the movement
- So how would you implement this?

Tile world example

- Consider this pseudo-Java implementation:

```
Point getTileCollision(Sprite s, Point new) {
    float fromX = min(s.x, new.x), fromY = min(s.y, new.y);
    float toX = max(s.x, new.x), toY = max(s.y, new.y);
    int fromTileX = pixelsToTiles(fromX); // divide by tile size
```

(and analogously define tile coordinates fromTileY, toTileX and toTileY here)

```
    for (int x = fromTileX; x <= toTileX; x++)
        for (int y = fromTileY; y <= toTileY; y++)
            if (tileMap.getTile(x, y).isSolid())
                return new Point(x, y)
}
```

- What can be improved here?

Tile world example

- If the player collides with a solid tile, we set their position so that they are not inside the tile (next to it will do)
- How do we handle collisions with several tiles at once?
- We can split up the movement into horizontal and vertical components and handle collisions separately
  - Then there is at most one collision per component
- If we do this, the implementation we described works
  - But if the time between updates is high, objects may pass through each other...

A discretization problem

- The discrete time issue:
  - What if objects collide between updates?
    - Imagine the following 2D scene: a black ball is standing still, and a white ball moves quickly towards the black ball
    - If the white ball can cover more pixels in one frame than the black ball's diameter, we may not detect the collision
  - Solution: interpolate between frames
    - Check some positions which are passed between frames
    - Again, we have a precision v.s. time trade-off

Eliminating tests

- Which objects can collide with each other?
  - If an object doesn't move, it won't hit other objects
  - An object can only hit objects which are relatively close to it
    - Assuming normal/predictable movement (e.g. no teleporting :)
- How implement *relatively close*?
  - Divide the world into a grid
  - An object can only hit objects in the same or neighbouring cells

Quick collision detection

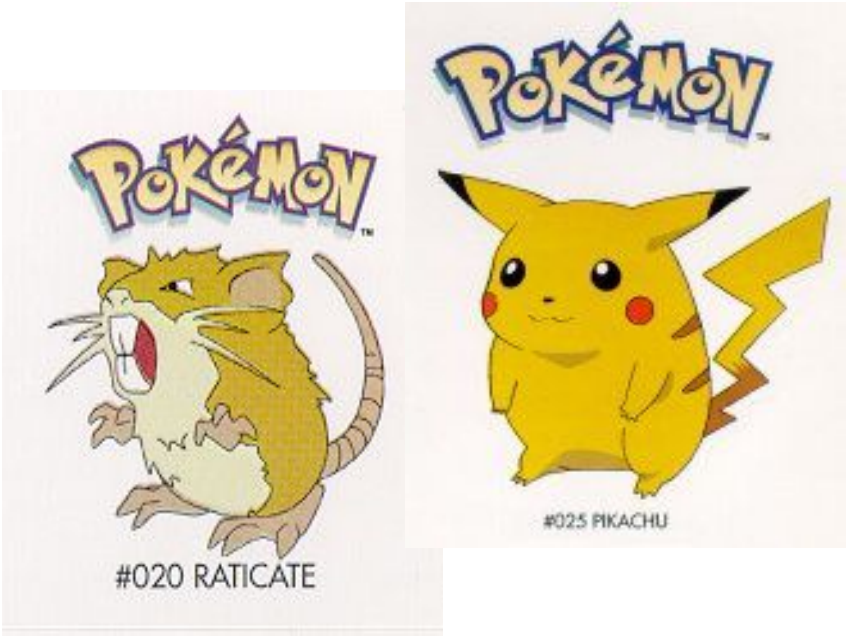
- Bounding spheres (circles in 2D)
  - Surround the objects closely by spheres
  - Collision detection as: if two object's spheres collide, we say that the objects have collided
- Improving precision for bounding spheres:
  - Use another level of spheres: surround the object by a union of several (small) spheres
  - Collision detection is then stepwise: if the first level spheres collide, check the second level spheres
  - You can add more levels for more precision

Quick collision detection

- Bounding cylinders (rectangles)
  - Upright cylinders are more precise for tall and thin objects than spheres are
  - Also quick to check
- You can now imagine a flexible general scheme:
  - For each object, use a simple geometrical figure which fits well around it (like an upright or lying cylinder or a sphere)
  - Improve precision by adding more levels (at each level choose a new nicely fitting figure)

Bounding rectangle example

- The objects collide if the bounding rectangles overlap
  - Here the test is imprecise: the rectangles overlap, but the objects don't actually touch...



Collisions with the environment

- Game worlds may be represented in a way which allows for more efficient collision detection with the environment
  - For example, if a 2D world is split up into tiles, we get exact bounding rectangles for walls, ceilings and floors for free. The project game has tiles.
  - If you are interested in representation of 3D worlds, you will find more info in the book (and you might want to take our Computer graphics courses :)

Nice collision handling

- If we just stop the player when it hits a solid object, it may be annoying for them
- Collision handling which does not annoy the player
  - Smooth effects, like sliding along walls etc.

Object-to-object sliding

- Sprites smoothly sliding off each other when colliding, instead of stopping
- How implement this?
  - Move the moving object the least distance away from the non-moving object so that their boundaries no longer overlap

Object-to-wall sliding

- Where should a sprite slide end up when sliding against a wall?
  - Direction: slide parallell to the wall :)
    - the direction vector is illustrated in fig. 11.14 in the book
  - Distance: computed by projecting the vector from the collision point to the goal point in the direction of the slide
- But this implementation gives jerky sliding

Smooth sliding and gravity

- Sliding upwards on stairs/upwards leaning walls can be nicely done by applying an acceleration on the sprite
- Falling can be implemented by actually applying gravity (surprise)
  - Simply add a downward acceleration to the sprite when its velocity is updated

Physics of jumping

- Jumping is quite simple once you have gravity
  - Just apply an upward velocity (and let it diminish as usual by gravity)
- If you want to control the height of a jump, you can compute the upward velocity which is needed for that
  - By the energy conservation laws (if they are called that) we have:
  - mgh = mv^2/2 (v is the upward velocity, h is the height)
  - which reduces to: v^2 = 2gh

Physics of collision

- We have good physics models thanks to Newton (and probably other scientists as well)
  - Friction, 3D, different weights and materials, many objects colliding
- Let us make things simpler for us
  - No friction, 2D, no mass (ok, mass 1), no weird materials, only 2 objects colliding at any one instant
- The basics of wall bounces: it's just reflection :)
  - angle: as if a ray from the ball in its velocity direction was reflected from the wall
  - size: unchanged – by momentum laws

Physics of collision

- What about ball to ball collisions?
- We can model it as two collisions with walls
  - (or we could compute the force components :)
  - the impact is along a vector normal to both ball surfaces at the point of collision
  - the "wall" which the balls bounce against is perpendicular to that normal vector
- So now we know simple 2D bouncing
  - But it is too simple for many games
  - Especially for pool: we really want friction and rotation (and probably 3D :)