Lecture 5

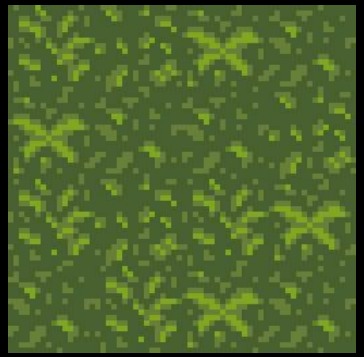
Announcements

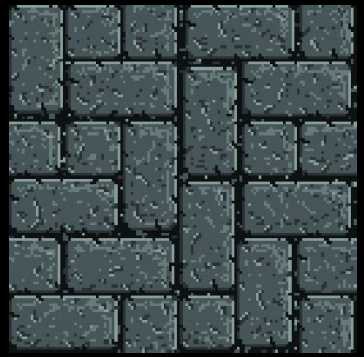
Wiz 2 Feedback

- TAs need to be able to beat your game to grade it
  - Add an easy mode
  - Let us cheat

Wiz 2 Feedback

- Doorways can be narrow, make sure you can easily traverse your dungeon
- Diagonal movement is cool
- Use sprites for your tiles!





Mid-Semester Feedback Forms

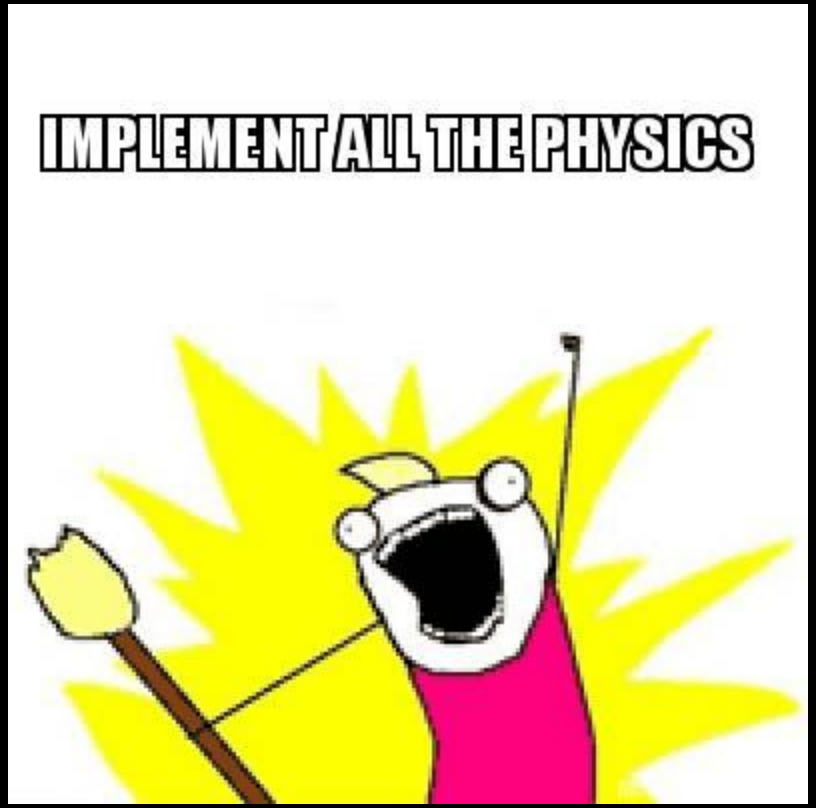
- Tell us how we're doing
- Please fill them out!
- Important for the rest of the semester
  - Also next year

Sunday Hours

- Is Sunday 8-10 more convenient than Saturday?

Nin: Your last (solo) project

- Platformer game
  - Physics
  - More physics
  - Externalizing game logic



Announcements

QUESTIONS?

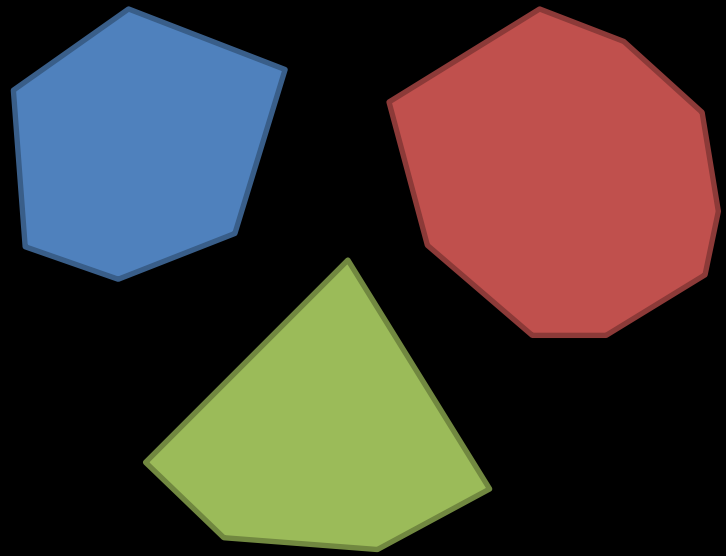
Lecture 5

Hang in there!

Collision Detection II

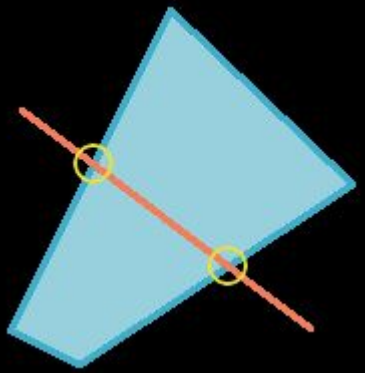
Polygons

- Lots of places that could collide (overlap)
- Irregular shapes
- Can’t test every point
  - What if both polygons are huge?
  - Can get false positives if pixels are close together

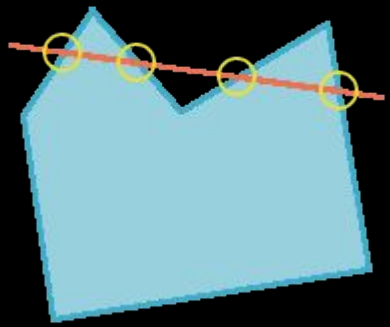


Polygon Definitions

- Polygon
  - Bounded by 3 or more straight line segments (edges)
  - Edges meet at vertices
- Convex Polygon
  - Every interior angle < 180°
  - Any line through the shape crosses only twice
  - Boundary never crosses itself
- Concave Polygon
  - Interior angle > 180°
  - Line through the shape crosses more than twice



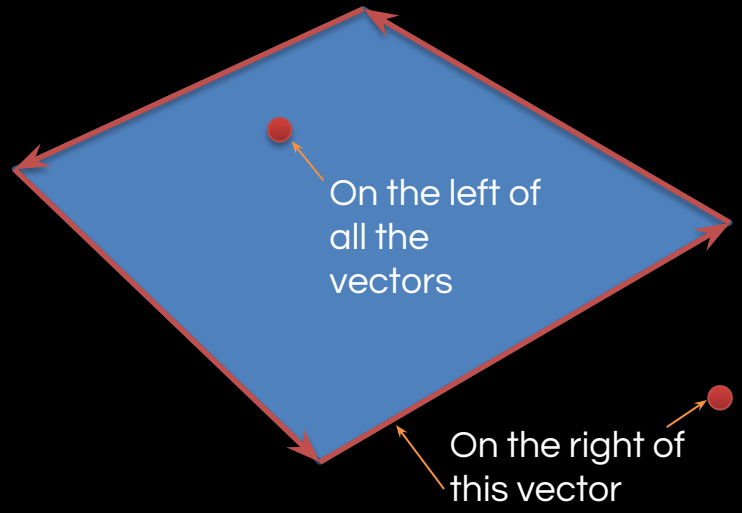
Convex Polygon



Concave Polygon

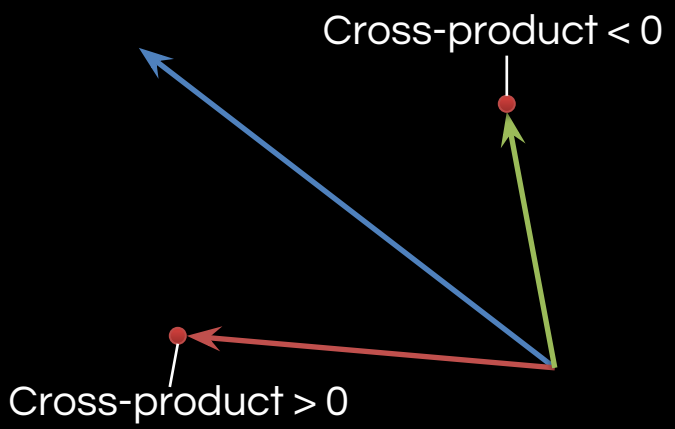
Point in Polygon

- Think of the border of a polygon as a path of vectors
  - Counterclockwise order!!!
- For convex polygons, points in the interior will be on the same side of all the vectors



Point in Polygon

- To determine what side of a vector *v* a point is on:
  - Draw another vector *p* from the base of the vector to that point
  - Take cross product $v \times p$
  - If result is negative, it's on the left

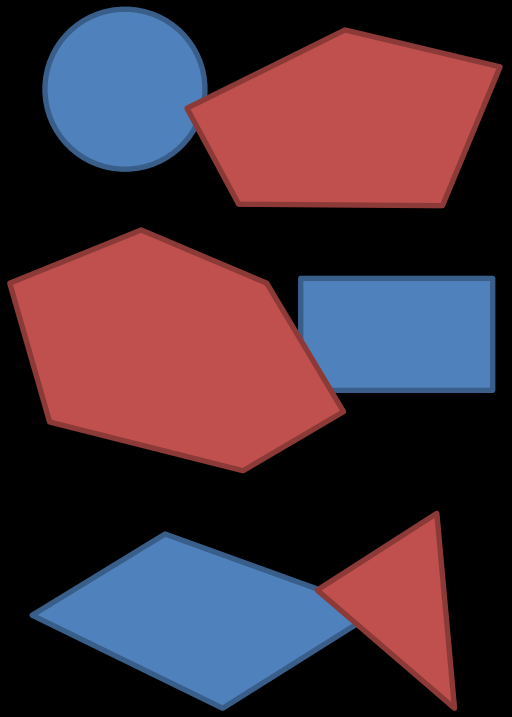


Point in Polygon Algorithm

1. Iterate over the edges (counterclockwise)
2. Construct a vector v along the edge
3. Construct a vector p to the point
4. Take the cross-product $v \times p$
5. If all cross-products are negative, point is inside
6. If any cross-product is positive, it's outside

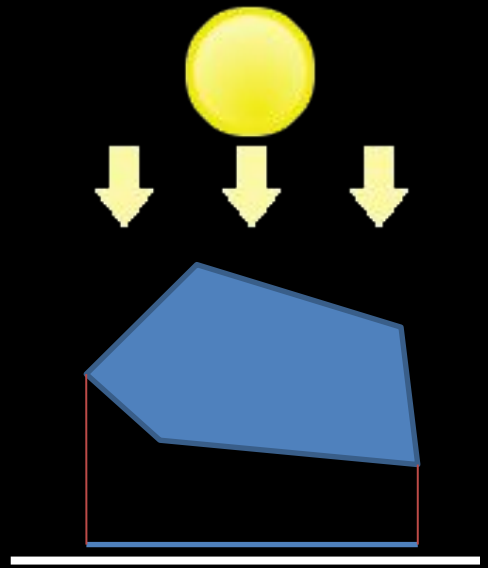
Shape-Polygon

- What about Circle-Polygon, AAB-Polygon, Polygon-Polygon collisions?
- Can all be handled with the same algorithm
- Based on Separating Axis Theorem



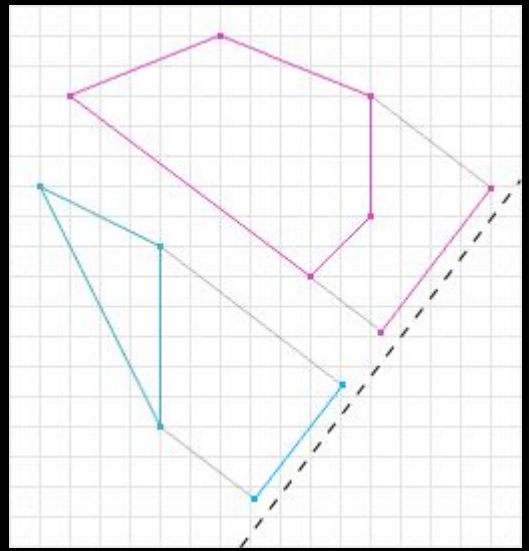
Projection

- Imagine a light source with parallel rays
- Shape is between light source and axis
- “Shadow” cast on axis is shape’s projection onto that axis



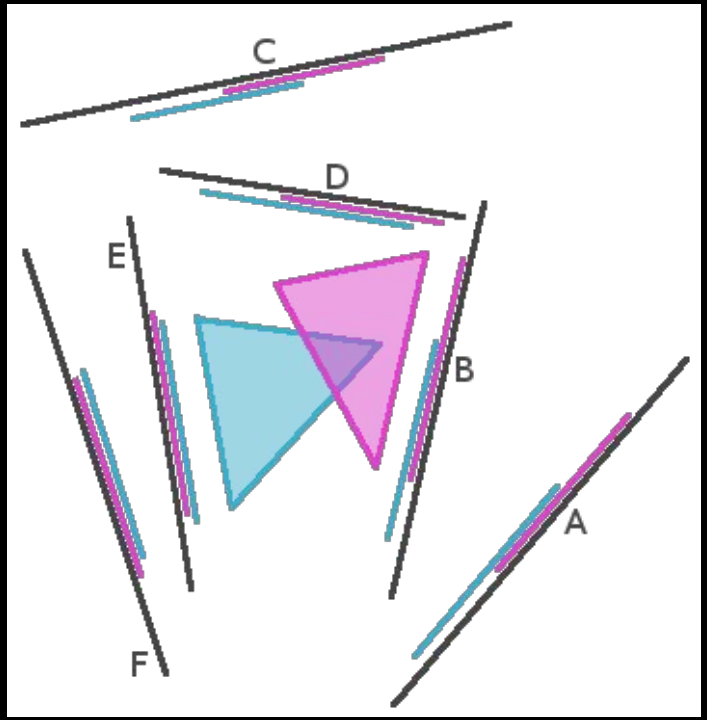
Projection

- Dashed line is the axis
- Red and blue lines are projections of the shapes



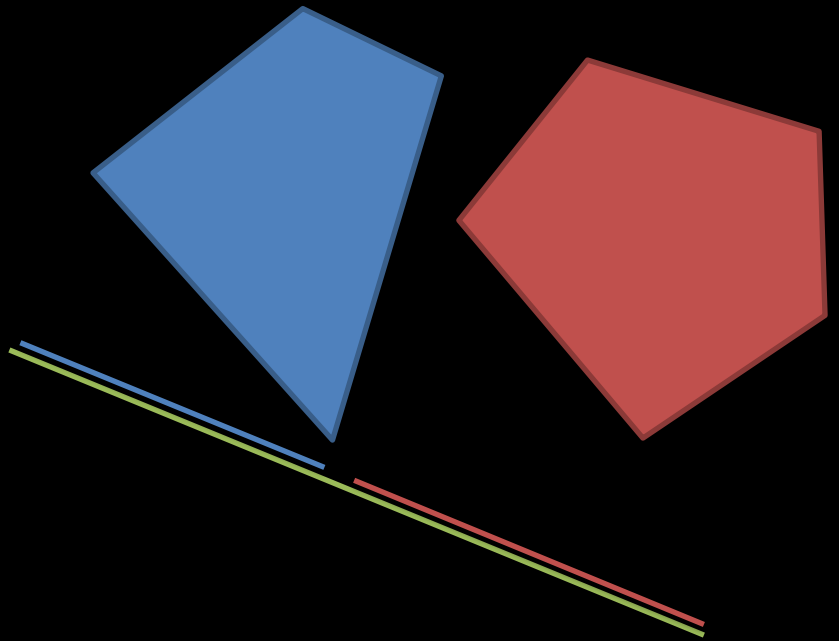
Intersection

- Shapes are intersecting when ALL possible projections overlap
- No matter what direction you look from, you can’t see between the shapes



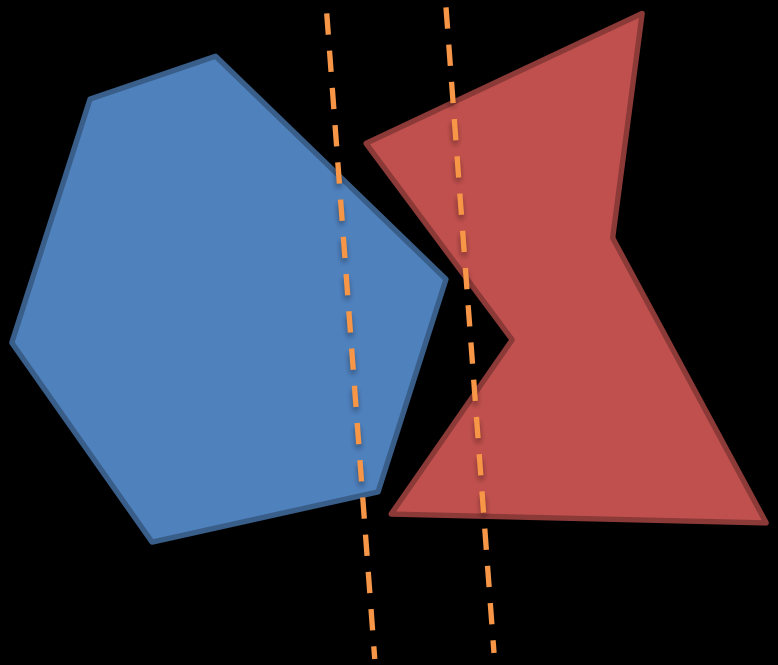
Separating Axis Theorem

- If two convex shapes are not overlapping, there exists an axis for which the projection of the shapes will not overlap.
- If we can find an axis along which the projection of the two shapes does not overlap, then the shapes aren't colliding.



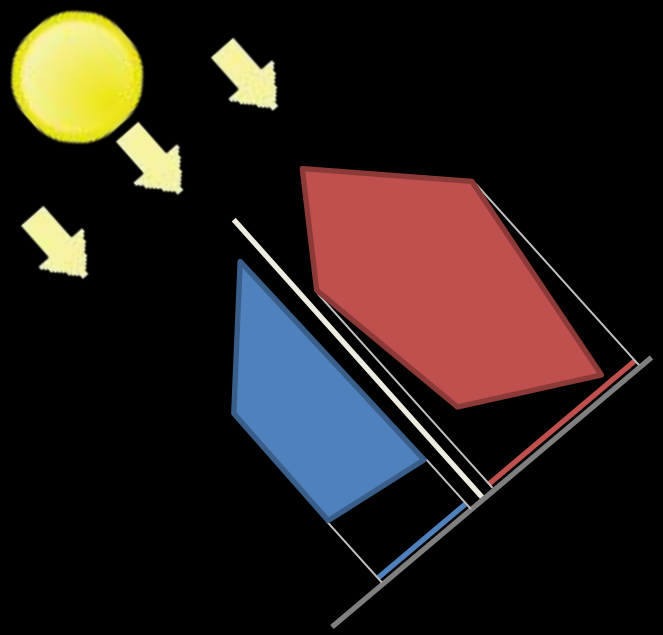
Caveat: Only Convex Shapes

- SAT only applies to convex shapes
  - Can't draw a line between concave shapes, therefore no separating axis
- Don't need to support concave polygons
- Compound shapes can be concave, but each component is convex



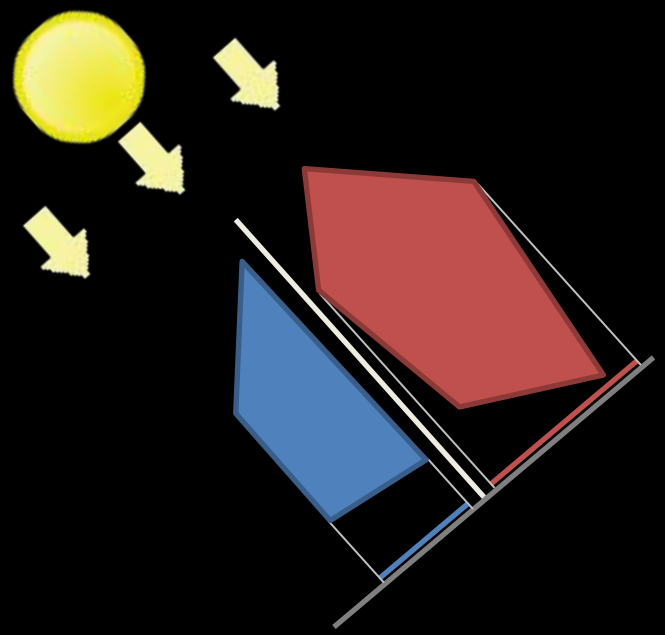
Separating Axes

- Consider an axis on which projections do not overlap
- There's a line perpendicular to it that goes between the shapes
- This is the line of sight



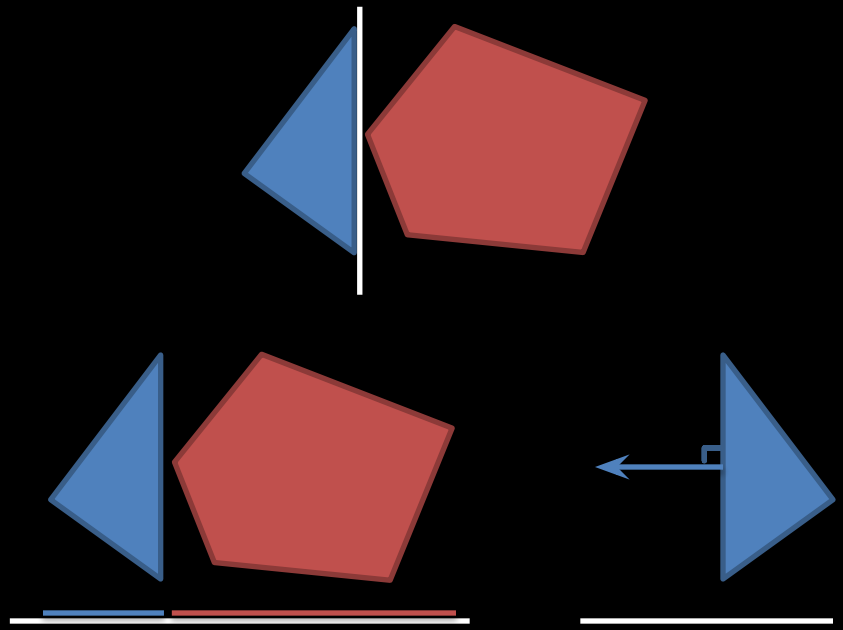
Separating Axes

- Line of sight = a line that can be drawn between two separate shapes
- Separating Axis = axis perpendicular to that line, onto which shapes are projected



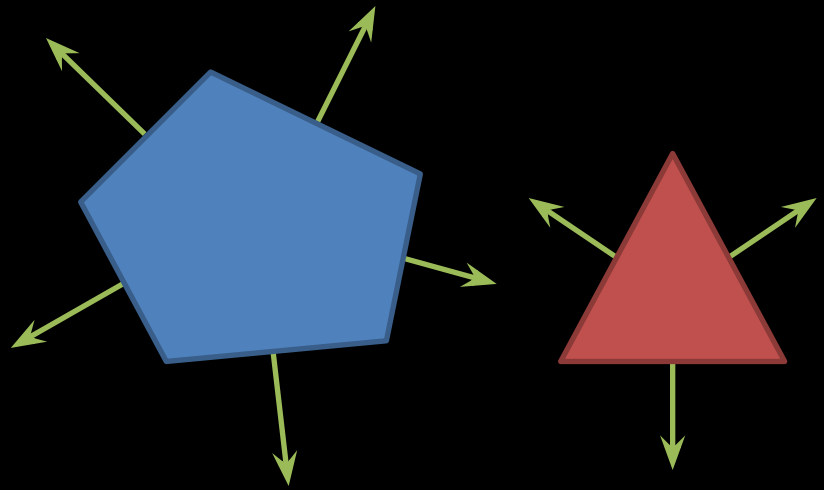
Finding Separating Axes

- If shapes are very close to colliding, a line parallel to an edge will always get through
- Minimal separating axis is always perpendicular to a shape's edge



Finding Separating Axes

- Lines perpendicular to each shape edge are all the separating axes you need
  - Sometimes called “edge normals”
- Consider each edge a vector, take perpendicular

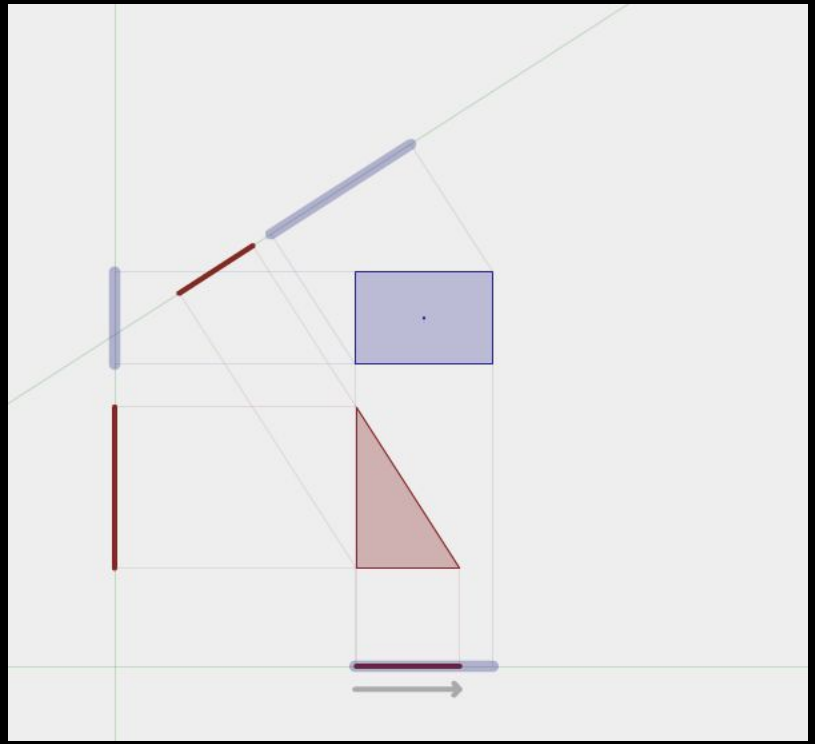


General SAT Algorithm

1. Create a vector for each edge of each shape
2. Take the perpendicular vector to get a separating axis
3. For each axis, project both shapes onto it
4. If the projections don’t overlap, no collision
5. If the projections overlap on every axis, the shapes are colliding

SAT

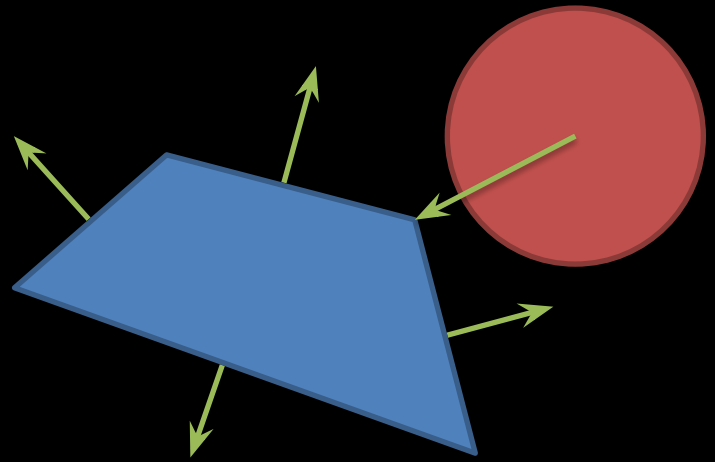
- Thin green lines are separating axes
- Red lines are projections of triangle, blue lines are projections of circle
- When shapes intersect, all projections overlap



www.metanetsoftware.com

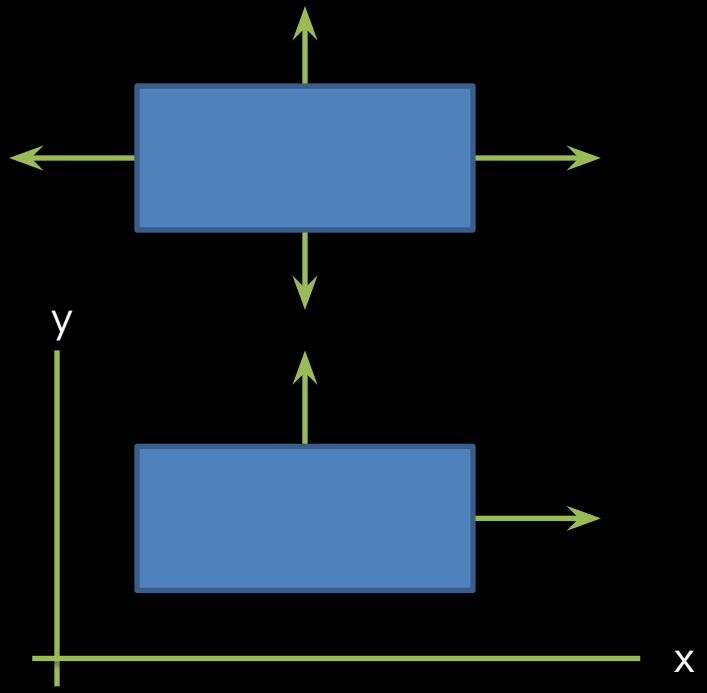
Special Cases: Circles

- What's perpendicular to the edges of a circle?
- Take the vector from the center to the closest vertex of the polygon
- No perpendicular – this is a separating axis



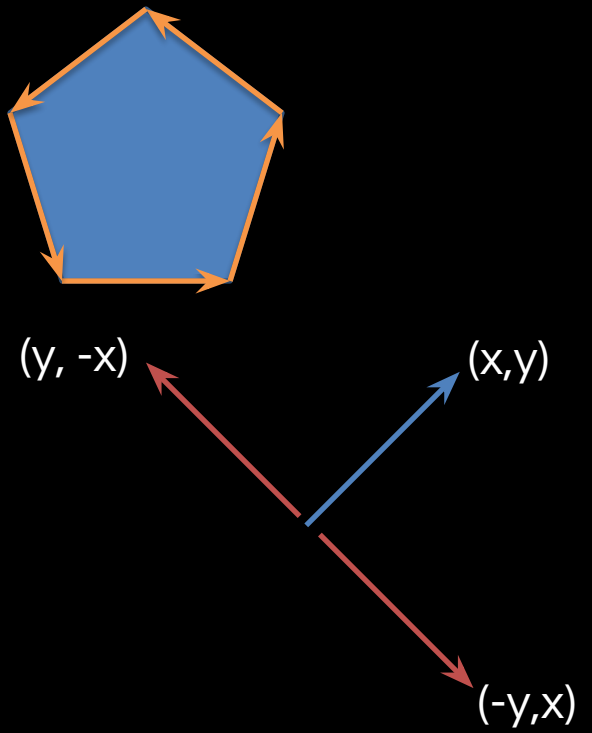
Special Cases: AABs

- Four axes perpendicular to the edges of an AAB
- Two of them are parallel, why test four?
- For an AAB, separating axes are just x and y axes



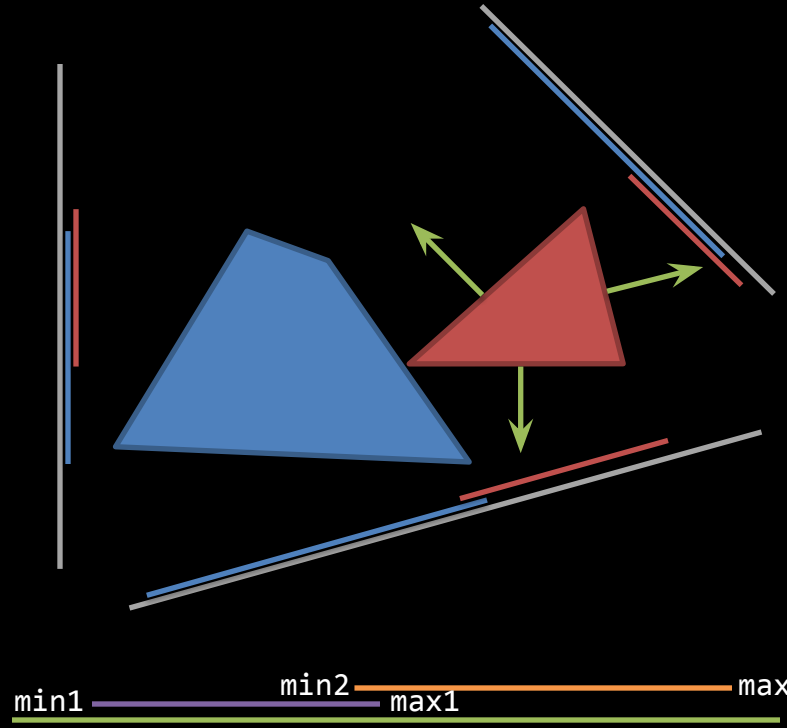
Implementation Notes

- To construct vectors for polygon edges, iterate around points counter-clockwise
- Two kinds of perpendicular: (-y, x) and (y, -x)
  - Make sure you're consistent



Implementation Notes

- Remember to check BOTH polygons' separating axes
  - Otherwise false positives
- Checking for overlap:
  `min1 <= max2 && min2 <= max1`



Collision Detection II

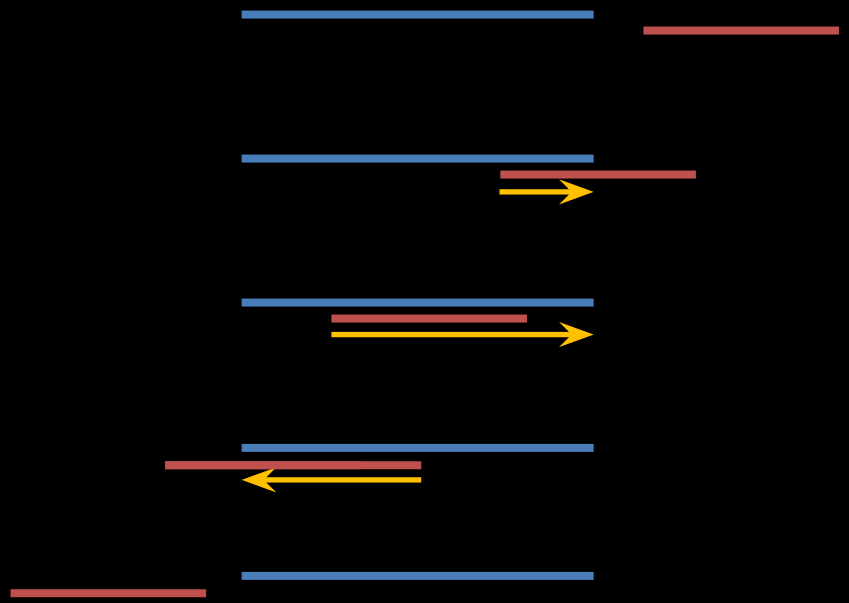
QUESTIONS?

Collision Detection II

MINIMUM TRANSLATION VECTOR

MTV in one dimension

- In 1D, convex shapes are line segments (intervals)
- These have a 1D MTV
  - Similar to overlap
  - But it has a sign
- Write a method that computes this
- Use it to find shapes' MTV



Computing MTV

1. For each (normalized!) axis, find 1D MTV of shapes' projections
2. Find the axis giving minimum 1D MTV
3. 2D MTV is 1D MTV times that (normalized) axis

Note: normalizing and projection are in the Vector source code!

Computing intervals' MTV

```
Float intervalMTV(Interval a, Interval b)
  Float aRight = b.max - a.min
  Float aLeft  = a.max - b.min
  if aLeft < 0 || aRight < 0
    return null
  if aRight < aLeft
    return aRight
  else
    return -aLeft
```

Computing polygons' MTV

```
Vec shapeMTV(Shape a, Shape b)
    Float minMagnitude = +infinity
    Vec mtv = null
    for Vec axis in allAxes
        Float mtv1d = intervalMTV(a.proj(axis),
                                  b.proj(axis))
        if mtv1d is null
            return null
        if abs(mtv1d) < minMagnitude
            minMagnitude = abs(mtv1d)
            mtv = axis.smult(mtv1d)
    return mtv
```

Special Cases

- Circle vs Poly
  - Use these axes:
    - Polygon's edge normals
    - Vector from circle center to closest vertex
- AAB vs Poly
  - Use these axes:
    - Polygon's edge normals
    - x axis and y axis

Collision Detection II

QUESTIONS?

An interesting aside on SAT

- The SAT is actually N-dimensional…
- To check two N-d convex shapes:
  - Find separating axes (N-1 dimensions)
  - Project all points onto this hyperplane
  - Use convex hull to get a (N-1)-D polygon
  - Run (N-1)-D SAT on these
  - Two N-d shapes overlap if all (N-1)-d projections overlap

Lecture 5

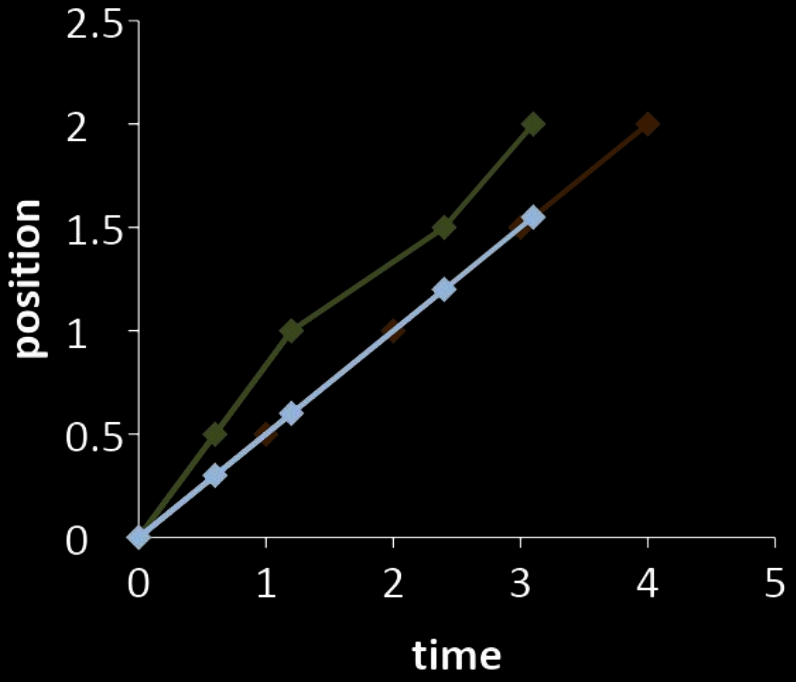
Physics II

Physics II

VELOCITY & ACCELERATION

Velocity

- Rate at which position changes
  - $\Delta x / \Delta t$
- Can't just increment position
  - Frame rates vary
- Multiply by time
  - $x \mathrel{+}= v * t$



Acceleration

- The rate that velocity changes
  - $\Delta v / \Delta t$
- Useful for gravity, springs, wind, etc.
- $v \mathrel{+}= a * t$

Which order to update?

Position first (Euler)

- pos = pos + vel * time
- vel = vel + acc * time

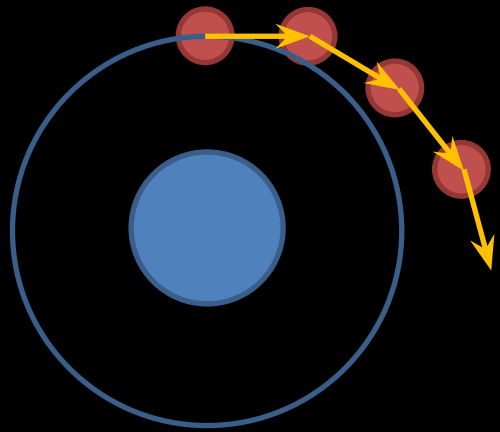
Velocity first (Symplectic Euler)

- vel = vel + acc * time
- pos = pos + vel * time

Which order to update?

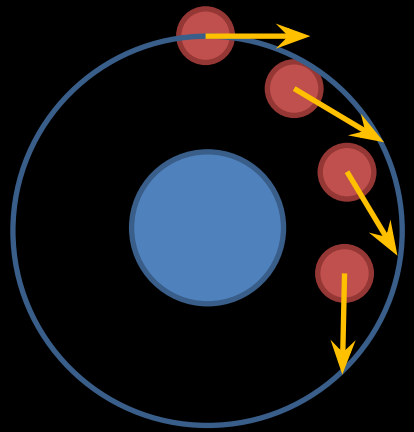
Position first (Euler)

- pos = pos + vel * time
- vel = vel + acc * time



Velocity first (Symplectic Euler)

- vel = vel + acc * time
- pos = pos + vel * time
- more stable, use this



Physics II

COLLISION RESPONSE

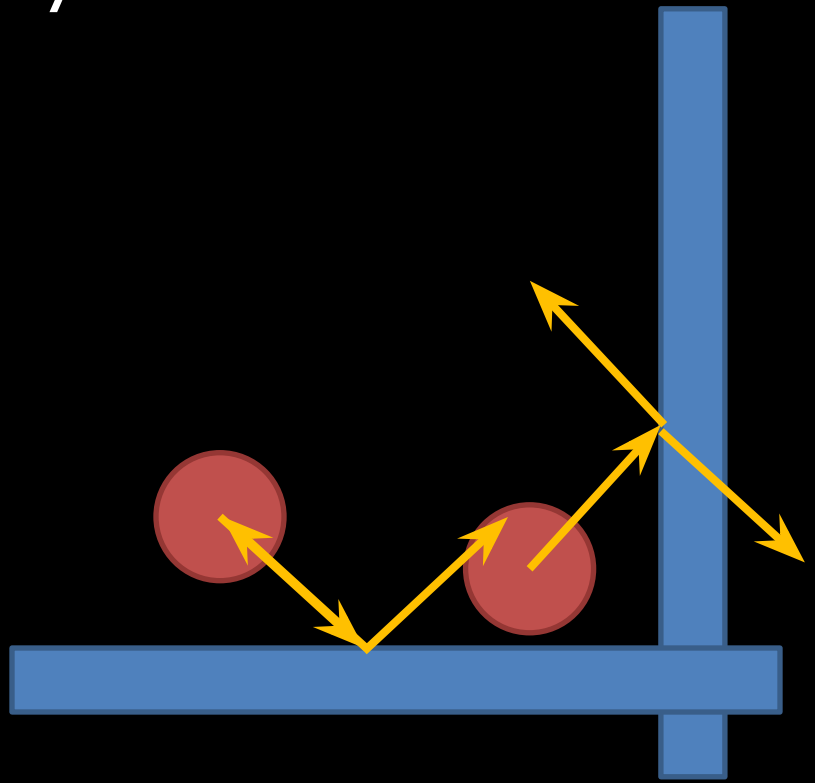
Changing velocity for collision

- Just reverse the object's velocity?

  $vel = -vel$

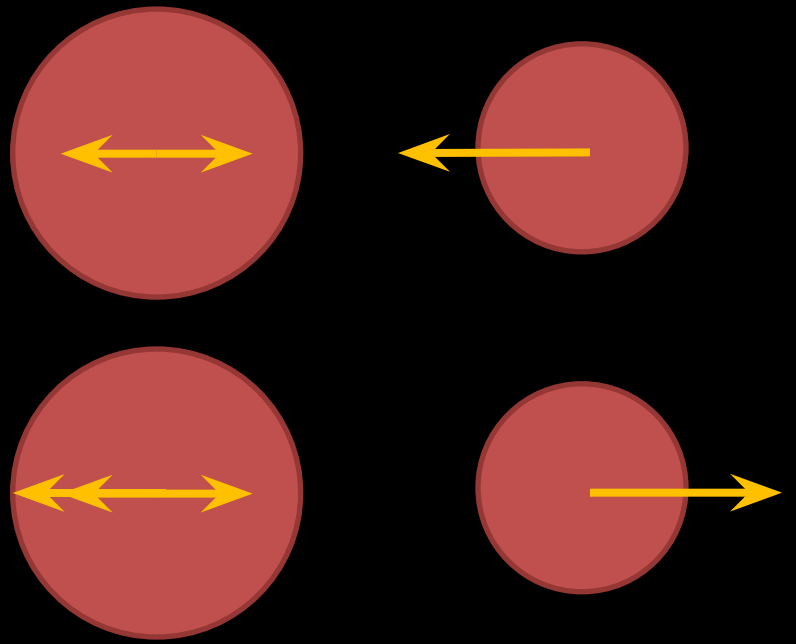
- Reverse the y component?

  $vel_y = -vel_y$



2-moving-object collisions

- Reverse both velocities?
- Doesn't always work
- Apply equal and opposite impulses
  - An impulse is an instantaneous force



Units

Without mass

- position m
  - $\vec{x}$
- velocity m/s
  - $\vec{v} = \Delta\vec{x}/t$
- acceleration m/s^2
  - $\vec{a} = \Delta\vec{v}/t$

With mass

- (no equivalent)

- momentum kg m/s
  - $\vec{p} = m\,\vec{v}$
- force $kg\ m/s^2$
  - $F = \Delta\vec{p}/t$
- impulse kg m/s
  - $\Delta\vec{p}$

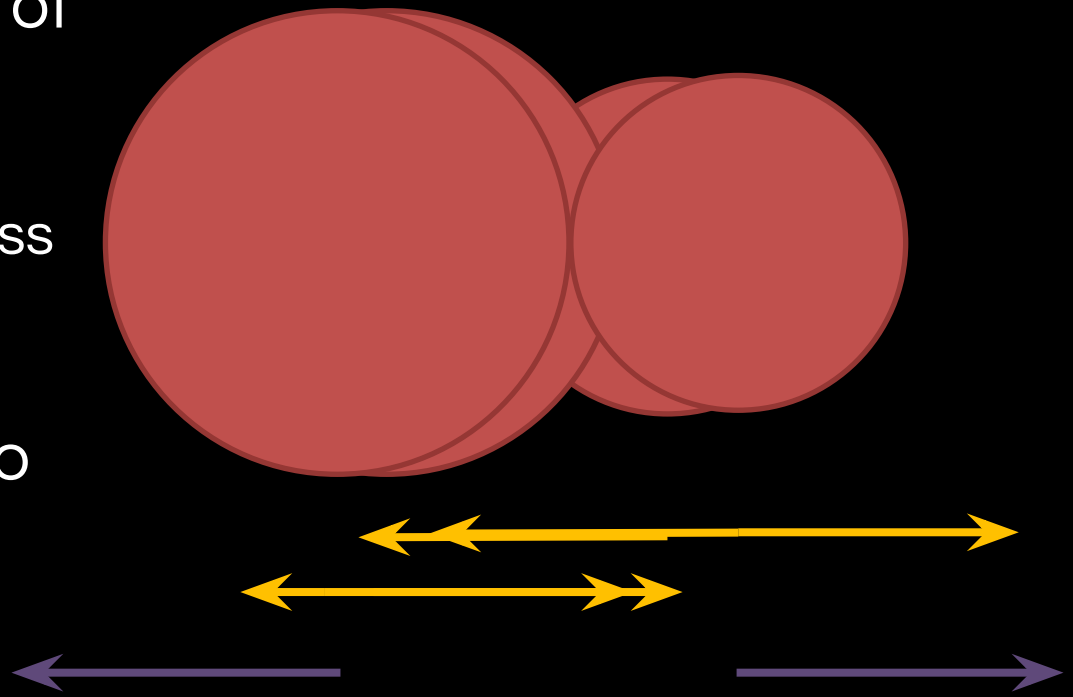
Implementing force and impulse

- applyForce(…) accumulates force
- applyImpulse(…) accumulates impulse
- onTick(…) applies force and impulse, clearing them for next frame
- Static (immovable) objects shouldn't be affected by applyForce (…) and applyImpulse (…)

```
class PhysicsBehavior {
  double mass;
  Vec2d pos, vel;
  Vec2d impulse, force;
  void applyForce(Vec2d f) {
    force += f;
  }
  void applyImpulse(Vec2d p) {
    impulse += p;
  }
  void onTick(float t) {
    vel += t*force/mass + impulse/mass;
    pos += t*vel;
    force = impulse = 0;
  }
}
```

Impulse collision response

- Translate objects out of collision
  - Each by MTV/2
  - Or proportional to mass in direction of MTV
- Apply some impulse proportional to MTV to each object
  - Details in next section

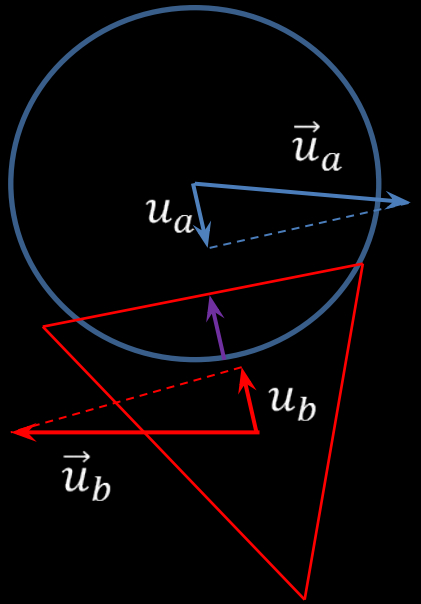


Physics II

QUESTIONS?

Note about Velocity

- When working with collisions, we only care about the velocity in the direction of the collisions
- Your engine is 2D, so your velocities are 2D vectors
- For all the math in the next slides, we'll need our velocity as a scalar
- To do this, take the dot product of the velocity and the normalized MTV



Physics II

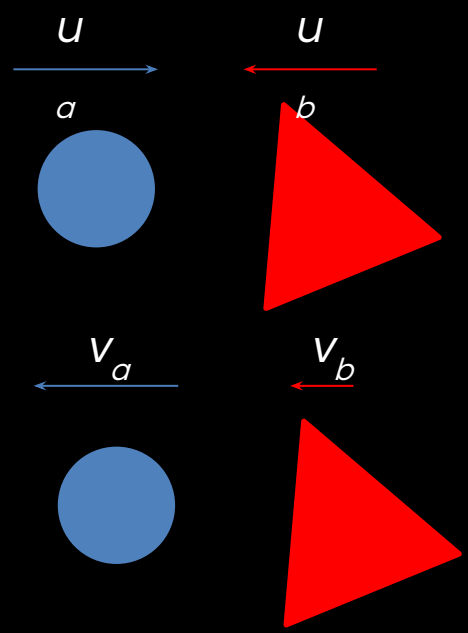
RESTITUTION

Restitution

- Property of physical entities
- Value between 0 and 1
- 0 is perfectly inelastic, 1 is perfectly elastic, etc.
- The **coefficient of restitution** (COR) between two entities is the geometric mean of their restitutions: $\sqrt{(r_1 r_2)}$

Correct Collisions

- How do we find the physically correct collision response?
  - i.e. given u_a and u_b, what are v_a and v_b?
- Use physical definition of the COR: $$\frac{v_a - v_b}{u_a - u_b}$$



Final Velocities

- Conservation of momentum:
  - $u_1 m_1 + u_2 m_2 = v_1 m_1 + v_2 m_2$
- The COR equation can be rewritten as
  - $COR * (u_a - u_b) = v_a - v_b$ $\quad COR = \mathrm{sqrt}(r_1 r_2)$
- So conservation of momentum becomes
  - $m_a u_a + m_b u_b = m_a v_a + m_b (a_v - COR * (u_a - u_b))$

Final Velocities

- Solving for v_a:

$$\frac{m_a u_a + m_b u_b + m_b COR * (u_b - u_a)}{m_a + m_b}$$

- Similarly for v_b (reversing a and b):

$$\frac{m_a u_a + m_b u_b + m_a COR * (u_a - u_b)}{m_a + m_b}$$

Final Velocities

- Can't just set velocities directly to v_a and v_b!
- Might interfere with other collisions
- Use impulse instead

Velocity Difference

- Impulse causes a change in velocity- we want to change from u to v

$$v_a - u_a = \frac{m_b(1 + COR)(u_b - u_a)}{m_a + m_b}$$

$$v_b - u_b = \frac{m_a(1 + COR)(u_a - u_b)}{m_a + m_b}$$

Final Impulse

- Multiplying by mass, we get:

$$I_a = \frac{m_a m_b (1 + COR)}{m_a + m_b} (u_b - u_a)$$

$$I_b = \frac{m_a m_b (1 + COR)}{m_a + m_b} (u_a - u_b)$$

- Implement these equations in your code

Static Shapes

- If a is static, then you can take the limit of I_b as the mass of a goes to infinity to get:

$$I_b = m_b(1 + COR)(u_a - u_b)$$

- Vice-versa if b is static
- You should special-case this

Putting it all together

Physically correct collision response:

1. Calculate COR with the restitutions of the shapes
2. Project velocities onto MTV
3. Apply impulse formula to calculate impulses
4. Apply corresponding impulse to each shape

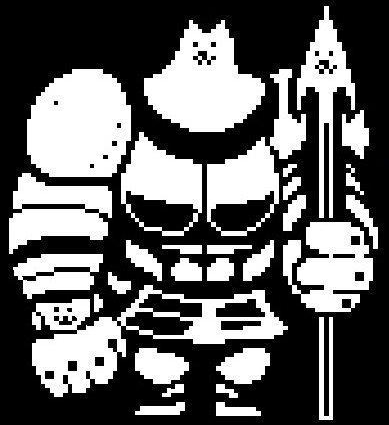
Reminders

- Our `projectOnto()` method gives a vector
- Our `dot()` method gives a scalar
- For physics equations, you want a scalar velocity, so use `dot()`
- To use `dot()` for projection, you need to give it a normalized vector

Physics II

QUESTIONS?

Lecture 5

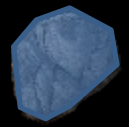


Greater Dog.

Tips for Nin I

More Accurate Sprites

- Give units a “bounding shape” that mimics visible outline of sprite



Behavior Trees/GOAP

- Give the enemies an AI instead of hardcoded behavior
- Keep track of the player's current level/difficulty

Axis / Range Classes

- Projection is tedious, can be messy
- An `Axis` class is a good idea
  - `project` methods for each shape
- May also want a `Interval` or `Projection` class
- `Interval`s should be able to compare themselves to other `Interval`s

```
public class Axis {
  public Axis(Vec2f direction)
  {…}
  public Interval project(Circle c)
  {…}
  public Interval project(AAB a)
  {…}
  public Interval project(Polygon p)
  {…}
  public Interval project(Compound c)
  {…}
}
```

Fixed Timestep

- Collisions can break if you have a really long tick
- This might happen, so consider:
  - Give the physics world some constant time on each tick
  - Tick as many times as possible on each game tick
  - Called separately from tick() and lateTick()

Known Bad Cases

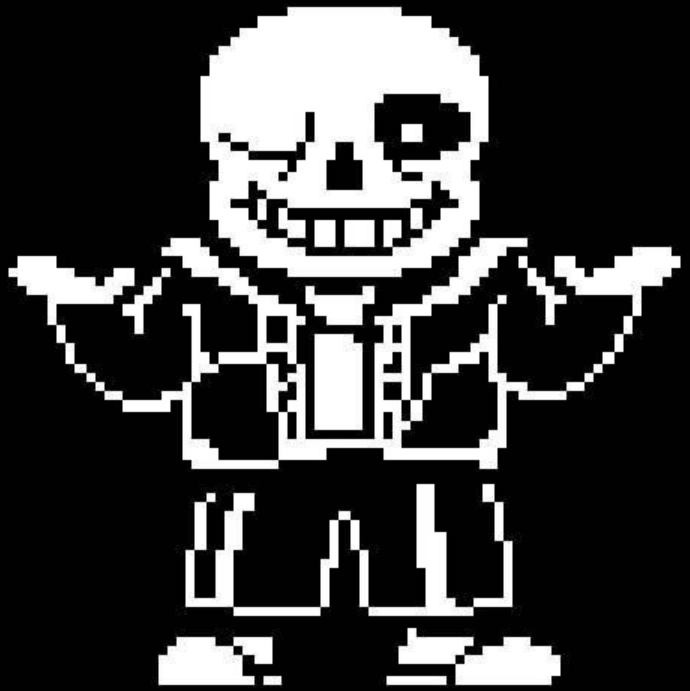
- Some things you know will make your code blow up
- Floating point values should never be `NaN`
- Vectors should never try to divide by zero
- Make sure there are never any NaNs
- Polygons should always be convex
- NaNs will ruin your life
- Vectors have `isNaN()` methods
- Use `assert`

Tips for Nin I

QUESTIONS?

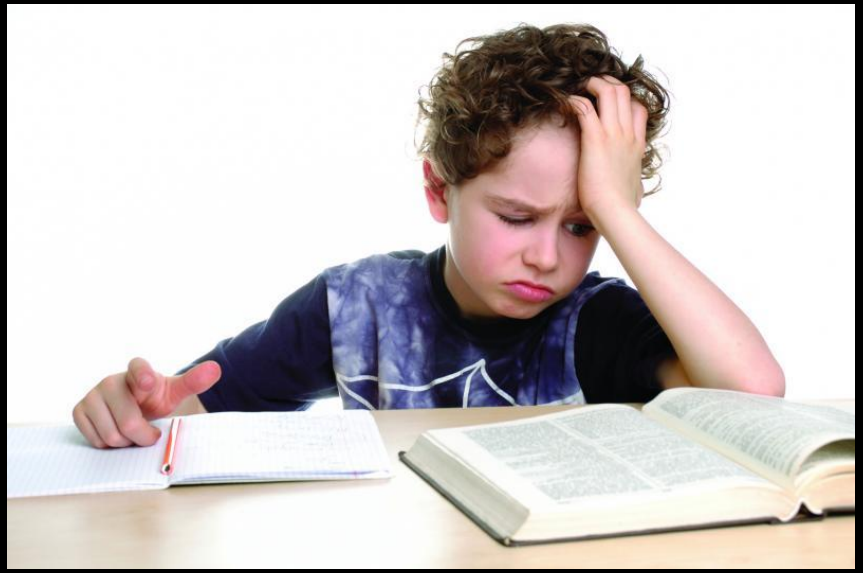
Game Design

Difficulty



What is difficulty?

- Games are “problem-solving activities approached with a playful attitude”
- The difficulty of solving the problems in the game determines how hard it is

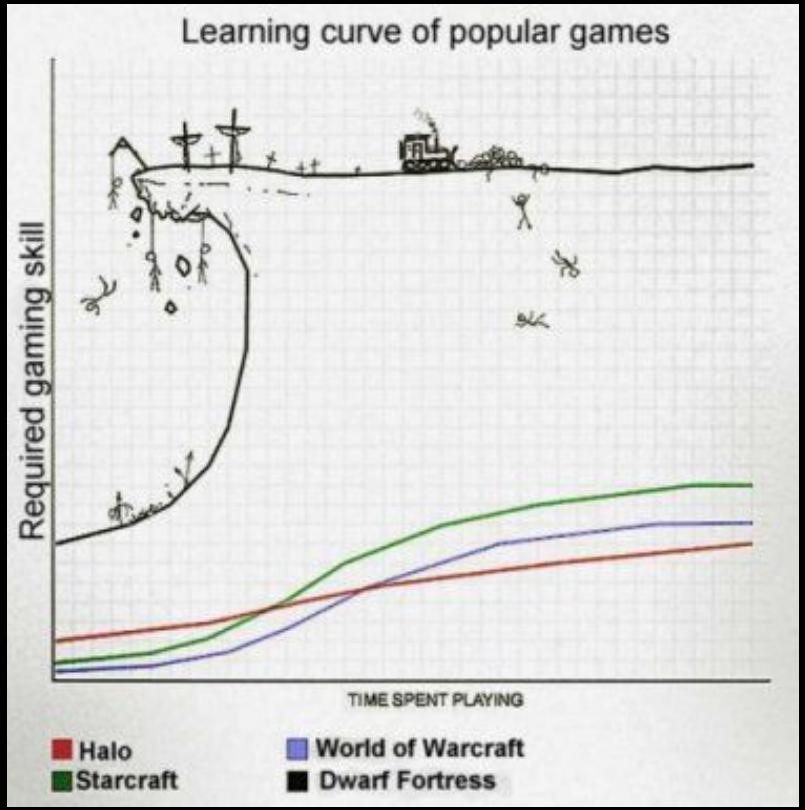


Why does it matter?

- Too easy: your players get bored and quit
- Too hard: your players get frustrated and quit

Components of difficulty

- Learning curve
  - Depends on complexity of controls and mechanics
- Punishment for failure
- Difficulty of sub problems
  - How do I defeat this enemy?
  - How do I make a Tetris with these blocks?



What makes a problem difficult?

- Clarity of problem statement
- Clarity of having reached a solution
- Transparency of solution
- Difficulty of executing the solution

Fair vs. Unfair difficulty

In a fair game…

- The player is responsible for failure
- The player clearly understands the objective
- The player knows what they are capable of doing

In an unfair game…

- Random factors determine failure
- The player doesn't know they're trying to do
- The player doesn't know what they can do

Gauging the difficulty of your game

- As the programmer and designer, you know your game inside and out
- General rule: actual difficulty is always at least one step up than what you think it is
- Playtesting is the best way to test how hard your game is
- No playtesters? Play your game with various levels of effort (from lazy to tryhard)
- TAs need to be able to beat your game to grade it

Adjusting difficulty

Play with the following:

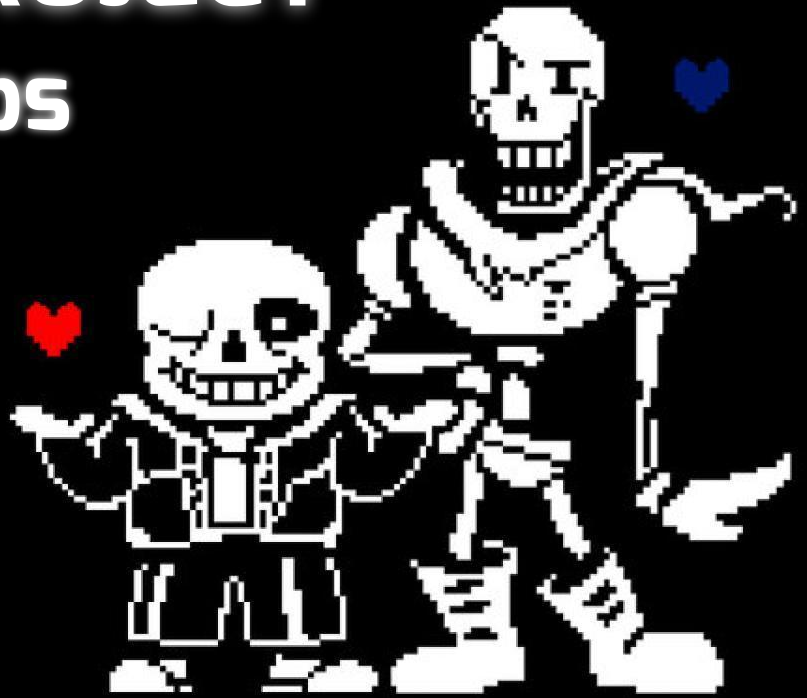
- Learning curve
  - How hard is it to learn to play the game?
- Degree of punishment for failure
  - How much is the player set back for messing up?
- Difficulty of subproblems
  - How hard is it to do things in the game?
- Number of distinct subproblems
  - How many different problems need to be solved?
- Clarity of problems (only if you know what you're doing!)
  - How well does the player know what they're supposed to be doing?

Difficulty

QUESTIONS?

Final Project

Groups



This Week

- Elevator pitches!

This Week

- Talk to your classmates!
- Form two groups of two or a group of four
- Have one member of your group email the TA list with all group members
- Need an email from each group
  - Tell us when you can meet for a group design check
  - Do this by Thursday

Final Projects

QUESTIONS?

lcohen2

- Multiplayer versus tower defense with randomized building components. Build the highest tower!

nfahey

- A game similar to the Tanks game from Wii Play. Two player game where players progress through procedurally generated levels and face different types of tanks.

akintisc

- You need to figure out who you are and what's going on in your limited world – without anyone catching on. A puzzle game where you can't trust anybody, especially yourself.

bgabinet

- The game is called “Did I Say You Could Stop Partying?” You play as a DJ at a party, and you are tasked with tackling anyone who tries to leave. If too many people manage to get out, you’ve failed as a musician.

jaw5

- You’re a janitor on an interstellar spaceship that also happens to be a zoo, travelling the universe showing off tons of exotic and dangerous animal-type things. But uh-oh! What’s this? A malfunction on the ship allowed for the animals to escape and they’ve murdered the entire crew! What are the odds?! Escape the spaceship without dying, that’s basically the entire thing.

czegura

- I’d like to work with two other people in order to create a game which privileges player interaction beyond combat. This game will focus on player choice to create a compelling and branching narrative with a plethora of both ally and enemy NPCs

tdgreen

- My game will be a multiplayer, 2D version of the incredibly fast growing and popular game, PUBG (PlayerUnknown's BattleGrounds). The goal will be to have a top-down version of the first person shooter, which may end up having a feel very much like the old flash-game BoxHead. The game will involve a decent amount of networking, low-latency actions, and a lot of late-night industry research.

rm28

- What I’m looking to make is a platformer puzzle game. The premise of the game is pretty simple. You start somewhere in the level and you have to solve a puzzle to get to the end. The mechanics are also pretty simple. You can move around and jump, but there is one caveat: you’re given the ability to warp through walls, but you can only do so once before you have to touch the ground to use it again. There’ll be lots of obstacles, like spikes that kill you, enemies that kill you, and walls you can’t warp through.

mbartle

- You’re in the CIT elevator, falling at an alarming speed towards hell.

Wiz 2 Playtesting

Hooray!