SIMPLE PHYSICS (MOSTLY) PLATFORMERS

FOR GAM 150 CLUB

Randy Gaul

Who am I?

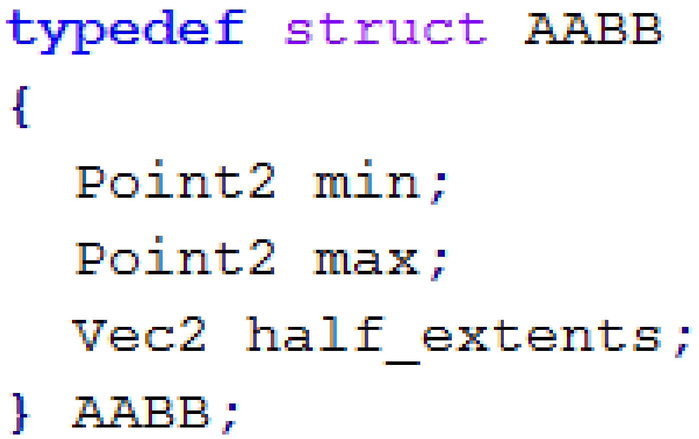
- RTIS Sophomore – Randy Gaul
- C game as Freshman
- Tech director for Ancient Forest and Grumpy Monsters
- Made engine in C during summer before Sophomore year
  - AsciiEngine – Implemented some features in this slideshow
- Love architecture with clean and powerful APIs

I will describe

- AABB collision
- Integration
- Jumping
- Collision resolution
- Slopes
- One way platforms
- Ladders
- Moving platforms

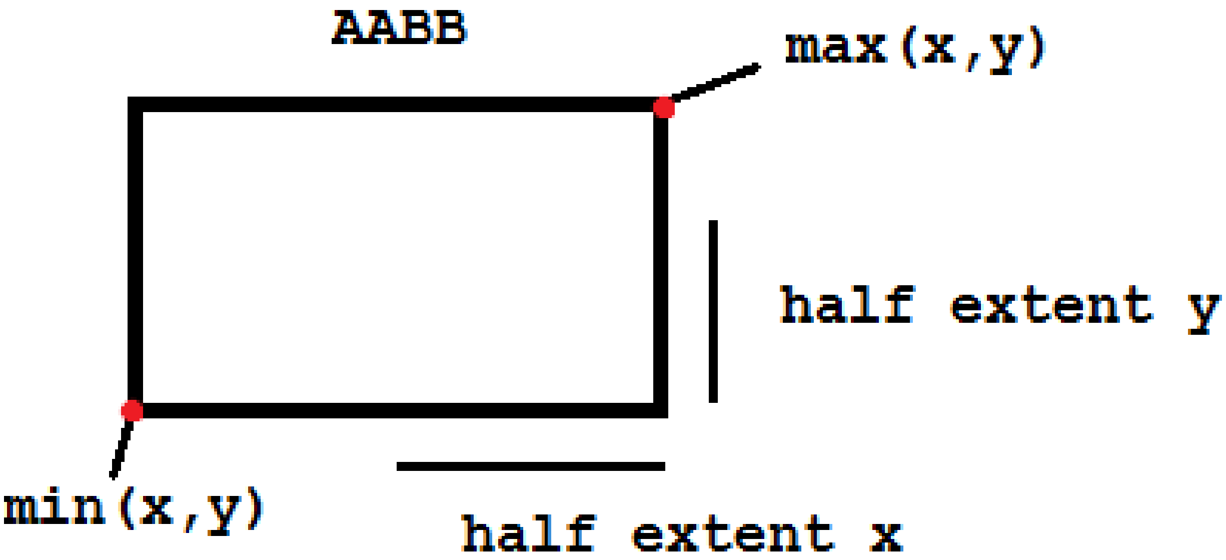
AABB Collision

- AABB – Axis aligned bounding box
  - Bounding box – a box that bounds another shape
  - In our case, you’re just using boxes, not bounds
  - Technically AAB
- Store AABB the smart way (extents optional):



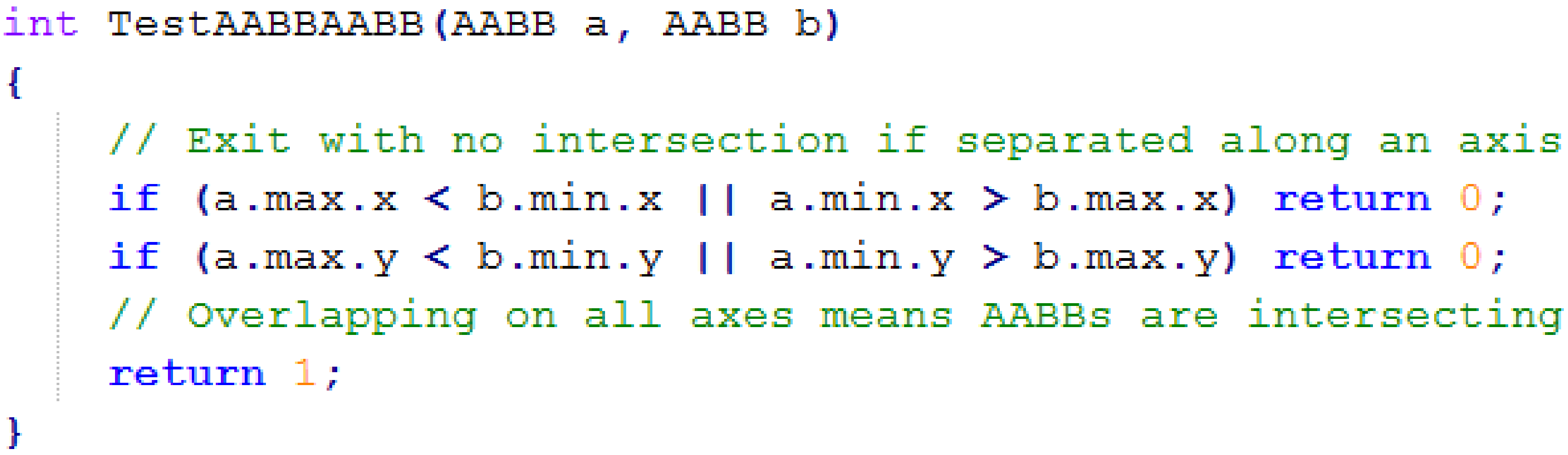
```
typedef struct AABB
{
   Point2 min;
   Point2 max;
   Vec2 half_extents;
} AABB;
```

AABB Collision



AABB Collision

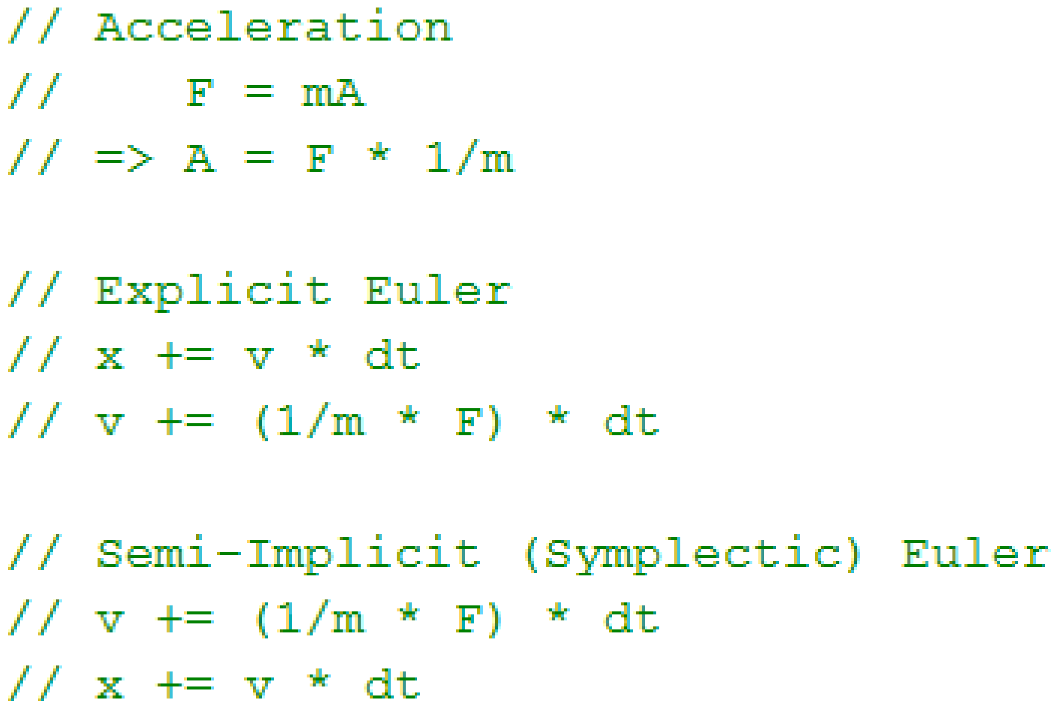
- Here’s the code
- If you want to know more:
  - Look up AABB collision detection
  - Look up SAT – separating axis test



```
int TestAABBAABB(AABB a, AABB b)
{
    // Exit with no intersection if separated along an axis
    if (a.max.x < b.min.x || a.min.x > b.max.x) return 0;
    if (a.max.y < b.min.y || a.min.y > b.max.y) return 0;
    // Overlapping on all axes means AABBs are intersecting
    return 1;
}
```

Integration

- □ Euler integration!
- □ Use symplectic, not explicit



```
// Acceleration
//    F = mA
// => A = F * 1/m

// Explicit Euler
// x += v * dt
// v += (1/m * F) * dt

// Semi-Implicit (Symplectic) Euler
// v += (1/m * F) * dt
// x += v * dt
```

Integration

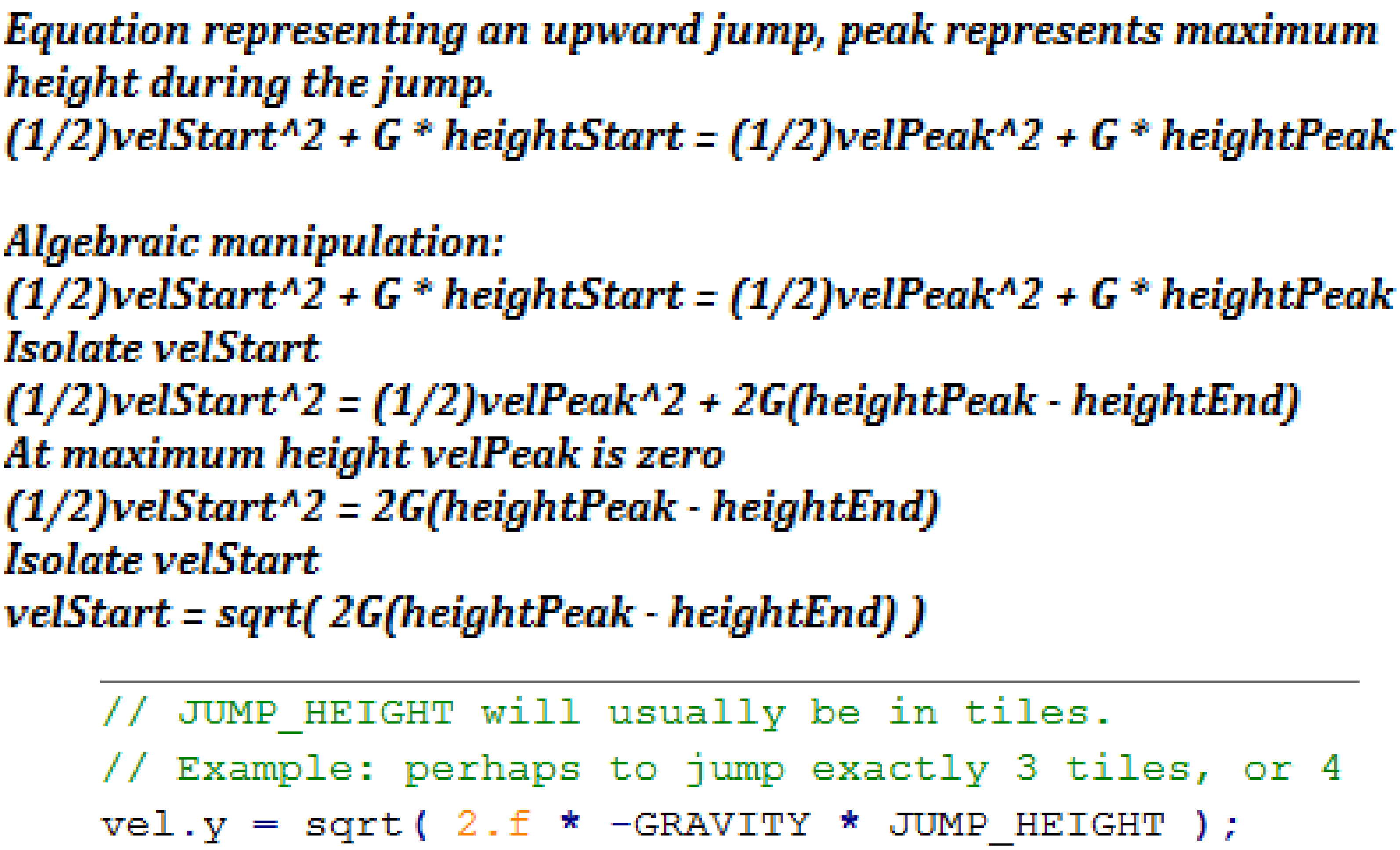
- Considerations:
  - Instead of (1/mass * force)
    - Can just use an acceleration vector
- What is force?
  - Mass * acceleration
  - Gravity!
  - Wind
- Simulating forces
  - Have a “accumulator” vector
  - Add a bunch of other forces with accumulator
  - Integrate with accumulator
  - Clear accumulator after integration and repeat

Questions?

Jumping

- Just apply impulse to velocity
  - Add a vector to velocity
    - Vec2(0. 5.0)
- How big of impulse to apply?
  - Guess and check
  - Use equation to solve for exact height
    - Useful to jump or throw objects exact tile heights

Jumping



Equation representing an upward jump, peak represents maximum height during the jump.
*(1/2)velStart^2 + G * heightStart = (1/2)velPeak^2 + G * heightPeak*

Algebraic manipulation:
*(1/2)velStart^2 + G * heightStart = (1/2)velPeak^2 + G * heightPeak*
Isolate velStart
(1/2)velStart^2 = (1/2)velPeak^2 + 2G(heightPeak - heightEnd)
At maximum height velPeak is zero
(1/2)velStart^2 = 2G(heightPeak - heightEnd)
Isolate velStart
velStart = sqrt(2G(heightPeak - heightEnd))

```
// JUMP_HEIGHT will usually be in tiles.
// Example: perhaps to jump exactly 3 tiles, or 4
vel.y = sqrt( 2.f * -GRAVITY * JUMP_HEIGHT );
```

Collision Resolution

- Don’t actually let things collide
  - Collision avoidance!
- The plan:
  - Integrate x one step forward (x before y!)
  - Run AABB to AABB
  - Find closest object collided with
    - AABB vs AABB from previous slide
  - Move object tangent to closest collidee
    - Move object so that it’s touching what it collided with
  - Zero velocity on x axis
  - Repeat for y

Collision Resolution

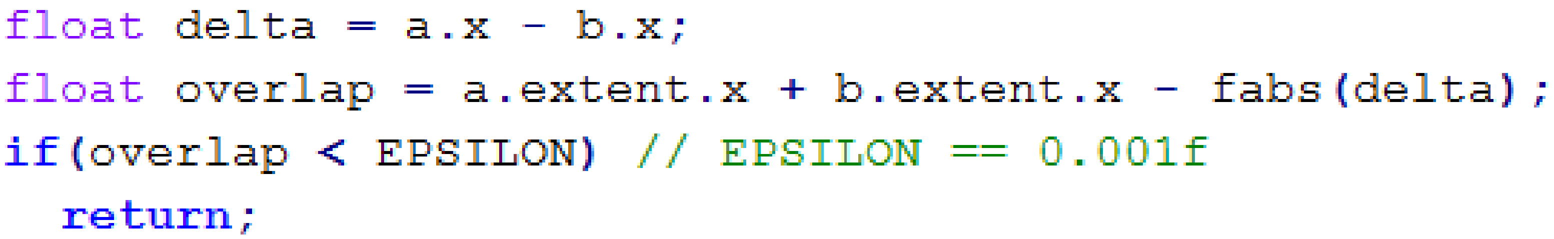
- Previous algorithm properties:
  - Pros
    - Boxes easily stack
    - Easy detecting direction of impact
    - Objects shouldn’t intersect
    - Easy to implement and debug
    - Extremely fast
  - Cons
    - Objects cannot push other objects without special code
    - No “bouncing” off of things without special code

Collision Resolution – A Problem

- If you store positions with integers you’re fine
- For floating point positions:
  - Placing an object tangent to another:
    - Involves a subtraction
    - Introduces floating point error
    - Possible to still be “colliding” after resolve
  - Solution:
    - Allow a fudge factor for overlap

Collision Resolution – Detection Fudge

- □ Only return a collision of the overlap is smaller than some very small constant



```
float delta = a.x - b.x;
float overlap = a.extent.x + b.extent.x - fabs(delta);
if(overlap < EPSILON) // EPSILON == 0.001f
  return;
```

- □ Try 0.001f for EPSILON, and lower until unstable

Friction

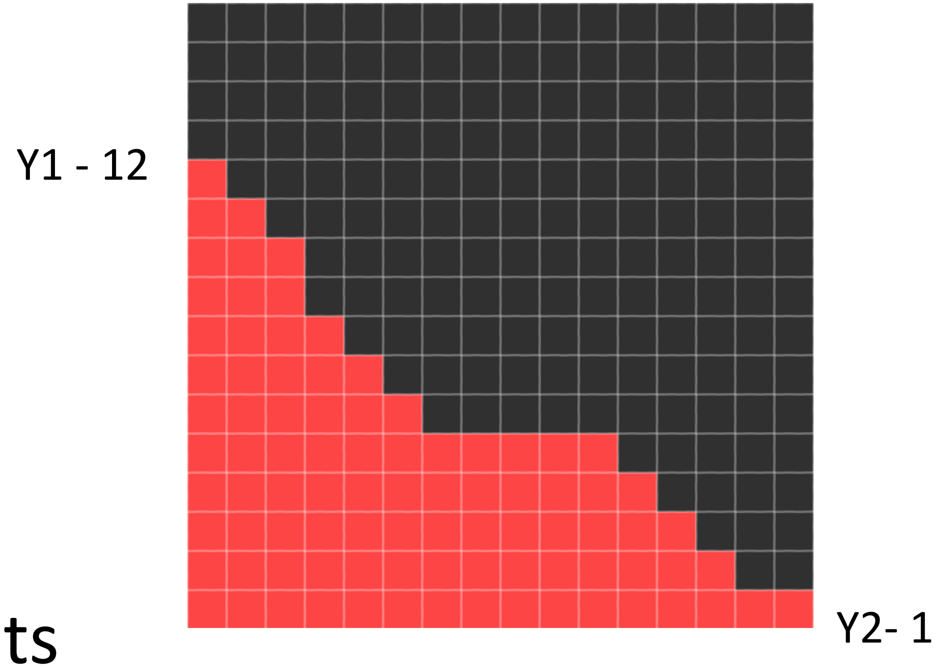
- Want to slow down when we hit something
  - On x integration
    - Multiply y velocity by a constant less than one
      - Try: 0.05, 0.1, 0.5
  - On y integration
    - Multiply x velocity by a constant less than one
      - Try: 0.05, 0.1, 0.5
- Can grab constants from objects
  - Each object can store a friction float

Slopes

- Two main methods:
  - Integer array for heights
  - Linear interpolation
- I will describe both

Slopes – Integer Array

- Each sloped tile contains array of heights



- Heights
  - 12, 11, 10, 8, 7, 6, 5, 5, 5, 5, 5, 4, 3, 2, 1, 1

Slopes – Integer Array

- If collision in tile
  - Resolve y position to index in array
    - Set object’s y position to y index <u>only if y position is lower</u>
      - Want to be able to jump and land on slopes
  - Use x position (center of object) to index into height array
- Considerations
  - What if you run into higher side first?
    - Can make special case – collide like solid box
    - Can setup tiles so this cannot happen

Slopes – Integer Array

- Slopes affect velocity?
  - Store arbitrary angle to approximate slope
    - velx += cos(angle) * A * dt
    - vely += -sin(angle) * A * dt
    - Keep track of the “real” velocity
- Apply the modified vel if collided with tile
  - Collide if:
    - Y less than height index
    - X extents overlap tile’s x extents

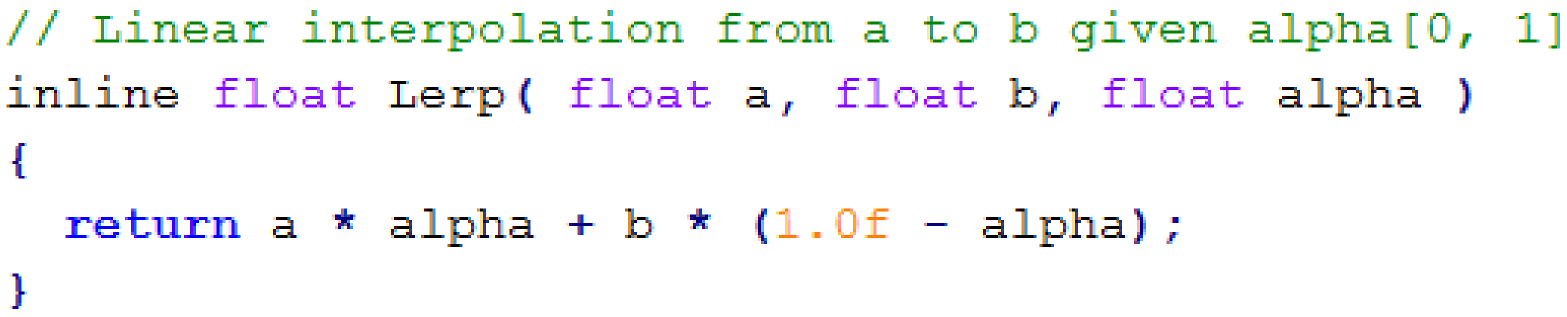
Slopes – Integer Array

- Pros
  - Support arbitrary heights at each index
  - Very fast
  - Fairly simple to implement
- Cons
  - Takes up a little more memory
  - Can be tedious to hand-craft each tile

Questions?

Slopes – Linear Interpolation

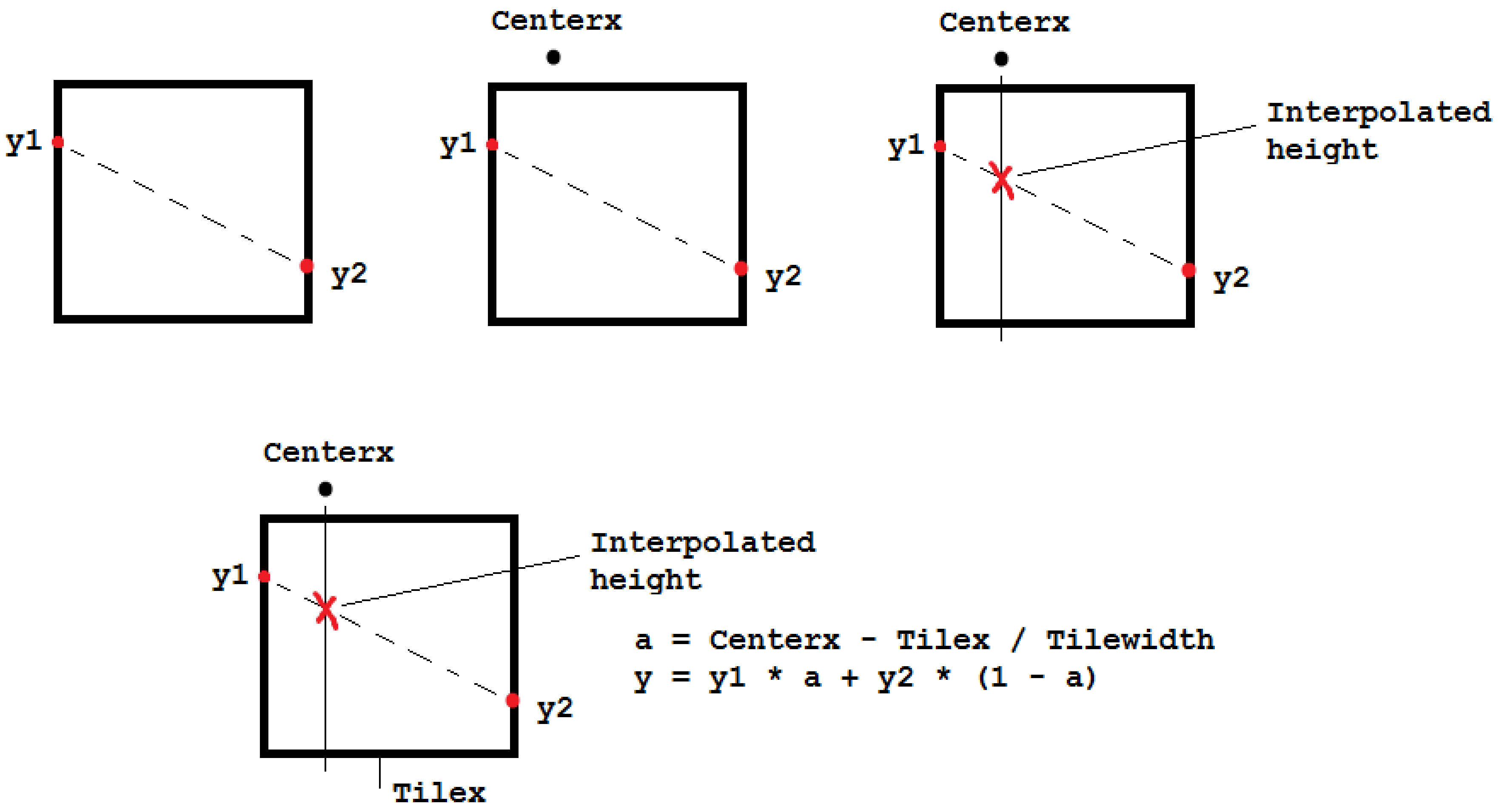
- Calculate y heights from interpolation



```
// Linear interpolation from a to b given alpha[0, 1]
inline float Lerp( float a, float b, float alpha )
{
  return a * alpha + b * (1.0f - alpha);
}
```

- alpha = object_x – tile_x / tile_width
- a = left starting point of line (12 from slide 14)
- b = right endpoint of line (1 from slide 14)

Slopes – Linear Interpolation

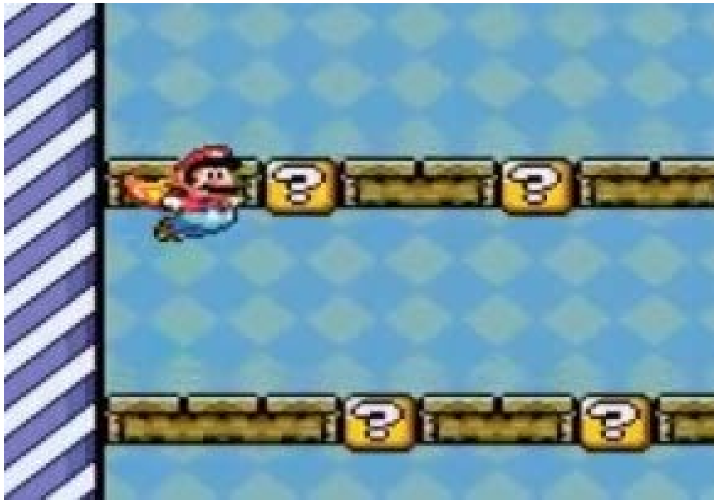


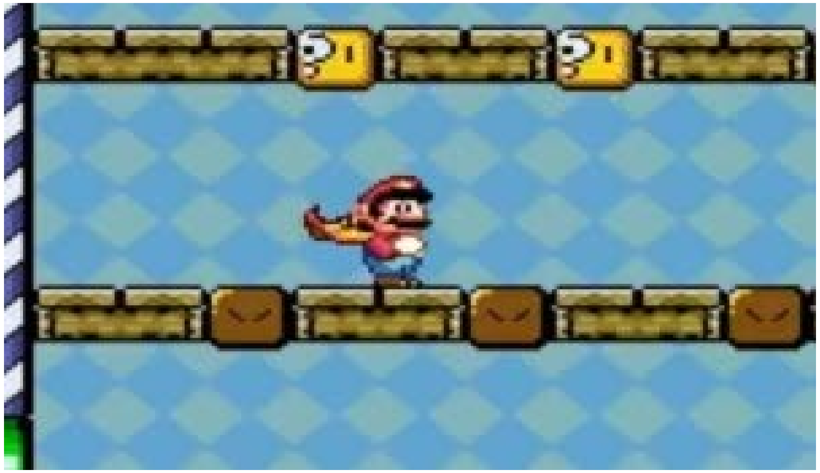
```
a = Centerx - Tilex / Tilewidth
y = y1 * a + y2 * (1 - a)
```

Slopes – Linear Interpolation

- Pros
  - Supports arbitrary slopes
  - Still very fast
  - Less memory per tile
  - Easier to create new tiles (less information to create)
    - Just set y1 and y2
- Cons
  - Single slope per tile
  - A bit harder to implement than integer array
- Note:
  - Only set object’s y to interpolated h only if y is less than h
    - We want to able to jump and land on slopes

One Way Platforms

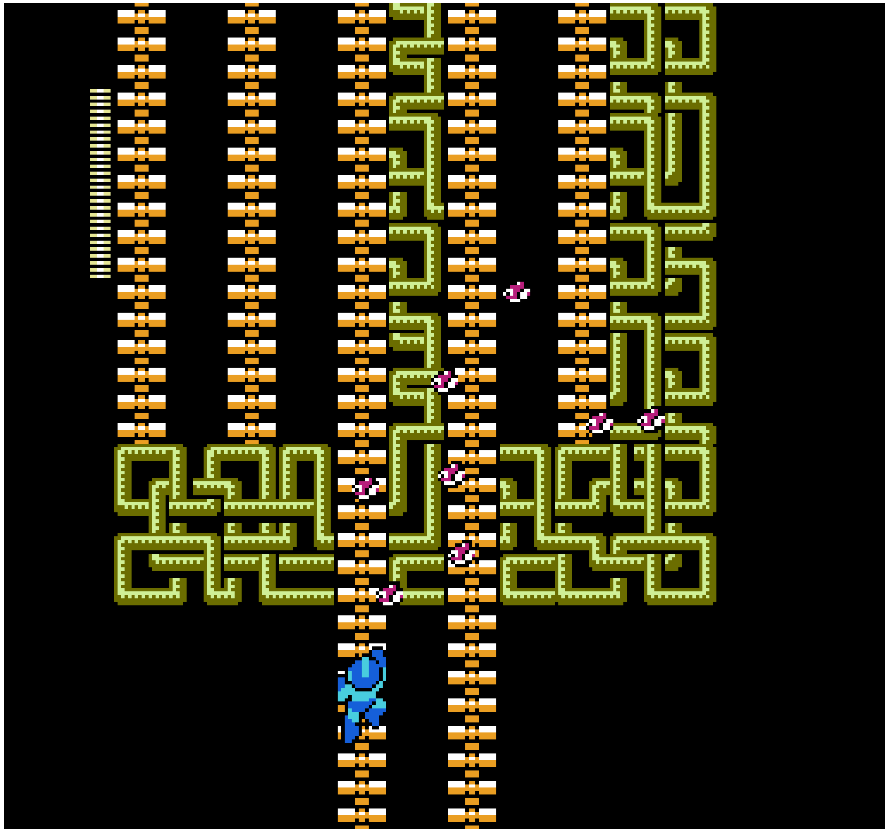




One Way Platforms

- Special cases!
  - Write some code to handle AABB vs moving platform in a special way
    - Don’t collide x axis step
    - Collide y only if previously *entirely* above platform
- Very simple
- A simpler method that doesn’t work
  - Collide only if falling
    - You can start falling mid-jump
    - This results in unintended behavior

Ladders



Ladders

- Will need two different states
  - Normal
  - On ladder
- Separating states make code simpler
- On ladder state:
  - No gravity
  - No left or right integration
  - Only move up/down
  - Very simple
- Most code will be for exit/enter ladder state

Ladders

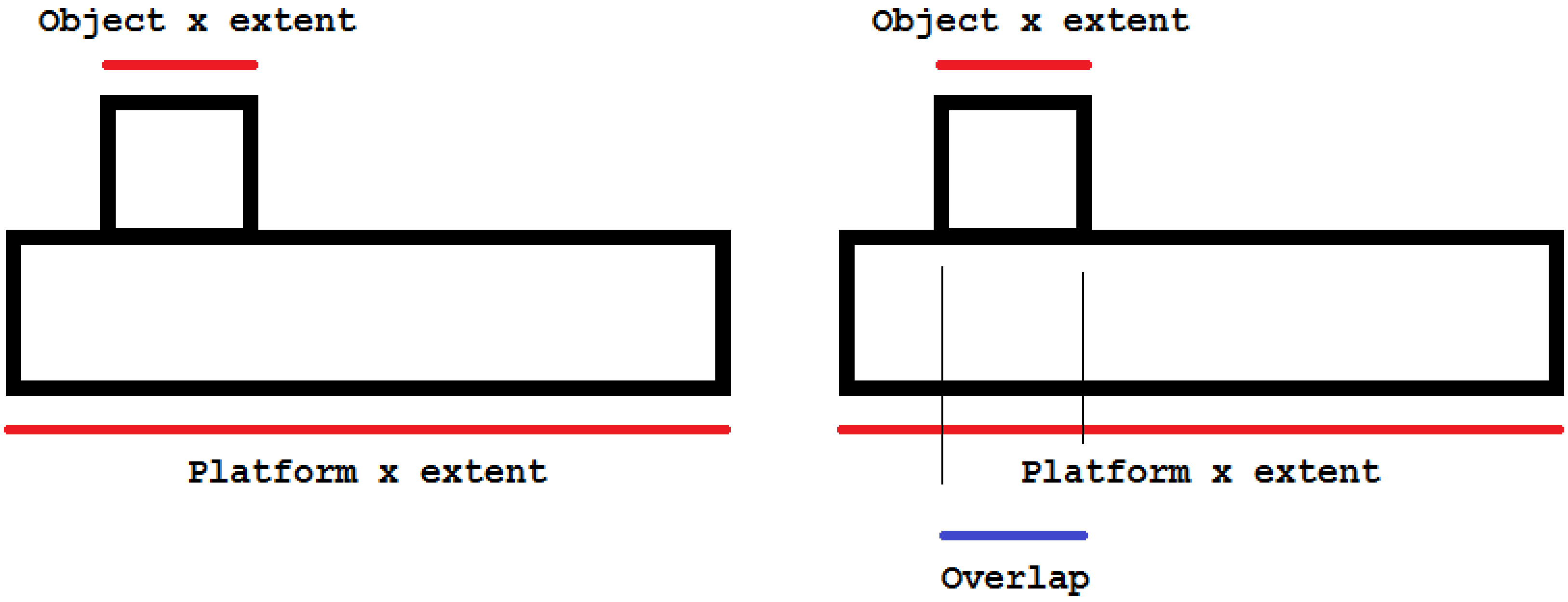
- Leave and exit ladder state
  - To enter:
    - Design dependent
    - AABB vs AABB player to ladder TRUE && player hit UP key
    - Stand on top and hit down
    - Clamp player’s x to ladder’s x
  - To exit suggestions:
    - Player hits jump button
    - Player_y outside bounds of ladder y extents
- Additional consideration:
  - Run one-way platform code
  - Allow to walk on top of ladders

Moving Platforms

- Can stand on platform
  - Player’s and monster move when platform does
- Have each platform store linked list of objects
  - When object collide on platform
    - Insert itself into platform’s list
  - On integrate platform, integrate objects in list too
  - When object leave platform
    - Remove self from platform’s linked list
- For fast removal use: doubly linked list
  - Lecture and example code on moodle

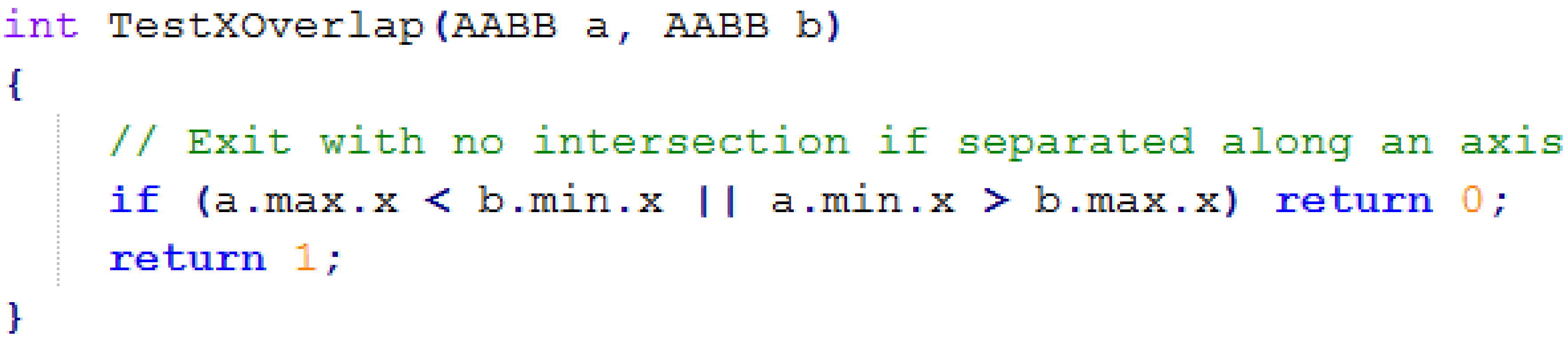
Moving Platforms - Leaving

- Object's y position changes upward
  - Jumps up, for example – detect this!
- Object's x extents do not overlap platform's x extents
  - A picture:



Moving Platforms - Leaving

- Calculate overlap for each object in platform list



```
int TestXOverlap(AABB a, AABB b)
{
    // Exit with no intersection if separated along an axis
    if (a.max.x < b.min.x || a.min.x > b.max.x) return 0;
    return 1;
}
```

- If found non-intersecting
  - Remove from platform’s list

Moving Platforms – A Problem!

- What if platform moves into an object?
  - Want object to be “pushed”
  - No way to model this
- Solution:
  - Update platforms *after* all other objects
  - All objects move about the game
    - Some collide into platform and register in the list
  - Then platform moves
    - Integrates it’s list of objects
  - Impossible for platform to run into anything

Final Tips

- Ask Doug Schilling for advice! He’s awesome
- Keep things as simple as you can
- Ask upper classmen questions
  - Email me: r.gaul@digipen.edu

Resources:

- http://higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/
- http://info.sonicretro.org/Sonic_Physics_Guide
- AsciiEngine (link front page)
  - Implemented this exact type of messaging
- Sample engine on Moodle

Questions

- Anybody have ‘em?