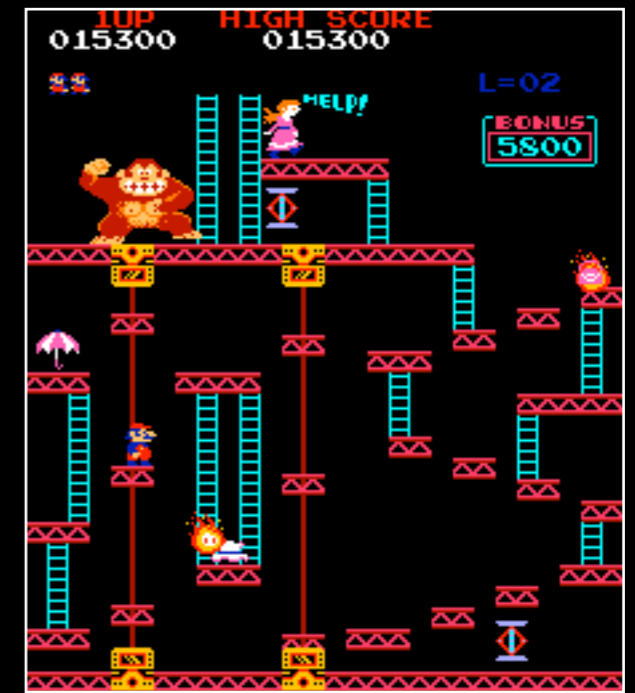


Platform Game Genre

A Mr. F Presentation

Genre Basics

- Began in early 1980's
- Main game element is jumping between suspended platforms
- May include ladders, spring boards, swing vines, etc



Topics for Today

- World Wrapping
- World Scrolling
- Basic Physics
- Hit Detection
- General Class Hierarchy
- Miscellaneous

World Wrapping

- Allows a character to exit one side of the screen and appear on the other side



World Wrapping

- Basic idea - when player's x coordinate is too larger (or too small) change it to be on the other side of the screen.



World Wrapping



- If mario's x coordinate is less than the opposite of its width, it should teleport to the world width.
- Similar for right side

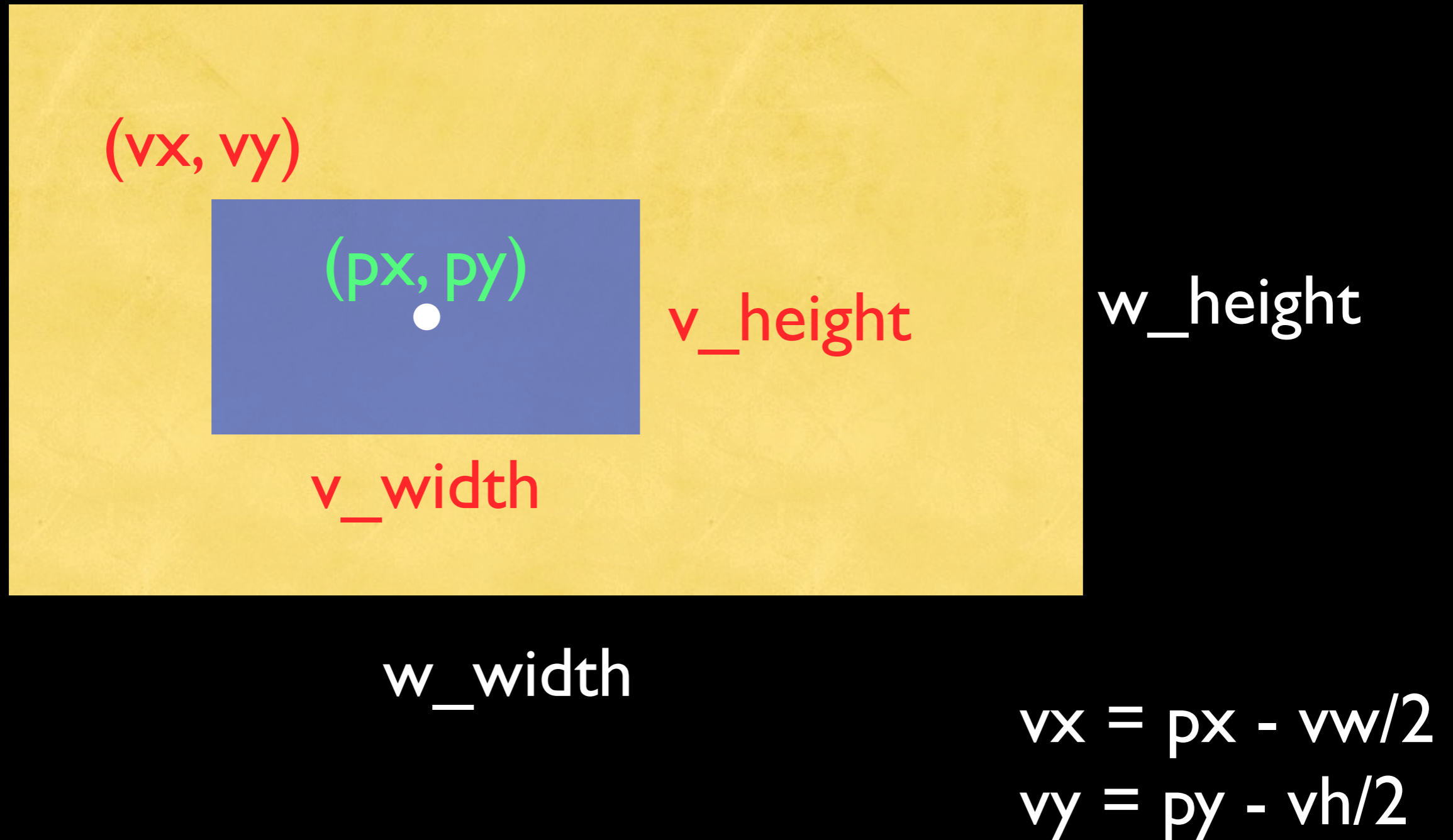
World Scrolling

- Allows a character to run around a large world, with the visible world area changing dynamically with the player's movement
- Precursor: Static Screen Transitions

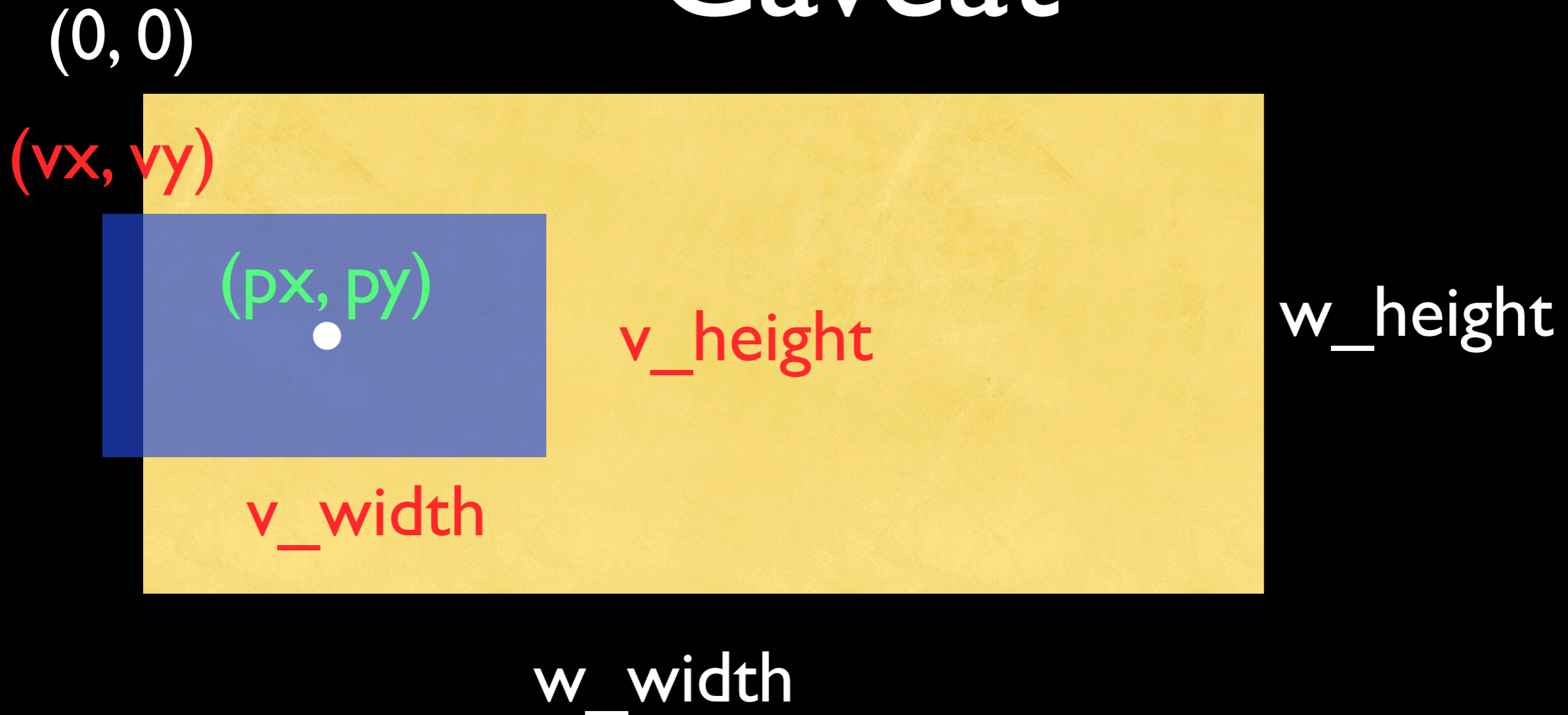


World Scrolling

(0, 0)



Caveat



Viewing Screen x and y need to be set to stay within the world bounds (unless you do world wrap as well!)

Caveat

$(0, 0)$

(vx, vy)

(px, py)

v_height

w_height

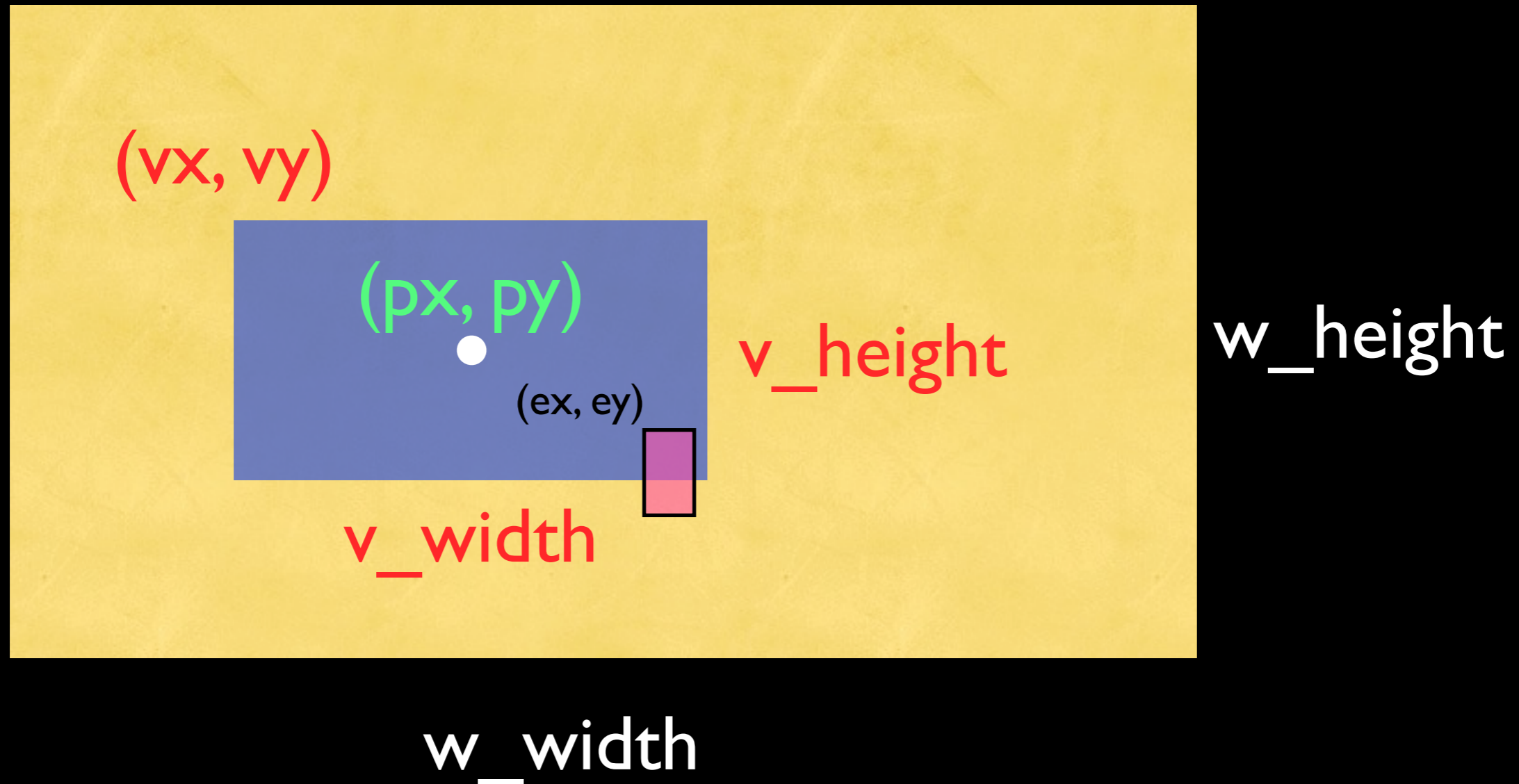
v_width

w_width

Viewing Screen x and y need to be set to stay within the world bounds (unless you do world wrap as well!)

Referenced Drawing

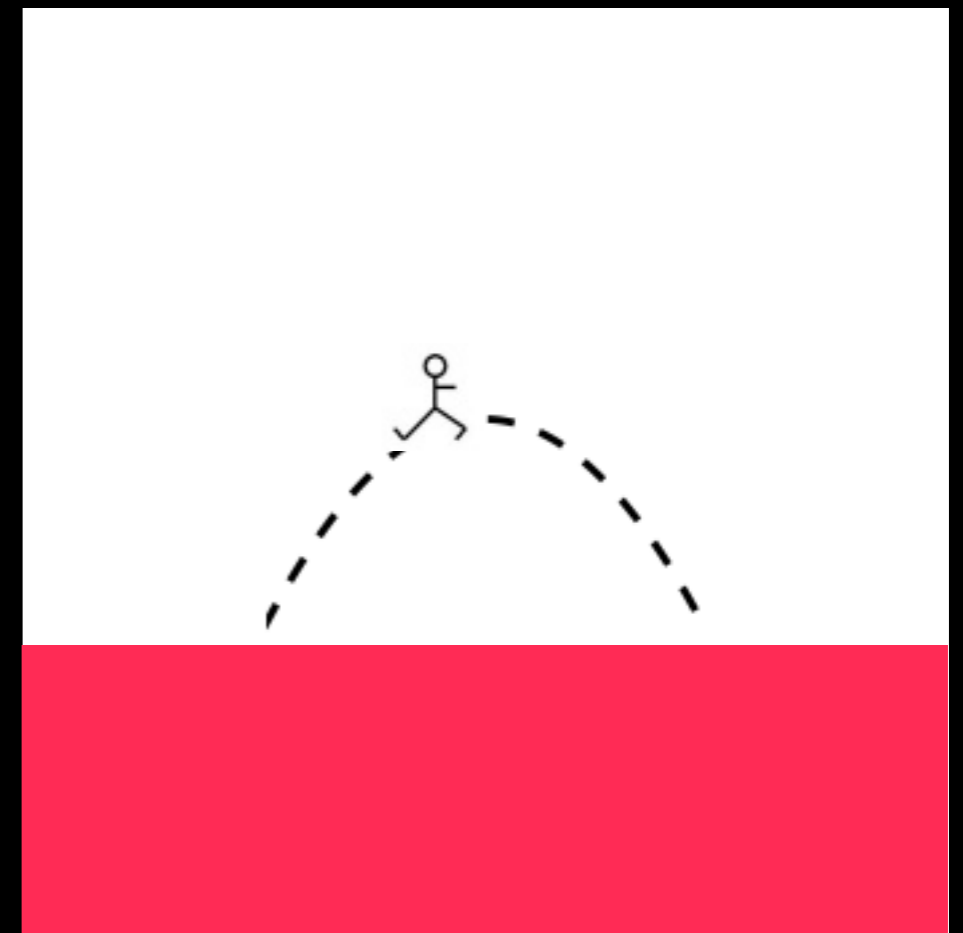
(0, 0)



```
g.drawImage(img, ex - vx, ey - vy, null);
```

Basic Physics

- The physics between in air and on ground character movement is somewhat different.
- Usually, a player's x position in the air can be controlled by keys



By land or air

- y_acc is a fixed **positive** constant
- each update, the y_vel should have the y_acc added to it
- similarly, the y_pos should have the y_vel added to it each update

Note on Velocity

- Should have a maximum possible velocity
- Without a limit, characters can accelerate off the screen at a ridiculous rate
- If characters move fast enough, they could potentially pass through objects and foil proper hit detection
- if speed of object exceeds width of another object, we might have a problem...

In the air



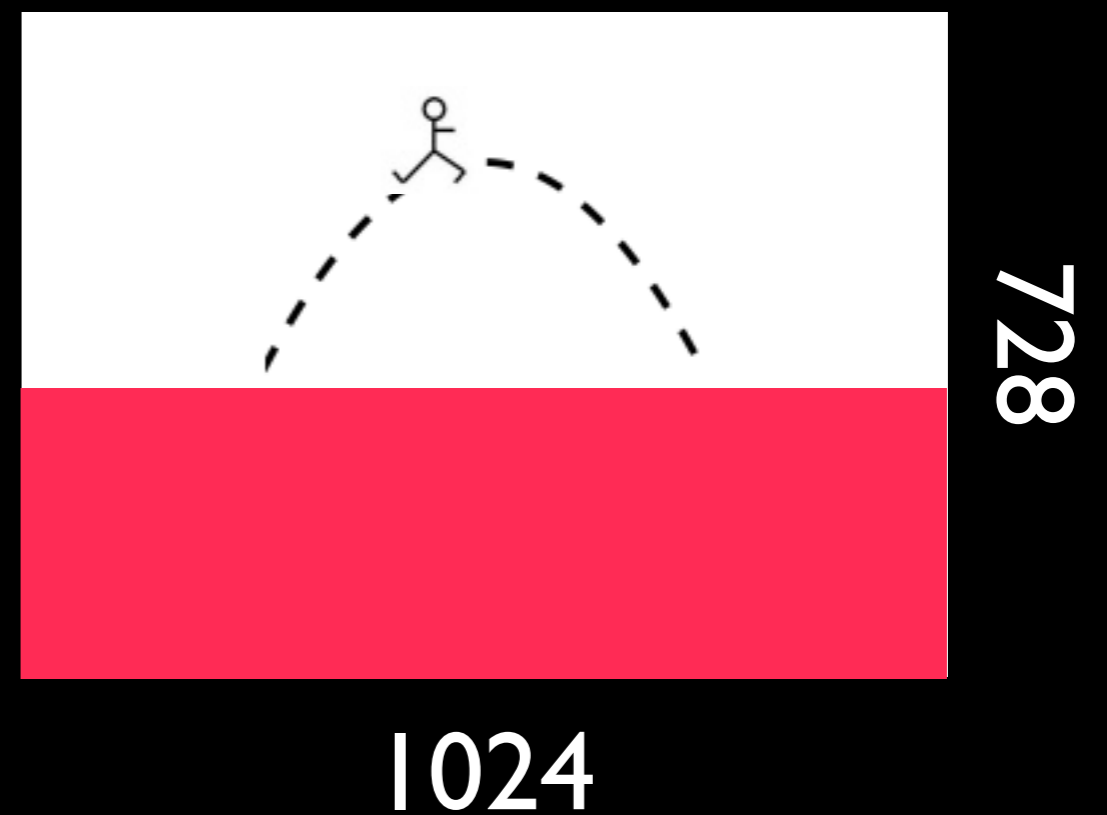
- x_acc is either
 - 0 if no keys are pressed
 - +/- a constant if left/right key is pressed
- each update, the x_vel should have the x_acc added to it
- similarly, the x_pos should have the x_vel added to it each update

On the ground

- x_acc is either
 - +/- a constant if left/right key is pressed
 - +/- a constant which is the opposite sign of the x_vel (deceleration due to friction)
- each update, the x_vel should have the x_acc added to it
- similarly, the x_pos should have the x_vel added to it each update

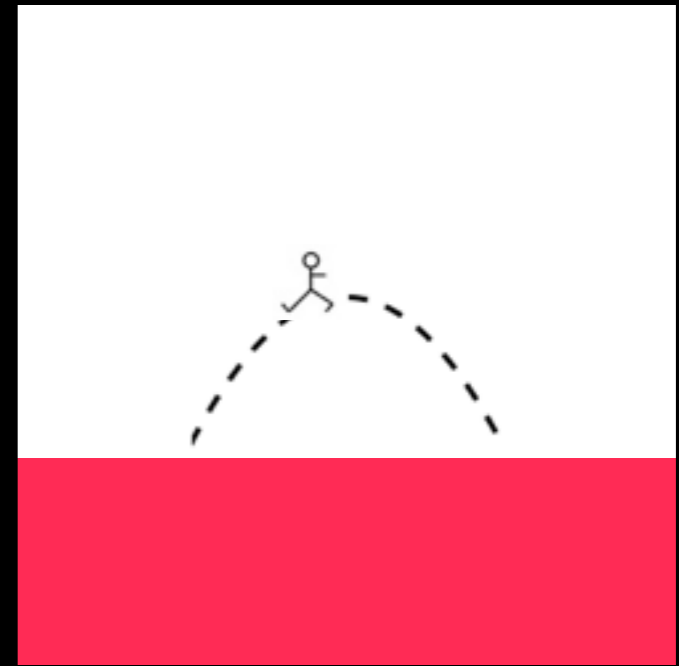
Physics by the Numbers

- Givens
 - 80 FPS
 - $x_vel = 3$
- Result
 - Player moves 240 pixels every second
 - Can cross screen in 4.2 seconds



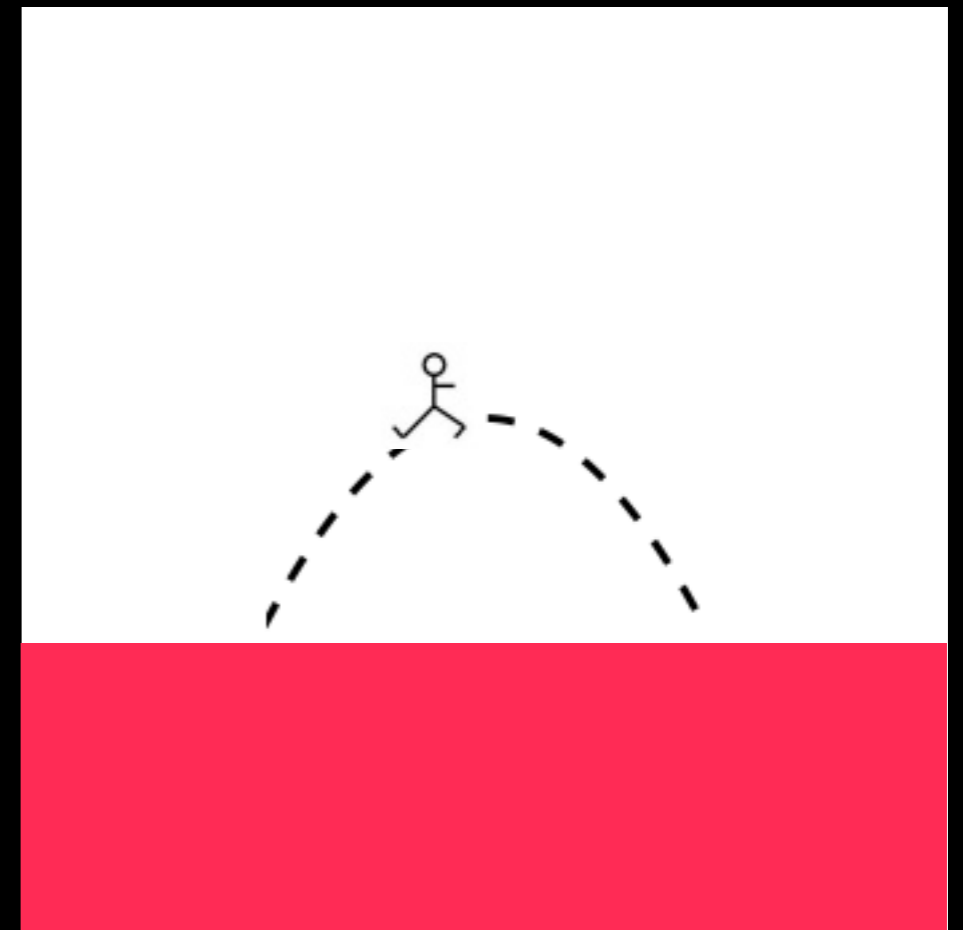
Vertical Physics

- Givens
 - 80 FPS
 - $v_{\text{initial}} = -5$
 - $\text{max_height} = 30$ pixels
 - $\text{time to max_height} = 1 \text{ second} = 80 \text{ updates}$
- Classic Physics Formula
 - $h = 0 + vt + 0.5at^2$
 - $30 = -5(80) + 0.5a(80)^2$



Vertical Physics

- Classic Physics Formula
 - $30 = -5(80) + 0.5a(80)^2$
 - $a = 0.134$
- Playing with the given initials can change the jump
- Greater the velocity, the longer the character will stay near the top of the jump



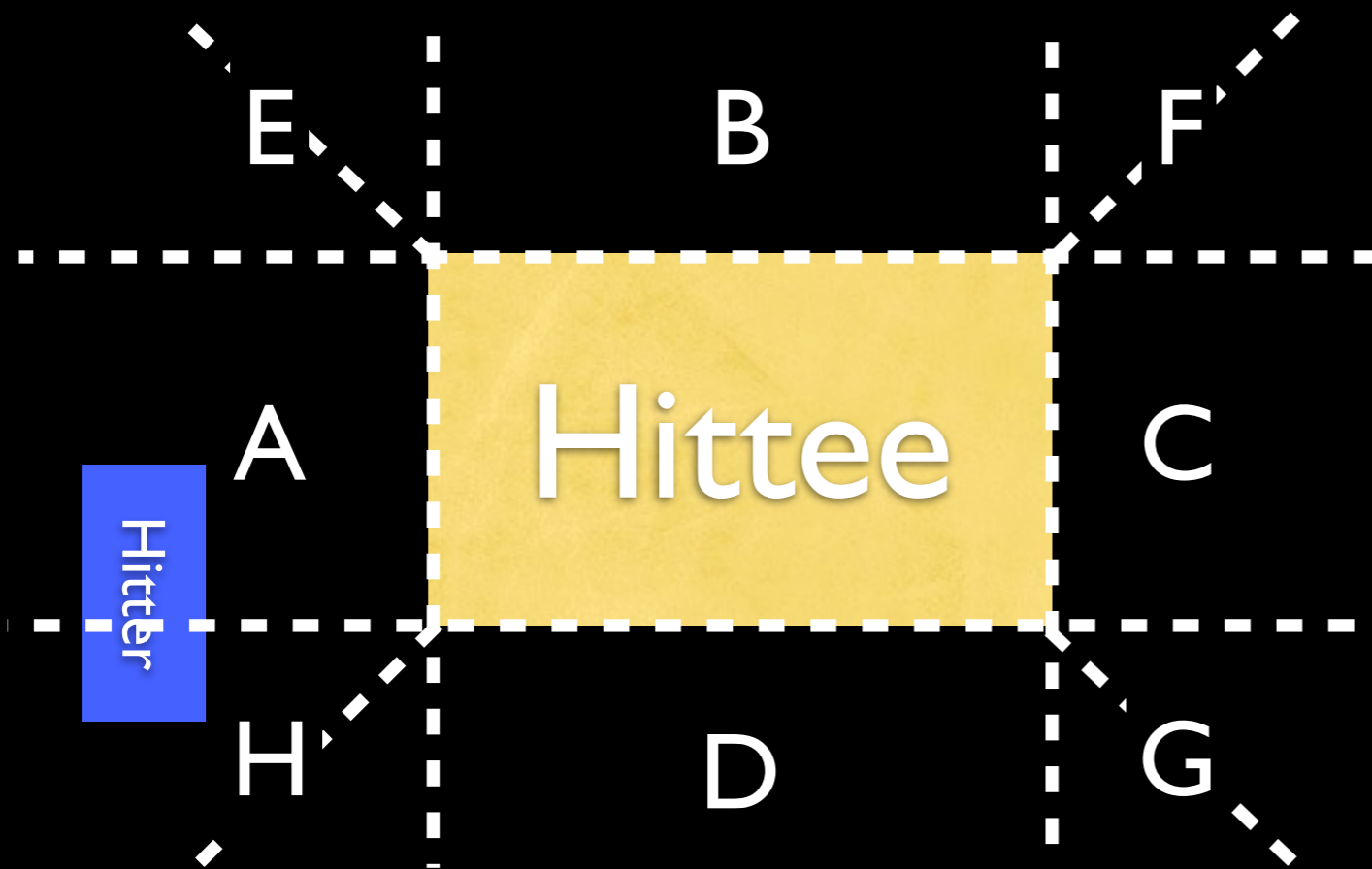
Hit Detection

The Eight Regions

Hittee

Hitler

Hit Detection

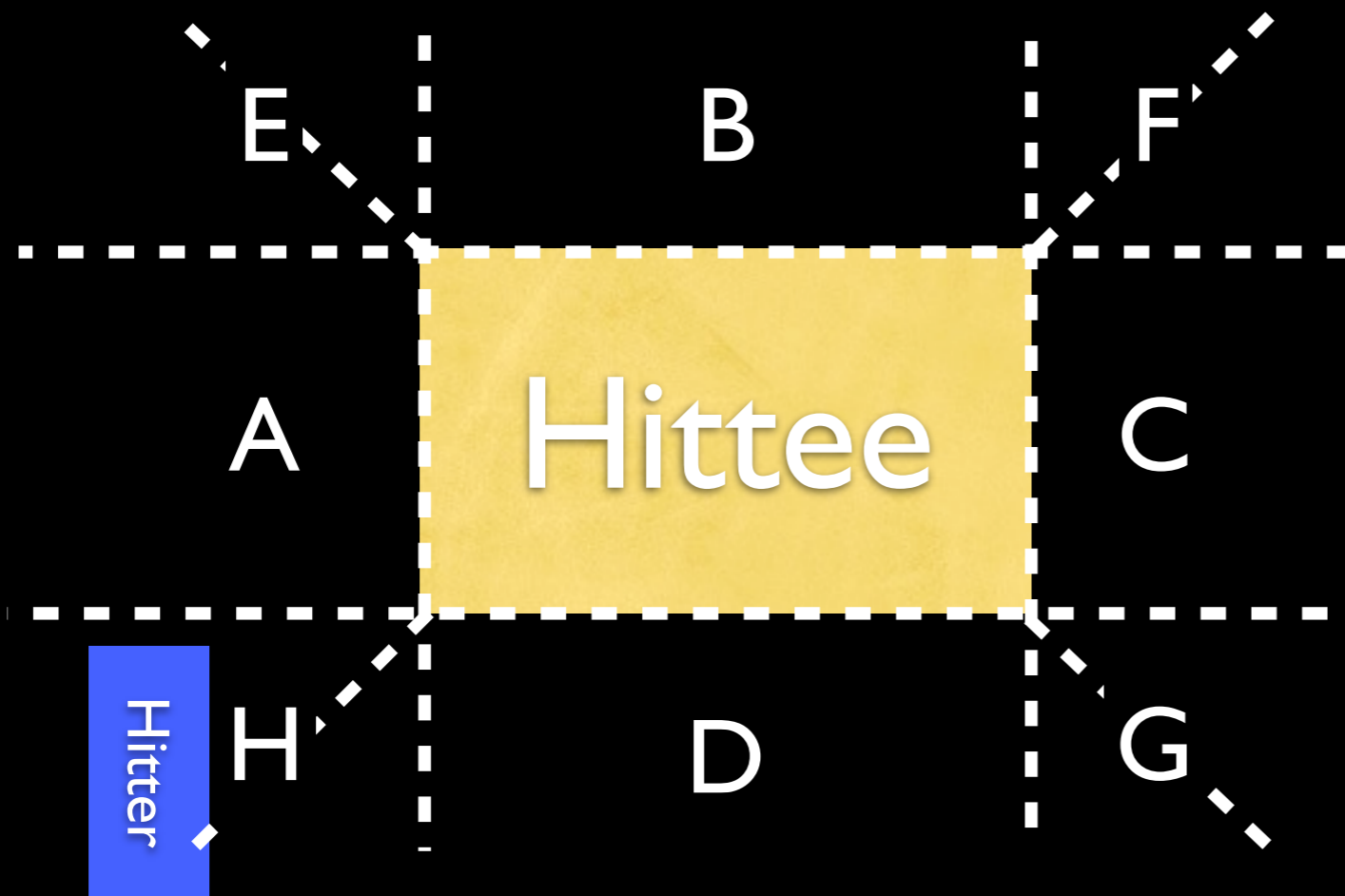


Pre-intersection
Diagram

Hitter is in
region A

The Hitter is in region A, B, C, or D if ANY part of the Hitter's OLD rectangle is in A, B, C, or D

Hit Detection

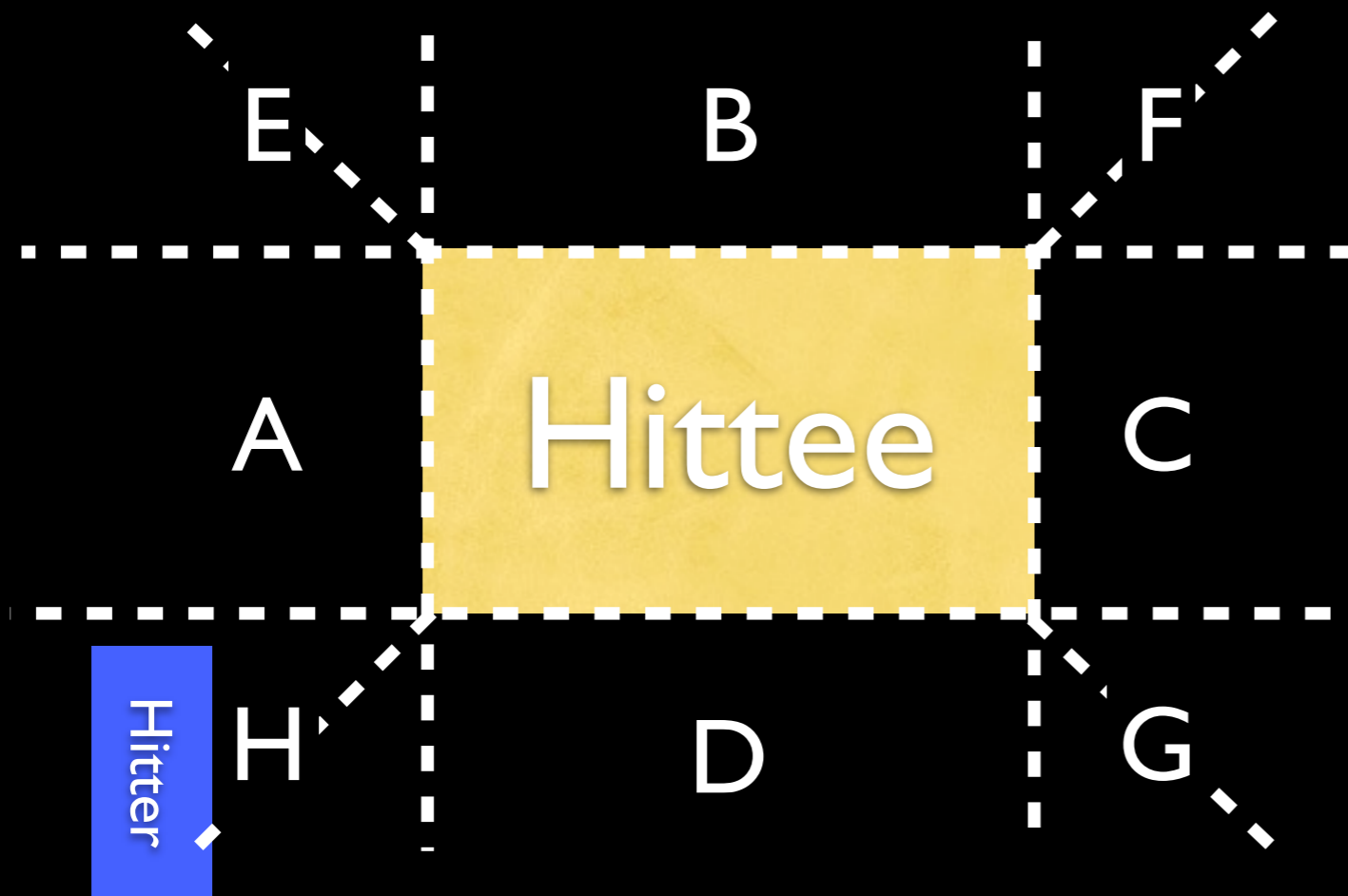


Pre-intersection
Diagram

Hitter is in
region H, to the
left of the Hittee

E, F, G, and H are split into two, based on the center of the Hitter.

Why all this work?



We need to know if the Hitter is coming from the left or from the right.

Big deal when Mario is trying to land on a surface.

Hit Detection

- Can use Rectangle2D.Double's class's intersection method
- Rectangle2D is an abstract class
- Rectangle2D.Double is a subclass which is declared as a field of Rectangle2D

Hit Detection

- When a hitter and a hittee collide...
 - The hitter should be told to push left, right, up, or down (to no longer intersect - usually)
 - The hittee should be told that it has been hit from above, below, the left or right

Hit Detection

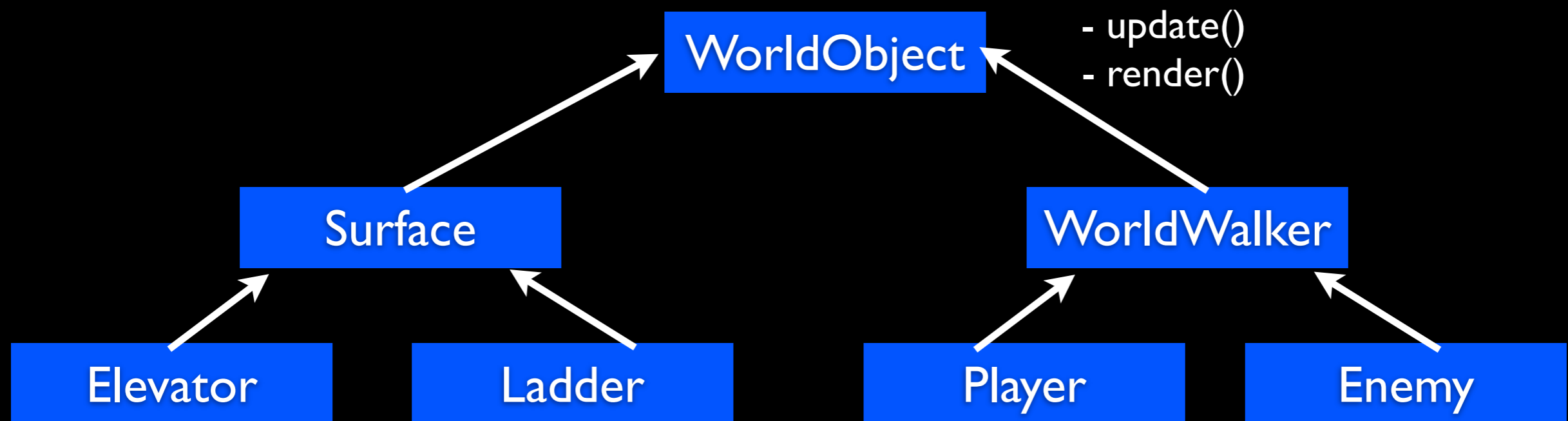
- When a character is on the ground, it is constantly being pushed up!
- Gravity pulls it into the ground, creating an intersection
- Intersection code pushes the character back up
- I've found leaving the character 1 pixel into the ground works well to reduce "bouncing"

Hit Detection

- Different types of objects may respond differently to "hits"
- When Mario intersects some surfaces, nothing happens unless he falls on them (no pushing).

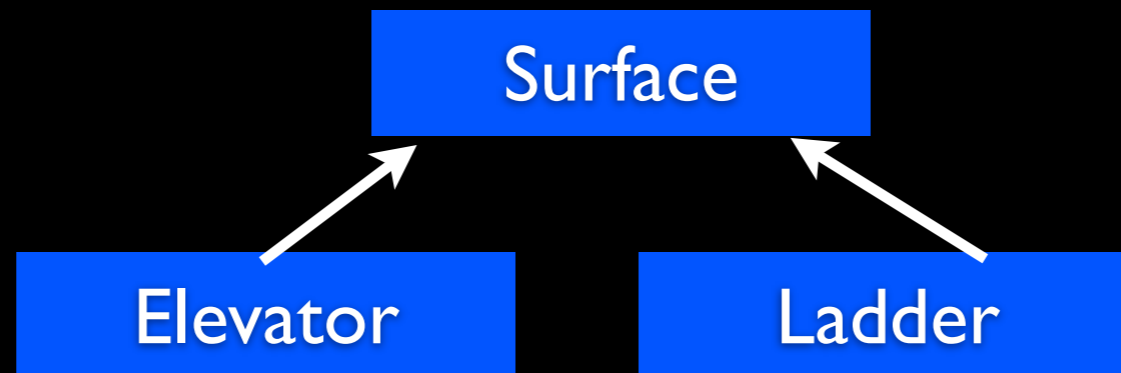


Class Hierarchy



- **WorldObject** and **WorldWalker** are probably abstract (maybe **Enemy**)
- Rest are probably concrete

Class Hierarchy



- Elevator's could be manual control (up/down button) or automatic looping
- Ladder's can be accomplished by using an invisible elevator beneath the Ladder (needs a little more work than that)

Miscellaneous Questions

- Can surfaces move?
- Are there ladders?
- Can certain things be picked up / thrown?
- Are there power ups?
- If you have fireballs, do they move in a straight line, or do they obey gravity?
- Can you jump through platforms from the side and below?

Miscellaneous Questions

- Do you kill enemies by shooting them, jumping on them, hitting them from below, etc?
- Is it important to kill bad guys to proceed in a level? (Mario versus Contra)
- Do surfaces have effects when jumped on or run into (springs or loose floors), hit from below (item blocks), etc

Where to go from here

- You need a SPECIFIC plan
 - Plan out features now
 - Adding them later could be messy, next to impossible, or break most other things
- Look for inspiration here:

http://en.wikipedia.org/wiki/Platform_game