

# Creating a Platformer in Game Maker

Foundations of Interactive Game Design  
Prof. Jim Whitehead  
February 7, 2007



Creative Commons  
Attribution 2.5

[creativecommons.org/licenses/by/2.5/](http://creativecommons.org/licenses/by/2.5/)

UC SANTA CRUZ



# Upcoming Events and Assignments

- **Game Concept Document**
  - ▶ Extended deadline until Friday
  - ▶ May turn in today, if ready
  - ▶ SOE Web servers experienced filesystem failure
  - ▶ Pages intermittently unavailable over past 24 hours
- **Gamelog**
  - ▶ Still due Friday
  - ▶ Game of your choice
- **Work Breakdown and Schedule**
  - ▶ Due next week, on Wednesday

# RPG Maker/Game Maker/Art help

- **Today:** RPG Maker design session
  - ▶ Led by Nate Emond <[llama971@gmail.com](mailto:llama971@gmail.com)>
  - ▶ 5:30PM, E2 215
  - ▶ Note different room for this week
- **Thursday:** Game Maker help/design session
  - ▶ Earth & Marine Sciences, room B214
  - ▶ 4-5:10PM
- **Game Maker reduced cost license keys**
  - ▶ See me after class to pay for keys
  - ▶ I will sell all unclaimed keys on Friday
  - ▶ Full version of Game Maker on ITS machines as well

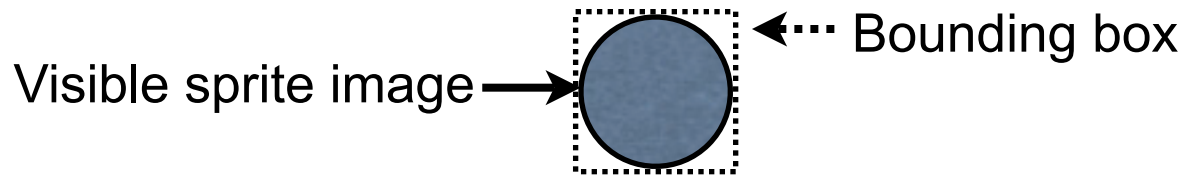
# Key Mechanics of Platformers

- To create a platform game, need to
  - ▶ Handle collision with platforms
  - ▶ Handle jumping
  - ▶ Scrolling view in large game world
  - ▶ Interactions with other enemies and items in gameworld
- Today, will focus on
  - ▶ collision detection
  - ▶ jumping
  - ▶ scrolling view

# Collision Detection

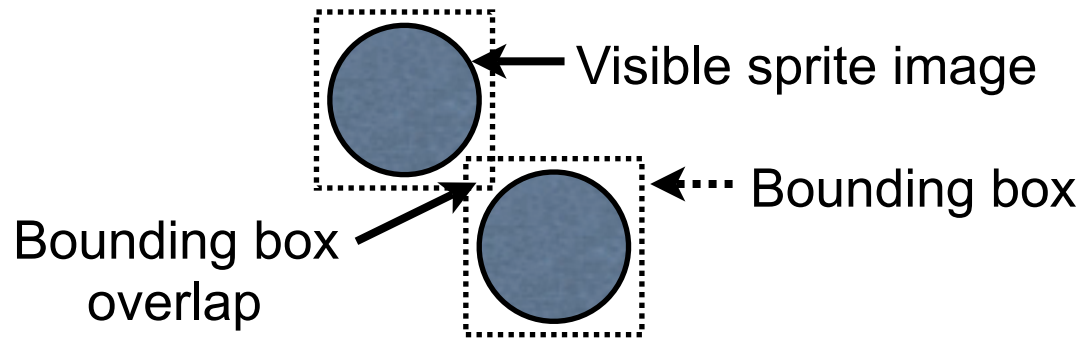
- Collision detection is the bane of platform game designers in Game Maker
- Several factors affect collision detection
  - ▶ Type of bounding box
    - ❖ Automatic, full image, manually defined bounding box (defined in Sprite)
  - ▶ Type of collision detection
    - ❖ precise, non-precise (defined in Sprite)
  - ▶ Whether object is solid
    - ❖ Defined in object
  - ▶ Actions taken in response to a collision event
    - ❖ Defined in object
  - ▶ May need to examine multiple dialog boxes to completely understand collision detection behavior

# Bounding Box



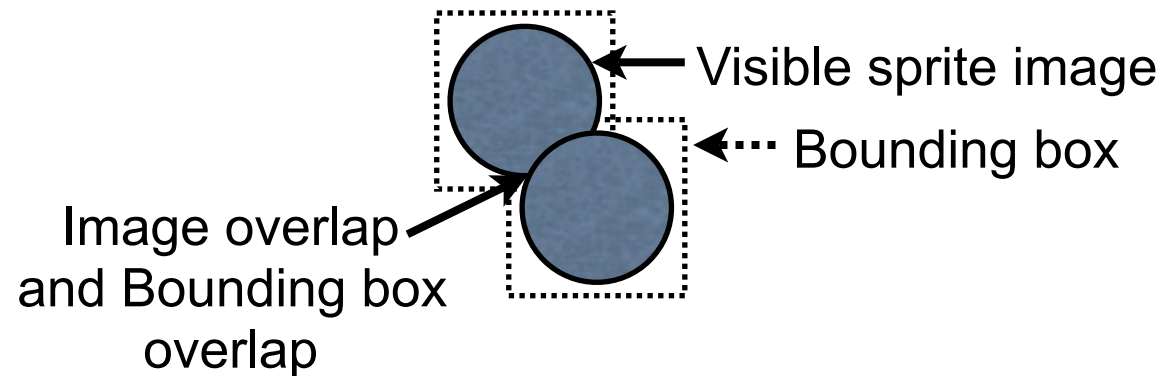
- Bounding box is a rectangle (or square) drawn around a sprite image, used for collision detection
  - ▶ Typically, is outer boundary of the sprite image
    - ❖ Game Maker selects this using the “Automatic” feature under “Bounding Box” in the Sprite dialog
  - ▶ Can manually set the size of the bounding box
    - ❖ Example: if creating a bullet hell (manic) shmup, may want to make the vulnerable region of the ship smaller than the visible sprite
    - ❖ Select “Manual” under “Bounding Box” in Sprite dialog, then fill in dimensions
  - ▶ Can also set to the full size of the sprite file
    - ❖ If sprite is defined in 48x48 pixel box, “Full image” option would make bounding box the full 48x48 pixels, even if sprite image is smaller

# Basic Collision Detection



## Non-precise collision detection:

Just overlap of bounding boxes.  
May mean sprite images do not overlap (player would visually perceive a near miss)



**To toggle between these collision detection modes:**  
Choose “Precise collision checking” in Sprite dialog box

## Precise collision detection:

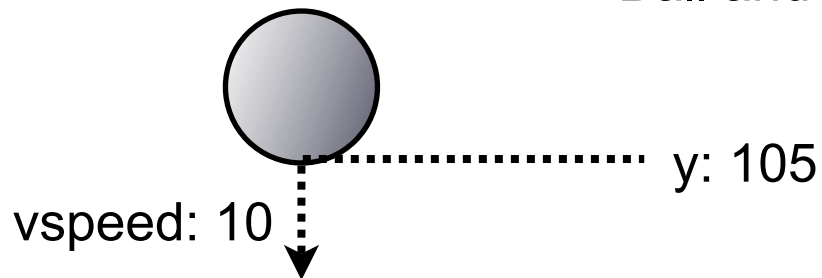
Must have overlap of bounding boxes **and** sprite images must overlap.  
Best collision detection, but more computationally expensive. Unless you choose otherwise, this is the collision detection behavior sprites use.

# Step sequence in Game Maker

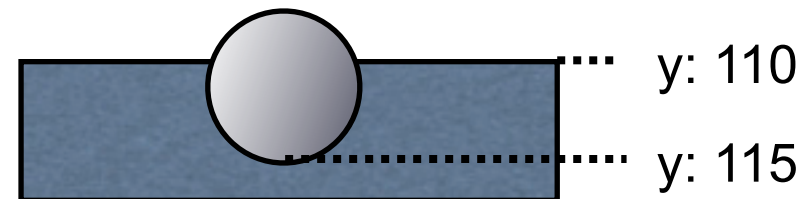
- Begin step: generate Begin Step event
- Save current position of object
- Compute new position
  - ▶  $x = \text{old\_x} + \text{hspeed}$
  - ▶  $y = \text{old\_y} + \text{hspeed}$
  - ▶ But, don't move sprite just yet!
- Determine if there is a collision at the new position
  - ▶ If either object in collision is **solid**: reset position:  $x = \text{old\_x}$ ,  $y = \text{old\_y}$
  - ▶ In both cases (solid, not solid), now generate Collision event
- Generate Step event
- Update visual representation of sprite on screen
  - ▶ Draw sprite at new location  $(x, y)$
- End step: generate End Step event

# My sprite is embedded in a wall: why?

Ball and Wall are both **not solid**



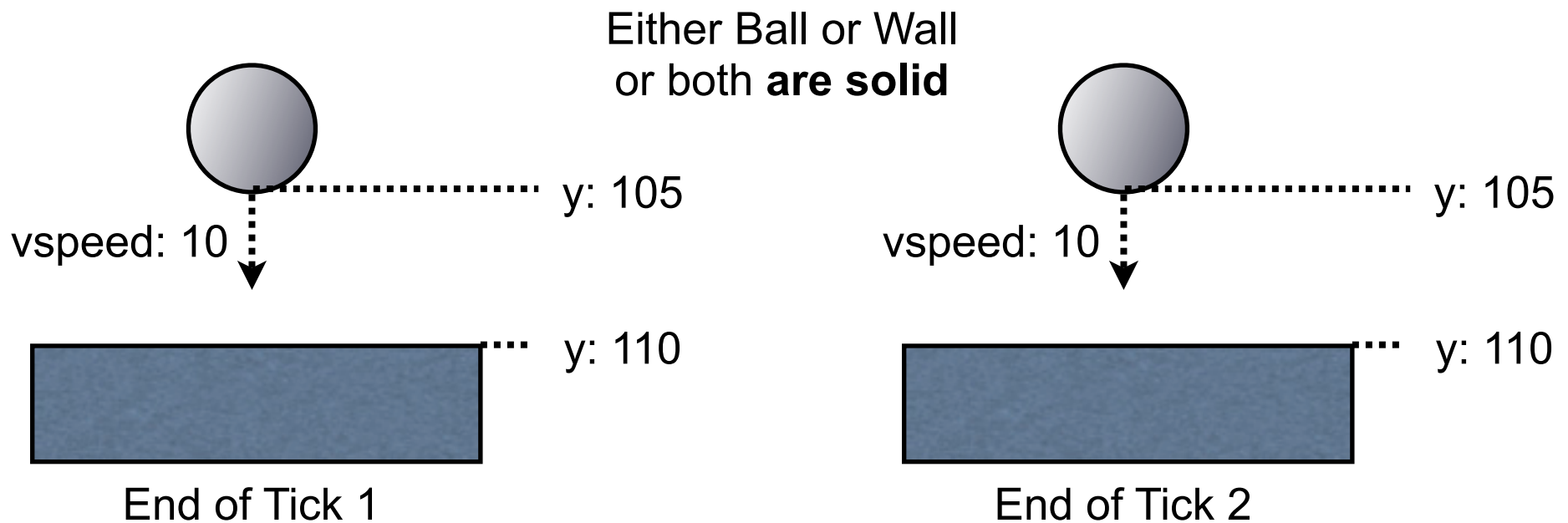
End of Tick 1



End of Tick 2

- In tick 2 (both objects not solid)
  - ▶ Compute new position:  $y = \text{old\_y} + \text{vspeed}$  ( $y = 105 + 10 = 115$ )
  - ▶ Actions in collision event just set vspeed to 0
    - ❖ Actions do not modify position of the object
  - ▶ Visual depiction of objects updated to new position
  - ▶ Sprite is now embedded in a wall

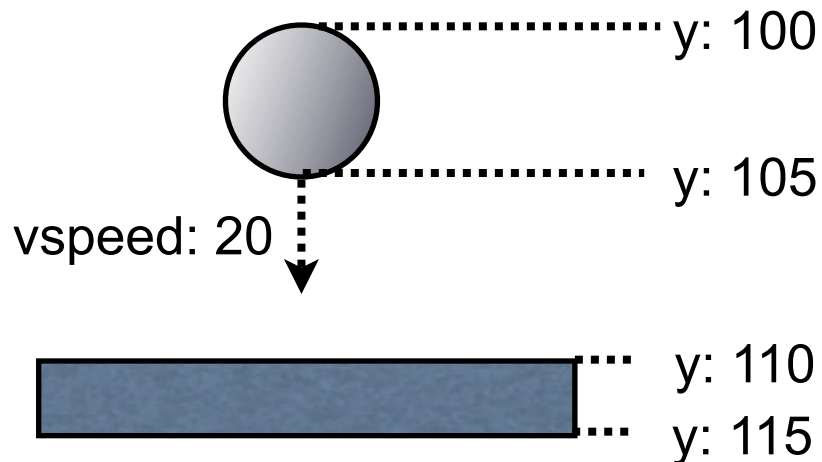
# My sprite stopped above the wall: why?



- In tick 2 (one or both objects solid)
  - ▶ Store old positions  $old\_y = y$ ,  $old\_x = x$
  - ▶ Compute new position:  $y = old\_y + vspeed$  ( $y = 105 + 10 = 115$ )
  - ▶ Check for collision at new position: **yes**
  - ▶ **Revert** to old position ( $y = y\_old = 105$ )
  - ▶ Do actions in collision event
    - ❖ Set vspeed to 0
  - ▶ Draw sprite: now stopped in mid-air

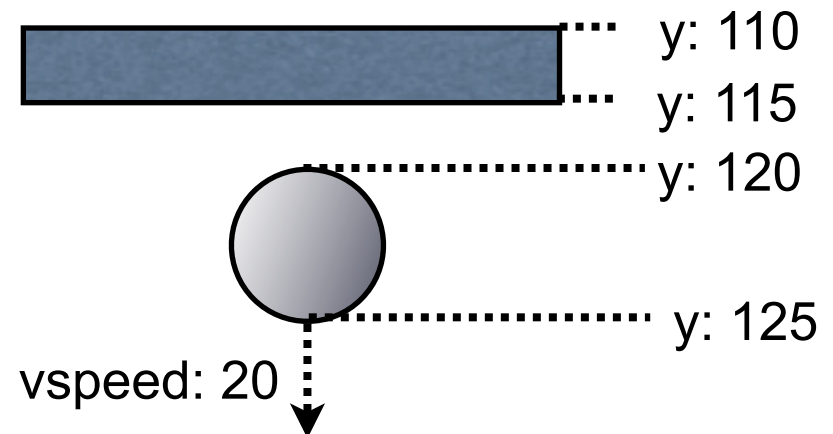
# My sprite fell through a wall: why?

End of Tick 1



**Will occur for solid and non-solid objects**

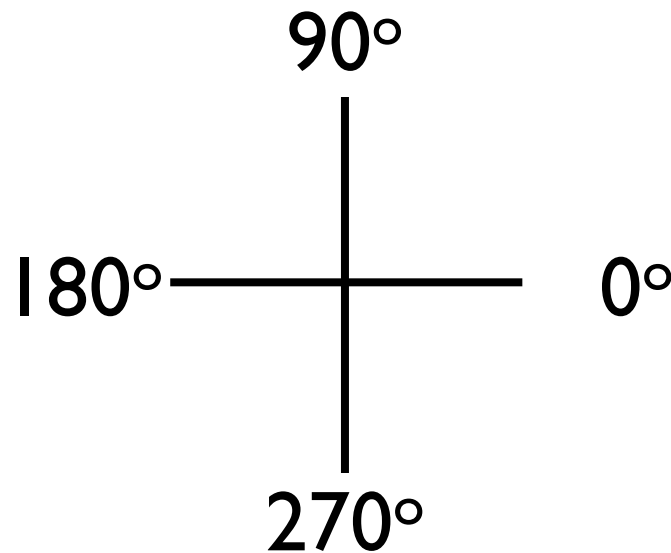
End of Tick 2



- Velocity of ball is so great that in one tick its position is incremented such that it never intersects the wall.

# Directions in Game Maker

- Direction angles in Game Maker are different from normal.



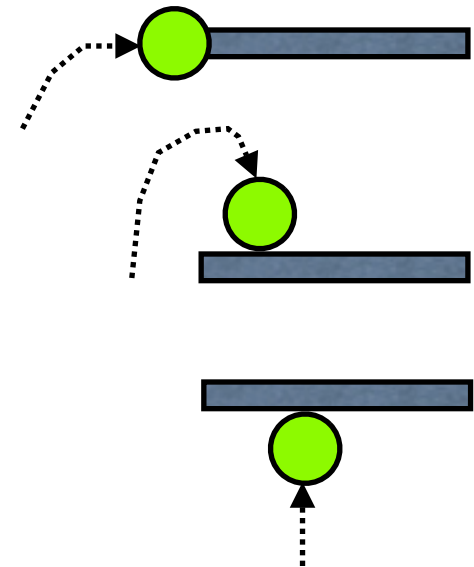
# Gravity

- Can set “gravity” in Game Maker
- Very simple
  - ▶ Define a speed and a direction
  - ▶ Every step, the speed is added to the player’s current speed
  - ▶ This is a  $dv/dt$  (change in velocity per change in time), hence is acceleration
  - ▶ Normal gravity is an acceleration downwards of  $9.8 \text{ m/s}^2$
  - ▶ To create sensation of downward gravity in Game Maker
    - ❖ Use “Set the gravity” action
    - ❖ Have direction be 270
    - ❖ Set speed to 2
      - Arbitrary value
      - This speed may vary with your gameworld, tune until it feels right

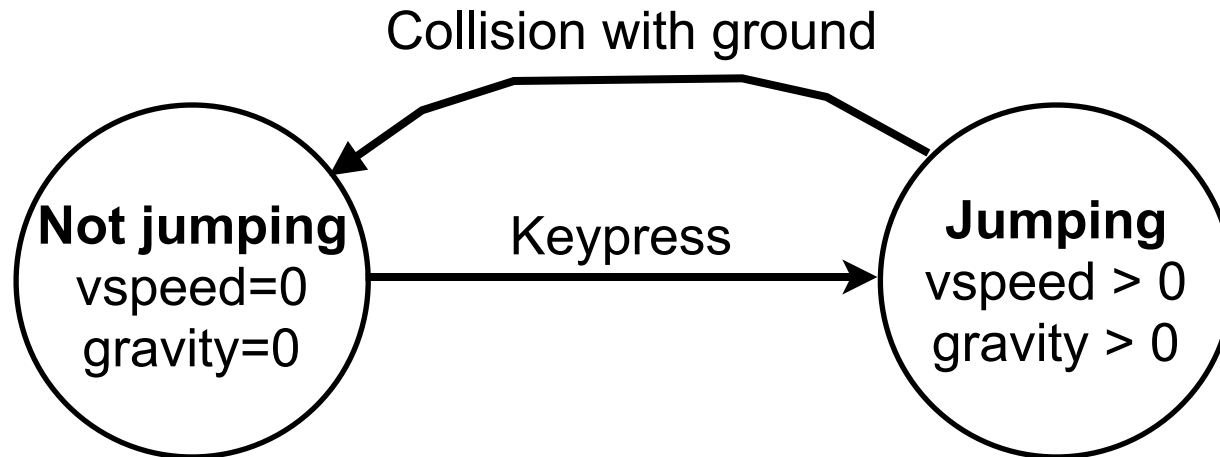
# Jumping

- Jumping behavior

- ▶ Press a key, and player jumps up
- ▶ Pressing key while in the air should not result in more jumping
  - ❖ Unless you intentionally want a “double-jump”, in which case you only want to allow two, and only two jumps.
- ▶ Jumping should not affect side-to-side movement
- ▶ Need to decide if player can change side-to-side direction in mid-air
- ▶ Need to handle collisions correctly
  - ❖ Not going into the side of platforms
  - ❖ Correctly landing on the top of platforms
  - ❖ Correctly handling jumps up into platforms



# Handling Jump Keypress



- Need a simple **state machine**
- Create a variable called “jumping” on player
  - ▶ Set to 0 initially
  - ▶ Can only jump when it is 0
  - ▶ Set to 1 when jumping

# Details of Creating State Machine

- Initialize variable jumping
  - ▶ Add “Set variable to a value” action to Create event for player
  - ▶ “Var” square on Control tab in Object window
    - ❖ Variable: jumping
    - ❖ Value: 0
    - ❖ Relative: must be unchecked
- Add jumping logic
  - ▶ Add keyboard event
  - ▶ Add variable conditional
    - ❖ Octagon with “Var” on Control tab of Object window
    - ❖ Applies to “self”
    - ❖ Variable: jumping
    - ❖ Value: 0
    - ❖ Operation: equal to
    - ❖ Creates conditional: **if** jumping = 0 **then** ... {then behavior on next slide}

# Details of Creating State Machine (2)

- Then behavior if jumping = 0 in keypress event
  - ▶ Add a “Start of a block” (grey up triangle on Control tab)
    - ❖ Equivalent to curly braces found in many languages
    - ❖ Means: “group the following actions together until the end of block”
  - ▶ Set vertical speed to a negative number
    - ❖ Downward red arrow on Move tab of Object window
    - ❖ Yes, click the down arrow to make your player go up
    - ❖ Tunable value, try vert. speed of -25 to -30 for starters
  - ▶ Set variable jumping to 1
    - ❖ That is, we have shifted to the “jumping” state in our simple state machine
    - ❖ “Var” square on Control tab of Object
      - Applies to “self”
      - Variable: jumping
      - Value: 1
      - Relative: unchecked
  - ▶ Add an “End of a block” (grey down triangle on Control tab)

# Details of Creating a State Machine (3)

- OK, have initialization and state transition to jumping state
- Now need transition back to non-jumping state
- When a collision is detected, need to set jumping to 0
  - ▶ In collision event with Wall
    - ❖ Set vertical speed to 0
      - Down red arrow on Move tab of Object
      - Set vert. speed to 0
    - ❖ Set variable jumping to 0
      - Var square on Control tab of Object
      - Applies to “self”
      - variable: jumping
      - value: 0

# Drawbacks of the State Machine

- Collision events between player and wall do not distinguish the three cases
  - ▶ Ball landing on top of wall
  - ▶ Ball hitting side of platform
  - ▶ Ball hitting underside of platform
- Currently, state machine leads to following behavior
  - ▶ Sticking to side of a platform (ball or wall is solid)
    - ❖ Player receives collision event, and sets vspeed to 0
    - ❖ Gravity should cause player to fall, but gravity never has chance to affect position
    - ❖ Solid object behavior means collision event occurs before ball actually moved
    - ❖ Horizontal movement means ball keeps going towards platform
    - ❖ Results in a collision every step!

# Drawbacks of the State Machine

- Hitting the underside of the platform also has problems
  - ▶ Ball strikes underside of platform
  - ▶ Causes collision event
    - ❖ Sets vspeed to 0
    - ❖ Also sets jumping to 0
      - Can't distinguish between landing on platform, and jumping up into underside of platform
  - ▶ Result: undesired perpetual double-jump ability
- How do I fix all of this?
  - ▶ Complicated: details in class on Friday...