DEPARTMENT OF INFORMATION SYSTEMS AND COMPUTER SCIENCE

Platformer Basics

Physics VS “Physics”

Lecture Time!

- Homework: A Preview
- Platformer: Basic Controls
- Physics: Adjustments
- Responsiveness: Avoiding Limpness and Rigidity
- Homework: Specs

Platformer

- A sub-type under the action game genre, the *platformer* is a game where the player-controlled character can move and jump to get to a destination or objective
- Note that the player must input a command to trigger the jump
  - If the jumping is automated and the height of the jumps are insignificant, the game is not a platformer

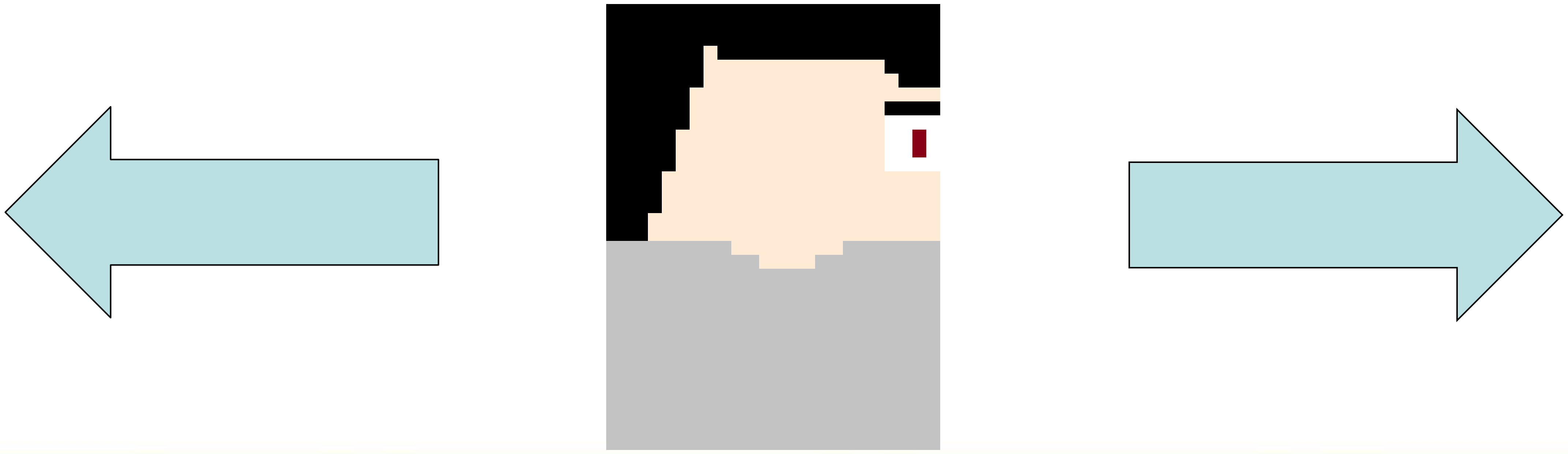
Platformer

- Basic controls for a 2D platformer
  - Moving horizontally
  - Jumping

Nyanko Days, episode 1

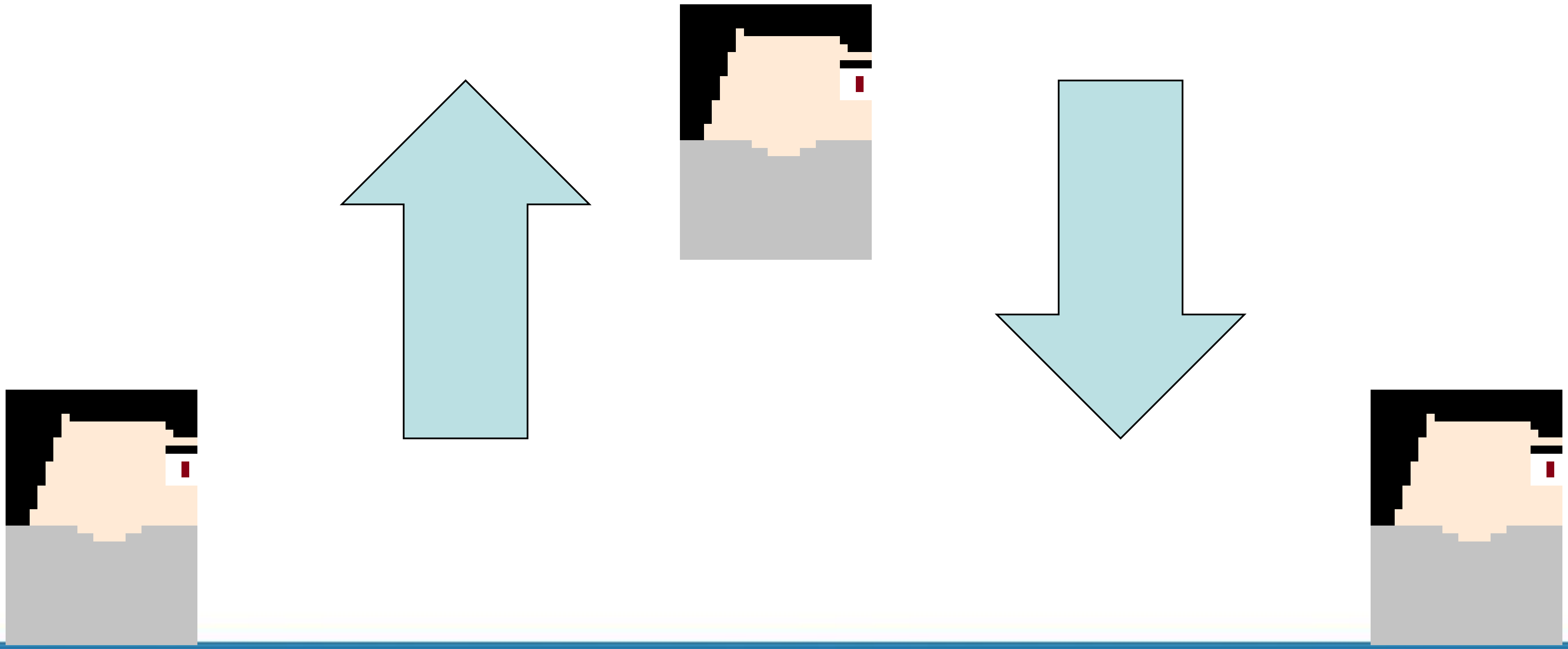
Moving Horizontally

- Pressing a button to move left/right affects character's acceleration
- Releasing the button normally triggers deceleration to zero velocity



Jumping

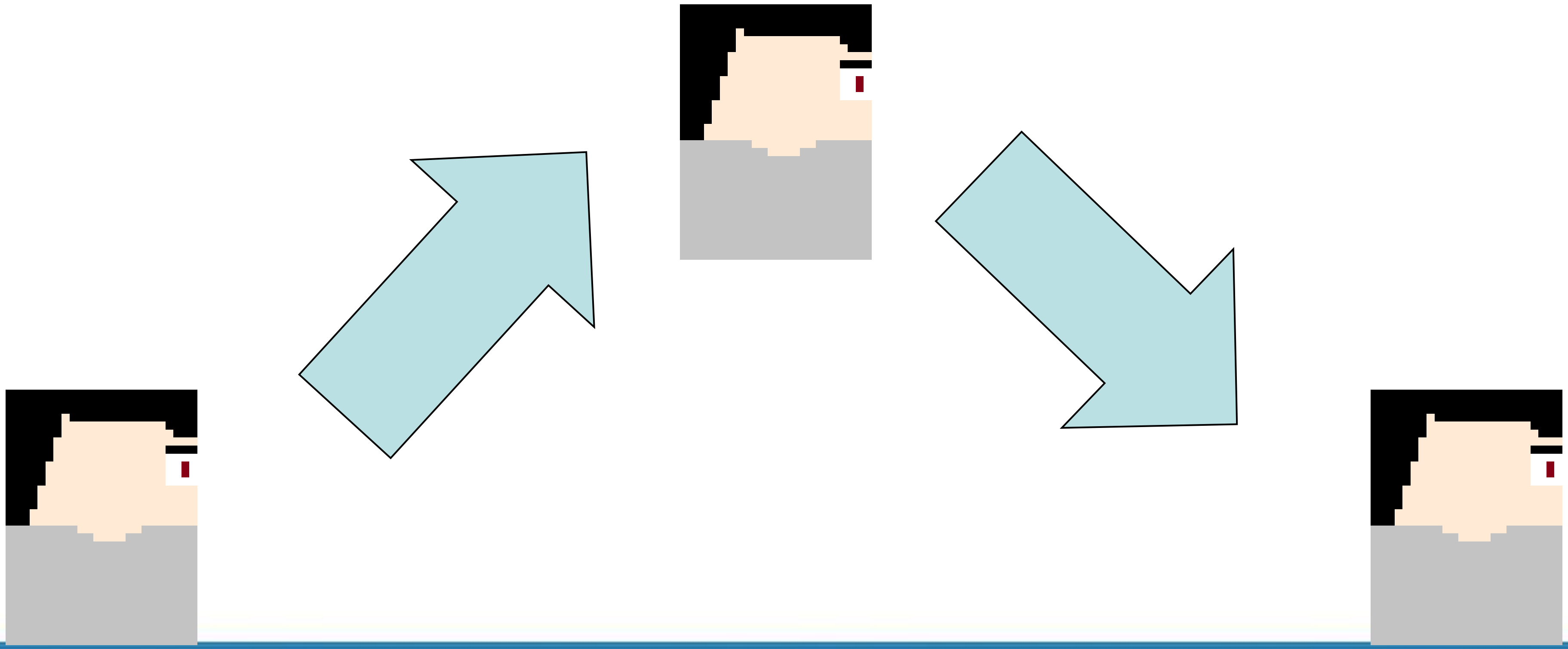
- Pressing a button to jump causes an instantaneous jump
- But what goes up (or what runs off a platform) must come down



DISCS

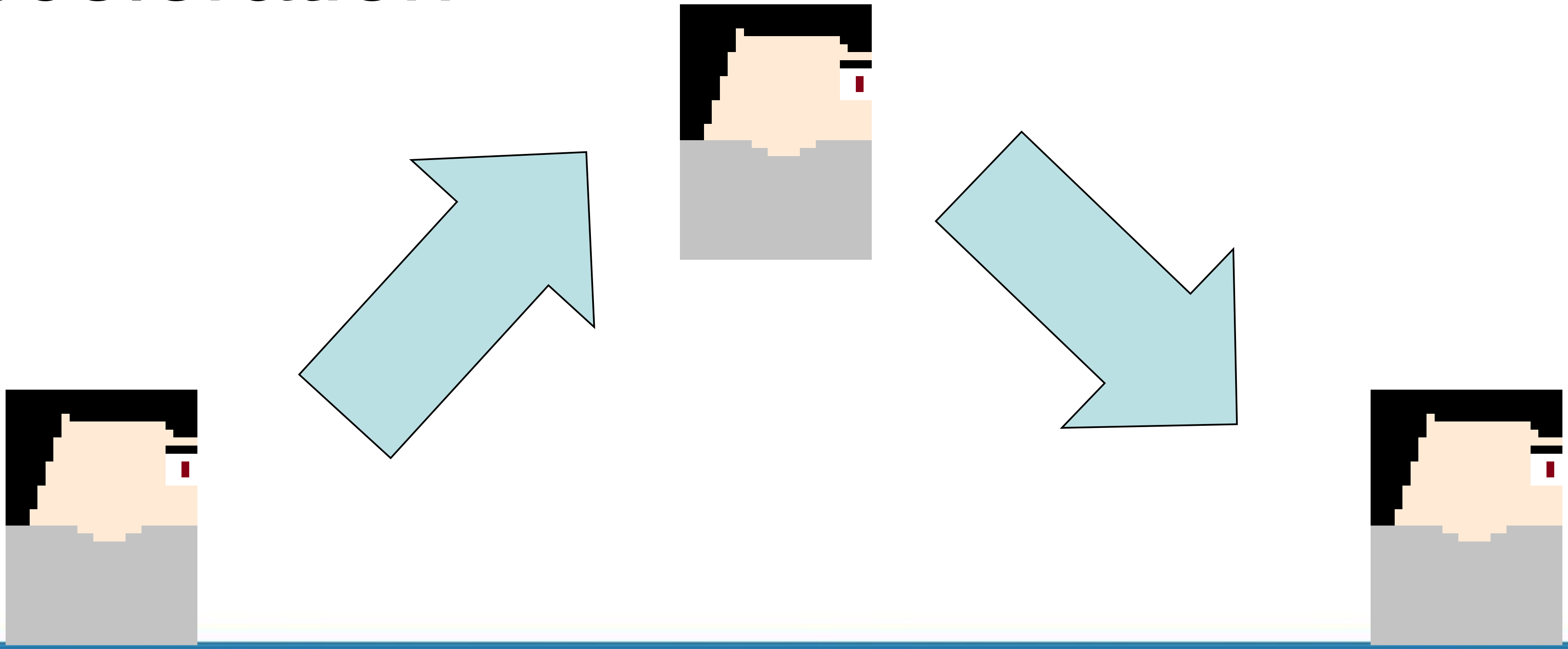
Handling Movement in 2D

- Character then moves along 2 axes at most simultaneously



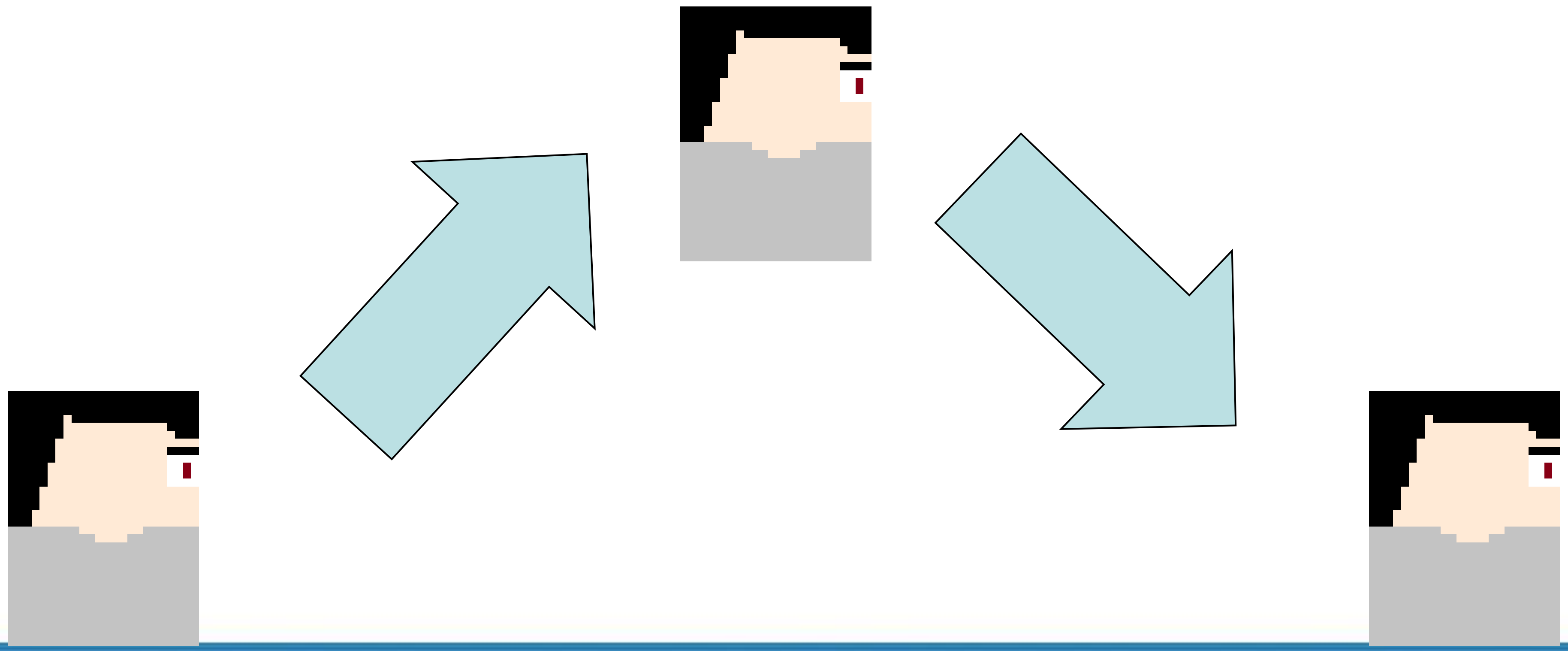
Handling Movement in 2D

- 3 attributes for each axis per entity
  - Position
  - Velocity
  - Acceleration



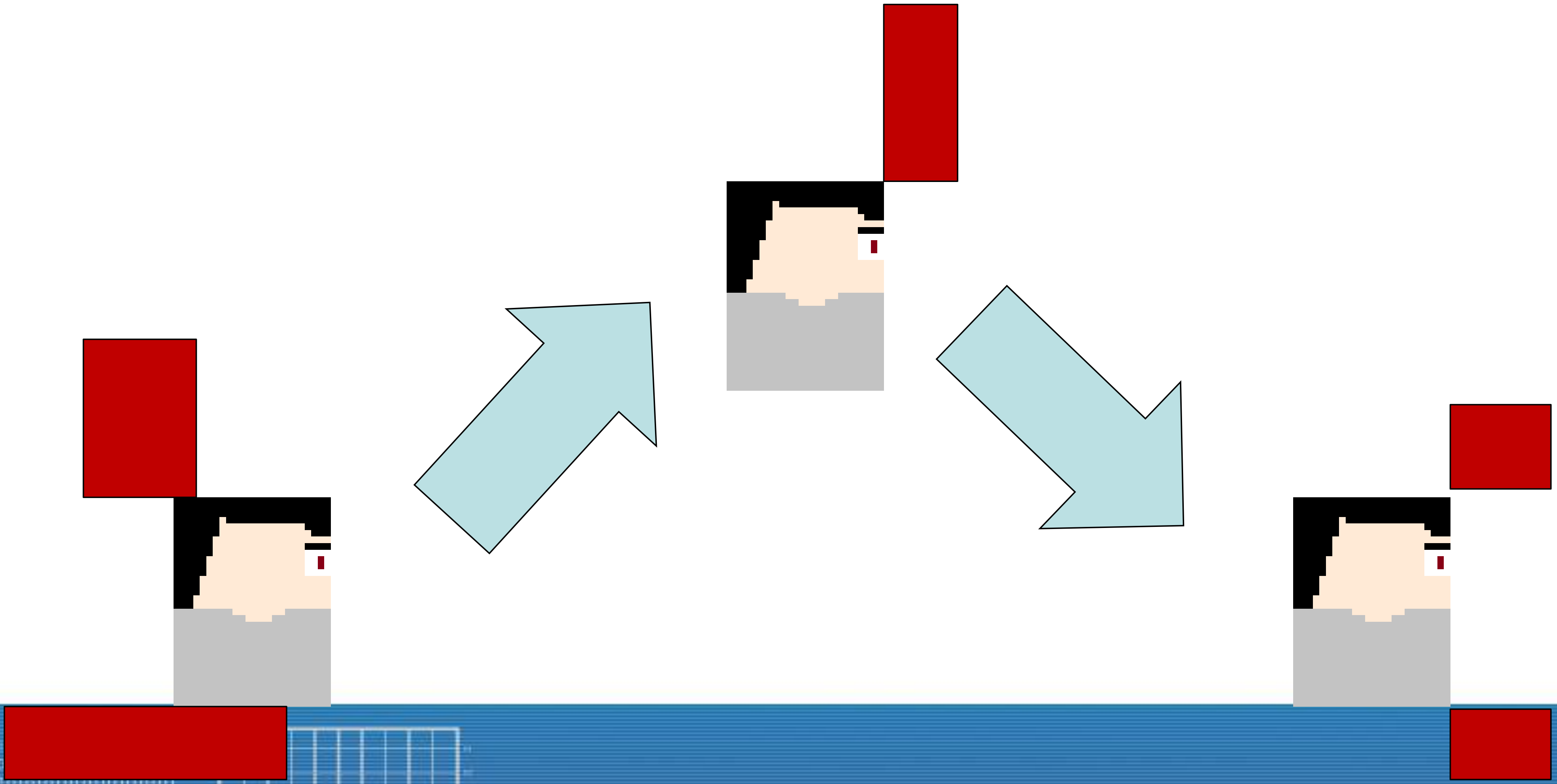
Handling Movement in 2D

- Given that horizontal movement and jumping are two different commands, perhaps they can be handled separately?
  - Then just combine into one vector?



Handling Movement in 2D

- Movement seems easy to implement until you realize you need to handle collision



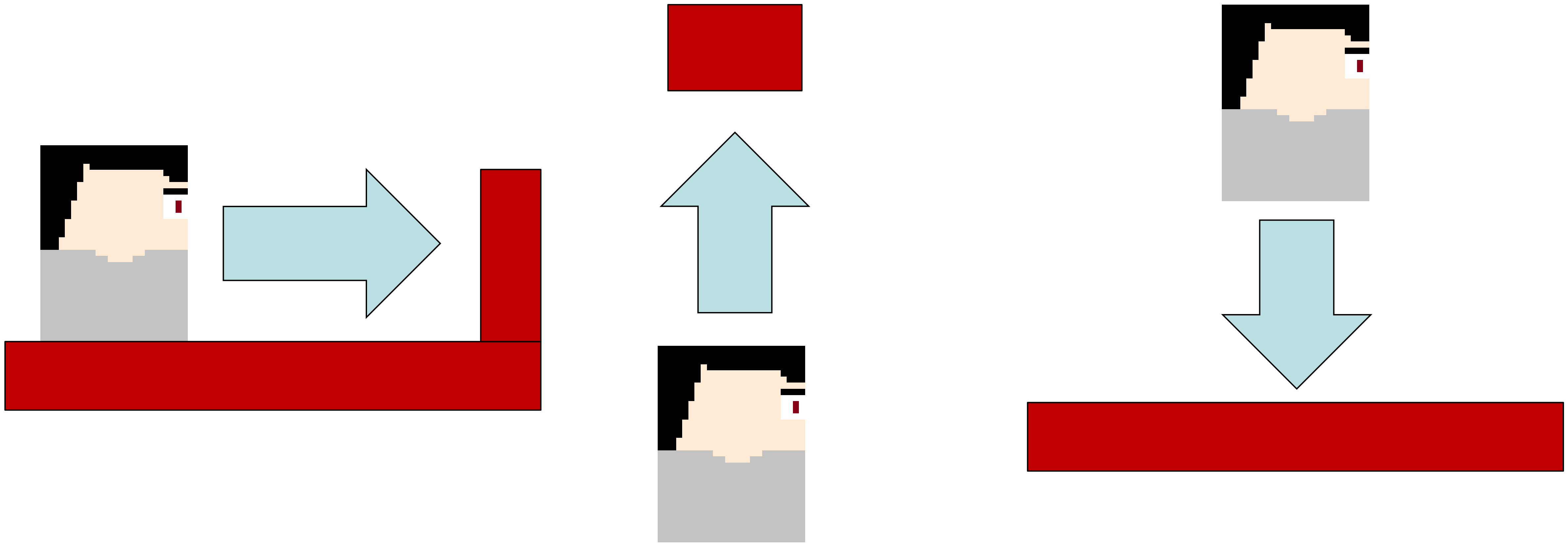
Handling Collisions

- While only mentioned briefly in CS179.14A, one way to handle collisions realistically would be to:
  - Check for intersection
  - Rewind movement by X% if there is,
  - Repeat until you get no intersection
- But let’s not go that far…

DISCS

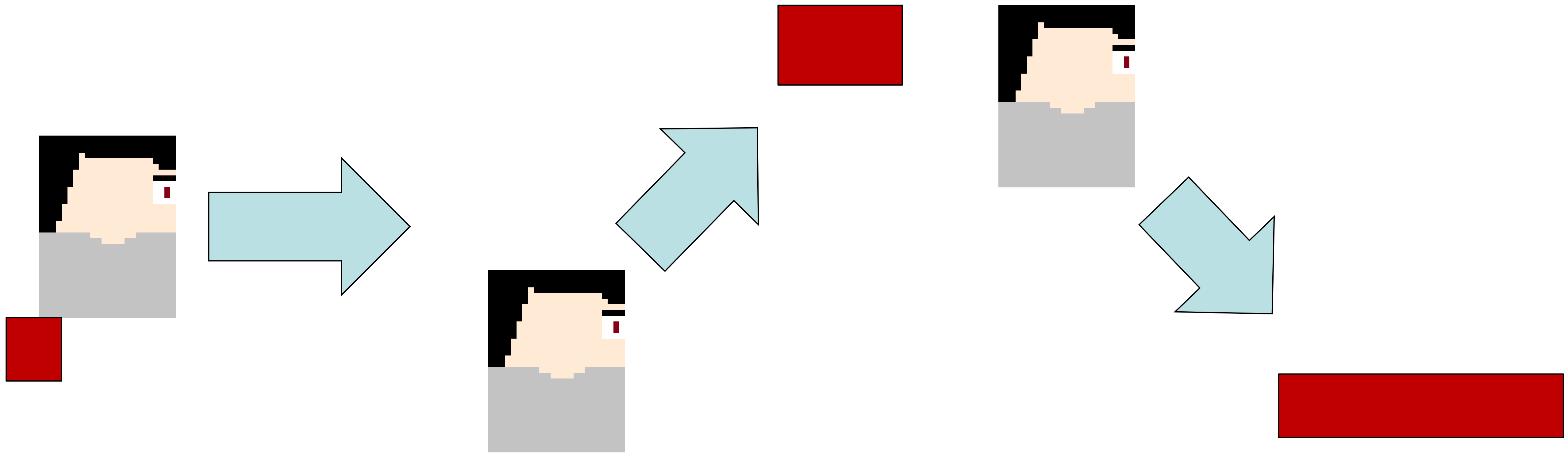
Handling Collisions

- Air hockey homework might not help here because the collision response looks different depending on the actual collision



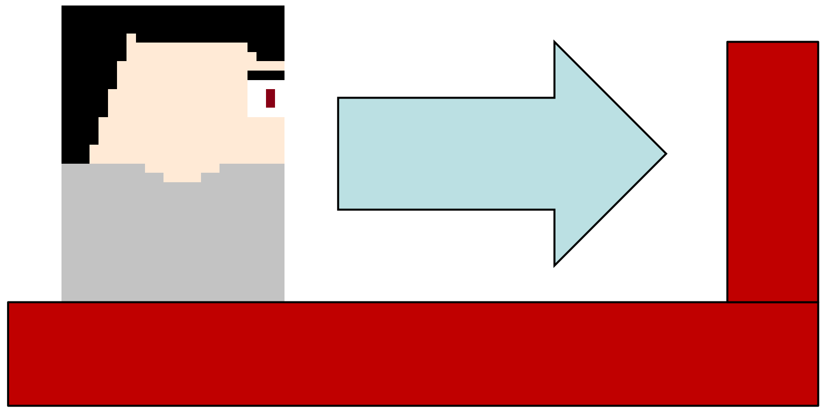
Handling Collisions

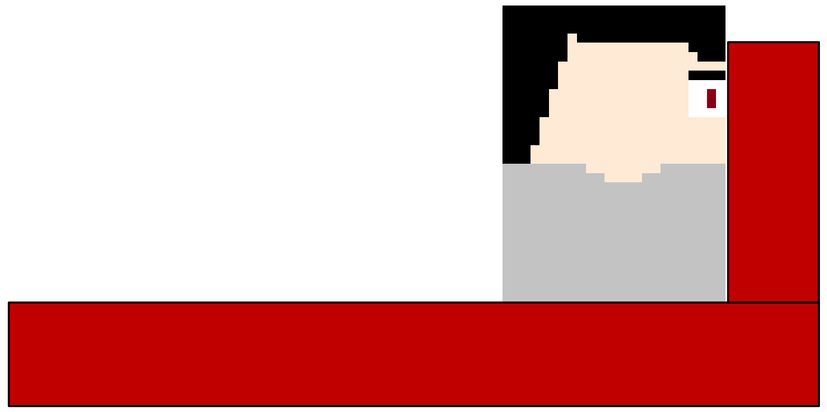
- Air hockey homework might not help here because the collision response looks different depending on the actual collision



Back to Movement

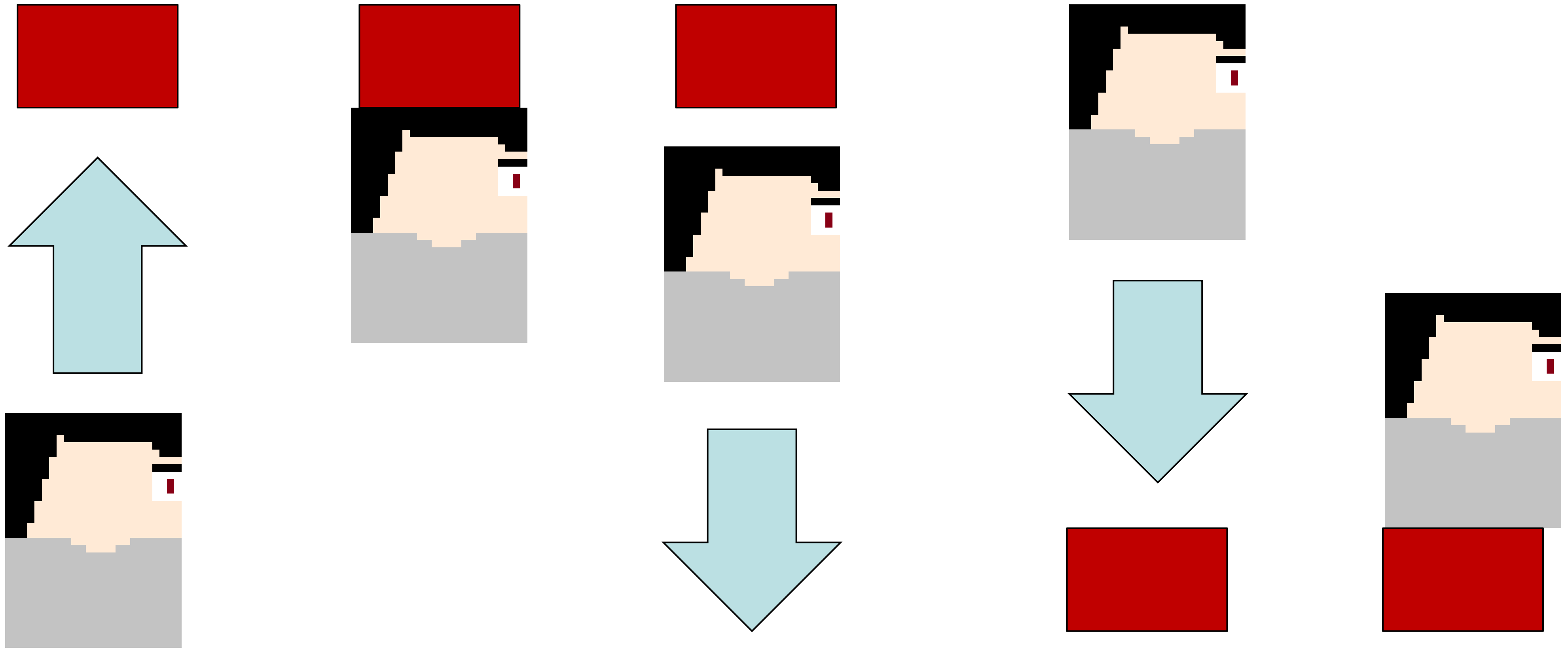
- Hitting a wall from the left or right should force character horizontal velocity to zero
  - Note that after all collisions have been resolved, the character should not look like it is overlapping with the wall





Back to Movement

- Hitting a wall from the top or bottom should force character vertical velocity to zero, but…



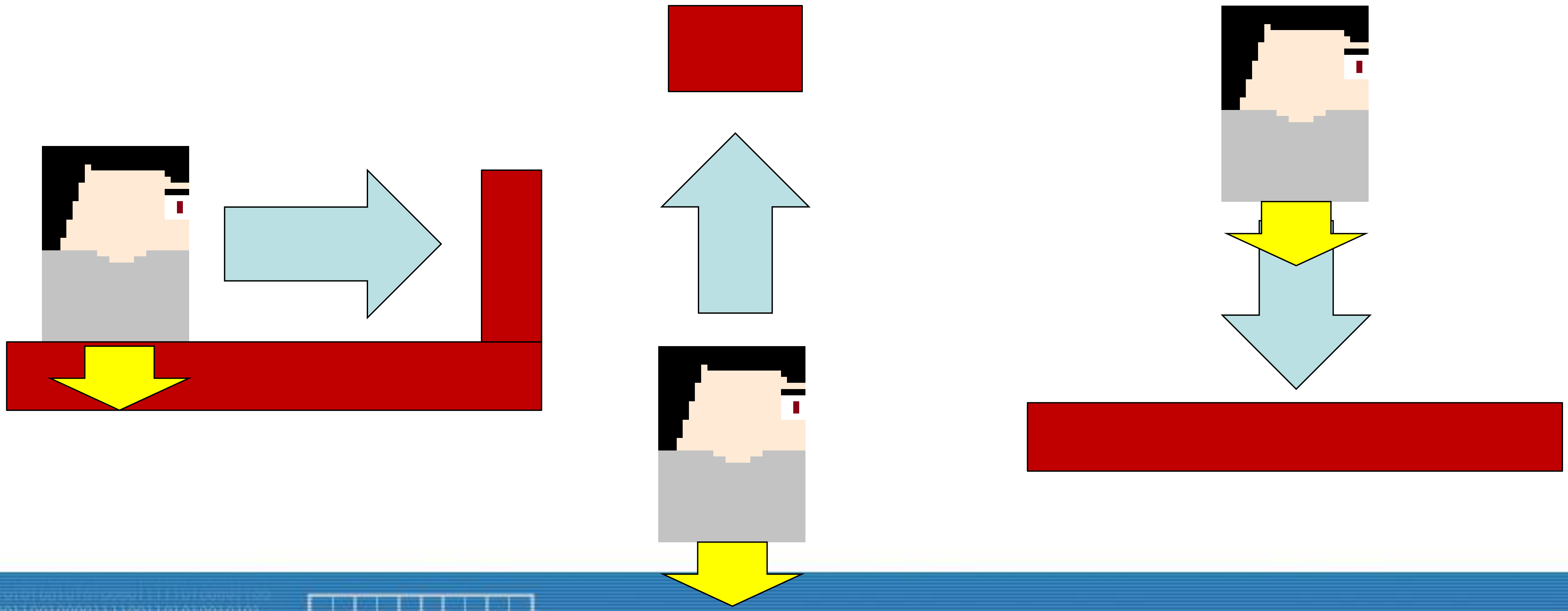
Back to Movement

- Oh right, gravity

No Game No Life, episode 1

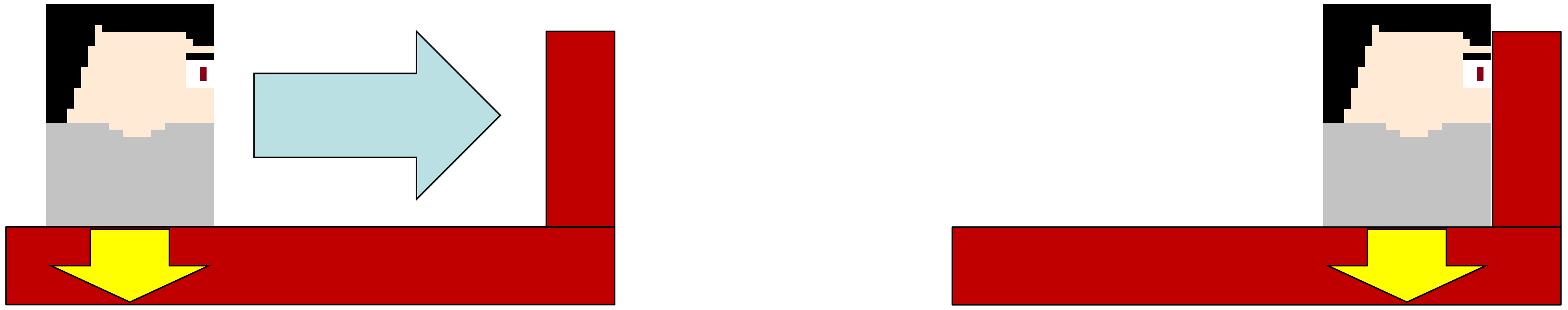
Gravity

- Gravity is an acceleration value that is constantly applied to the character regardless of player input



Gravity

- This means that, while grounded, the character will always trigger a collision response with the platform below
- The result is a possible complication when running into another wall



Collision Checking

- Remember that the collision response will depend on how the character collided with the wall
  - How do you know it was from the side?
  - How do you know it was from the top?
  - How do you know it was from the bottom?

Collision Checking

- In your air hockey homework, relative velocity was used to determine:
  - If there was a collision between two entities, and
  - The collision response (post-collision velocities)

Collision Checking

- For our platformer, if we assume that walls are immobile, the relative velocity is simply the player character’s velocity
- Collision check is also simplified to an intersection check
  - Which we can assume to be an AABB/rectangle intersection check

Collision Checking

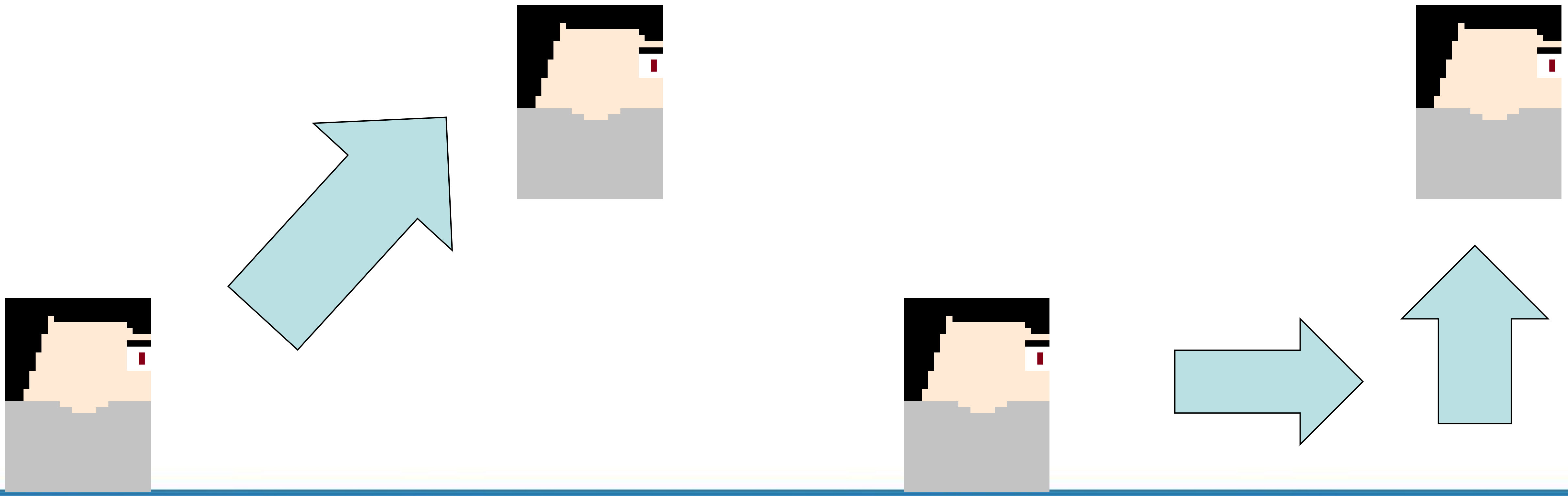
- For our platformer, if we assume that walls are immobile, the <u>relative velocity is simply the player character's velocity</u>
- Collision check is also simplified to an intersection check
  - Which we can assume to be an AABB/rectangle intersection check

Velocity

- As previously mentioned, velocity is a 2-D vector that represents change in position
  - Are there velocity values make collisions with walls impossible?
  - What if we reduce scope to collisions from either side?
  - From above?
  - From below?

Back Up a Bit

- Given that horizontal movement and jumping are two different commands, perhaps they can be handled separately?
  - Combining might not be a good idea!



(Lack of) Realism

- Real-life physics dictates that isolating horizontal and vertical movement will result in unrealistic collision detection, especially in cases where velocity has a relatively high value
- But those cases are normally addressed by restricting maximum velocity to a value lower than the object's size

(Lack of) Realism

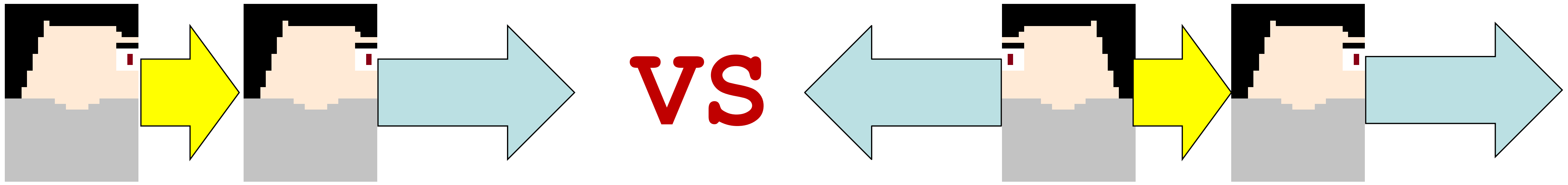
- High framerates also help address the problem, as they effectively allow us to cut velocity into a relatively small value applied per frame
- Since the velocity value is small, splitting velocity into its horizontal and vertical components should not be an issue

(Lack of) Realism

- The code for handling player physics then becomes:
  - Apply horizontal movement
  - Check for collision and assume all collisions are from the sides
  - Apply vertical movement
  - Check for collision and, depending on player velocity, assume all collisions are from the top or from the bottom

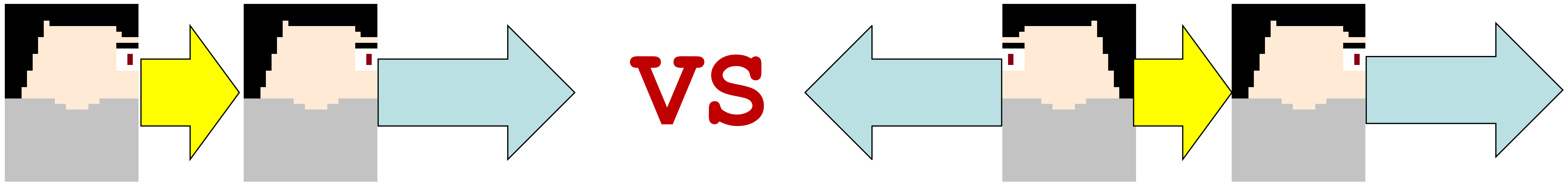
Moving Horizontally, Part II

- Assuming acceleration from input is constant, there will be a noticeable difference between these two situations:
  - From a full stop, accelerate in one direction
  - While moving in a direction, accelerate in the opposite direction



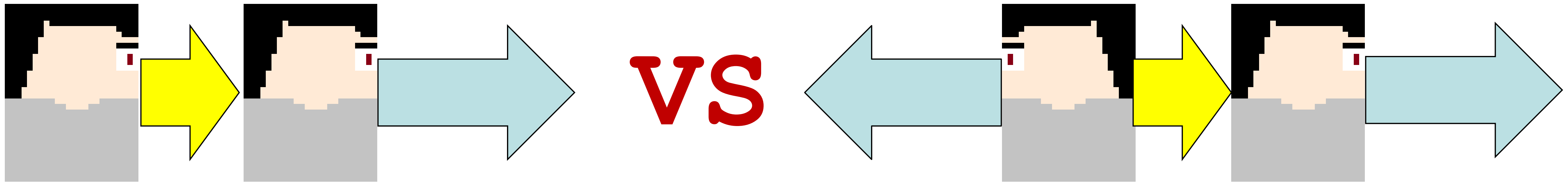
Moving Horizontally, Part II

- Consider 10m/s^2 acceleration
  - Time to get to +20m/s from 0m/s?
  - Time to get to +20m/s from -20m/s?



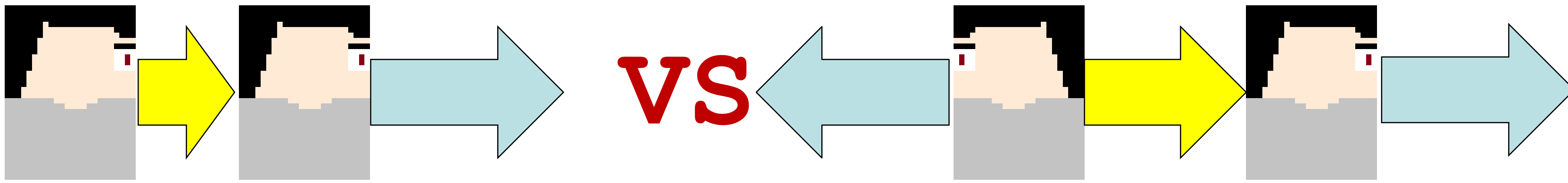
Moving Horizontally, Part II

- While the times should be different, the gap shouldn't be too large in games with little to no room for error
- The solution is to apply additional acceleration should the player want to change direction



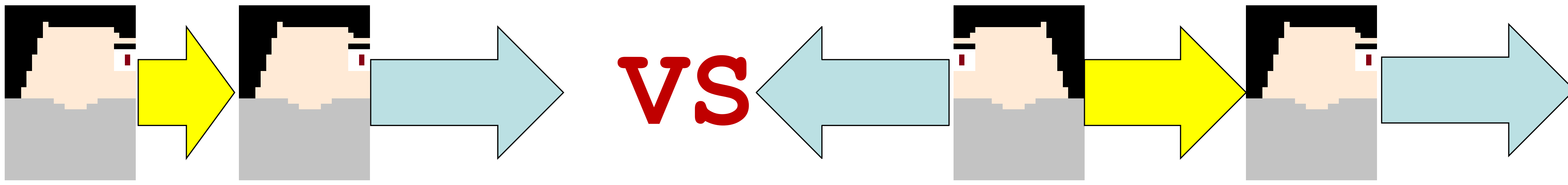
Moving Horizontally, Part II

- On "move right" input, check velocity:
  - Currently moving left?
    - Apply base acceleration times an adjustment factor greater than one
  - Currently moving right or not moving?
    - Apply base acceleration



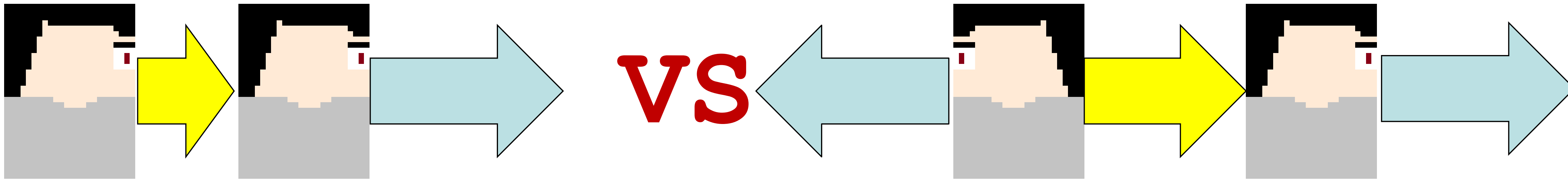
Moving Horizontally, Part II

- Note that you only apply the adjusted acceleration value for as long as the current velocity indicates that the player is moving in the opposite direction
- Normal acceleration applies once direction has been changed



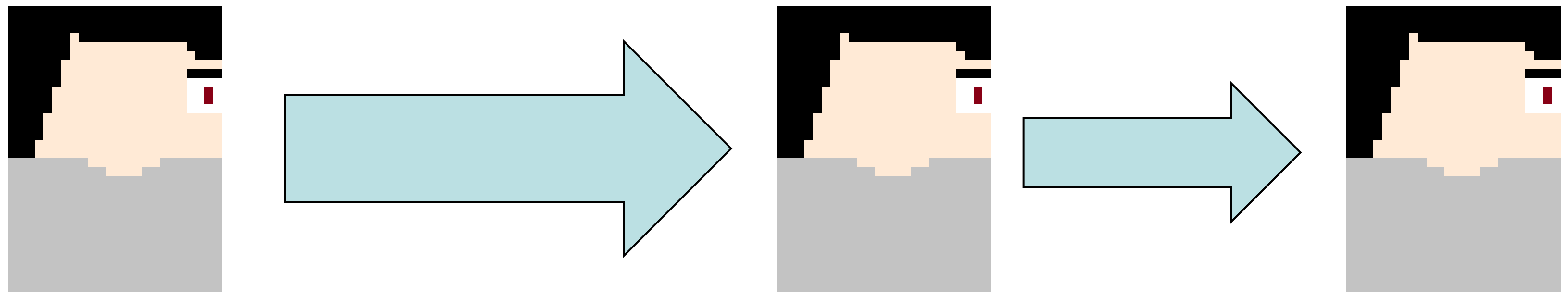
Moving Horizontally, Part II

- Don't forget to set a maximum speed
  - Player movement within a single frame should not exceed a percentage of the player character size
- Always clamp current velocity after applying acceleration



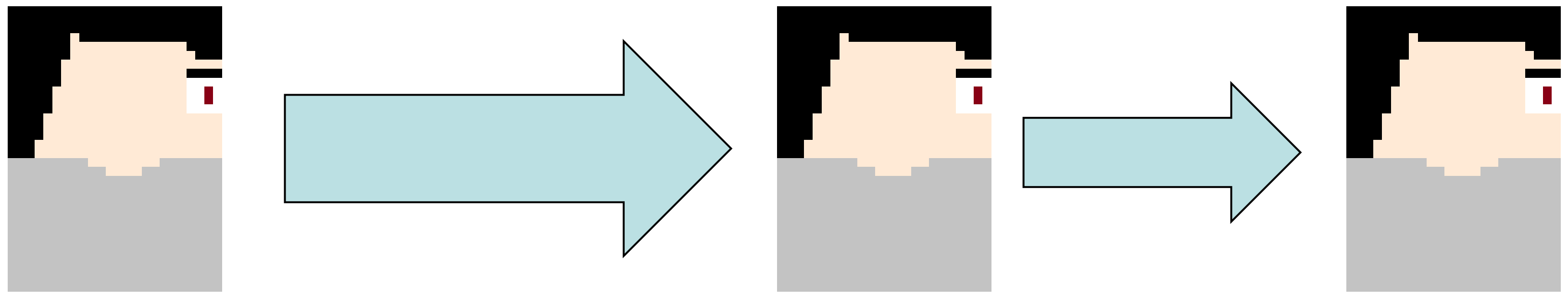
Moving Horizontally, Part II

- Deceleration due to lack of input should also result in a relatively quick stop
- Should also force a stop due to the use of floating-point values for velocity



Moving Horizontally, Part II

- On no input for horizontal movement:
  - Multiply current velocity by some factor less than one
  - If speed is less than a very small number, set current velocity to zero

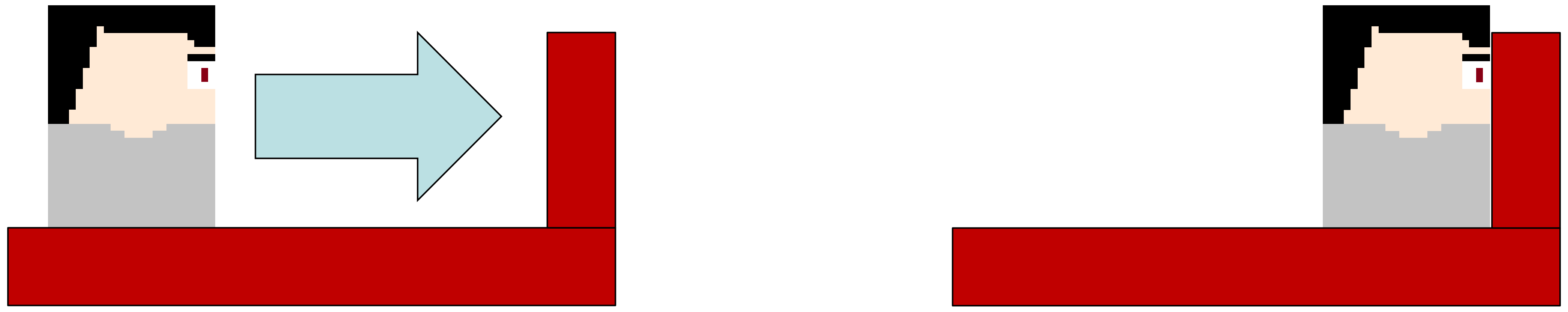


Collision from Horizontal Movement

- While pixel positions are whole numbers, actual position will be represented by floating-point numbers
- This may also cause issues with collision detection

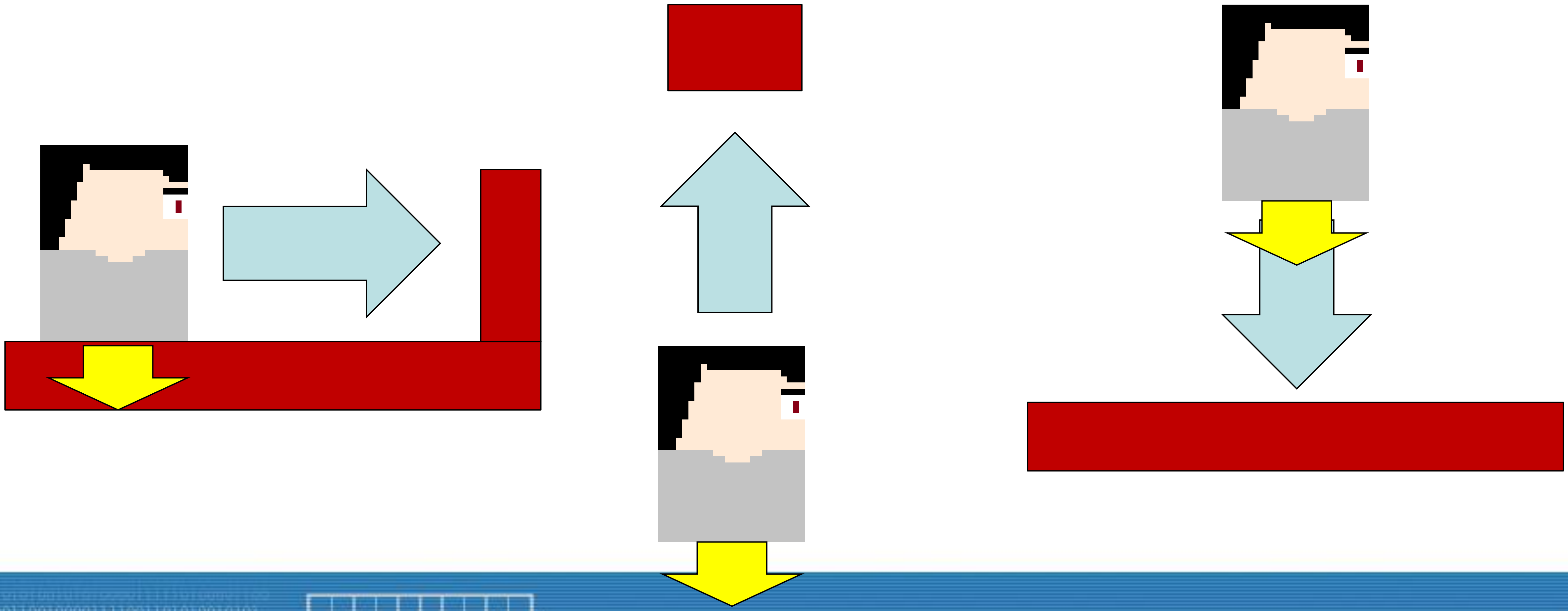
Collision from Horizontal Movement

- Add a small allowance when resetting player position due to a collision
  - Note that this will also apply to collisions from vertical movement



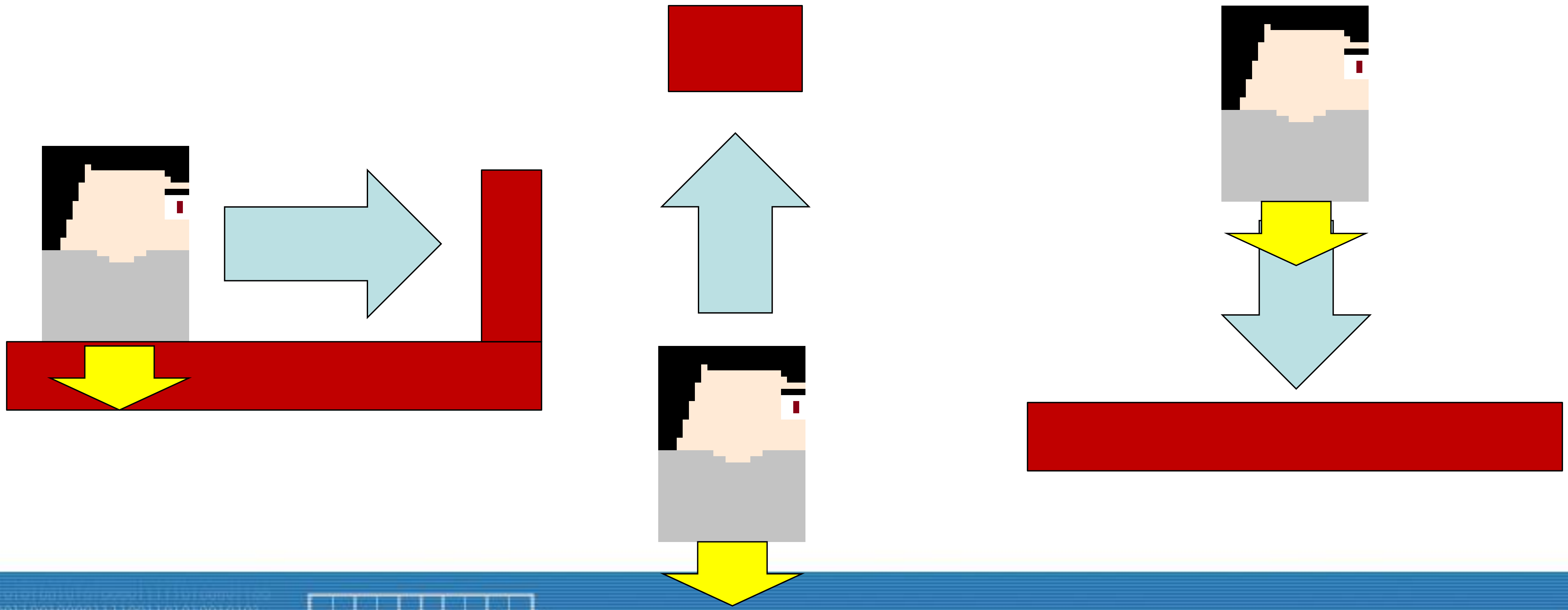
Gravity, Part II

- By default, the vertical acceleration should be your gravity constant



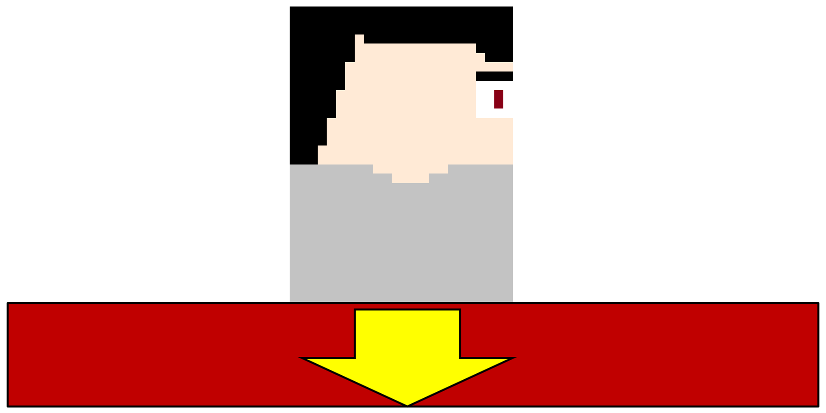
Gravity, Part II

- You also need a maximum downward velocity in case the stage requires a fall from a great height



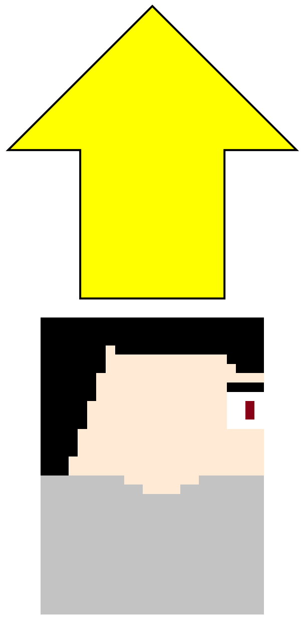
Collision from Vertical Movement (Down)

- Because of gravity, this collision occurs the most number of times
- This collision must flag the player as being grounded and able to jump



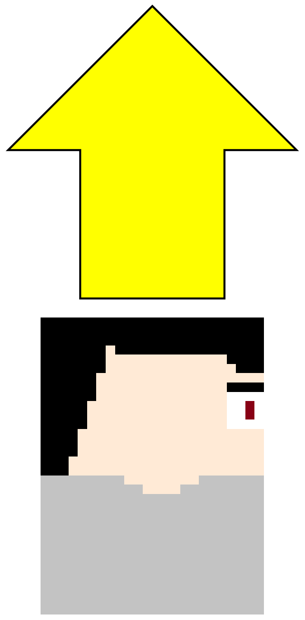
Jumping, Part II

- The player should be able to jump or initiate lift-off only if grounded
  - Jumping is an impulse or a large acceleration value applied in one frame
  - Thrusters can be simulated (or the jump height poorly controlled) using a small acceleration value applied over several frames



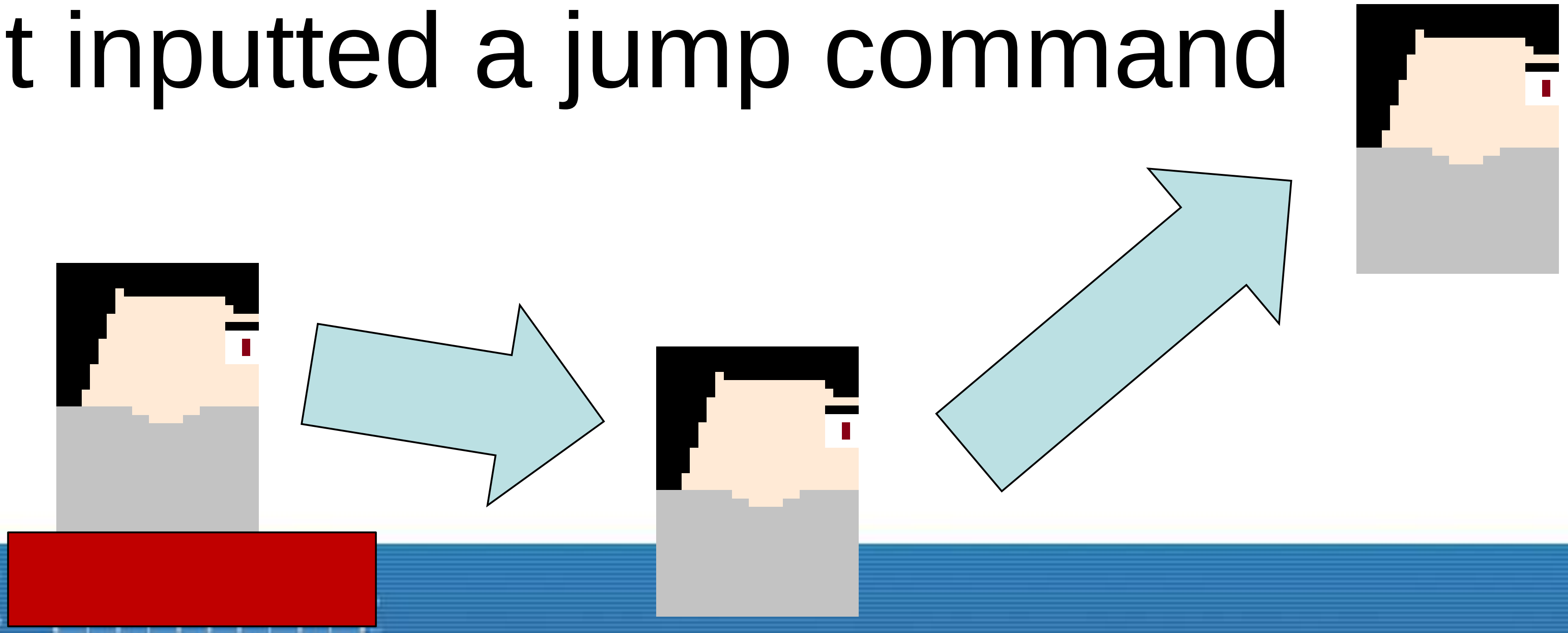
Jumping, Part II

- On jump input, apply jump acceleration if:
  - Player is grounded or
  - Jump has already begun and the number of frames that have passed since then has not yet exceeded the frame count limit
    - Player forfeits the remaining frames if s/he lets go of the "jump" input



Jumping, Part II

- Some fast-paced games allow the player to jump for a very short time (usually one-tenth of a second) after falling off a ledge
  - Player is still considered grounded for that short period of time as long as player has not inputted a jump command



Jumping, Part II

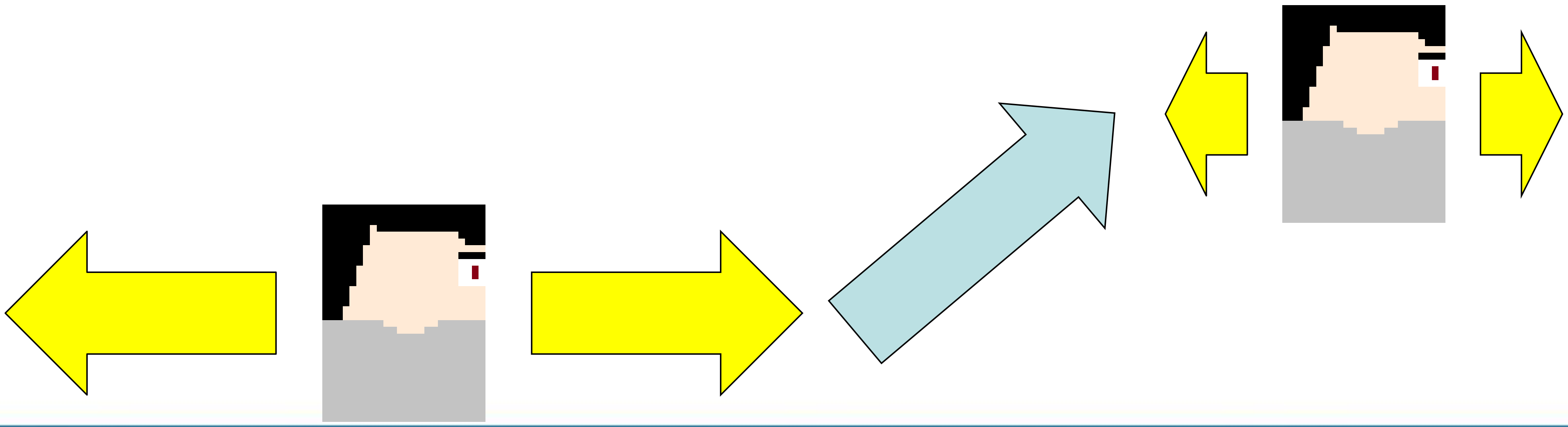
- Better jump control can be achieved by allowing the player to cut a jump short by letting go of the jump input

Jumping, Part II

- On no jump input:
  - If upward velocity exceeds a certain value (usually a negative percentage of gravity), make velocity equal to that value

Moving Horizontally, Part III

- You may also want to adjust horizontal acceleration if the player is not grounded
  - Simply multiply acceleration by an air control factor



Homework

- Create a platformer that, on initialization, will read from two text files
  - How you open the files is up to you, but the name of at least one file will have to be provided by a command-line argument (preferred) or by a string constant

Homework

- The first file is a properties file containing values that will affect the platformer
- For acceleration and velocity values, it is up to you if you want them to be per-second or per-frame
- For other values, the choice above makes no sense

Homework

```
H_ACCEL        10     // per-second
H_COEFF        0.3    // N/A
H_OPPOSITE     2.0    // N/A
H_AIR          1.0    // N/A
MIN_H_VEL      0.01   // per-frame
MAX_H_VEL      200    // per-second
GRAVITY        20     // per-second
V_ACCEL        -600   // per-second
V_HOLD         1      // N/A
V_SAFE         6      // N/A
CUT_V_VEL      -20    // per-second
MAX_V_VEL      400    // per-second
GAP            0.1    // N/A

// comments may be removed in the actual file
// per-second values are divided by FPS before use
```

DISCS

Homework

```
H_ACCEL = horizontal acceleration from input

H_COEFF = factor multiplied to current horizontal
         velocity while there is no left/right input

H_OPPOSITE = horizontal acceleration adjustment when
    moving in direction opposite to current velocity

H_AIR = horizontal acceleration adjustment in-air

MIN_H_VEL = full horizontal stop if horizontal
            velocity goes below this value

MAX_H_VEL = maximum horizontal speed

GRAVITY = vertical acceleration from gravity
```

Homework

```
V_ACCEL = vertical acceleration from input

V_HOLD = maximum number of frames jump acceleration
         is applied if jump input is held down

V_SAFE = number of frames during which player is
         considered grounded if collision with
         ground is no longer detected

CUT_V_VEL = velocity applied if jump input is
     released and current velocity exceeds this value

MAX_V_VEL = maximum vertical velocity (falling only)

GAP = extra space to add when resetting player
      position as a collision response
```

Homework

- The second file is the level data
  - First line contains a pair of numbers Px and Py representing player position
    - Width and height of player assumed to be 24 and 32 respectively
  - Second line contains a number N representing number of walls
  - Next N lines each contain numbers Wx, Wy, Wwidth, and Wheight

Homework

- Note: It is assumed that all entity positions are indicated by their centers and not by their upper-left corners
- While you can choose to set all origins to the default upper-left corner, you will need to give me a very good reason for doing so

Homework

```
400 50
11
400 580 800 40
10 300 20 600
790 300 20 600
400 100 50 20
100 500 40 20
600 400 140 20
50 400 100 20
250 300 200 20
700 300 100 20
750 200 50 20
600 100 50 20

// for best results, use an 800x600 window
```

References

- http://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php
- http://www.gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php