CS 134

More Graphics

Homework

Almost everyone did the extra credit.
If the average grade gets above 100...

Homework

- Here's a common mistake from homework 1. Can you find the bug?

```
if(kbState[KeyEvent.VK_W]) {
    if(spritePos[1] > 0) {
        spritePos[1] = spritePos[1] - 5;
    } else {
        spritePos[1] = 0;
    }
}
```

Homework

- Here's a common mistake from homework 1. Can you find the bug?

```
if(kbState[KeyEvent.VK_W]) {
    if(spritePos[1] > 0) {   // what happens if spritePos[1] == 3?
        spritePos[1] = spritePos[1] - 5;
    } else {
        spritePos[1] = 0;
    }
}
```

Homework

- Better version

```
if(kbState[KeyEvent.VK_W]) {
    spritePos[1] = spritePos[1] - 5;

    if(spritePos[1] < 0) {
        spritePos[1] = 0;
    }
}
```

Homework

Questions on homework 2?

Level Representation

- Data / Defs
  - Split up actor information into changing (Data), unchanging shared (Defs). Share Defs among all actors.
  - Level data is Defs and per-actor data.
- Prototype based
  - Combine Data and Defs. Let both change.
  - Level data is Prototype and per-actor data.

Data / Defs

```
class AnimationDef {
    String name;
    FrameDef[] frames;
}

class FrameDef {
    int image;
    float frameTimeSecs;
}

class AnimationData {
    AnimationDef def;
    int curFrame;
    float secsUntilNextFrame;

    void update(float deltaTime);
    void draw(int x, int y);
}
```

Data / Defs

```
class AnimationDef {
    String name;
    FrameDef[] frames;
}

class FrameDef {
    int image;
    float frameTimeSecs;
}

class AnimationData {
    AnimationDef def;
    int curFrame;
    float secsUntilNextFrame;

    void update(float deltaTime);
    void draw(int x, int y);
}
```

```
struct CharacterDef {
    String name;
    String walkAnimDef;
    String attackAnimDef;
}

class CharacterData {
    float x;
    float y;
    float health;
    bool isWalking;
    AnimationData curAnimation;

    void update(float deltaTime);
    void draw();
}
```

Data / Defs

```
class AnimationDef {
    String name;
    FrameDef[] frames;
}

class FrameDef {
    int image;
    float frameTimeSecs;
}

class AnimationData {
    AnimationDef def;
    int curFrame;
    float secsUntilNextFrame;

    void update(float deltaTime);
    void draw(int x, int y);
}
```

```
class LevelCharacterDef {
    String actor;
    float intialX;
    float initialY;
    float initialHealth;
}
```

```
struct CharacterDef {
    String name;
    String walkAnimDef;
    String attackAnimDef;
}

class CharacterData {
    float x;
    float y;
    float health;
    bool isWalking;
    AnimationData curAnimation;

    void update(float deltaTime);
    void draw();
}
```

Data / Defs

- Advantage:
  - All Defs only exist once
  - Easy to understand and reason about what data changes and doesn't change
  - Avoids having ”bad to customize” fields be customizable
- Disadvantage:
  - What can be customized is controlled by code

Prototype Based

- Rarely is EVERY system prototype based, it doesn't make much sense
  - What would a prototype for Animation be?
- Choose key classes and make them prototype based

Prototype Based

```
class AnimationDef {
    String name;
    FrameDef[] frames;
}

class FrameDef {
    int image;
    float frameTimeSecs;
}

class AnimationData {
    AnimationDef def;
    int curFrame;
    float secsUntilNextFrame;
}
```

```
struct CharacterDef {
    String name;
    String walkAnimDef;
    String attackAnimDef;
}

class CharacterData {
    float x;
    float y;
    float health;
    bool isWalking;
    AnimationData curAnimation;
}
```

// No need for this, just store
// CharacterData in your level directly!

~~class LevelCharacterDef {~~
~~   String actor;~~
~~   float intialX;~~
~~   float initialY;~~
~~   float initialHealth;~~
~~}~~

Prototype Based

- Advantage:
  - Full flexibility, every single field can be customized
  - Less classes to deal with
- Disadvantage:
  - Full flexibility, every single field can be customized, including ones that could get out of sync
  - Less data sharing can go on

Prototype Based

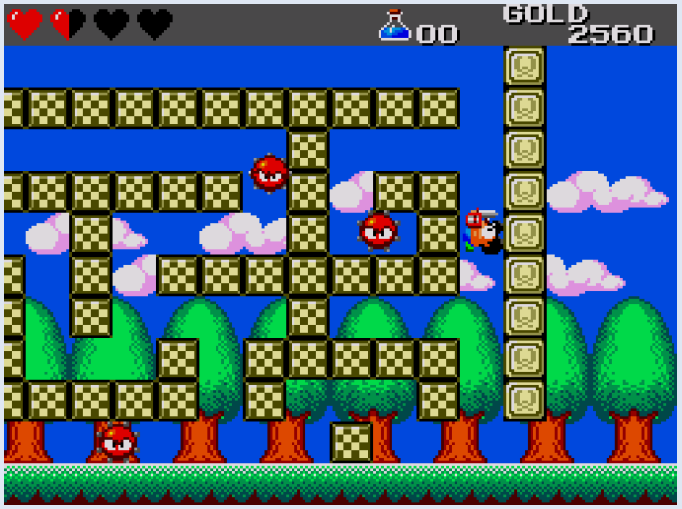
Questions?

Summary

- Backgrounds are easy
  - Simple for loop!
  - Can have multiple backgrounds to have stuff in front of and behind sprites.
- Sprites are a bit harder
  - They have state!
  - But ultimately, you have a list of sprites and you call update() and draw() on each of them.

The Game Loop So Far

```
while (!shouldExit) {
    System.arraycopy(kbState, 0, kbPrevState, 0, kbState.length);

    // Actually, this runs the entire OS message pump.
    window.display();
    if (!window.isVisible()) {
        shouldExit = true;
        break;
    }

    // Check keyboard input for player
    // Update positions and animations of all sprites

    gl.glClearColor(0, 0, 0, 1);
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT);

    // Draw background(s)
    // Draw sprites
    // Draw more background(s)

    // Present to the player.
    window.swapBuffers();
}
```



Scrolling

- All sprites AND the camera should have their position stored in world space.
- All drawing commands take pixel space values.

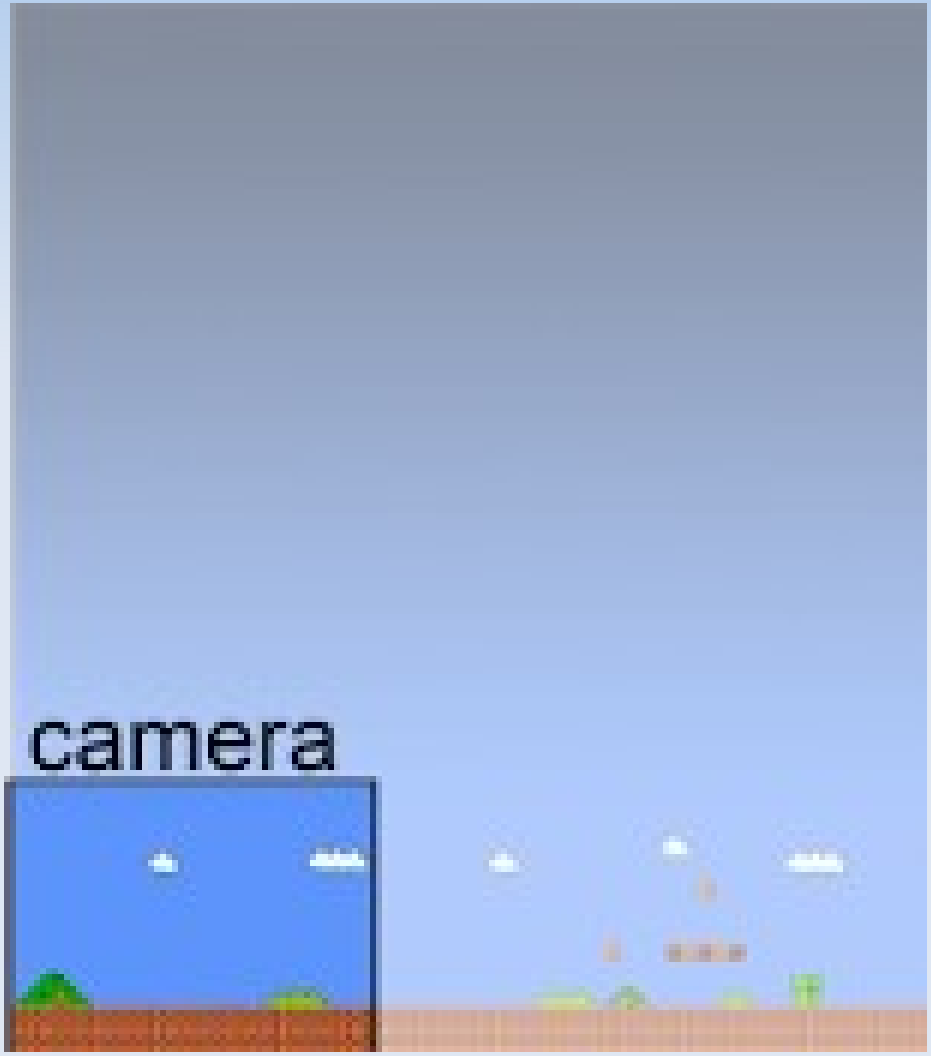
- When scrolling, you UPDATE the camera's world space position so that you CALCULATE the sprites' pixel space position.

Scrolling

Really, the camera is in the world!

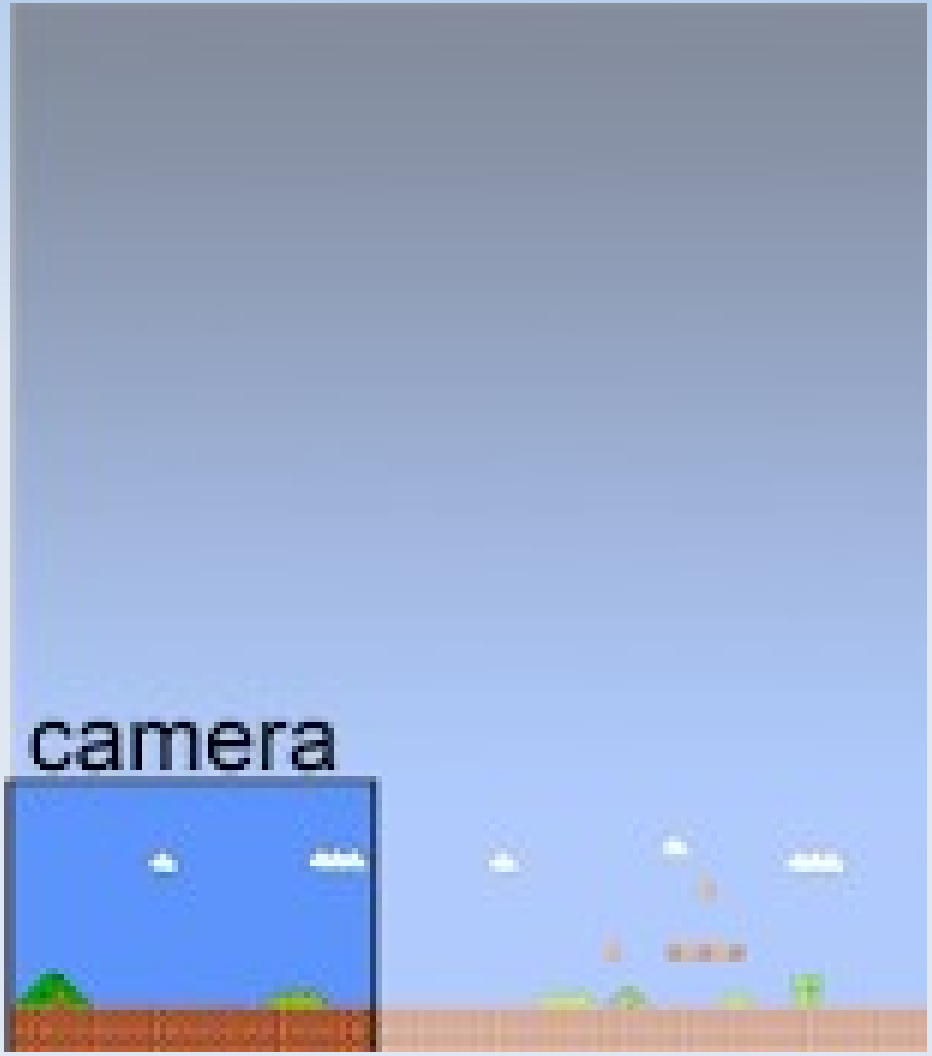
Scrolling

- Think of the camera as being positioned in the world as well
  - class Camera {
    public int x;
    public int y;
    }



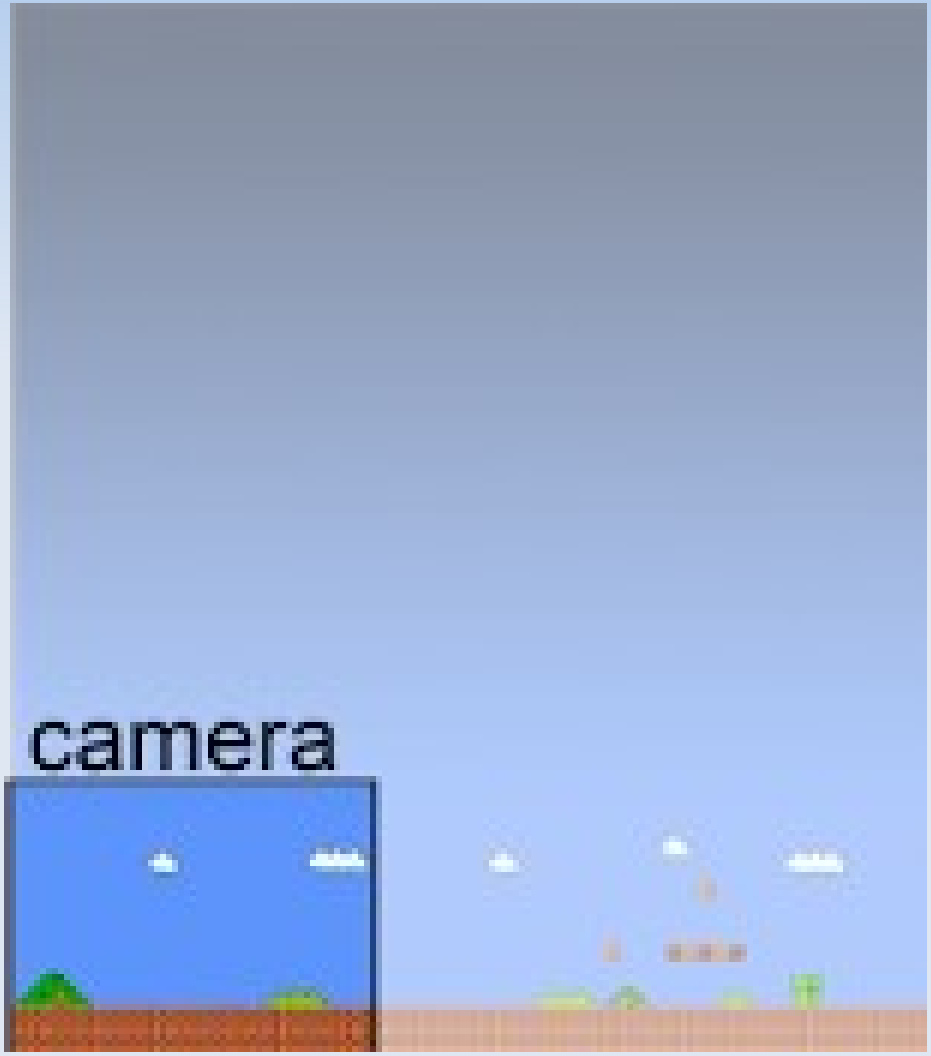
Scrolling

- Given a camera c, and a sprite s, where do you draw the sprite?
  - c.x, c.y
  - s.x, s.y



Scrolling

- Given a camera c, and a sprite s, where do you draw the sprite?
  - c.x, c.y
  - s.x, s.y
- s.x - c.x
- s.y - c.y



Scrolling

- Other features to think about with cameras:

- Prevent the camera from leaving the world
- Calculate the camera's position based on the player's position

- Which of the two must win?

Timing & Scrolling

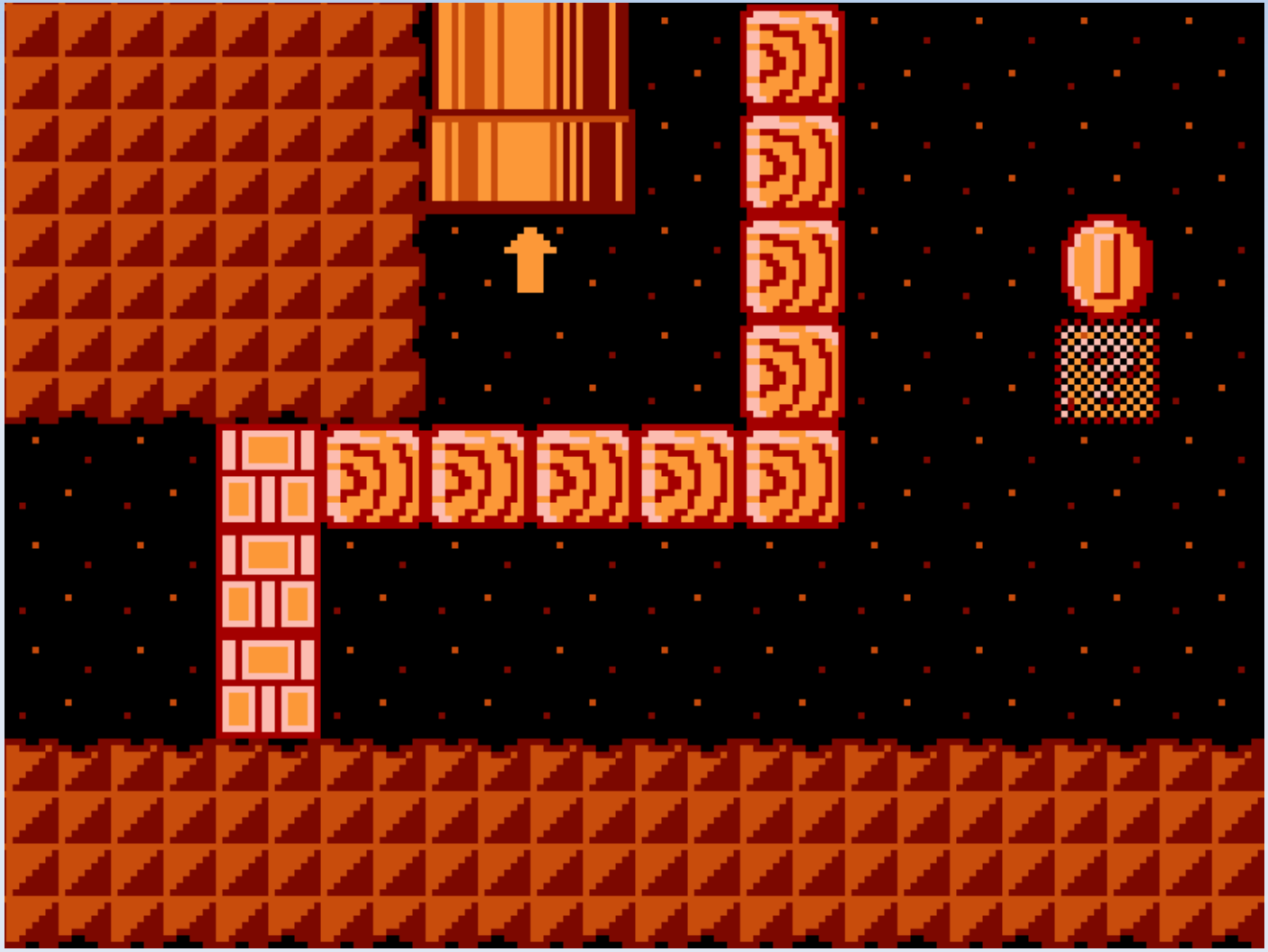
Questions?

Backgrounds++

- Side/Top down
- Isometric
- Hex grid

- 3/4ths View

Side/Top Down



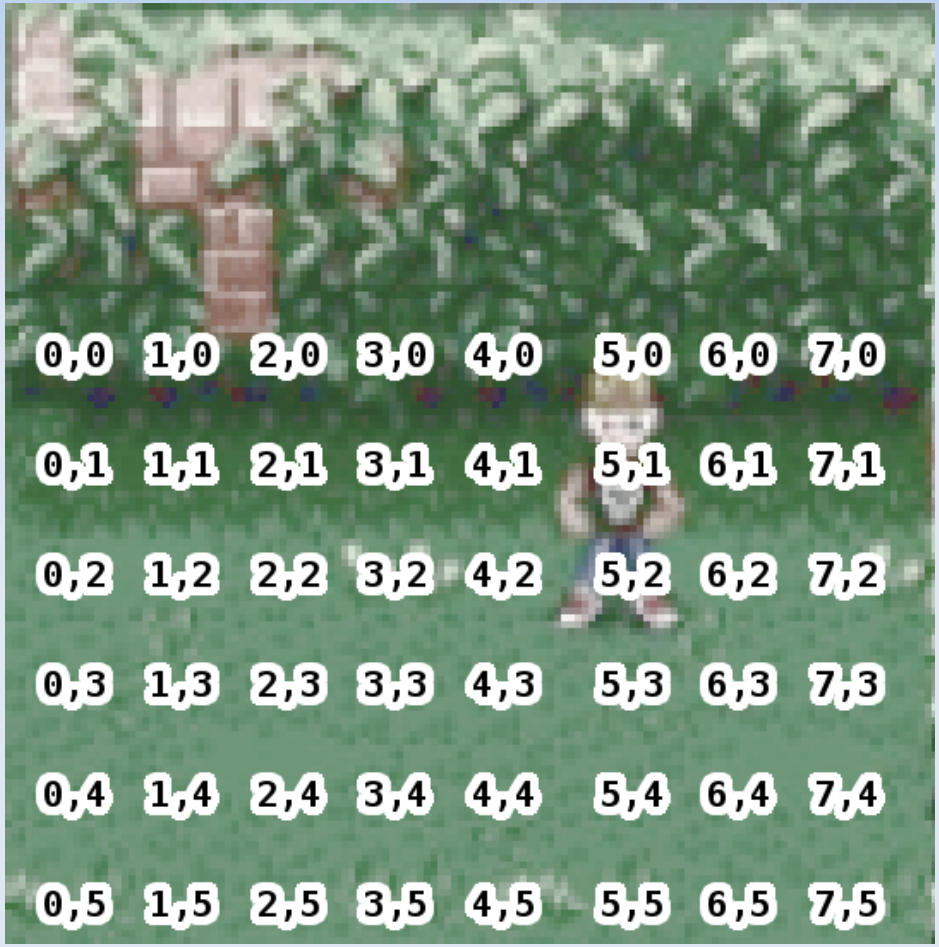
Side/Top Down

- Level is 2D array
- Tile position:
  - x*w
  - y*h

3/4ths View

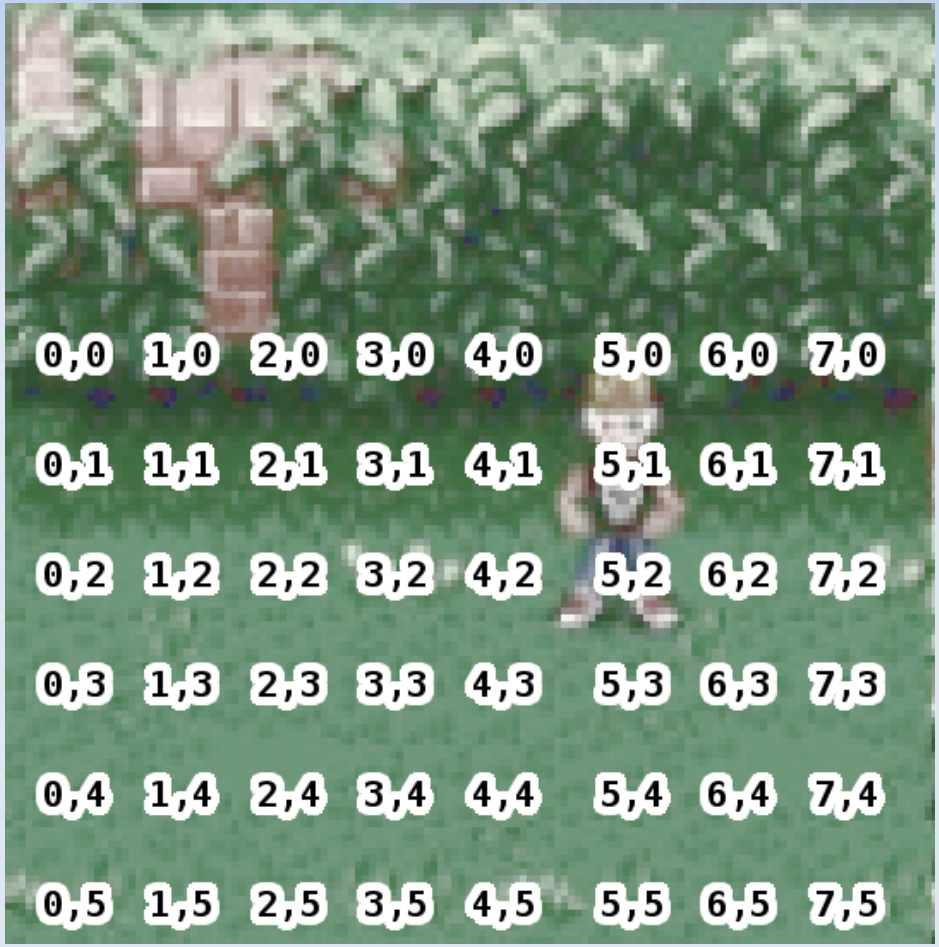
3/4ths View

- No change from normal rendering
- Draw top to bottom, to follow the Painter's Algorithm
- If you have height, you must make sure top to bottom follows floor position.



3/4ths View

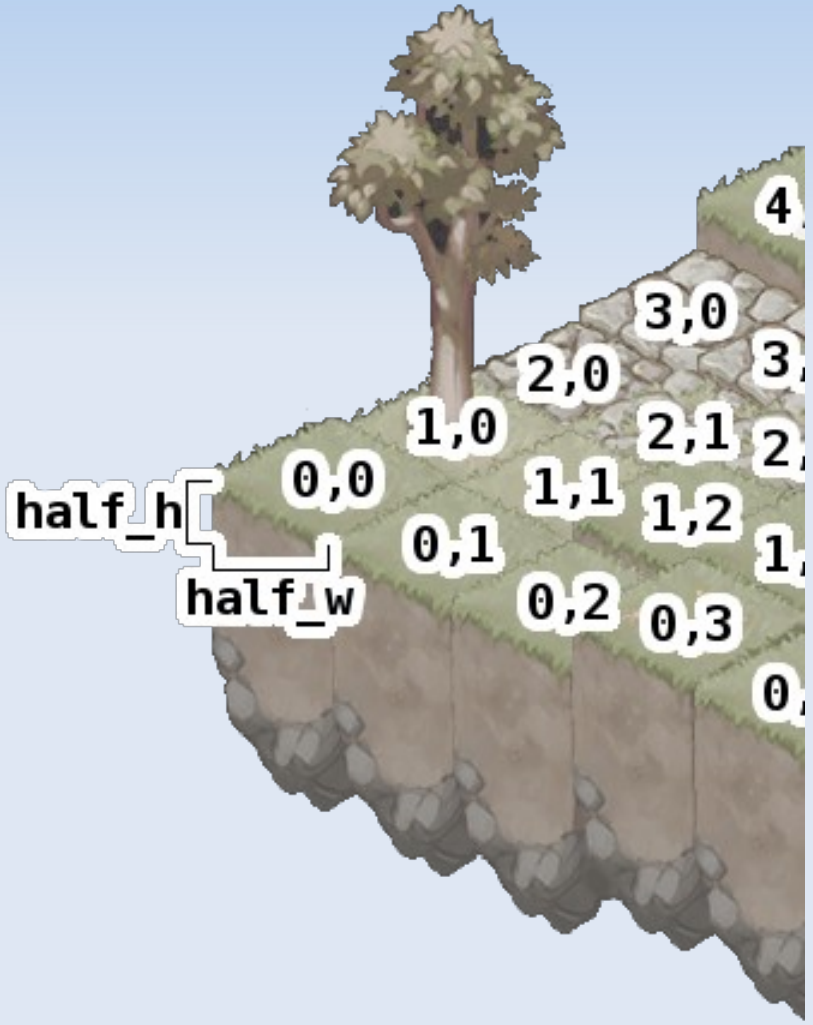
- Known info:
  - tileW
  - tileH
  - imageW
  - imageH
- Tile position:
  - x * tileW
  - y * tileH



Isometric

Isometric

- Level is still 2D array of indexes
- Tile position:
  - (x+y) * half_w
  - (-1-x+y) * half_h



Hex Grid

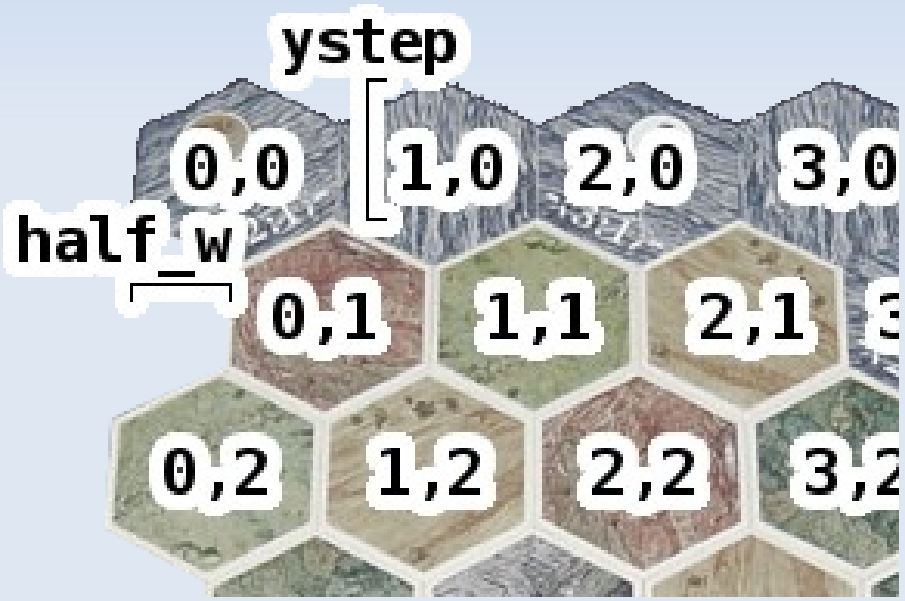
Hex Grid

- Still 2D array of indexes

- Tile position:
  - x*w

    (if y is even)
  - (x+0.5)*w

    (if y is odd)
  - y*ystep



Backgrounds++

Defs are still always a 2D array

Questions?

Parallax Scrolling

- Let's look at a video

Parallax Scrolling

- Simple parallax
- Farther away things appear smaller, move slower
- Can have multiple layers

Parallax Scrolling

- Consider having multiple layers
- For example
  - BG1: offset by ???
  - BG2: offset by ???
- You can also have
  - FG2: offset by ???

Parallax Scrolling

- Consider having multiple layers
- For example
  - BG1: offset by camX/2, camY/2
  - BG2: offset by camX/4, camY/4
- You can also have
  - FG2: offset by camX*1.5, camY*1.5

Parallax Scrolling

Questions?