THE COMPOSTIAN

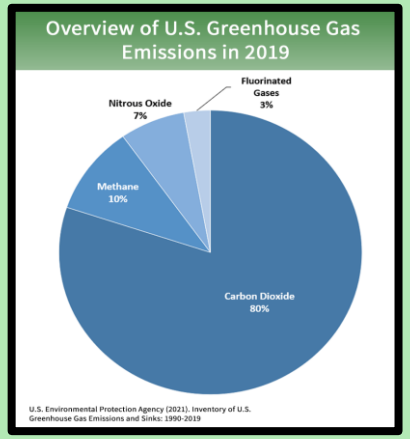
MAKE OUR TOWN LOVELY AGAIN

PROJECT PRESENTATION

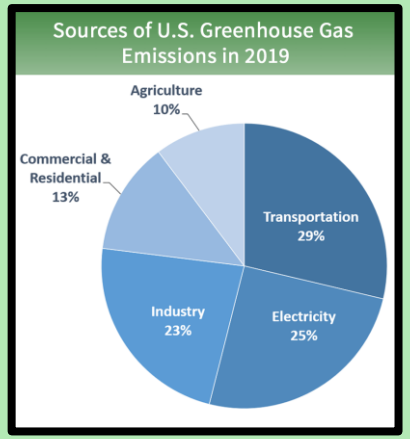
Eray Dindas - M. Giray Nacakcı - Umut Can Günay - Ayberk Aygün

THE IDEA

- According to the 2019-dated report of the United States Environmental Protection Agency (EPA), U.S. greenhouse gas emissions totaled 6,558 million metric tons of carbondioxide equivalents.



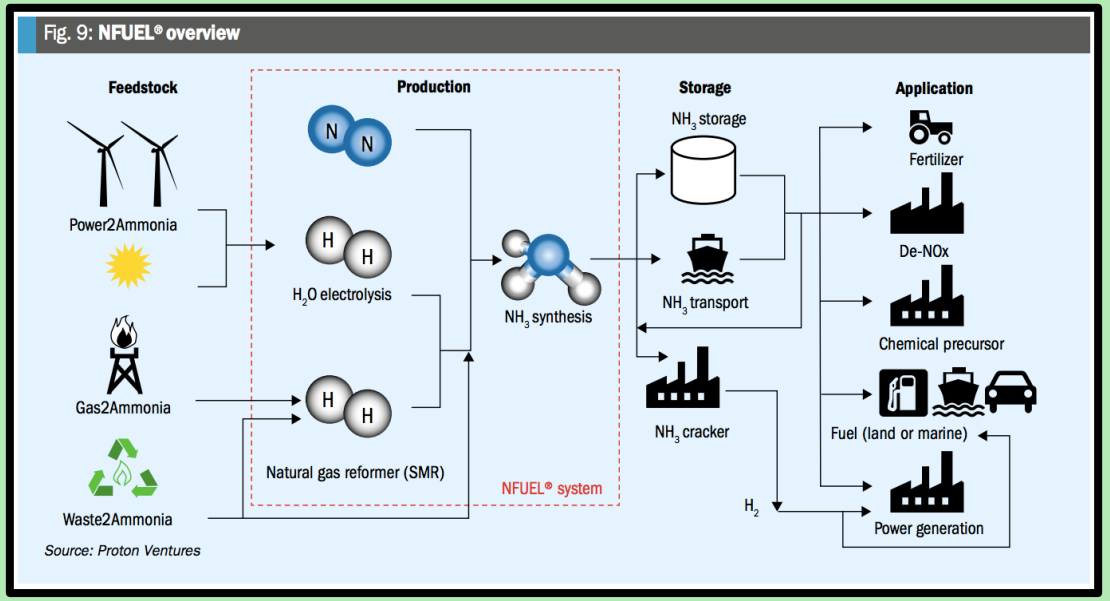
- The Inventory of U.S. Greenhouse Gas Emissions and sinks has reported that in 2019, approximately 114.5 million metric tons of carbondioxide equivalent methane was emissioned from landfills.



Source: https://www.epa.gov/lmop/frequent-questions-about-landfill-gas#landfillgas

THE IDEA

- **Carbondioxide equivalent landfill gas and the collected organic waste can be repurposed for electricity and fertilizer production.**
- **We've thought of a machine that will produce electricity from the encapsulated methane gas to power the fertilizer production from organic wastes.**



Source: https://www.ammoniaenergy.org/articles/small-scale-ammonia-where-the-economics-work-and-the-technology-is-ready/

THE GAMIFICATION

- **We couldn’t stress enough our will of introducing reproduction of electricity and fertilizer from landfills. To make our concept more accessible, we’ve decided to present this idea in a video game format.**

- **Some fantasy elements are included for gamification & challenge purposes, the garbage piles that’re going to be collected are monsters who chase and try to stop a garbage collecting person (controlled by the player).**

THE GAMIFICATION

- **The player will vaccuum the garbage monsters and drop them into the machine that will produce fertilizer.**

- **After collecting the fertilizer, the player will throw it to the tree saplings around the map to make them grow. (trees don't grow instantly by simply throwing fertilizer, but again, we had to introduce some fantasy elements)**

- **We've also thought of a background story to make the context behind fantasy elements more acceptable.**

THE STORY

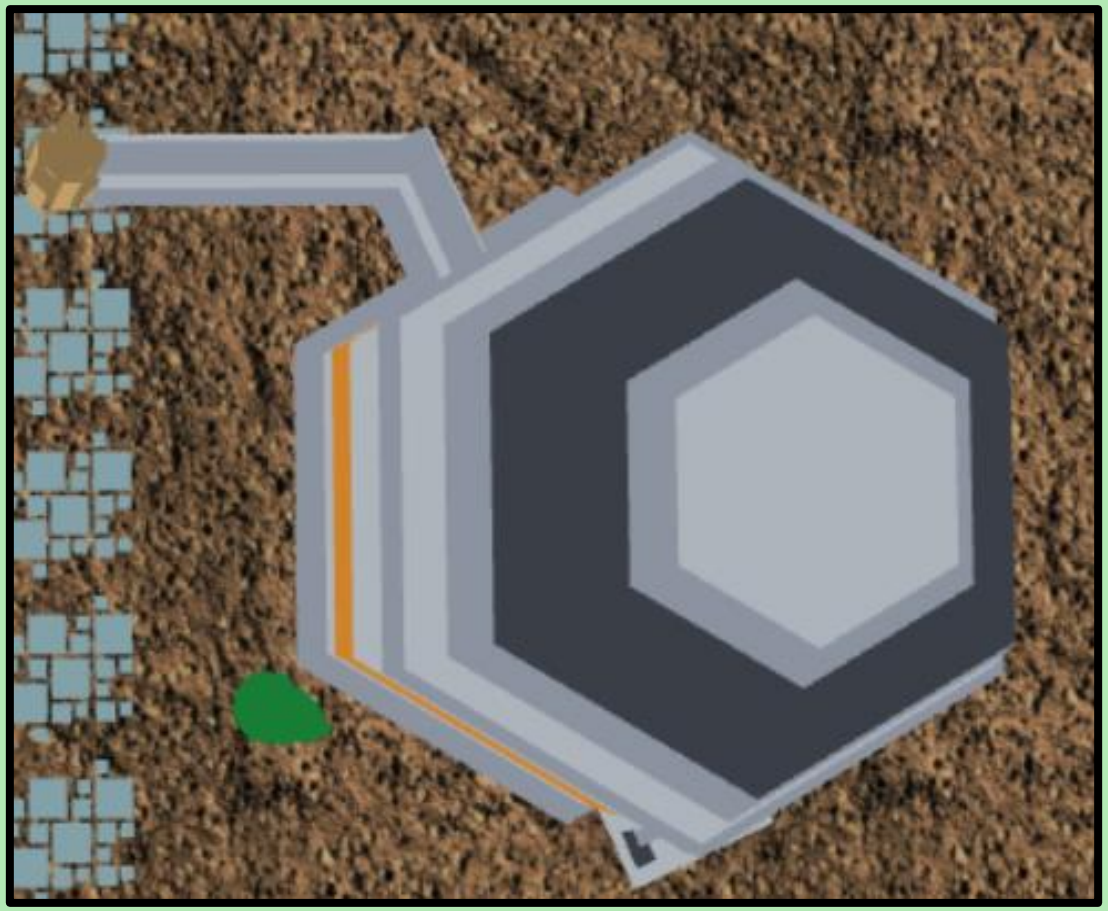
- In a near future, a group of scientists have finally completed a secret project that has been ongoing for years: a secret formula that would make the organic garbage piles into safe and healty food composts when poured.

- But the things didn't go as expected and the garbage piles have turned into living, violent organisms that attack everyone instead of food.

- As the threat goes on, a genius boy has developed a machine to produce electricity and insta-fertilizer from the garbage monsters. Calling himself a **Compostian** (compost + guardian), with his vaccuum gun, he has decided to fight with the monsters to save the planet.

THE STORY

Our hero, The Compostian



The machine designed to compost the monsters

TECHNICAL DETAILS & ALGORITHMS

THE PATHFINDING ALGORITHM

Modified Depth-First-Search with a Target Direction

Visual Draft Image to Demonstrate Tile Graph

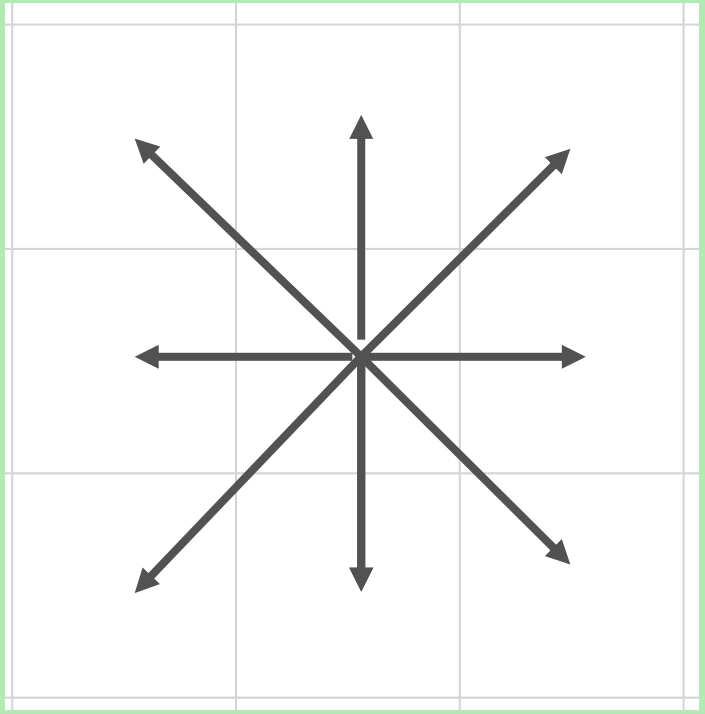
Garbage monsters move towards the player on 2D tile graph.

THE PATHFINDING ALGORITHM

Modified Depth-First-Search with a Target Direction

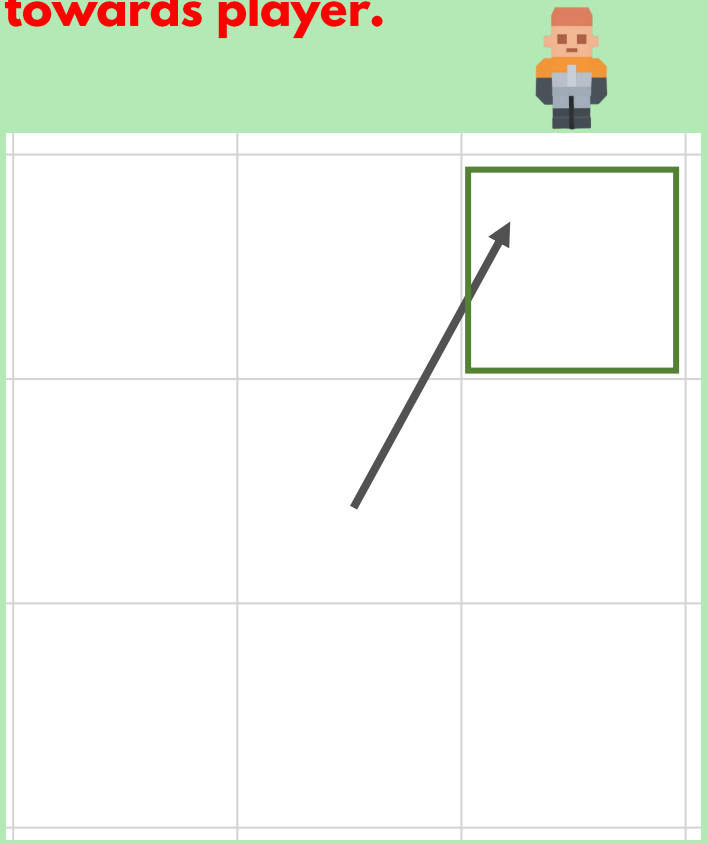
Classic DFS

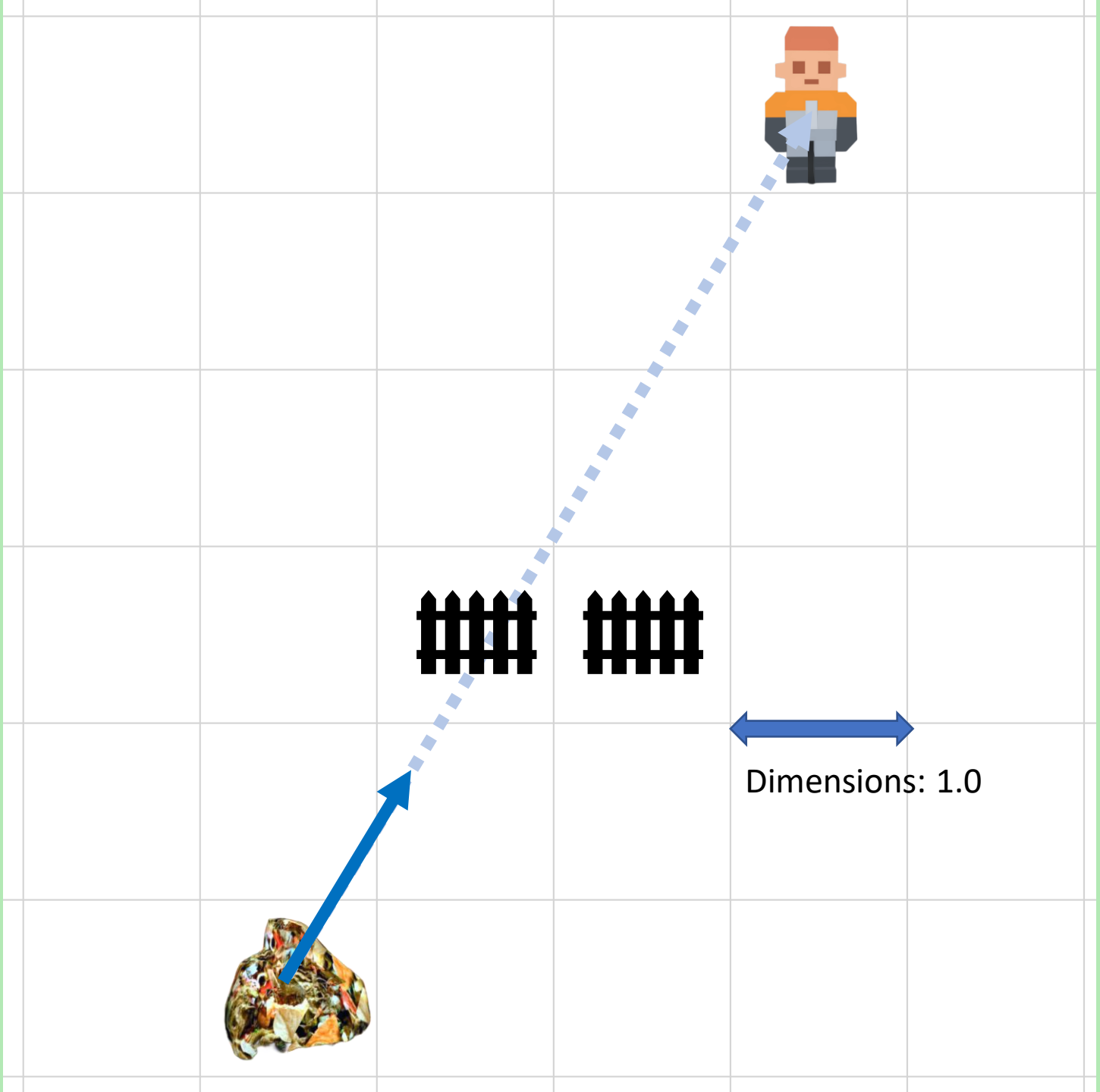
Choose any of the neighbor tiles.



Our Algorithm

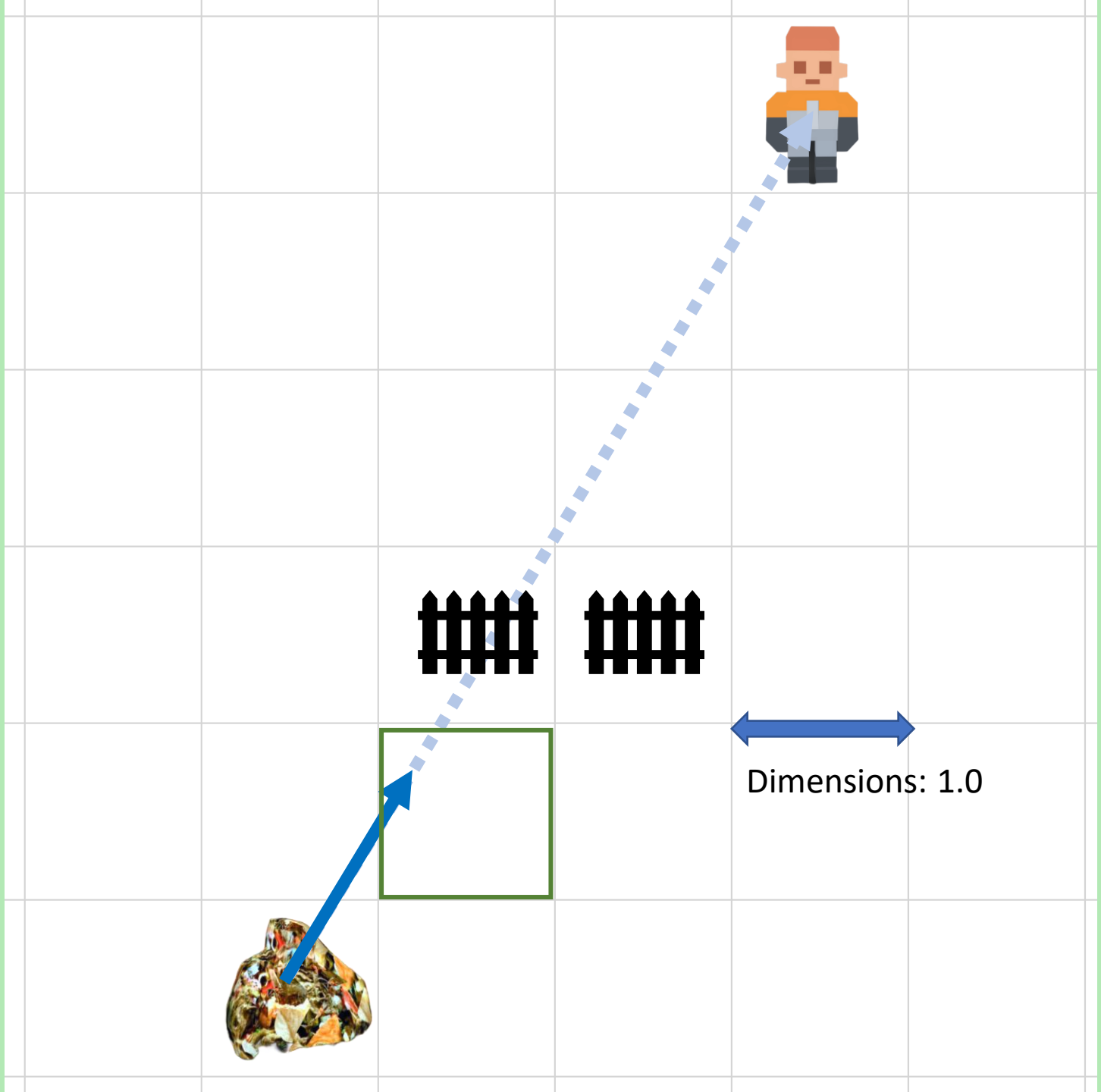
Choose the neighbor pointed by vector towards player.





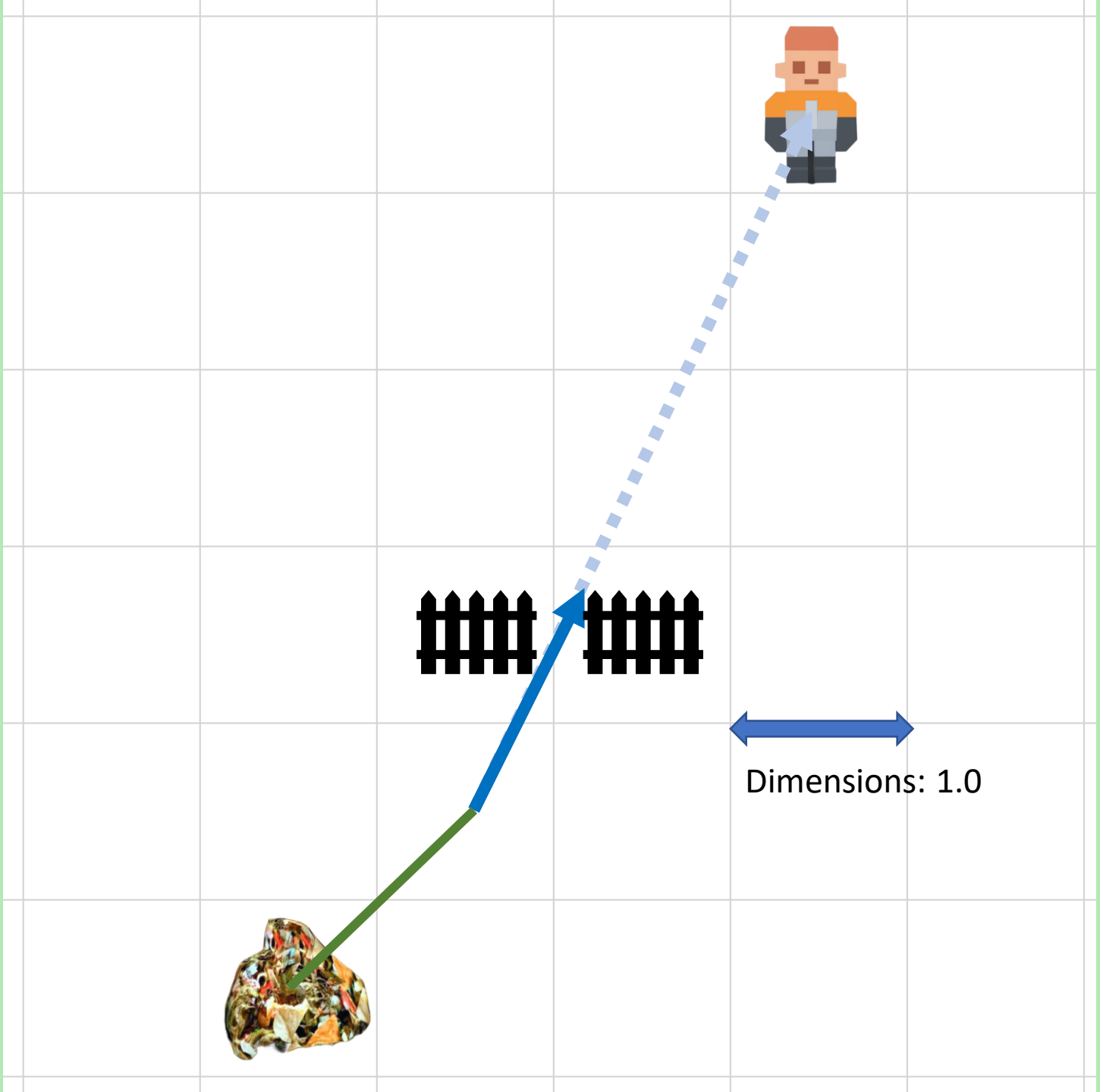
Next tile ?

Normalized vector from current_tile to player



Next tile ?

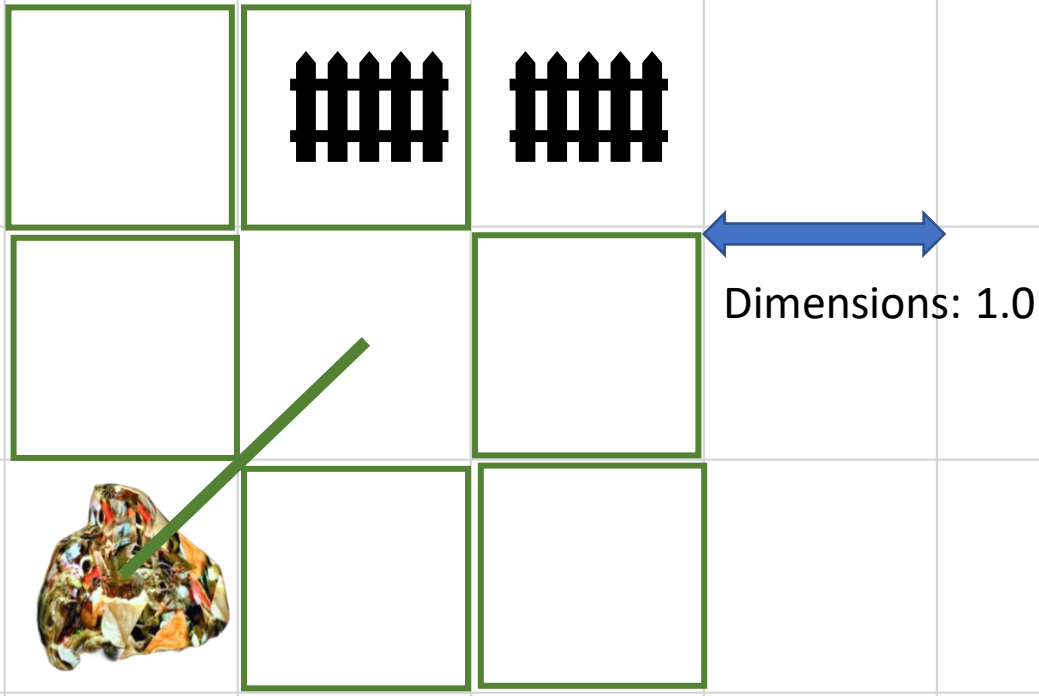
Normalized vector from current_tile to player



Next tile ?

Normalized vector from current_tile to player

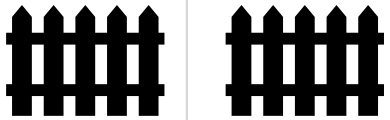
Occupied!

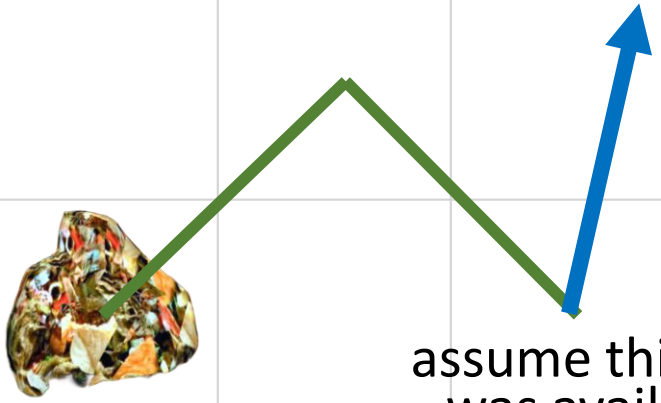


Occupied or **Visited**

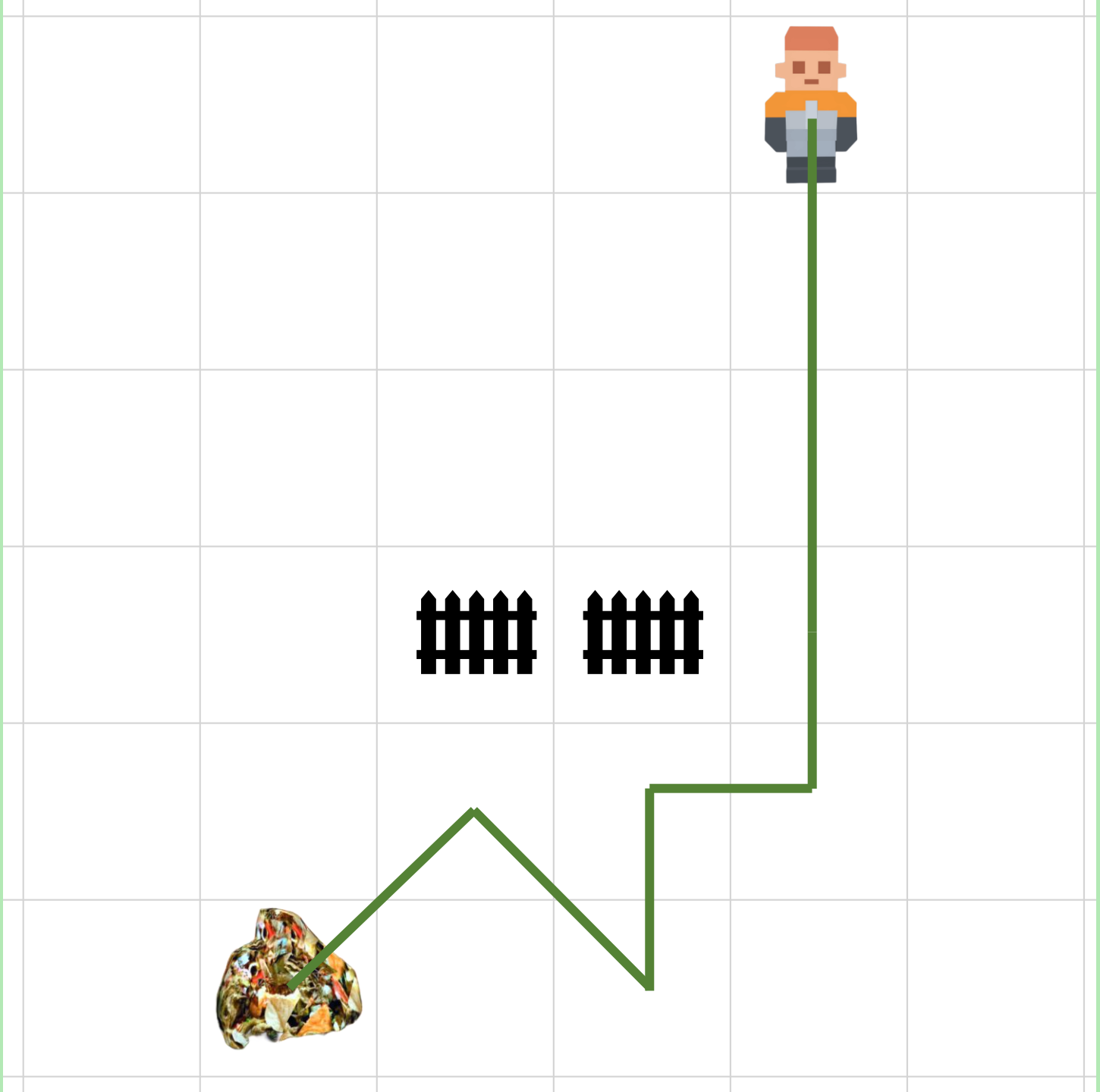
Try other neighbors

(for loop)
(just like classic DFS)



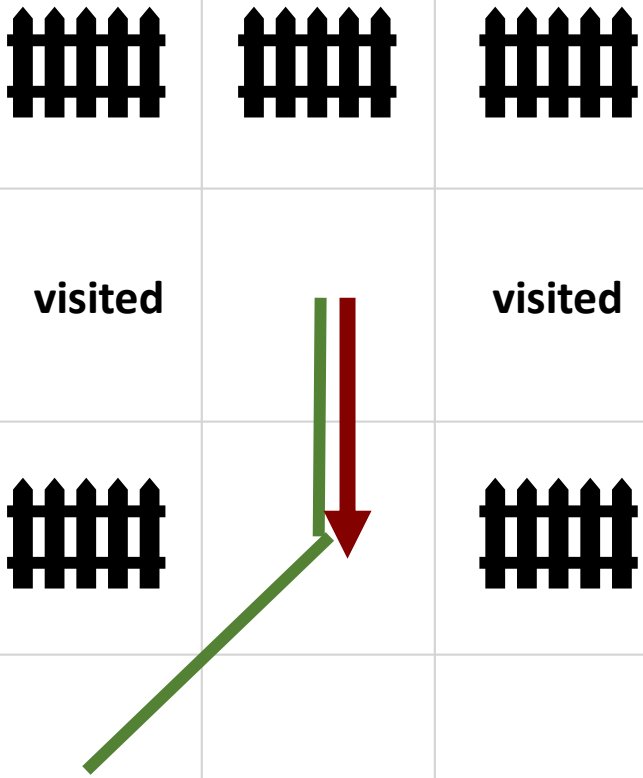


assume this one
was available



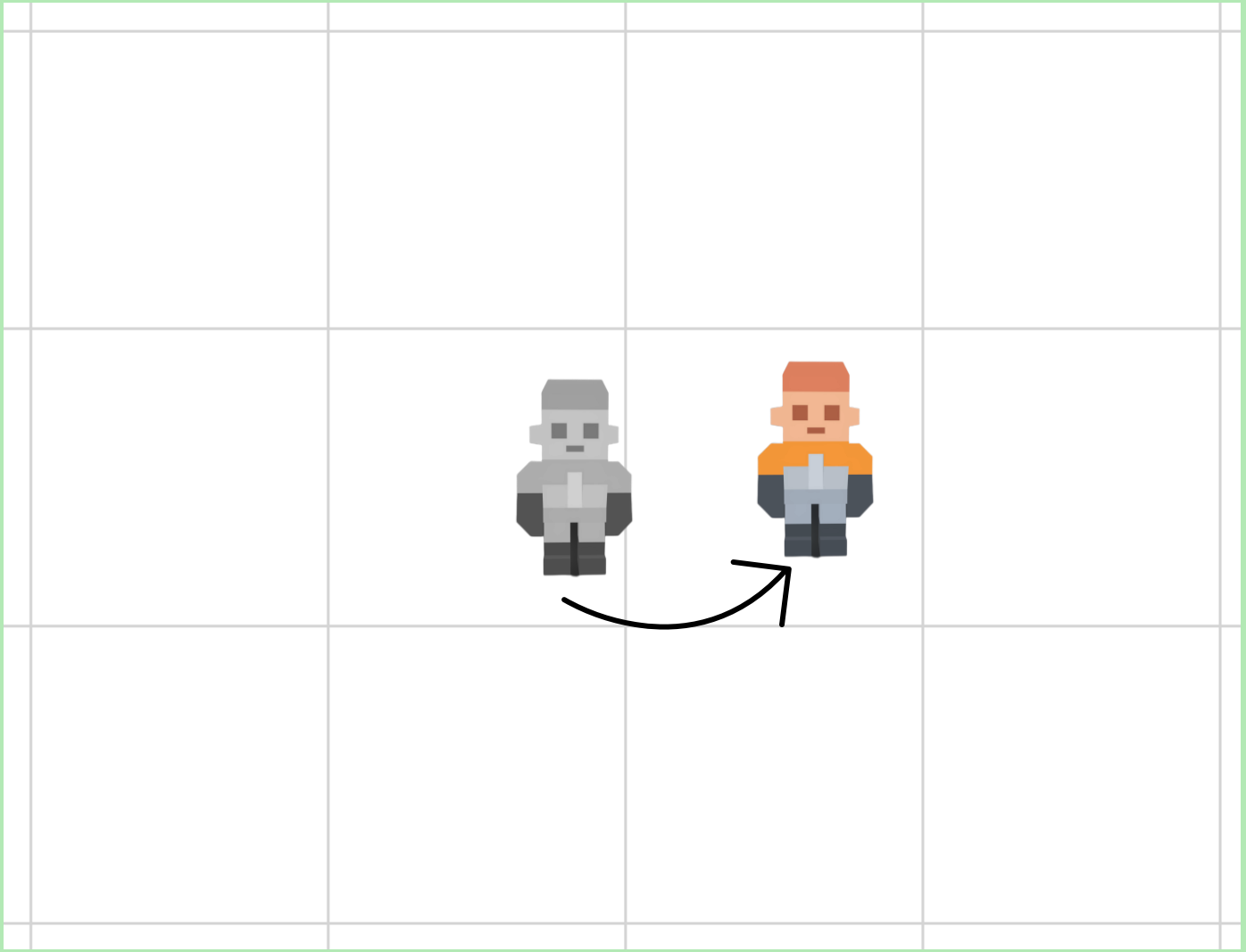
Path is found

Monster starts moving



No available neighbor

Step back

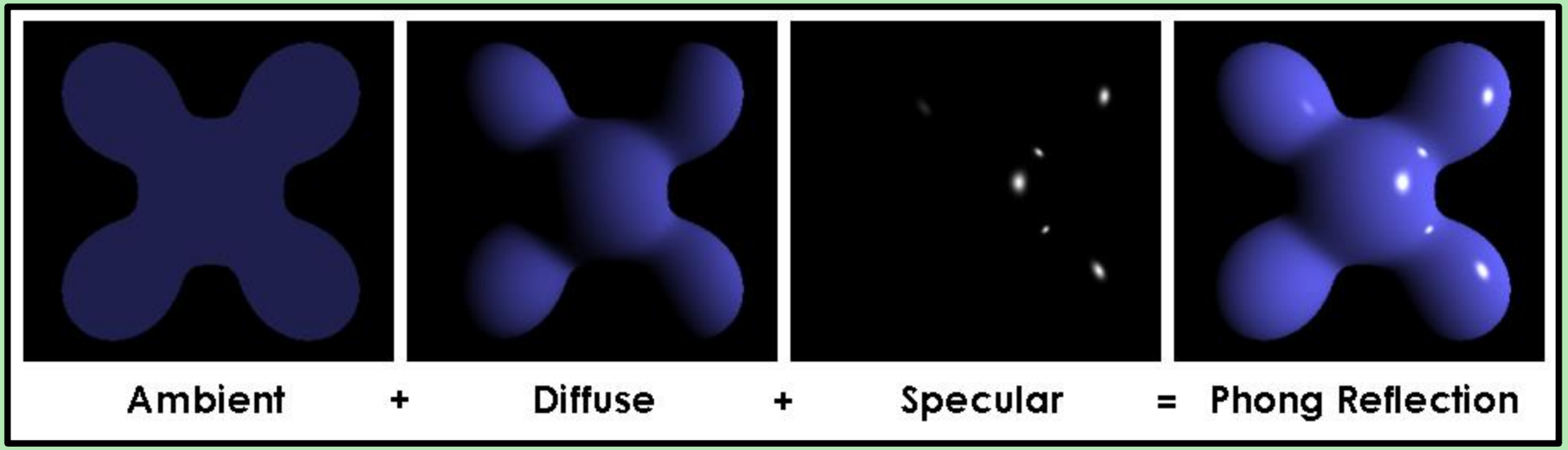


Player moves into another tile => recalculate path

PHONG VERTEX & FRAGMENT SHADERS

- With the Blinn-Plong equation, we make the game objects in our scene respond to ambiance daylight, spotlight with diffuse and specular component.

- Our shaders interpolate surface normals across rasterized polygons and computes pixel colors based on the interpolated normals and a reflection model.



By Brad Smith - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=1030364

PHONG VERTEX & FRAGMENT SHADERS

In-Game Screenshot, Demonstrating Phong Reflection (Visible Better on Trees)

PHONG VERTEX & FRAGMENT SHADERS

Phong Vertex Shader (Basic Color)

```
const vertexShader_Phong_basicColor = `

    uniform vec3 spotlightPosition; uniform vec3 spotlightDirection;

    // Built-in "in" attributes such as "position" and "normal" are not declared here,
    //  since they are automatically inserted and filled by BufferGeometry.
    out vec3 L; out vec3 N; out vec3 V; out float attenuation; out vec3 shineAt;

    void main(){

        /* LIGHTING */

        // !!! Three.js provides "position" attribute in object coordinates. We need world coordinates.
        vec3 vertexPosition = ( modelMatrix * vec4(position, 1.0) ).xyz;

        // Transform vertex position into eye coordinates
        vec3 pos = ( modelViewMatrix * vec4(vertexPosition,1.0) ).xyz;

        // Vector from vertex to spotlight in eye coordinates.
        L = normalize( (modelViewMatrix * vec4(spotlightPosition, 1.0)) - vec4(pos, 1.0) ).xyz;

        // view vector in eye coordinates from vertex to camera
        V = normalize( - pos );

        // Transform vertex normal into eye coordinates
        N = normalize( normalMatrix * normal );

        // light direction in eye coordinates
        shineAt = normalize( modelViewMatrix * vec4(spotlightDirection, 0.0) ).xyz;

        float distanceFromVertexToSpotlight = length ( vertexPosition - spotlightPosition );
        attenuation = 1.0 / ( 1.0 +  ( 0.5 * normalize( distanceFromVertexToSpotlight * distanceFromVertexToSpotlight ) ) );


        gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);

    }
`;
```

PHONG VERTEX & FRAGMENT SHADERS

Phong Fragment Shader (Basic Color)

```
const fragmentShader_Phong_basicColor = `

   precision highp float;

   in vec3 L; in vec3 N; in vec3 V; in float attenuation; in vec3 shineAt;
   uniform vec3 objectColor;
   out vec4 outColor;

   uniform float ambientDaylight;
   /* Daylight is implemented as ambient. Looks better than a directional or point source, because
those sources
      make many edges appear darker since we do not ray-trace reflections of reflections.

      Daylight gets darker in late hours of day via animation.
   */


   /* Spotlight */
   uniform vec3 diffuseProduct; uniform vec3 specularProduct; uniform float shininess;
   uniform float isSpotlightEnabled;


   void main(){

      // Blinn-Phong illumination equation

      float Kd = max( dot(L, N), 0.0 );
      vec3  diffuse = Kd * diffuseProduct;

      vec3 Halfway = normalize( L + V );
      float Ks = pow( max(dot(N, Halfway), 0.0), shininess );
      vec3  specular = Ks * specularProduct;

      if( dot(L, N) < 0.0 ) {
         // light is behind the object or camera does not see it
         specular = vec3(0.0, 0.0, 0.0);
      }
```

Jump to the next slide for the reminder of the code

PHONG VERTEX & FRAGMENT SHADERS

Phong Fragment Shader (Basic Color)

```
/*** SPOTLIGHT */

      /* A Spotlight is a point-source light, with an imaginary cone around it that restricts the light
beams
         within the scope of cutoff angle.

      To establish this restriction, first, lets look at the Dot Product Formula:
         A dot B  =  |A| * |B| * cos( angle between )
         cos( angle between )  =  (A dot B) / (|A| * |B|)

      We want to light ONLY the parts of the scene where
      the angle between  spotlightDirection "shineAt"  and  vectorFromLightToVertex "-L"  is less
than the cutoff angle we set.
      */

      float cutoffAngle = radians (20.0);    // arbitrary
      float cutoffValue = cos(cutoffAngle);
      float currentValue = dot(-L, shineAt) / length(L) / length(shineAt);

      if( currentValue < cutoffValue ) {
         // No light beams outside the desired scope cone of Spotlight.
         specular = vec3(0.0, 0.0, 0.0);
         diffuse  = vec3(0.0, 0.0, 0.0);
      }


      vec3 spotlightIntensity = isSpotlightEnabled * attenuation * (diffuse + specular);

      vec3 totalLight = vec3(ambientDaylight, ambientDaylight, ambientDaylight) +
spotlightIntensity;

      outColor = vec4 ( totalLight * objectColor  , 1.0);

   }
`;
```

PHONG VERTEX & FRAGMENT SHADERS

Phong Vertex Shader (Basic Texture)

```
const vertexShader_Phong_basicTexture = `

    uniform vec3 spotlightPosition; uniform vec3 spotlightDirection;
    out vec2 UV_coordinates;

    // Built-in "in" attributes such as "position" and "normal" are not declared here,
    //  since they are automatically inserted and filled by BufferGeometry.
    out vec3 L; out vec3 N; out vec3 V; out float attenuation; out vec3 shineAt;

    void main(){

        /* LIGHTING */

        // !!! Three.js provides "position" attribute in object coordinates. We need world coordinates.
        vec3 vertexPosition = ( modelMatrix * vec4(position, 1.0) ).xyz;

        // Transform vertex position into eye coordinates
        vec3 pos = ( modelViewMatrix * vec4(vertexPosition,1.0) ).xyz;

        // Vector from vertex to spotlight in eye coordinates.
        L = normalize( (modelViewMatrix * vec4(spotlightPosition, 1.0)) - vec4(pos, 1.0) ).xyz;

        // view vector in eye coordinates from vertex to camera
        V = normalize( - pos );

        // Transform vertex normal into eye coordinates
        N = normalize( normalMatrix * normal );

        // light direction in eye coordinates
        shineAt = normalize( modelViewMatrix * vec4(spotlightDirection, 0.0) ).xyz;

        float distanceFromVertexToSpotlight = length ( vertexPosition - spotlightPosition );
        attenuation = 1.0 / ( 1.0 +  ( 0.5 * normalize( distanceFromVertexToSpotlight * distanceFromVertexToSpotlight ) ) );


        UV_coordinates = uv;
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }
`;
```

PHONG VERTEX & FRAGMENT SHADERS

Phong Fragment Shader (Basic Texture)

```
const fragmentShader_Phong_basicTexture = `

   precision highp float;

   in vec3 L; in vec3 N; in vec3 V; in float attenuation;  in vec3 shineAt;
   in vec2 UV_coordinates;
   uniform sampler2D basicTexture;
   out vec4 outColor;

   uniform float ambientDaylight;
   /* Daylight is implemented as ambient. Looks better than a directional or point
source, because those sources
      make many edges appear darker since we do not ray-trace reflections of
reflections.

      Daylight gets darker in late hours of day via animation.
   */

   /* Spotlight */
   uniform vec3 diffuseProduct; uniform vec3 specularProduct;
   uniform float shininess; uniform float isSpotlightEnabled;

   void main(){

      // Blinn-Phong illumination equation

      float Kd = max( dot(L, N), 0.0 );
      vec3  diffuse = Kd * diffuseProduct;

      vec3 Halfway = normalize( L + V );
      float Ks = pow( max(dot(N, Halfway), 0.0), shininess );
      vec3  specular = Ks * specularProduct;

      if( dot(L, N) < 0.0 ) {
         // light is behind the object or camera does not see it
         specular = vec3(0.0, 0.0, 0.0);
      }
```

Jump to the next slide for the reminder of the code

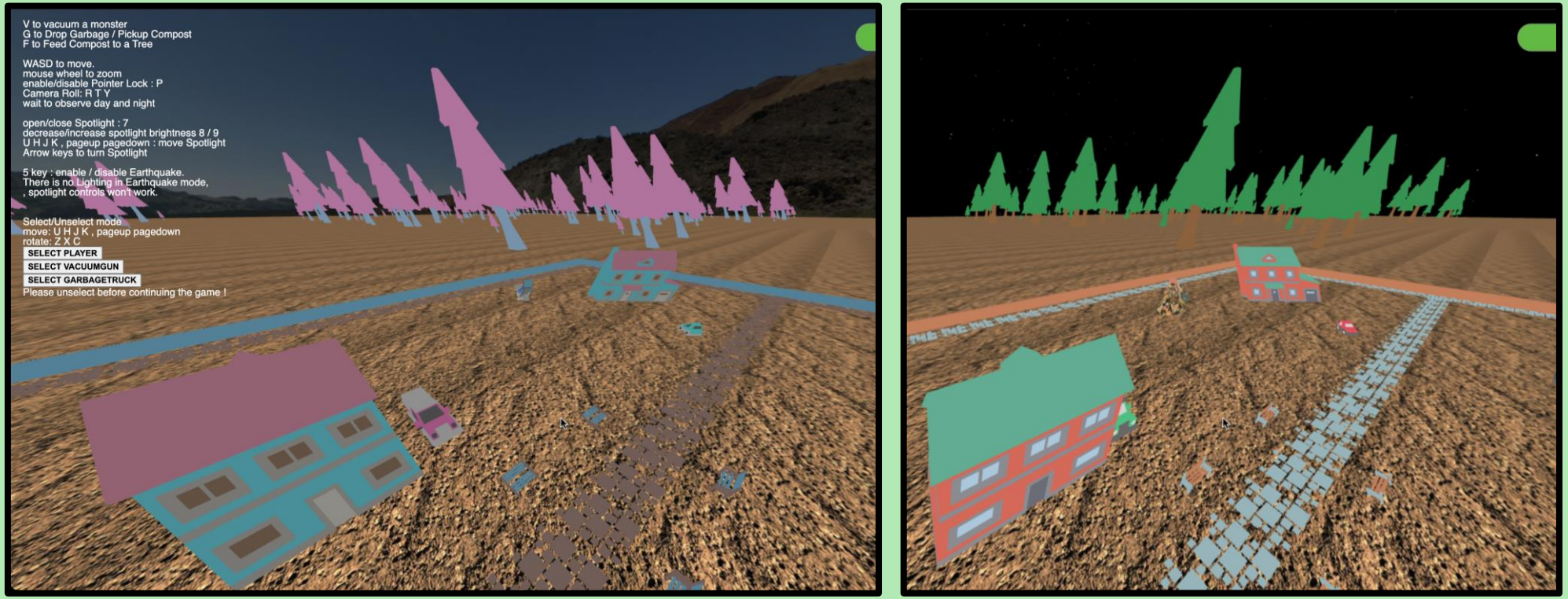
PHONG VERTEX & FRAGMENT SHADERS

Phong Fragment Shader (Basic Texture)

```
/*** SPOTLIGHT */

    /* A Spotlight is a point-source light, with an imaginary cone around it that restricts the light beams
       within the scope of cutoff angle.

    To establish this restriction, first, lets look at the Dot Product Formula:
       A dot B  =  |A| * |B| * cos( angle between )
       cos( angle between )  =  (A dot B) / (|A| * |B|)

    We want to light ONLY the parts of the scene where
    the angle between  spotlightDirection "shineAt"  and  vectorFromLightToVertex "-L"  is less than the cutoff angle we set.
    */

    float cutoffAngle = radians (20.0);    // arbitrary
    float cutoffValue = cos(cutoffAngle);
    float currentValue = dot(-L, shineAt) / length(L) / length(shineAt);

    if( currentValue < cutoffValue ) {
       // No light beams outside the desired scope cone of Spotlight.
       specular = vec3(0.0, 0.0, 0.0);
       diffuse  = vec3(0.0, 0.0, 0.0);
    }

    vec3 spotlightIntensity = isSpotlightEnabled * attenuation * (diffuse + specular);

    vec3 totalLight = vec3(ambientDaylight, ambientDaylight, ambientDaylight) + spotlightIntensity;

    vec3 textureColor = (texture(basicTexture, UV_coordinates)).xyz;

    outColor = vec4( totalLight * textureColor , 1.0);

  }
`;
```

EARTHQUAKE VERTEX & FRAG. SHADERS

- The earthquake is a custom feature implemented by our team. It's shaders provide vertex displacement (propotional to the heights of objects) along x and z axes periodically and (also periodic) color inversion for a psychedelic effect.



In-Game Screenshots to Demonstrate Color Inversion

EARTHQUAKE VERTEX & FRAG. SHADERS

Earthquake Vertex Shader (with Color & with Texture)

```
const earthquake_vertexShader_withColor = `

   uniform float time;
   out float time_;

   void main(){

      time_ = time;

      float jiggle = sin(time) / 10.0;

      float jiggled_x = position.x + (jiggle * position.y);
      float jiggled_z = position.z + (jiggle * position.y);

      vec3 jiggledPosition = vec3(jiggled_x + jiggle/3.0 , position.y,
jiggled_z);

      gl_Position = projectionMatrix * modelViewMatrix *
vec4(jiggledPosition, 1.0);
   }
`;
```

```
const earthquake_vertexShader_withTexture = `

   uniform float time;
   out vec2 UV_coordinates;
   out float time_;

   void main(){

      time_ = time;

      float jiggle = sin(time) / 10.0;

      float jiggled_x = position.x + (jiggle * position.y);
      float jiggled_z = position.z + (jiggle * position.y);

      vec3 jiggledPosition = vec3(jiggled_x + jiggle/3.0 , position.y,
jiggled_z);

      UV_coordinates = uv;
      gl_Position = projectionMatrix * modelViewMatrix *
vec4(jiggledPosition, 1.0);
   }
`;
```

EARTHQUAKE VERTEX & FRAG. SHADERS

Earthquake Fragment Shader (Color Changing) (with Color & with Texture)

```
const colorChanging_fragmentShader_withColor = `

   precision highp float;

   uniform vec3 objectColor;

   in float time_;

   out vec4 outColor;

   void main(){

      float colorJiggle = abs( sin(time_/4.0) ) * 0.8;

      vec3 invertedColor = vec3 ( 1.0 - objectColor.r , 1.0 - objectColor.g, 1.0 -
objectColor.b);

      float red   = mix( objectColor.r , invertedColor.r, colorJiggle );
      float green = mix( objectColor.g , invertedColor.g, colorJiggle );
      float blue  = mix( objectColor.b , invertedColor.b, colorJiggle );

      outColor = vec4(red, green, blue , 1.0);
   }
`;
```

```
const colorChanging_fragmentShader_withTexture = `

   precision highp float;

   in vec2 UV_coordinates;
   uniform sampler2D basicTexture;

   in float time_;

   out vec4 outColor;

   void main(){

      float colorJiggle = abs( sin(time_ / 4.0) ) * 0.6;

      vec3 textureColor = (texture(basicTexture, UV_coordinates)).xyz;

      vec3 invertedColor = vec3 ( 1.0 - textureColor.r , 1.0 - textureColor.g, 1.0 -
textureColor.b);

      float red   = mix( textureColor.r , invertedColor.r, colorJiggle);
      float green = mix( textureColor.g , invertedColor.g, colorJiggle);
      float blue  = mix( textureColor.b , invertedColor.b, colorJiggle);

      outColor = vec4(red, green, blue , 1.0);
   }
`;
```

EARTHQUAKE VERTEX & FRAG. SHADERS

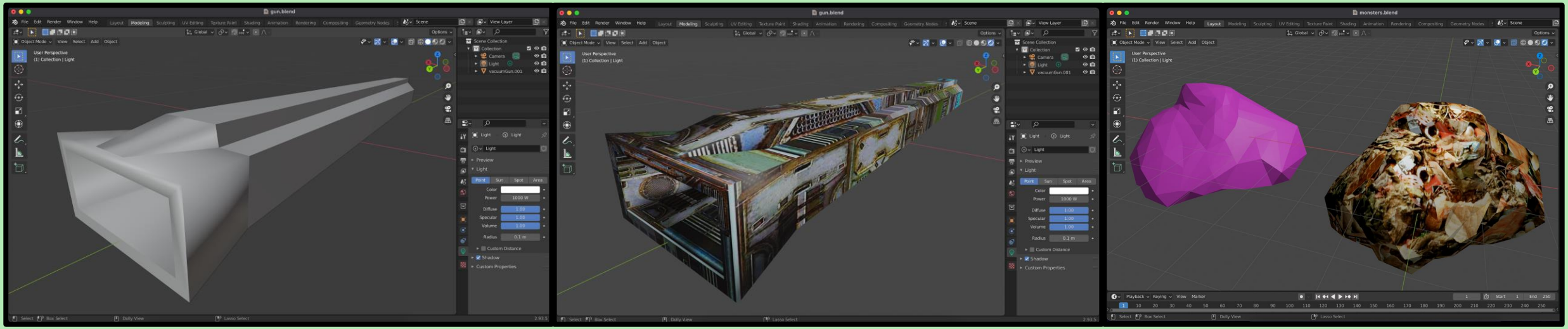
Earthquake Vertex & Fragment Shader (Earthquake Resistant)

```
const earthquakeResistant_vertexShader = `

   out vec2 UV_coordinates;

   void main(){
      UV_coordinates = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position,
1.0);
   }
`;
```

```
const earthquakeResistant_fragmentShader = `

   precision highp float;

   in vec2 UV_coordinates;
   uniform sampler2D basicTexture;
   out vec4 outColor;

   void main(){
      outColor = vec4( (texture(basicTexture, UV_coordinates)).xyz , 1.0
);
   }
`;
```

3D MODELLING

- For the 3D assets in the game, we used a mixture of pre-ready and custom-built models. To create the custom-built ones, we used **Blender**.
- In Blender, we modeled and textured our Vacuum Gun and Garbage Monster models using extrusion, sculpting and UV unwrapping.



Vacuum Gun and Garbage Monster Models Before & After UV Unwrapping

THANKS FOR PAYING ATTENTION

COMING UP NEXT: TRAILER-LIVE DEMO-Q&A SESSION