Computer Science Colloquium
MicroRogue with Pixelbox.js

Dr. Micah Schuster

Summer 2020

Wentworth Institute of Technology

Download Pixelbox.js (it's free!):
https://pixwlk.itch.io/pixelbox

Download the project:
https://github.com/mdschuster/MicroRogue

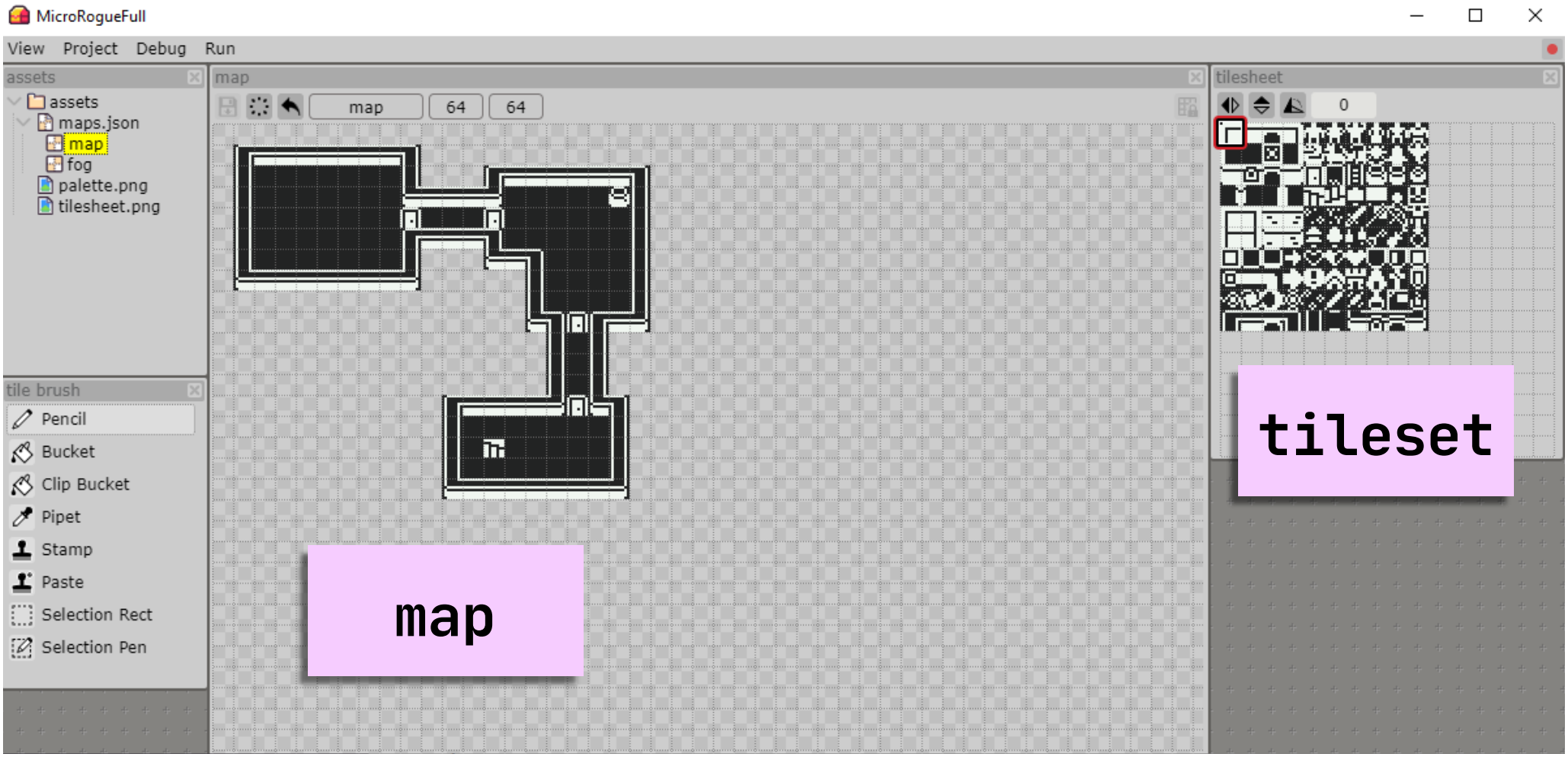
First: Tools

- **Any editor You Like**: I'll be using VSCode, but you can use any editor you like that can handle JavaScript.
- **Pixelbox.js:** It's Free! **https://pixwlk.itch.io/pixelbox**
- **Starter Code (and Finished Code):**
  - **https://github.com/mdschuster/MicroRogue**
  - Includes:
    - Pixelbox.js project with starter code and finished code
    - This presentation!
    - The finished code is in the .full directory

Second: Setup

- **Pixelbox.js:** Run the Pixelbox app and open the project.pixelbox file from the GitHub repository (that you cloned or downloaded).
  - You'll be greeted with a small map that I started for you.
  - There's also a second map called '`fog`'.
  - You'll see a tileset that you can paint with (Kenney's Micro Roguelike)
  - Click Debug->Devtool. This gives debugging info when the run the program.
- **Write Code:** Use your editor to follow along and edit the javascript files as we go though the lecture.

Second: Setup



Quick Intro to JavaScript

- **Variable Player:** Just Do It

```
c = new Player();
let value = 0;
```

No Type!

- **Control Flow:** Same as Other Languages

```
for(let i = 0; i < array.length; i++){
  //stuff
}
```

```
if(currentTime < 0){
  //stuff
}
```

Quick Intro to JavaScript

- Functions (Outside of Classes):

```
function keyPressed(){
  //stuff
}
```

No Return Type, but you can still return a value

- Classes (ES6+):

```
class Player{
  constructor(){
    this.x = 0;
  }

  jump(){
    //stuff
  }
}
```

Constructor:
Just `constructor()`

`this` is everywhere!

Functions start with the name only

Pixelbox.js

- Editor and JavaScript library that lets you create graphics, levels, music, and sound.
- Easily create a project and test you game in one click.
- Includes a simple tile editor, a music tracker and a sound effect creator too.
- Free and open source.

A couple of games created using Pixelbox.js:

Monkey Warp

Trisk

The Starter Code

`main.js`

Where the program starts and where the main game loop is located. Also imports all of our other classes

`camera.js`

Controls our view. I've written this one for you already.

`character.js`

All the data and methods for our interaction with the player.

`fog.js`

Controls what is visible on our map. Simple square visibility around our player.

`enemy.js`

All the data and methods for our enemies.

The Starter Code

main.js

Where the program starts and where the main game loop is located. Also in other

camera.js

Controls our view. I've written this one for you already.

There's already some utility code the Character and Enemy class and I'll explain what it does as we fill out those classes.

er.js

l methods for

ith the player.

fog.js

Controls what is visible on our map. Simple square visibility around our player.

enemy.js

All the data and methods for our enemies.

The Character Class

- Contains a few important variables:
  - The tile/(x,y) pixel position of the player
  - Information about the full map
  - What sprites in the map are walls
  - Information about enemies
  - What sprite represents the player
  - A few `booleans` and `ints` to help us out

Quick Note:

I've tried to make the code as understandable as possible, even the utility code, so it's certainly not implemented in the best way.

Character Constructor

```
export default class Character{
  constructor(map, enemies){
    this.spriteNumber = 4;
    this.x = 16;
    this.y = 16;
    this.row = this.y / 8;
    this.col = this.x / 8;
    this.moveAmount = 8;

    this.map = map;
    this.boundarySprites = [144,145,147,148,150,
                            153,39,40]
    this.enemies = enemies;
  }
}
```

(x,y) pixel position and tile position

Pixel move amount

Sprite number for objects, like walls

Let's Move (Still in Character Class)

```
move(direction){
   let chosen = this.getTile(direction);
}
```

`getTile()` gives us back the tile object or `null` if there is no tile there.

Let's Move (Still in Character Class)

```
move(direction){
   let chosen = this.getTile(direction);
   if(chosen==null){
      return false;
   }
}
```

`getTile()` gives us back the tile object or `null` if there is no tile there.

`move()` returns whether we actually moved

Let's Move (Still in Character Class)

```
move(direction){
   let chosen = this.getTile(direction);
   if(chosen==null){
      return false;
   }
   //we will add more ifs here

   this.row = chosen.y;
   this.col = chosen.x;
   this.x = this.col * 8;
   this.y = this.row * 8
   this.tile = this.map.get(this.col, this.row);
   return true;
}
```

`getTile()` gives us back the tile object or `null` if there is no tile there.

`move()` returns whether we actually moved

Update the (row, col), (x, y) pixel, and tile information

Over to `main.js`

```
//imports..

exports.update = function(){

}
```

This update function is run every frame.
All your drawing, movement, etc. should be done in here.

Over to `main.js`

```
//imports..

let background = getMap("map");
paper(0);
let enemies = [];

const c = new Character(backGround, enemies);

exports.update = function(){

}
```

`c` is the player

`backGround` is the "map" that we created in the editor.

`enemies` is the array that will hold all our enemies.

`paper()` is the background color that is not a tile.

Over to `main.js`

```
//initial stuff..

var moved = false;
exports.update = function(){
  if(btnp.up) {
    moved = c.move("up");
  }
  else if(btnp.right) {
    moved = c.move("right");
  }
  else if(btnp.down) {
    moved = c.move("down");
  }
  else if(btnp.left) {
    moved = c.move("left");
  }
}
```

`btn` is how we get the button that was pressed. Pixelbox.js only supports up, down, left, right, A, B

`btnp` and **`btnr`** get pressed and released in the current frame

Over to `main.js`

```
//initial stuff..

exports.update = function(){
  //all the input stuff

  cls();
  draw(background, 0, 0);

  c.update();

}
```

We sill need to write the character **`update()`** function.

`cls()` is clears the screen (so we can draw everything again)

`draw()` draws the object at a specific pixel coordinate

Character `update()`

```
update() {
  this.draw();
}

draw() {
  sprite(this.spriteNumber, this.x, this.y);
}
```

`draw()` is already written. But, it's important. To draw a sprite on the screen, you need the sprite number (from the sheet) and the (x, y) pixel location.

All **`update()`** will do for us is draw the sprite

Boundaries

```
move() {
  let chosen = this.getTile(direction);
  if(chosen==null){
    return false;
  }
  if(this.boundaryCheck(chosen) == true){
    return false;
  }
  //we will add more ifs here

  this.row = chosen.y;
  this.col = chosen.x;
  this.x = this.col * 8;
  this.y = this.row * 8
  this.tile = this.map.get(this.col, this.row);
  return true;
}
```

`boundaryCheck()` just looks at the input tile and checks if it's in our `boundarySprites` array

Simple "Camera"

I've already written the camera class for you.
It takes a target (the player) and keeps them within a box, so it's not a tile-by-tile follow.

There's lots of different ways to handle a camera, play around with the code I've given you.

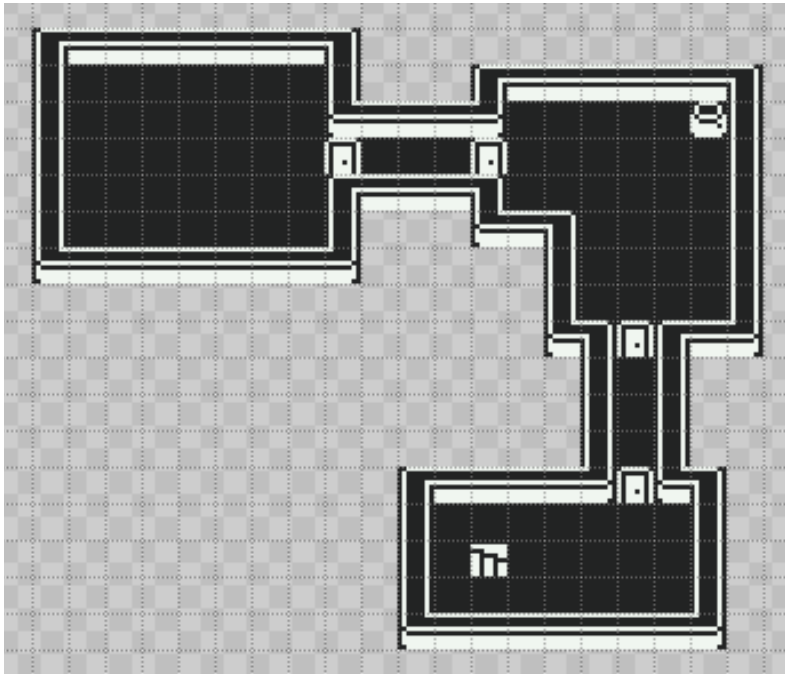
```
const camera = new Camera(c);
exports.update = function(){
  //all the input stuff
  //character update

  camera.update();
}
```

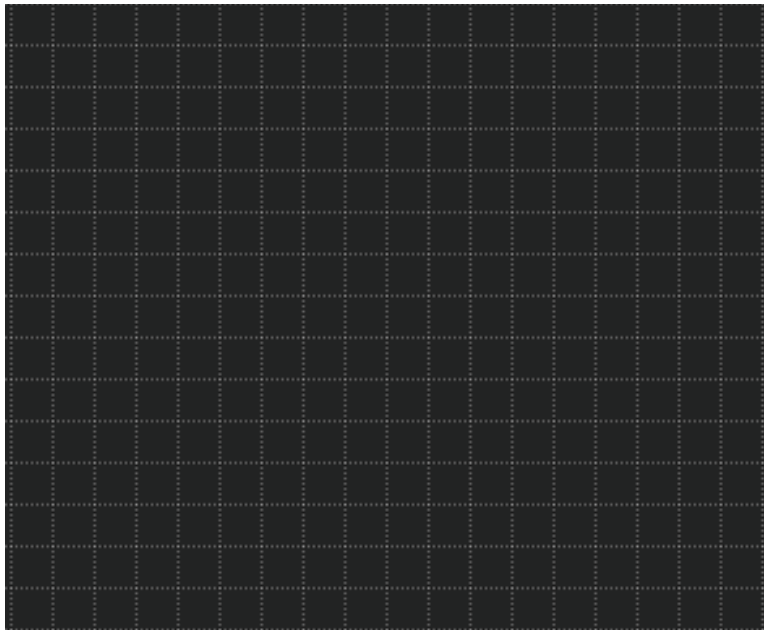
main.js

“Fog”

`map`



`fog`



Essentially, we are overlaying the fog tilemap on everything (the **`draw()`** must be at the end of our main update function). Then we remove tiles from the fogMap to uncover what’s under.

"Fog"

Many rogue like games don't allow the player to see the entire map all the time. So, I've written a quick fog class that unveils a square area around the player.

```
let fogMap = getMap("fog");
const fog = new Fog(fogMap);
exports.update = function(){
  //all the input stuff
  //character update

  camera.update();

  fog.updateFog(c);
  draw(fogMap, 0, 0);
}
```

Essentially, we are overlaying the fog tilemap on everything (the `draw()` must be at the end of our main update function). Then we remove tiles from the fogMap to uncover what's under.

`main.js`

Intermission: Where are we?

At this point, we almost have a working game:

- Interactable player character
- Interesting environment to move around in

What we still need:

- Enemy + "AI"

Tweakable:

- Fog clearing size
- Camera tracking
- Enemy AI

Ultimately, tweak the game to make it play the way you want. Dev studios spend years before and after release tweaking their games.

Enemy

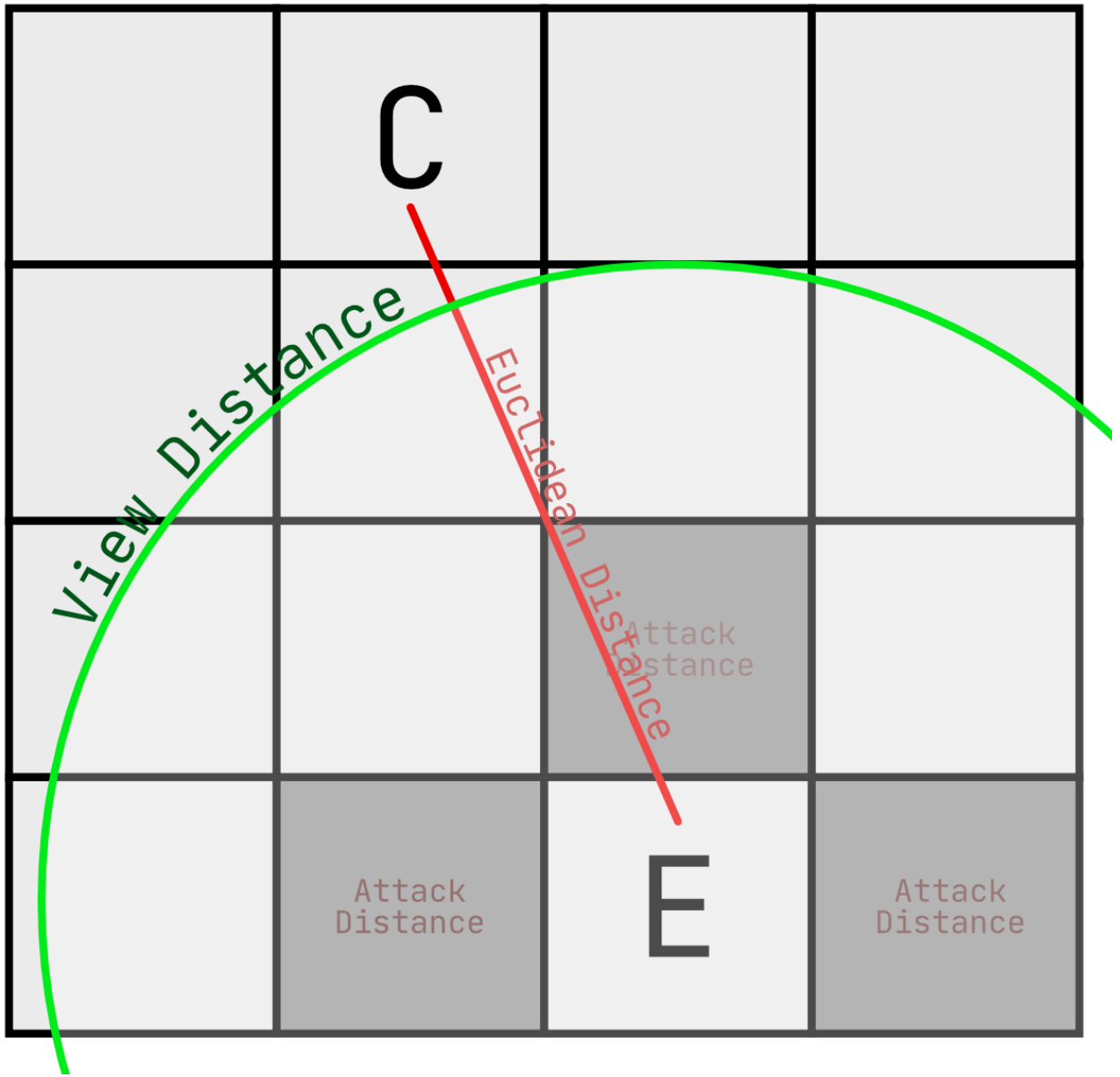
```
export default class Enemy{
  constructor(row, col, player, map){
    this.row = row;
    this.col = col;
    this.x = col * 8;
    this.y = row * 8;
    this.map = map;
    this.tile = this.map.get(col, row);
    this.boundarySprites = [144,145,147,148,150,
                            153,39,40];
    this.player = player;
    this.spriteNumber = 20;
  }
}
```

Enemy has some of the same data and methods as Character. This would be a good place to use some ES6 inheritance.

Enemy Movement

We're going to base the movement on the Euclidean distance between the enemy and the player.

We'll use two values to help us, a view distance and an attack distance



Wentworth INSTITUTE OF TECHNOLOGY

Enemy Movement

```
move(){
  let distance = this.computeDistance(this.tile,
                                      this.player.tile);
  if(distance > 5) {
    return;
  }

  if(distance <= 1){
    //do the attack
    return;
  }
}
```

If the distance to the player is > 5, don't do anything.
If the distance is <= 1, do our attack.

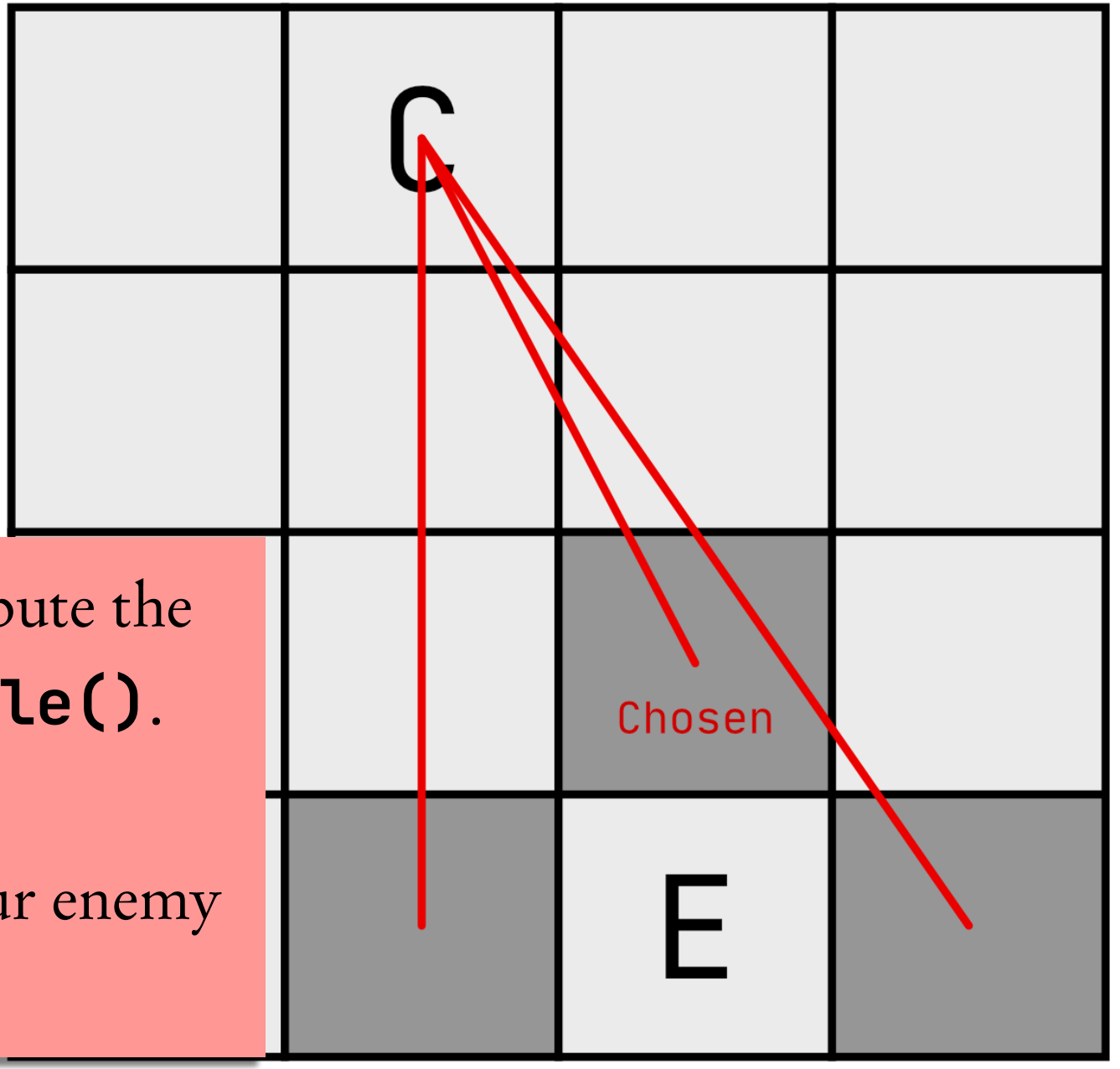
The function to compute the Euclidean distance is **`computeDistance()`**

Which Tile to Move to?

Same process, compute the Euclidean distance to the player, but for the 4 adjacent tiles.

The function to compute the closest tile is **`getTile()`**.

This is the extent of our enemy "AI"



Which Tile to Move to?

```
move(){
   //other stuff

   let chosen = this.getTile();
   if(chosen != null) {
      this.row = chosen.y;
      this.col = chosen.x;
      this.x = this.col * 8;
      this.y = this.row * 8;
      this.tile = this.map.get(this.col, this.row);
   }
}
```

Get the closest tile

Update all our data

Back to Main

```
//other stuff
const enemy1 = new Enemy(6, 18, c, background);
enemies[0]=enemy1;
exports.update= function(){
  // movement stuff

  c.update();

  if(moved) {
    enemies.forEach(e => {e.update()});
    moved = false;
  }
  enemies.forEach(e => {e.draw()});

  camera.update()

  //fog stuff
}
```

```
void update(){
  move();
}
```

If the player actually moved, then the enemies can move

Draw all the enemies

The Sky is the Limit

You now have the basic layout of the game.

For the next steps:

- Add some health to the player and enemy so you can implement the attack function.
- Create more levels. You can use one large map that teleports the player or multiple maps and load them when needed
- Implement some better design, an FSM would be better to handle turns, some inheritance would probably be useful as well.