FINAL PROJECT "LIGHTS OUT"

By Tom Horn & Patrick Feltey

WHAT IS OUR PROJECT?

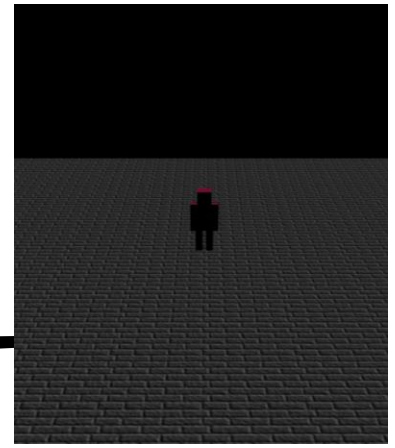
For our project we decided to use our skills of THREE.js we have learned this semester to develop a game which we are calling "Lights Out".

Combining aspects from all the projects with our own raycasting based collision system.

HOW IS THIS RELATED TO THE STUDY OF GRAPHICS?

- Like stated before in the introduction we are using aspects from all the projects and bringing everything together.
- In this project we have implemented custom WEBGL shaders, transformations using matrices for the model animations, ray casting to create collision detection, adding a skybox and various textures, and finally a handful of in game mechanics like a timer, collectables, game over screen, and a points system.
- ThreeJS has no collision system built in suitable for interacting with walls – use of raycasting to create a collision system without a third party library

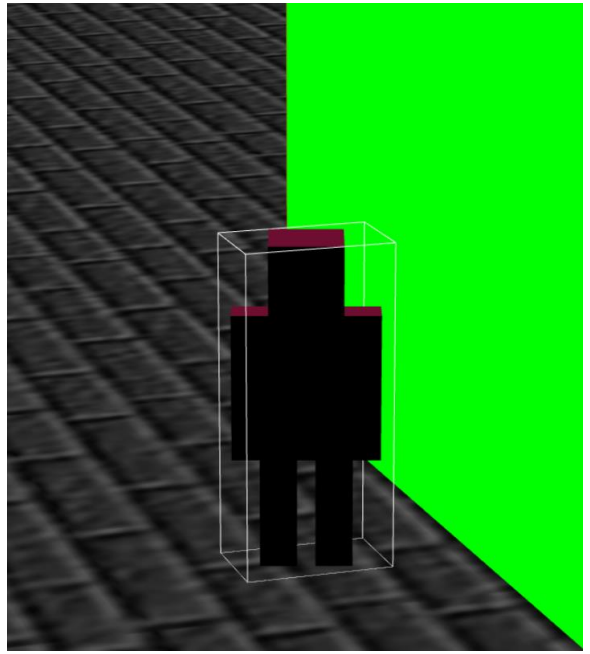
THE APPROACH

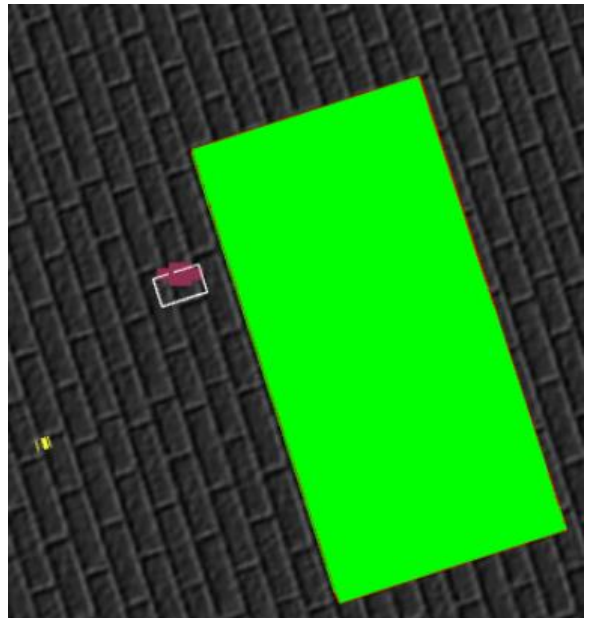


- Started with project 3 because of the rendering loop
- We chose to use this one primarily because of the implementation of a rendering loop appropriate to a game.
- Then using this we could create models using THREE.js to create walls, a player model, and torch collectable.
- Afterwards the plan was to fix a camera onto the player to have a third person perspective.
- Begin development of our collision system and expand maze structure from single wall.
- Following the camera placement, we were going to find and implement textures for various objects and apply some neat shaders to the torch model to give a sense of glow.
- Implement gameification to add the timer, picking up torches granting extra time and bonus points, model animations, adding a game over screen, and an exit.
- Finally, if time permitted - sprint/stamina, ambient light decrement with time, score keeping, hard difficulty, torch lighting, infinite torches, etc.

IMPLEMENTATION PART 1 - COLLISION AND THE MAZE

- Basic raycasting from the player model to walls
- Researched and attempted collision with bounding boxes - some improvement
  - Decided against using any 3rd party library like physjs or ammo.js
  - "Bounce back" implemented but bounding boxes allowed purposeful clipping through walls
  - Manipulating bounding boxes did not feel good as a player
- Reimplemented raycasting with lessons about clipping distances using bounding boxes for measurements
- Progression from one wall, to corners, to smaller corridors - negative QA (break it)





BOUNDING BOX VS RAYCASTING

- Bounding Box:
  - Update head movement
  - Get bounds of each wall piece and head location
  - Check for intersection
- Raycasting:
  - Player movement is constrained to 2 directions
  - Only need to check potential vertex collisions +/-Z
  - "Predictive Movement" to not let players clip through walls = +/-z * x steps

```
let wallBBoxHelper = new THREE.BoxHelper(walls[i]);
head.updateMatrixWorld();
walls[i].updateMatrixWorld();
let headPos = head.geometry.boundingBox.clone();
headPos.applyMatrix4(head.matrixWorld);
headPos.applyMatrix4(new THREE.Matrix4().makeTranslation(x,y,z))

let wallPos = walls[i].geometry.boundingBox.clone();
wallPos.applyMatrix4(walls[i].matrixWorld);

if (headPos.intersectsBox(wallPos)) return true;
```

```
for (let i = 0; i < head.geometry.vertices.length; i++)
    // Create the raycasting angles and the raycaster
    let localPos = head.geometry.vertices[i].clone();
    let globalAngle = localPos.applyMatrix4(head.matrix);
    let rayAngle = globalAngle.sub(head.position);

    // Get any collisions
    let raycaster = new THREE.Raycaster(headPos, rayAngle.clone().normalize());
    let collisions = raycaster.intersectObjects(scene.children);

    // Check if the the collision is TOUCHING the actual wall (or less)
    if (collisions.length > 0)

        if (collisions[0].distance < rayAngle.length() + predictAmt)
            if (collisions[0].object.type == "wall")
            ...
```

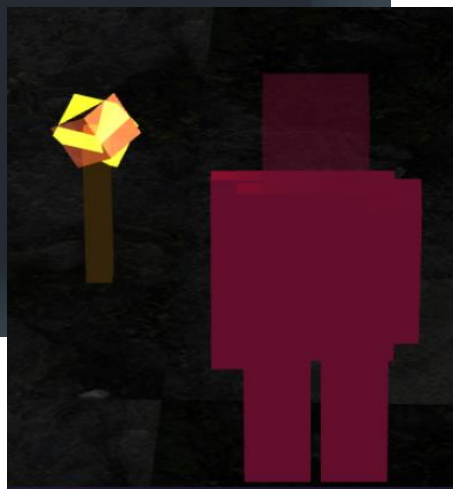
IMPLEMENTATION PART 2 - MODELS, TEXTURE, SHADING

- Player, walls, torch model development occurred concurrently with collision
- Custom shader for a glow effect on the flame of the torches
- Then added the basic skybox like in project 3 and then we searched the internet to find some textures for the game to give off the feel we wanted.
- Following that came creating matrices to allow for transitions and rotations to have the player and torch to have some animations.
- For the flame rotation and along with the glow shader it ended up giving off a very cool looking flame exactly how we wanted it to appear.
- Synthesized components needed for a basic workable game

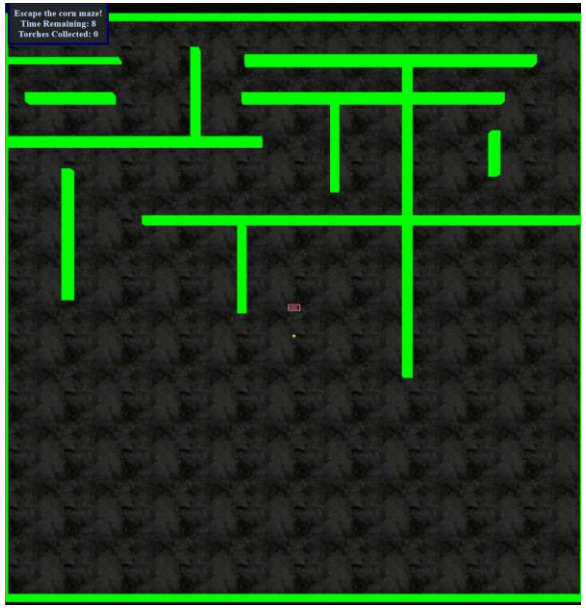
ARMS / TORCH ROTATION

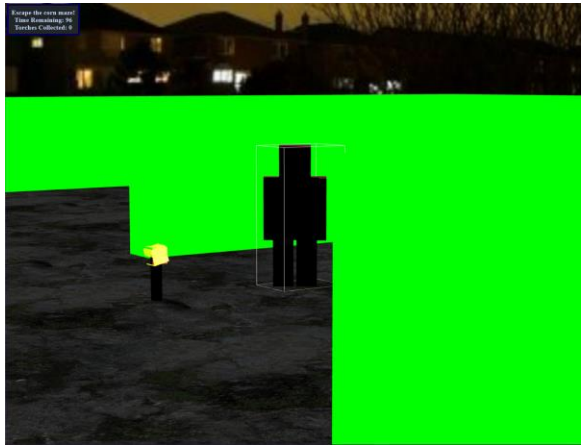
- Although the animation was simple for the torch flames to look good the arms another story.
- The arms also required translations to appear more realistic and look connected.
  - Took hours of trial and error so the arms did not appear to be falling off the player's body.
- Need to limit the arc that the arms move and have them move in opposite directions at the same time.
  - Floating point variables not 100% accurate. (issue with arms stopping movement)

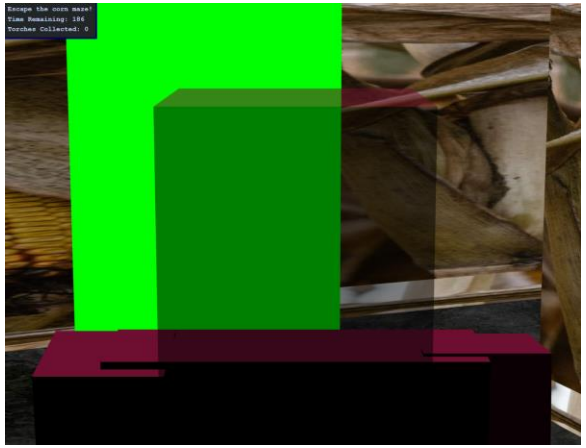
```
armRight.geometry.applyMatrix4(armMatrix);
armRight.geometry.verticesNeedUpdate = true;
armMatrix.set(
    1, 0, 0, 0, //d messes with x translation
    0, 1, 0,-1*(transY), //h messes with y translation
    0, 0, 1, -1*(transZ), //p messes with z translation
    0, 0,0,1

);

armLeft.geometry.applyMatrix4(armMatrix);
armLeft.geometry.verticesNeedUpdate = true;

rotX += rotXTheta;
if(rotX < -.5)
{
    rotXTheta *= -1;
    transY *= -1;
    transZ *= -1;
    flip = false;
    console.log("Flip = "+flip);
}
```

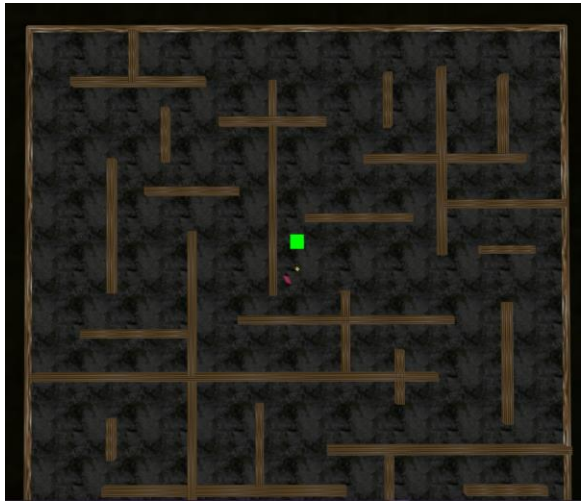


BASIC GAME PROGRESSION





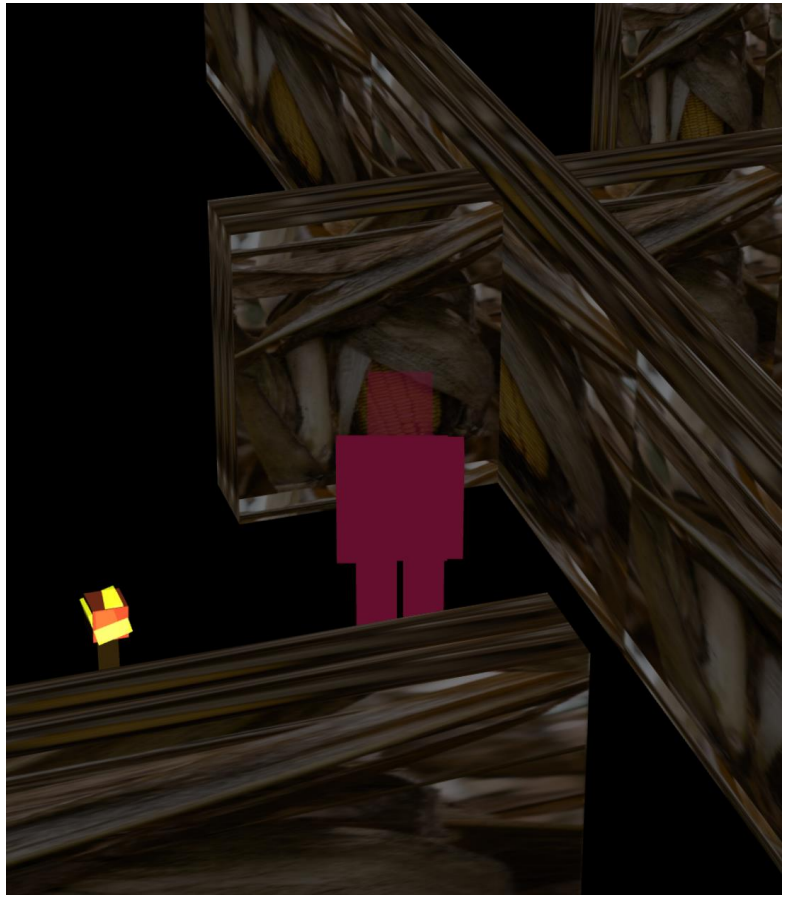




66 commits

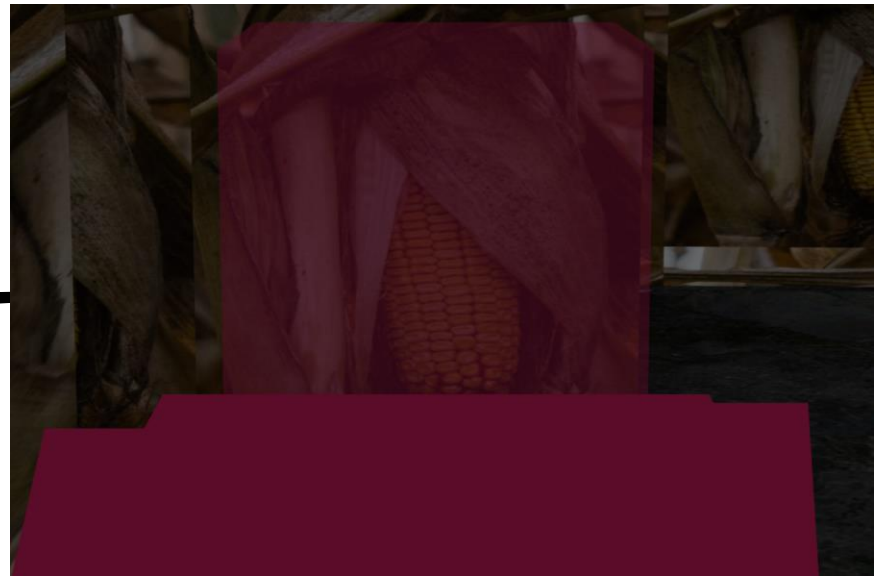
IMPLEMENTATION PART 3 - TOWARDS A GAME

- After workable win/loss the focus shifted towards extra game elements
- Refined torch system into infinite semi-random torches
- Tied time and torch system to score system
- Ambient light decrease as time goes on
- Two difficulties related to camera type - raycasting collision-based camera
    - First person perspective inspiration from a progression piazza reply (Thanks Andrew Phipps!)



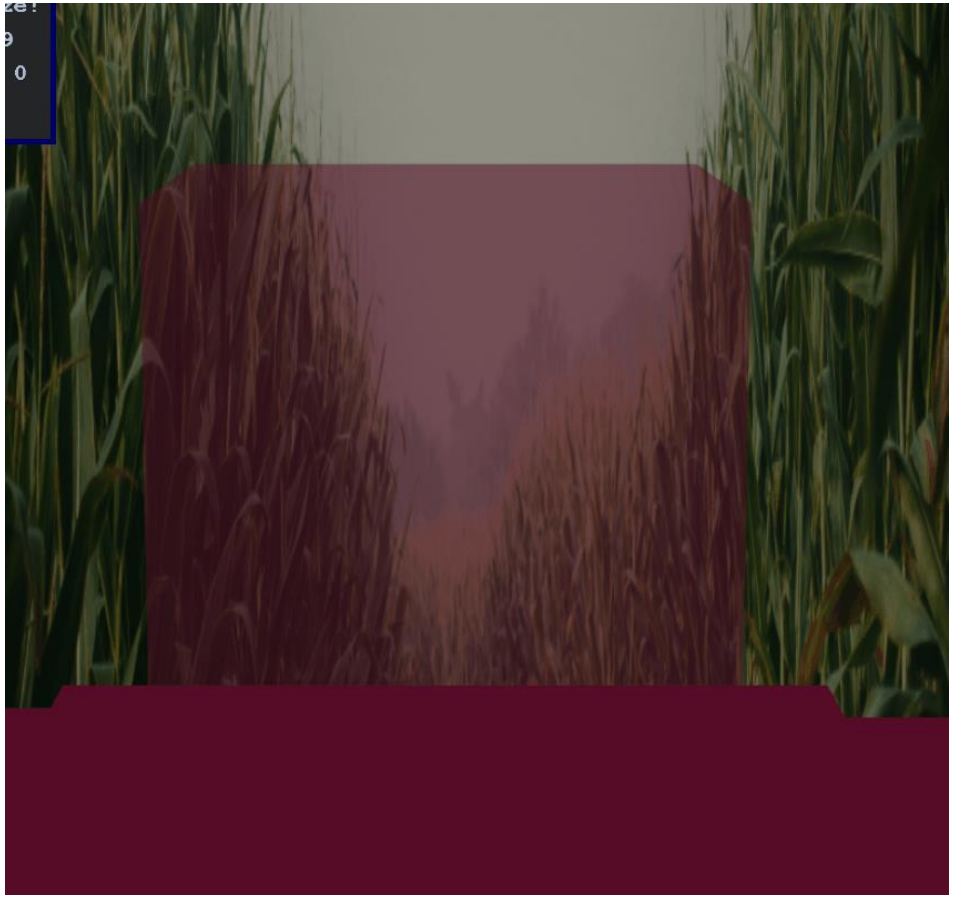
CAMERA COLLISIONS

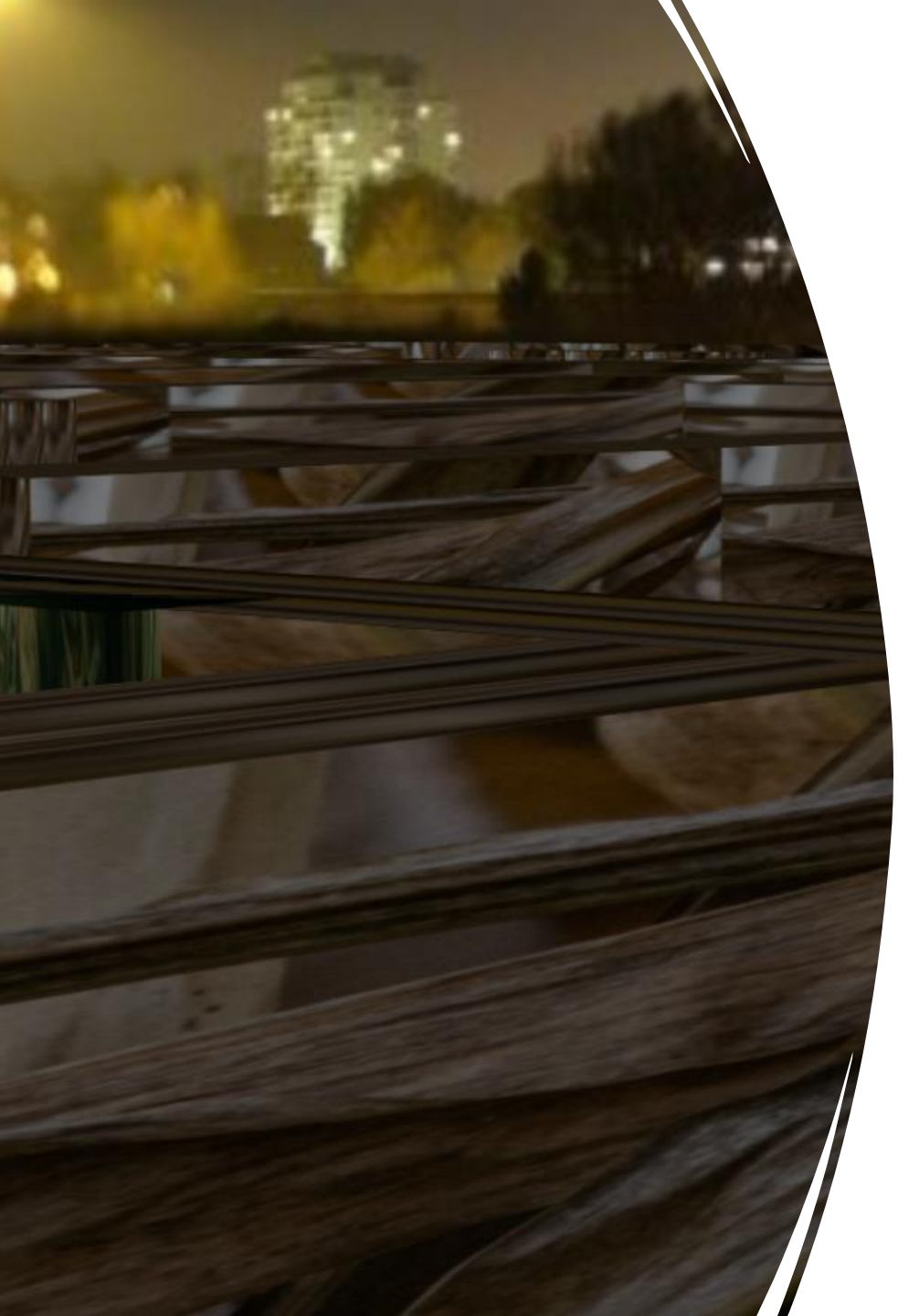
- Backing up close to a wall/corner clips the camera through wall
- Raycasting to prevent this
  - Invisible box added to camera
  - With each backup or rotation predict camera location
  - Move camera in if collision



IMPLEMENTATION PART 4

- Quality of life changes:
  - 90 degree turning with collision
  - Music controls
  - Torch point light
  - Final bug check
  - Scoring difficulty turn up





MILESTONES

1. Blank template of ground by stripping P3 as a render loop template
2. Player model created
3. Collision first attempt with raycasting - can detect walls but no actual collision
4. Torch model and small maze model created
    1. Began work on customized shader to implement to give off appearance of glow with two cubes.
5. Collision switched to bounding boxes, collision is very jank but exists
    1. "Bounce off wall refined" but bounding boxes were not consistent
    2. Reimplemented as raycasting with lessons learned
6. Player and torch animation refined
    1. Using matrices instead of the preimplemeted THREE.js positions
    2. Arms of player required a large amount of adjustments.
7. Raycasting collision refined and tested, full maze created, torches added, ambient light decrease - working base game
8. "Gameification" – score keeping, randomization, intro, game over

RESULTS

Overall, we are both very happy with Lights Out. We were able to use the graphics techniques from this semester and create a working game from near scratch.

We implemented all things in our project proposals, including tentative ones, and were able to add in new features as we went, such as using collision for a camera mode.