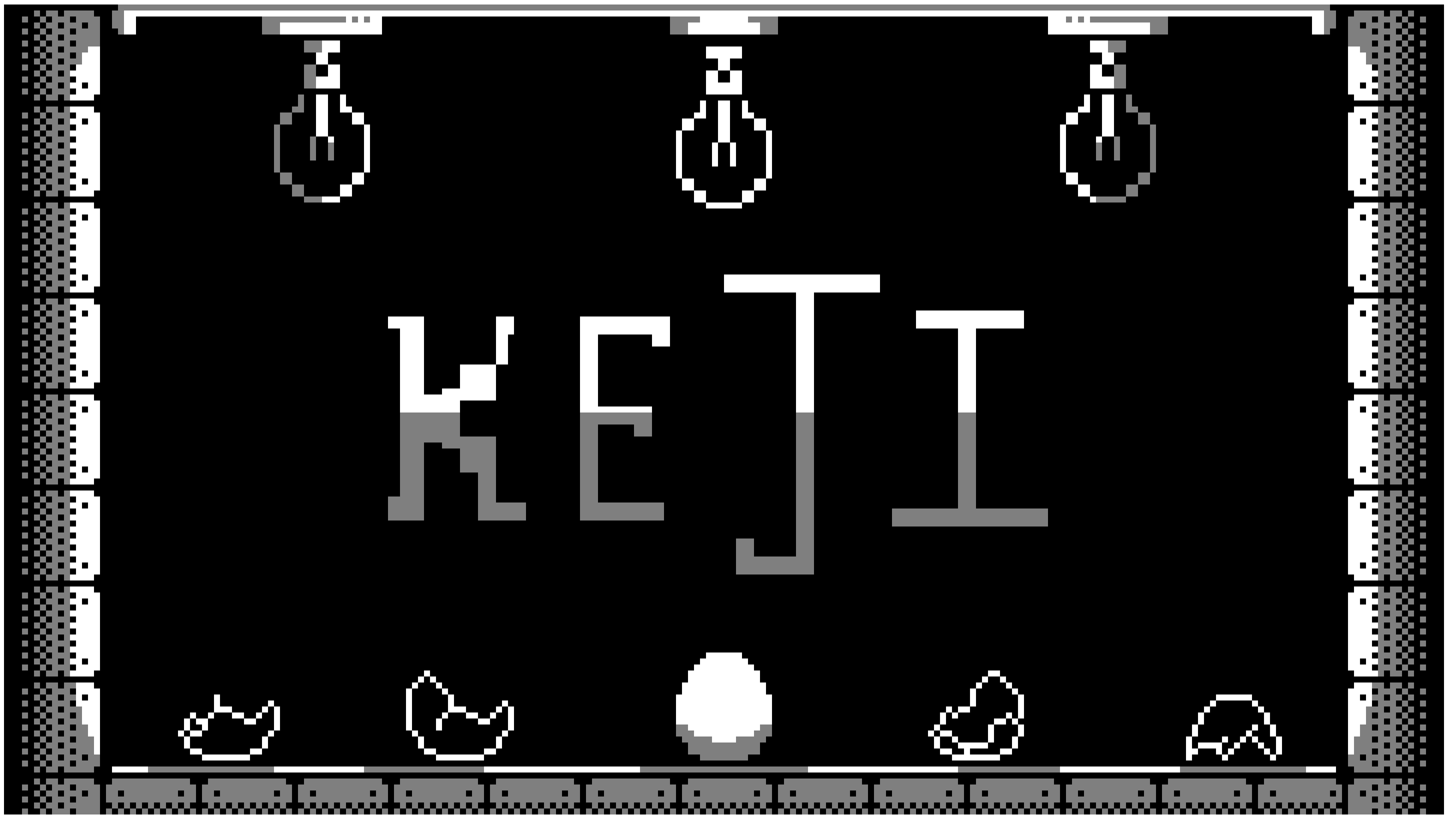
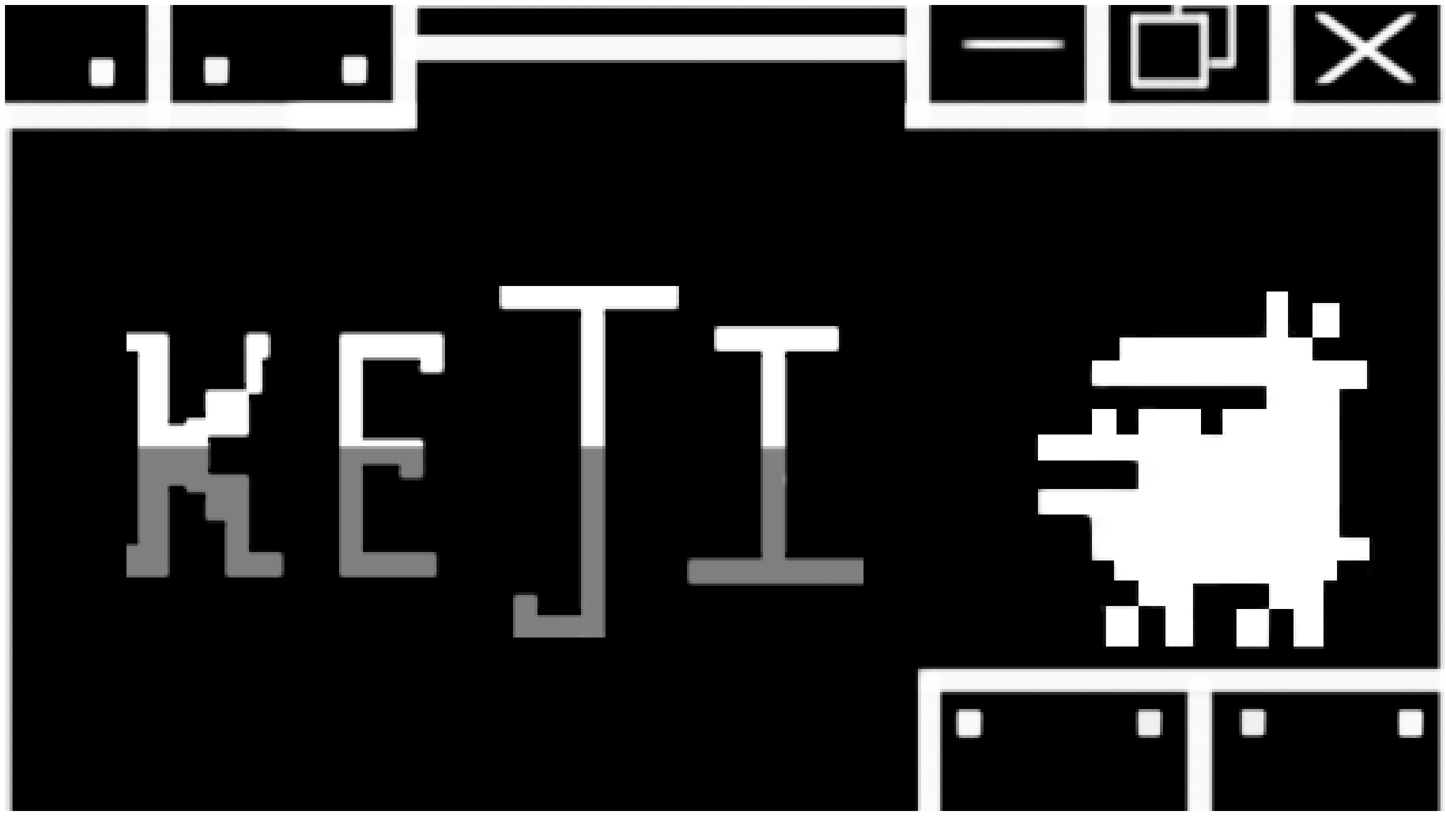
KEJI

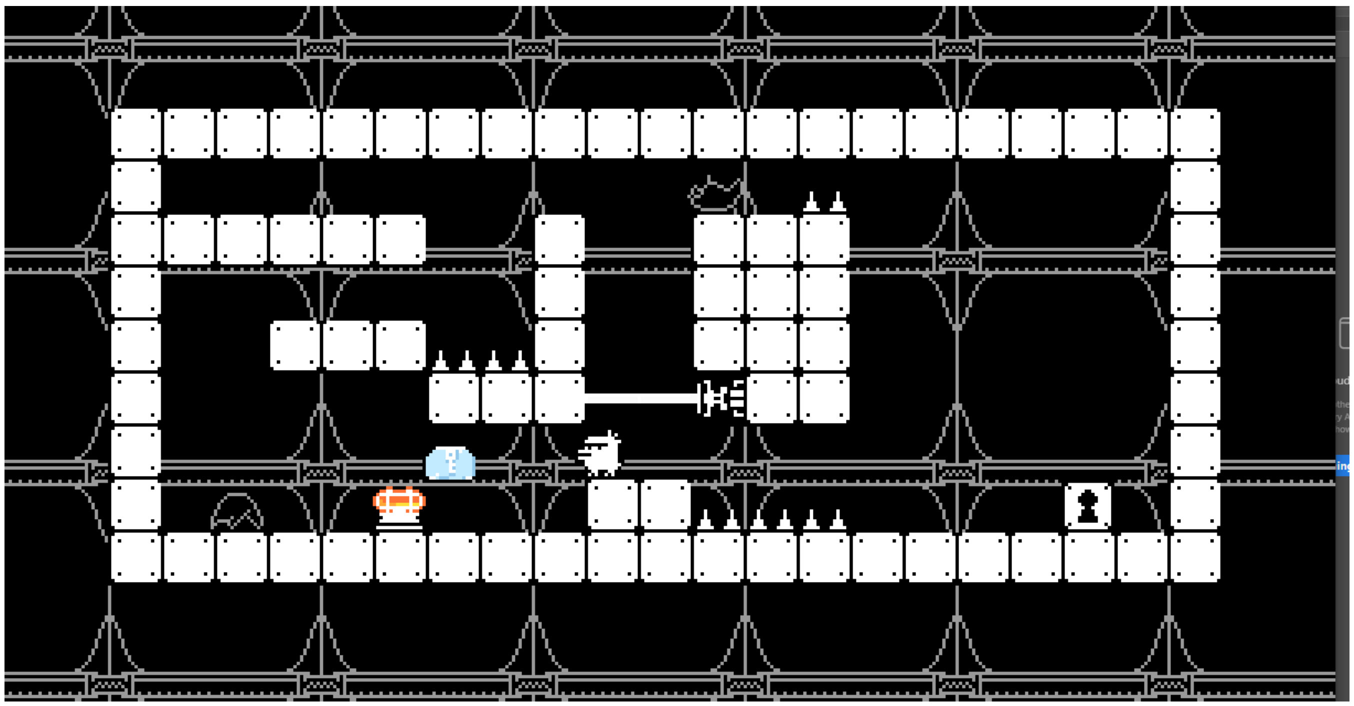


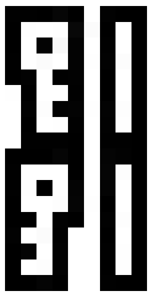
Nathan Muck @na10	Programmer
Tommy Xiong @tommyxiong3	Artist
Simon Meyer @simonsaysdesign	Artist Programmer
Carson Johnson @cranso	Programmer

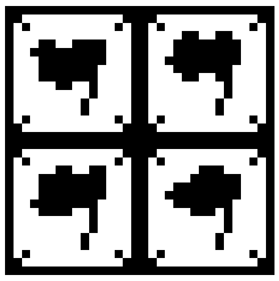

KEJI

Elevator Pitch:

Keji is a puzzle platformer where players resize the browser window to affect the level's size and shape. They play as a baby chicken that pushes an ice block around to warm up by a heater. Once the ice block melts, the key inside is used to advance to the next level. Players need to figure out how to navigate the level while being faced with various obstacles.







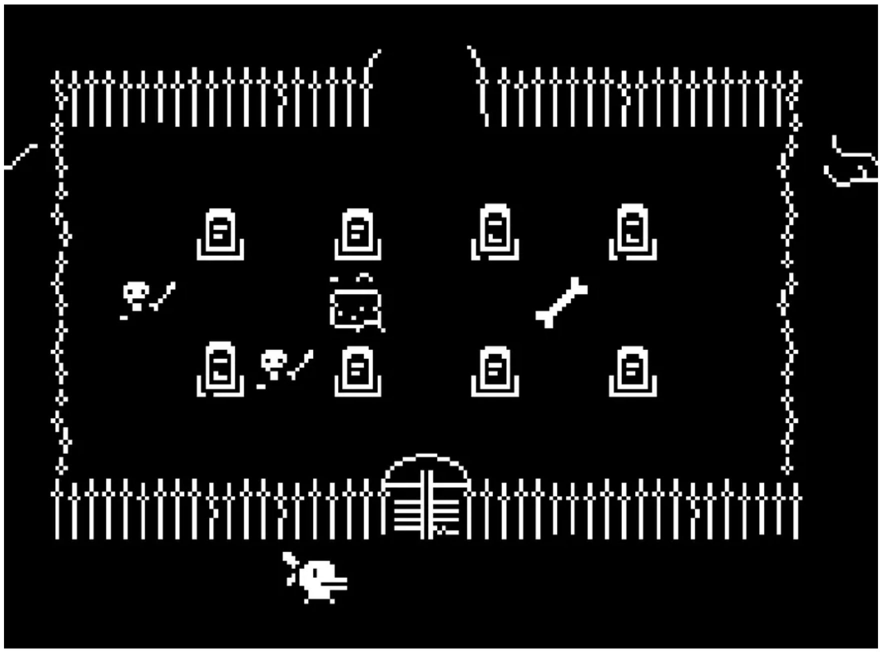
Introduction:

The word keji means “cage” in Japanese. It is also the name of our player character. Keji, is “caged” in a broiler farm where he will suffer a very cruel fate. The day he is born he will be misstreated by farm workers and treated like a disposable object rather than a living being. If he survives the assembly line process, where dozens don’t, he will spend the rest of his life with other chickens in a dark room that is cramp and never cleaned. He will be over fed to the point where his own weight will crush himself from the inside. It isn’t just the overfeeding that he has to worry about, but also the other chickens in the farm that are either sick, or unaware, that will try to peck him to death. Once he reaches 40 days old he will have lived his life only knowing fear and be sent to the slaughter house where he will meet his end.

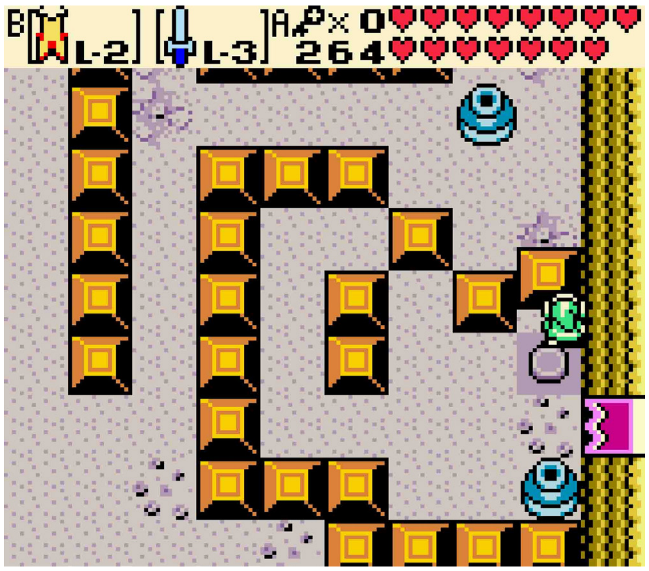
Inspirations:

We have been inspired by many sources, but our main ones are The Legend of Zelda, Minit, Inside, Little Nightmares, Tamagotchi, and Limbo. We wanted to have simple and readable sprites that convey a friendly mannter to the player. That being said, we also wanted it to be creepy in a suble way. We tried doing this with music and ambigous sprites such as floating hands, zombie-like chicks, and spikes.

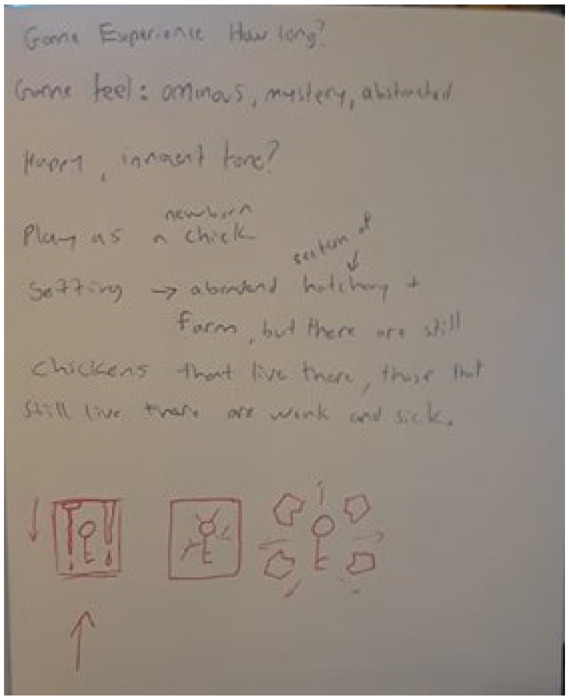
A lot of the art was heavily influenced by Minit and Tamagotchi. The simplistic nature of them is what we wanted to capture.



In the end, gameplay and puzzles used were influenced by Zelda, and Little Nightmares. Moving crates and blocks from one area to another is the goal and the player must do this by resizing the browser window to solve the puzzles. Zelda in the early games had a lot of block moving in their puzzles. We made it so Keji wouldn't be able to harm the enemies directly. It must be done by pushing the enemies into spikes with crates that spawn in the level or with the ice block itself. This is similar to how in games like Little Nightmares the players are left vulnerable to enemies.



Concept Art and Ideas:

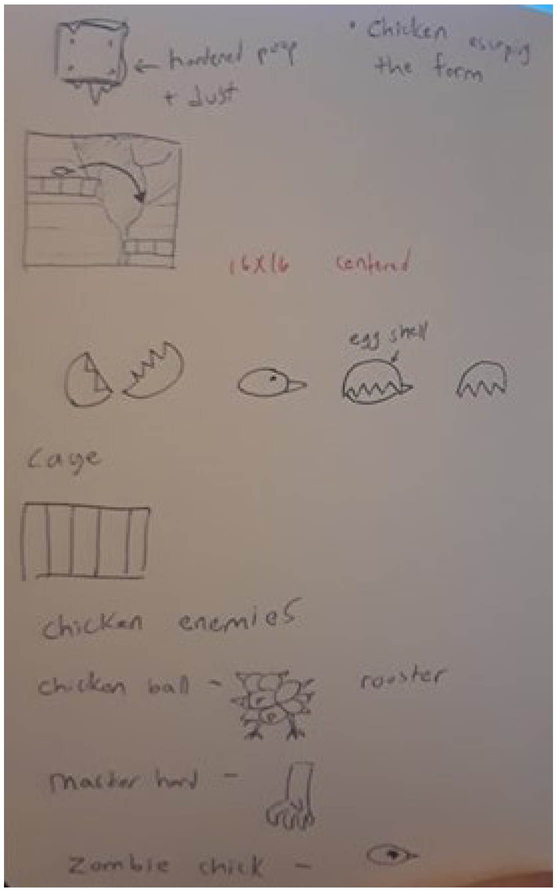

Game Experience How long?

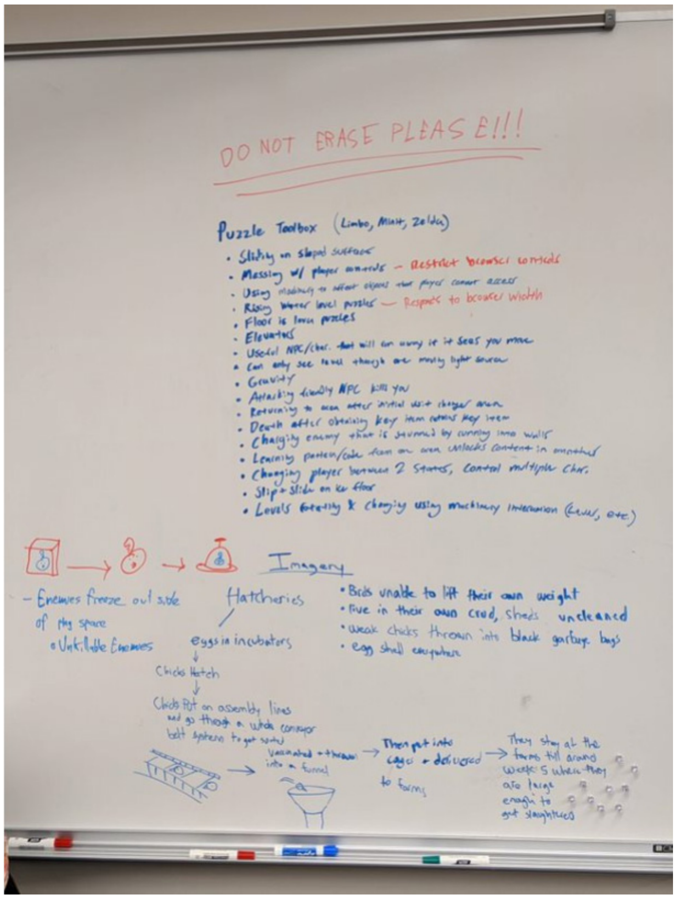
Game feel: ominous, mystery, abstracted

Happy, innocent tone?

Play as a newborn chick

Setting → abandoned section of hatchery + farm, but there are still chickens that live there, those that still live there are weak and sick.




DO NOT ERASE PLEASE!!!

Puzzle Toolbox (Limbo, Mist, Zelda)

- Sliding on sloped surfaces
- Messing w/ player controls – Restrict browser controls
- Using mobility to affect objects that player cannot access
- Rising water level puzzles – Responds to browser width
- Floor is lava puzzles
- Elevators
- Useful NPC/char. that will run away if it sees you move
- Can only see level through one moving light source
- Gravity
- Attacking friendly NPC kills you
- Returning to area after initial visit changes area
- Death after obtaining key item retains key item
- Charging enemy that is stunned by running into walls
- Learning pattern/code from one area unlocks content in another
- Changing player between 2 states, control multiple char.
- Slip + slide on ice floor
- Levels rotating & changing using machinery interaction (levers, etc.)

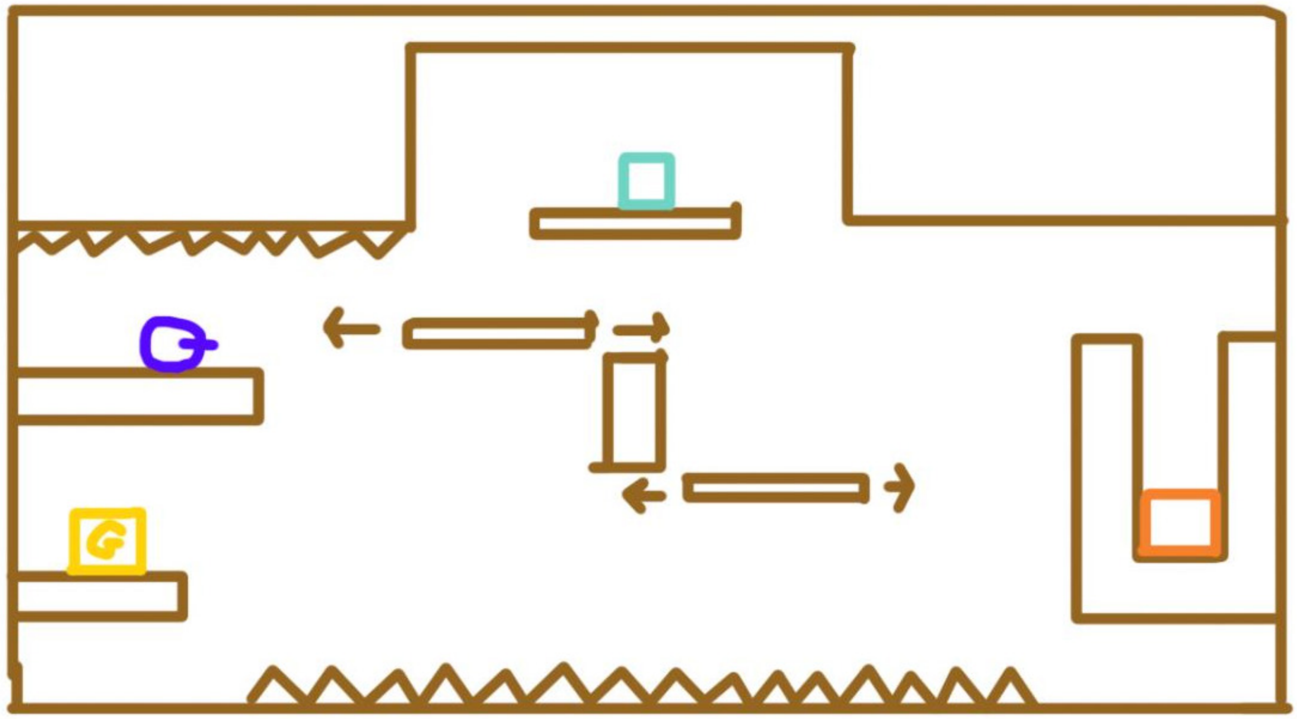
– Enemies freeze outside of the space
- Unkillable Enemies

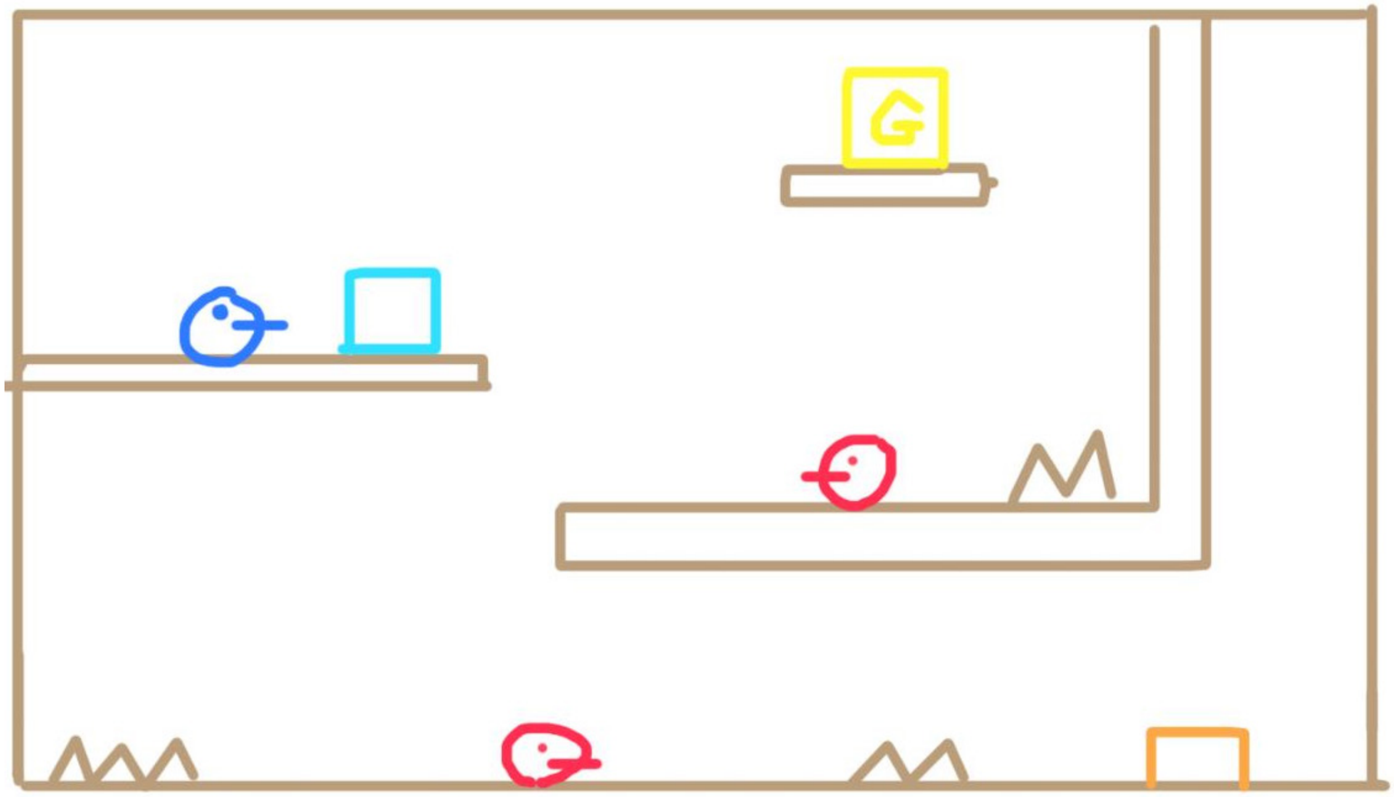
Hatcheries

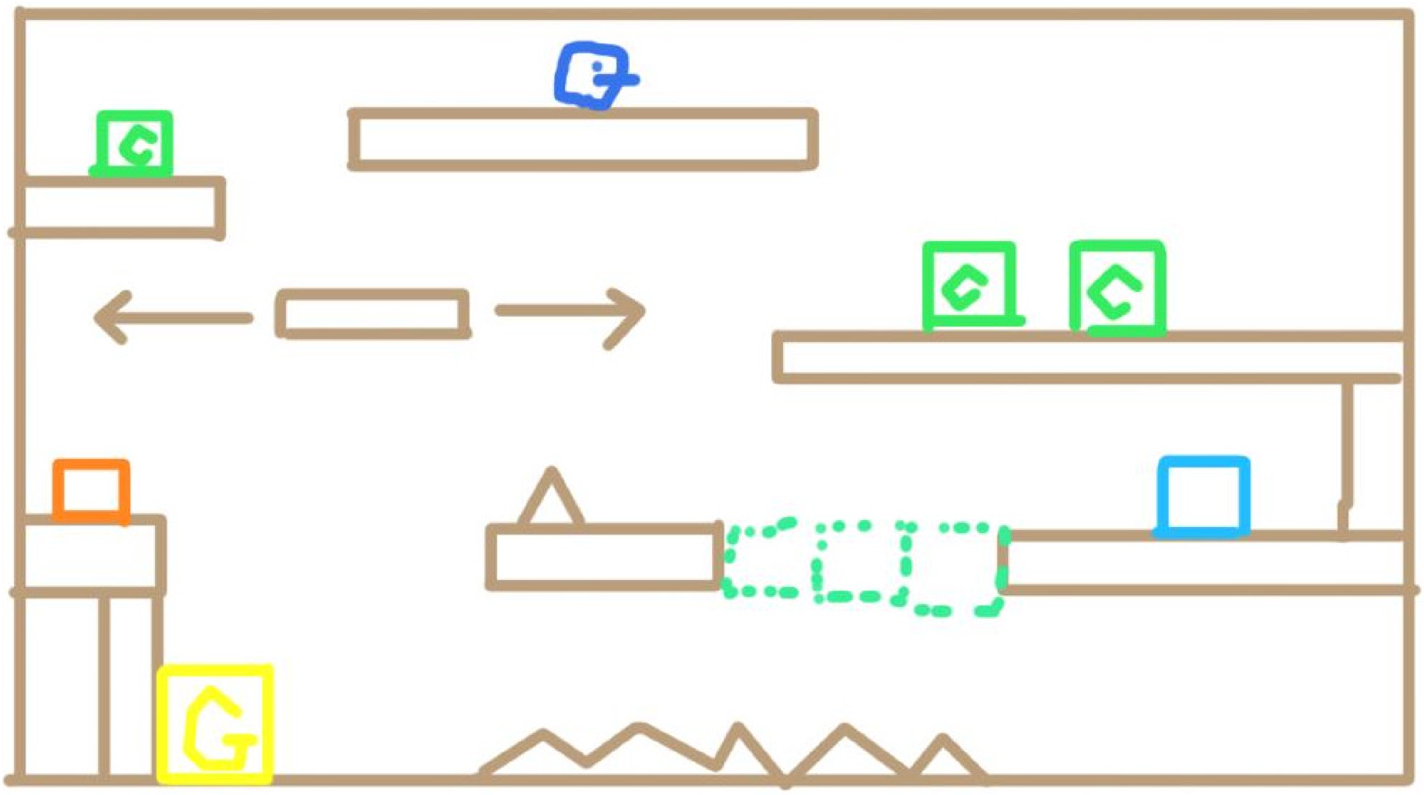
eggs in incubators → Chicks Hatch → Chicks Put on assembly lines and go through a whole conveyor belt system to get sorted → Vaccinated + thrown into a funnel → Then put into cages + delivered to farms → They stay at the farms till around week 5 when they are large enough to get slaughtered

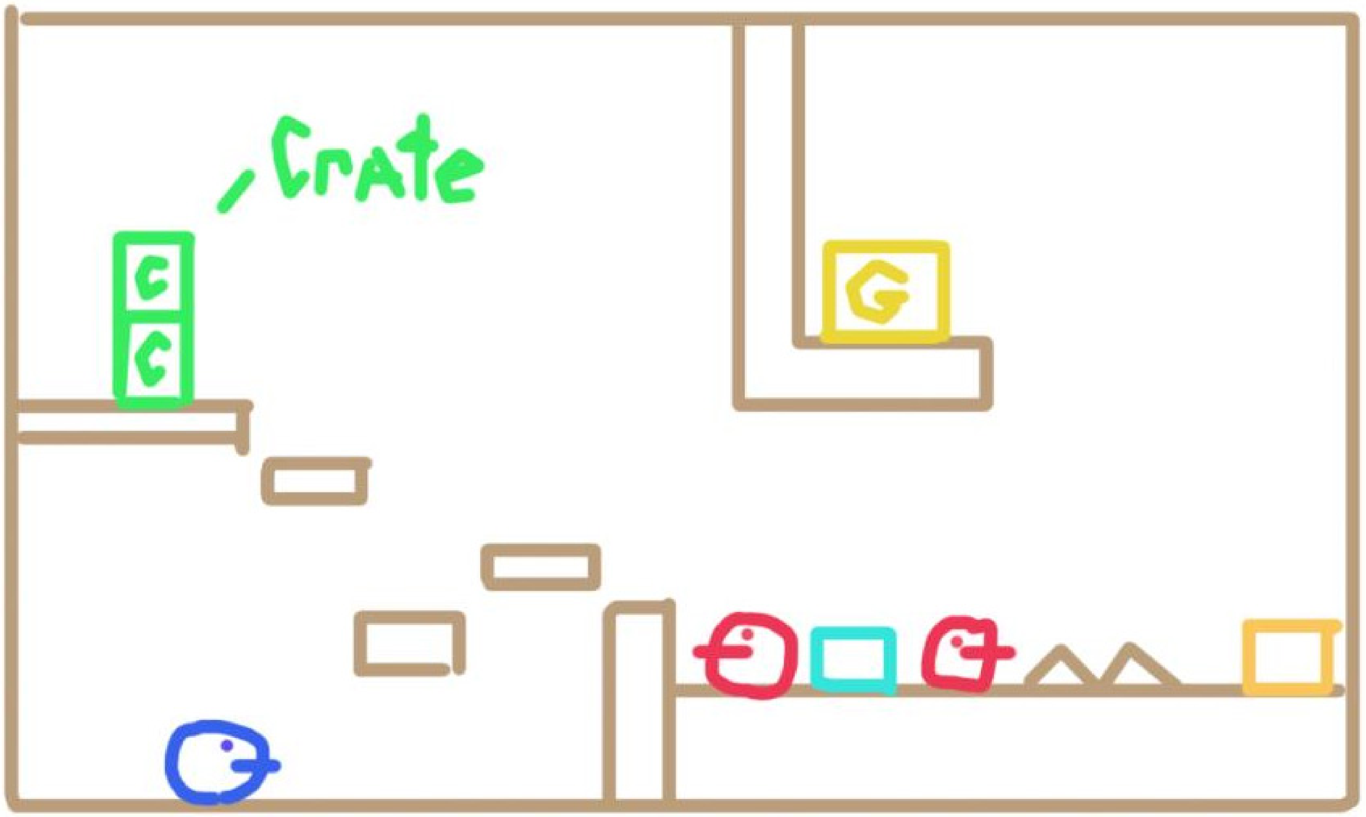
Imagery

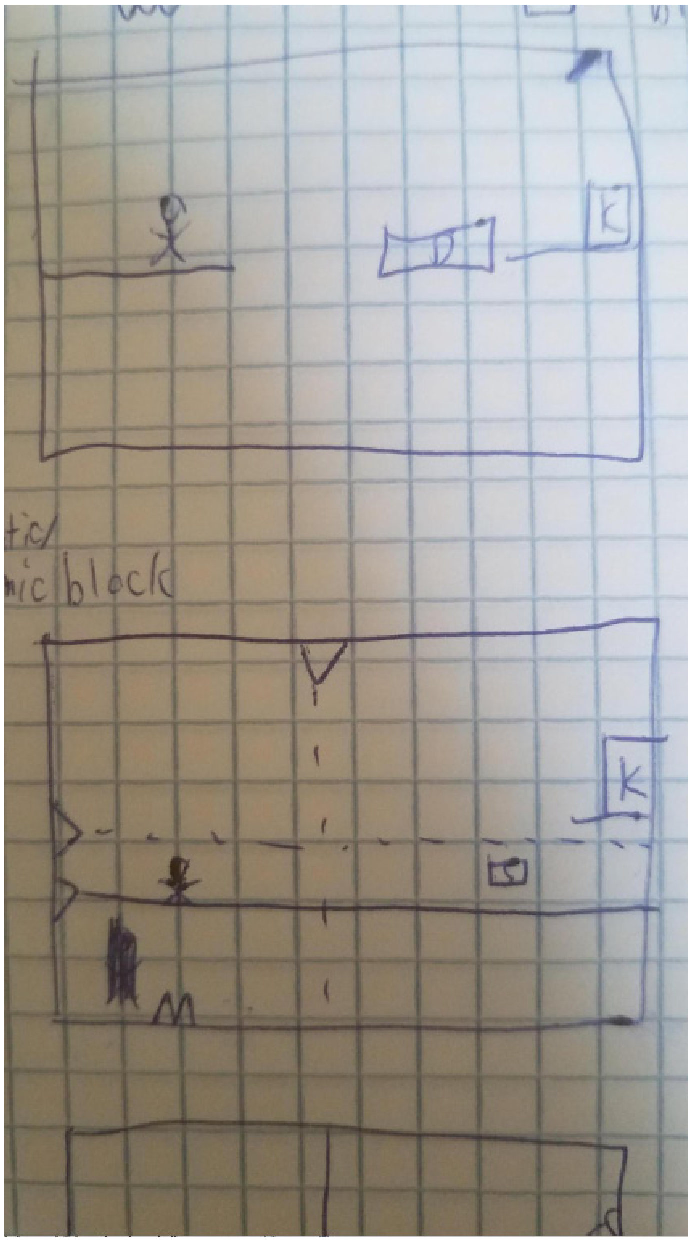
- Birds unable to lift their own weight
- live in their own crud, sheds uncleaned
- weak chicks thrown into black garbage bags
- egg shell computers

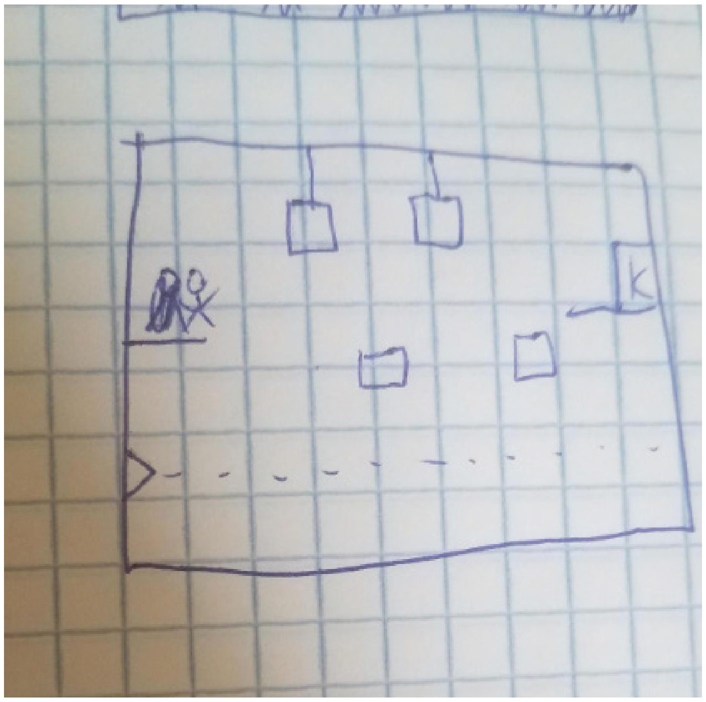

G

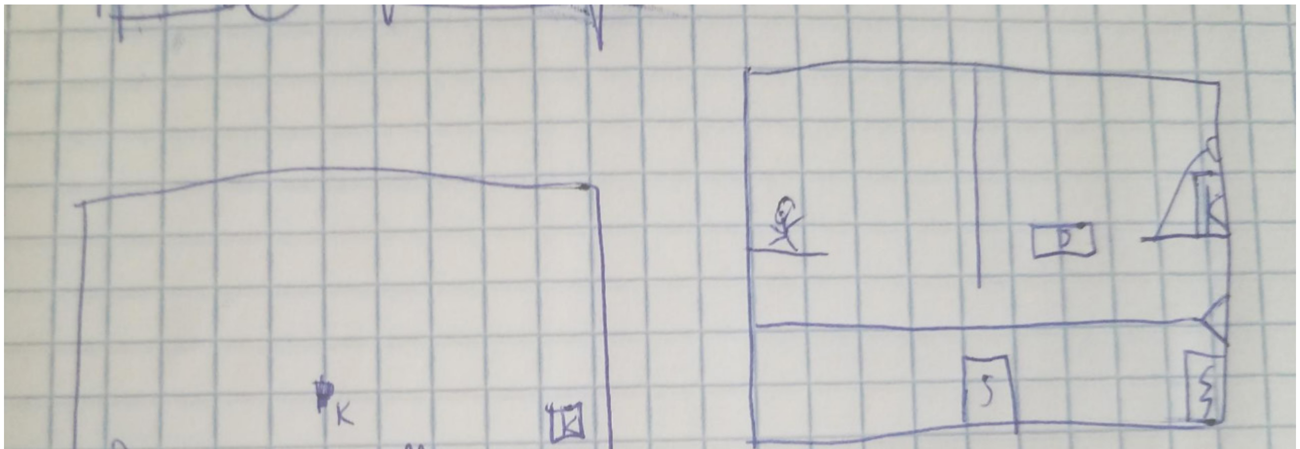

G

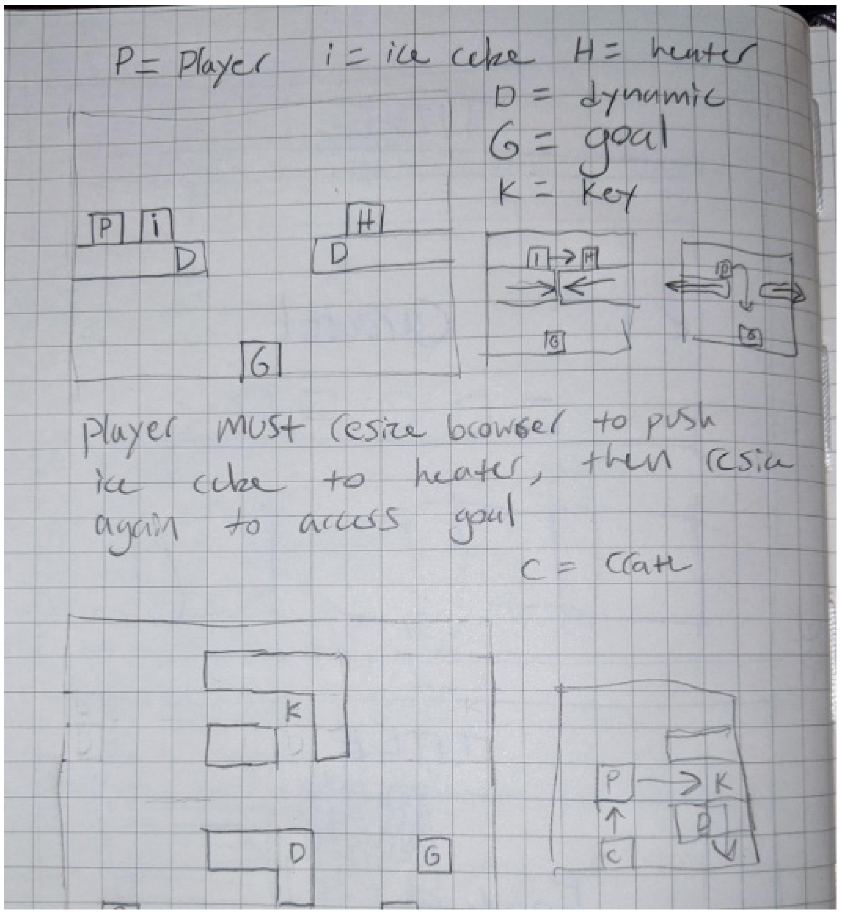

G


Crate
G

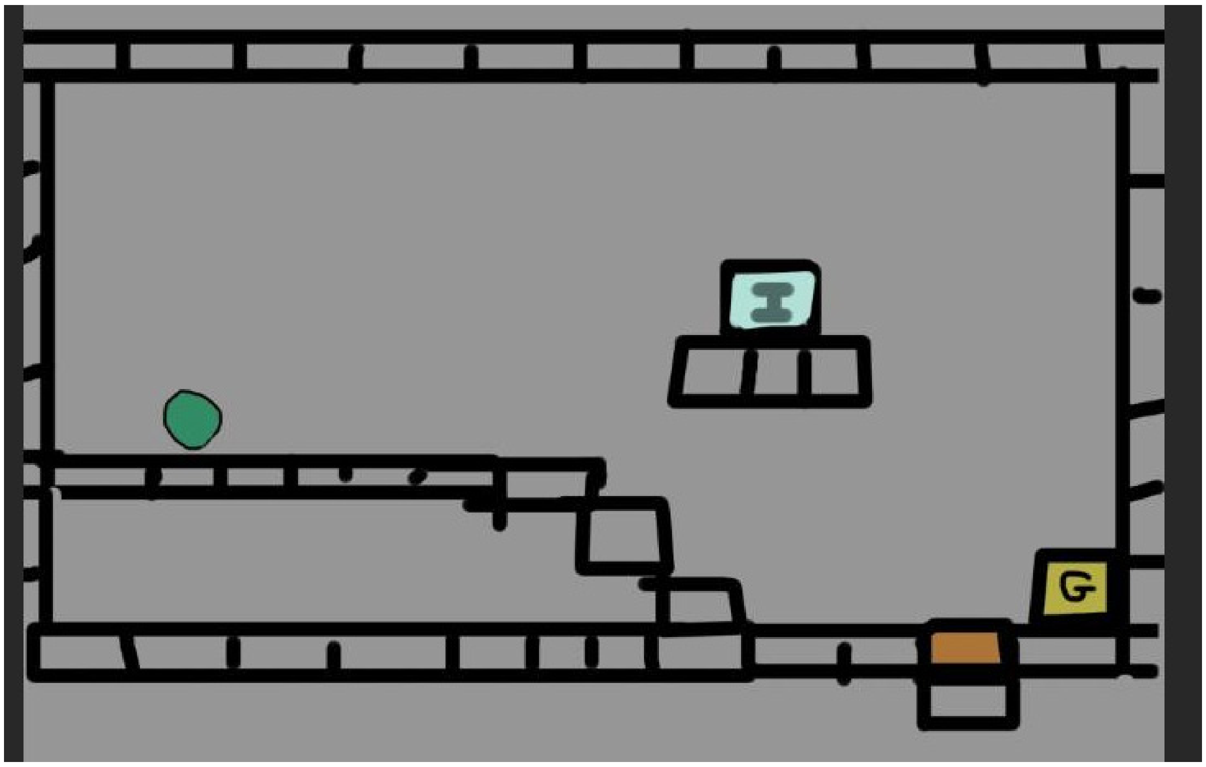

K
D
tic/
mic block
K
S

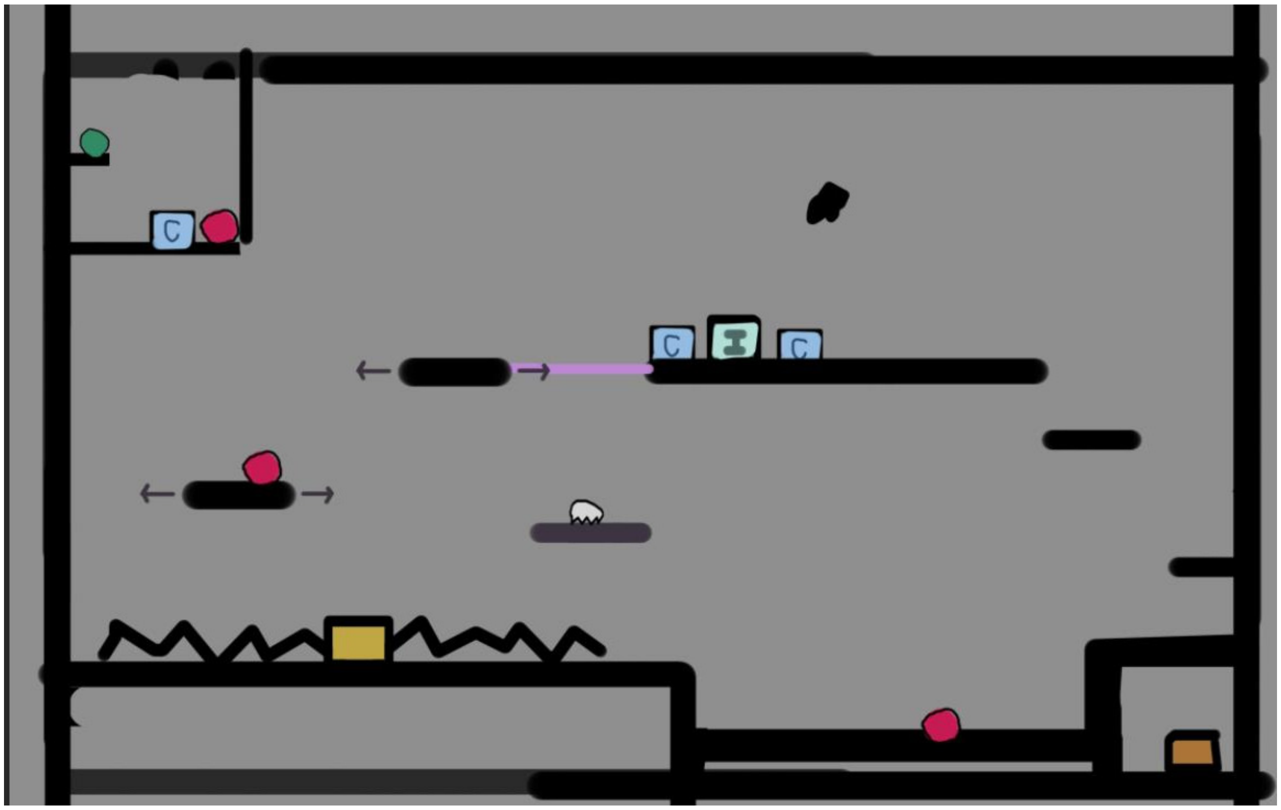

K

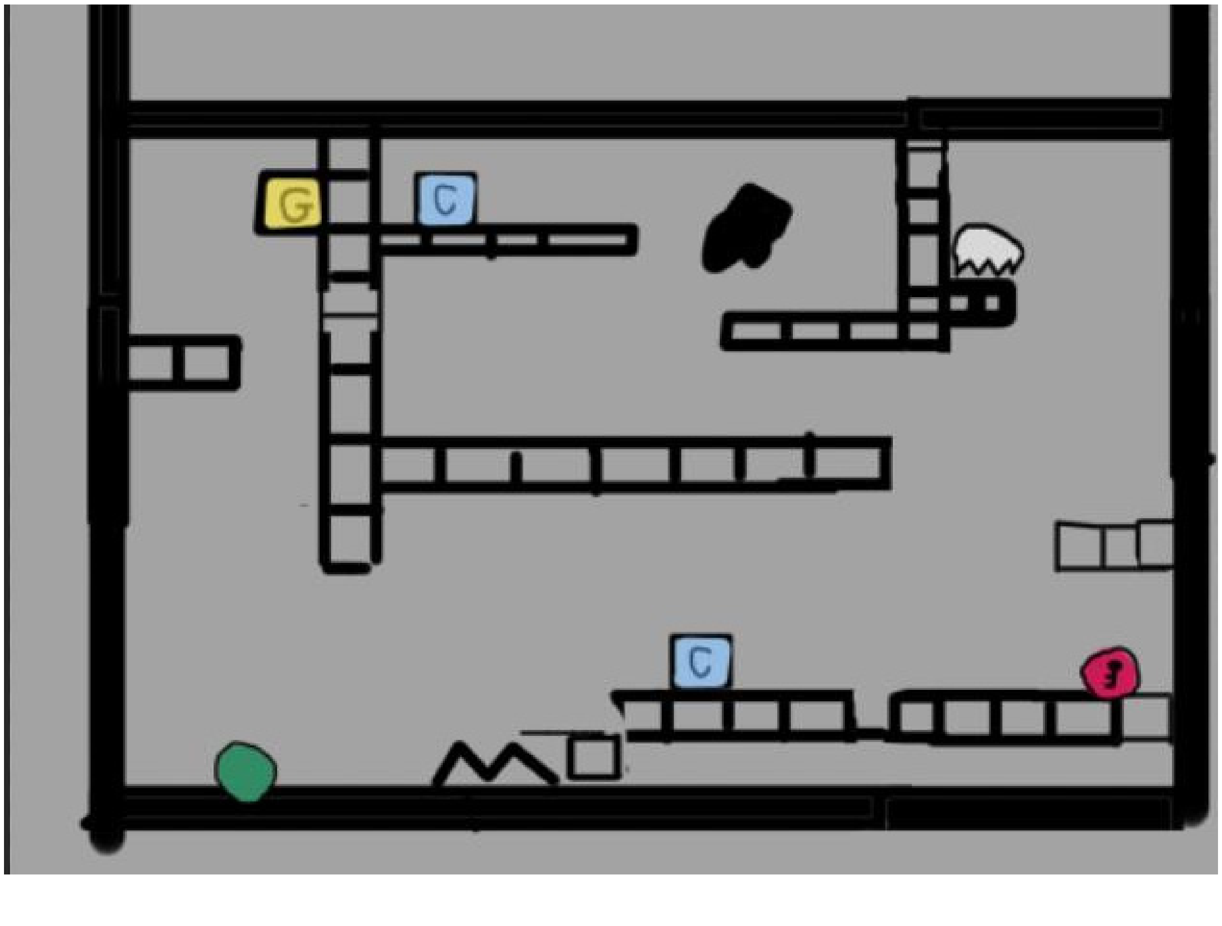

K
K
D
S

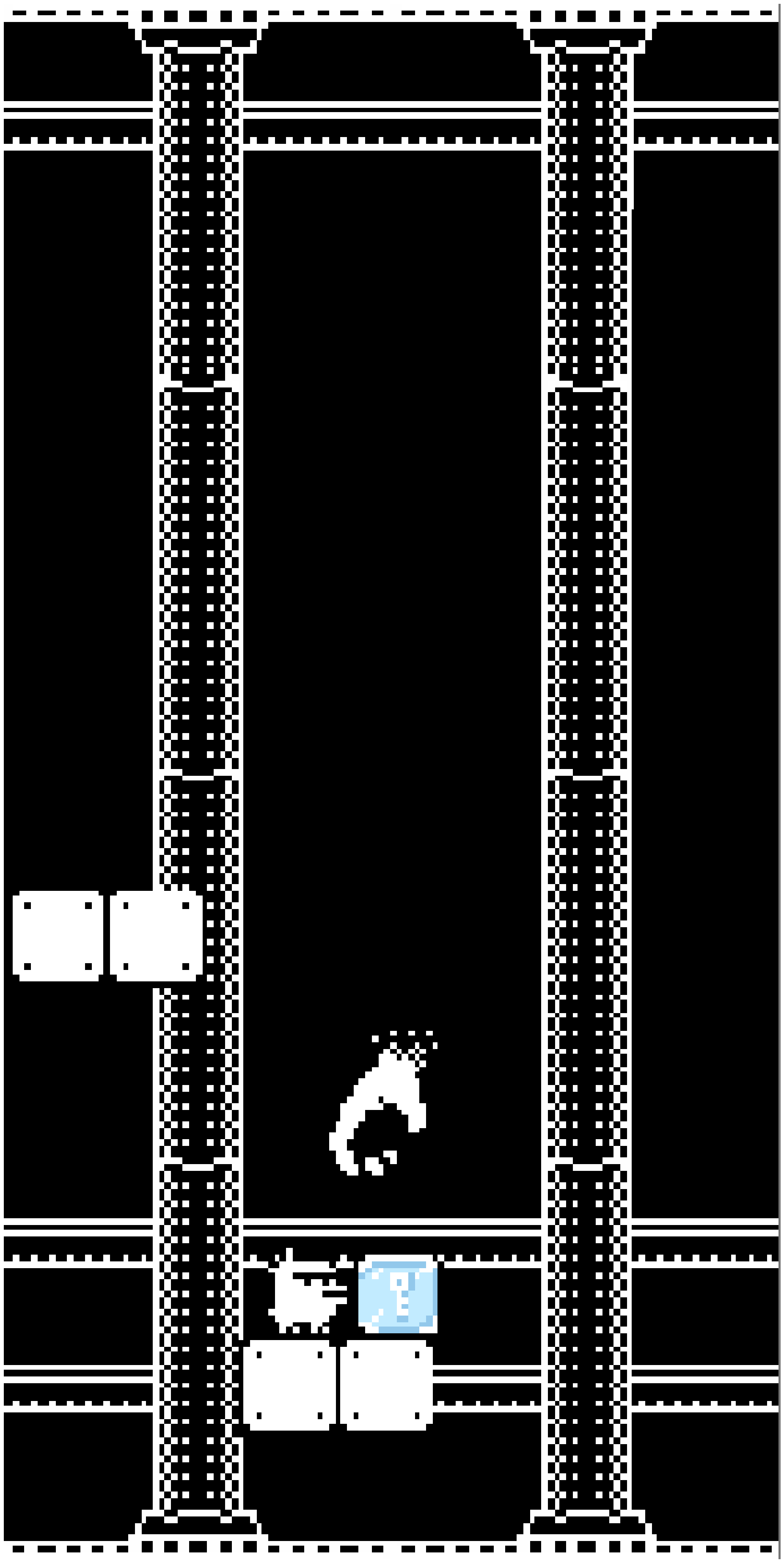


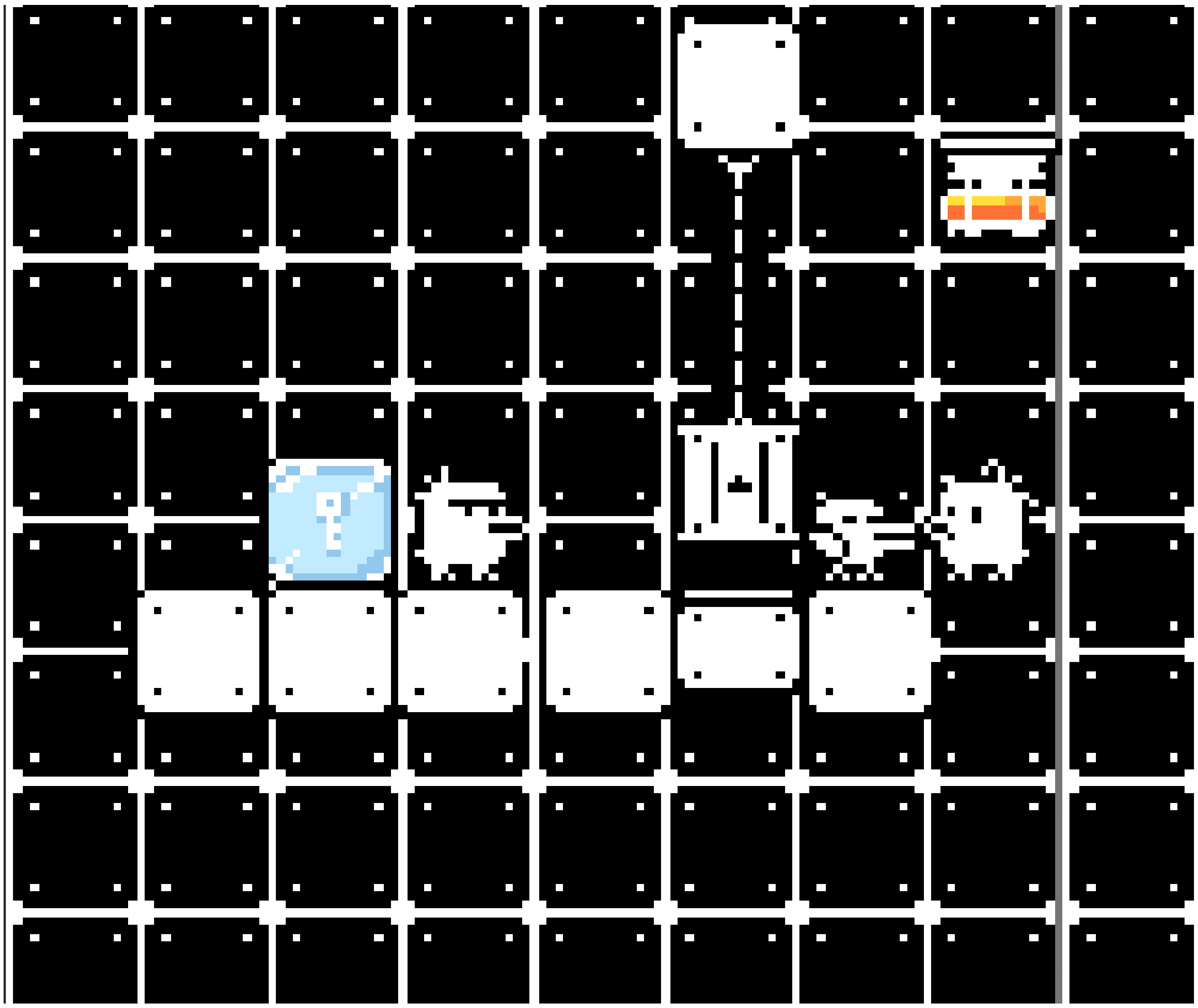
player must resize browser to push ice cube to heater, then resize again to access goal

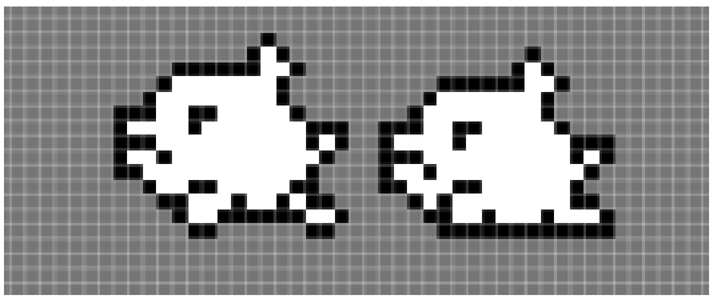



C
C
I
C

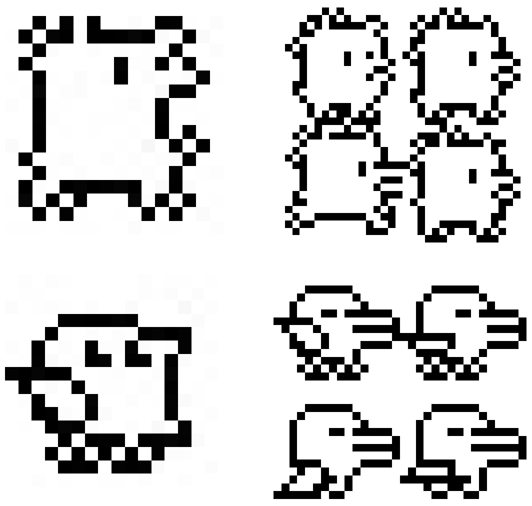
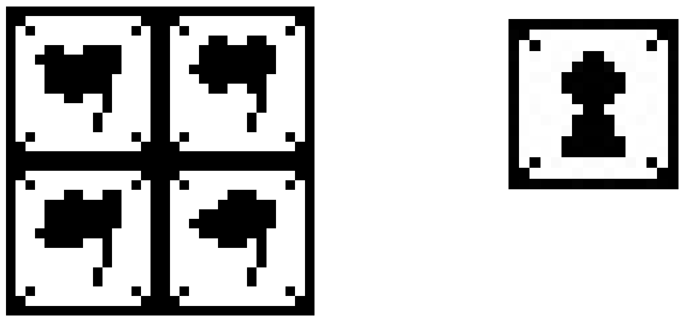
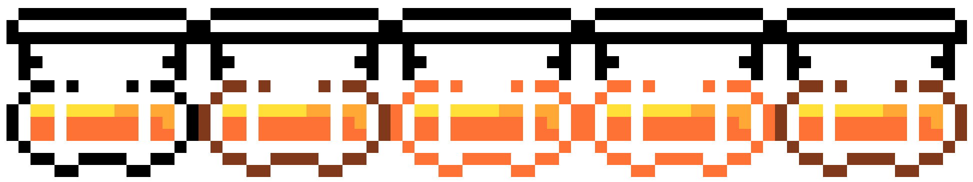
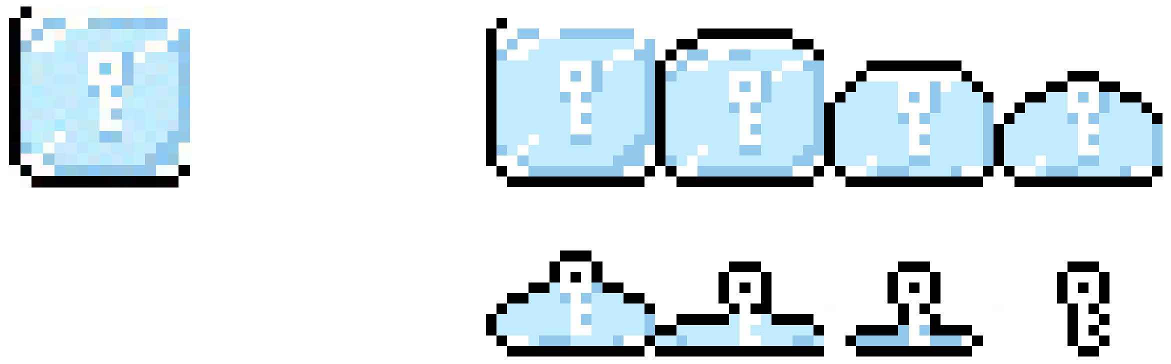

G
C
C

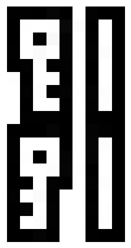


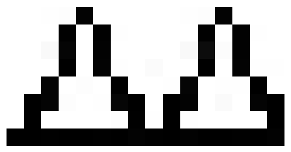


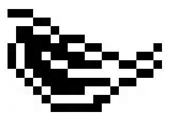


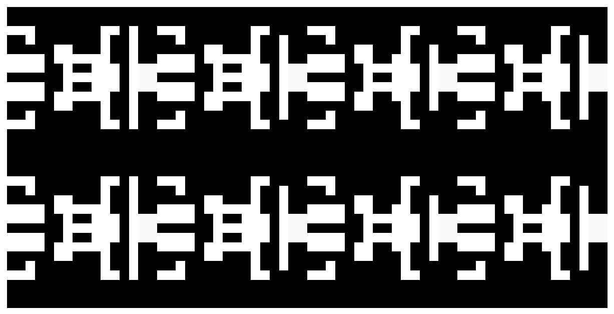
Sprites:

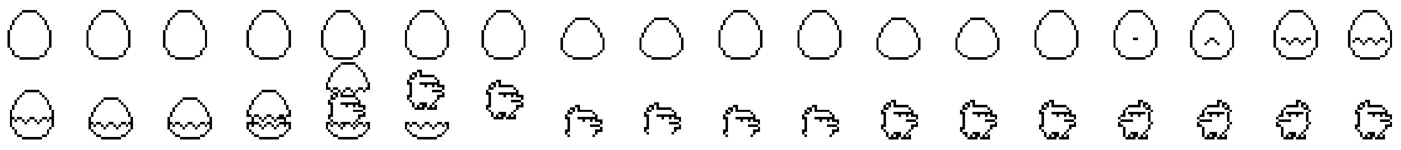





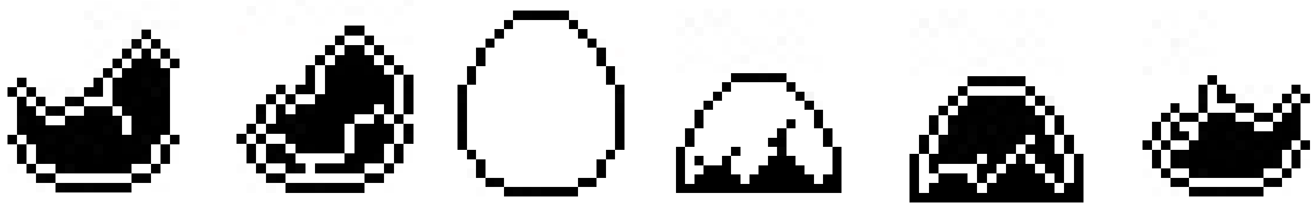


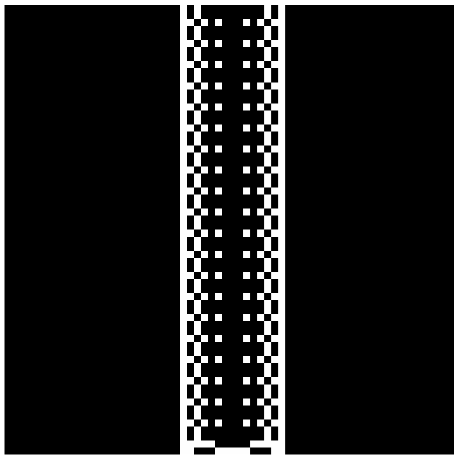


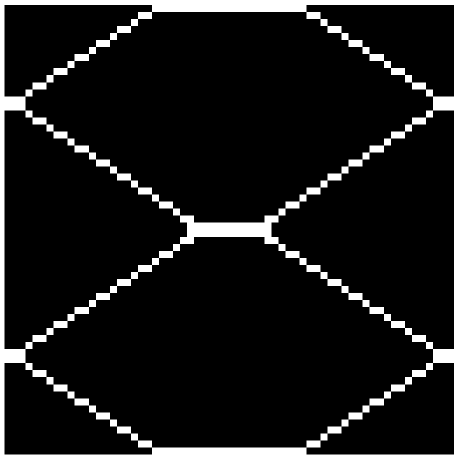


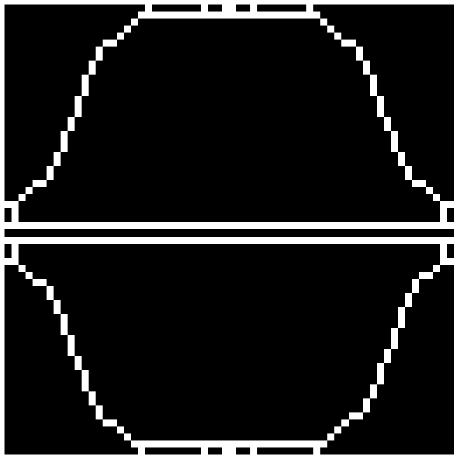


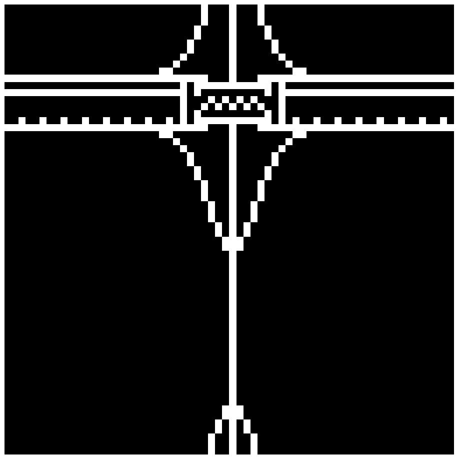


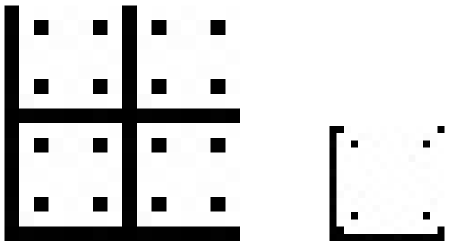




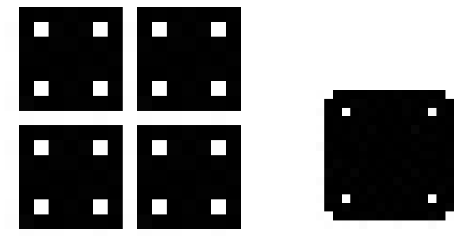






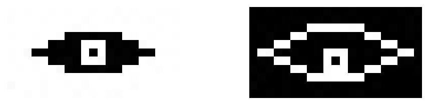


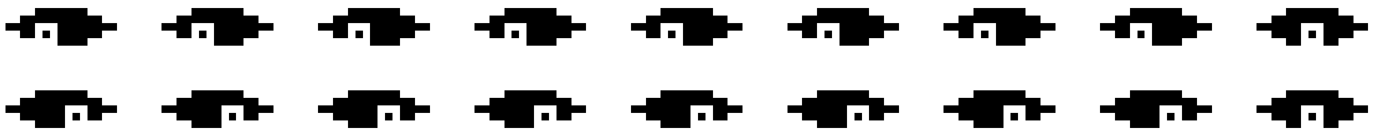
KEJI

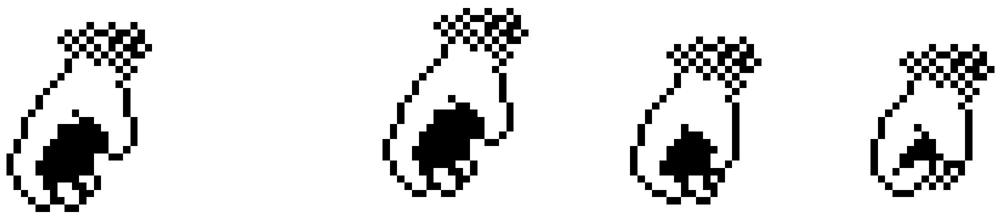


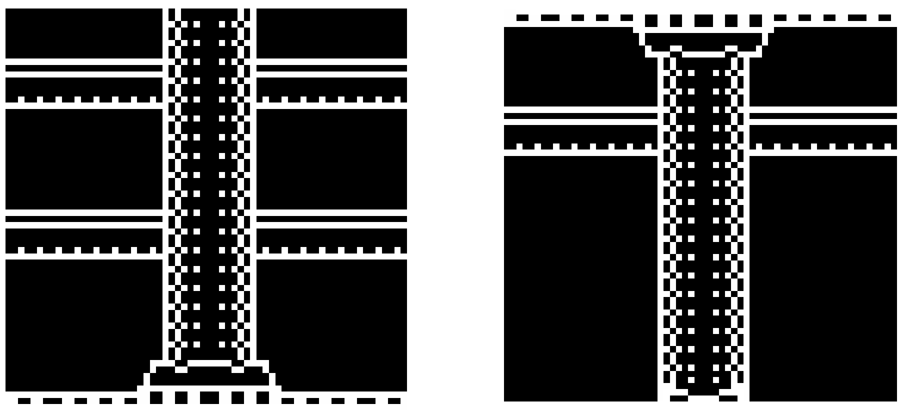
you win!

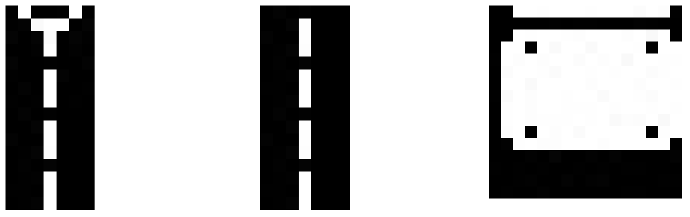
Unused Sprites:

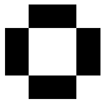


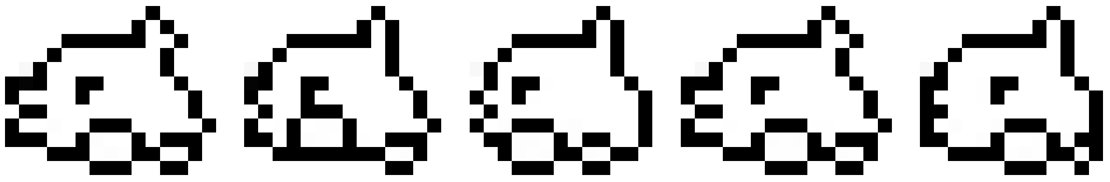




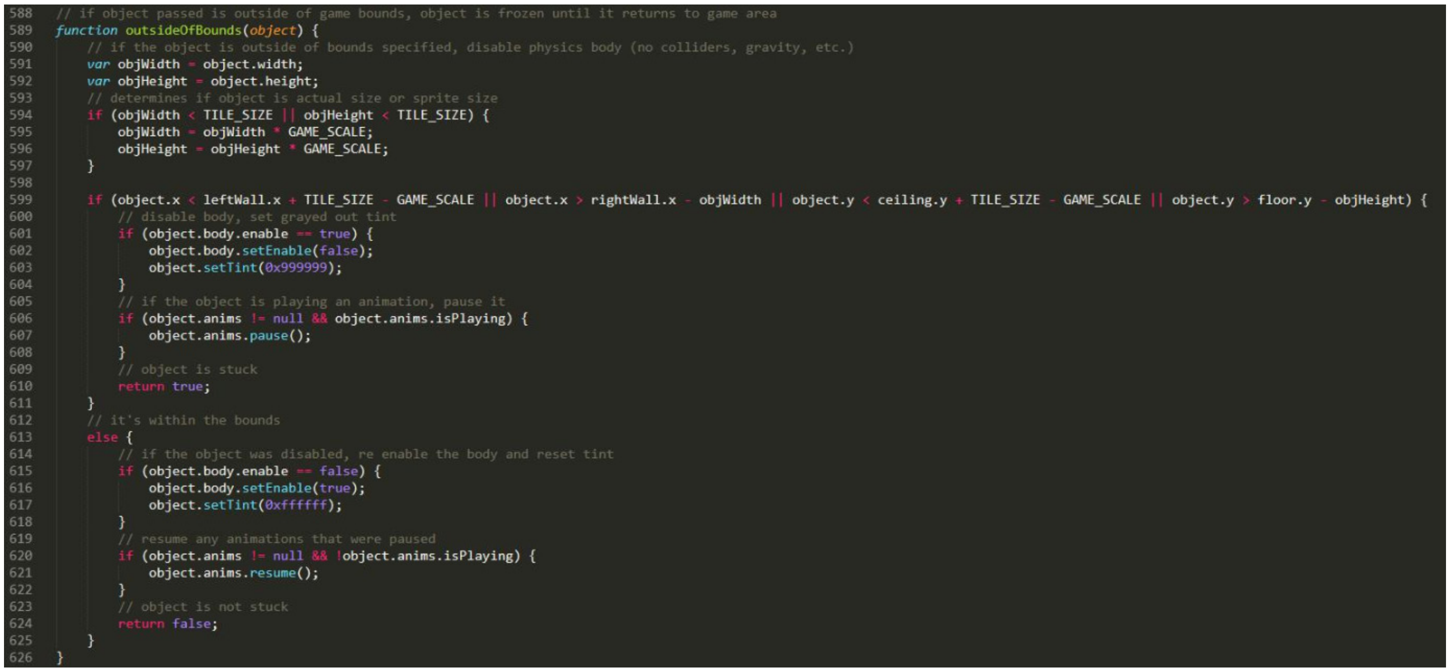




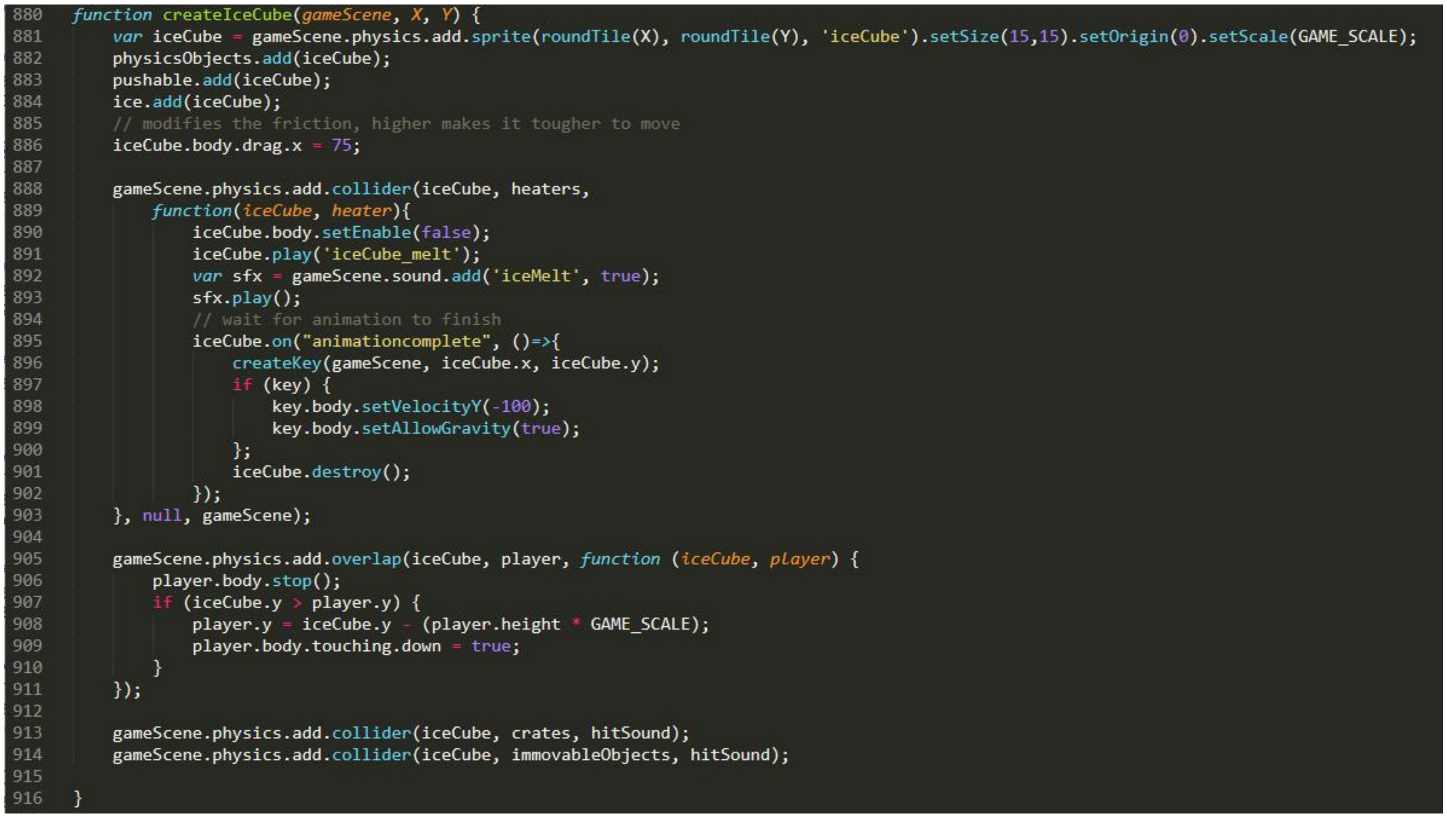




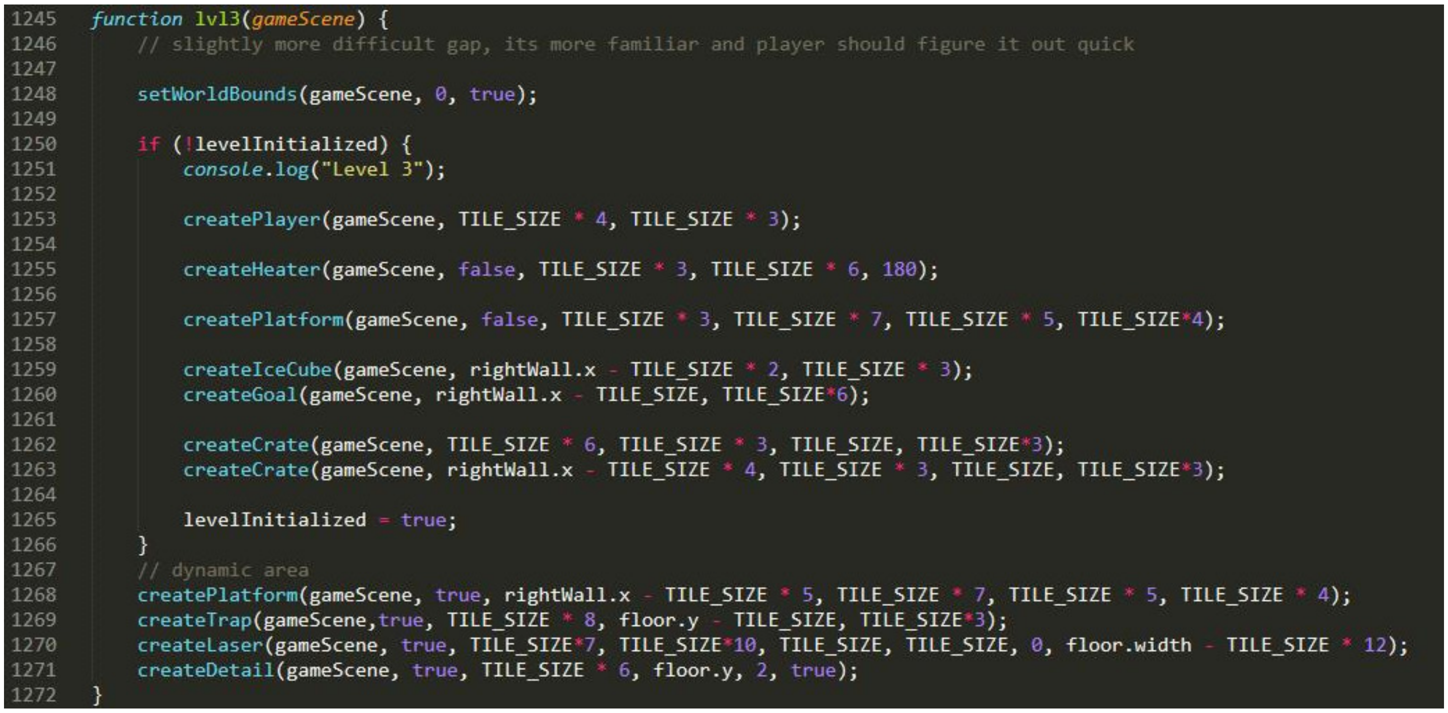
Code:



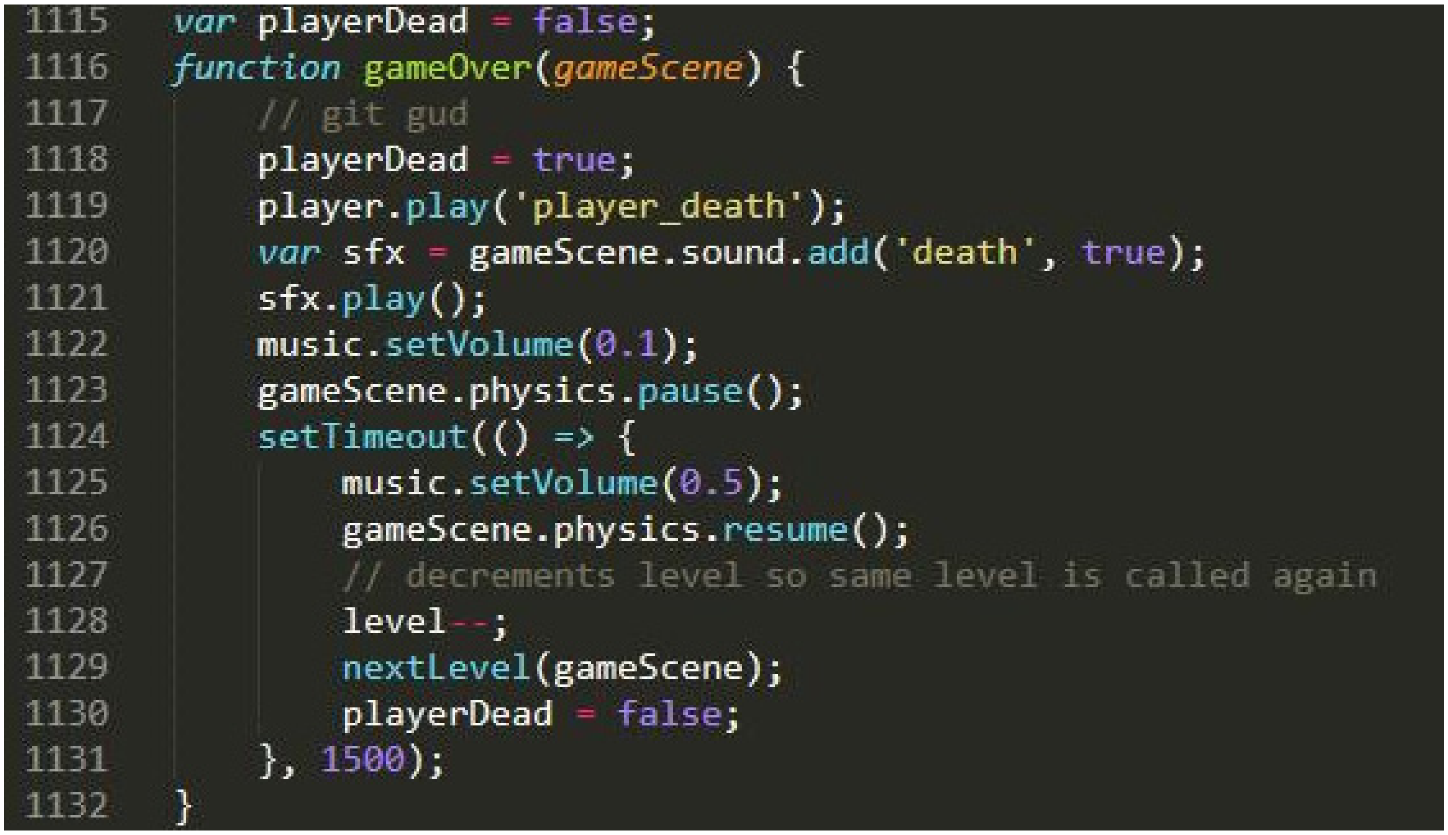
```javascript
// if object passed is outside of game bounds, object is frozen until it returns to game area
function outsideOfBounds(object) {
    // if the object is outside of bounds specified, disable physics body (no colliders, gravity, etc.)
    var objWidth = object.width;
    var objHeight = object.height;
    // determines if object is actual size or sprite size
    if (objWidth < TILE_SIZE || objHeight < TILE_SIZE) {
        objWidth = objWidth * GAME_SCALE;
        objHeight = objHeight * GAME_SCALE;
    }

    if (object.x < leftWall.x + TILE_SIZE - GAME_SCALE || object.x > rightWall.x - objWidth || object.y < ceiling.y + TILE_SIZE - GAME_SCALE || object.y > floor.y - objHeight) {
        // disable body, set grayed out tint
        if (object.body.enable == true) {
            object.body.setEnable(false);
            object.setTint(0x999999);
        }
        // if the object is playing an animation, pause it
        if (object.anims != null && object.anims.isPlaying) {
            object.anims.pause();
        }
        // object is stuck
        return true;
    }
    // it's within the bounds
    else {
        // if the object was disabled, re enable the body and reset tint
        if (object.body.enable == false) {
            object.body.setEnable(true);
            object.setTint(0xffffff);
        }
        // resume any animations that were paused
        if (object.anims != null && !object.anims.isPlaying) {
            object.anims.resume();
        }
        // object is not stuck
        return false;
    }
}
```



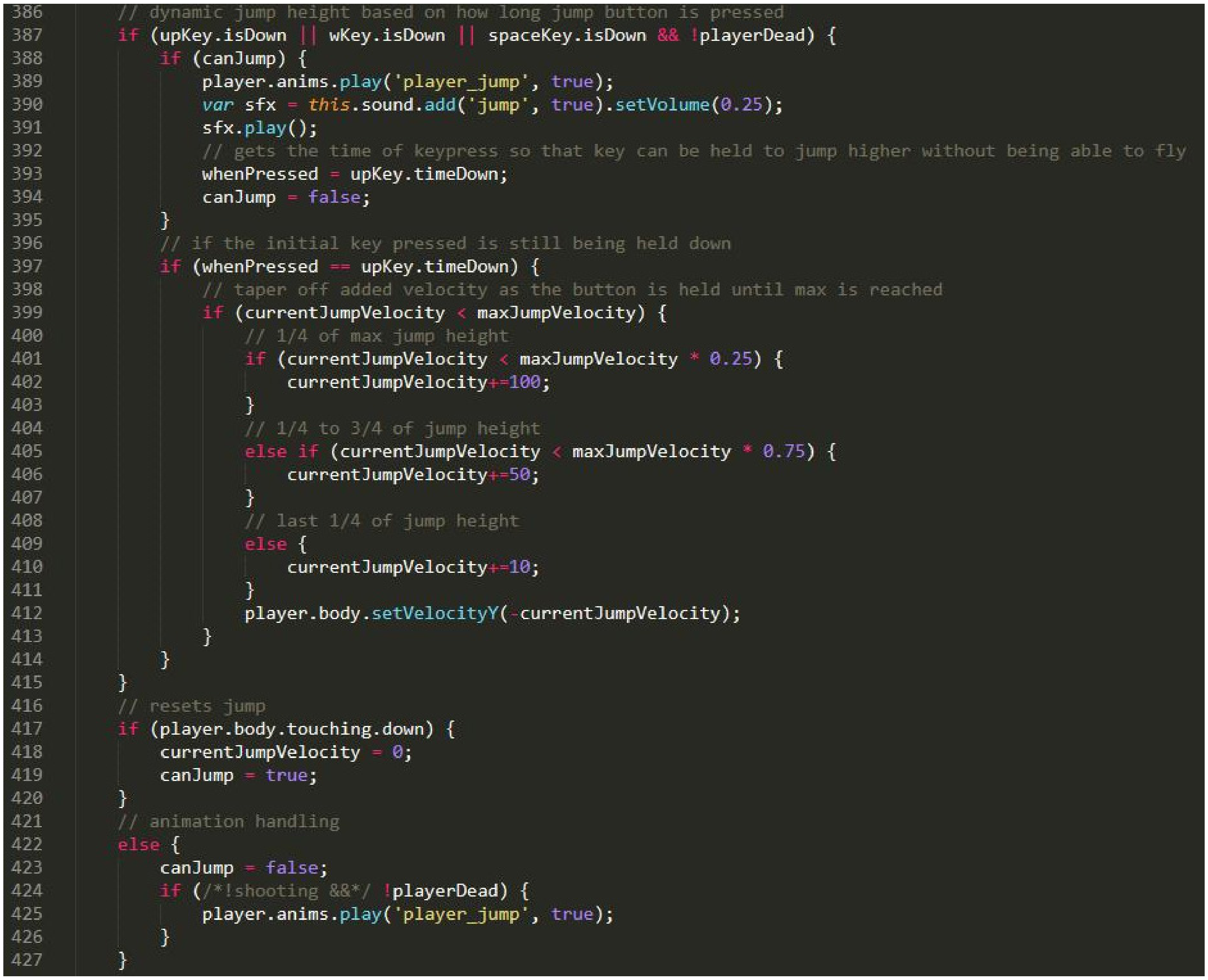
```javascript
function createIceCube(gameScene, X, Y) {
    var iceCube = gameScene.physics.add.sprite(roundTile(X), roundTile(Y), 'iceCube').setSize(15,15).setOrigin(0).setScale(GAME_SCALE);
    physicsObjects.add(iceCube);
    pushable.add(iceCube);
    ice.add(iceCube);
    // modifies the friction, higher makes it tougher to move
    iceCube.body.drag.x = 75;

    gameScene.physics.add.collider(iceCube, heaters,
        function(iceCube, heater){
            iceCube.body.setEnable(false);
            iceCube.play('iceCube_melt');
            var sfx = gameScene.sound.add('iceMelt', true);
            sfx.play();
            // wait for animation to finish
            iceCube.on("animationcomplete", ()=>{
                createKey(gameScene, iceCube.x, iceCube.y);
                if (key) {
                    key.body.setVelocityY(-100);
                    key.body.setAllowGravity(true);
                };
                iceCube.destroy();
            });
    }, null, gameScene);

    gameScene.physics.add.overlap(iceCube, player, function (iceCube, player) {
        player.body.stop();
        if (iceCube.y > player.y) {
            player.y = iceCube.y - (player.height * GAME_SCALE);
            player.body.touching.down = true;
        }
    });

    gameScene.physics.add.collider(iceCube, crates, hitSound);
    gameScene.physics.add.collider(iceCube, immovableObjects, hitSound);

}
```

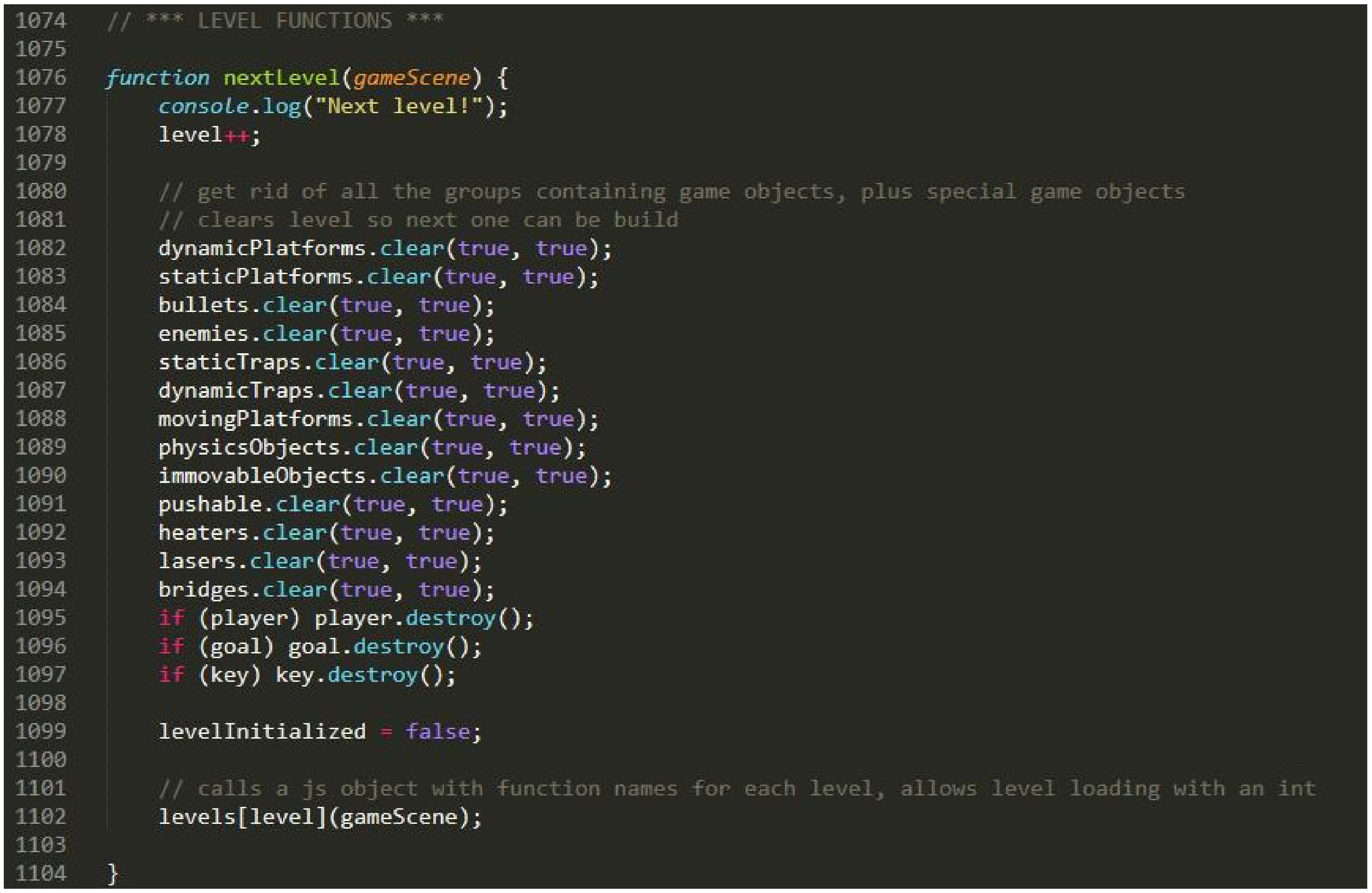


```
function lvl3(gameScene) {
    // slightly more difficult gap, its more familiar and player should figure it out quick

    setWorldBounds(gameScene, 0, true);

    if (!levelInitialized) {
        console.log("Level 3");

        createPlayer(gameScene, TILE_SIZE * 4, TILE_SIZE * 3);

        createHeater(gameScene, false, TILE_SIZE * 3, TILE_SIZE * 6, 180);

        createPlatform(gameScene, false, TILE_SIZE * 3, TILE_SIZE * 7, TILE_SIZE * 5, TILE_SIZE*4);

        createIceCube(gameScene, rightWall.x - TILE_SIZE * 2, TILE_SIZE * 3);
        createGoal(gameScene, rightWall.x - TILE_SIZE, TILE_SIZE*6);

        createCrate(gameScene, TILE_SIZE * 6, TILE_SIZE * 3, TILE_SIZE, TILE_SIZE*3);
        createCrate(gameScene, rightWall.x - TILE_SIZE * 4, TILE_SIZE * 3, TILE_SIZE, TILE_SIZE*3);

        levelInitialized = true;
    }
    // dynamic area
    createPlatform(gameScene, true, rightWall.x - TILE_SIZE * 5, TILE_SIZE * 7, TILE_SIZE * 5, TILE_SIZE * 4);
    createTrap(gameScene,true, TILE_SIZE * 8, floor.y - TILE_SIZE, TILE_SIZE*3);
    createLaser(gameScene, true, TILE_SIZE*7, TILE_SIZE*10, TILE_SIZE, TILE_SIZE, 0, floor.width - TILE_SIZE * 12);
    createDetail(gameScene, true, TILE_SIZE * 6, floor.y, 2, true);
}
```

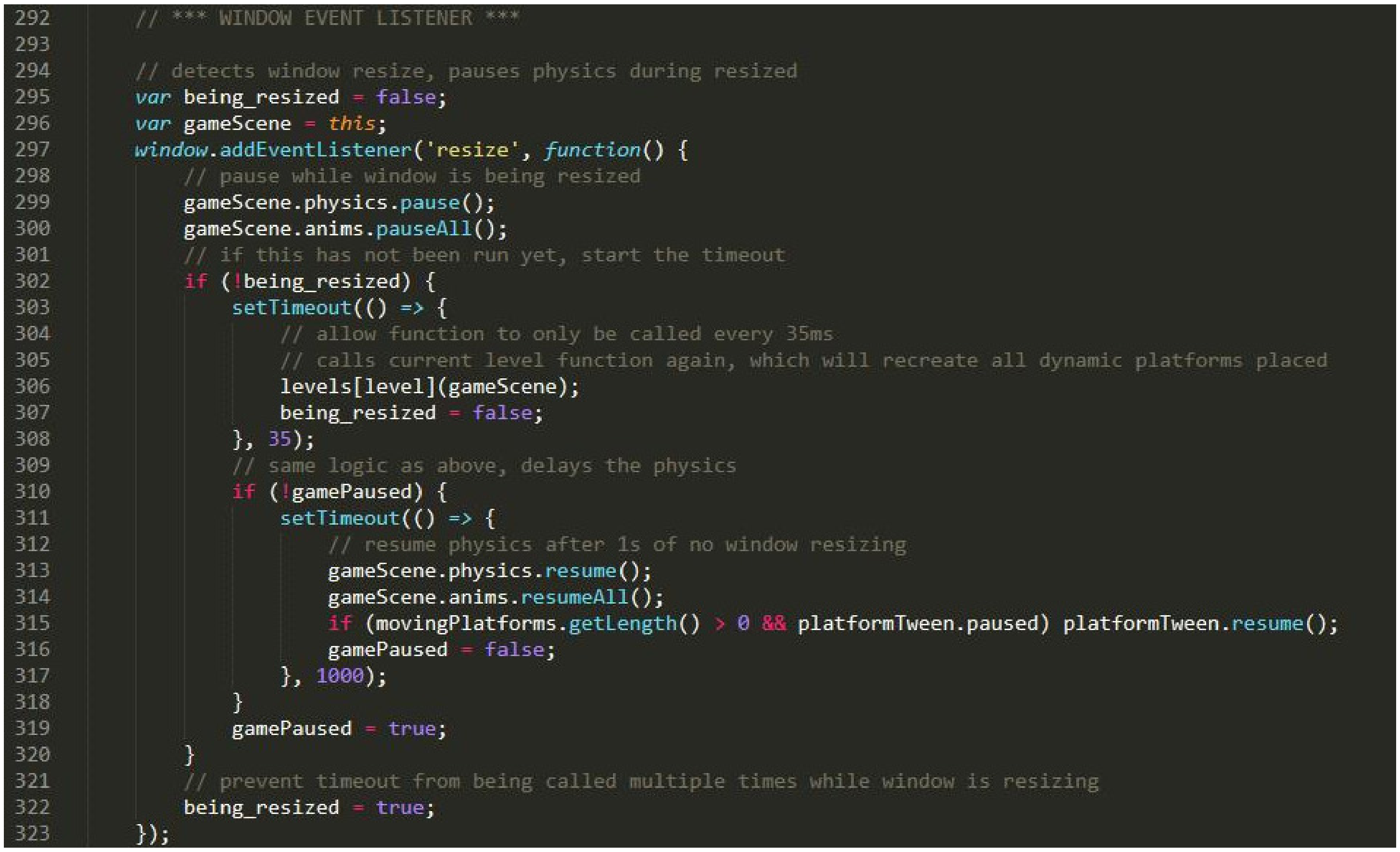


```
var playerDead = false;
function gameOver(gameScene) {
    // git gud
    playerDead = true;
    player.play('player_death');
    var sfx = gameScene.sound.add('death', true);
    sfx.play();
    music.setVolume(0.1);
    gameScene.physics.pause();
    setTimeout(() => {
        music.setVolume(0.5);
        gameScene.physics.resume();
        // decrements level so same level is called again
        level--;
        nextLevel(gameScene);
        playerDead = false;
    }, 1500);
}
```

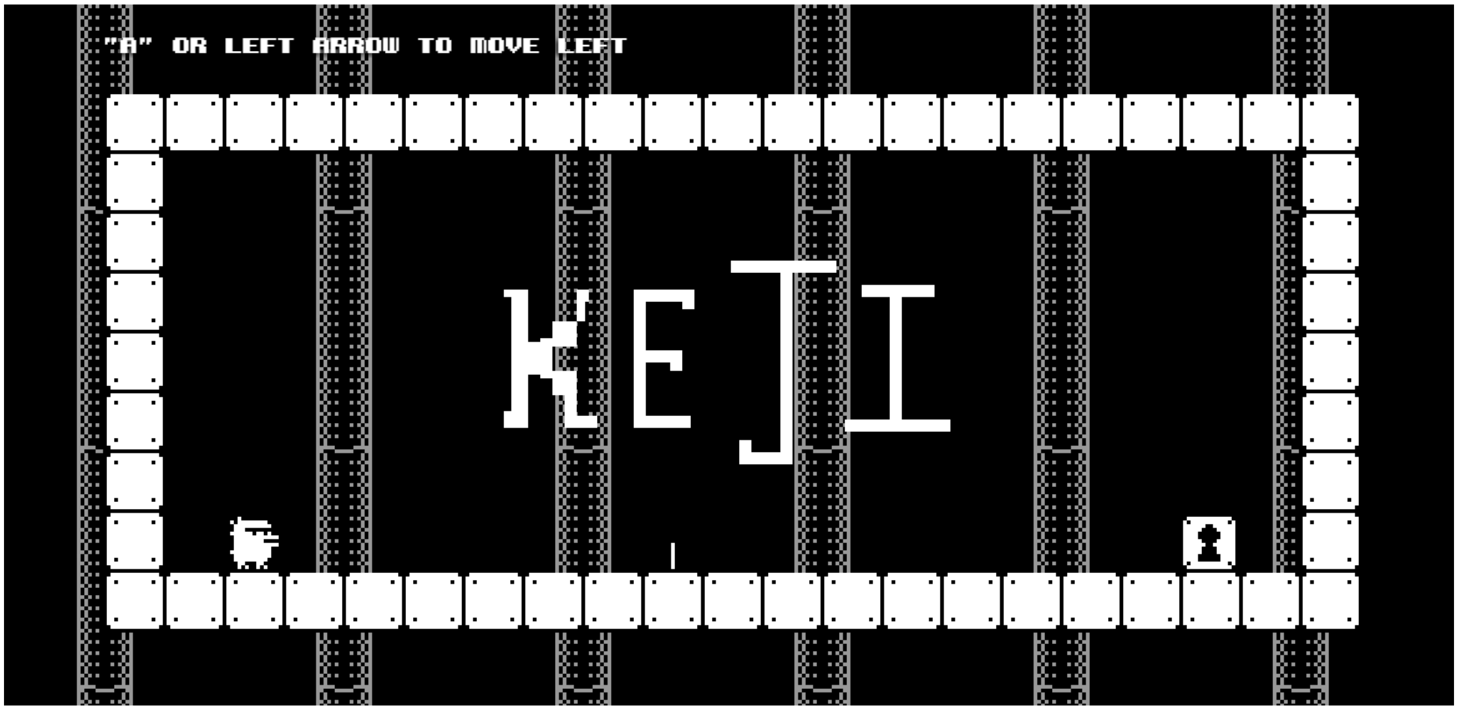

```
    // dynamic jump height based on how long jump button is pressed
    if (upKey.isDown || wKey.isDown || spaceKey.isDown && !playerDead) {
        if (canJump) {
            player.anims.play('player_jump', true);
            var sfx = this.sound.add('jump', true).setVolume(0.25);
            sfx.play();
            // gets the time of keypress so that key can be held to jump higher without being able to fly
            whenPressed = upKey.timeDown;
            canJump = false;
        }
        // if the initial key pressed is still being held down
        if (whenPressed == upKey.timeDown) {
            // taper off added velocity as the button is held until max is reached
            if (currentJumpVelocity < maxJumpVelocity) {
                // 1/4 of max jump height
                if (currentJumpVelocity < maxJumpVelocity * 0.25) {
                    currentJumpVelocity+=100;
                }
                // 1/4 to 3/4 of jump height
                else if (currentJumpVelocity < maxJumpVelocity * 0.75) {
                    currentJumpVelocity+=50;
                }
                // last 1/4 of jump height
                else {
                    currentJumpVelocity+=10;
                }
                player.body.setVelocityY(-currentJumpVelocity);
            }
        }
    }
    // resets jump
    if (player.body.touching.down) {
        currentJumpVelocity = 0;
        canJump = true;
    }
    // animation handling
    else {
        canJump = false;
        if (/*!shooting &&*/ !playerDead) {
            player.anims.play('player_jump', true);
        }
    }
```

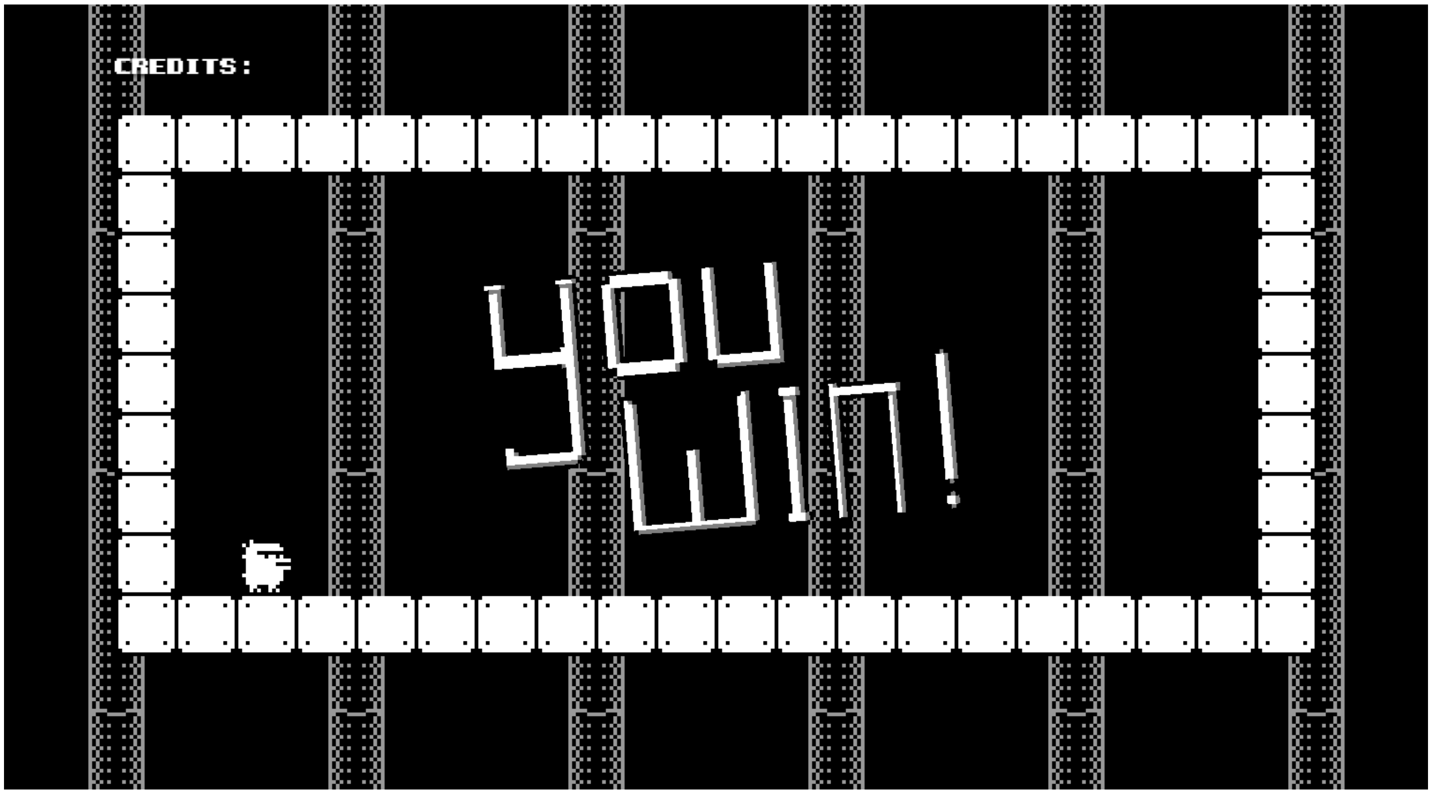


```
// *** LEVEL FUNCTIONS ***

function nextLevel(gameScene) {
    console.log("Next level!");
    level++;

    // get rid of all the groups containing game objects, plus special game objects
    // clears level so next one can be build
    dynamicPlatforms.clear(true, true);
    staticPlatforms.clear(true, true);
    bullets.clear(true, true);
    enemies.clear(true, true);
    staticTraps.clear(true, true);
    dynamicTraps.clear(true, true);
    movingPlatforms.clear(true, true);
    physicsObjects.clear(true, true);
    immovableObjects.clear(true, true);
    pushable.clear(true, true);
    heaters.clear(true, true);
    lasers.clear(true, true);
    bridges.clear(true, true);
    if (player) player.destroy();
    if (goal) goal.destroy();
    if (key) key.destroy();

    levelInitialized = false;

    // calls a js object with function names for each level, allows level loading with an int
    levels[level](gameScene);

}
```



```
// *** WINDOW EVENT LISTENER ***

// detects window resize, pauses physics during resized
var being_resized = false;
var gameScene = this;
window.addEventListener('resize', function() {
    // pause while window is being resized
    gameScene.physics.pause();
    gameScene.anims.pauseAll();
    // if this has not been run yet, start the timeout
    if (!being_resized) {
        setTimeout(() => {
            // allow function to only be called every 35ms
            // calls current level function again, which will recreate all dynamic platforms placed
            levels[level](gameScene);
            being_resized = false;
        }, 35);
        // same logic as above, delays the physics
        if (!gamePaused) {
            setTimeout(() => {
                // resume physics after 1s of no window resizing
                gameScene.physics.resume();
                gameScene.anims.resumeAll();
                if (movingPlatforms.getLength() > 0 && platformTween.paused) platformTween.resume();
                gamePaused = false;
            }, 1000);
        }
        gamePaused = true;
    }
    // prevent timeout from being called multiple times while window is resizing
    being_resized = true;
});
```


"A" OR LEFT ARROW TO MOVE LEFT
KEJI


CREDITS:
YOU WIN!