# Xiaoyu Yan

# c4yanxia

# 1001670293

1. Object
   1. Bug:
      * 1. x: x coordinate.
        2. y: y coordinate.
        3. counter: calculate fade time.
        4. size: bugs size.
        5. live: if live=false, bug stops moving and begins to fade out.
        6. Target: object to save the closest food’s info.
        7. rad: the angle toward target.
        8. score: bugs value.
        9. alpha: bug’s alpha value, used to control fade out.
        10. img: bug’s image.
        11. defColor(): randomly decide new bug’s color.
        12. search(): function to search nearest food.
        13. update(): calculate bug’s new coordinate, speed, moving angle.
        14. render(): draw bug in canvas.
   2. Target:
      * 1. x: bug closest food’s x coordinate.
        2. y: bug closest food’s y coordinate.
        3. dx: x coordinate distance between bug and target.
        4. dy: y coordinate distance between bug and target.
        5. cos: the cos value of the angle between bug and target.
        6. sin: the sin value of the angle between bug and target.
        7. dst: the distance between bug and target.
   3. Food:
      * 1. x: x coordinate, x = foodSize + Math.random() \* (canvas.width - 2 \* foodSize);
        2. y: y coordinate, y = Math.random() \* (canvas.height / 2) + canvas.height / 2 - foodSize; # y below 50% of the canvas height.
        3. size: food size.
        4. color: food color.
        5. render(): draw food in canvas.
   4. Global:
      * 1. canvas: the canvas element.
        2. btn: Play/Pause button
        3. TIME: Constant set the game counter.
        4. time: store game counter.
        5. clickRadius: click detection rang to kill bugs.
        6. state: game state: paly, pause, over, win.
        7. foodSize: constant to define foodsize.
        8. level: store current game’s level.
        9. score: store player’s score.
        10. fadeTime: constant to define bug fade out time.
        11. rdm : a random number between 1 to 3, to control randomly generate bugs time.
        12. foods[]: array to store foods in the game.
        13. bugs[]: to store active bugs in game.
        14. frame: frame variable to control frame.
        15. counter: to track game time.
        16. loop(): game loop
        17. changeState(): button function, to switch game state.
2. Data:
   1. local storage variable:
      1. scoreRecord: store the highest score the player ever get.
      2. level: which level the player selected.
      3. levelRecord: the highest level the player reached.
      4. score: store player’s score in a game.
3. Algorithm:
   1. Initialize Game:

function init() {

ctx.clearRect(0, 0, canvas.width, canvas.height);

time = TIME;

counter = 0;

score = 0;

state = "play";

rdm = Math.round(Math.random() \* 2 + 1);

bugs = [];

foods = [];

createFoods(5);

bugs = [new Bug()];

document.getElementById("debug").innerHTML = "local level: " + localStorage.getItem("level");

document.getElementById("state").innerHTML = state;

document.getElementById("score").innerHTML = "Score: " + score;

document.getElementById("currentLevel").innerHTML = "Level: " + level;

}

* 1. Game clock:  *counter+1* after every frame.

if (counter % 60 == 0) {

time--;

document.getElementById("counter").innerHTML = time + " sec";

}

* 1. 1-3 seconds create a new bug:

*rdm* is a random number between 1-3. Generate a new value every time a new bug enters the game.

// born new bug counter

if (counter % (60 \* rdm) == 0) {

rdm = Math.round(Math.random() \* 2 + 1);

counter = 0;

bugs.push(new Bug());

}

* 1. Random X coordination for new bug:

this.x = this.size + Math.random() \* (canvas.width - this.size);

* 1. Randomly generate foods

// Food position, random x and random y > canvas.height/2

this.x = foodSize + Math.random() \* (canvas.width - 2 \* foodSize);

this.y = Math.random() \* (canvas.height / 2) + canvas.height / 2 - foodSize;

* 1. search the closest food:

this.target.dst = canvas.width + canvas.height; // initialize the closest distance larger than any food.

dx = foods[i].x - this.x;

dy = foods[i].y - this.y;

dst = Math.sqrt(dx \* dx + dy \* dy); // calculate distance.

Using a for loop to find out the closest one.

* 1. Calculate rotation angle:

// calculate rotation angle.

// this.target.dx: food.x – bug.x.

// this.target.dy: food.y – bug.y

if (this.target.dx > 0 && this.target.dy > 0)

this.rad = Math.asin(this.target.sin);

else if (this.target.dx < 0 && this.target.dy > 0)

this.rad = Math.PI - Math.asin(this.target.sin);

else if (this.target.dx < 0 && this.target.dy < 0)

this.rad = Math.atan2(this.target.dy / this.target.dx) + Math.PI;

else if (this.target.dx > 0 && this.target.dy < 0)

this.rad = -Math.acos(this.target.cos);

* 1. Moving and speed:

this.x += this.speed \* Math.cos(this.rad);

this.y += this.speed \* Math.sin(this.rad);

* 1. Draw bugs:

Bug.prototype.render = function() {

// image

ctx.save();

ctx.globalAlpha = this.alpha;

ctx.translate(this.x, this.y);

ctx.rotate(this.rad);

ctx.drawImage(this.img, -this.img.width/2 , -this.img.height/2);

ctx.restore();

};

* 1. Get click position:

function *getPosition(e)*

var rect = canvas.getBoundingClientRect();

var x = e.clientX – rect.left;

var y = e.clientY – rect.top;

* 1. Kill bugs and fade out:

// check click

if (dst < clickRadius) {

// Update score

score += bugs[i].score;

bugs[i].score = 0;

document.getElementById("score").innerHTML = "Score: " + score;

// Kill bug

bugs[i].live = false;

}

// Fade out

bugs[j].counter++;

bugs[j].alpha = 1 - bugs[j].counter / fadeTime;

bugs[j].render();

if (bugs[j].counter == fadeTime)

bugs.splice(bugs[j], 1);

When a bug is clicked, set bug.live to false, and start to count time, the bug’s *alpha = 1 – counter/fadetime*. when *counter = fade time*, *alpha = 0,* delete the bug object.

* 1. Eat food:

// Eat food

if(foods[i].x == this.target.x && foods[i].y == this.target.y){

foods.splice(i, 1);

break;

}

when the bug.position = target.position delete the food object. bugs keep searching nearest food before updating position.

* 1. Game Over:

document.getElementById("state").innerHTML = state;

window.cancelAnimationFrame(frame);

if (score > localStorage.getItem("score"))

localStorage.setItem("score", score);

var restart = confirm("Game Over!\n" + "Your score: " + score + "!\n New Game?" + " bugs: " + bugs.length);

if (!restart) {

document.location.href = "start.html";

} else {

init();

loop();

}

* 1. Game Win: chose go to next level or exit.

// Record new level record.

if (level + 1 > localStorage.getItem("levelRecord"))

localStorage.setItem("levelRecord", level + 1);

if (level < 2) {

var next = confirm("You Win!\n" + "Next Level?");

if (next) {

init();

level++;

document.getElementById("currentLevel").innerHTML = "Level: " + level;

} else {

document.location.href = "start.html";

}

} else {

alert("You Finished the Game!");

document.location.href = "start.html";

}

* 1. Play/Pause switch

btn.addEventListener('click', function changeState() {

if (state == "pause") {

state = "play";

loop();

} else if (state == "play") {

state = "pause";

}

document.getElementById("state").innerHTML = state;

document.getElementById("debug").innerHTML = state;

}, false);

1. Challenging:
   1. calculating bugs rotating angle.
   2. delete bug and food in proper time.
   3. switch between different state.
   4. Concurrency control.
2. Test:
   1. Open *start.html*, level 2 is unavailable.
   2. change to mobile mod, start page looks fine.
   3. change back to desktop mod.
   4. select level 1, click *start* button.
   5. counter starts to count down from 60s.
   6. click far away bugs, nothing happen.
   7. click on bug, bug fades out in 2s, and score increased.
   8. 1-3s a new bug entered from a random x coordinate.
   9. bugs moved toward closest food.
   10. food ate by bugs disappeared and bugs moved toward a new target.
   11. let bugs eat all foods, game over, a pop-up appears and select *ok* to play again.
   12. Game is initialized in the same level.
   13. win the game, a pop-up appears, choose ok to go to next level.
   14. Game level becomes level 2, bugs moving in a faster speed.
   15. win the game, no higher level available, go back to *start page*.
   16. level 2 is available now, high score becomes the highest score of last game.
   17. start game, get a lower score, lose game, go back to start page, high score doesn’t change.
   18. start game, get a new highest score, go back to start page, high score becomes the new score.