Bug Smush 20XX

Ryan Fung - c2fungry, 999747706
Matthew Faenza - g4faenza, 998507316

Game Rules

For 60 seconds, keep at least one of the five food bits intact from the bugs. Bugs may be killed with a click or tap, on or around them for a small (30 pixel) radius. Any bug within the hitbox will be eradicated. If the time runs out, you win.

Challenges

- Getting a working animation was challenging as we didn't know how to constantly update the process of drawing on the canvas.
- Rotation of the canvas would rotate the full canvas including all other fruits and bug objects. This proved a problem without the save and restoration of canvas states.
- proper movement of the bugs, originally we had bug movement that would rapidly move and direct itself towards the exact x or y coordinate of its target, instead of taking the direct shortest path.
- creating a proper hitbox for bug detection upon click, creating proper values for the hitbox and checking if bugs were within the range, proved to be a nuisance. This is because originally, the coordinates of the bugs to be affected would be defaulted on the top left of the bug image instead of the center.

Game Features

- Two difficulty levels, with faster bugs and more points per bug
- High score and Current score system
- 3 unique bugs with their own speeds, points, and different probability of encounter
- Functional Pause, Resume, and game timer
- Functional hitboxes

Data Design (objects, attributes, their relationships, why we used what structure)

Classes:

Bug: Implemented within the Bug function, and initialized within spawnBug. Within the Bug object we store an x and y position via parameters passing into the function, with the other attributes set conditionally depending on the third attribute, bugProbability, which determines our bug type. Each type has its own unique colour and image, as well as two unique speeds for each difficulty level. To initialize the bug's first (and hopefully only) target we call the shortestDistance function with its x and y position as parameters. We also store the initial angle from the bug's current position to its target to allow for a rotation towards the food bit to begin.

Food Bit: Implemented within the FoodNode function, and initialized with spawnFood. The FoodNode object stores an x and y position given as parameters, as well as its index as the current length of the foodBits array (since it will be tail-inserted). We also store hitbox information via a left, right, top and bottom attribute.

Data Variables:

Score and hscore: Score and hscore are both global integer variables that are updated on kills, and on a game win or game loss, respectively.

seconds: used to store the default value and the decremented values for display on the countdown timer.

countD: Holds the interval timer that repeatedly calls the timerCount function every 1 second.

foodBits: An array that holds the existing food objects.

swarm: An array that holds the existing bug objects.

foodSpawnCount: A simple global integer variable that is used to initialize the number of food objects.

failedSpawn: A global integer variable that keeps track of how many failed attempts in a row of bugSpawn have been called.

bugSpawner: Holds the interval timer that repeatedly calls the bugSpawn function every 1 second.

MyReq: Hold the value of the requestAnimationFrame(update) function call.

restart: A sessionStorage element that keeps track of whether the “OK” button was clicked on the end game pop-up.

level: A global integer variable that keeps track of the value of which radio button was selected on the start screen.

updater: Holds the interval timer that repeatedly calls the MyReq variable every 1000/60 seconds, or more specifically 60 frames per second.

Design Elements

HTML

Beginning with the header, we included our CSS and JavaScript externally to keep up good practices regarding clean HTML and faster web pages (via the browser cache). Although for the purposes of this project it would not have caused significant issues to keep them within the HTML file, we chose to maintain best practices.

Within the body there are two divs which hold elements pertaining to the two screens of the game. The first div keeps track of an html form that holds two radio buttons for level choice and a button to start the game (resulting in a screen change) and a text element used to portray the high score. The second div, only visible during game play, is a simple info bar to keep track of the game's countdown, a pause/resume button, and a text element to keep track of the user's current score. Finally, there is a canvas element that makes up the viewable portion of the actual game.

CSS

The CSS modifies the body's general font-family and text-alignment. The two screens have had their heights and widths changed to match the specifications. However, the gameScreen canvas and its elements have their display ~~changed~~ set to 'none' until the game begins. This was to prevent any invisible blocks from interfering with the startup screen. Elements within the infoBar simply have their positions floated left to ensure a specific order of timer, pause/resume button, and score.

JavaScript

For the game, we have initialized a large number global variables to manage each algorithm, such as arrays for our bug and food objects, and counts for the objects and game time. However, of particular importance is the event handler : 'window.onload = startClick'. This handler is ensures that the whole window is loaded before any functions can be called.

Algorithms

startClick

The startClick function maps out the actions needed for each button. The actions are received and forwarded by using event handlers for each clickable element on the screen. So clicking the "Start" button would start the game. Likewise, "Pause" would pause the game, and "Resume" would resume the progress of the game.

changeScreens

The changeScreens function acts as an initializer for many aspects of the game. The function first change the starting screen to a new game screen by changing the css of the html elements. It will continue to set the highscore and the level of the game through many value checks. Then the function would create 5 food objects and draw them on the canvas. Finally, it starts the game's countdown timer, the frame update.

timerCount

The timerCount function simply displays the countdown of the game and constantly updates it's decrementing value every second. In this function, each second will check if the game is finished and decrement the counter if it is not. If it detects that the game is finished, then it will update the highscore accordingly and create a prompt for the user to choose whether to restart or quit the game.

pause

The pause function simply stops all action on the current game. It freezes the bug spawning, the timer countdown, and the frame update.

resume

The resume function will undo anything that the pause button has halted. It will resume the progress of bug spawning, the timer count down, and the frame update.

spawnFood

The spawnFood function will create one food object on a random canvas position and append it to the foodBits array.

spawnBug

The spawnBug function will randomly create a bug object along the top of the canvas. The x position of the bug and its type are randomly chosen. After its coordinates are set, the bug utilizes an external image to draw on the canvas.

attack

The attack function is used whenever a mouseclick is detected on the canvas. It creates a hitbox by increasing the range of the mouseclick's x and y coordinates. Then the function will check if any bug object's coordinates are within the hitbox, deleting any bug that is found in the range.

update

The update function will renew the canvas so that new bugs can be drawn to accomodate for the change in bug movement. The function also checks if a bug has touched a food object and deletes the object accordingly. The update function is recursively called by the requestAnimationFrame function, in order to keep updating the changing variables of bugs.

shortestDistance

The shortestDistance function is called by bug objects to find the closest food object. This is done by using the pythagoras theorem to find the hypotenuse among all fruits. The smallest hypotenuse will represent the shortest path towards the closest food object.

checkRestart

The checkRestart function simply checks if the restart option was chosen on the end game screen. This is accomplished through the session storage of the restart value. If the restart option is chosen, then the game is reloaded and restarted using event handlers.