Interactive Game In JavaScript

132148

Multimedia Design & Applications

Computer Science

04/05/2017

1 Introduction

This report will demonstrate the implementation of an interactive game in JavaScript that makes use of HTML5 canvas to draw the graphics. This will be done by describing the functionality the game provides such as keyboard event detection, collision detection of objects, score/life systems and implementation of video. The game is a version of the Asteroids [1] arcade game.

2 Design and Development

2.1 Media Elements

All images are either SVG or PNG files, displayed using the context of the canvas' drawImage() method. All images/videos come from PixaBay, and do not require referencing of the original source. Image (and video) files are initialised in the html file, and then referenced in the JavaScript file using the html DOM.

2.2 Keyboard Events

Properties of the browser running the game can be accessed using the 'window' object. A window 'onload' method encapsulates all JavaScript code for the game, and runs it when the html file for the game is opened in the browser. Another window method, 'addEventListener', runs whenever a key is pressed or lifted (depending on what parameter the method is given). It then executes a function, which determines what key was pressed and acts accordingly.

2.3 Updating the Display

To update the graphical display, a function 'draw' [2] is consistently executed using the 'setTimeout' window method.

2.4 Game Information

During a game, the current life and score are displayed at the top left and right of the screen respectively. At the start of the game (upon refreshing the window), the player starts with three lives and a score of 0. Life heart icons are displayed using the context of the canvas' 'drawImage' method. The score (and all text in the game) is displayed with the context of the canvas' fillText method. font and textAlign methods set size, font and position of the text. The score is stored as an integer, and is converted to a string with the toString method when it needs to be displayed. (Impact is the font used in all text, with a red colour).

2.5 Player Movement

In the game, the player moves a rocket with the left, right and up arrow keys. The left and right arrows determine direction and the up arrow moves the rocket in that direction. If the rocket is still, it accelerates when the up arrow is pressed to a certain maximum speed, until the key is let go in which case it slows back down again.

When the game starts and the up arrow is pressed, an event listener is executed which sets an 'up' variable to true. 'draw' sees this, and increments a 'thrust' variable, which is the rocket's speed. When the player then releases the up arrow, a 'lowerThrust' variable is set to true which will do the opposite and decrement the thrust ('up' has already been set to false as the key was released).

It is possible for the player to change direction (i.e. press the left or right arrow) while moving, then press the up arrow. A 'slowThrust' variable is set to true if and only if:

- The up arrow is pressed
- The rocket has moved direction (rotation) since the up arrow was last pressed

The rocket will then slow down to a certain speed (thrust == 2.5), then start increasing speed as normal. This functionality simulates (in a basic form) the physics of acceleration and deceleration.

To calculate the exact position the rocket needs to be displayed, a 'moveImg' function is called each time the thrust is changed. This function uses the current angle of the rocket (a 'deg' variable) and Math JavaScript methods (cos, sin, PI), to know where to place the rocket on the canvas [3].

2.6 Bullets

The player can shoot bullets from the rocket using the space bar, which travel in the direction the rocket is facing when they are shot. An event listener is executed when the space bar is pressed, which adds an element to an 'activeBullets' array. This stores information such as angle and position. In 'draw', a 'shootBullet' function is then called which iterates through each active bullet and:

- Increments 'bulletIncrement' (to move the bullet further away from its initial position)
- Determines its exact position (using initial position, Math methods and 'bulletIncrement')
- Rotates its image and draws it (using the context of the canvas' 'translate', 'rotate' and 'drawImage' methods)
- Determines whether the bullet has 'finished' (collided or travelled far enough), and should be removed from the 'activeBullets' array using the array splice method.

2.7 Asteroids

At the start of a game (level), a set number of asteroids appear around the screen. Each individual asteroid is initially created using a 'createAsteroid' function. It is given an integer between 0 and 2 as a parameter, which determines the size of the asteroid (there are small, medium and large sized ones). Through a process of randomization, a number of properties of the asteroid are set, including initial position, direction of movement and image displayed (out of a possible three). A function 'getRndInteger' [4] is used throughout to select properties randomly. It accepts two integers and chooses a number between the integers (inclusive).

The newly created asteroid is added to an 'activeAsteroids' array (with all of its properties). 'activeAsteroids' is then iterated through in 'draw', and for each asteroid, an 'animateAsteroid' function is called which is given the current asteroid as a parameter. Here, the asteroid is:

● Moved. Each asteroid element stores a 'modX' and 'modY' value as well as initial position values [2]. The initial values are incremented by these mod values at each call to 'animateAsteroid'. Therefore, no Math functions are needed for asteroids as with the rocket as all they require is mod values and variable initial values (that keep each asteroid of a certain size the same speed).
● Rotated. This is just for visual appeal, as a rotating image looks better than a still image simply moving position. In 'animateAsteroid', only a rotation value of the asteroid element is changed.

Then, back to the for loop, the asteroid is drawn using its changed values of position and rotation (and the same context 'translate' 'rotate' and 'drawImage' methods are used as with the rocket).

2.8 UFO

At a certain point in the game, a UFO will appear and move across the top of the screen from left to right, shooting at random positions 6 times. When it is time for the UFO to appear, an 'initUfo' function is called. Here, an 'isUfo' variable is set to true, and a 'ufoCounter' is set to 0. Also, 6 variables are given random values in particular ranges, which determine where the UFO will shoot its bullets. Then in 'draw', 'ufoCounter' is increased to 300 after a certain amount of time, at which point the following code is executed:

● 'ufoCounter' is (continued to be) incremented
● A mod value is used to change the position of the UFO.
● The UFO's displayed image is updated depending on how much life it has, then it is drawn.
● If the ufoCounter corresponds to any of the values of the 6 bullet variables, a new bullet will be added to 'activeBullets' to be shot from the current position of the UFO.
● If the UFO has gone of the screen, the process repeats i.e. 'initUfo' is called again (assuming the UFO has not lost all its life)

2.9 Collision Detection

If a bullet hits an asteroid or UFO, the asteroid/UFO will become damaged. If the player's rocket collides with an asteroid (or UFO bullet), it will be destroyed.

Although different functions are used, all collision functionality uses the same fundamental implementation. A check is made to see if the x and y coordinates (of the top left position) of a particular object is within the bounds of another object (using e.g. (top left y pos of object A + size of object A in y direction) > top left y pos of object B. This is done for all bounds) [2]. The different cases are as follows:

● A function 'checkCollision'. This is used for checking if the rocket has collided with a particular asteroid, or a UFO bullet. It accepts the position and size of the asteroid/bullet as parameter, and if there is a collision, a rocket explode event occurs (see 2.10).
● A function 'checkBulletCollision'. It is given the x and y coordinates of the bullet in question. This is used for three cases:

1) A bullet colliding with an asteroid. All asteroids (in 'activeAsteroids') are iterated through, and if a collision has occurred, an 'updateAsteroid' function is called which decrements the life of the asteroid by one (small = 1, medium = 3, large = 6 life to start with). There are different images for different stages of the asteroids life cycle; they become redder in colour when they are lower in health, so the image object for each asteroid element is changed in 'updateAsteroid' too. If the asteroid has lost all its health, the game score is increased (small = 20, medium = 60, large = 120 points) and the asteroid is removed from 'activeAsteroids'. An animation also plays for a short time where the asteroid was destroyed (which displays two frames of the asteroid crumbling). An array 'destroyedAsteroids' is used to let 'draw' know where and when to display destroyed animations (with incremental counters).
(An asteroid is added to 'destroyedAsteroids' in 'updateAsteroid', if its life is now 0.)
2) A rocket bullet colliding with a UFO. If a collision has occurred, the UFO's life is decremented, and if its life has become 0 as a result, 'isUfo' is made false (so the UFO is no longer displayed) and the game score is increased by 200.
3) A UFO bullet colliding with the rocket. If a collision has occurred, a rocket explode event occurs (see 2.10).

2.10 Rocket Explode Event

As discussed in 2.9, there are two times when a rocket can be destroyed. The process is the same for both:

1) A ‘rocketExplode’ variable is set to true, to let ‘draw’ know to execute the event.
2) The asteroids in ‘activeAsteroids’ are replaced with new asteroids of the same sizes (by checking the previous sizes with an array ‘prevAsteroids’). This is done to make sure a game does not start with an asteroid too close to the rocket i.e. the middle of the canvas, since all starting positions assigned in ‘createAsteroid’ are near the borders.
3) A ‘rocketExplCounter’ is continuously incremented in ‘draw’ which displays a certain frame of animation at different times. The animation is of the rocket exploding into a flame that gets bigger. Then, when the counter reaches a certain value, the game is reset. This delay gives the player time to prepare (a game over occurs if the player’s life is now 0).

2.11 Levels

There are 10 levels. Levels get progressively harder by adding 1 or 2 more asteroids each time. A function, ‘newLevel’ is called when it is time to display a new level (when all asteroids have been destroyed for the previous level). It is given the new level number as a parameter. A switch statement is used with the parameter to create the appropriate number of asteroids (and sizes) for the given level number. If the level is 11, a game over occurs i.e. the game is completed.

A ‘resetGame’ function is called also, which sets the default position and angle of the rocket, and other default settings. If the level number is more than two, then ‘initUfo’ is called (the UFO is introduced in level 3). Finally, a ‘levelDisplay’ variable is set to true which lets ‘draw’ know to display level information.

In ‘draw’, the level number is displayed for a short time before the level begins (a counter is used once again to time this).

2.12 Game Over

If the player’s life is 0, a game over occurs (once the third and final rocket explode event has finished). Here a screen is displayed with a game over message and the final acquired score. A ‘gameOver’ variable is made true to let ‘draw’ know to only display the game over screen.

2.13 Video

A short 10 second video was added to the start of the game. In the html file, the video file is referenced and autoplayed. Then, in the JavaScript file, a ‘gameIntro’ function draws the current frame of the video. setTimeout is used to consistently call ‘gameIntro’ until the video has finished (using the ended method of the DOM video object) or the space bar has been pressed, then the game starts i.e. ‘draw’ is called and false is returned to leave the repeated ‘gameIntro’ calling.

2.14 Wrap-around

Asteroids and the rocket 'wrap-around' the canvas, i.e. they go off the screen then come back on the other side. This is implemented by simply checking if the asteroid/rocket's position is out of a particular threshold, then they are placed to the opposite end of the canvas by changing the position value.

2.15 Audio

A HTMLAudioElement interface allows easy access to <audio> elements of html. All audio files are created in JavaScript using 'new Audio (filename)'. A play method of this Audio object is then called at the appropriate times [2].

2.16 Design Choices

All images used have a cartoon look to them, since they are all made up of vector shapes. This fits with the arcade aesthetic. The video works quite well with the game visuals as the images of space are quite similar (and the bright light at the end of the video helps to transition smoothly into the game).

2.17 Other External Sources

A tutorial was used to understand how to take frames from a video and display them on the canvas [5]. To know how to move the rocket in the direction it is facing, a tutorial was used [3]. Finally, a tutorial was used to find coordinates of objects after canvas rotation [6]. All audio sounds come from the same source [7].

3 Conclusion

Overall, I think the project was successful in that the basic game functionality criteria was met, and a few additional criteria have been met (physics with the thrust, level system, collision of objects independent from the moveable object, different graphics/animations). Something that could have been added is objects appearing in asteroids when they are destroyed, giving power-ups or increasing the score. Also, in other Asteroids games the larger asteroids multiply into smaller ones after being destroyed, which then become asteroids that need to be destroyed. These features could have added to the enjoyment. Also, because the collisions are between squares, there are empty areas e.g. around the edge of asteroids that still count towards the collision. This cannot be fixed easily and it means that collisions are not as accurate as they could be in this project.

4 References

[1] Atari. Asteroids Arcade. *Atari.com*, 2017. [Online] Available from: http://www.atari.com/arcade#!/arcade/asteroids/play
[Accessed 04/05/2017]

[2] P. Newbury. Labs. *Study Direct,* 2017. [Online] Available from: http://studydirect.sussex.ac.uk/course/view.php?id=27803&topic=2
[Accessed 04/05/2017]

[3] K3N. Move an object in the direction its facing using 2d html5Canvas and ctx.rotate function. *Stack Overflow*, 2013. [Online] Available from: http://stackoverflow.com/questions/19355026/move-an-object-in-the-direction-its-facing-using-2d-html5canvas-and-ctx-rotate-f
[Accessed 04/05/2017]

[4] w3schools. A proper random function. *w3schools.com*, 2017. [Online] Available from: https://www.w3schools.com/js/js_random.asp
[Accessed 04/05/2017]

[5] T. Atkins. video + canvas = magic. *html5doctor*, 2010. [Online] Available from: http://html5doctor.com/video-canvas-magic
[Accessed 04/05/2017]

[6] K3N. Finding coordinates after canvas Rotation. *Stack Overflow*, 2013. [Online] Available from: http://stackoverflow.com/questions/17047378/finding-coordinates-after-canvas-rotation
[Accessed 04/05/2017]

[7] classicgaming. Asteroid sounds. *classicgaming.cc,* 2017. [Online] Available from: http://www.classicgaming.cc/classics/asteroids/sounds
[Accessed 04/05/2017]