Candidate number: 118435
Word count: 2058

Multimedia Design and Applications Coursework

Introduction

This report will outline the work taken out as part of the multimedia design and applications coursework. The purpose of this project was to create a game using JavaScript. Raphael.js was also used, which is a JavaScript library used to simplify working with vector graphics.

Game Inspiration

The genre of game that was the main inspiration for this project was tower defence games (1). In these types of games, the player has to place towers or turrets of some kind, usually along a path, in order to prevent the enemies from reaching the end of the path.

The turrets attack the enemies within range and once placed are usually not movable by the player. The turrets are bought with points that are usually gained by defeating the enemies.

The enemies usually only move along the path and do not fight back. They have a set life which is depleted as they are attacked by the turrets. Each level usually consists of a number of enemies moving down the path before the level ends. If any of the enemies reaches the end of the path the player loses lives.

Some of the more well-known tower defence games include the following:

- Plants Vs Zombies (2)
- Desktop tower defence (3)
- Orcs Must Die! (4)

The game that initially inspired this project was Tempest, a game originally created for arcades by Atari in 1981 (5). In this game the player only controls a single unit which can attack the enemies. This mechanic inspired the use of a player controlled unit for the game in this project. The other major influence of this game was the shape of the map. The enemies come from the centre of the screen and follow paths to the edges. This means that there is more than one path that the enemies can follow and this allows for more challenge and complex gameplay.

Game Overview

The map:

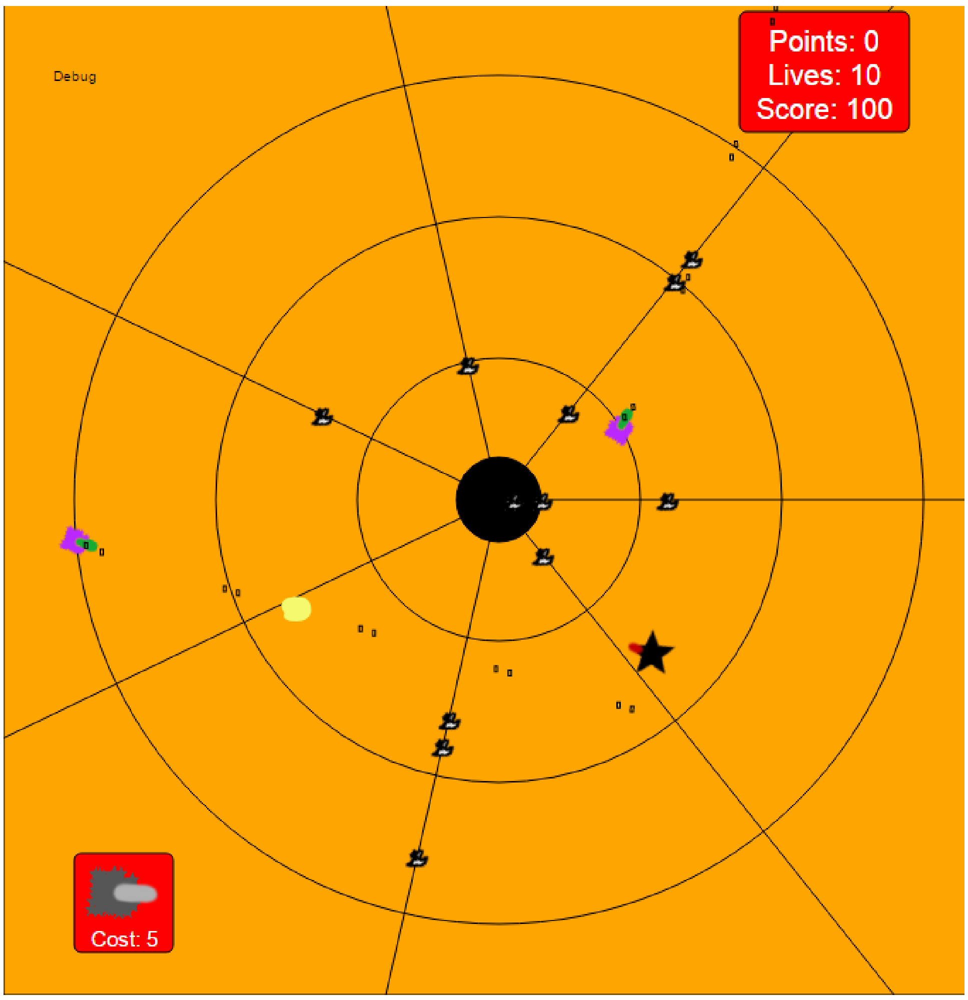

Figure 1: The game map.

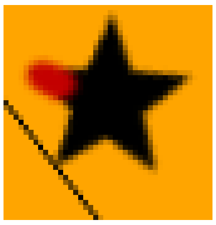

Figure 2: The player's sprite.

Figure 3: The dummy turret. Greyed out when no turrets can be bought.

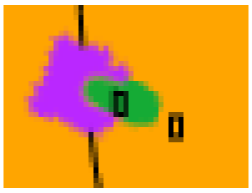

Figure 4: A turret and its bullets.

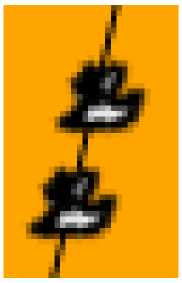

Figure 5: The enemies.

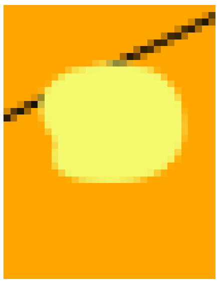

Figure 6: Coins dropped by defeated enemies.

The game will have the following functionality:

- The map will have three circles. The player can only place turrets on these circles and the turrets will snap to the nearest circle when placed.
- Each level will have a number of lines which the enemies can move along. The number of lines increases with each level.
- The enemies will come from the central hole at regular intervals.
- The player can move turrets from the menu by clicking and dragging them. Releasing the mouse button then allows them to be placed.
- Turrets can only be placed if enough points are available.
- Points can be collected by moving the player's sprite into the coins dropped by the enemies when they are defeated.
- If the enemies reach the edge of the screen the player loses a life. If the player loses all of their lives it is game over.
- The goal of the game is to survive as many waves as possible and get the high score.
- The score increases with each enemy that is defeated.
- The player can move their sprite around using the arrow keys and the "WASD" keys.
- The player's sprite can only move in the direction it is facing (using the up arrow or W) or backwards (using the down arrow or S) and it can rotate (using the left and right arrows and the A and D keys).

Functions

- Update – The update function is the main game loop and is called every 100 milliseconds. It is used to call the update function for all of the objects in the game as well as providing some functionality of its own. Collisions between the player and the coins and between the enemies and the bullets are checked for by calling the collision detection method. If collisions are detected appropriate action is taken such as the player's points being updated if they touch a coin or an enemy's health being reduced if they are hit by a bullet. The update function also prevents the turrets from firing continuously. This is done by only allowing them to fire if the difference between the last time they fired and the current time is greater than 800 milliseconds.
- Drawing the enemies' lines – The enemies can only move down line that are drawn from the centre of the map. These lines are drawn by first calculating the angle that each line is offset by. This is done by dividing 360 by the number of lines to be created. Multiplying this value by the current line index gives the angle of that line. The start point of the line is known to be the centre point of the map. The x coordinate of the end of the line is worked out by multiplying the desired length of the line by the cosine of the line's angle. This means if the line is at 0 degrees, which by default is pointing directly to the right, then the end of the line will only be affected by the line length in the x direction. The same is done for the y coordinate except the sine of the angle is considered instead. These two values are then used to draw the line.
- Create the player – The function to create the player simply creates a new player class and sets the coordinates and rotation of the player.
- Create the dummy turret and check if the player can place another turret – When the player has enough points to buy a new turret the turret symbol is shown in colour. This is done by simply hiding the black and white version and showing the colour one. The opposite is done when the player does not have enough points to buy a turret. The grey version of the turret is only an image and cannot be manipulated by the player. The colour version is a dummy turret. It still has the same functionality of a regular turret, but it can be dragged around by the player and used to place new turrets. It also cannot fire at the enemies since it is not added to the turret update array. This was done because if the turret rotates while being dragged it causes the x and y coordinates of the turret to be offset based on the rotation. The drag function of the dummy turret was done in a similar way to working out the coordinates of the lines for the enemies. The distance of the turret from the centre of the map was known based on the radius of the circles on which the turrets can be placed. The angle of the turret was worked out using a built-in function of Raphael which gives the angle between two given points. With this information the x and y coordinates of the turret when snapped to the circle could be worked out by multiplying the radius of the circle with the cosine or sine of the angle respectively. These values where then added to the coordinates of the centre point to get the final coordinate of the turret.
- Create the enemies – The enemies are given initial coordinates at the centre of the screen and their starting health is set. When the angles of the enemy lines were calculated they were stored in an array. A random index into this array is given to each of the enemies so they all move down randomly selected lines.
- Update the enemies – The new value of the enemy's coordinates is worked out in the same way as the enemy lines are calculated. The speed of the enemy is multiplied by the cosine or sine of the angle at which the enemy is moving. These values are added to the enemy's current x and y coordinates in order to work out the new position. If the enemy's

coordinates are outside the bounds of the map the enemy is removed from the enemy array and the player's lives are reduced by one. Any turrets that are currently targeting the enemy have their targets set to null.

- Create the turrets – The turrets where created by taking the coordinates worked out from the dummy turret and placing the turret in the same location. The turret is then added to the list of turrets for updating.
- Update the turrets – The turrets cannot change their coordinates so the only thing the turrets need to do in terms of movement is rotate. The rotation is done based on the angle between the turret and their current target. The current target is only chosen if the turret has no target. The function checks the enemy array for an enemy that is within the turret's range of attack. Once it finds one that enemy is set as the target. The target is set to null if the current target leaves the range of the turret.
- Make sure the player's sprite does not leave the screen – If the player's sprite leaves the edge of the screen its coordinates are changed to bring it back to the other side of the screen.
- Make the turrets fire at the enemies – The angle between the turret that is firing and its target must be worked out in order to work out the angle that the bullet must travel at. This is done using Raphael's built-in angle function which gives the angle between two given points.
- Create the bullets – The bullets are rectangles with their angle fixed at the point they are created. The starting rotation is offset by 180 degrees however as the bullets were firing backwards otherwise. The starting coordinates are also offset a bit in order to make it look like the bullet is fired from the turret and not its corner.
- Update the bullets – Moving the bullets is done in the same way as moving the enemies. For the x coordinate the cosine of the angle of the bullet is taken and multiplied by the bullet's speed. This value is added to the x coordinate of the bullet. The y coordinate is done in the same way except the sine of the angle is taken instead.
- Check for keyboard input – The player class has variables to check for user input; left, right, up and down. These Booleans are set to true when the appropriate keys are pressed, and set to false once the key is released. This is done by adding an event listener to the document. The action to take once these Booleans are set is handled in the update player function.
- Update the player – If the player pushes the left arrow the rotation of the player's sprite is decreased. If they push the right arrow the rotation of the player's sprite increases. The speed is increased or decreased by pushing the up or down arrows respectively. The player's new position is worked out in the same way as the enemies' and bullet's movement.
- Create the coins when an enemy is destroyed – A coin object is created at the same location an enemy was when it is destroyed.
- Collision detection – For this project the rotation of an object when colliding did not need to be taken into account since the enemies and the coins do not change their rotations. This meant the collision detection could be done by checking if a given point, the coordinate of the bullet for example, fell within the box created by the enemy.

Advanced functionality

The player has to focus on both placing turrets and moving their sprite which makes the gameplay more challenging than standard tower defence games.

The levels get harder as the game continues. Each subsequent level has more lines for the enemies to move on and more enemies spawned to defeat.

References

1. **Wikimedia Foundation, Inc.** Tower defence - Wikipedia. *Wikipedia.* [Online] 7 May 2016. [Cited: 19 April 2016.] https://en.wikipedia.org/wiki/Tower_defense.

2. **Electronic Arts Inc. .** Plants vs zombie games. *PopCap.* [Online] 2016. [Cited: 19 April 2016.] http://www.popcap.com/plants-vs-zombies.

3. **Preece, Paul.** Desktop tower defence. *Kongregate.* [Online] 3 March 2007. [Cited: 19 April 2016.] http://www.kongregate.com/games/preecep/desktop-tower-defense.

4. **Robot Entertainment, Inc.** Orcs Must Die! *Robot Entertainment.* [Online] 2015. [Cited: 19 April 2016.] http://origin-www.robotentertainment.com/games/orcsmustdie/.

5. **Atari Inc.** Tempest (video game). *Wikipedia.* [Online] October 1981. [Cited: 15 April 2016.] https://en.wikipedia.org/wiki/Tempest_%28video_game%29.