The Jolly Dodger

Joshua Lee

January 2018

Contents

1 Game description **2**
- 1.1 Lighthouse . . . 2
- 1.2 Enemy ships . . . 2
- 1.3 Treasure . . . 2

2 How to play **3**
- 2.1 Aim . . . 3
- 2.2 Controls . . . 3
- 2.3 Level end . . . 4
- 2.4 Game over . . . 4

3 Game Design **4**
- 3.1 Procedural generation and binary tree . . . 4
- 3.2 Lighthouse . . . 4
  - 3.2.1 Ray direction . . . 5
  - 3.2.2 Drawing shadows . . . 5
  - 3.2.3 End result . . . 6
- 3.3 Collisions . . . 6
  - 3.3.1 Player collisions . . . 6
  - 3.3.2 Island collisions . . . 7
- 3.4 Movement . . . 7
  - 3.4.1 Player movement . . . 7
  - 3.4.2 Intelligent computer movement . . . 7
- 3.5 Difficulty increase . . . 8

1 Game description

My game focuses on a pirate sailing the Jolly dodger ship. His aim is to to steal as much treasure as possible from the enemies without dying himself. Each day a new location is set as the target for the Jolly dodger. He must navigate the map and avoid being detected to have the best chance of surviving.

1.1 Lighthouse

The central piece of my game is the lighthouse feature. It is in the best interest for the player to avoid being detected by the lighthouse. Once detected the enemies will send for back up and new enemies are generated on the island. This makes it harder to complete each level and all subsequent levels, as there will be more enemies from now on. To avoid the light the player can use islands and enemies to hide in their shadows.

1.2 Enemy ships

A twist in the story line is that there are also some enemies that are in fact undercover. These ships attempt to steal and make off with the treasure while all the attention is focused on you. The player must steal more treasure than the enemies if he wants to impress his boss enough to keep him alive. Players can also let the ships steal the treasure before shooting them down and claiming it for themselves.

1.3 Treasure

Collected treasure can be one of 4 types corresponding to the corner the boat must drop the treasure off to in order for it to be collected. If the boat tries to drop the treasure off in the incorrect location the treasure will not be collected see figure 1.

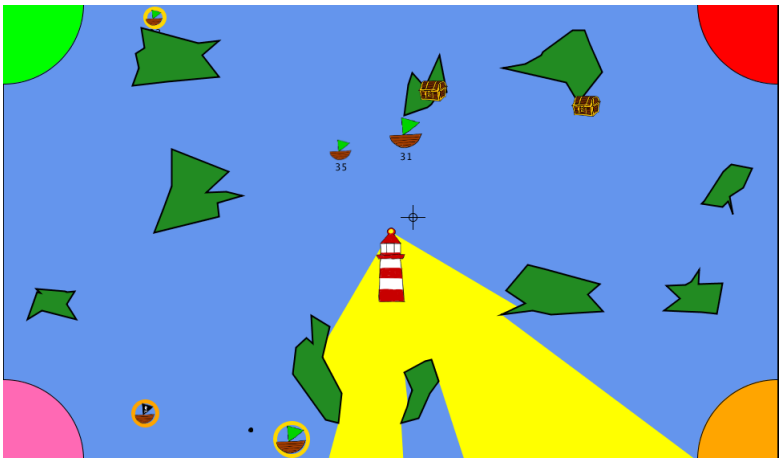

Figure 1: Coloured corners of the map corresponding to where treasure must be dropped off, as you can see the player has an orange circle round him so he must head to the orange corner

2 How to play

This section gives a game summary on how to play.

2.1 Aim

Collide with the islands to pick up treasure and then take it back to the corresponding coloured corner. Enemies will at first all appear the same but after a while it will be clear who the under cover boats are. At the end of each day you must be within £100 of the enemy treasure to survive the day. Avoid the light beam because after every time your detected a new enemy is created that will spawn on the map and respawn in all subsequent levels.

2.2 Controls

- **Speed up/ Slow down** - Up/ Down key or W/ S key
- **Rotate boat left/ right** - Left/ Right key
- **Cannon fire** - Mouse position, click to fire
- **new game** - n key
- **new level** - x key (for testing)
- **pause** - p key

2.3 Level end

The level/day will end when all treasure has been collected

2.4 Game over

Keep health above 0 and collect more treasure than your opponent to avoid game over and to get the highest score.

3 Game Design

3.1 Procedural generation and binary tree

I used a binary tree to provide the random structure of the islands on the map. This split the map into different areas allowing there to be different game objects drawn in each one, this helped in generating islands and treasure. See figure 2 to see how the play area might be split up. This is done by taking an area, if the area size is greater than a preset minimum size then randomly choose to either vertical split or horizontal split the area somewhere along the edges. These two created areas are now represented as the left and right child of the original parent. The process is repeated until no more can be created.

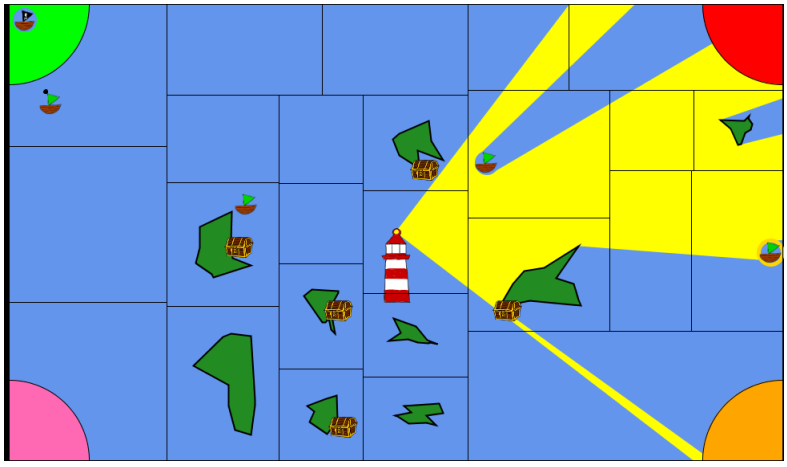

Figure 2: Binary Tree splitting up map

3.2 Lighthouse

The most important part of my game design was the necessity for rays to be able to be shined outwards. This was a pivotal requirement for the shadows to be drawn. It was also used in collision detection and in the movement algorithms

3.2.1 Ray direction

I realised that the rays only needed to be considred that were towards the verteces of the shapes. I then sent a ray 0.0001 radians either side of every vertex. This allowed all edges of the shape to be shined on as well as to send the shadow just either side of the shape. This technique could also be used for the circle shadows. The points to consider here were the tangents to the circle that pass through the point of initial ray location. The rays were again sent a fraction of a radian either side of this tangent point to detect the edge of the circle and send the shadow behind. see figure 3

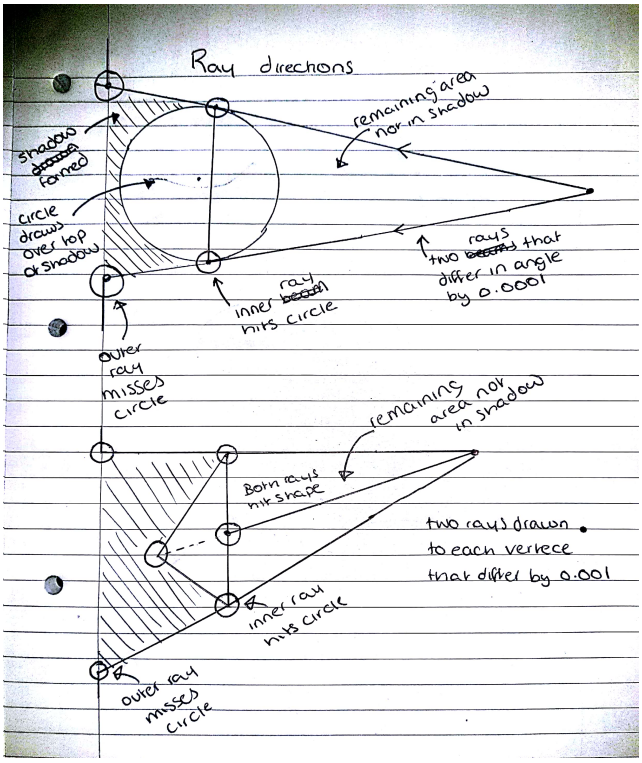

Figure 3: Representation of how rays are shone to get shadows

3.2.2 Drawing shadows

Once all rays were collected from all the shapes I kept track of all end verteces of the rays that would be used in drawing the shadow. I ordered all the rays

again using merge sort with the central vertex being the point from where all the rays originated from. Then by drawing a line between all these verteces the shadow was drawn. Finally for the shadows to reflect how a lighthouse behaves I only considered the verteces between the current angles of light shining. To pull this together I just needed to add rays to the min and max angles of the light and include these verteces along with the central point.

3.2.3 End result

The end result was the shadow effect shown in figure 4 that shows how the shadows are drawn between the angles of fire. As you can see this accurately shines a shadow on all the boats as well as the islands to create the lighthouse effect.

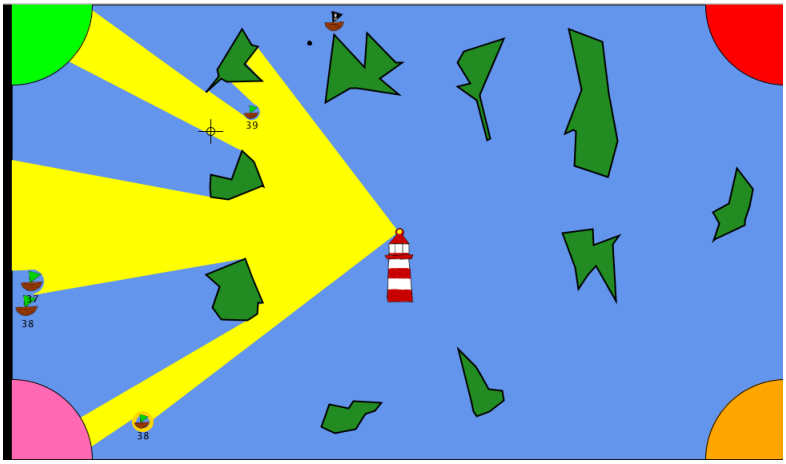

Figure 4: Final product from all the rays of light

3.3 Collisions

3.3.1 Player collisions

Player collisions with enemies were resolved based on the velocity and size of the boats. The first step was to check the distance been the central point of the boats. if the distance was ¡ less than the sum of both the radii then the boats were colliding. Once this was confirmed the momentum was calculated for both boats and also the relative velocity of both the boats. The move away velocity was then calculated based on this for both ships. An important addition to prevent overlapping was to move the position of both the boats equally away until there is no longer overlap. The reason for overlap is the delay in calculating the positions of the boats in each frame.

3.3.2 Island collisions

More considerations were needed for the Island collisions as the distance, move away velocity, and the collision point were a lot harder to calculate. I decided therefore to reuse rays in calculating the point of collisions. The first step was to locate which shape was in the direction path of motion. In order for all possible collisions to be considered, I needed to include the rays from the tangents of the circle in the direction of movement. The point of consideration was then the one of closest distance from the centre point of the circle out of all the points the rays had detected. From this the next potential collision could be obtained. Once this shape was found all edges of this shape were considered. By again representing the circle in the required form the intersection point between this and each edge could be found if it exists. A check was made to see if this point lies on the considered edge and If this was the move away velocity could be calculated based on the current velocity,the point of intersection and the angle of contact.

3.4 Movement

An important point for me was to make all movements act similarly to how a normal boat would move on water and to also allow the enemies to make intelligent movement decisions based on what roles it was tasked with and its position on the map.

3.4.1 Player movement

The player uses the arrow keys to move, the right and left keys change the direction of movement clockwise and anti-clockwise respectively. The up and down arrows move the ship forwards and backwards.

3.4.2 Intelligent computer movement

I put quite a lot of work into the seek algorithm for the enemies that could be used when the enemy was seeking treasure, looking for a drop off point or targeting the player. This made the game fun to play and gave the enemies a realistic feel.

1. In order to implement this a ray was drawn from the enemy to the seek location. If there was nothing in the way then the enemy can move directly to that location. Again to consider the collisions of an edge, a ray was also drawn from the tangents of the circle in the direction of movement.

2. If an island was detected then a work around was needed.

3. A ray was then drawn from the enemy to all verteces of the island detected.

4. The rays were ordered based on the angle between this line and the straight ahead line. The min and max angles could now be assumed to be the outer

edges of the shape and it was now required for enemy to move around one of them to get round the shape.

5. Once the vertex was found, a move away force was needed to avoid the vertex. this was calculated using the normal angle of the vertex from the line between the vertex and the enemy position. Now the enemy would intelligently move towards this position to avoid the shape effectively. See figure 5 for a visual representation of the above.

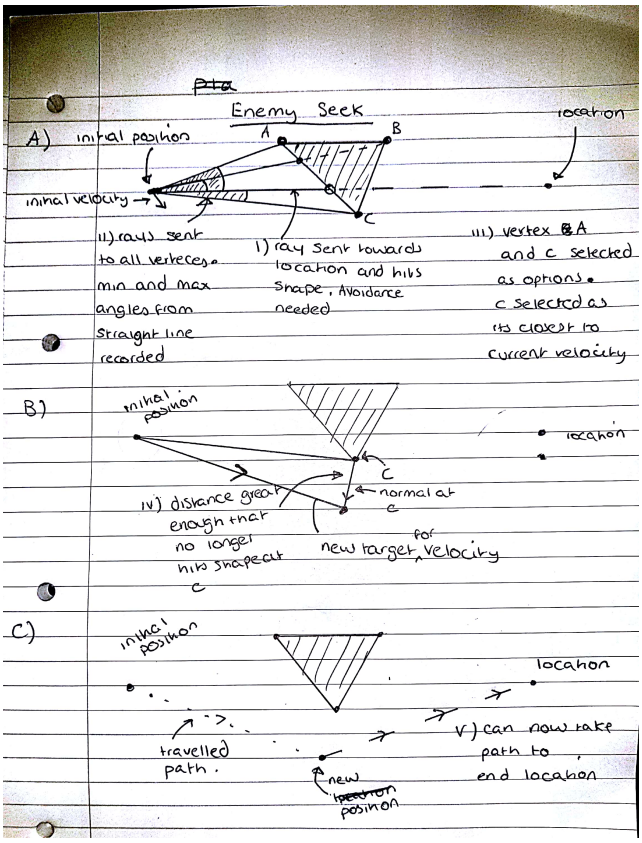

Figure 5: Example of how an island is avoided to reach destination

3.5 Difficulty increase

It is important that as the game progresses the game is getting harder each time. The first thing i thought to do was increase the size of the map. Although this increased the difficulty, the game because less enjoyable to play, the thing I liked

about the game was the clear playing location. Instead I increased the difficulty in a number of ways. I first increased the strength of the enemies as each level was completed, this included increasing the health of the enemy and increasing the fire power and fire rate. Another thing that made the game significantly harder as the game progressed was the increased number of enemies. Although the game starts off with just 3 enemies. Each time the player is detected by the lighthouse a new enemy is added. Although the player can kill the enemies in each level the number of enemies persists at the start of the next level. This gave the player a real incentive to avoid being hit by the beam.