Doge-otron: 2017 – AI Overview

Written by Jack Mair and Lance Chaney

Overview

Doge-otron: 2017 is a top down endless shooter game inspired by the 1982 arcade game, *Robotron: 2048*. *Doge-otron: 2017* is a networked game that allows players to each control an avatar, which they use to dodge enemy attacks, pick up score points, and destroy enemies in order to advance through never ending levels until they run out of lives.

Doge-otron: 2017 includes six steering behaviours. These behaviours are seek, flee, arrival, pursue, evade, wander, flock, and follow leader. By combing these behaviours in unique ways, each of the three enemy types (and the pickups!) all have distinct artificial intelligence, which creates the enjoyment and the challenge of the game.

Steering Behaviours

An explanation of how each of the behaviour types were implement are as follows:

Seek

Seek allows an entity to move towards a desired location in world space.

It achieves this by first creating a displacement vector, which is defined as the desired location subtracted by its current location.

It then finds the length of this vector by taking its magnitude. A desired velocity is then calculated by normalizing the displacement vector (by dividing it by its length) and then multiplying it by the maximum movement speed of the entity.

Finally, a steering vector is calculated by taking the desired velocity and subtracting the entity's current velocity from it. This results in a matrix that steers the entity towards the desired location.

Flee

Flee allows an entity to move in the opposite direction of a desired location in world space. It is calculated identically to seek, except that the resultant steering vector is multiplied by negative one. This creates a new vector that moves the entity in the exact opposite direction of the desired location.

Arrival

Arrival is modification of seek that allows an entity to reduce its velocity when reaching a desired location based on how close it is to it. Without arrival, the entity will ocelot around its desired location, always moving past it as it travels at its maximum speed and never stopping.

Arrival creates a displacement vector and a magnitude of that vector in an identical way to seek. After this, it then checks if the left of the magnitude is below constant value.

If it is above this value, the entity is not close enough to its desired location yet and a desired velocity is calculated the same way as seek.

If it is below the value, the entity is getting close to its desired location and must slow down. The desired velocity is calculated in the same way as seek, except it is now also multiplied by the displacement magnitude divided by a constant. This means that as the entity gets closer to its desired location the displacement magnitude will decrees and the entity's desired velocity will reduce in size.

After the desired velocity is calculated, a steering vector is derived the same way as it is in seek.

Pursue

Pursue is another modification of seek. Pursue allows an entity to be more effective at chasing a moving target. This is achieved by predicting where they will move to and moving towards that location.

First, a future position is calculated by adding the target's current position with the target's velocity.

This is then multiplied by a number calculated from the equation:

Magnitude((target's position – current position) / target's move speed)

This allows a shorter future position away from the target to be calculated based on how close the entity and the target are from each other. This allows the entity it to be tighter in its prediction of where the player will be and move more directly towards it the closer it gets.

Once this future position is calculated, the seeking logic is then applied using future position as its target position in the equation.

Evade

Evade allows the entity to run away form a moving target more effectively then flee using the same logic as pursue.

A future position is calculated identically to pursue. The flee logic is then used with future position being set as the target position in the equation.

Wander

Wander allows an entity to move randomly around in natural looking way. Without wander, and applying a random velocity at random times, the entity moves sporadically, flipping around in large turning angles and doubling back on where it previously came from.

Wander works by first calculating a position in front of the entity by normalizing the current velocity, multiplying it by the maximum move speed, and then multiplying it by 10.

A random position inside a circle at the origin with a radius of four is then calculated.

A target position is then derived by adding the current position, the position in front of the entity, and the random coordinates of the circle.

The result of this produces a position in front of the entity with a random divergence away from the position its current velocity will take it to dependant on the radius of the circle.

Once this is calculated, the seek logic is then applied so that the target moves towards the determined target position.

Once the wander equation is called again, there is only a 1/50 chance the entity will create a new target to wander to, otherwise it will seek towards the target created by the previous call of the wander function. This creates a more natural looking behaviour as there is a random length of time for how long the entity travel in a straight line before finding a new target.

Flock

Flock allows entities to move together in a pack more naturally. Without flock, and when multiple entities are pursuing a target, the entities bunch up together in a single point and move over the top of one another.

Flock works by the entity first creating a list of all the neighbours around it that it should flock with.

It then calculates three different velocities: alignment, cohesion, and separation.

Alignment synchronizes the entity's velocity up to the neighbours. The average velocity of all the neighbours is calculated by adding up all the velocities of the neighbours and dividing it by the total number of neighbours. The alignment vector is set as the normal of this.

Cohesion keeps the entity moving close to the average position of the neighbours. The average position of all the neighbours is calculated by adding up all the positions of the neighbours, dividing it by the total number of neighbours, and subtracting the entity's current position. The cohesion vector is set as the normal of this. The cohesion matrix is multiplied by 0.5 to keep the entities from bunching up even more so.

Separation moves the entity away from the same position as its neighbours. The average distance between the entity and each neighbour is calculated by adding up all the distances and dividing by the number of neighbours. In order to reverse it, this vector is then multiplied by negative one. The separation vector is set as the normal of this.

Once the three vectors are calculated they are added together to create a single vector. This vector is then added to the current position of the entity and is used with seek and arrival as the target position of the entity.

Leader Follow

Leader Follow allows an entity to follow a target, staying behind it even when it is stationary. Without follow leader, and just using seek, the entity will travel to the exact location of the target and sit on top of it when it is stationary.

Follow leader first calculates a follow position behind the target by creating a vector equal to the targets velocity multiplied by negative one. If the target is currently stationary, the targets velocity from the last time they were moving is used instead.

This vector is then normalised and multiplied by 2. Multiplying by a larger constant means the entity follows the target from a further away position.

Seek and arrival is then applied with the follow position being used as the target position.

Steering

Each entity uses more than one ai behaviour simultaneously to decide where to move. It does this by calling each function and adding the resultant steering acceleration vectors together. Once the final steering vector has been created, it limits the size of the acceleration by reducing its size to the entity's maximum acceleration. This allows the entity to move more smoothly and realistically, meaning they have to slow down before turning 180 degrees on themselves.

AI Behaviours

An explanation of how each of the AIs work are as follows:

Zombie Enemy

Zombie Enemies are the basic enemy of the game. When a player is not within its aggro range, it wanders.

When a player is within its aggro range, it pursues the closest player and flocks with other Zombies that are close to it.

If a zombie touches a player, it causes them to lose a life.

Shooter Enemy

Shooter Enemies are the more advance enemy of the game. When a player is not within its aggro range, it wanders.

When a player is within its aggro range, it pursues the closest player and flocks with other Shooters that are close to it.

If the player gets too close to it, it stops pursuing and evades away from the closest player.

If a shooter touches a player, it causes them to lose a life.

A shooter occasionally shoots a bullet that moves at a fixed velocity in the direction of the closest player's current position + their current velocity.

Snake Body Enemy

Snake Body Enemies are the wild card enemy of the game.

The Snake Body spawns with an id number that is equal to its position in the 'spawn queue'. If it is the first to spawn in its id is one, if it was second its id is two, etc.

If a Snake Body exists in the level that has an id equal to their own – 1, then the Snake Body pursues to it. Otherwise, the Snake Body wanders. The maximum speed of the Snake Body is equal to either the Snake they are following or, if it is wandering, is proportional to how many Snake Body are following it.

This results in a single slow moving Snake that divides into faster moving smaller snakes the more the middle Snake Body parts are killed.

Pickups

There are three types of Pickups in the game: Small Score Pickup, Large Score Pickup, and Health Pickup.

All Pickup types share the same AI.

When a player has not entered its aggro range, it wanders.

When a player enters its aggro range, and only when the Pickup is currently wandering, then the Pickup leader follows that player.

The pickup will continue to leader follow the player until either the player dies or the level is over.

If the player dies, the pickup begins to wander again, and again leader following the first player to enter its aggro range.

If the level is over then the score/health value of the Pickup is added to the player it is currently leader following.

This results in the gameplay that the players score from the Pickups are only safe when the round is over. If the player dies whilst holding a lot of Pickups, they become available again for other players to steal whilst the dead player is still respawning.