University of South Wales

BSc Computer Games Development

School of Computing and Mathematics

Artificial Intelligence for Game Developers Applying AI Techniques

Student ID: 13060988

https://github.com/PlaceholderGames/DinoPark-2017/tree/13060988

March 23, 2018

Abstract

Through out this report the methods chosen to implementation Predator and Prey AI are shown, each state of the AI controllers is discussed and the decision trees are broken down to explain why these methods were chosen. Additional features have been implemented to improve the AI's interaction with the environment.

Contents

1 Ankylosaurus

1.1 Description

The Ankylosaurus is a herbivore dinosaur that moves in herds, It grazes on nearby grass and searches for nearby water if thirsty. if a single predator approaches, the Ankylosaurus will stand its ground, face and attack the predator as they come within range. If there are multiple predators the Ankylosaurus will break off from the herd and flee away from the closest predator, if they manage to survive this attack they regroup with the herd. Ankylosaurus' have very strong armour and a huge tail club to defend against predators, because of the huge mass of an Ankylosaurus it's around 3 times slower than a Velociraptor, this makes it hard for them to escape. On death the Ankylosaurus' body can be consumed as meat by nearby predators. The Ankylosaurus AI Controller is designed to reflect this information.

1.2 States

1.2.1 Grazing

To check if the Ankylosaurus can see predators, firstly the list of nearby predators needs to be updated. The list is created using the Field of View Behaviour, it adds Transforms that exist in the Visible Targets List that have the Tag "Rapty".

Ankylosaurus::UpdatePredators::139

```
Predators = new List < Transform > ();

foreach(var target in Sight.visibleTargets) {
        if (target.gameObject.CompareTag("Rapty")) Predators.Add(target);
}
```

In the Grazing State the check for nearby Predators is carried out each update, If there are nearby Predators it changes to the Alert State.

Grazing::Update::40

```
if (parent.Predators.Count > 0) {
        parent.Search.enabled = false;
        parent.Path.enabled = false;
        parent.SearchAgent.enabled = false;
        parent.Search.target = null;
        parent.Wander.enabled = false;
        parent.Wander.target = null;
        parent.State.Change(Alerted.Instance);
}
```

Hunger is initialised in the Ankylosaurus script, it decays over time, this causes the Ankylosaurus to get hungry. If hungry and the point below an Ankylosaurus' mouth contains grass the Ankylosaurus changes to the Eating State. The check to see if there's Grass below the Ankylosaurus' mouth is performed by comparing the point a ray cast hits the Terrain from the Ankylosaurus' mouth with the List of Grass stored in the Grass script of the Terrain.

Grazing::Update::50

```
else if (parent.Hunger < 65.0f) {
        RaycastHit hit;
        var rayLength = 2.0f;

        var mouth = parent.transform.Find("Armature/Chest/Neck/HeadRig/DownMouthStart/DownMouthRig/DownMouthRig_end").position

        Debug.DrawRay(mouth, -parent.transform.up * rayLength, Color.red);

        // Shoot raycast
        if (Physics.Raycast(mouth, -parent.transform.up, out hit, rayLength)) {
                if (parent.Terrain.GetComponent < Grass > ().Details[(int) hit.point.z, (int) hit.point.x] != 0) {
                        parent.Wander.enabled = false;
                        parent.Wander.target = null;
                        parent.State.Change(Eating.Instance);
                }
        }
}
```

Thirst is initialised in the Ankylosaurus script, it decays over time, this causes the Ankylosaurus to be thirsty. If thirsty it will search for closest water using the A* Search algorithm, the path to the water will followed until the Ankylosaurus is able to drink the water, when this is true it changes to the Drinking State.

Grazing::Update::71

```
else if (parent.Thirst < 65.0f) {
        if (parent.transform.position.y > 35.0f) {
                if (parent.Path.path.nodes.Count == 0) {
                        parent.Search.enabled = true;
                        parent.Path.enabled = true;
                        parent.SearchAgent.enabled = true;

                        List < Vector2 > waterTiles = new List < Vector2 > ();

                        for (var y = 0; y < 2000; y += 25) {
                                for (var x = 0; x < 2000; x += 25) {
                                        if (parent.Terrain.GetComponent < Terrain > ().SampleHeight(new Vector3(x, 0, y)) < 3
                                }
                        }

                        Vector2 closestWater = new Vector2();

                        foreach(var waterTile in waterTiles) {
                                var distance = Vector2.Distance(new Vector2(parent.transform.position.x, parent.transform.pos

                                if (closestWater == new Vector2(1000, 1000)) closestWater = waterTile;
                                else if (distance < Vector2.Distance(new Vector2(parent.transform.position.x, parent.transform
                        }

                        parent.Search.mapGrid.seaLevel = 20.0f;
                        parent.Search.target = Object.Instantiate(new GameObject(), new Vector3(closestWater.x, 30.0f, closest
                        parent.Path.path = parent.Search.path;
                }
                else {
                        parent.move(parent.Path.getDirectionVector());
                }
        }
        else {
                parent.Search.enabled = false;
                parent.Path.enabled = false;
                parent.SearchAgent.enabled = false;
                parent.Search.target = null;
                parent.Path.path = null;
                parent.Wander.enabled = false;
                parent.Wander.target = null;
                parent.State.Change(Drinking.Instance);
        }
}
```

The herds centre is calculated by the HerdCentre Script. If the Ankylosaurus moves to far away from the centre of the herd. it moves towards the centre using the A* path finding algorithm.

Grazing::Update::130

```
else if (Vector3.Distance(parent.transform.position, parent.herdCentre.transform.position) > 40.0f && parent.Path.path.nodes.C
        parent.Wander.enabled = false;

        if (!parent.Search.enabled) {
                parent.Search.enabled = true;
                parent.Path.enabled = true;
                parent.SearchAgent.enabled = true;

                parent.Search.target = parent.herdCentre;

                parent.Path.path = parent.Search.path;
        }
}
else if (Vector3.Distance(parent.transform.position, parent.herdCentre.transform.position) > 15.0f && parent.Path.path.nodes.C
        parent.move(parent.Path.getDirectionVector());
}
```

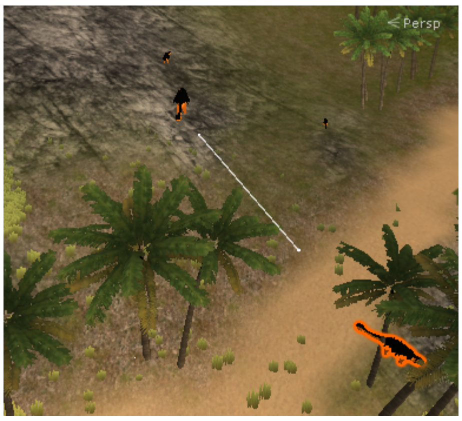


Figure 1: Ankylosaurus returning to the Herd

If none of these conditions are true it wanders using the Wander Behaviour.

Grazing::Update::163

```
if (!parent.Wander.enabled) parent.Wander.enabled = true;
```

This implementation is successful in emulating the behaviour of Ankylosaurus while grazing. If the Ankylosaurus catches sight of a predator it will enter an alerted state. If its hungry it will eat grass, if its thirsty it will search for the nearest water. It keeps close to the herd to make sure its safe and wanders around within the herd.

1.2.2 Alerted

While the State is Alerted the Ankylosaurus checks if there's nearby Predators, if true the State is changed to Grazing. The second check is performed to see if there's a single predator, if true the State is changed to Attacking. A Final Check is performed to see if there's multiple Predators, if true the State is changed to the Fleeing State.

Alerted::Update::40

```
if (parent.Predators.Count == 0) parent.State.Change(Grazing.Instance);

if (parent.Predators.Count == 0) {
        parent.GetClosestPredator();
        parent.State.Change(Attacking.Instance);
}

if (parent.Predators.Count >= 2) parent.State.Change(Fleeing.Instance);
```

1.2.3 Eating

While the State is Eating, the Ankylosaurus regenerates Hunger over time, once full it returns to the Grazing State.

Eating::Update::43

```
if (parent.Predators.Count != 0) parent.State.Change(Alerted.Instance);
else {
        if (parent.Hunger > 100.0f) {
                parent.Hunger = 100.0f;
                parent.State.Change(Grazing.Instance);
        }

        if (_time > _tick) {
                _time = 0.0f;
                parent.Hunger += _regeneration;
        }
}

_time += Time.deltaTime;
```

This implementation is successful in emulating the behaviour of Ankylosaurus while eating. If the Ankylosaurus was eating grass it would continue to eat until its no longer hungry then return to grazing.

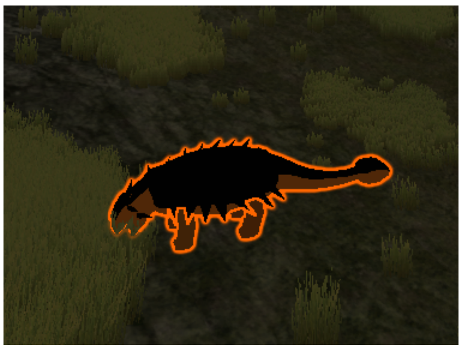


Figure 2: Ankylosaurus Eating Grass

1.2.4 Drinking

While the State is Drinking, the Ankylosaurus regenerates Thirst over time, once its Thirst is quenched it returns to the Grazing State.

Drinking::Update::43

```
if (parent.Predators.Count != 0) parent.State.Change(Alerted.Instance);
else {
        if (parent.Thirst > 100.0f) {
                parent.Thirst = 100.0f;
                parent.State.Change(Grazing.Instance);
        }

        if (_time > _tick) {
                _time = 0.0f;
                parent.Thirst += _regeneration;
        }
}

_time += Time.deltaTime;
```

This implementation is successful in emulating the behaviour of Ankylosaurus while drinking. If the Ankylosaurus was drinking it would drink until its thirst is quenched then return to grazing.

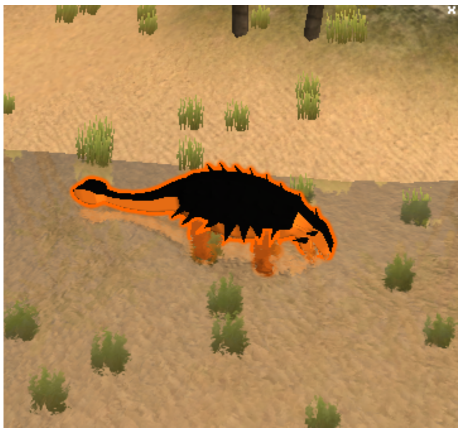


Figure 3: Ankylosaurus Drinking Water

1.2.5 Attacking

While the State is Attacking, If Health is less than 0 the State is Changed to Dead. If there are multiple predators the State is changed to Flee. If not the Face behaviour is enabled and the face target is set to to the closest predator by calling the getClosestPredator() method. If the Ankylosaurus collides with the Velociraptor in this State the Velociraptor health is reduced dramatically. If the Ankylosaurus face target State is Dead, the State is changed to the Alert State.

Attacking::Update::39

```
if (parent.Health < 0.0f) {
        parent.Face.enabled = false;
        parent.Face.target = null;
        parent.State.Change(Dead.Instance);
}

if (parent.Predators.Count == 0) {
        parent.Face.enabled = false;
        parent.Face.target = null;
        parent.State.Change(Alerted.Instance);
}

if (!parent.Face.enabled) {
        parent.Face.enabled = true;
        parent.Face.target = parent.GetClosestPredator().gameObject;
}
```

This implementation is successful in emulating the behaviour of Ankylosaurus while being attacking by a single predator, it would stand its ground in a defensive stance, fighting using its large tail club and relying on its strong defensive armour.

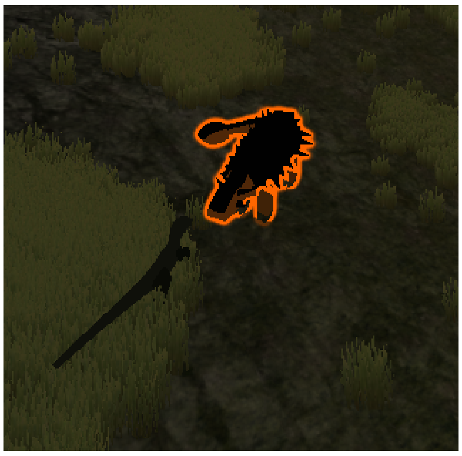


Figure 4: Ankylosaurus in a Defensive State

1.2.6 Fleeing

While the State is Fleeing, the Ankylosaurus enables the Flee behaviour and sets the flee target to the closest predator by calling the getClosestPredator() method. If the distance between the Ankylosaurus and the flee target is greater than 150.0f, The State is changed to Alerted, as the Escape from danger has been a success.

Fleeing::Update::39

```
if (!parent.Flee.enabled) {
        parent.Flee.target = parent.GetClosestPredator().gameObject;
        parent.Flee.enabled = true;
}
else {
        if (parent.Health <= 0.0f) {
                parent.Flee.enabled = false;
                parent.Flee.target = null;
                parent.State.Change(Dead.Instance);
        }
        else if (Vector3.Distance(parent.transform.position, parent.Flee.target.transform.position) > 150.0f) {
                parent.Flee.enabled = false;
                parent.Flee.target = null;
                parent.State.Change(Alerted.Instance);
        }
}
```

Although the Ankylosaurus wouldn't be able to outrun the Velociraptor individuals from the herd will escape, The fleeing is successful in emulating the behaviour of how a Ankylosaurus is described to of behave when fleeing a predator.

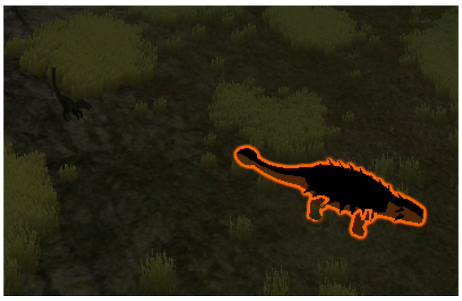


Figure 5: Ankylosaurus fleeing from a Predator

1.2.7 Dead

While the State is Dead, if the death boolean is true the Death() function is called that destroys the Ankylosaurus' GameObject.

Dead::Update::42

```
if (parent.death)
        parent.Death();
```

Ankylosaurus::Death::170

```
Destroy(gameObject);
```

This implementation is successful in emulating the behaviour of Ankylosaurus while in death.

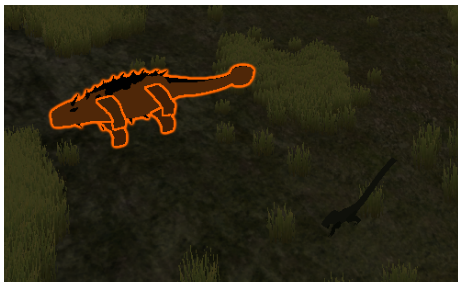


Figure 6: Dead Ankylosaurus

1.3 Future Development

- Creating a defensive herd formation to scare and defend against predators.
- Removing the grass after finished eating at the coordinates of mouth.
- Using fuzzy logic to increase running speed based of the distance from predator.
- Improving on the herding and wandering behaviours.

2 Velociraptor

2.1 Description

The Velociraptor is a carnivorous dinosaur that moves in packs, each pack members will follow an an Alpha. They roam the land, in search of prey and water. A Velociraptor' will attack Prey if its in good health as they needs food to survive, if badly injured a Velociraptor can flee to recover because its runs at 24mph, it's estimated to be the the fastest dinosaur. Velociraptor' are very fast and have strong claws and teeth to kill their prey, They continuously launch small and try to keep distance from the larger Prey such as the Ankylosaurus. On death the Velociraptor' body can be consumed as meat by other carnivorous dinosaurs. The Velociraptor AI Controller is designed to reflect this information.

2.2 States

2.2.1 V-Hunting

The Velociraptor check to see if there's any dead prey in the vicinity, if so the Arrive behaviour is used to move the Velociraptor to the position of the dead body and then it changes to the eating state.

Velociraptor::Update::40

```
if (parent.DeadPrey.Count > 0) {
        parent.Wander.enabled = false;
        parent.Wander.target = null;

        if (Vector3.Distance(parent.transform.position, parent.getClosestDeadPrey().transform.position) < 10.0f) {
                parent.Arrive.target = null;
                parent.Arrive.enabled = false;
                parent.State.Change(V_Eating.Instance);
        }
        else {
                if (!parent.Arrive.enabled) {
                        parent.Arrive.target = parent.getClosestDeadPrey().gameObject;
                        parent.Arrive.enabled = true;
                }
        }
}
```

The Velociraptor checked to see if it can see and living prey, if so it changes state to the Alerted State.

Velociraptor::Update::60

```
else if (parent.LivingPrey.Count > 0) {
        parent.Wander.enabled = false;
        parent.Wander.target = null;
        parent.State.Change(V_Alerted.Instance);
}
```

The Velociraptor uses the same implementation as the Ankylosaurus to find and follow a path to the nearest water, the changes to the drinking state.

Velociraptor::Update::66

```
else if (parent.Thirst < 65.0f) {
        if (parent.transform.position.y > 35.0f) {
                if (parent.Path.path.nodes.Count == 0) {
                        parent.Search.enabled = true;
                        parent.Path.enabled = true;
                        parent.SearchAgent.enabled = true;

                        List < Vector2 > waterTiles = new List < Vector2 > ();

                        for (var y = 0; y < 2000; y += 25) {
                                for (var x = 0; x < 2000; x += 25) {
                                        if (parent.Terrain.GetComponent < Terrain > ().SampleHeight(new Vector3(x, 0, y)) < 3
                                }
                        }

                        Vector2 closestWater = new Vector2();

                        foreach(var waterTile in waterTiles) {
                                var distance = Vector2.Distance(new Vector2(parent.transform.position.x, parent.transform.pos

                                if (closestWater == new Vector2(1000, 1000)) closestWater = waterTile;
                                else if (distance < Vector2.Distance(new Vector2(parent.transform.position.x, parent.transform
                        }

                        parent.Search.mapGrid.seaLevel = 20.0f;
                        parent.Search.target = Object.Instantiate(new GameObject(), new Vector3(closestWater.x, 30.0f, closest
                        parent.Path.path = parent.Search.path;
                }
                else {
                        parent.move(parent.Path.getDirectionVector());
                }
        }
        else {
                parent.Search.enabled = false;
                parent.Path.enabled = false;
                parent.SearchAgent.enabled = false;
                parent.Search.target = null;
                parent.Path.path = null;
                parent.Wander.enabled = false;
                parent.Wander.target = null;
                parent.State.Change(V_Drinking.Instance);
        }
}
```

If all the other decisions are false the Velociraptor will wander the terrain in search of prey, meat and water.

Velociraptor::Update::127

```
if (!parent.Wander.enabled) parent.Wander.enabled = true;
```

This behaviour could of been improved with increased development time, a pack behaviour could be implemented to allow a pack to follow the alpha unless distracted by prey, food or water.

2.2.2 V-Drinking

While the State is Drinking, the Velociraptor regenerates its Thirst over time, once its Thirst is quenched it returns to the Hunting State.

V-Drinking::Update::43

```
if (parent.Thirst > 100.0f) {
        parent.Thirst = 100.0f;
        parent.State.Change(V_Hunting.Instance);
}
else {
        if (_time > _tick) {
                _time = 0.0f;

                parent.Thirst += _regeneration;
        }
}

_time += Time.deltaTime;
```

This implementation is successful in emulating the behaviour of Velociraptor while drinking. If the Velociraptor was drinking it would drink until its thirst is quenched then return to hunting.

2.2.3 V-Alerted

While the State is V-Alerted, if there's no Living Prey the State is changed to V-Hunting. if Health is greater than 20.0f, The State is changed to V-Attacking else its changed to V-Fleeing.

V-Alerted::Update::39

```
if (parent.LivingPrey.Count == 0) {
        parent.State.Change(V_Hunting.Instance);
}
else {
        if (parent.Health >= 20.0f) parent.State.Change(V_Attacking.Instance);
        else parent.State.Change(V_Fleeing.Instance);
}
```

This implementation of the Alert behaviour is successful but it could be improved. The Velociraptor would flee when injured to fight another day as they are intelligent creatures.

2.2.4 V-Attacking

While the state is attacking, the Velociraptor will seek out and attack the targeted prey. If the prey dies it'll change state to continue hunting, If the Velociraptor is injured it would flee from the closest prey to recover its injures. If the Velociraptors health is 0 it dies.

V-Attacking::Update::39

```
if (!parent.Seek.enabled) {
        parent.Seek.target = parent.getClosestLivingPrey().gameObject;
        parent.Seek.enabled = true;
}
else {
        if (! (parent.Seek.target.gameObject.GetComponent < Ankylosaurus > ().State.CurrentState is Dead)) {
                if (parent.Health < 0.0f) {
                        parent.Seek.enabled = false;
                        parent.Seek.target = null;
                        parent.State.Change(V_Dead.Instance);
                }
                else if (parent.Health < 20.0f) {
                        parent.Seek.enabled = false;
                        parent.Seek.target = null;
                        parent.State.Change(V_Fleeing.Instance);
                }
        }
        else {
                parent.Seek.enabled = false;
                parent.Seek.target = null;
                parent.State.Change(V_Hunting.Instance);
        }
}
```

This state simulates a wolf attacking is prey and the decisions it would make in this process. In combination with the collision detection to cause damage to the target it works well.

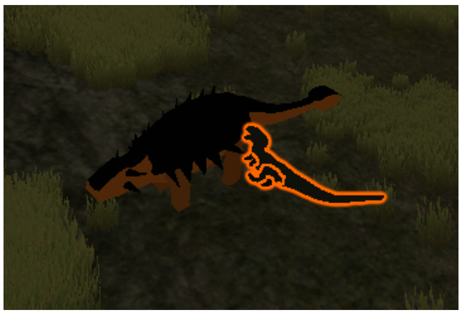


Figure 7: Velociraptor attacking its Prey

2.2.5 V-Fleeing

While the State is V-Fleeing, the Velociraptor enables the Flee behaviour and sets the flee target to the closest alive prey by calling the getClosestLivingPrey() method. If the distance between the Velociraptor and the flee target is greater than 300.0f, The State is changed to V-Alerted, as the Escape from danger has been a success.

V-Fleeing::Update::39

```
if (parent.Health < 0.0f) parent.State.Change(V_Dead.Instance);
else {
        if (!parent.Flee.enabled) {
                parent.Flee.target = parent.getClosestLivingPrey().gameObject;
                parent.Flee.enabled = true;
        }
        else if (Vector3.Distance(parent.transform.position, parent.Flee.target.transform.position) > 300.0f) {
                parent.Flee.enabled = false;
                parent.Flee.target = null;
                parent.State.Change(V_Hunting.Instance);
        }
}
```

While injured the Velociraptor flees, to recover as long as he's eaten and has drank water, his health will regenerate. He can they return to the hunt.

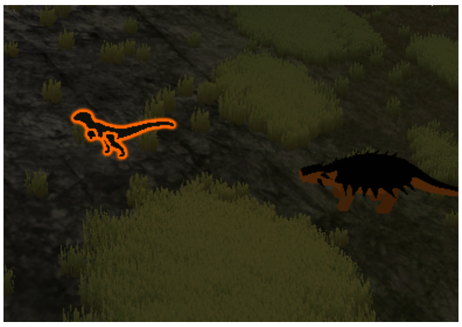


Figure 8: Velociraptor Fleeing while injured

2.2.6 V-Eating

While the State is V-Eating, the Velociraptor regenerates Hunger over time, once the Meat of the closestDeadPrey() is less than 0, The preys death boolean is set to true*, then the Velociraptor returns to the V-Hunting State.

* This calls for the GameObject of the Prey to be destroyed.

V-Eating::Update::43

```
if (parent.Predators.Count != 0) parent.State.Change(Alerted.Instance);
else {
        if (parent.Hunger > 100.0f) {
                parent.Hunger = 100.0f;
                parent.State.Change(Grazing.Instance);
        }

        if (_time > _tick) {
                _time = 0.0f;
                parent.Hunger += _regeneration;
        }
}
_time += Time.deltaTime;
```

This implementation is successful in emulating the behaviour of Velociraptor while eating. The Velociraptor's would continue to eat the meat of the dead prey until there was no more, then return to the Hunt.

2.2.7 V-Dead

While the State is V-Dead, if the death boolean is true the Death() function is called that destroys the Velociraptor' GameObject.

V-Dead::Update::42

```
if (parent.death)
        parent.Death();
```

Velociraptor::Death::189

```
Destroy(gameObject);
```

This implementation is successful in emulating the behaviour of Velociraptor while in death.

2.3 Future Development

- Creating a pack behaviour that enables members of the pack to follow the alpha, and the leader to issue commands to the beta members.
- Sneaking behaviour to sneak up on and attack from behind the Prey.
- Velociraptors to scavenging meat from dead or injured Velociraptor.
- Velociraptors are thought to fight over food, this could be implemented into the eating state.

3 Additional

3.1 Finite State Machine

A Finite State Machine was implemented that ensures the dinosaurs are only in 1 state at a time and enables the execution code on entrance and exit of these states, this is a perfect chance to update the animation states. A template state class from the Finite State Machine can be used to create state instances such as the Ankylosaurus Grazing State.

Implementation

The finite state machine was implemented following instructions from a YouTube tutorial. (Joey The Lantern, 2010). The source code is displayed below.

```
namespace FiniteStateMachine
{
    public class FiniteStateMachine<T>
    {
        public State<T> CurrentState { get; private set; }
        public T Parent;

        public FiniteStateMachine(T parent)
        {
            Parent = parent;
            CurrentState = null;
        }

        public void Change(State<T> newState)
        {
            if (CurrentState != null)
                CurrentState.Exit(Parent);
            CurrentState = newState;
            CurrentState.Enter(Parent);
        }

        public void Update()
        {
            if (CurrentState != null)
                CurrentState.Update(Parent);
        }
    }

    public abstract class State<T>
    {
        public abstract void Enter(T parent);
        public abstract void Exit(T parent);
        public abstract void Update(T parent);
    }
}
```

3.2 Dynamic Grass

The dynamic grass system allows dinosaurs access to the coordinates of the grass, it also spawns new or increases the length of exisiting grass on random coordinates of the terrain that are above the sea level. A screen shot to demostrate the dynamic grass system is shown in Figure 9 below.

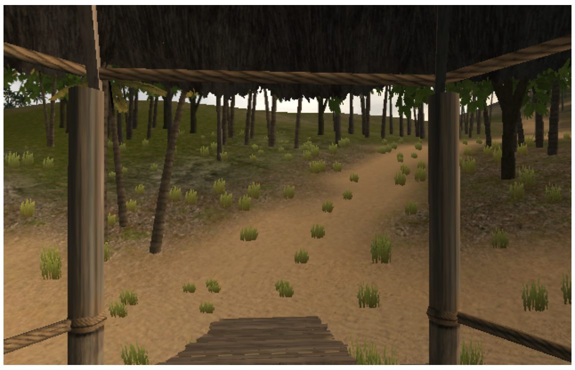
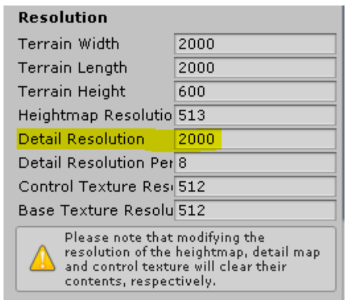


Figure 9: Demonstration of Dynamic Grass System

Implementation

For this script to be implemented the detail resolution of the terrain needs to be updated to match the size of the terrain.



Resolution

Terrain Width	2000
Terrain Length	2000
Terrain Height	600
Heightmap Resolutio	513
Detail Resolution	2000
Detail Resolution Per	8
Control Texture Res	512
Base Texture Resolu	512

Please note that modifying the resolution of the heightmap, detail map and control texture will clear their contents, respectively.

Figure 10: Updating Detail Resolution of Terrain

A script called Grass is added to the terrain on start it captures the detail layer of the terrain where the grass is stored.

Grass::Start::16

```
Terrain = GetComponent < Terrain > ();

Details = Terrain.terrainData.GetDetailLayer(0, 0, Terrain.terrainData.detailWidth, Terrain.terrainData.detailHeight, 0);
```

In the update it updates the terrain with any grass that added or updated.

Grass::Update::25

```
if (grass_time > grass_ticks) {
        Growth();
        Terrain.terrainData.SetDetailLayer(0, 0, 0, Details);
}

grass_time += Time.deltaTime;
```

The growth of the grass and height checking is carried out in the Growth function.

Grass::Growth::36

```
var z = Random.Range(0, 2000);
var x = Random.Range(0, 2000);

if (Terrain.GetComponent < Terrain > ().SampleHeight(new Vector3(x, 0, z)) > 35.0f && Details[z, x] < 16) Details[z, x] += 1;
else Growth();
```

3.3 Herding

The herding is implemented by creating a transform named Herd Centre attaching a script that calculates the centre point of the herd, and moves the herd transform to the centre of the herd, this is performed in the update.

HerdCentre::Update::36

```
var centre = new Vector3();
var count = 0;

foreach(var ankylosaurus in FindObjectsOfType < GameObject > ()) {
        if (ankylosaurus.gameObject.tag == "Anky") {
                centre += ankylosaurus.transform.position;
                count++;
        }
}

transform.position = centre / count;
```

The Herd Centre Transform has 2 sphere collides that act as a trigger. The first sphere is the Max range of the Herd, the second on is the point the herd returns to after wandering to far from the herd. This can be seen below in Figure 11.

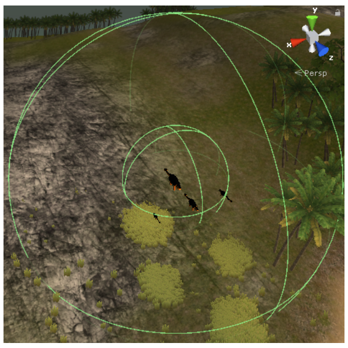


Figure 11: Herd Cenre and Radius'

3.4 Collision Detection

Collision Detections while in the attacking state causes the targets Health to reduce depending on the attack value of the Dinosaur. This is the cause of the Ankylosaurus and Velociraptor changing to the Dead state. It checks if the target is alive and reduces the health of the target.

Ankylosaurus::OnCollisionEnter::150

```
if (col.gameObject.tag == "Rapty" && State.CurrentState is Attacking && collision_time > collision_ticks && !(col.gameObject.G
        collision_time = 0.0f;
        col.gameObject.GetComponent < Velociraptor > ().Health -= Attack;
        Debug.Log(gameObject.name + " dealt " + Attack + " Damage to " + col.gameObject + ".");
}
```

Velociraptor::OnCollisionEnter::171

```
if (col.gameObject.tag == "Anky" && State.CurrentState is V_Attacking && collision_time > collision_ticks && !(col.gameObject.
        collision_time = 0.0f;
        col.gameObject.GetComponent < Ankylosaurus > ().Health -= Attack;
        Debug.Log(gameObject.name + " dealt " + Attack + " Damage to " + col.gameObject + ".");
}
```

3.5 AI Controllers Individual Stats

Each AI Controller has been attached Stats such as Health, Hunger, Thirst, Attack and Meat. The Health regenerates over time, Hunger and Thirst decay over time. Meat gets reduces while in the Dead State while a Predator eats the dead body.

Velociraptor/Ankylosaurus::Initialisation

```
public float Health = 100.0f;
public float Hunger = 100.0f;
public float Thirst = 100.0f;
public float Attack = 12.5f;
public float Meat = 100.0f;

[HideInInspector] private float collision_time;
[HideInInspector] private float health_time;
[HideInInspector] private float hunger_time;
[HideInInspector] private float thirst_time;

[HideInInspector] private float collision_ticks = 1.75f;
[HideInInspector] private float health_ticks = 4.75f;
[HideInInspector] private float hunger_ticks = 2.35f;
[HideInInspector] private float thirst_ticks = 3.5f;
```

Velociraptor/Ankylosaurus::Update

```
health_time += Time.deltaTime;
if (health_time > health_ticks) {
        var regeneration = (Hunger / 2 + Thirst / 2) * 0.03f;

        health_time = 0.0f;
        if (Health < 100.0) Health += regeneration;
        else Health = 100.0f;
}

hunger_time += Time.deltaTime;
if (hunger_time > hunger_ticks) {
        var decay = 0.75f;

        hunger_time = 0.0f;
        Hunger -= decay;
}

thirst_time += Time.deltaTime;
if (thirst_time > thirst_ticks) {
        var decay = 0.45f;

        thirst_time = 0.0f;
        Thirst -= decay;
}

collision_time += Time.deltaTime;
```

3.6 Future Development

- Optimising the A* path finding algorithm to use a int [,,] instead of creating gameObjects because of the performance costs.
- Decaying the meat value of Velociraptor and Ankylosaurus at death, emulating rotting.
- Mating between male and females creating offspring based on the variables and size of the parents.
- Pursue and Flee behaviours to include optional face and face away.
- Optimise wander to have a min and max y value.

4 References

Joey The Lantern (2017) *Unity 3D - Make a Basic AI State Machine*
Available at: https://www.youtube.com/watch?v=PaLD1t-kIwM (Accessed: 23/03/2018)

5 Bibliography

DinoPit (2018) *Ankylosaurus - DinoPit*
Available at: https://www.dinopit.com/ankylosaurus/ (Accessed: 23/01/2018)

DinoPit (2018) *Velociraptor - DinoPit*
Available at: https://www.dinopit.com/velociraptor/ (Accessed: 23/01/2018)