How to build a Boss in Unity with a good foundation to build upon.

Sprint 3

Kelsey van Engelenburg, R201

Inhoudsopgave

Self-Assessment

Overzicht van de relevante leeruitkomsten.

Persoonlijk leiderschap:
- Geen bewijs voor deze activiteit.
- Je bespreekt je motivatie & ontwikkeling en onderneemt actie indien nodig.
- **Je neemt verantwoordelijkheid voor ontwikkeling en studievoortgang.**
- **Je werkt voltijd aan je studie en laat dit terug zien in communicatiekanalen en je werk.**
- Je loopt vooruit op reviewmomenten door voortgang te bespreken met docenten en assistenten.
- Je werkt aan je ontwikkeling met behulp van een competentie ontwikkelplan (COP).
- **Je staat open voor feedback en reflecteert daarop.**
- Je neemt verantwoordelijkheid voor je handelen.
- **Je maakt verantwoord gebruik van AI-tools door ervoor te zorgen dat ze je leerproces niet belemmeren en door het gebruik ervan correct op te nemen in je bronvermelding**.
- Je oriënteert je op de arbeidsmarkt en vormt een beeld van mogelijke toekomstige rollen in de ICT, door o.a. de stagemarkt te bezoeken.
- Je bereid je voor op je stage door te werken aan een CV, portfolio en sollicitatiebrief.

Analyseren:
- Geen bewijs voor deze activiteit.
- Je voert een playtest plan uit om de gebruikersinteractie te evalueren.
- Je analyseert gameplay-data en testresultaten om inzicht te krijgen in de gebruikerservaring en technische prestaties van de game.
- Je test en valideert game-functionaliteiten met unit tests en andere methodieken om fouten vroegtijdig op te sporen.
- **Je evalueert bestaande code en architectuur om verbeterpunten te identificeren op het gebied van efficiëntie, leesbaarheid en onderhoudbaarheid.**
- Je onderzoekt hoe game mechanics en ontwerpkeuzes bijdragen aan de beoogde spelervaring en documenteert je bevindingen.

Adviseren:
- Geen bewijs voor deze activiteit.
- Je analyseert de resultaten van playtests en adviseert op basis hiervan over aanpassingen in gameplay en gebruikerservaring.
- **Je onderzoekt en beargumenteert de keuze voor programmeerpatronen, datastructuren en optimalisatietechnieken in de game-architectuur.**

Ontwerpen:
- Geen bewijs voor deze activiteit.
- Je kent verschillende ontwerpconcepten binnen game development en past deze toe in je ontwerpen waaronder 'win conditions', 'reward systems' en 'gameplay phases'.
- Je werkt features uit in design documentatie, presentaties en andere communicatie volgens passende (game) designmethodes.

- **Je beschrijft de code inclusief kwaliteitskenmerken met behulp van de SOLID principes en 'programming design patterns'**.
- Je gebruikt verschillende Unified Modelling Language (UML) diagrammen om de architectuur en werking van de software te beschrijven.
- Je stelt een styleguide op om de codeconventies te beschrijven en zorgt er samen met je team voor dat jullie code hieraan voldoet.

Realiseren:

- Geen bewijs voor deze activiteit.
- Je realiseert een gebruikersinterface met behulp van de Unity UI-toolkit.
- Je verbetert de gebruikerservaring van een spel door middel van 'juice' technieken.
- Je bewaakt het designdocument tijdens de realisatie van de game om consistentie en kwaliteit te waarborgen.
- Je selecteert en zet verschillende datastructuren correct in, zoals HashSet, Dictionary en Queue.
- **Je kunt een complexe 3D game ontwikkelen binnen de Unity game engine en werkt met features zoals monobehaviour, prefabs, scenes, mesh renderer en animation controllers.**
- Je kan snel je weg vinden binnen een bestaand project en hier aanpassingen in maken.
- **Je schrijft objectgeoriënteerde code die voldoet aan codeconventies, maakt gebruik van passende datastructuren en design patterns in C#, en past meer geavanceerde features van C# zoals generics, delegates, events en LINQ correct toe.**
- **Je beheerst de wiskunde die nodig is voor 3D game development zoals meetkunde en goniometrie.**

Manage & Control:

- Geen bewijs voor deze activiteit.
- Vastleggen van de verandering in het designdocument en issue-board tijdens het iteratieve proces.
- Ontwerpdocumentatie loopt, gedurende het project, voor op gerealiseerde game features.
- Je automatiseert het build- & publiceerproces van je game met behulp van CI/CD-pipelines.
- **Je waarborgt de kwaliteit van het project door te werken met merge requests die voorzien zijn van waardevolle (gericht op performance, datastructuren, design patterns, codeconventies, etc.) feedback.**
- Je gaat professioneel om met versiebeheer door te werken met een de 'feature branch workflow'.

Introduction

During this task, my goal Is to learn how to create a solid base for the boss so that our game can be built upon properly by implementing fitting design patterns and object structuring. Building this solid base will prevent a lot of refactoring and confusion when building further upon the boss.

Questions

How to build a Boss in Unity with a good foundation to build upon.?

- What are design patterns commonly used for boss enemies?
- How do I structure my code to be built upon dynamically instead of changing every script when something new gets added?

Methods

Peer review

- https://ictresearchmethods.nl/showroom/peer-review/
- I have the privilege of knowing a couple of well versed coders, one of which is a jack of all trades and the other has worked as a game developer himself, asking them to review my code can save me a lot of time trying irrelevant methods an can give me useful tips to use on my current code. Our project also has a minimum requirement of two approvals for every merge request, by discussing what I will be doing and getting peer reviewed by my teammates who have read up on my implementation as well I can see if I implemented my choices correctly.

Available product analysis

- https://ictresearchmethods.nl/library/available-product-analysis/
- Boss enemies are a commonly used concept in games, there is a good chance that I will be able to find an example that comes close to what I am trying to create and can look up and examine how parts of it work and if it follows the design pattern I want to use.

Design pattern research

- https://ictresearchmethods.nl/library/design-pattern-research/
- The best way to learn how to structure things is by learning design patterns, especially when you want to create something to be a good foundation to build upon.
-

Community research

- https://ictresearchmethods.nl/library/community-research/
- This is one of the best ways to find people that ran into similar problems, which saves a lot of research time. In a way, it's a peer review that you can look at as an outsider.

Results

Benoem welke verschillende mogelijke oplossingen / werkwijzes voor je ontwerp-/onderzoeksvraag je hebt gevonden door het gebruik van je onderzoeksmethode(s). Beschrijf welke oplossing / werkwijze je uiteindelijk gaat gebruiken en waarom je voor deze oplossing hebt gekozen.

From my research I acquired the following answers to my questions:

What are design patterns commonly used for boss enemies?

From my research I came across a couple of outcomes:

- Singleton
- Observer
- State Machines

The Boss that I am creating is a multi phase lava wyrm that has a couple of attacks that increase in difficulty per phase and has a set reset point. The most logical design patterns for me to follow in this project would be Singleton and State Machines, seeing how my boss doesn't really need observe the player or environment's actions due to it's set cycle.

Singleton

Singleton is a very good way to ensure that a game object is consistent throughout a project and also makes it easier to link scripts together and to access the right values without having to search for a certain component. One of the downsides that I kept seeing online is that it might be harder to debug and find where issues are coming from when working with a Singleton. When I weighed my options I came to the conclusion that the ease of use saves more development time than I would loose by debugging any eventual problems, especially if good, descriptive comments are placed into the code and if the Singleton has been planned beforehand, instead of changing and adding everything to it as a project advances.

Observer

The Observer pattern is also a very good pattern to use when making games, especially with Singletons. The Observer pattern can make it easier to change the internal workings of a Singleton without having to change the implementation everywhere that the changed or removed elements have been used. This can be done through Events in an event manager, which means that there is a centralized place you'd have to change instead of the entire projects code (Modular event triggers).

I however opted to not use the Observer pattern (yet) due to time constraints and the fact that it is something that will have to be implemented project wide. I will definitely bring it up to my team for the next sprint and will hopefully be able to implement it after.

State Machine

State Machines are almost always a perfect fit for game objects such as enemies, players and other game objects that can behave differently depending on the "current state" that they are in be that an enemy that is patrolling, a player that is low on health or hazards such as platforms that can crumble, respawn or break. I have decided to implement it onto my boss because it centralizes all the logic for the attack patterns of the boss and can make it behave more natural, instead of following a set list of things to do with no player feedback when it gets hurt.

How do I structure my code to be built upon dynamically instead of changing every script when something new gets added?

With my research I found that most game objects that have multiple components or children have a centralized script, with components such as movement and attacks being separate. I have found that this is also necessary to implement my Singleton and State machine properly

Something that is specific to my boss is that it will move with a snake like movement pattern through the air. Most of the examples that I could find were about ground snake like movement, but luckily this translates quite easily to my air movement. The core concept of all of them was that there is a tracking point that got followed by segments (usually capsules) that turn themselves using the direction they are moving in. this creates fluid snake like movement.
The tracking point also makes it easy to implement any movement scripts, as it will be the only point that has to be moved.

Another important part to keep in mind is that the arch movement can best be done using a Bezier curve, since a Bezier function can be molded to create the necessary curvature.

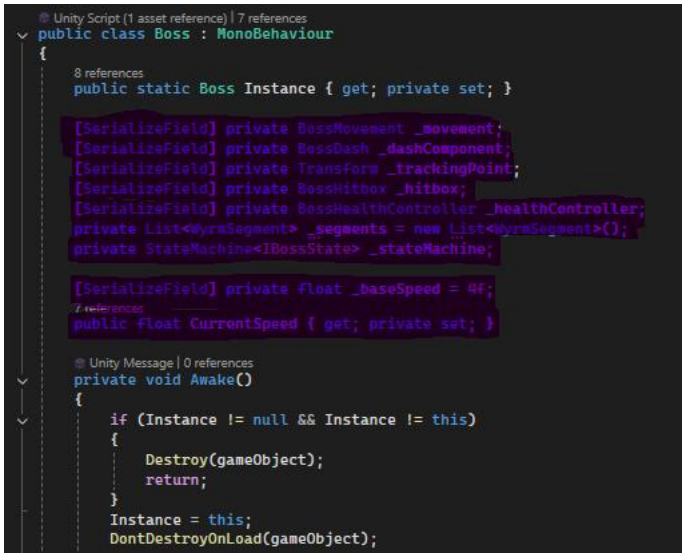
What will I use?

I have decided that I will create a boss with a centralized boss script and multiple components for things like movement, attacks, segments and hitbox. The boss(script) will be a Singleton. Alongside that I will build a state machine that the boss will use (also gets called in the boss script) with a base state that I can inherit from for the boss states states.

Execution

Creating the Boss Script

The most important part of the boss's foundation is the boss script. I started off by making the boss script a singleton so that all the functions and variables are publicly accessible.



As you can see in the above image I've created a public static Boss Instance that can be called publicly but not changed using {get; private set;}. This ensures that the boss instance can not be changed using other scripts.

Underneath that you can see that in awake the script checks if there is an existing instance using the getter and if there is one it destroys the one that this script has made. If no it doesn't find an Instance it sets the Instance to this Boss object.

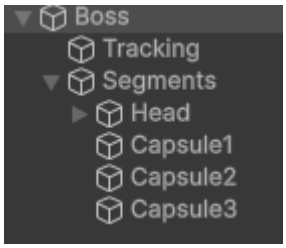
Underneath that is the DontDestroyOnLoad(gameObject) line that makes sure that the boss can persist through scene switching.

This code block is essentially the core of Singleton implementation.

After that, the core of the boss script has been created and it's time to make some components to implement.

The Segments

After creating the boss script it's time to make the boss itself and set it up so it can be easily maneuvered using the tracking point.



To achieve this, I started off stringing multiple capsules together on my boss and naming the top one head to make it easy to see which way the boss is facing using the inspector. After that, I created the tracking point

This is what the boss looks like now

Now it’s time to make the segments properly follow the tracking point

```
using UnityEngine;

Unity Script (4 asset references) | 5 references
public class WyrmSegment : MonoBehaviour
{
    [SerializeField] private Transform _leader;
    [SerializeField] private float _spacing = 1.8f;

    private bool _initialized = false;

    2 references
    public void SetLeader(Transform newLeader)
    {
        _leader = newLeader;
        _initialized = true;
    }

    Unity Message | 0 references
    private void LateUpdate()
    {
        if (!_initialized || _leader == null || Boss.Instance == null)
            return;

        // Follow speed dynamically synced with boss
        float _followSpeed = Boss.Instance.CurrentSpeed;

        // Vector toward the leader
        Vector3 moveDirection = _leader.position - transform.position;
        float distanceToLeader = moveDirection.magnitude;

        if (distanceToLeader > _spacing)
        {
            Vector3 directionNormalized = moveDirection.normalized;

            // Move toward leader
            transform.position = Vector3.MoveTowards(
                transform.position,
                _leader.position,
                _followSpeed * Time.deltaTime
            );

            //Segments look towards where it is going
            Quaternion targetRotation = Quaternion.LookRotation(directionNormalized);
            Quaternion adjust = Quaternion.Euler(90f, 0f, 0f); // Adjust X by 90°
            transform.rotation = Quaternion.Slerp(
                transform.rotation,
                targetRotation * adjust,
                _followSpeed * Time.deltaTime
            );
        }
    }
}
```

Pictured above is my WyrmSegment script

The SetLeader function is made for the boss script and will be explained later alongside the CurrentSpeed, for now the most important part to take away from it is that the leader is the transform of the segment above it, or the tracking point if it's the head.

In LateUpdate I do all the movement and turning of the segments, because this will happen after all other movement calculations in the scene. If the leader or boss hasn't been created yet it returns, which prevents null exceptions.

After that the segment gets the speed it should move at from the boss script and then saves the distance between the leader and its own position. It then checks if the segment is spaced further from the segment than the spacing between them (the tracking point also has the same spacing, this makes the boss move smoother and eliminates the need for another segment movement script.)

If the distance is greater, meaning that the leader has moved, it will normalize the moveDirection and then moves towards the position of the leader segment, after which the segment turns towards the leader by using the normalized distance to turn towards where it is going, with x being adjusted by 90 degrees due to the orientation of the capsule.

To make this work, I needed to have a current speed of the boss that was universal, luckily the boss script is a Singleton, making it easy to implement and adjust a variable such as CurrentSpeed.

```
[SerializeField] private float _baseSpeed = 4f;
7 references
public float CurrentSpeed { get; private set; }

1 reference
public void SetSpeed(float newSpeed)
{
    CurrentSpeed = newSpeed;

    if (_movement != null)
        _movement.SetMoveSpeed(CurrentSpeed);
}

1 reference
public void ResetSpeed()
{
    CurrentSpeed = _baseSpeed;

    if (_movement != null)
        _movement.SetMoveSpeed(CurrentSpeed);
}
```

To achieve this, I created a CurrentSpeed that can privately be set using a ResetSpeed and SetSpeed function, with SetSpeed being able to use any speed and ResetSpeed setting the boss back to its base speed, defined in _baseSpeed.

Alongside that being added to the player script, I made it so it automatically sets the leaders for the segments, and places them in a readonly list so other scripts can access them if needed.

```
[SerializeField] private Transform _trackingPoint;
private List<WyrmSegment> _segments = new List<WyrmSegment>();
 public Transform TrackingPoint => _trackingPoint;
public IReadOnlyList<WyrmSegment> Segments => _segments;
```

These are the needed variables for the implementation

```
if (_trackingPoint == null)
    _trackingPoint = transform.Find("Tracking");

  if (_segments.Count != 0)
      return;

  Transform segmentsParent = transform.Find("Segments");
  if (segmentsParent == null)
      return;

  foreach (Transform child in segmentsParent)
  {
      WyrmSegment seg = child.GetComponent<WyrmSegment>();
      if (seg != null)
          _segments.Add(seg);
  }
```

These were added to the Awake function

```
Unity Message | 0 references
private void Start()
{
    LinkSegments();
}

1 reference
private void LinkSegments()
{
    if (_segments.Count == 0 || _trackingPoint == null)
        return;

    _segments[0].SetLeader(_trackingPoint);

    for (int i = 1; i < _segments.Count; i++)
        _segments[i].SetLeader(_segments[i - 1].transform);
}
```

And this is how they get linked together

LinkSegments starts off with a null check, to prevent possible errors. After that it uses the SetLeader function to put the tracking point as the leader of the first segment in the list. After that, it iterates through all segments and puts the segment before it in the list as its leader using a for i loop.

Movement script

The most essential component for my boss is the movement script, this will be called by other scripts to move around and should include functions that can be called to move from point a to point b with a set pattern, be that a curve or straight line.

The most essential part of the movement will be moving in an arch, seeing how the boss will pop out of the lava and move around in the air for most of the moves. This can best be done using a Bezier curve

```
/// <summary>
/// A function translating the Bezier curve into code
/// </summary>
/// /// <param name="t">The point in the curve that gets returned (0-1).</param>
1 reference
private Vector3 Bezier(Vector3 a, Vector3 b, Vector3 c, float t)
{
    Vector3 ab = Vector3.Lerp(a, b, t);
    Vector3 bc = Vector3.Lerp(b, c, t);
    return Vector3.Lerp(ab, bc, t);
}
```

This is the Bezier function I ended up making to build arch movement on, it's a literal translation of the Bezier curve formula into code and moving it using Lerp.

```
/// <summary>
/// A function moving the object to its target in a curve
/// </summary>
9 references
public IEnumerator MoveToTargetArch(Vector3 targetPos, bool invertArch = false)
{
    Vector3 startPos = _rb.position;
    float archDirection = invertArch ? -1f : 1f;

    Vector3 controlPoint = new Vector3(
        (startPos.x + targetPos.x) / 2f,
        ((startPos.y + targetPos.y) / 2f) + (_archHeight * archDirection),
        (startPos.z + targetPos.z) / 2f);

    float distance = Vector3.Distance(startPos, targetPos);
    float duration = distance / GetCurrentSpeed();
    float elapsed = 0f;

    while (elapsed < duration)
    {
        float t = elapsed / duration;
        Vector3 bezPos = Bezier(startPos, controlPoint, targetPos, t);

        _rb.MovePosition(bezPos);
        elapsed += Time.deltaTime;
        yield return null;
    }

    _rb.MovePosition(targetPos);
}
```

This is the arch function I decided to create. It grabs the starting position and calculates the mid point of the curve using _archHeight and the target position. The arch can also be inverted by setting invertArch to true, this makes the height negative resulting in the arch going downward. The while loop makes sure that the rigidbody moves to the correct place in the Bezier curve by checking what position it should be in using the elapsed time and duration and inserting it into the t parameter of the Bezier function alongside the starting position, control point and target position.

```
/// <summary>
/// Moves boss in a straight line
/// </summary>
1 reference
public IEnumerator MoveToTargetLerp(Vector3 targetPos)
{
    Vector3 startPos = _rb.position;
    float distance = Vector3.Distance(startPos, targetPos);
    float duration = distance / GetCurrentSpeed();
    float elapsed = 0f;

    while (elapsed < duration)
    {
        float t = elapsed / duration;
        _rb.position = Vector3.Lerp(startPos, targetPos, t);
        elapsed += Time.deltaTime;
        yield return null;
    }

    _rb.position = targetPos;
}
```

The next function is quite straightforward as it just makes sure that the rigidbody moves to the proper position in between the starting position and the target position by using Lerp and a while loop structured the same way as the MoveToTargetArch function but instead of the Bezier function it uses unity's Lerp.

For both of these movement methods I referred back to my prototype and improved the arch function and built the Lerp one following the same logic.

The last thing I added is a reset function

```
/// <summary>
/// Gradually moves the boss back down to y = -20 using the same curved movement.
/// </summary>
2 references
public IEnumerator ResetBossPosition()
{
    Vector3 targetPos = new Vector3(0f, -20f, 0f);
    yield return StartCoroutine(MoveToTargetArch(targetPos, true));
}
```

This function can be called to set the boss to its default position. When I look back on it now I would set the starting position of the boss in initialization of the boss and refer back to that, so I will do that during my next Issue, instead of hard coding it.

Now to implement the component into the boss, I will add it onto the tracking point, as that is the part that will be moved.

```
[SerializeField] private BossMovement _movement;

if (_movement == null)
    _movement = GetComponent<BossMovement>();

public BossMovement Movement => _movement;
```

The boss loads the movement component in during the awake function, and other scripts can get the component's functions by calling it through the boss instance.

The way that the movement component has been implemented will be the way that the boss acquires every other component, such as the health system, any attacks, hitbox etc. this keeps it centralized within the boss Singleton.

The State Machine

With the base of the boss being done and ready to add components on to, an important step still remains, the implementation of the logic made for the boss. This will be done using a state machine, which I will now set up

```
public interface IState
{
    2 references
    void Enter();
    2 references
    void Update();
    2 references
    void Exit();
}
```

The first step is to make a state interface, this will be the base for all state types. It has a function for the logic when the object enters the state, an update function (also called Execute in some examples) that handles the things the object should do while in the state and an exit function, which contains the logic for when it exits the state

```
public interface IBossState : IState
{
    1 reference
    new void Enter();
    1 reference
    new void Update();
    1 reference
    new void Exit();
    1 reference
    void OnHurt();
}
```

After that, I created a IBossState that will be specifically used for boss states, which also has a OnHurt function which will be executed whenever the boss gets hit, as that should result in him switching states aswell.

```
public class StateMachine<T> where T : IState
{
    T currentState;

    1 reference
    public void SwitchState(T newState)
    {
        if (currentState != null)
            currentState.Exit();

        currentState = newState;
        currentState.Enter();
    }

    1 reference
    public void Update()
    {
        if (currentState != null) currentState.Update();
    }
}
```

After that, I created a StateMachine class, which has a generic typing. I chose generic typing as it allows me to lock states being passed to an object to be specifically of the type bound to the object, such as IBossState for the boss.

The state machine has a switchState function that triggers the exit function of the current state, sets the currentstate to the parameter state new state and then runs that state's enter function.

The state machine also has an update function that runs the update function of the state.

```
private StateMachine<IBossState> _stateMachine;
```

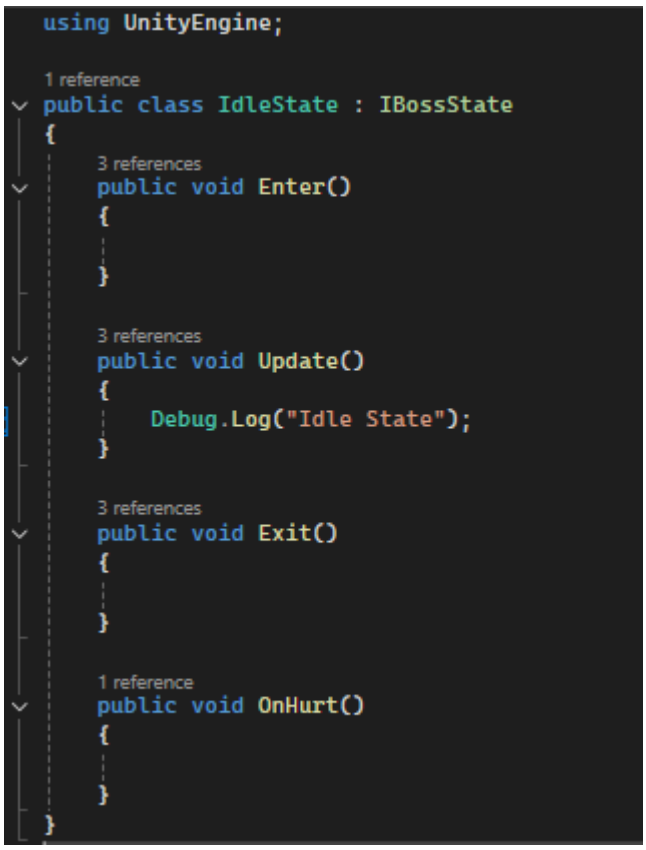
To implement the state machine, I added a StateMachine variable with the type being IBossState

```
_stateMachine = new StateMachine<IBossState>();
_stateMachine.SwitchState(new IdleState());
```

In awake, I create the state machine and switch it's state to Idle State

```
Unity Message | 0 references
private void Update()
{
    _stateMachine.Update();
}
```

In Update, the boss runs the Update function of the state machine.



```
using UnityEngine;

1 reference
public class IdleState : IBossState
{
    3 references
    public void Enter()
    {

    }

    3 references
    public void Update()
    {
        Debug.Log("Idle State");
    }

    3 references
    public void Exit()
    {

    }

    1 reference
    public void OnHurt()
    {

    }
}
```

And this is a state I created to see if it works, and to set as the base state before the states get made

Validation

Usually I make feature scenes to test my work, which I did for the movement. However, I deleted these scenes before we received this assignments so I don't have visuals for this part.

To test the movement, I made a feature scene where the boss jumps out of the lava in an arch, and sinks back in. I did this using the MoveToTargetArch function and preset points in the scene. This worked smoothly.

After that, I created a dash attack, added it to my boss and made a feature scene for it. This also worked without issues.

To simplify the dash attack I made the ResetPosition function, which also had its own feature scene that worked smoothly.

With these test I could conclude that my code worked, but it's also important that my code is clean and follows my design patterns and code convention.

To begin with, I asked my partner that has coding knowledge across a variety of different fields to look at my code, which made me change a couple of things and made me think critically about what I had made and with that comes a better understanding of my own code as well.
Funnily enough, after I made my Singleton, the workshop about data structures and algorithms took place. This was an extra check to see if I fully understood what I had made. The workshop also helped me start with my state machine.

As to the sources I used, a lot of them have been lost when it comes to the movement and singleton, as those were made before the assignment.

For the Singleton, I was luckily able to find the following sources:

https://gamedevbeginner.com/design-patterns-in-unity-and-when-to-use-them/
this source was also helpful for the Observer pattern research, especially the part about how well it works with Singleton.

https://gamedevbeginner.com/singletons-in-unity-the-right-way/

When it comes to the state machine however, I found one source that really put me on the right track:

https://discussions.unity.com/t/c-proper-state-machine/613267/2

as well as this one:

https://gamedevbeginner.com/state-machines-in-unity-how-and-when-to-use-them/#interface_fsm

these really helped me understand *how* to implement a state machine in unity instead of how the flow should go, such as using a state interface and a state machine class to call in my object/manager scripts.

I also use ChatGPT to look up the names of concepts that I can explain in words, but don't know the name of, or to write example code sometimes if I really struggle to find example code of a concept I don't quite grasp yet. Whenever I do this, I look up individual terms and concepts it gives me to verify the authenticity. I personally use it more like a peer than a coding tool.

While I usually delete ChatGPT logs quite often, I kept the ones I used for my research when we got the assignment, and will put the prompts in my sources.

Conclusion

My questions for this research were

How to build a Boss in Unity with a good foundation to build upon.?
- What are design patterns commonly used for boss enemies?
- How do I structure my code to be built upon dynamically instead of changing every script when something new gets added?

The design patterns I found were without a doubt some of the most important design patterns in game development and lead me to create something that someone outside of my project with proper game development knowledge could also work with little to no explanation.

Discussion

I am very happy with the choices I made, as it has been a breeze to work on top of my current code. It's well structured and easy to modify.

As I talked about before however, I would like to use the observer pattern as well, as that can prevent a lot of debugging issues later on.

I also wish that I kept more of my sources throughout my coding process, especially ChatGPT prompts as that's the biggest part of my research that I am missing. This research was made after a big chunk of my code had already been made.

I would also like to experiment with different research methods in the future, as these are usually my go to research methods. For this project however, I think that the methods I chose gave me very good results, and that other research methods might've been a bit overkill for this issue.

Bronnen

https://discussions.unity.com/t/c-proper-state-machine/613267/2

https://gamedevbeginner.com/state-machines-in-unity-how-and-when-to-use-them/#interface_fsm

https://gamedevbeginner.com/design-patterns-in-unity-and-when-to-use-them/

https://gamedevbeginner.com/singletons-in-unity-the-right-way/

ChatGPT prompts:

How to use a generic to enforce the usage of an IState type in the context of a state machine class?

how to set up a state machine in unity