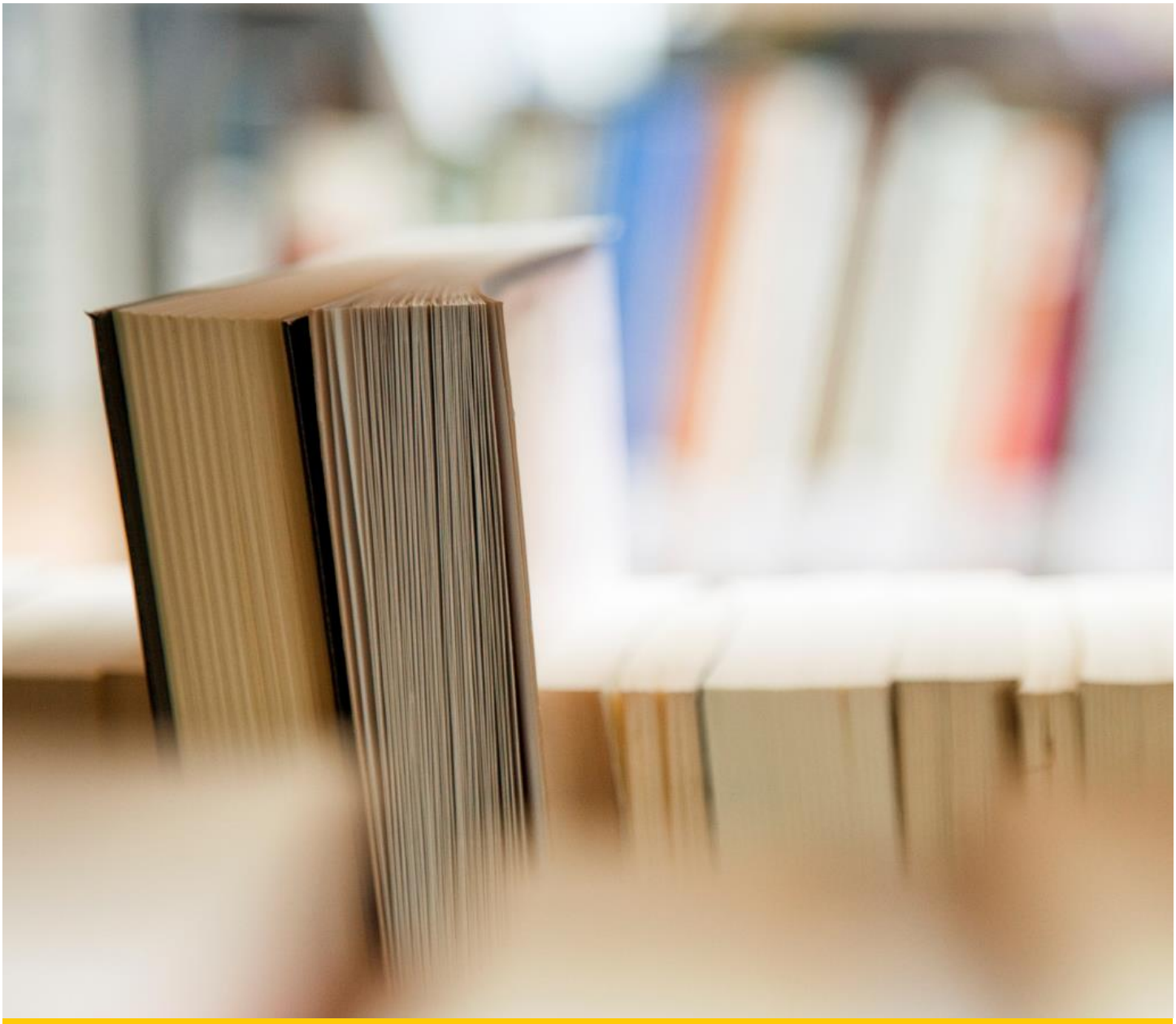

Process Journal

Sushanth Panikkar 11/2/18 28204344

CONTENTS

Contents

This is my project and process journal entry. There are elements of both in some entries. I have headed different parts as 'PROCESS' where I think content relates to research, brainstorming and development. I have headed other parts as 'PROJECT' where I think content relates our group interactions and decisions that impact on the progress of the project itself.

WEEK 8

Week 8 Project

This week was dedicated to our preps for the alpha playtest coming up in Week 9. Putting together our individual tasks into one solid Unity project, we realized that things didn't go exactly as planned. But we saw that coming as we had intentionally overloaded the week to see how much we could accomplish. That is why we saved the last 4 weeks as 'buffer' weeks to catch up on any work we were unable to complete previously. The problem we were currently facing was to make sure we had enough playable content to present during the playtest. It wasn't like we hadn't done anything, it's just that most of the segments had bugs or other minor issues that needed to be fixed or updated before testing. We wanted to have 2 bosses fully functional as well as all the other challenges the player had to overcome to reach the first boss stage for the playtest. However, in the current situation, we only had one functional boss while the other was broken, with no other challenges leading to the boss as planned. We also had the added the player's ability to switch between cat, bird and fish which is activated in later stages. Due to lack of time (since we barely had a week to add everything we initially wanted), there was a slight change of plans. We decided that we'd have two different cameras – one for the player character and the other fixed at the Big Fish (Boss 1) area. During the playtest, we will start off with the fixed camera at the boss and once the player defeats the boss, we manually swap out the cameras (which will later be done by code, this version was only for the playtest).

Week 8 Process

To fix the issues previously mentioned and get the sample version ready for playtesting, I started off with researching about all the possible way I could fix this and alternative methods I could use temporarily. I found some links on the Unity forums[1] which explain how you can use camera variables in unity scripts to get access to the different cameras in the scene. But for some reason, I could not swap out the cameras. If I had both cameras active, Unity would choose one of the two cameras and set that as the main camera, and if I choose to disable one, I couldn't re-enable it unless I had access to it through code, which would make both cameras too interdependent for my liking (as one script could not exist without the other). Coming across all these problems, I dropped the idea of making a temporary fix (for the playtest) as I didn't want to commit a bad fix that would overcomplicate the future code.

WEEK 9

Week 9 Project

This week, after receiving feedback from our alpha test, we decided to incorporate the changes we noticed were frequently pointed out by the testers. However, most of the points which were mentioned, were things we had only implemented as a temporary fix to allow a working prototype for the alpha test. This included the use of multiple cameras and the not so self-explanatory UI. We also further discussed how we could take a few more steps to ensure the game would be ideal for our target audience, which led us to come to the conclusion that we would need a UI which would not clutter the screen and ideally would not require any explanation to understand. As for the camera changing in the boss areas, I took it up to figure out a solution for the same, mainly because I was the one who made the initial camera so it would save some time and effort to make any necessary changes to the camera class to incorporate our idea of having a stationary camera at the boss levels. With this, we decided to have Liem work on the UI, Andrew work on the bosses (as per schedule in the google sheets we made) and I work on figuring out a good fix for the camera issue.

Week 9 Process

Fixing the camera took me longer than I would like to admit. However simple the fix appeared to be logically, I just could not figure out an efficient code to do so without making the classes too inter dependent on each other. Ideally, the plan was to have multiple cameras, and swapping the camera being used when a certain trigger area was touched by the player. After more trials, none having any sliver of success, I decided to take a break from coding, hoping that a fresh mind would enable me to come up with a good solution next week.

WEEK 10

Week 10 Project

After a break during the mid-semester, we resumed our work on the project individually. We checked up on each other and discussed our individual process. Since most of our work was going according to the schedule we made, we decided to continue our work individually for this week, using google drive for version control every time one of us made a major update in the game.

Week 10 Process

Continuing where I left off, I had two ideas to work around the camera issue. My first solution was to use a single camera which would be zoomed out far enough from the beginning so that would eliminate the need for another camera altogether. However, upon implementing this and playtesting it with my group, we came to the conclusion that this caused issues during the boss fight as the camera jittering around was too distracting for a boss fight. Before implementing my second solution (which was a similar concept as the first one, just a more detailed way of doing it), I had an idea – as my current camera follows the player by making it the target, I could create a trigger in the Boss area where the target is changes to a fixed point in space. Here is an updated snippet of the code doing so:

```
public class BossCameraSwapper : MonoBehaviour
{
    cameraFollow2DPlatformer cref;
    public int zone;

    void Start()
    {
        cref = GameObject.Find("Main Camera").GetComponent<cameraFollow2DPlatformer>(); //get reference to the camera following player script to change its current target
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if(other.CompareTag("Cat") || other.CompareTag("Fish") || other.CompareTag("Bird"))
        {
            cref.setInBossArea(true,zone); //set the boolean flag for being in boss area in order to change the target to a fixed point
        }
    }

    void OnTriggerStay2D(Collider2D other)
    {
        if (other.CompareTag("Cat") || other.CompareTag("Fish") || other.CompareTag("Bird"))
        {
            cref.setInBossArea(true,zone); //ensure that as long as the player stays in the boss area, the target is the fixed point
        }
    }
}
```

Fig 10.1 Checks when player is in the boss area and flips a Boolean in the camera script to change the target

```
else if (inBossArea) //if player is in the boss area,
{
    if (areaNo == 1) //area one is the fish boss
    {
        target = GameObject.FindGameObjectWithTag("FishBossCamera").GetComponent<Transform>();
    }
```

Fig 10.2 Swaps the target to a pre-determined point in space representing the boss area. (an empty game object in unity editor with a transform component does the work)

```
public class BossCameraExit : MonoBehaviour {
    cameraFollow2DPlatformer cref;

    void Start()
    {
        cref = GameObject.Find("Main Camera").GetComponent<cameraFollow2DPlatformer>();
    }

    void OnTriggerEnter2D(Collider2D other)
    {

        if (other.CompareTag("Cat") || other.CompareTag("Fish") || other.CompareTag("Bird"))
        {
            cref.setInBossArea(false,0);
        }
    }

    void OnTriggerStay2D(Collider2D other)
    {

        if (other.CompareTag("Cat") || other.CompareTag("Fish") || other.CompareTag("Bird"))
        {
            cref.setInBossArea(false,0);
        }
    }
}
```

Fig 10.3 Checks when player is out of the boss area and flips the Boolean to ensure the camera script knows about the event.

WEEK 11

Week 11 Project

To start off this week, I added my version of the fixed camera incorporated with our latest version and reuploaded it to the google drive. Liem worked on all the HUD photoshop required, making a new interface which we all loved instantly. However, as it was nearing week 12, the stress of other units had caught up to all of us so we did not do how much work we initially planned to do. For starters, the minions had not been coded yet (even though they are derived from the boss attack mechanics, it had to be a separate script), most of the Narwhal boss wasn't ready yet including the animations and assets. But that's exactly why we left the last 4 weeks as Buffer weeks, we anticipated that by week 12 work would slow down as compared to when we initially started.

Week 11 Process

I began my work this week by planning out exactly how I would handle the minions code. The way I saw it, I had two choices, either stick to the plan and do only boss derived attacks OR introduce new attack mechanics for minions. This week was dedicated purely to researching the different ways of making enemies in unity, whether it be using complex AI or simple hard coded logic. Some videos which gave me a much better understanding of the flow of unity engine and the way its pipeline was Casanis's obstacle tutorial[2] in the platformer series as mentioned earlier which helped me understand how even obstacles would be considered "enemies" and how a simple one could be made, which I could adapt into our game. Another video I found really helpful was to handle movement of the enemies, a youtuber by the name of Devin Curry had an interesting tutorial[3] of a slime like enemy moving in a platform. Although Devin's tutorial would not directly be of much help to me, it made me realise the importance of rigidbody2D[4] of unity and the power it had to enhance character movements very easily which might come into use later in the future. I did a bit more extensive research on enemy movement, but I didn't feel anything else other than these two really contributed to my knowledge so I did not feel the need to mention them.

WEEK 12

Week 12 Project

Week 12 was to be the beta test of our game. However, I had a test around the same time as our class, hence I had to leave the handling of the beta test to Andrew and Liem, who were completely understanding about my circumstances. I had made a mock up in paper of how I wanted the minion script to be handled and told them my idea in our group chat. Since I had no real major changes in the game, I explained to Andrew how the boss camera code worked, so he could add it in for the bird and narwhal bosses too for the beta test. From the remarks I received from my team, our beta test went well with not too much information we didn't already know about, so we went according to our usual plan to have everything finished by week 13 – mid week 14 to leave a couple of days for bug fixing and memory management.

Week 12 Process

As mentioned earlier, due to the test I had this week, I had not much work done except for planning out how to handle the minion script. One major change I had planned was to have all the logic in one script rather than having a different script for each type of minion we make. Looking back in our code, I feel like our perception of c# scripting had greatly improved, and we could anticipate problems before they even occurred.

WEEK 13

Week 13 Project

At this point, since our individual unity projects were getting more complex, we decided to collaborate all our work into one project to set as a base. We used the beta test unity build as a base, however I downloaded the file earlier to that and kept working on that without realising. I figured, it would be best to finish making the minion script in this version and then incorporate it to the latest version at the time of finishing.

Week 13 Process

Reading about the header attribute[5] in unity, which made different sections for a script in the inspector, I figured this would be the perfect way to make all my minion logic in a single script. For our game this semester, we decided to have a basic 4 types of minions, a jumping fish, a spitting fish, a egg bomb dropping bird and a swooping bird. This week I made the base for the minion script and completed the logic for the jumping fish and spitting fish.

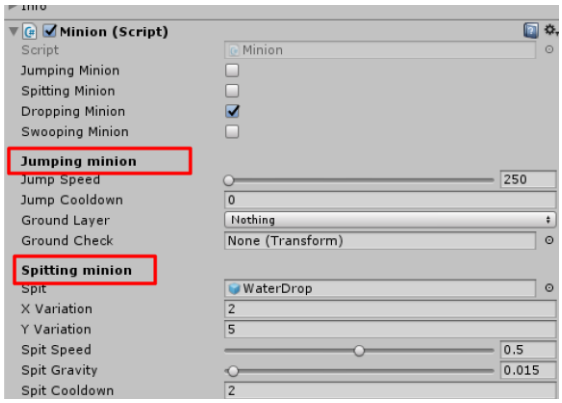

Fig 13.1 Using the header attribute in unity to separate the minions.

However, there was certain bugs I had to fix before I was ready to upload this as a new build, so I let Andrew and Liem fix the narwhal assets and animation before incorporating my script. Instead, I decided to fix the bug I had where the fish wouldn't stay in the water and would fall through the ground, this seemed to only happen to the jumping fish. After an intensive 4 hour debugging spree, I found the root for the jumping fish falling through the ground. It was the most simplest mistakes but least expected one, I had the collider set to be a trigger and hence no physical collisions were done for it. Frustrated, I fixed the bug and left the script for this week as is, deciding to do the bird minions the following week.

WEEK 14

Week 14 Project

With 90% of what we had planned on doing done, I uploaded the script with the unity project I had on 26th October as a final minion script (with the bird egg dropping minion logic working as well). Since Andrew had worked on the boss scripts, he had a better idea of how the attack mechanism worked, and I couldn't figure out how to get the swooping attack working so I delegated that task to Andrew as this was the last task we needed to do to complete the game as we had promised. However, Andrew did point out an interesting flaw, with the minions always being active, addition of multiple minions caused the game to greatly degrade in performance. The fix was a simple one, which Andrew had already done for the bosses, where they are inactive until the player reaches a certain area around them, which triggers them to awake.

After this upload on 26th October, Liem uploaded a new version the next day with a lot of assets changes, so I downloaded this version and redid the changes I mentioned above and re uploaded as a new base to use, all that was left was for Andrew to do the swooping bird logic and Liem to place the minions in the game as we had decided.

Having done this, we would finally be ready to present a prototype of Grim Catastrophe.

Week 14 Process

This week was a very busy week in terms of the amount of coding I had to do. I added the bird dropping logic to my minion script, which had some complications. Unlike the spitting fish which used a separate script to spit water, the dropping bird had all the logic in the boss script itself. This made it difficult to change the speed of the droppings if needed, so I added a setSpeed function in the bird bomb script which Andrew had worked on for the bird boss. I also noticed that somewhere in the versions, the player stopped checking for collisions properly, so I altered all player controller scripts to check for projectile collision with colliders and triggers. (the boss projectiles are trigger colliders so hence the change). I also had a fully functional minion script for 3 out of 4 minion types (all except the swooping bird). Here are snippets of the final minion script I made:

```
void Update () {
    if (JumpingMinion)
    {
        if(Time.fixedTime > jumpTimer)
        {
            jumpTimer = Time.fixedTime + jumpCooldown; //reset the cooldown
            myRB.velocity = Vector3.zero;
            myRB.AddForce(new Vector2(0.0f, jumpSpeed));
        }

        checkGrounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, groundLayer);

        if (myRB.velocity.y > 1 && !checkGrounded) {
            transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.Euler(0, 0, -90),0.2f);
        } else if (myRB.velocity.y < -1 && !checkGrounded) {
            transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.Euler(0, 0, 90),0.2f);
        } else if (checkGrounded) {
            transform.eulerAngles = new Vector3 (0, 0, 0); // Can't lerp for reset on ground, otherwise it bugs out or changes the minions x position
        }
    }

    if (SpittingMinion)
    {
        if(Time.fixedTime > spitTimer)
        {
            myAnim.SetTrigger ("spit"); // LV
            spitTimer = Time.fixedTime + spitCooldown;
            GameObject tempSpit = Instantiate(spit, transform.position, transform.rotation);
            tempSpit.GetComponent<FishSpit>().setTarget(new Vector3(transform.position.x + xVariation, transform.position.y + yVariation, transform.position.z), spitSpeed, spitGravity);
        }
    }

    if (DroppingMinion)
    {
        if(Time.fixedTime > dropTimer)
        {
            myAnim.SetTrigger ("drop"); // LV
            dropTimer = Time.fixedTime + dropCooldown;
            GameObject tempBomb = Instantiate(bomb, transform.position, transform.rotation);
            tempBomb.GetComponent<BirdBomb>().setSpeed(dropSpeed);
            tempBomb.GetComponent<BirdBomb>().setDirection(direction);
        }
    }
```

Fig 14.1 Logic of the minion script

```
    public bool JumpingMinion;
    public bool SpittingMinion;
    public bool DroppingMinion;
    public bool SwoopingMinion;
    Rigidbody2D myRB; //rigidbody component of the minions

    //jumping minion variables
    [Header("Jumping minion")]
    [Range(250, 1500)]
    public float jumpSpeed; //players max jump height
    public float jumpCooldown; // The initial value for the jumping time in milliseconds
    float jumpTimer; //for jumping minions to jump in a specified interval
    // LV - need ground check to properly rotate minion when jumping
    float groundCheckRadius;
    public LayerMask groundLayer;
    public Transform groundCheck;
    bool checkGrounded;


    //spitting minion variable
    [Header ("Spitting minion")]
    public GameObject spit; //reference to the gameobject for spitting
    public float xVariation; //variation in x axis for the spit
    public float yVariation; //variation in y axis for the spit
    [Range(0, 1)]
    public float spitSpeed; //speed of the spit
    [Range(0, 1)]
    public float spitGravity; //gravity the spit if affected by
    public float spitCooldown; //The initial value for the spitting time in milliseconds
    float spitTimer; //for spitting minions to spit in a specified interval

    //dropping minion variable
    [Header("Dropping minion")]
    public GameObject bomb;
    [Range(-1, 1)]
    public int direction;
    [Range(0, 1)]
    public float dropSpeed;
    public float dropCooldown;
    float dropTimer;
```

Fig 14.2 Separation of the different variables for each minion using the header tag

REFERENCES

[1] https://answers.unity.com/questions/16146/changing-between-cameras.html

[2] https://www.youtube.com/watch?v=h6NOn2Uj6Mc

[3] https://www.youtube.com/watch?v=LPNSh9mwT4w

[4] https://docs.unity3d.com/ScriptReference/Rigidbody2D.html

[5] https://docs.unity3d.com/ScriptReference/HeaderAttribute.html