Game Physik Dokumentation (Shrinkless Planet)

Structure:

1. **Project Dokumentation**
   a. Requirement
   b. Software used
   c. Tasks planning and implementation
   d. Challenges and solutions
   e. Post mortem

2. **Code documentation**
   a. Functionality
   b. Method
   c. Presentation of the most important functions

AnisProduction2020 (kontakt@anhammami.com)

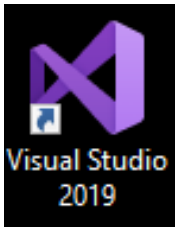
Project Dokumentation

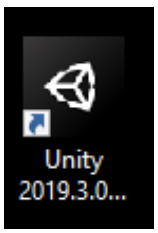
1.a.

The game requirement was about to build a world where physics are applied more likely similar to our real world. Our character named Billy would surface the planet with initiative force constant forward. In the mine while some enemies inform of bugs would be spawned around the planet too, with same Physics applied for example (Gravity, Wind, etc.). Two of the Nord & South Polar are checkpoints to lunch the Shrinking system on. Blue for expanding and Red for the exact opposite. Which means attributes for example Mass, Distance and Speed will drop or raise depends on real physics algorithm.

1.b.

In this project, I will be using Unity3d personal as main engine core for the game and Microsoft Visual Studio as code reader.





1.c.

Build a map in form of a planet. Add an attraction field called Gravity. Create a player to interact with the world under the control of physics around. Give a collectable mission for the level as a motivation.

1.d.

(1) The more complexity the form of the object was forming, the more plays center of gravity one big deal.

The solution was to use simple forms of shapes like cylinder and cycle.

(2) Using a simulator program would accidently run a loop without being noticed correctly. For example, two different shapes having collision and can also fuse surface between the two of them. Which is not incorrect form of material.

The solution was to include some exception functions to intercept null values.

1.e.

The goal with these physics mechanics would be implemented in any other game and get different types of level designs fits any category.

Code documentation

2.a.

*Create movement for the player with smoothly follow with the camera. (PlayerController.cs – SmoothFollow.cs)

```
1 reference
void PlayerMove()
{
    rotation = Input.GetAxisRaw("Horizontal");
    rb.MovePosition(rb.position + transform.forward* moveSpeed * Time.fixedDeltaTime);
    Vector3 yRotation = Vector3.up * rotation * rotationSpeed * Time.fixedDeltaTime;
    Quaternion deltaRotation = Quaternion.Euler(yRotation);
    Quaternion targetRotation = rb.rotation * deltaRotation;
    rb.MoveRotation(Quaternion.Slerp(rb.rotation, targetRotation, 50f * Time.deltaTime));
    //transform.Rotate(0f, rotation * rotationSpeed * Time.fixedDeltaTime, 0f, Space.Self);

}
```

```
1 reference
void SmoothlyFollow()
{
    if (target == null)
    {
        return;
    }

    Vector3 newPos = target.TransformDirection(offset);
    //transform.position = newPos;
    transform.position = Vector3.SmoothDamp(transform.position, newPos, ref velocity, smoothness);

    Quaternion targetRot = Quaternion.LookRotation(-transform.position.normalized, target.up);
    //transform.rotation = targetRot;
    transform.rotation = Quaternion.Lerp(transform.rotation, targetRot, Time.deltaTime * rotationSmoothness);

}
```

*Spawner for the bugs, which initiate them randomly around the map. (BugSpawner.cs)

*Starts bugs with an initiate force to move and give bonus when catched.(BugBrain.cs)

```
private void OnCollisionStay(Collision col)
{
    //Debug.Log(col.transform.name);
    if (col.transform.name == "Player")
    {
        info.GetComponent<InfoUI>().addScore();
        Destroy(this.gameObject);
    }

}

1 reference
void StartForce()
{
    Rigidbody rb = GetComponent<Rigidbody>();

    rb.AddForce(startForce, ForceMode.Impulse);
}
```

*Create a HUD to display world’s infos and controls. (InfoUI.cs)

```
void Update()
{
    text.text = Planet.Score.ToString("0.#") + "m (distance)";
    texts.text = (126f/Planet.Score).ToString() + "m/s (speed)";

    rt.anchoredPosition = Vector2.Lerp(Vector2.zero, startPos, Planet.Size);


    ENDGAME();

}

1 reference
public void ENDGAME()
{
    if (Input.GetKeyDown(KeyCode.Escape))
        Application.Quit();
    if (Input.GetKeyDown(KeyCode.R))
        SceneManager.LoadScene(0);
    if (Input.GetKeyDown(KeyCode.Space) || Input.GetKeyDown(KeyCode.P))
    {
        if (Time.timeScale == 1.0)
        {
            Time.timeScale = 0.0f;
            endGame.text = "PAUSE <SPACE>";
        }

        else
        {
            Time.timeScale = 1.0f;
            endGame.text = "";
        }

        if(endGamebool)
        {
            SceneManager.LoadScene(0);
        }
    }
}
```

2.b.

ShrinkSpeed

```
private static Transform myTransform;

public float shrinkSpeed = .05f;

0 references
void Awake()
{
    myTransform = transform;
}

0 references
void Update()
{
    if (transform.localScale.y >= 4.0f || transform.localScale.y <= -4.0f)
        shrinkSpeed = -shrinkSpeed;
    transform.localScale *= 1f - shrinkSpeed * Time.deltaTime;
}
```

SpawnBug

```
2 references
IEnumerator SpawnBug()
{
    if (count < 20)
    {
        Vector3 pos = Random.onUnitSphere * distance;
        Instantiate(bugPrefab, pos, Quaternion.identity);
        count += 1;
    }
    yield return new WaitForSeconds(1f);

    StartCoroutine(SpawnBug());

}
```

3.c

Attractor

```
foreach (Attractor _body in Bodies)
{
    if (_body == this)
        continue;

    float m1 = rb.mass;
    float m2 = _body.rb.mass;

    float r = Vector3.Distance(transform.position, _body.transform.position);

    float F_amp = (m1 * m2) / Mathf.Pow(r, 2);
    F_amp *= GRAVITY_CONST;

    Vector3 dir = Vector3.Normalize(_body.transform.position - transform.position);

    Vector3 F = (dir * F_amp) * Time.fixedDeltaTime;
    //Debug.Log ("Force: " + F);
    rb.AddForce(F);

}
```

RotateBody to face UP

```
void RotateBody(Rigidbody body)
{
    Vector3 gravityUp = (body.position - transform.position).normalized;
    Quaternion targetRotation = Quaternion.FromToRotation(body.transform.up, gravityUp) * body.rotation;
    body.MoveRotation(Quaternion.Slerp(body.rotation, targetRotation, 50f * Time.deltaTime));
}
```

Statutory declaration

I hereby declare an oath that I have done the work independently and without using any other tool than the one specified. The thoughts taken directly or indirectly from external sources are identified as such.

The work has so far not been submitted to any other examination authority in the same or similar form and has not yet been published.

Berlin, am 1.15.2020

Signature