Steve GRAPHOS

CAPSTONE

Mobile Level Demo

Fall 2014

Prof. POLLAK

DOCUMENTATION

Project Summary

My project was to create a single level game demo. Its technical limits would be similar to those found in the mobile gaming market (Nintendo 3DS, smart phones, and tablets). The art style was to be comparable to games like Pokémon, Bravely Default, and DOTA. Each asset was to be under five thousand triangles with a single diffuse texture 256x256 pixel or smaller.

Game mechanics that were to be included in the demo are: a health system for the player and the enemy, a melee system and attack animation, player and enemy movement with animations, a camera to follow the player, a few sound effects, and a basic enemy AI.

The final deliverable was an .exe file containing all progress made on the demo. Source files were provided in case of loading issues.

Target Audience

The target audience for the demo are mobile gamers that are interested in hack ‘n slash games. If the project were a finished game, a more focused target audience would include mobile gamers that: enjoy hack ‘n slash and are curious about or want to try role-playing games (RPGs), but are possibly intimidate or confused about the game mechanics that are part of the genre.

Design Problem

The design problem was to make a working demo that would set up some of the core mechanics that would be featured a full game before moving onto the RPG elements at a later date.
The idea for the full game is that it would make RPGs more accessible to an inexperienced audience, and also be good enough for core gamers to enjoy as well. The full game would be a hack ‘n slash with an introductory amount of RPG elements. This kind of game is great for the mobile market, it presents the fun of hack ‘n slash and RPGs without the overbearing menu systems and user interface that they usually bring.

Objectives

The main objective of the demo was to develop some of the simpler game mechanics (described in the Project Summary and Asset List), act as a preview to the art style, and generate interest in the project. I needed to create an asset list and environment that would save time and make sure everything would work by the deadline. I developed a simple theme for the demo that, if finished would, support the reuse of assets.

Theme

The player is on top of a basalt formation, possibly to collect a quest item when he sees the enemy at the other end. He looks made of stone and is positioned by four strange pillars not made from the surrounding rock. The enemy does not advance towards the player immediately and stays near the pillars. His appearance reminds the player of himself, but corrupted by something unknown.

A stretch goal for the demo was that if the player was killed he would re-spawn and find a copy of his body resting against a pillar each time he dies. If the player killed the enemy, it would respawn and also have a copy of itself resting against the pillar. Once all pillars had a “sacrifice”, the game would end.

Schedule

My schedule for Capstone initially seemed sufficient, and probably would have been for a more experienced developer. I separated the schedule into three equal sections: Modeling and Rigging, Texturing and Animation, and Scripting. When I presented my schedule, the class suggested scripting as I went so I would have more time to fix any of the issues that might appear.

Asset List

-Player / Enemy
  +Weapon

-Animations
+Idle
+Walk
+Attack

-Environment
  +Wall
  +Floor
  +Pillar

-Sound Effects
+Ambiance
+Walk
+Swing

Research

While researching for programming tutorials and examples for my demo's game mechanics, I came across three major sources that would help me script in Unity: Brackeys, BurgZerg Arcade, and the Unify Community Wiki. I also looked into Digital Tutors for a while but stopped after my trail ran out, since I felt I was making better progress with different sources.
Most of my reference images come from DeviantArt, Pinterest, and basic searches on Google Images.
While searching for reference material, I kept in mind that my assets needed to be modular: the player and the enemy would be the same model with different textures; the pillar's model would be different enough on each side so a simple rotation would break up the repetition of using it multiple times; the environment would need to be made of a material that is naturally symmetrical or gridded, to justify its repetition and simplicity.

The Character

The design behind the player / enemy is a knight. While building reference images I realized that most concept art of knights follows historical reference. The characters are in full armor usually with a layer of chain mail or padded clothing, following the basic idea behind armor: layers, layers, layers. While this isn't a bad thing, it really only allows you to design what it looks like, and not what pieces of armor the character would wear. The result makes that character slow, heavy, and gives him limited movement in the arms and legs.

I wanted my character to feel as light as possible and let the majority of the weight come from the sword. His armor should be simple, allow for a greater range of movement, and still protect the key areas of the body.

I eventually stumbled upon a picture of the main character from Dragon Age 2 and liked the lightweight design of his armor. While his armor's harsh angles allow for weapons to glance off more easily, they also make the character seem more aggressive. I wanted to make something more neutral that could fit both the player and enemy.

The Character PT2

Combining this idea with other reference, I made a softer silhouette to the individual pieces and also gave them a more rounded curve, allowing the armor to work for both the player and the enemy. I added an extra chest plate segment to protect the stomach and a set of cuisses to cover the outer thigh. I decided to place spikes on the enemy's armor to give it a darker look, and give him a slightly different silhouette than the player.

In retrospect, I might have been able to make a more angular armor design for the enemy by removing some of the topology after finalizing the model's unwrap. The first image on the left is the basic sketch I put together for modeling. This design was later changed when a classmate mentioned that it might be too thin for the low-poly limitations I was going for.

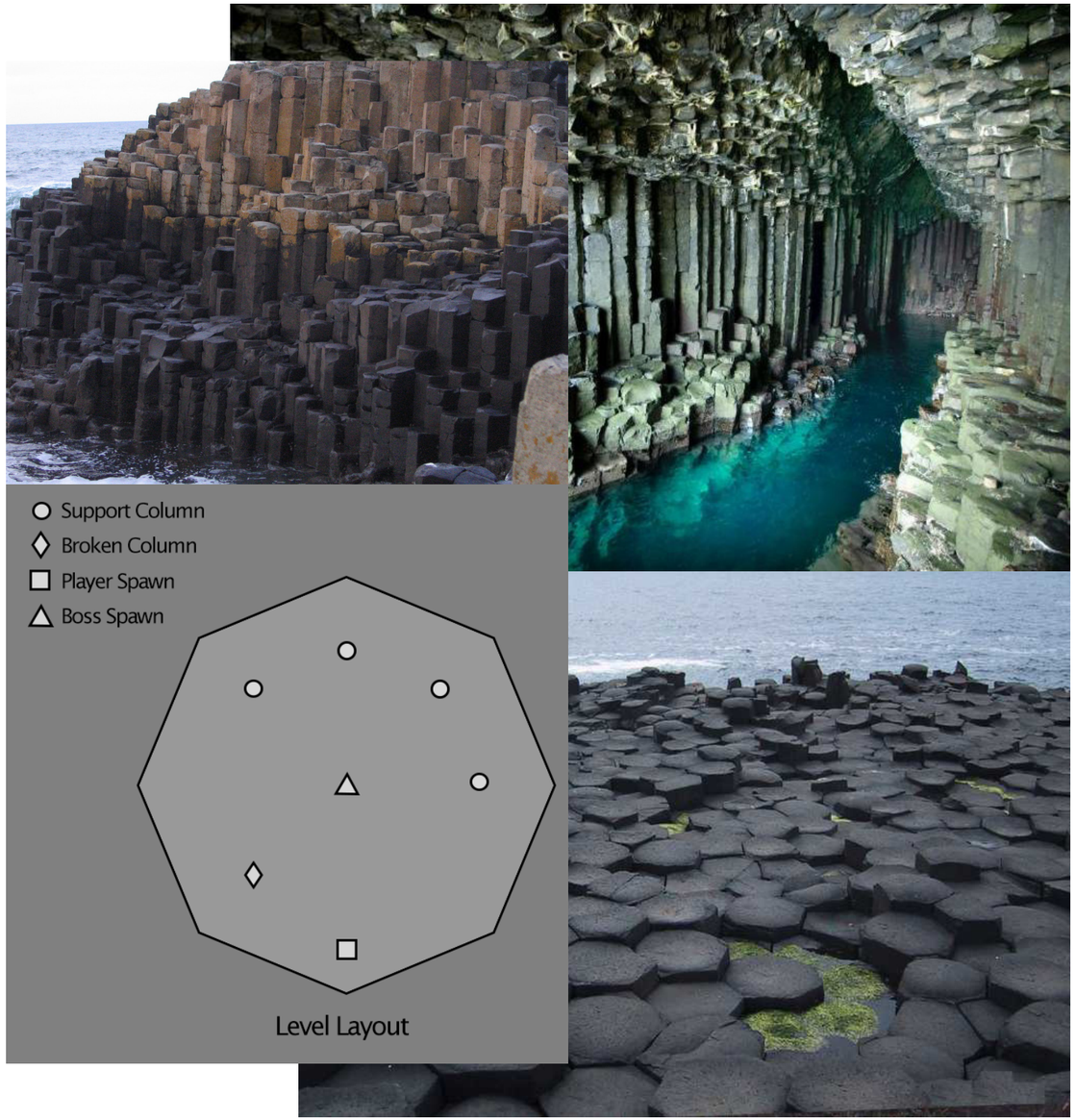
The Weapon

I had the idea early on to make the character lightweight and the weapon the heavy. I like the idea of big, exaggerated weapons that can pummel your enemy into the ground or strike them down with huge swings. I got inspiration from the greatsword and ultra greatsword designs in the Dark Souls Wiki.
I sketched the three before deciding on the left one. Most of the influence came from the simplest greatsword, but I changed the design to fit my character. I gave it a flat edge instead of a pointed tip, adjusted the length so it wouldn't be to unbelievable or collide with the floor as often, and fitted the rectangle hilt to the blade instead of shaping the end of the blade to the hilt like the concept art does.

The Environment

While looking for interesting environment reference that fit the modular criteria I set, I found out about basalt formations. Basalt formations are the result of a lava flow slowly cooling and forming polygonal columns ranging from four to eight sides. The coloration generally is grey to black, but can become red or brown over time from oxidation. The average width can vary greatly and is dependent of the rate of cooling.

Initially I created the environment out of the width of the columns found in the reference, but after it was finished it looked too busy and was felt washed out by the texture I made. The issue was resolved by massively increasing their size and reducing the number of columns used. Had I made the texture less modular, created an additional texture, or modeled wear into the columns. I concepted a simple level layout to get a sense of scale for the environment and how much space the characters would have. The area was scaled down a bit after I used the larger columns and the asset placement was adjusted slightly.



DEVELOPMENT

The Pillar

For the pillar, I wanted its design to fit a purpose within the theme if I had the time to expand upon the game mechanics. The reference I was using had the pillars sculpted out of the rock. Some had the lower portion of the pillar left alone, which help me come up with the idea of having the pillar affect the environment. The idea behind the pillar was that it was unnaturally pulled out of the ground, bringing up some of the rock around it as it rose. Since the pillar is not made of the surrounding basalt, the rock covering the pillar would be made of a slate-like material to give it a flat and square form. I decided to make the pillar into the shape of ancient obelisks, which gave it more purpose and fit better with the theme I roughed out. During its modeling, I made sure to vary each of its sides enough so that it could be rotated and used multiple times.

The Programming

I followed Brackeys' *Create a Survival Game* tutorials when creating my demo's game mechanics. It's a great series and covers several subjects that I didn't have time to add to the demo. His tutorials helped me create the melee system, enemy health, and the basic AI. I was able to get through several of the tutorials with no issues, until I tried getting the enemy to attack correctly. For some reason the enemy would not switch to the attack state and stayed in the move state. After spending too much time trying to find a solution, I moved on to finish the melee and enemy health scripts.

The first script is the enemy's health. It creates a health variable to be removed by the player's sword by calling the function 'TheDamage' from the melee script. It also destroys the enemy if health reaches zero.

The next script is the melee system. It creates the 'TheDamage' function referenced in the enemy health script. It also checks if the sword has touched the enemy's collider and defines it as a hit using Physics.Raycast and RaycastHit.

```
#pragma strict

var Health = 100;


function Update()
{
    if(Health <= 0)
    {
        Dead();
    }
}

function ApplyDamage (TheDamage : int)
{
    Health -= TheDamage;
}

function Dead()
{
    Destroy (gameObject);
}
```

```
#pragma strict

var TheDamage : int = 50;
var Distance : float;
var MaxDistance : float = 1.5;
var TheSword : Transform;

function Update ()
{
    //Attack function
    var hit : RaycastHit;
    if (Physics.Raycast (TheSword.transform.position, TheSword.transform.
    TransformDirection(Vector3.forward), hit))
    {
        Distance = hit.distance;
        if (Distance < MaxDistance)
        {
            hit.transform.SendMessage("ApplyDamage", TheDamage,
            SendMessageOptions.DontRequireReceiver);
        }
    }
}
```

The Programming PT2

The following script is the player's logic. It calls the animator and plays the animations during player input. The animator dictates when an animation can be played by referencing variables like 'speed', stick input, or button presses. The character movement is handled by the arrow keys and the attack animation is assigned to the space bar. The attack animation is very simple: the function being called with a button press; the animation is initially turned off and then played one time when the button is pressed. No scripting is needed to apply damage in the player logic since the melee system applied to the sword handles it.

Known issues with the player logic include: an animation not being referenced while the player is turning, and an issue where the attack animation plays at half the speed is should. The turn animation issue is because there is no state within the animator to reference nor an 'anim.SetBool' to set the walking animation to true while rotating.

```
using UnityEngine;
using System.Collections;

public class PlayerLogic : MonoBehaviour
{
    private CharacterController controller;
    public float speed = 6.0f;
    public float turnSpeed = 60.0f;
    private Vector3 moveDirection = Vector3.zero;

    private Animator anim;

    void Start ()
    {
        controller = GetComponent <CharacterController>();
        anim = gameObject.GetComponentInChildren<Animator>();
    }

    void Update ()
    {
        float turn = Input.GetAxis("Horizontal");
        transform.Rotate(0, turn * turnSpeed * Time.deltaTime, 0);
        moveDirection = transform.forward * Input.GetAxis("Vertical") * speed;
        anim.SetFloat ("speed", controller.velocity.magnitude);
        controller.Move(moveDirection * Time.deltaTime);

        if (Input.GetButton ("Attack"))
        {
            Attack();
        }
    }

    void Attack()
    {
        anim.SetBool("playerAttack", false);

        if(Input.GetButtonDown ("Attack"))
        {
            anim.SetBool("playerAttack", true);
        }
    }
}

```

The Programming PT3

The final script is the enemy AI. It basically references functions based on the variables set above. The ‘Distance’ variable is the major component of the script. It is the position of the player compared to the enemy. If the player comes within a certain distance, the enemy will either rotate to follow the player’s position, walk towards the player, or revert to the idle position and wait for the player to come closer. Each time one of these requirements is met, an animation is played.

I found that creating a follow camera required no coding, so I tried creating an orbit ability for it using the John Mac Youtube channel’s tutorials. While his tutorials a well made, I quickly found the topics involved a little advanced for my skill level and eventually had to stop progress. I kept the scripts in the project files so I could continue them when I expanded on the demo at a later date.

```
var Distance;
var Target : Transform;
var lookAtDistance = 25.0;
var attackDistance = 15.0;
var moveSpeed = 5.0;
var Damping = 6.0;
var anim : Animator;

function Start(){
    anim = GetComponent ("Animator");
}
function Update (){
    Distance = Vector3.Distance(Target.position, transform.position);

    if (Distance < lookAtDistance){
        //renderer.material.color = Color.yellow;
        lookAt();
    }
    if (Distance > lookAtDistance){
        //renderer.material.color = Color.green;
        idle();
    }
    if (Distance < attackDistance){
        //renderer.material.color = Color.red;
        move ();
    }
}
function idle(){
    anim.SetBool("enemyIdle", true);
    anim.SetBool("enemyWalk", false);
}
function lookAt (){
    var rotation = Quaternion.LookRotation(Target.position - transform.position);
    transform.rotation = Quaternion.Slerp(transform.rotation, rotation,
    Time.deltaTime * Damping);
    anim.SetBool("enemyIdle", true);
    anim.SetBool("enemyWalk", false);
}
function move (){
    transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
    anim.SetBool("enemyWalk", true);
}

```

Evaluation

While I was able to complete the major project objective and implement most of the assets listed, the demo feels unfinished. The major issue that occurred was time management. While I able to adhere to the schedule at the beginning, I lost about three weeks of time during the midterm due to various reasons. I made a list of issues and incomplete assets:

-There is no menu system.

-There are no sound fx.

-The combat is slow, one dimensional and unsatisfying.

-The texture maps used feel basic and look unfinished.

-The enemy AI is unable to attack the player.

-The enemy and player clip through each other.

-There is no turning animation.

-The game must be manually restarted to play again.