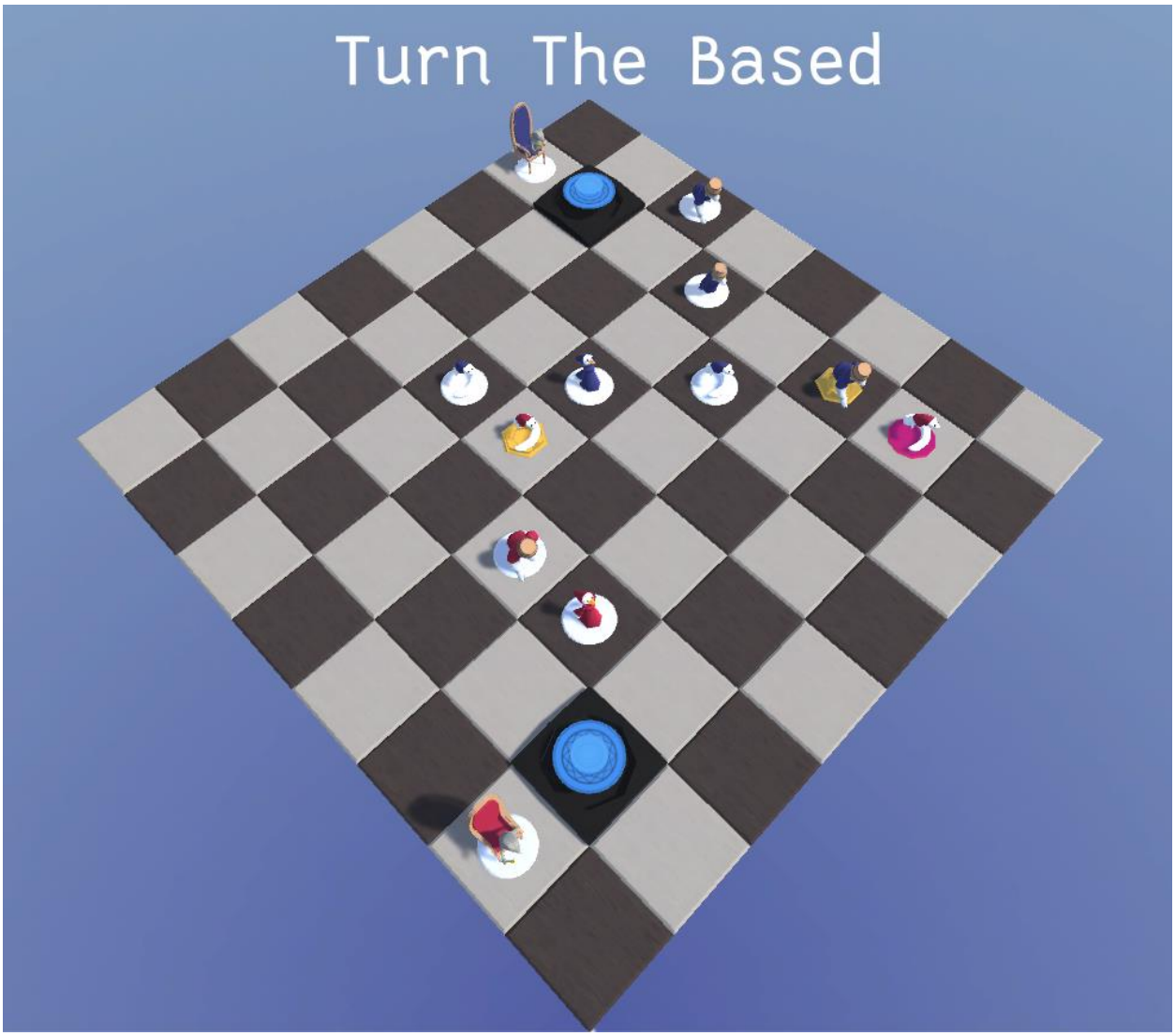
Turn The Based

Student: Ruud Schouten
Profile: Game Developer
Course: GTO4



Introduction

This document is meant to help me reflect on my research methodology and to document changes in my project before the assessment.

Contents

Deliverables

For my deliverables I will be making a working prototype which has the following features;

- A generated map.
- Two teams which are distinguishable from each other with colors.
- Units which can move and attack others.
- Units which have rarity and traits.

I did deviate a little bit on my deliverables, I was planning on generating a map which would also generate hills. This turned out harder than I thought it would be, especially with the movement.

Initially I was also planning to add pathfinding in my game, but I ditched this because of a lack of time.

Research phase

Library

For my game I needed units, I already knew what kind of game I wanted to make, and I had a general idea about what my units should be able to do. But I wasn't quite sure what stats my units should have.

To get a better idea, I performed the "Best, good & bad practices" from the Library research, and analyzed Disgaea 5. Units in this game have quite a few stats;

- HP: Health Points. If this hits 0 your unit dies.
- SP: Special Points. Used in Skills.
- ATK: Physical Damage.
- INT: Magical Damage.
- HIT: Determines chance to hit and used as Damage modifier for Bows and Guns.
- DEF: Physical Defense.
- RES: Magical Defense and used as multiplier for healing skills.
- SPD: Determines chance to evade and a damage modifier for Fists.
- : How many tiles the unit can move.
- : How high it can jump.
- : The range of its' attacks.
- : How far it can throw stuff.
- : How many times it can counter.
- : Critical hit chance.
- : Fire resistance.
- : Wind resistance.
- : Ice resistance.

Units also have Evilities. These are modifiers that can change anything about a unit. These can range from 10% increased ATK to Normal attacks will occur twice if attacking without moving.

Unit also have equipment. This consists of a weapon and three armor pieces. Items also have stats and can have Specialists. These are mostly simple stat boosts.

I really liked the Evilities system and the stats which are used by Disgaea fit well in my game. This is mostly because there aren't too many stats, but there are enough of them to keep it interesting enough, especially with the Evilities influencing stats.

After looking at the units of Disgaea 5 I decided on giving my units about the same stats as are found in Disgaea. I did leave out levels, throws, counters and critical hit chance.

In Disgaea units are gained by spending HL. You first select the class you'd like, and then choose how much HL you'd like to spend on this character. The higher the cost, the more points you can allocate to this characters' stats.

This system works well when there isn't another playing waiting for you, so Instead of recreating this system I chose to randomly generate units.

Areas in Disgaea are made up from tiles. Tiles can be next to one another, or on top of each other.

If a character doesn't have enough jump power to go up to a tile, it has to find a way around it.

Workshop

Now that I had a good idea what my units needed and how players can purchase them, I went ahead and performed the “Prototyping” from the Workshop research.

I made a few generators for the creation of units;

- `CharacterGenerator`
- `NameGenerator`
- `StatsGenerator`
- `TraitGenerator`

Generating a unit is done by calling the `Generate` method in `CharacterGenerator`.

```
public GameObject Generate(Rarity rarity, Transform parent) {
    GameObject go = new GameObject();
    go.transform.SetParent(parent, false);
    go.AddComponent<Character>();
    character = go.GetComponent<Character>();
    character.Name = nameGen.GetName();
    character.Rarity = rarity;
    character.MoveType = MovementType.Straight;
    character.Ownable = go.AddComponent<Ownable>();
    character.Ownable.Initialize(TurnManager.CurrentPlayer);

    character.UnitUI = UnitUiManager;
    character.TurnManager = TurnManager;
    Instantiate(BasePrefabs[(int) character.Rarity], go.transform);
    for (int i = 0; i < (int) character.Rarity; i++) {
        character.Traits.Add(traitGen.GetTrait(go.transform));
    }

    InstantiateModel(go, TurnManager.CurrentTeam);
    AddAttack(go);
    character.Stats = statsGen.AlterWithTraits(statsGen.GetStats(character.Type), character);
    go.name = string.Format("[{0}] {1} {2}", character.Rarity, character.Name, character.Type);
    return go;
}
```

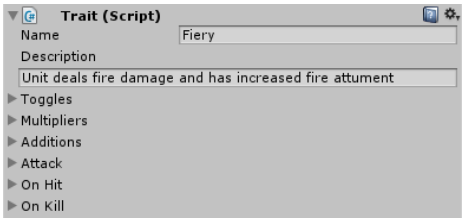
This method creates a new `GameObject`, adds a Character `MonoBehaviour` script to it and assigns values to it. It then checks the rarity of the unit, and if it’s Magic or Rare, it’ll add Traits using the `TraitGenerator`.

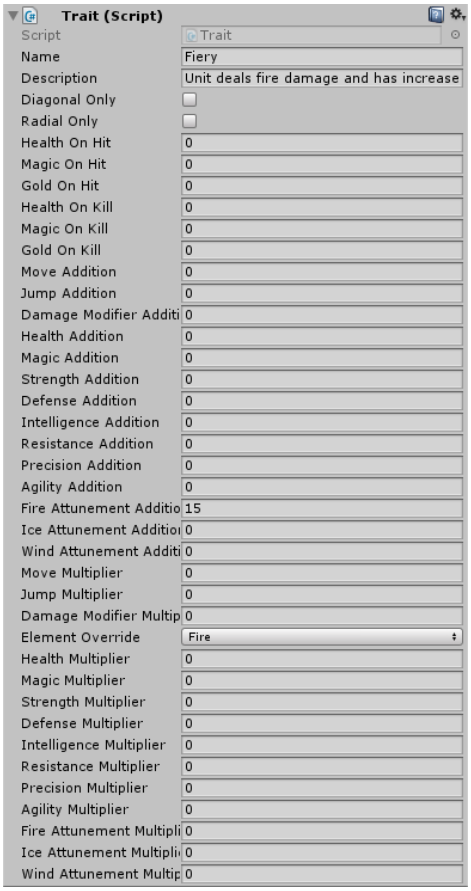
```
public Trait GetTrait(Transform parent) {
    GameObject traitGameObject = TraitPrefabs[GetRandom(TraitPrefabs.Length)];
    traitGameObject = Instantiate(traitGameObject, parent);
    return traitGameObject.GetComponent<Trait>();
}

int GetRandom(int length) {
    if (length <= 1) { return 0; }
    int randomIndex = lastIndex;
    while (randomIndex == lastIndex) { randomIndex = Random.Range(0, length); }
    lastIndex = randomIndex;
    return randomIndex;
}
```

This randomly selects a trait from the prefabs added to the `TraitGenerator`.

To make it easier to create Traits, I made a Custom Inspector for them, so the Unity Inspector looks clearer.
(Left is with inspector, right without)





After adding Traits, the model is instantiated.

```
private void InstantiateModel(GameObject go, Player.TeamColor color) {
    int charaRoll = Random.Range(0, RedCharaPrefabs.Length - 1);
    character.Type = (CharacterType) charaRoll;
    character.Skills = skillGen.GetSkills(character.Type);
    if (color == Player.TeamColor.Red) {
        Instantiate(RedCharaPrefabs[charaRoll], go.transform);
    }
    else if (color == Player.TeamColor.Blue) {
        Instantiate(BlueCharaPrefabs[charaRoll], go.transform);
    }
}
```

Then the Attacks are added.

```
private void AddAttack(GameObject go) {
    GameObject attack = null;
    switch (character.Type) {
        case CharacterType.Acolyte: attack = Instantiate(AttackPrefabs[1]); break;
        case CharacterType.Esquire: attack = Instantiate(AttackPrefabs[0]); break;
        case CharacterType.Brute: attack = Instantiate(AttackPrefabs[0]); break;
        case CharacterType.Rogue: attack = Instantiate(AttackPrefabs[2]); break;
        case CharacterType.Ruler: attack = Instantiate(AttackPrefabs[0]); break;
    }
    if (attack == null) throw new Exception("Failed to generate attack");
    attack.transform.SetParent(go.transform, false);
    character.Attack = attack.GetComponent<Attack>();
}
```

And finally, the stats are added, and the trait changes are applied to them.

```
public Stats GetStats(CharacterType type) {
    Stats stats = new Stats();
    //35 points in total.
    switch (type) {
        case CharacterType.Acolyte:
            stats.Health = 40;
            stats.Magic = 60;
            stats.Move = 2;
            stats.Jump = 1.5f;
            stats.Strength = 5;
            stats.Defense = 8;
            stats.Intelligence = 14;
            stats.Resistance = 14;
            stats.Precision = 10;
            stats.Agility = 8;
            break;
```

```
public Stats AlterWithTraits(Stats stats, Character character) {
    foreach (var trait in character.Traits) {
        Movement

        Additions

        Multipliers

        #region  Attack

        if (trait.ElementOverride != Element.None) {
            character.Attack.Element = trait.ElementOverride;
        }

        if (trait.DamageModifierAddition != 0) {
            character.Attack.DamageModifier += trait.DamageModifierAddition;
        }

        if (Math.Abs(trait.DamageModifierMultiplier) > 0) {
            character.Attack.DamageModifier *= trait.DamageModifierMultiplier;
        }

        #endregion
    }

    stats.MaxHealth = stats.Health;
    stats.MaxMagic = stats.Magic;
    return stats;
}
```

The stats I used are heavily inspired by Disgaea. I have 3 damage stats, and 3 defense stats.
I also used the Move and Jump stats but ended up not using the Jump because I don’t have hills.

Now that I had my units ready I needed an area for my units.

I made `AreaGenerator` for this. I originally planned on adding hills in my maps as well, but this proved to be harder to do than I thought. I did get it working randomly, but that didn't give me the results I wanted.

I start with generating an 8 by 8 grid.

```
private void SpawnGrid() {
    float xoffset = 0;
    int index = 0;
    for (int x = 0; x < GridSize; x++) {
        float yoffset = 0;
        for (int y = 0; y < GridSize; y++) {
            int modulo = (x + y) % 2;

            GameObject newTile = Instantiate(TilePrefabs[modulo]);
            newTile.transform.SetParent(transform);
            newTile.transform.localPosition = new Vector3(xoffset, 0, yoffset);
            newTile.name = string.Format("Tile {0}x{1} [{2}]", x, y, index++);

            newTile.GetComponent<Tile>().Position = newTile.transform.localPosition;
            newTile.GetComponent<Tile>().X = x;
            newTile.GetComponent<Tile>().Y = y;
            _tiles.Add(newTile);
            yoffset += WidthBetween;
        }

        xoffset += WidthBetween;
    }
}
```

After that, I add the Base tiles from which players can purchase Units.

```
private void SetBase() {
    var tile = GetTile(1, 1);
    int height = 0;

    _redBase = Instantiate(BaseTilePrefab, new Vector3(0, height * HeightBetween, 0), new Quaternion());
    _redBase.transform.SetParent(tile.transform, false);
    BasePanel redPanel = _redBase.GetComponent<BasePanel>();
    redPanel.TurnManager = TurnManager;
    redPanel.UiManager = _uiManager;
    redPanel.Ownable = _redBase.AddComponent<Ownable>();
    redPanel.Ownable.Initialize(TurnManager.Players[0]);

    tile = GetTile(GridSize - 2, GridSize - 2);
    height = 0;

    _blueBase = Instantiate(BaseTilePrefab, new Vector3(0, height * HeightBetween, 0), new Quaternion());
    _blueBase.transform.SetParent(tile.transform, false);
    BasePanel bluePanel = _blueBase.GetComponent<BasePanel>();
    bluePanel.TurnManager = TurnManager;
    bluePanel.UiManager = _uiManager;
    bluePanel.Ownable = _blueBase.AddComponent<Ownable>();
    bluePanel.Ownable.Initialize(TurnManager.Players[1]);
}
```

Now that I have my map, I have to add movement to my map so that the game is actually playable.

I had some trouble with getting movement working on my grid, Nick Hammes helped me with this by explaining how he programmed his movement and by showing me his code. This is what I ended up with;

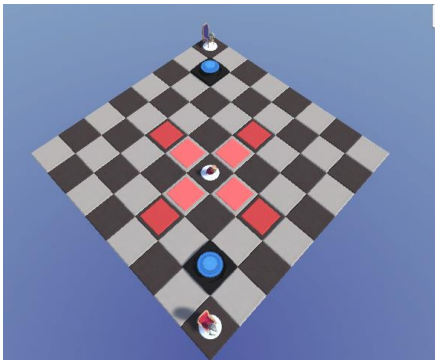
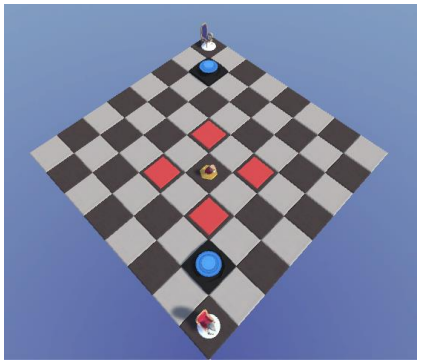
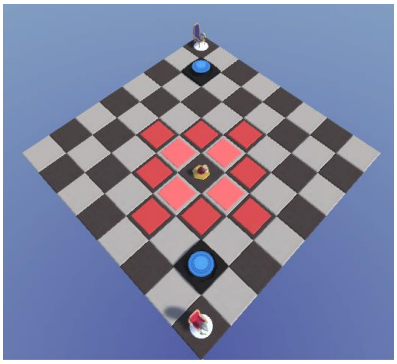
```
public bool InRange(Tile origin, Tile target, float range, MovementType movementType) {
    switch (movementType) {
        case MovementType.Straight:
            return ((target.X == origin.X && Mathf.Abs(target.Y - origin.Y) <= range) ||
                    (target.Y == origin.Y && Mathf.Abs(target.X - origin.X) <= range));
        case MovementType.Radial:
            return ((Mathf.Sqrt(Mathf.Pow((target.X - origin.X), 2) + Mathf.Pow((target.Y - origin.Y), 2)))
                    <= range);
        case MovementType.Diagonal:
            return (((Mathf.Abs(target.X - origin.X)) == (Mathf.Abs(target.Y - origin.Y))) &&
                    ((Mathf.Sqrt(Mathf.Pow((target.X - origin.X), 2) + Mathf.Pow((target.Y - origin.Y), 2)))
                     <= range));
        default:
            return false;
    }
}

public List<GameObject> GetTilesInRange(Tile origin, float range, MovementType moveType, bool addUnitTiles = false) {
    var tiles = new List<GameObject>();

    foreach (var tile in _tiles) {
        var t = tile.GetComponent<Tile>();
        if (InRange(origin, t, range, moveType)) {
            if (addUnitTiles) {
                tiles.Add(t.gameObject);
            }
            else {
                if (t.GetUnit() == null) {
                    tiles.Add(t.gameObject);
                }
            }
        }
    }

    return tiles;
}
```

This gives me three different movement types;

Attacking uses the same method. Now that I had all of this together, I had a working prototype for my deliverable.

Showroom

With my prototype ready to be tested, I asked a few classmates for feedback about my game.

Gideon van Doorn & Rik van Maaren

I had them play against each other. While playing, Rik found a bug in my code which occurred when you clicked on the unit instead of the tile it's on. Thanks to this bug the test had to be postponed for a bit.

After I fixed the bug they played it once more.

They didn't ask any questions and just started playing. They both bought the most expansive unit, which was at the time 150 gold. These gave them both very powerful units. They had their units fight against each other for a few turns when they asked me how they could win, which was by killing the Ruler. Rik's unit had more movement than Gideon's, so he charged ahead and was able to surround the Ruler and quickly finish him before Gideon could do anything.

Gideon thought it was a bit unfair because his units couldn't catch up to Rik's units. Which was just bad luck because of the randomness I added to the game.

Rik said the units were a bit too cheap, why should a player go for regular of magic units instead of going for rare ones? This piece of feedback made me up the price of Rare units to 250 which meant players could now get 2 rare units at most.

Tom Valkenburg

I had him play against me.

He didn't like the movement I implemented in the game, which is defaulted to Straight movement. When he got a unit with the Radial Trait (which grants Radial movement) he liked it more.

He was a bit confused about how much damage he was doing since there was no indicator of how much damage he was doing beside in the console. I was going to add an indication of how much damage was done by adding some text which floats up after dealing damage, but I had a shortage of time.

Sources

http://cmdmethods.nl/

http://disgaea.wikia.com/wiki/Disgaea_Wiki

https://docs.unity3d.com/ScriptReference/EditorGUILayout.Foldout.html