Piszemy RPGo-Platformówkę (4) – Przygotowanie do walki (sheadovas/poradniki/proj_platf_rpg/4-przygotowanie-do-walki/)

Kwi 01, 2017 / proj_platf_rpg (sheadovas/category/poradniki/proj_platf_rpg/)

Przygotowujemy podwaliny pod wprowadzenie walki do gry

Hej, dzisiaj zrobimy sobie wstępne wprowadzenie do systemu walki. Z racji, że sam system jest dość spory to zdecydowałem się go wprowadzić w dwóch częściach (kolejna część za tydzień). Omawiam idee wprowadzone w commicie [132c3ed (https://github.com/sheadovas/proj_platf_rpg/commit/132c3edb41829d47b3ab5d7965452fa3fecd0347)] (zachęcam do przynajmniej pobieżnego przejrzenia diffa).

Odrobina teorii

Tradycyjnie chciałbym poddać się paru rozważaniom przed przejściem do właściwego kodu.

Wiele z gier ma system walki, nie da się ukryć że może ona przejawiać się na wiele sposobów: od najprostszego skakania po głowach przeciwnikach i unikania ich (*Mario*), po tworzenie łączonych kombinacji ataków dających lepszy efekt (*Shadow of Mordor*). Można ją zrealizować na wiele sposobów, ale jedno jest pewne: powinna być w miarę intuicyjna i sprawiać przyjemność grającemu.

Sama walka może odbywać się na wiele sposobów, np na specjalnym ekranie oddzielonym od reszty gry (*SouthPark).*

(https://i2.wp.com/core0.staticworld.net/images/article/2014/03/2014-03-03_00047-100248765-orig.jpg)

Ekean walki (South Park)

Jednakże w grach platformowych przyjęło się, że walka jest bezpośrednia, jest integralną częścią gry gdzie bez żadnych zmian ekranów (scen) gracz ma okazję zmierzyć się oko w oko z przeciwnikiem.

(https://i0.wp.com/i1-news.softpedia-static.com/images/news2/salt-and-sanctuary-review-playstation-4-501908-9.jpg)

Walka w Salt and Sanctuary

Jak się okazuje przeciwnikiem wcale nie musi być „żywa” postać, równie dobrze może to być uzbrojona pułapka / przeszkadzajka, po której dotknięciu gracz traci przynajmniej część zdrowia.

Teraz zejdźmy błyskawicznie nieco niżej do gamedevu – czym charakteryzuje się (i skutkuje) walka? Na tym, że jeden z obiektów traci swoje punkty życia. A czym jest to powodowane? Zajściem kolizji pomiędzy dwoma obiektami uczestniczącymi w walce.

Powyższy fakt daje nam podwaliny pod napisanie systemu walki i właśnie tym zajmiemy się w tym wpisie.

Walka (część 1)

Jak już zdążyłem wspomnieć w walce bezpośrednio uczestniczą 2 „osoby”: atakujący i obrywający. Obaj muszą posiadać podstawowe statystyki potrzebne do walki, a więc punkty życia (hp) oraz punkty obrażeń (dmg).

CharacterStats.cs C#

```cs
using UnityEngine;
using System.Collections;

public class CharacterStats : MonoBehaviour
{
  public PlayableCharacter owner;

  public float hp
  {
    get
    {
      return m_hp;
    }
  }

  public float dmg
  {
    get
    {
      return m_dmg;
    }
  }
```

Oprócz tego jedna ze stron może być nieśmiertelna, tj. niewrażliwa na zadawane obrażenia (np. pułapko-kolce).

CharacterStats.cs C#

```cs
  public bool hitable = true;
```

W powyższej klasie pojawiło się pole „owner”, które jest stosowane dla zwykłych przeciwników i gracza (nie-pułapek), jak już zdążyłem wspomnieć nie zawsze zadawać / otrzymywać obrażenia musi gracz, a więc warto wyłączyć powiadamianie drugiego obiektu o otrzymywanych obrażeniach.

CharacterStats.cs C#

```cs
  public bool ownerNotificationsEnabled = true;
  public float timeInvulnOnHit = 2.0f;

  [SerializeField]
  private float m_hp = 100.0f;

  [SerializeField]
  private float m_dmg = 1.0f;

  public bool m_invulnerable = false;
```

Powyższy listing pozwoli nam na rozważenie sytuacji w której unikniemy otrzymania tych samych obrażeń wielokrotnie, w tym celu tymczasowo wyłączamy ich otrzymywanie.

CharacterStats.cs C#

```cs
  private IEnumerator StayInvulnOnHit()
  {
    m_invulnerable = true;
    yield return new WaitForSeconds(timeInvulnOnHit);
    m_invulnerable = false;
  }
```

Samo otrzymywanie obrażeń skupia się na sprawdzeniu odpowiednich flag, zmniejszeniu punktów życia i poinformowaniu o zmianie „właściciela” postaci do której należą te cechy.

CharacterStats.cs C#

```
  protected void GetHit(CharacterStats attackerStats)
  {
    if (!hitable) // if cannot hurt object...
      return;     // skip hitting it!

    if (m_invulnerable) // if got hit...
      return;           // stay invulnerable on hit for seconds!

    m_hp = Mathf.Max(m_hp - attackerStats.dmg, 0.0f);

    StartCoroutine(StayInvulnOnHit()); // temporary disable hitting player

    if (ownerNotificationsEnabled)
      owner.NotifyDamageTaken();
  }

  private void OnCollisionEnter2D(Collision2D collision)
  {
    CharacterStats enemy = collision.gameObject.GetComponent<CharacterStats>();
    if(enemy != null)
    {
      GetHit(enemy);
    }
  }

  private void OnTriggerEnter2D(Collider2D other)
  {
    CharacterStats enemy = other.gameObject.GetComponent<CharacterStats>();
    if (enemy != null)
    {
      GetHit(enemy);
    }
  }
```

Powyższy kod zakłada też, że jeżeli obiekt nie posiada klasy *CharacterStats* to nie może uczestniczyć w walce. Warto też zauważyć, że nadaje się on zarówno na postacie, jak i ich uzbrojenie, np. nałożone na miecz mogą zadawać obrażenia tak aby postać ich nie otrzymywała. Warto też zauważyć, że świetnie nadają się na stworzenie zużywalności broni w sytuacji gdy ta jest używana (to tak wybiegając do gry RPG).

Następnie warto stworzyć przynajmniej prototyp metody, którą zmienimy sobie w kolejnej części:

PlayableCharacter.cs C#

```
  public virtual void NotifyDamageTaken()
  {
    float hp = stats.hp;
    Debug.Log("Got hit, current hp: " + hp.ToString(), this);
    GameMaster.gm.specialEffects.ChangeTempColor(m_renderer, Color.red, stats.timeInvulnOnHit);

    if (hp == 0)
    {
      Debug.Log("Play death anim!");
    }
  }
```

Tutaj widzimy debugowe logi z informacją o otrzymaniu obrażeń lub śmierci. Z racji, że tylko śmierć gracza może spowodować koniec gry, to nie możemy stworzyć przejścia do końca gry w tym miejscu. Musimy to zrealizować bezpośrednio w klasie *Player*.

Player.cs C#

```
  [SerializeField]
  protected GameOverCondition m_conditionDeath;


  public override void NotifyDamageTaken()
  {
    base.NotifyDamageTaken();
    m_conditionDeath.CheckConditions();
  }
```

ConditionPlayerDead.cs C#

```
public class ConditionPlayerDead : GameOverCondition
{
  public CharacterStats targetStats;
  string message = "Player is alive";

  public override string GetProgressInfo()
  {
    return message;
  }

  protected override bool isFailure()
  {
    return false;
  }

  protected override bool isSuccess()
  {
    if (targetStats.hp == 0.0f)
    {
      message = "Player is dead :(";
    }
    return false;
  }

  private void Start()
  {
    AddActionOnSuccess(() => { GameMaster.gm.NotifyFailure(this); });
  }
}
```

Powyższe kody to prototypy, które ulepszymy w następnej części, bo jak widać kodu będzie całkiem sporo (pojawi się także uzbrojenie).

Jako ciekawostkę na koniec dodam jeszcze klasę, którą w trakcie rozwoju gry będziemy ulepszać o nowe elementy. Na razie mamy specjalny efekt, który zmienia kolor gracza po trafieniu na z góry określony czas:

SpecialEffects.cs C#

```
using UnityEngine;
using System.Collections;

public class SpecialEffects : MonoBehaviour
{
  public void ChangeTempColor(SpriteRenderer renderer, Color color, float time)
  {
    Color oldColor = renderer.color;

    StartCoroutine(changeColor(renderer, color));
    StartCoroutine(changeColor(renderer, oldColor, time));
  }

  IEnumerator changeColor(SpriteRenderer renderer, Color color, float wait=0.0f)
  {
    yield return new WaitForSeconds(wait);

    renderer.color = color;
  }
}
```

Podsumowanie

Dzisiaj zrealizowaliśmy wstęp do systemu walki, który jak widać będzie całkiem spory i (mam nadzieję) uniwersalny dla wielu obiektów.

To tyle na dzisiaj, liczę że wpis Wam się podobał. Tradycyjnie zapraszam do pobrania [dema (https://github.com/sheadovas/proj_platf_rpg/releases/tag/1.4)] (wciąż bardzo ubogiego w funkcjonalność), a także do systemu komentarzy, dzielenia się wpisem i śledzeniu przez social-media (wszystkie linki w bocznym menu).

W kolejnej części dokończymy system walki, a więc dodamy efekty na utracenie wszystkich punktów życia, animacje walki i dodamy broń.

Code ON!