Dragon Punch

A game by Kevin Slackie

Table of Contents

* Design Fundamentals
  + Narrative
  + Core Mechanic
  + Game Structure
  + Other Mechanics
  + Player Motivation and Experience
* Game Systems Overview
  + Written Systems Description
  + System Description
* Addressing Project Requirements
  + Procedural Generation
  + C# and Unity API
* Term Glossary

Design Fundamentals

The fundamental question this game answers is “how would you combine a JRPG with a fighter game?” By bringing together the different aspects of both genres the hope is to make a truly unique game.

Narrative

Players take the role of Sir Cessibit Forore, a knight just about to clock out and go home. Suddenly the Verbose Dragon decides to kidnap the princess (to find someone to listen to his endless stories). Of course the king mobilizes all his troops, forcing Forore to work overtime. When the first battle starts off as turn based, Forore loses his patience, throwing away his weapons and fighting the monsters more directly.

Core Mechanic

The core mechanic of any fighter game is to use a combination of buttons and player learned skills to defeat your opponent before they beat you. How the player does that is further explained in the other mechanics section. Battles will take place on a 2D plane, like Street Fighter or early Mortal Kombat. Defensive moves the play can take include blocking and using movement to not be where the hitbox of the enemy is.

Other Mechanics

World Traversal – Navigating through the world will be done in a traditional 2D orthographic top-down view. Reminiscent of older JRPGs, this navigation allows people to get into the mindset needed for the systems outside of combat such as equipment and item management, stat tracking, and the job system.

Statistic System – A classic system in RPGs, statistic or “stats” are numbers that define an aspect of a game entity. Hit Points are an excellent example, a number for the amount of damage a player can receive before losing a life or receiving a game over. In this game there will be stats for

Hit Points – the vitality the player has

Attack – affects the damage a player can do

Defense – affects the amount of damage a player takes

Agility – affects the players movement speed in battle

Job System (Not in Prototype) – JRPGs of old prided themselves on job systems, though the system will be re-defined like the other tropes in this game. Different job classes will control like different fighter character archetypes, so while the starter class might play similarly to Ryu with easy to learn controls and straightforward moves, another class can play like a charge-based character or grappler. Leveling up a job to certain milestones will give the player permanent bonuses like extra buffs to their stats or even new moves.

Equipment System (Not in Prototype) – The player will be able to equip different weapons and armor based on their current level and job. This will affect the various stats of the player character, though they will have to be thoughtful about what they equip. Different armor will be weak to certain elements, so wearing a set that’s weak to fire in a lava area will end badly.

Enemy Adjective System – Rather than have every encounter feel similar, Dragon Punch is going to give enemies a way to stand out with procedural generation. Enemies will have base stats that are altered based on the adjective randomly assigned. Some examples include an enemy with the “Brutish” adjective will have higher attack but less defense, or a “Meek” enemy that has lower attack and defense but higher speed.

Impatience System (Not in Prototype) – A silly take on the super meters found in modern fighter games, the Impatience System will let players deal out big, momentum changing skills depending on how long the fight has been, how much damage they’ve already dished out, and by how much damage has been taken. The meter has 3 different segments, but also has the second one cut in half, allowing players do to specific moves that take 1 ½ bars of the meter.

Game Systems Overview

A general overview of the systems the game will use to create a unique experience. This section will be especially important as this game combines two genres that are normally seen as contradictory.

Written System Descriptions

Here you’ll find the planned game objects, their attributes, and behaviors in the game world.

The Player, Sir Cessibit Forore

* Attributes
  + Health – The amount of damage the player can take
  + Attack – The multiplier used to calculate the damage the player deals to enemies
  + Defense – The multiplier used to calculate the damage the player takes from enemy attacks
  + Speed – The multiplier used to determine the movement speed of the player
  + Standard Moves – The moves the player gets from pushing buttons without a specific combination required
  + Special Moves – Any move the player needs to do a specific button combination to get
  + Super Moves – Special moves that use the super meter resource to do. Often stronger because of the cost
  + Sprite – The image that represents the player. Should change based on different states
* Behaviors
  + Movement: The player can move left and right, crouch, and jump. This movement is affected by the Speed stat.
  + Fighting: The player can use a combination of standard, special and super moves to damage the enemy. They can also block to lower the amount of incoming damage.

Enemy Slime

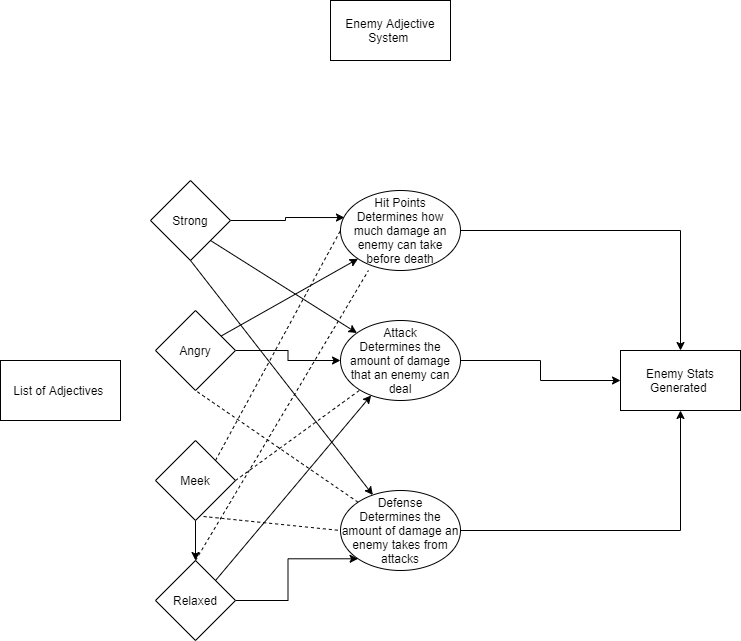
* + Health – The amount of damage the slime can take
  + Attack – The multiplier used to calculate the damage the slime deals to the player
  + Defense – The multiplier used to calculate the damage the slime resists from player attacks.
  + Standard Moves – The pool of moves the A.I. chooses from to use most often
  + Special Moves – The pool of moves the A.I. does with less regularity that deal more damage or are situational
  + Super Move – The strongest move the A.I. can perform, requires their own super meter to use
  + Sprite – The image that represents the slime. This changes based on the damage the slime currently has and the actions it performs.
  + Adjective – The part of the Enemy Adjective System the player sees. Changes the Slimes stats such as health, attack, defense. Stretch goals include having this affect the A.I. as well.

Stage

* Attributes
  + Flat – The stage is fully flat, with sprite animations in the background
  + Walls – The ends of the stage will have invisible walls. The player can run at this wall for an amount of time to run away from the fight, if the enemy doesn’t interrupt the process.
  + Ceiling – An invisible ceiling on the top keeps the player within the screen
* Behaviors
  + Camera Pan – Camera focuses on the player, with a priority of also focusing on the nearest enemy

System Diagrams

Here is a section of diagrams describing the systems above, the ways they interact, and which ones will be featured in the prototype.



Addressing Project Requirements

This section will show how this concept meets the requirements of the assignment.

Artificial Intelligence

The enemy A.I. will be more sophisticated than my current skillset allows. So this is guaranteed learning experience, even if I don’t get the exact result I want for this prototype. Enemies will have to respond to player’s inputs and resources. Conversely the A.I. can’t be too strong or it’ll keep the player from having fun. The A.I. will keep the adjectives in mind when picking behaviors. Certain behaviors are exclusive to specific adjectives, Strong has access to the defense boosting Bulk Up, Angry has the double-edged sword move Rage, Meek has the ability to Run Away from the player, and Relaxed may sometimes go to Sleep

Procedural Generation

Randomness here comes in two forms: random enemy battles and the Enemy Adjective System. As each encounter loads in, the game will randomly assign different adjectives to the enemy. You can see an example of the system above, but the general idea is that each enemy will be a little unique, having altered stats based on their adjective.

C# and Unity API

I have never made a sophisticated A.I. like this or a 2D combat system either. There will be a fair amount of if/else depending on the player input and resources. There will need to be a list of enums for the enemy adjective system, and most definitely a loop for the enemy behavior.

As for the Unity API features, On/Collision will be vital to the game working as intended. Update will be important for managing the player and enemy resources, along with the consequences for their use. Finally there will need to be various sound cues from the enemy to let players better notice the pattern for its attacks. I’m sure there will be more components used that I haven’t thought of yet.

Term Glossary

RPG – Role Playing Game. While this term can truly mean any game, here we use it to mean the JRPGs from the Super Nintendo and Sega Genesis console generation and the series they inspired. These games had a large focus on story, preparation, and planning.

Job System – a set of mechanics first introduced in Dungeon and Dragons that allowed the player to define their role in a fight. An classic example is one player choosing to “tank” as a knight so they can protect a healer that in turn keeps the knight sufficiently healed.

Super Meter – a resource in modern fighter games that charges as players take or receive damage. The resource can then be spent on certain, more powerful moves.

Charge-based Character – An archetype in fighter games where the character must hold an input down or “charge” the move, before continuing the sequence to perform a special move. These characters tend to reward patient, methodical gameplay. Guile from Street Fighter is the most iconic charge character.

Grappler – An archetype in fighter games where the character is better suited for close combat than long distance fighting. Characters of this type tend to have slow movement speed along with a ton of health and a massively damaging grab special move. This rewards players who charge in and keep the momentum of the fight in their favor. Zangief from Street Fighter is the original grapple fighter.