**Magi Coliseum**

by Timothy Miller

**Assessment**

If I were to create a proof of concept again, the one thing I would absolutely do differently is to build prototypes of the concept while I am refining the concept. If a concept is built in theory and not practice, then the larger it becomes, the more likely it is to stray away from what is possible to create. Building pieces of the concept along the way can help keep the ideas grounded. Also, it creates a new perspective to gauge the concept through. A concept that is solid in theory may prove to be convoluted in practice, which may lead to the concept being retooled or scrapped. Furthermore, it is a lot easier to sell the tangible products of a concept than the concept itself. The early prototypes can be used to help sell the concept.

**Game Community**

I did not reach out to any game community for help developing my concept because I did not feel it to be necessary. I was able to refine the concept and create a solid plan quickly on my own without issue. It’s not that I was opposed to reaching out to the game community to help refine my concept. It’s just that I came into this class with plenty of technical knowhow from previous classes. So, the question of how to build the concept never crossed my mind, I instinctively broke the concept down into problems I know how to solve.

Although I did not reach out to the game community for help in building this concept, I did learn plenty about how to best build my portfolio and sell myself as a gameplay programmer. I did not actually reach out for input myself, instead the relevant questions have been asked, and answered, many times over by other people. I figured why bother asking for information that is already freely available. The first resource I came across, and probably most significant one to inform my actions moving forward, is a blog post on codingame.com called *Working on Video Games as a Gameplay Programmer*. This blog focuses on the insight the author has gained along their 12 years of working as a gameplay programmer, as well as what people who are looking to break into the role can do to stand out from the crowd. I took three tips for getting into the field away from this blog. First, actively build my portfolio with my own projects instead of passively filling it with my schoolwork. By making my own projects, I can push the limits of my abilities and build a portfolio that shows off what I can do and not just what I was asked to do. Second, I should aim for the job that no one wants. There is a shortage of experienced game developers, but also an over abundance of people with zero experience trying to become game developers. Many inexperienced people make the mistake of being picky with the job they apply for and push try to land a job at a prestigious studio with no experience. However, if I shoot for a job on a low budget mobile game, or ill-conceived movie tie-in game, then I will have a significantly greater chance of getting the job and that experience will allow me to transition to a big-name studio. Lastly, while I am trying to get a job in the field of game development, I should get a job as a programmer. The last job on my resume is Laborer; the money was good, but it isn’t going to do me any favors when I apply to a game studio. A programming job will look much better on my resume and allow me to build experience (RockyMullet, 2018).

I also found other sources on gamedev.net which reinforced my takeaways from the blog and offered further insight into how to build my skills and my portfolio. I can practice languages I am unfamiliar with by recreating existing games in various languages (3Ddreamer, 2013). I can then take the copycat games and unique twists to them to show off my ability to create creative mechanics (Mosker, 2019). For example, I could make Tic-Tac-Toe and then make it three dimensional, or four dimensional.

**Overview**

The prototype I intend to develop is a combat system centered around customizable magic spells. The primary thrust of the concept is the system for customizing the spells, as well as the behavior of the spells. Spells are comprised of a finite set of properties which the player may exchange in the customization system. The properties the player has set for their spells determine what the spell will do when cast. In a completed game, the properties available to the player would be determined by the progression system, but in the proof of concept the progression system will be a simple level up system wherein players will be granted levels by just pressing a button. Also, players incur a Mana Point, or MP cost whenever they cast their spells. All spells have a base MP cost, which is increased by changing the emission style or impact effect. All elements have the same cost. MP regenerates over time. Health points, or HP does not regenerate over time. HP, MP and MP regeneration are also determined by the progression system.

Players fight using custom spells. Players may equip five spells, one active spell in each hand and three in reserve. Players may swap their active and reserve spells at any time. Only when outside of combat, players may access a menu wherein they may freely exchange the properties of their spells.

Each spell has three type of properties. The first property is the elemental alignment which modifies the damage the spell does to the target based on what the target has equipped, e.g. if the target has fire spells equipped then they will take extra damage from water spells. The second property is the emission style which determines what form the spell takes and how it travels from the caster to the target. The third property is the impact effect which is what happens when the spell hits something.

**Approach**

This prototype is related to my specialization of gameplay programmer because it combines many related gameplay behaviors and, if properly structured, can easily be increased in scope to include more behaviors. To that end, my approach to development is to create three objects to act as databases, holding all the properties the spells can have. Each entry in the element class includes a field for the element, the element weakened by the element, and the element that resists the element. Each entry in the emission style class includes the emission style name and description. Each entry in the impact effect class includes a description of the effect and the script to create the effect. These data structures will act as the backbone for the customization system and combat system. When the customization menu is opened, the menu will read the player’s stats and compare it to the data to determine what properties should be presented as unlocked. The customization menu will then write identifiers to the player’s class, so that then when the player casts their spell an appropriate spell class may be spawned and act according to the information imprinted on it by the caster.

Data structure examples

|  |  |  |  |
| --- | --- | --- | --- |
| Key Property | Element | Weak to | Strong against |
| Level 1 | Fire | Water | Ice |
| Level 2 | Water | Lightning | Fire |
| Level 3 | Lightning | Earth | Water |

|  |  |  |  |
| --- | --- | --- | --- |
| Key Property | Emission Style | Emission Style Description | Damage formula |
| Level 1 | Ballistic | Lob a magic sphere along a parabolic arc | If element.StrongAgainst = target.Mainhand.Element:  Target.HP -= caster.Level \* 1.5  Elseif element.WeakAgainst = target.Mainhand.Element: Target.HP -= caster.Level \* 0.5  Else  Target.HP -= caster.Level |
| Level 3 | Laser | Fire a straight beam which instantly hits the current target | Same as ballistic |
| Level 5 | Conical Spread | Spreads out like a shotgun | If element.StrongAgainst = target.Mainhand.Element:  Target.HP -= caster.Level \* 1.5 \* 1/(caster.WorldPos – target.WorldPos)  Elseif element.WeakAgainst = target.Mainhand.Element: Target.HP -= caster.Level \* 0.5 \* 1/(caster.WorldPos – target.WorldPos)  Else  Target.HP -= caster.Level \* 1/(caster.WorldPos – target.WorldPos) |

|  |  |  |
| --- | --- | --- |
| Key Property | Impact Effect | Impact Effect Script |
| Level 3 | Target Bleeds HP | Def bleedHP (target, caster):  Foreach i in caster.Level:  Target.HP -= i |
| Level 5 | Target Bleeds MP | Def bleedMP (target, caster):  Foreach i in caster.Level:  Target.MP -= i |
| Level 7 | Steal Target HP | Def stealHP (target, caster):  Foreach i in caster.Level:  Target.HP -= i  Caster.HP += i |

**Defense**

This approach is meant to create a system which can be scaled with relative ease. I cannot say for certain whether my choice of development approach is the best way to build the concept. However, I can say with confidence that this approach will produce a serviceable system. Braking the spells down into component properties and building systems that assemble those properties into functioning behaviors is the only methodology I can conceive of which will not result in a mess of spaghetti code. The alternative would be to create all possible spells individually and write a monstrous switch case statement to make sure the spell matching the player’s settings is created when they cast a spell. But in that sort of approach, the total number of spells would grow exponentially with each new property. In fact, just the eight elements, six emission styles, and twelve impact effects I have already come up with would result in 576 unique spells. It’s far more reasonable to build a system around 26 properties of spells than 576 spells.

**Genre**

The overall genre of the game is action rpg. Players fight in an arena; using magic spells to defeat groups of NPCs who wield similar magic.

**Platform**

The ideal targeted platform is pc because I want HD graphics which rules out mobile, and the only hardware I can test the game on, besides my phone which has already been ruled out, is my pc.

**Spell Properties**

* Elemental alignment
  + Fire - bonus damage to ice
  + Wind - bonus damage to earth
  + Water - bonus damage to fire
  + Lightning - bonus damage to water
  + Ice - bonus damage to wind
  + Earth - bonus damage to lightning
  + Light - bonus damage to dark
  + Dark - bonus damage to light
* Emission styles
  + Ballistic trajectory - a simple ball
  + Laser beam - instantly strikes where the caster is aiming
  + Conical spread - spreads like a shotgun
  + Landmine - self explanatory
  + Vortex - pulls target towards the spells
  + Explosion - damage falls of based on distance
* Impact effects
  + Target bleeds HP
  + Target bleeds MP
  + Steal target HP
  + Steal target MP
  + Target movement slowed
  + Target MP regen slowed
  + Explosion at the point of impact
  + Elementally aligned conflagration spawned at point of impact
  + Target knockback
  + Elemental effectiveness temporarily inverted
  + Target confused (movement inverted)
  + Swap position with target (caster remains oriented to face target)

**Target Rating and Target Market**

* The expected ESRB rating is E because, unless a story gets added, the basic concept of people competing with magic to defeat each other does not warrant a higher rating.
* The target demographic will be pc gamers

**Competitive Analysis**

* Final Fantasy XV: Players create single use spells by filling magic vessels with elemental energies and consumable items. The energies used determine base damage and elemental alignment, and the consumable item determines if the spell has a bonus effect like poison or damage up. No matter what spell is created they are all cast in the same way, by physically throwing them at the opponents like a grenade.
* Runescape: Players can equip items in a mainhand and offhand slot. Shields and similar items exclusively go in the offhand slot whereas most weapons have a mainhand and offhand equivilant, allowing for dualwielding.
* Borderlands: The weapons in Borderlands are generated in a similar manner to how spells will be generated in the POC. Every weapon is really just a collection of parts, with each part contributing to the collective properties of the weapon.

**Plan for Creation**

1. *Create three objects to hold all possible spell properties, including elements, emission styles, and impact effects*
2. *Create classes to serve as the emission styles and handle transmitting the information of the spell from the caster to the target*
3. *Add scripts to the emission style classes to apply some damage to the target based on the caster’s level, the elemental properties of the spell verses the element of the target’s spells, and the emission style of the spell if it should have damage falloff with distance like conical spread or explosion.*
4. *Create a menu system which the players will use to swap out the properties of their spell from a list of available properties*
5. *Create impact effect scripts*

**Concept Points**

The player will be able to have two spells active at once, one mainhand spell and one offhand spell. Attack spells are at full power when equipped in the player’s mainhand and half power when equipped in the player’s offhand slot. Defensive and restorative spells are at full power when equipped in the offhand slot and half power when equipped in the mainhand slot. The player can only cast spells in their mainhand and offhand slots. Spells in the inactive slots can be moved into mainhand or offhand slots on the fly, even during combat.

**Deliverable Format**

The most appropriate method of getting across the elements of the proof of concept is pseudocode. Well written pseudocode will fully encapsulate the desired behavior and communicate it in a way that can be discerned regardless of technical skill.

References

3Ddreamer. (2013). Game programmer. Retrieved 24 November 2019, from https://www.gamedev.net/forums/topic/638098-game-programmer/

Mosker. (2019). Developing Skills as a Gameplay Programmer. Retrieved 24 November 2019, from https://www.gamedev.net/forums/topic/703259-developing-skills-as-a- gameplay-programmer/?tab=comments#comment-5410332

RockyMullet. (2018). Working on Video Games as a Gameplay Programmer. Retrieved 21 December 2019, from https://www.codingame.com/blog/video-games-gameplay-programmer/