**Cheshire’s Game Design Document**

**ALLNIGHTER TODO LIST FUCK YEAH**

**1.Fix outputting issues with everything else (store, town, menu)**

**3.Expand the story a little bit**

**5. Fix stupid bug where the player has to press enter twice to go to town in game (meh)**

**6. Maybe test if multibattles between enemies and allies work(they should work)**

**WOOWHWIHREIHOHFDSREWRRR AHHHH LOL**

**TODO**

* **[suggestion]** when player buys stuff, show stats of the weapons, not just the cost
* **[incomplete]** cant talk to townspeople
* **[suggestion]** There is a “T” on the dungeon map so that when the player goes to it, they go back to town.
* **[note]** after i finish a dungeon, it takes me back to town. I have to input the number twice before it goes to that option
* **[bug]** Invalid inputs break game at almost every stage

**NOTES ABOUT DUNGEON/ENTITY IF YOU WANNA USE IN SCENES**

* Beginner boss = Enchanted Fox /// Easy boss = Elka the Spider Queen
* Medium boss = Rica the Infernal Tyrant /// Hard = Acquilo the Frost Lord

**Demo Features**

For the demo, we need a couple of components MOSTLY done:

-One town

-Shops

-One field

-Multi Monster Battles

-Items

-Loot

-Player creation/stats/levels

-Jobs (One or two will suffice)

-Skills

**What will the demo do?**

The demo will showcase what would be our typical rundown of a day in the main game. We will allow the player to create their own character and jump them straight into a town where they are equipped with items and can access certain places within the town. The player will then be able to explore an area, making choices as they move along the map. Finally, the player will be able to go through a dungeon and allow them to grind using our map based system. There will be no A.I., but the player will simply go through the game like a regular RPG.

|  |
| --- |
| *Changelog*  3/25/15 Added a changelog. Also organized the document into a table of contents.  3/25/15 Expanded on items and added descriptions for shop, stats, and basic menu. |

Table of Contents

[Feature List](#_7rp4f7pefbbz)

[Save System](#_wqalvif6stno)

[Nature/Types](#_ivn90dvx9eex)

[A.I.](#_lppnklvw4u95)

[Stats](#_an3jlqmq09p0)

[Item](#_ze6an2mwx9xk)

[Shop](#_jcmmpj4ypxzs)

[Contestants](#_hpzy59gnc7h5)

[Towns](#_mxag22ntldan)

[Entities](#_i2r0slanmge1)

[Time System](#_avquhi2hgavb)

[Dungeons](#_ue351p7sf353)

[Map System](#_uzrxwhg63cu)

[Menus](#_t8f1odlcjbmf)

[Basic Menu](#_zbvnjjcmmixm)

[Quests](#_r1di02qvt4by)

[Game (Or uh, Game Setup?)](#_artk3f9k2gla)

[Game Progression](#_l09aqzagnfaf)

[Outline of Game Day](#_lekxipcz7l7h)

[A Typical Day’s Rundown](#_sr3gr5cx9hba)

[Going Into a Dungeon](#_z00y1l9t1oof)

[Prototype Stuff](#_rlbdsgbiztu)

# Feature List

- Shops

- Items

- Combat

-Minigames

-Multiple choices/Consequences with each action

-Allies?

-Trust System?

**- Jobs**

**- Skills**

-Stats, health, etc. (Basic stuff; Expands upon player)

## Save System

Description: The save system does what it implies. When the player saves the game, all of its data needs to be collected. This includes the players health, level, money, current location, and day, etc. As more features are added to the game more stuff will be saved.

## Nature/Types

Description: Note: This may coincide with A.I.

Contestants are given personalities to spice up the competition and provide the player with a unique experience in each run. Of course, players can’t always get a profoundly new experience per run, but not knowing the pattern of each contestant will make each game interesting with each run. The best way to do this is to give each contestant a trait. There will be multiple types, and each will decide how the contestant will react to the player, progress through the game, and its nature in general. Here are some examples of natures:

Aggressive: Willing to attack more against enemies and the player; very reckless

Timid: Cautious about fighting in general; tends to run away often

Intelligent: Chooses who to attack carefully; treads through the game slowly, but avoids conflicts that cannot be won

Sly: Chances of running are high; Very elusive in general

Strong-willed: Extremely weak from the start, but becomes powerful as days past

Early-Gamer (Can’t find the word for this yet): Strong from the start, but does not get any better late in the game

Adaptive: Well-rounded; will fight when necessary

Early-Bird: Very exceptional and powerful during the day; vulnerable at night

Nocturnal: Very exceptional and powerful during the night; vulnerable at day

## A.I.

Description: This is the brain of the contestant. How each contestant will react to each day will be calculated in the A.I. What should the contestant do in the beginning of the day? What should it do later on? In which areas will the contestant want to train in? This will calculate the chances for the contestant to do certain things. This is based on the A.I.’s nature, so it want to do certain In a sense, we are simulating a player’s actions based on its own personality.

## Stats

-Players have “hidden stats” that are influenced by regular stats. These are basic stats that are essential to the player:

Minimum Attack: The least damage a player will do on an enemy.

Maximum Attack: The most damage a player will do on an enemy.

Defense: Reduces damage taken by the enemy.

-Each stat will affect the player in such ways:

Strength: Increases the minimum and maximum attack of the player.

Agility: If the player’s agility is higher than the enemy’s, then the player attacks first. A player’s agility also influences the chances of dodging an attack. The chances decrease if the player’s agility is lower than an enemy’s dexterity.

Vitality: For every 5 points of vitality, the player’s maximum health will increase by 3%. It’s points will also influence how much damage they will take while defending. (Note: Maybe add a hidden stat for defense?)

Dexterity: Influences the chances at which a player can actually hit the enemy. A simple formula will be used to grant the player a better chance of hitting the enemy if the player’s dexterity is higher than the enemy’s agility. (Maybe have it so that the player can hit weaker monsters more often than stronger monsters too?)

Luck: The player will have a 5% chance of inflicting critical hits on the enemy regardless of class. For now, all critical hit attacks will do 1.25x damage. Luck will increase the chances of critical hits, as will as increase the chances of crafting materials.

Equipping weapons will influence the maximum damage a player can do.

Equipping gear will increase the defense of the player.

## Item

Description: Base class for every item in the game. Derived classes include: Consumables, weapons/armor, quest items.

Items are split into different types: Consumables, weapons, gear, materials, and key items.

Consumables: Items that grant an effect upon usage. Usually these will provide health or give a stat boost to a player. Consumables should have a variable for how many hitpoints it will recover to the player.

Weapons: Items that can be used to attack enemies. They will all provide a damage boost to the player. Some items are restricted to by class.

Gear: Items that are worn by the player. They will provide a defense boost and may be restricted by class.

Materials: For the most part these are items that are sold for gold. However, some materials are needed to craft or refine items.

Key Item: These are like materials, but they will trigger an event however they are used. These are simply classified as such to differentiate them from materials.

Note: ALL types of items except for key items must have a gold valued attached to it.

## Shop

Description: Every RPG needs a shop so that a player can purchase goodies from it. These shops will contain items that the player can buy whether it be weapons or consumables. A basic shop should have the capability of selling any item in the game. These shops can also be read into a file so that they are loaded and preset with the said items when the player enters the shop. The shop will also be upgraded as the player progresses through the game.

## Contestants

Description: Aside from the player and entities, there are contestants. Contestants are a mix of the player and entity class in that they can encountered as an enemy as well as have the ability to have a job and inventory like a player. This is the shell of our A.I.: A class that will house essentially what is in the player class and more. The difference with working on this and the player is the need to create a generator for contestants. This will generate that base stats that the contestant will have, the job that will suit these stats (or not…), and the personality the contestant will have.

## Towns

Description: There will be one main town that the player will be in. This town functions as any RPG town, but it can be upgradeable. The higher level the town is, the more things you can do in the town. For now, the town will be utilized as a place to restore health, buy goods,and take on quests. There will be more stuff added to it later.

Expanding on towns: A town object will simply implement a town with its specific features. Within the class will be functions that will call upon other functions (shops, dungeons) to provide options for the player to do things. However, there are other options in towns that will be housed within its class because it would be easier to have it there. One example is a tavern, where players will use to rest, take on new quests, or check out the gossip around town.

## Entities

Description: The entities class will be extended to have more functionality for the game. A list will need to be made to make monsters so that they can be called upon other classes to spawn them. Entities will be more versatile with the ability to use skills and have specific stats put onto them.

## Time System

Description: When the player is put into the game, they will have certain options that will take some time for them to do in the game. This includes going to dungeons, crafting items, partaking specific quests, etc. This time system will serve as a way for players to spend some of their time before the day ends. To do this, the player must be able to have a specific amount of time “points” each day. Make sure that each action will take away points and move onto the next day once the player runs of time.

## Dungeons

Description: The player will go through different dungeons (or dimensions) throughout the game. These dungeons will range in difficulty and be in a list. The standard format for a dungeon should include the monsters that exist within it as well as its difficulty. The dungeon should also be able to initialize its appropriate map. More may be added to this.

## Map System

Description: When the player reaches a specific dungeon, there should be some map UI allowing the player to traverse through the whole area. The maps will be displayed as the playing moves through it. The design of these maps are still being decided upon, but they need to be updated each time the player moves. The player should be able to have the option to move in all directions of the map (excluding diagonal-like movements). Also, there should be obstacles and certain ways to get through the map (like rooms).

## Menus

Description: *Note: This relates particularly to the title screen menu and I will probably just do this unless someone really wants to do it.* Since we’re out of our prototype phase, we need a new game menu. This will be put in the main class and will consist of making a new game, saving, loading, adding credits, and exiting the game. More options will be added when necessary.

There is also a need to add other menus for the game, but I think those who are working on their assigned parts can make the menus themselves.

### Basic Menu

If the player types ‘menu’ at any point of the game, they will be able to access a basic menu which will include the following:

-A party list

-Inventory viewing

-Equipments

-Saving

## Quests

Description: Every RPG is never complete without quests, so we need these in order to make the player play longer than they need to. With that, you will need to create a base class that will initialize the player into a specific quest. For now, you can create your own “quest” for the player to follow. This includes marking the player ready for the quest (if they choose to accept it), show the progress of said quest, and declaring when the player actually finishes the quest. Finally, a reward should be given to the player when the quest is completed.

Requirements: -Quest Log

-Checks for when quest is complete

-Quest Name/Description

-Rewards

Quick Example: A quest is given to the player requiring them to kill 5 green slimes. Finishing the quest will give the player 500 EXP.

From this example, I want to be able to give the quest (for now just display the quest details in a function) to the player, allow the player to accept it, and track the progress of said quest. Once the quest is finished I want to reward the player somehow.

Quests will be split into different types, but this will be covered later in development.

Technical Idea of Quests:

We have a quest, and a universal list of quests. Each quest is stored into a vector. This vector will be put into a town, where the tavern can easily access it. In our free for all mode (to be explained later), quests will be randomized in every run. They will be restricted by level (so you don’t hit an impossibly hard quest), and will not show up once completed (unless it is a REPEATABLE QUEST). We do this by making a new vector of completed quests in either the player or town. Once the player completes a quest, it will be pushed onto the vector. There will also be a ‘completed’ flag, which marks whether or not the quest is completed.

## Game (Or uh, Game Setup?)

This class setups the game itself, defining a specific player and respective game features to be put into this one file. This combines the whole game together so that each class can be used for one specific object.

# Game Progression

## Outline of Game Day

1. Title Screen
2. Character Creation
3. Opening Scene
4. Move into new town
5. Go to first dungeon
6. FIGHT
7. End the day.

## A Typical Day’s Rundown

1. Enter the town. Decide what options to do (buy items, farm for materials, talk to NPCs, go on dungeons)
2. Grind, finish specific quests, run into instances or events
3. Use up your action points (or not) and end the day.

## Going Into a Dungeon

**To do List:**

**-DEBUG EVERYTHING**

**-Fix things needed to be fixed**

**-Put inventory into a new class**

**-Create a list of items, weapons, and gear**

**-Run updates on player when they put on gear**

**-Add actual leveling**

**-More story/dungeons (?)**

**1. Update stats**

**2. Implement Quests**

**3. BALANCE**

**4. Add a new scene for quests**

**5. Display gold for some things**

**6. BALANCE**

**7. Inspect Item**

TO DO:

Make sure combat systems works FINE

More features to combat system

More enemy variety

Implement Skills/Classes

More dungeon variety

Time System?

Quests

More Town Features?

Longer storyline for demo

# Prototype Stuff

\*\*This is just archived descriptions of stuff that’s either in the game already or scrapped. May be used in the future.

The prototype we will be making for this break will showcase the combat system for the game. There will be a player with preset health and stats (but if we finish early we’ll allow the player to set the stats and more) fighting a monster that is easily killable. The prototype ends when the monster is defeated. There will also be another option to simulate a shop and stat distributions (and maybe the map system if we ever do it in time) as well.

The classes required for this are as follows:

**Health and Stats**

This class stores all health and stat values in the game. These get updated in many instances such as when the player is in combat, levels up, equips a new weapon, etc. Many classes will require this, so it is the foundation for many aspects of the game. THIS CLASS NEEDS TO BE DONE FIRST.

Simple: The health value needs to be incorporated into the class as a function. It needs to be updated in any instance (Ex. When a player is takes or is healed) and be modifiable. Optionally, the framework for the stats can be set up (There will be 5; See the Complete version for more info). For now, you can simply establish one variable for the STR (strength) stat.

Complete: All the stat types (Strength, Agility, Vitality, Dexterity Luck) will the implemented in the game. They will all be empty variables for the most part, but there usages will vary in other classes. As with health, it should be modifiable. Here is a basic description of each stat:

Strength: How much attack the player can do to an enemy.

Agility: Used to increase the evasion rate of the player and whether or not the player attacks first.

Vitality: Increases the amount of health the player has, and possibly will provide the player with extra defense.

Dexterity: Increases the accuracy of the player.

Luck: Increases the chances of landing a critical hit to an enemy.

**Combat system**

NOTE: THIS CLASS REQUIRES THE HEALTH/STAT/MONSTER/PLAYER CLASSES BUT YOU CAN MAKE A TEST CLASS AND INCORPORATE THESE IN LATER.

This class will utilize the combat mechanics of the game. The player will be given a couple of choices (attack, defend, use an item, run) and the battle will commence in a turn-based fashion. The battle ends when either the player or the enemy reaches zero health. There will be two functions (for now) to be used for the class: One requiring the player to make a decision, and one simulating the combat phase.

Simple: Make the function that will prompt the player for input. The player will only need to choose from attacking or defending. Then, simulate the instance where the player attacks the enemy. This is where the class may become complex, but since we aren’t really incorporating stats and other things that will influence the game’s combat you do not need to do much. Here are the steps for simulating combat for the player:

1. Prompt the player to make a decision.
2. Calculate damage that the player and enemy will do
3. Decide who attacks first (we will compare the levels of the player/enemy)
4. Start the attack phase and subtract the amount of damage taken from the player and enemy.
5. Repeat the cycle again.

If the player is defending, do not make the player attack. The player will take reduced damage instead. Also, since we need some way to influence how much damage a player makes, a random number generator function will need to be made. For now, the damage the player makes will be influenced by their level.

Another person can work on outputting the messages for combat and display necessary information for battle.

Complete: This makes the combat system VERY complicated because we have to put in stats and the other decisions. For now, we won’t worry about this until later.

**Entity**

NOTE: THIS REQUIRES THE HEALTH/STAT CLASS

This class will simply be a universal class that will spawn a monster of a certain health and level. Eventually we will make another monster class that will be list of monsters in the game with health, stats, and levels.

Simple: Make the monster function and give it health and a specific level. This monster should be able to be “spawned” in the main class and be used in the combat system.

Complete: Add stats to the monster along with how much experience it’ll give to the player.

**Player**

NOTE: THIS REQUIRES THE HEALTH/STAT CLASS

This is another universal class that simply is a “shell” for the player. All the player’s stats, health, level and more will go here. When the player levels up, its stats will be updated in this file.

Simple: Make a player function, giving it health and a specific level.

Complete: Add stats to the player, experience, inventory, and more.

**SaveData**

NOTE: THIS REQUIRES THE PLAYER CLASS BUT YOU CAN MAKE A TEST CLASS TO TEST THIS

We will want to save the data of our player into this class, and output it onto a text file. Basically, everything that is in the player class file will be saved and be used to load once more at anytime. This class does both the saving and the loading functions. While this isn’t really required, it’ll be nice to add onto the prototype.

Simple/Complete: Incorporate the save/load features of the game.

**Shop**

In the game a shop will be implemented, allowing the player to buy equipment, goods, etc. For the prototype this will be included a demo feature for now.

Simple: Create a shop with an assortment of random items that they player can buy. For now, you can preset the amount of money the player has and subtract the amount from the player when he/she buys an item.

Complete: Complete the shop by calling upon a list of items with set prices. The money should be called upon the player class and the player should receive the item when he/she buys it.

**Items**

Items exist in many forms, ranging from stuff that helps aid the player to gear. This class will simply store all of these items and categorize each item to suit their purposes.

Simple: Create a list of random items that can be called upon other classes.

Complete: Categorize each items and provide the relevant information/parameters for each item.

**Main**

This is where all the functions and classes are strung together into the prototype. The main class will be done last. It’ll just be a simple file that prompts the user to start the prototype and/or try out the other demos in it.

**Waves\***

When the prototype starts we need a class that actually sets up the waves for the player. This means that as the player progresses through each battle, there should be a counter that notifies the player what wave he/she is on.

Simple/Complete: Make a function that allows the player to move on when all the enemies are defeated. You may perhaps spawn harder monsters within each wave. There should be a message on wave completions and starting off a new wave.

**Map**\*

This is essentially the visual aid given to the player when they traverse through different areas. Possibly the hardest class to work on, we will most likely not do this till after break.

Simple: Generate a simple map.

Complete: Generate maps that the player can traverse, updating their location with each movement.

\*Would not recommend doing; May not be done due to time constraints

I have another idea for the prototype which relates closely to what we will be doing for our game. It consists of the player running through a single dungeon level fighting monsters and gaining loot along the way. The same combat system will apply, and the player’s goal is to reach the end. However, I don’t really have an idea of how to make the map system work, so it may take too long to finish.