**Game Planning - Rougelike**

Structure:

* Load program
  + Main Menu
    - Settings
    - Load game
    - New game
  + Load game state
    - Main game loop
      * Look for input
      * Update game state
      * Render
    - In game menu
      * Save and quit
      * Resume

**Considerations:**

* Setting
* Game board
* Game Engine
* Combat
* Player
* Enemies
* Player/enemy states
* Entering and exiting rooms
* Loading rooms
* Items

**Setting**

The legends of the Mabinogion are coming back to life. You are a Welsh Indiana Jones investigating rumors of strange goings on near your rural village. This means that you will have modern items as weapons and ability items (e.g. taser).

**Game Board**

A series of rooms, generated as the player progresses.

The rooms should be random dimensions within certain boundaries.

Each room has an entrance on the left and an exit on the right.

Each room has a number of enemies and/or items.

**Possible gameboard implementation:**

Gameboard object containing:

* The player
* An ordered array of room objects containing:
* 2d array of “tile” objects to describe the room layout and floor type (wall, door, floor, (lava?) etc)
* Array of items on the floor
* Array of enemies in the room

The gameboard object would be passed to wherever it is needed, e.g. to an enemy’s AI methods, to the rendering engine.

**Game Engine**

Order of events:

* Player actions (choice of one action per round)
  + Move
  + Attack
  + Use item
* Update render engine
* Cycle through each enemy on the board in turn. For each:
  + If it is active then call the object’s updateEnemy method.
  + Check the game over conditions.
  + Render the game state

Note: if we implement weapon speed then some enemies will move before the player.

**Combat**

Important part of the game, needs some decisions made before we start coding.

Possible systems:

* Classic HP/MP - Both could regen by a small amount with each new room entered or they could require potions to regen. Attacks would do a certain amount of damage, mitigated by armor.
* Zelda style very few hit points where each time struck is 1hp lost. Would make gear less interesting though.
* HP but no MP, instead special abilities (linked to items?) have a certain cooldown (again measured in rooms?).

Also worth noting that depending on the setting and the type of character, attacks could be ranged or melee. E.g. in a sci-fi setting ranged guns.

There needs to be a way of deciding which enemies can be hit depending on range of the weapon and cover, should be an interesting algorithm to make.

**Player**

**Possible implementation:**

Player object containing:

* HP
* MP (if using a classic magic system)
* Location (room number)
* Position in room (x,y coordinates)
* Buffs
* Debuffs
* Gear/weapons
* Rendering

**Enemies**

Enemies will be an interface class containing a load of methods that can be implemented by any subsequent enemy object. Each enemy will update by calling an update method and passing a reference to the game board to it. This way different enemies can have totally different AI routines and access to all the information needed to act.

**Possible implementation:**

An interface class with the methods

* attack (called when the player attacks it, should pass attack info with it such as “attack power” and “attack type” to take into account types of resistances and armor).
* updateAction

**Player States/Enemy States**

**! This is something to think about later in development. Keep it in mind when designing to make sure we don’t make it impossible to do.**

Need a way of implementing states like burning, slow, empowered, stuck etc. Caused by player trinkets and enemy attacks.

Perhaps a way of implementing this is creating a “state” class that can be held in a collection by each actor in the game. The states could take their effect at the end of each round or simply be included in damage calculations.

**Entering and exiting rooms**

After player movement check to see if player is on a door tile. If so, move to the entrance of the next room. Some enemies should also be able to move between rooms.

**Loading Rooms**

Room factory class. It should receive a request for a new random room at a certain difficulty level and return a constructed room object.

The inner workings of the room factory will be hard coded random generators at first but could later be developed to use a database to create more intricate rooms and special “boss” rooms.

The rooms need to be populated with a combination of enemies, loot and health potions. Most of the rooms will have enemies.

Loot and potions should drop less often but fairly regularly - could use a semi random distribution (as in for every room without a drop, the odds of one being in the next one increase).

**Items**

Items consist of gear and potions.

Gear has a specific slot where it fits. Properties could include armor, max HP, “magic” resistance, damage, and special abilities.

The item factory class should create random items from tables of possibilities.

E.g. Armor table

|  |  |  |  |
| --- | --- | --- | --- |
| **Slot** | **Quality (0-100)** | **Main Stat** | **Special Ability\*** |
| Head | Broken | Strength (attack power) | Invincible for x turns |
| Chest | Poor | Defence (armor) |  |
| Legs | Unremarkable | Health (HP) |  |
| Hands | Sturdy | Magic/ability resist |  |
| Feet | Quality |  |  |
| Trinket | Fine |  |  |
|  | Epic |  |  |
|  | Legendary |  |  |

**\*might be randomly triggered on attack or have a key assigned to triggering it**

So for example, in a low level room you might find an Unremarkable Chest Plate of Health, the quality of the item would mean that it gives quite a small amount of HP.

**Weapon**

Given the turn based grid system we could have the following properties of weapons:

* Raw damage
* Range (in squares)
* Attack speed (this could determine the order of moves in a round, faster speed means attacking/moving before an enemy)
* Additional effect (knockback, immolate, freeze, lifesteal, stun)
* Quality (same as the armor system, affects how effective the damage and additional effect is).

Types of weapon could include:

* Spear
  + Range: 2
  + Damage: medium
  + Speed: medium
* Club
  + Range: 1
  + Damage medium
  + Speed slow
  + Special: knockback/stun
* Sword
  + Range 1
  + Damage medium
  + Speed medium
* Dagger
  + Range 1
  + Damage low
  + Speed fast
* Bow
  + Range 10
  + Damage low
  + Speed low
* Plasma rifle (why the fuck not lol)
  + Range 100
  + Damage high
  + Speed medium
* Flamethrower
  + Range 5
  + Damage medium
  + Speed fast
  + Special: Immolation, AoE damage

...etc etc etc. E.g. Sturdy Sword of Immolation (because who doesn’t love a burning sword?)

By creating a weapon interface we can simply add more and more weapons to the factory class as we see fit without having to change any other code.

For simplicity no inventory screen needed. Just a description and an option to equip when standing on it.

Rendering

Keeping the rendering completely separate from the game engine means it can be represented however we like, upgrading as we see fit. To start with the rendering will be done using ascii characters. Each enemy could have a specific letter attached to it, turns capital when highlighted.

E.g.

Room 10

X-----------------X

| e | HP: 137/220

\_| i | Armor: 34

\_ @ | Damage: 23

| E | Speed: 50

| | Special: Knockback, freeze immunity

| e |\_

| e \_

| i | Item description when standing on it.

X-----------------X Pickup item? (y/n)

Move: wasd

Attack: Tab through enemies and space to attack

Combat log:

Angry Squirrel dies

You burn Angry Squirrel for 20 damage

Angry Squirrel hit you for 5 damage