Game Design Document: Random Tanks

A Rogue-like Game, With Procedurally Generated Maps

Foreseeable Complex Elements

Procedural Dungeons
- The Player will be tasked with finding and killing a boss creature on each floor of a generated dungeon
- Doors of any given room are closed until all enemies have been killed in the room
- Rooms may drop/contain pseudo-randomised loot upon completion
- A floor is generated once each time the player completes the previous floor, with most of each rooms contents predetermined on floor generation

Frequent Model Changes/Rendering

Decomposition of Item Effects

Generating Contents of Rooms

Development of Testing Tools
- Creating any single room
- Give character any set of items

Should all be accessible from a debug menu

Introduction to the Concept

In a rogue-like game, the player character traverses a generated floor of a dungeon, containing many rooms that can be explored and looted. The character develops through finding, creating or looting objects that forge the player's play-style in some way, by making them faster, stronger, or, in some other way, better (or worse). Players must decide which items they wish to pick up, depending on their intended play-style.

In this iteration within the genre, i wish to create an evolving character, as usual, with simple graphics reminiscent of the *.IO branch of online games, often involving simple, coloured blobs. The colour, shape, and ‘weapon’ of the character may change over time, indicating how the character works to the player, as they try to challenge other similar, autonomous enemy units scattered throughout the level. Each level may see an increase in difficulty.

Item Decomposition

Items will give some key components, which will be a raw numerical/discrete value (likely as a String) and %values. The final effect of the stat change will be the raw value multiplied by the %value. A discrete stat change could be something like an on-hit effect, with some %chance of happening, or a form that player takes, for example having equidistant or parallel weapons. There must be a list of possible effects/stats a user can have.
Stats that can change:

- Discrete [Possible Values] *multiple possible:
  - Player On-Hit: [Poison/Frost/Fire/Vampiric/Thorns]*
    - A list of [effect,%chance] values
  - Turret Form: [Parallel/Equidistant]
  - Turret Type: [Basic/Piercing/Shotgun]
  - Turret On-Hit: [Poison/Frost/Fire/Vampiric]*
    - A list of [effect,%chance] values

- Continuous (Range of values):
  - Maximum Health (1-15) - Measured as quarter hearts - Integer only
  - Damage (1 - 15) - Measured as quarter hearts - Integer only
  - Rate of Fire (0.1 - 10) - Measured as projectiles per second
  - Projectile Speed (0.5 - 10) - Measured as tiles per second
  - Movement Speed (0.5 - 10) - Measured as tiles per second
  - Rotation Speed (10 - 180) - Measured as degrees per second

All stats will be stored as their value and %value terms, and then re-calculated when either of these stats are affected. This means rounding each time does not affect the long term effect of %value increases. Every item that has effects will have a combination of the above, making the number of methods I have to write to handle a potentially infinite number of items minimal.

Projectiles will likely work like visitors, where colliding with another entity that has health will cause it to take damage by visiting the object, and applying its damage calculation. Any object this projectile hits should be visited. If the projectile is not a piercing projectile, it should self-destruct upon first collision.