# Python Turn-Based RPG

## Project Description

Using pygame, I will make a turn-based RPG in a similar style to early Final Fantasy or Dragon Quest games. The player moves on a grid in the overworld and is transported to a separate battle screen in they run into an encounter.

The demo I will produce for the final submission will consist of a single dungeon. Features of this dungeon include:

* A room that generates a maze when you enter it
* (at least) Three regular enemies
* One boss

The battle system will include a “counterattack” feature. When using this command on an enemy, the player is presented with a list of the possible actions the enemy can take. If the player guesses correctly, they will nullify the enemy’s turn and deal lots of damage to that enemy.

In order to make this battle system more fun, the enemy will also try to predict the player’s actions and act accordingly. Different enemies will have different personalities: some enemies prefer to counter, others will defend, yet others will try and attack and defeat you first before you can make your move.

## Competitive Analysis

On GitHub, I found a project called [The Stolen Crown: A mini-RPG](https://github.com/justinmeister/The-Stolen-Crown-RPG).

This project, like mine, uses states to organize its code. Each state is in its own file in the “data/states” directory. Interestingly, the GUI code is not in its own folder and is unorganized in the “data” directory.

As far as game mechanics go, this game seems to be much simpler battle-wise. There is only one player character, unlike my game which has two.

However, this game also implemented an item/inventory and shop system, which I have no plans for. This game also has a more complicated overworld system, while I am only implementing one dungeon. However, the overworld here appears to be hardcoded, while parts of my dungeon will be randomly generated.

## Structural Plan

The game runs in three main states, with sub-states as necessary:

* Title Screen
* Overworld
  + Walking
  + Dialogue/reading a sign
* Battle
  + Various menus
  + Battle animations playing out
  + Game over

Code related to each state will be stored in files in the root directory, unless there end up being more files than I anticipate in which case each state will get its own directory.

Game data such as sprites, overworld room data, enemy data, and player battle data will have their own directories.

## Algorithmic Plan

There are two tricky parts to this project: the maze generator and the enemy AI.

### Maze Generator

This will be run with a maze generator algorithm. In order to be a valid maze, there has to be a clear path from the bottom of the maze to the top.

Maze squares consist of nine spaces each. # marks will always be walls. H marks will be hallways if there’s a connecting adjacent room. \_ marks will always be open.

|  |  |  |
| --- | --- | --- |
| # | H | # |
| H | \_ | H |
| # | H | # |

The maze generator will use a random number generator and a backtracking recursive function to generate the maze. It will, at each open space in the maze, pull three random bits and use them to determine whether a hallway connects to the current room there. If no open spaces are available, it will backtrack and continue at any remaining open spaces.

### Enemy AI

The enemy AI will have two main parts: prediction and decision.

Whenever the player takes their turn, what they did is recorded to a file in the game’s memory. The game will then read this file to try and determine what the player will do on their next turn. Data points to be used in decision making include:

* Whether the player’s characters are low on health
* Whether the enemies are low on health
* What the player did the turn prior to the current turn
* The order in which battle actions are executed
* How long ago the data point being examine took place
* How likely the predicted action is to occur

In order to save memory and improve run time, the enemy AI will only save the last ten battles worth of data (this figure may need to be adjusted during testing). This also prevents the AI from using outdated info to make predictions, as the player’s tactics will probably change over time.

Enemies will have different “personalities” – different kinds of enemies will do different actions in response to similar predictions. Furthermore, enemies may behave differently if they are low on health. The following are a few kinds of behavior patterns I brainstormed. Note that not all of these patterns may be used in the final project.

* “Offensive”. On the first turn, if it has a strengthening move, prefers to use one. After that, prefers to attack, but may counterattack if it is reasonably sure of the player’s intentions.
* “Aggressive”. Only ever attacks, no counterattacks. Bases most of its decisions on who moves first and combatant health.
* “Healer”. If about to be attacked, defend. Else, if enemies are low on health, heal them up. Else, attack.
* “Shutdown”. If it predicts a powerful attack coming, prefers to immobilize the attacker with status conditions. Otherwise, prefers to attack.
* “Support”. If possible, strengthen an ally. Else if possible, heal an ally. Else if about to be attacked, defend. Else, attack.
* “Erratic”. Makes actions based purely on the random number generator. This personality will probably only be used for enemies that are low on HP.
* “Loafer”. Prefers to waste turns if not expecting to be attacked. Otherwise, prefers to attack or counterattack. This personality is useful for not overwhelming the player if they end up in a bad spot.

## Timeline Plan

* Nov. 21: Maze generator mostly done or finished
* Nov. 22: Title screen, preliminary battle states and basic GUI implemented
* Nov. 23: Battles fully completable, preliminary predictive battle system
* Nov. 24: Continue predictive battle system
* Nov. 25: Finish predictive battle system
* Nov. 26: Fine-tuning previous elements, minimum viable product

## Version Control Plan

I would like to learn Git for this project. Until I figure out how to use it, I will save each code iteration as a new file.

## Module List

* Pygame
* Lukas Peraza’s Pygamegame object

# TP3 Update

A reduction in scope, since I dreamt too big in this proposal:

* Only four personalities implemented (“erratic”, “aggressive”, “offensive”, and “grand\_wizard”). “Erratic” is not currently used by any enemies, since it would be too similar to “aggressive” with the current features.
* All battles are one-on-one.
* Enemy AI only takes into account previous turn history, although since it was thoroughly beating me during testing, that might be OK…
* The player can’t counterattack. Gameplay still requires predicting the AI’s behavior, though.