Dungeon Crawler Game

Cohort 4 Group 2: Bryan, Ryan, Colin, Muzi, Joseph

Video Demonstration

Setup

From Github

Download file and unzip. Run DungeonCrawler.py to start the game.

From Repl.it

Click Run at the top of the page to start the game.

Game Description

Background of Genre

Inspired by the popular fantasy roleplaying table top game of Dungeons and Dragons, Dungeon Crawlers simulates the navigation of a dungeon in which the protagonist slays monsters and aquires loots in an attempt to fulfil a certain objective (killing the main antagonist or escaping). The randomised nature of each run via procedually generated map layouts, enemies and loot results in emergent gameplay where the circumstance of each playthrough is fresh and unique.(Due to time limiations however, we opted not to include procedural generation.) The potential for replayability and relative simplicity of implementation (generally no need for fancy graphics or computations) makes Dungeon Crawlers popular even among older systems.

Objectives

The objective of this game is to find the dungeon key and escape from the dungeon in the least amount of turns without dying from starvation or combat while carrying as much gold as possible. Players that escape will be given a score that can be compared with others to determine whoever had the best playthroughs.

Controls

Available actions are shown at all time during the game in the form [UserInput:ResultantAction]. Type one of the keys and press enter to perform/select the corresponding action/choice. Inputs are always case insensitive. Press enter to continue whenever prompted.

In general, the controls are as follows (unless otherwise specified in-game):

Selection controls:

- `1-9` (or more) to choose from a list of available items/choices (Browsing shop, casting skills, using items etc.).
- `X` to leave/go back from a menu if possible.

Map Controls:

- `WASD` to move up, left, down and right respectively.
- `M` to view map
- `I` to access inventory
- `E` to check equipment and unequip
- `C` to check player profile/stats
- `Q` to quit.

Combat Controls:

- `A` to attack
- `W` to wait
- `S` to cast a skill
- `D` to describe the enemy
- `I` to access the inventory for using an item
- `E` to check equipement and/or unequip them
- `R` to attempt to run away.

Attributes

- `Name` : Character name
- `Class` : Character class
- `Health` : Player loses when current health reaches 0
- `Attack` : Current attack determines raw damage dealt to enemies
- `Defence` : Current defence negates incoming attack by a given amount
- `Speed` : Increases dodge, determines turn order and affects chance to run away
- `Accuracy` : Affects chance to hit enemies with attack. Hit chance is calculated as ((Your Speed-Enemy Speed) + ((Your Accuracy-Enemy Dodge)/Your Accuracy))%, with a minimum hit chance of 5%
- `Dodge` : Affects chance to completely negate damage from enemy attacks

- **`Equipments`** : Shows currently equipped items in your various slots
- **`Status`** : Shows status effects that are currently affecting the character
- **`Skills`** : Shows skills that can be used in combat
- **`Inventory`** : Shows items currently in the inventory
- **`Gold`** : Can be spent on equipments and consumables when trading with merchants. Also determines final score
- **`Torch`** : Determines how far you can reveal the tiles around you as you traverse the Dungeon. Reaching 0 prevents the revealing of adjacent tile
- **`Food`** : High food levels allow for regeneration of health when moving across the map. Reaching 0 food results in starvation which causes health to degenerate instead

Map Tiles

- `P` : The player.
- `0` : Impassable Wall.
- `''` : Empty Space.
- `C` : Chest. Generates a random loot box for the player upon walking on it, after which it disappears.
- `M` : Merchant.Impassable. Buy and sell everything here.
- `E` : Exit. Move here with the key to win the game (or go to next level maybe? If that's ever implemented).
- `K` : Key. Required to exit the Dungeon.
- `S` : Slime: Weak but durable. Only drops torch fuel. 33 Health, 1-4 Attack. Easiest.
- `G` : Goblin: Typical enemy, variety of useful drops. 25 Health, 2-6 Attack. Easy.
- `D` : Dire Wolf: Glass cannon, low Health but high attack. Usually drops meat in form of food ration. 20 Health, 4-7 Attack, high dodge and accuracy. Medium.
- `H` : Hobgoblin: Strong version of goblin, fairly dangerous with good attack and defence. 40 Health, 4-8 Attack. Hard.
- `R` : Revenant: Strongest enemy,Dungeon Boss found blocking the exit. 50 Health 5-10 Attack. Very Hard.

Items

Weapons

- `Empty` : +0-0 Attack
- `Dagger` : +1-1 Attack
- `Gladius` : +2-1 Attack
- `Short Sword` : +1-3 Attack
- `Sword` : +2-3 Attack
- `Spear` : +1-5 Attack
- `Halberd` : +2-6 Attack
- `Longsword` : +3-5 Attack

Armor

- `Empty` : +0 Defence
- `Leather Armor` : +1 Defence
- `Chainmail` : +2 Defence
- `Scale Armor` : +3 Defence
- `Plate Armor` : +4 Defence
- `Dragonscale Armor` : +5 Defence

Trinket

- `empty` : No effect
- `Bracelet of Accuracy` : +5 Accuracy
- `Ring of Speed` : +2 Speed
- `Focus Shard` : +25 Accuracy, -5 Dodge, -1 Speed
- `Gloves of Haste` : +5 Dodge, +5 Speed, -10 Accuracy
- `Talisman of Evasion` : +10 Dodge, +1 Speed
- `Cloak of Darkness` : +25 Dodge, -15 Accuracy, -1 Speed
- `Obfuscating Shroud` : +40 Dodge, -25 Accuracy, -3 Speed

Consumables

Amount restored depends on size of consumable (Small/Normal/Large)

- `Health Potion` : Restores 15/25/35 health
- `Torch Fuel` : Restores 5/10/20 torch
- `Food Ration` : Restores 9/18/25 food and 2/3/5 health

Special

The `Dungeon Key` is a unique item that cannot be used on the map or during combat. Having it in your inventory is a requirement for exiting the Dungeon to win the game.

Map Loot (Chest)

Walking over the `C` tile on the map will cause the player to receive one of the following loot set (with equal chances for each set):

- `Treasure Chest` : High amount of gold, with the potential to provide a large variety of items
- `Box of Supplies` : Usually contains consumables like health potions, food and torch fuel.
- `Caverneer's Stash` : Almost always contains torch fuel, with a small chance of containing trinkets
- `Box of equipment` : Often contains at least one low-level equipment, with a small chance of containing equipments in greater qualities/quantities
- `Potion Pouch` : High chance of containing health potions

Classes

- `Warrior` : Good all-rounder. Can use Guard and Disarm.
- `Ranger` : Dextrous at the sacrifice of power and durability. Casts Poison and Inner Focus.
- `Beserker` : Powerful in combat but neglects defences and supplies. Can use Empower.
- `Survivalist` : Durable and starts with more supplies. Can Daze and Slow.

Skills

Can be used in combat, with different classes having different skills.

- `Guard` : Guards target, increasing defence. Used by Warriors.
- `Disarm` : Disarms target, decreasing attack. Used by Warriors.
- `Poison` : Poisons target, removing health each turn. Used by Rangers.
- `Heighten Senses` : Heighten senses of target, increasing target accuracy and dodge. Used by Rangers.
- `Empower` : Empowers target, increasing attack. Used by Beserkers.
- `Daze` : Dazes target, decreasing accuracy of target. Used by Survivalists.
- `Slow` : Slows target, decreasing dodge and speed of target. Used by Survivalists.

References

Hit chance formula modified from: https://www.gamedev.net/forums/topic/685930-the-simplest-but-most-effective-and-intuitive-way-to-implement-accuracy-and-dodge-chance-in-an-rpg
Basic map movement and updating learnt from: https://www.youtube.com/watch?v=G17XPI6t6kg

Documentation

Imports

Python modules:

```
 import math
from colorama import Fore, Style, Back
from random import randint, choice
from copy import deepcopy, copy
from time import sleep
```

Rationale:

- `math`, `randint` and `choice` are used whenever there is a need to calculate chance of events, from attack damage to loot received.
- `Fore` and `Style` from `colorama` are used to enhance visual experience, increase readiblity and provide a sense of reward/danger during positive/negative events. This is important especially in the absence of any other visual libraries.
- `deepcopy` and `copy` are used to generate fresh copies of enemies and player statistics from refrence dictionary variables. Also used to create a version of the map that the player sees.
- `sleep` is used to enhance gameplay experience by providing pauses during the display of relevant or important sets of information.

Custom modules:

```
 from mapLoot import mapLoot
from classes import classes
from maps import maps
from nonPlayableCharacters import nonPlayableCharacters
from items import weapon,armor,trinket,consumables,priceSheet
from statusSkill import *
```

Rationale: We created and imported variables and functions from other files in the game folder to allow for seperation of concerns and to prevent clutter in the main game file due to their heavy nesting/occuying a large space. It also allows for easy editing and referencing of variables.

Functions

Main

- `main()` : Runs the game. Pseudocode is as follows:
  1. Prints welcome texts and initializes game by prompting player for their name, class choice, map choice and initial purchases.
  2. Displays map and provides instructions.
  3. Enter into main game loop.
  4. Prompts the player for an action.
  5. Handles player action and any events that occur because of it.
  6. Checks if player action consumes turn.
  7. Upon turn consumption, updates map state depending on direction and player vision level.
  8. Updates player statistics based on current food and torch levels.
  9. Displays the updated map, a description of surrounding objects, vital player statistics and warns the player if their health or supplies are low.
  10. Loops back to step 3 until player quits, loses all their health or reaches the exit with the key

General-purpose

- `playerAction(availableActions)` : Takes in a dictionary of the form {UserInput:ResultantAction}, displays the available actions for players to choose, handles invalid inputs and returns an action in the form of a string.

- `handleInventoryDescription(inputCharacter)` : Accepts a character in the form of a dictionary, checks their inventory and prints out their items in a presentable format. Does not return anything.

- `handleSkillDescription(inputCharacter)` : Accepts a character in the form of a dictionary, checks their skills and prints out their skills in a presentable format. Does not return anything.

- `handleUse(inputCharacter, inputItem)` : Accepts a character in the dictionary form and an item represented by a string. Determines if item is to be consumed or equipped, updates character statistics as required and prints the result. Returns nothing.

Map-related

- `fogMap(inputMap)` : Takes in a map in the form of a list of lists and creates a fogged version of it.

- `printMap(map)` : Takes in a map in the form of a list-of-lists and prints it in a more human-readable format (Without punctuations etc.). Returns nothing.

- `describeSurroundings(inputPlayerMap,x,y)` : Accepts a map list-of-lists and the player's current coordinates. Prints a description of adjacent map tiles for every movement of the player. Returns nothing.

- `handleMerchant(inputPlayer, startOfGame=False)` : Takes in the player's character in dictionary form. Provides different interactions if `startOfgame=True` by displaying recommended starting purchases. Handles transactions with merchants by displaying useful information like player gold and purchasable items. Utilises the `playerAction(availableActions)` function to navigate the shop menu and to select items. Updates player inventory and gold for each successful transaction. Returns nothing.

- `updateFog(inputPlayerMap, inputCurrentMap, inputX, inputY, inputPrevX, inputPrevY, vision=0)` : Takes in the map that the player sees, the actual map current coordinates, previous coordinates and vision level of player that is determined by player torch levels. Reveals the 8 tiles surrounding the player on vision 2, the 4 adjacent tiles on vision 1 and the one tile the player steps into on vision 0. Returns an updated version of the map visible to the player.

Combat-related

- `handleTurnStart(inputCharacter)` : Used in combat. Takes in a character dictionary and updates their current combat statistics according to their modifiers. Returns nothing.

- `handleTurnEnd(inputCharacter)` : Takes in a character dictionary and updates it by resetting their modifiers, removing expired statuses and handling skill cooldowns. Returns nothing.

- `enemyAction(inputEnemey)` : Takes in an enemy character in the form of adictionary, check behaviour type and returns an appropriate action in the form of a string. Used during combat. (Currently only 1 type of behaviour is coded).

- `handleSkill(castedSkill,casterCharacter,targetCharacter)` : Takes in a skill in the form of a string and 2 dicionaries representing the caster and the target. Invokes the respective skill function on the target and causes caster's skill to go on cooldown. Returns nothing

- `handleUse(inputCharacter, inputItem)` : Accepts a character in the dictionary form an item represented by a string. Determines if item is to be consumed or equipped, updates character statistics as required and prints the result. Returns nothing.

- `handleEncounter(inputCharacter, inputNPC)` : Accepts 2 dictionaries: the player character and the NPC. Initiates combat if the NPC is hostile. Returns an outcome. Pseudocode for combat is as follows:

  1. Creates a copy of the inputNPC for player to fight.
  2. Compare speed to see if player or enemy acts first.
  3. Resets skill cooldown and statuses of player.
  4. Combat loop begins.
  5. `handleTurnStart()` function is invoked for both characters.
  6. Combat log is shown and player is prompted for an action.
  7. Handles player actions and check if resulting action consumes a turn after which the enemy will act.
  8. `handleTurnEnd()` function is invoked for both characters.
  9. Combat-ending events are checked.
  10. Loops back to step 5 until conditions for ending combat are met.
  11. Returns an outcome in the form of a string (`"victory"`, `"defeat"` or `"retreated"`).