Testing Plan

Scope of tests

The scope of our tests are to test all functions in the backend. Testing the frontend, stress, and performance are out of our testing scope

Testing Types

Throughout the duration of testing, we aim to use testing types such as unit, integration and system tests.

Requirements

- Unit and integration tests that ensure the correctness of individual aspects of your model
- System tests, tests on the controller methods to ensure the backend works correctly at the level of abstraction
- You will need to write your code test-first and have git commits which show evidence of this
- Reach 85% **branch coverage**.
- Usability test your code (run the game on the frontend and play it).

Test Schedule

1. Create interfaces and classes with method signatures and pre/post conditions
2. Create test sets
3. Method implementation

Test Sets

Class/Function	**Tests**
newGame()	Parsing json dungeon and config file names and then unit test via asserting value from function output is equal to expected value.
tick(String)	Create mock objects and functionality by adding items to players inventory. Executing function then unit test by asserting player's state (when using potion) or

	asserting entities on map (when using bomb)
tick(Direction)	Testing for all moving entities that are involved in tick(Direction) How to test: via lots of print statements. Spider ● Need to ensure spiders change direction when running into something it cant pass ● Need to make sure it can pass items it can pass ● Needs to make sure that to test it moves in a circular direction ● Ensure that the spider spawns ever 2 ticks ● Testing that ensures that when the spider move INTO a player that it will commence a battle Merc ● Need to test that the merc chooses the best path -> dijkstra? Bfs? Need to get a pred and distance list to determine it takes the shortest path zombieToaster/spawner ● Test zombies move randomly and will run away when the player is invisible ● Zombie toasters -> test that it can't pass boulders and that it shouldn't be able to push them All can be tested by checking the positional patterns of the enemies each tick
build(String)	Mock objects and functionality to mimic player picking up items, by adding mock objects to player's inventory. Execute function then unit test to assert buildable item exists in player's inventory and recipe items are removed
interact(String)	Merc ● Need to test that an error if thrown if merc is not

	in radius and if the player doesn't have money to bribe it ● Need to test that is bribe is successful, the the player loses money and that the zombie does become an ally ● Need to test that ally behavior to correctly implemented -> like when it's next to a player, it should battle it, it should walk onto the players position, it should always take take players previous step ZombieToastSpawner ● Have to test that the toaster will disappear when its destroyed ● Have to test what can destroy the toaster -> what weapons can destroy it?
Boulder	Tested movement of boulder by moving player and comparing expected position of player and actual position of player(different). Test a blocked boulder by moving into a blocked boulder and comparing initial position and actual position(should be the same).
Portal	Tested simple success case of portal teleport by moving player into portal with given direction. Expected position is the direction of the linked portal and is compared with actual position. Same for chained portals but getting the position of the final portal + direction. Tested for blocked positions by blocking the initial teleport position and checking the cardinally adjacent tiles. Tested for completely blocked portals by surrounding the linked portal with walls and checking the player position is the same as the portal it moved into.
ZombieToasterSpawner	Similar to a portal in terms of blocking. Checked if zombie was spawned in cardinal positions after spawnRate ticks.
Door	Created a dungeon with multiple doors and keys. Tested if a door could be opened with the right key, only one key could be held in inventory by checking player inventory after moving pass 2 keys. Tested that

	the wrong key would not open a door.
Animation	Visually inspect animation in the frontend
SwampTile	Tested all moving entities working as intended with differing multiplying factors. Checked boulders and players are not affected by swamp tile.