DUNGEON MANIA TESTING PLAN

H13B APPLE PIE

CREATE GAME/DUNGEON (1 POINT)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Test dungeon creates correct name and ID fields Input Space: - The given JSON files	New Dungeon-Controller	Call dungeon constructor passing in file paths	New instance of dungeon created and has correct ID and Name
Unit	Test dungeon creates a correct mock list of entities from the given pathname Input Space: - The given JSON files	New Dungeon-Controller	Call dungeon constructor passing in file paths	New instance of dungeon created and has matching (mock entities)
System	Test newGame with dungeon and config files and test dungeonResponse is created with mock entities Input Space: - The given JSON files	New Dungeon-Controller	Call dungeon constructor passing in file paths. Then request dungeon-Response from controller	New instance of dungeon created and returns dungeon response with matching (mock) entities

CREATE PLAYER CLASS (3 POINT)

Test Type	Test	PreCondition	Action	PostCondition
System	Created on newGame with correct AD[1] and health values from	New Dungeon-Controller	Call newGame and check getPlayer	Player exists and has correct values as listed

[1] AD: Attack Damage

	config file Input Space: - The given JSON files		(dungeon method)	in JSON file
System	Created on newGame with correct starting position from config file Input Space: - The given JSON files	New Dungeon-Controller	Call newGame and check getPlayer (dungeon method) and getCurrent-Position	Player exists and has correct starting position as listed in JSON file
Unit	Stores a mock inventory and returns it as a List<Entities> (for now since CollectableEntities not implemented) Input Space: - Mock Entities following Entity superclass	New Player	Add mock entities to list and call getInventory()	Entities all present and returned as a List<Entities>
Unit	Generates correct EntityResponse Input Space: - Player Position	New Player with mock inventory	call getEntity-Response	Name, ID, Position, and isInteractable are correct values

IMPLEMENT PLAYER MOVEMENT (3 POINT)

(Collision tested with other respective entities)

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Unit	Test movement in all directions with no collision conditions Input Space: - Direction	Player exists in empty map	Call Player.move	Player position translated as expected

[1] AD: Attack Damage

Unit	Test movement with simulation collision Input Space: - Direction	Player exists in new map	Call Player.move and Player.goBack	Player is in starting position
Integration	Test movement from controller Input Space: - Direction	DungeonController has called newGame	Call tick(Direction)	Player position translated as expected

CREATE ENTITY CLASS (1 POINT)

(This is difficult to test as will be an abstract class)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Test getEntityResponse generated correctly Input Space: - Mock Entity	Entity constructed (temporarily non abstract)	Call getEntityResponse	Data in response is correct

CREATE STATIC ENTITY CLASS (2 POINT)

CREATE COLLECTABLE ENTITY CLASS (2 POINT)

CREATE MOVABLE ENTITY CLASS (2 POINT)

(This is difficult to test as will be an abstract class)

WALLS (1 POINT)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Create a wall with correct position Input Space: - mock Wall entity JSON data	Entity constructed (temporarily non abstract)	Call getEntityResponse	Data in response is correct
System	Create a wall with	DungeonControl	Call newGame	Check in

[1] AD: Attack Damage

	correct position from string in controller Input Space: - Dungeon JSON containing a wall	ler exists	with a valid map	dungeonRespon se for wall and check instanceof and position
Integration	Update player on collision Input Space: - Player and Wall dungeon	DungeonControl ler exists with newGame called on a valid map	Move player into wall	Position should remain at position adjacent to wall and in player previous path
Integration	Update enemies on collision (when they are implemented)			
Integration	Update boulders on collision (when they are implemented)			

MOVEMENT INTERFACE (3 POINT)

(Inappropriate to test interface on its own, first few tests with implementing classes should indicate success)

IMPLEMENT MOVEMENT BEHAVIOUR: SPIDERMOVEMENT(2 POINT)

(Will be tested with spider ticket)

SPIDER (3 POINT)

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
System	Test random spawning on ticks Input Space: - config file with spider spawn rates	New DungeonControl ler with newGame called	Move Player to cause tick (only tick(Direction) implemented currently)	Entities list size should increase by 1 every tick since no other spawning behaviour implemented
System	Test spider stats from config	New DungeonControl	get stats of spider	Check values match config

[1] AD: Attack Damage

	Input Space: - Valid config files	ler with newGame called on a map containing an existing spider		files
Integration	Test first movement after spawn Input Space: - Valid dungeon file with existing spider	DungeonController exists with newGame called on a valid map	Move player once (1 tick)	Position of spider should be directly above spawning position
Integration	Test movements after initial movement Input Space: - Valid dungeon file with existing spider	DungeonController exists with newGame called on a valid map	Move player (increment ticks) at least 9 times	Position of spider should follow movement pattern specified in spec
Integration	Test movements with boulder in path Input Space: - Valid dungeon file with existing spider and boulder and path	DungeonController exists with newGame called on a valid map	Move player (increment ticks) at least 9 times	Position of spider should follow movement pattern specified in spec

[1] AD: Attack Damage

INVISIBILITY POTION AND STATE (3 POINT)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Whitebox Test switching states on player directly Input Space: - mock Player	New Player	Switch player state to invisible then back to normal using PlayerState methods	PlayerState should go to invisible state then back to normal
Unit	Test switching states based on controller and timer Input Space: - mock Player and artificial potion timer input	New dungeonControl ler calling newGame on valid dungeon and config files	Switch player state and apply artificial potion timer. Then cycle ticks and assertions	Should be in invisible state for 2 ticks then on third tick back to normal state
Unit	Test item response is generated correctly Input Space: - mock Potion JSON	New Potion from mock data	call getItemRespons e from potion	itemResponse should be equal to what is expected
Integration	Test item is added to inventory on pick up Input Space: - Valid dungeon file with existing potion	DungeonControl ler exists with newGame called on a valid map	Move player to potion position	potion should be deleted from entity list and added to player inventory
Integration	Test apply potion directly from potion Input Space: - Valid dungeon file with existing potion	DungeonControl ler exists with newGame called on a valid map and player has picked up potion	call potion.use(Playe r)	Should be added to statusEffects and on next tick will switch player to invisibility state
System	Test apply potion from controller tick method	DungeonControl ler exists with	call tick(String)	potion playerState

[1] AD: Attack Damage

	Input Space: - Valid dungeon file with existing potion	newGame called on a valid map and player has picked up potion		should change to invisibility
Integration	Test Queuing of potions Input Space: - Valid dungeon file with at least 2 existing potion - config file with potion duration of 5	DungeonController exists with newGame called on a valid map and player has picked up potions	call tick(String) until potions wear off	playerState should be correct on every tick

INVINCIBILITY POTION AND STATE (3 POINT)

(Same tests as above with integration test using/queue both types of potions)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Whitebox Test switching states on player directly Input Space: - mock Player	New Player	Switch player state to invincible then back to normal using PlayerState methods	PlayerState should go to invincible state then back to normal
Unit	Test switching states based on controller and timer Input Space: - mock Player and artificial potion timer input	New dungeonController calling newGame on valid dungeon and config files	Switch player state and apply artificial potion timer. Then cycle ticks and assertions	Should be in invisible state for 2 ticks then on third tick back to normal state
Unit	Test item response is generated correctly Input Space: - mock Potion	New Potion from mock data	call getItemResponse from potion	itemResponse should be equal to what is expected

[1] AD: Attack Damage

	JSON			
Integration	Test item is added to inventory on pick up Input Space: - Valid dungeon file with existing potion	DungeonController exists with newGame called on a valid map	Move player to potion position	potion should be deleted from entity list and added to player inventory
Integration	Test apply potion directly from potion Input Space: - Valid dungeon file with existing potion	DungeonController exists with newGame called on a valid map and player has picked up potion	call potion.use(Player)	Should be added to statusEffects and on next tick will switch player to invincibility state
System	Test apply potion from controller tick method Input Space: - Valid dungeon file with existing potion	DungeonController exists with newGame called on a valid map and player has picked up potion	call tick(String)	potion playerState should change to invincibility
Integration	Test Queuing of different potion types Input Space: - Valid dungeon file with at least 2 existing potion of different types - config file with potion duration of 5	DungeonController exists with newGame called on a valid map and player has picked up potions	call tick(String) until potions wear off	playerState should be correct on every tick

[1] AD: Attack Damage

COMPLEX GOALS (2 POINT)

Test Type	Test	PreCondition	Action	PostCondition
System	Test controller can handle a dungeon input with complex goal and print the correct goal syntax in dungeon response Input Space: - dungeon JSON with complex goal	New Dungeon-Controller	Call newGame passing in file path	Assert getGoals matches spec syntax
Unit	Test goals can check success correctly for the OR conditional Input Space: - A dungeon JSON with a complex goal with an OR conditional	New Dungeon-Controller with newGame called	Complete one of the subgoals	the entire complex goal is removed from the getGoal string
Unit	Test goals can check success correctly for the AND conditional Input Space: - A dungeon JSON with a complex goal with an AND conditional	New Dungeon-Controller with newGame called	Complete one of the subgoals assert goal is not removed then complete the other	the entire complex goal is removed from the getGoal string
	ADD MORE TESTING ONCE BATTLES ARE IMPLEMENTED			

[1] AD: Attack Damage

PORTALS (3 POINTS)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Create a Portal with correct position Input Space: - mock portal entity JSON data	New Dungeon-Controller	Call dungeon constructor passing in file paths	New instance of dungeon created and has correct ID and Name
Unit	Create a portal with correct position from string in controller Input Space: - Dungeon JSON containing a portal	New Dungeon-Controller	Call dungeon constructor passing in file paths	New instance of dungeon created and has matching (mock entities)
System	Test newGame with dungeon and config files and test dungeonResponse is created with mock entities Input Space: - The given JSON files	New Dungeon-Controller	Call dungeon constructor passing in file paths. Then request dungeon-Response from controller	New instance of dungeon created and returns dungeon response with matching (mock) entities
Integration	Check that player is in a cardinally adject square to the corresponding portal after they walk into a portal	New Dungeon-Controller	Call dungeon constructor passing in file paths. Then move the player into a portal and check that their position is as expected.	Player is in cardinally adjacect square to corresponding portal.
Integration	Check that player does not teleport into a wall when walking into a portal.	New Dungeon-Controller	Create map with a portal where the corresponding	Player teleports to the only available square at the

[1] AD: Attack Damage

			portal has walls in 3 of it's cardinally adjacent cells	corresponding portal
Integration	Check that player does not teleport if there are no available cardinally adjacent squares at the other portal, they do not teleport.	New Dungeon-Controller	Create map with portal pair, but corresponding portal has walls on all four adjacent squares. Walk the player into the portal.	After walking into the portal the player should not move.
Unit	Test portal chaining	New Dungeon-Controller	Creat map with 3 portals in positions where they should chain	Player ends up at the position at corresponding portal at the end of the chain

TREASURE, WOOD AND ARROWS(TWA) (1 POINT EACH)

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Unit	Create a TWA with correct position Input Space: - mock TWA entity JSON data	New Dungeon-Controller	Call dungeon constructor passing in file paths	New instance of dungeon created and has correct ID and Name
Unit	Create a TWA with correct position from string in controller Input Space: - Dungeon JSON containing a portal	New Dungeon-Controller	Call dungeon constructor passing in file paths	New instance of dungeon created and has matching (mock entities)
System	Test that when player walk on the item, it appears in their inventory and is removed from the map.	New Dungeon-Controller	Create map with TWA on the map and have the player walk into them.	TWA should appear in inventory and be removed from the map.

[1] AD: Attack Damage

SWORD (1 POINT)

- Same format as treasure, wood and arrows

BOW AND SHIELD (buildables) (1 POINT EACH)

Test Type	Test	PreCondition	Action	PostCondition
Integration	Check expected behavior for building a buildable: - Correct exceptions thrown - Items removed from inventory - Buildable is added to inventory	New Dungeon-Controller	- Build item with not enough items in inventory - Check that builables response is correct with enough items - build items with enough materials	- Invalid action exception - Response should contain name of item that can be built - Materials used are removed form inventory, built item is now in inventory
Integration	Check correct exception is thrown when trying to build something other than bow or shield	New Dungeon-Controller	Call build controller method with a string not equal to bow or shield.	IllegalArgument Exception

[1] AD: Attack Damage

Door and Key (4 POINTS)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Test creation of Door Input Space: - dungeon JSON with player and door	New Dungeon-Controller	Door should be loaded from dungeon file	Entity response should show that a door exists
Unit	Test pick up Key Input Space: - dungeon JSON with player and key	Player inventory is supported Player movement	Player walks to the position of Key	Assert the player's inventory has a key
Integration	Test opening door with key Input Space: - A dungeon JSON with one door and one key (correspoding ids)	Doors and Keys can be created	Player walks to Key position, player walks to door position	Assert the key is gone after opening the door Assert player's position shows that he walked through door
Integration	Test opening 2 doors with 2 keys (one key corresponds to one door)	The relationship of key to door is implemented	Player walks to Key 1, walks to Door1, Walks to Key 2, Walks to Door2	Assert one key is gone each time a door is opened.

[1] AD: Attack Damage

Integration	Test opening door with key where ids do not match.	The relationship of key to door is implemented	Player walks to key, player walks to door	Assert the player's inventory still has one key and his position does not change because he cannot open door with unmatching key
Integration	Test picking up key and walking over another key	More than one key cannot be picked up	Player walks to and picks up Key1, player walks to and does not pick up Key2	Assert the player's inventory only has one key

Unplaced and Placed Bomb(5 POINTS)

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Unit	Test creation of unplaced bomb Input Space: - dungeon JSON with player and bomb	New Dungeon-Controller	Dungeon is loaded	Assert there is a bomb in entity list
Unit	Test picking up bomb	Player inventory is supported Player movement	Player walks to the position of bomb	Assert the player's inventory has a bomb

[1] AD: Attack Damage

Integration	Test placing bomb near activated switch	Boulder and switch relationship works.	Player pushes boulder on switch to activate, a bomb is placed cardianlly adjacent.	Assert the the entity's within a 1 square radius are removed after detonation of bomb Assert the player is not removed
Integration	Test placing bomb near inactivated switch	Boulder and switch relationship works.	Player places a bomb cardianlly adjacent to an inactive switch.	Assert the bomb is not detonated by checking entities with tis radius and assert that the bomb has not been removed
Integration	Test bomb detonation with radius 2 Needs dungeon with entities within and outside the radius to check behaviour	Radius 2 is supported	Player places bomb cardinally adjacent to an active switch.	Assert the bomb is removed, assert entities with the radius of two are removed. Assert entities outside radius are not removed.

Zombie Toast and Zombie Toast Spawner (5 POINTS)

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Unit	Test creation of spawner Input Space: - dungeon JSON with spawner	New Dungeon-Controller	Dungeon is loaded	Assert there is a zombie spawner in entity list
Unit	Destroying spawner	Interact is implemented	Player walks to square	Assert the entity list does not

[1] AD: Attack Damage

		and sword is implemented	cardinally adjacent to spawner and interacts	have a spawner
Integration	Spawning zombie Input Space: Dungeon with spawner and config with zombie spawning set to 3	Zombie spawning is implemented	Player walks for three ticks.	Assert the entity list contains a zombie
Integration	Spawning multiple zombies Input Space: Dungeon with spawner and config with zombie spawning set to 3	Zombie spawning is implemented	Player walks for nine ticks.	Assert the entity list contains 3 zombies

[1] AD: Attack Damage

Boulder Testing Plan

Test Type	Test	PreCondition	Action	PostCondition
Unit	Create a boulder in correct position Input Space: • mock Boulder entity JSON data	Entity constructed (temporarily non abstract)	Call getEntityResponse	Data in response is correct
System	Create a boulder with correct position from string in controller Input Space: • Dungeon JSON containing a wall	DungeonControlle r exists	Call newGame with a valid map	Check in dungeonResponse for boulder and check instanceof and position
Integration	Test player can push boulder Input Space: • Player and boulder dungeon	DungeonControlle r exists with newGame called on a valid map	Move player into square with boulder	Player should be at where boulder previously was and boulder position should be shifted in direction of player movement
Integration	Test player cannot push boulder into wall Input Space: • Player and boulder dungeon with walls	DungeonControlle r exists with newGame called on a valid map	Move player into square with boulder	Player should remain adjacent to boulder and boulder should not have moved
Integration	Test player cannot push boulder another boulder Input Space:	DungeonControlle r exists with newGame called on a valid map	Move player in the direction of a square with boulder which has a boulder adjacent to it	Player should remain adjacent to boulder and both boulders should not have moved

[1] AD: Attack Damage

	• Player and boulder dungeon			
Integration	Test player cannot push boulder into door (open/closed) Input Space: • Player and boulder with door	DungeonControlle r exists with newGame called on a valid map	Player attempts to push boulder into adjacent square with open/closed door	Player should remain adjacent to boulder and boulder should not have moved

Switch Testing Plan

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Unit	Create a switch in correct position Input Space: • mock switch entity JSON data	Entity constructed (temporarily non abstract)	Call getEntityRespons e	Data in response is correct
System	Create a switch with correct position from string in controller Input Space: • Dungeon JSON containing a switch	DungeonControl ler exists	Call newGame with a valid map	Check in dungeonResponse for switch and check instanceof and position
Integration	Test player cannot activate switch Input Space: • Player and switch dungeon	DungeonControl ler exists with newGame called on a valid map	Move player into square with switch	Player should be at where switch is but switch remains inactive

[1] AD: Attack Damage

Integration	Toggle switch Input Space: • Player and boulder dungeon with switch	DungeonController exists with newGame called on a valid map	Move boulder into square with switch and push off again	Switch should be inactive, active then inactive
Integration	Test multiple switch activation • Player and dungeon with multiple switches and boulders	DungeonController exists with newGame called on a valid map	Move boulders onto switches and then push one of them off	Initially all off, then they all should be active after boulders pushed, then all but one is active.

Mercenary Testing Plan

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Unit	Create a mercenary in correct position Input Space: • mock mercenary entity JSON data	Entity constructed (temporarily non abstract)	Call getEntityResponse	Data in response is correct
System	Create a mercenary with correct position from string in controller Input Space: • Dungeon JSON containing a mercenary	DungeonController exists	Call newGame with a valid map	Check in dungeonResponse for mercenary and check instanceof and position
Unit	Check instantiated mercenary is hostile Input Space:	DungeonController exists with newGame called on a valid map	Create a mercenary from dungeon json and check state	Should be hostile state

[1] AD: Attack Damage

	• Dungeon with mercenary			
Integration	Test basic bribe behaviour Input Space: • Valid dungeon file with mercenary, player and treasure	DungeonController exists with newGame called on a valid map	Move player to pickup treasure and try to bribe mercenary	Mercenary should enter bribe state
Integration	No treasure exception Input Space: • Valid dungeon file with mercenary, player	DungeonController exists with newGame called on a valid map	Move player within range of mercenary (1 tile minimum so use that) and try to bribe	Throw invalid action exception: Should not be able to bribe with no treasure
Integration	Not in range exception Input Space: • Valid dungeon file with mercenary, player, treasure and player 4 blocks away	DungeonController exists with newGame called on a valid map	Have player pick up treasure and attempt to bribe mercenary outside of range	Throw invalid action exception: Should not be able to bribe out of range
Integration	Cannot bribe an already bribed mercenary Input Space: • Valid dungeon file with mercenary, player, 2	DungeonController exists with newGame called on a valid map	Have player pick up the treasure and bribe the mercenary, then attempt to interact again	Isinteractable() for mercenary entity class should be false

[1] AD: Attack Damage

	treasures and player			
Integration	Basic mercenary movement (hostile) Input Space: • Valid dungeon file with mercenary, player a few squares away	DungeonController exists with newGame called on a valid map	Have player run in one direction	Check that the distance is either lessening or equal every tick
Integration	Basic mercenary movement (bribed) Input Space: • Valid dungeon file with mercenary, treasure and player	DungeonController exists with newGame called on a valid map	Have player pickup treasure and bribe mercenary. Have the player do zig zags or run in circles	Mercenary should occupy every square the player was in
Integration	No path case Input Space: • Valid dungeon file with mercenary, player but mercenary surrounded by walls	DungeonController exists with newGame called on a valid map	Have player run to increment tick	Mercenary should not move as there is no path
Integration	Blocked by boulder Input Space: • Valid dungeon file with mercenary, player but mercenary at end of	DungeonController exists with newGame called on a valid map	Have player back and forth to increment tick, then push boulder to block entrance to corridor	Mercenary should move initially but then stop

[1] AD: Attack Damage

	cortridor of walls with boulder at end but not blocking			
Integration	Blocked by boulder Input Space: • Valid dungeon file with mercenary, player but mercenary at end of cortridor of walls door/key at end	DungeonController exists with newGame called on a valid map	Have player back and forth to increment tick, then use key to open door before running back	Mercenary should not move initially but then start moving
Integration	Invisible player Input Space: • Valid dungeon file with mercenary, player and invisible potion	DungeonController exists with newGame called on a valid map	Have player pick up potion and run a bit then drink potion	Mercenary should move initially by path finding but then go random movement (
Integration	Invisible player Input Space: • Valid dungeon file with mercenary, player and invincible potion	DungeonController exists with newGame called on a valid map	Have player pick up potion and run to the left a bit then drink potion and continue running	Mercenary should move left initially but then move to the right after player drinks potion
Integration	Pathfinding through two swamps Input Space:	DungeonController exists with newGame called on a valid map	Player moves left to trigger ticks.	Mercenary should take the path through swamp with less movement penalty

[1] AD: Attack Damage

	• Valid dungeon with mercenary, player and only two available paths, both through swamps • One path has less movement penalty than the other			
Integration	Pathfinding to avoid swamp tiles Input Space: • Valid dungeon with mercenary, player and swamp tiles between them, but there is a clear path.	DungeonController exists with newGame called on a valid map	Player moves left to trigger ticks	Mercenary should follow the clear path through the swamp, as this is faster than going through the swamp tiles.

[1] AD: Attack Damage

Integration	Pathfinding to go around a block of swamp tiles, even if cardinally longer route Input Space: • Valid dungeon with mercenary, player and swamp tiles between them, but there is a clear path.	DungeonController exists with newGame called on a valid map	Player moves left to trigger ticks	Mercenary should go around the swamp tiles, even if it is a cardinally longer route
Integration	Dijkstra pathfinding through swamp Input Space: • Valid dungeon with mercenary, player and only swamp tiles between them. But there is a path with lower movement cost	DungeonController exists with newGame called on a valid map	Player moves left to trigger ticks	Mercenary should go through swamp block, but only moving through swamps that minimise its overall cost.

[1] AD: Attack Damage

SWAMPS

Integration	Test player is not slowed be swamp Input Space: • Dungeon with player and swamp tile	DungeonController exists with newGame called on a valid map	Move player onto swamp tile and off again	Player should not have taken more than 2 ticks to do this. I.e. not stuck.
Integration	Test player is not slowed be swamp Input Space: • Dungeon with player, mercenary and swamp tile. Only path to player is through tile	DungeonController exists with newGame called on a valid map	Move player away from the mercenary to force the mercenary to walk onto the swamp	Mercenary should be stuck at the swamp for the required number of ticks, before moving off after.
Integration	Test allied mercenary gets stuck on swamp tile (not necessary tbh) Input Space: • Dungeon with player, treasure, mercenary and swamp tile	DungeonController exists with newGame called on a valid map	Bribe mercenary, then move player onto swamp tile and off again. Move until the mercenary walks onto the tile.	Ally should be stuck on tile. This is actually undefined but we will say they get stuck.
Integration	Test zombie gets stuck on swamp tile Input Space: • Dungeon with player, zombie surrounded by swamp tile	DungeonController exists with newGame called on a valid map	Move the player around, eventually the zombie will accidentally step on a swamp.	Zombie should be stuck for the required amount of ticks, before moving off.

[1] AD: Attack Damage

Integration	Test spider gets stuck on swamp tile Input Space: • Dungeon with player, spider surrounded by swamp tile	DungeonController exists with newGame called on a valid map	Move the player around, eventually the spider will accidentally step on a swamp.	Spider should be stuck for the required amount of ticks, before moving off.
Integration	Test swamps have individual movement factors Input Space: • Dungeon with player, mercenary, consecutive swamps with different movement factors. Mercenary can only go through them.	DungeonController exists with newGame called on a valid map	Move the player so the mercenary walks onto the swamp tiles. The mercenary should walk on all of them to try and reach the player.	For all the swamps, Mercenary should be stuck for the required different number of ticks, before moving off after.

[1] AD: Attack Damage

PERSISTENCE (5 POINT)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Player Data to JSON Input Space: - A mock player to emulate a player midway through a dungeon	Mock player	Call SaveUtility.PlayerToJson()	JSON is produced with relevant data
Unit	Entity to JSON Input Space: - A collection of mock entities	A collection of mock entities	Call SaveUtility.EntityToJSON()	JSON are produced with correct relevant data for each entity
Unit	Weapon to JSON Input Space: - A mock weapon	Mock weapon	Call SaveUtility.WeaponToJSON()	JSONproduced with correct relevant data for weapon
Unit	Inventory to JSON Input Space: - A player with items in inventory	Mock player with mock items loaded into its inventory	Call SaveUtility.InventoryToJSON()	JSONproduced with correct relevant data for players inventory
Integration	Create JSON from dungeon in play Input Space: - Mock playable dungeon with some progress made from start state	Call dungeon controller, start a new game and make some moves	Call SaveUtility.saveGameToJSON()	JSON produced should contain all data relevant to start a new game correctly
Integration	Load a dungeon from a save Input Space:	Create a saveGame	Construct dungeon from saveGame	dungeonResponse generated by new dungeon should match

[1] AD: Attack Damage

	- a save game generated from SaveUtility			response made by save
System	Call Save and Load using controller methods Input Space: - a save game generated from controller	Start a new game and make some progress, then call saveGame	create a new controller and load the save	dungeonResponses should match
Unit	Test lastposition behaviour when save is called on a brand new game Input Space: - dungeonController with new game	Start a new game from controller without making any moves	call playerToJSON	check lastposition defaults to startposition
Unit	Test sword entity is saved correctly Input Space: - map with a sword	Start a new game from controller with given map	Call saveGame and check for sword JSONObject details. Then pick up sword and call saveGame again to check inventory JSONObject	JSONObject containing sword date should have correct fields in both cases
Integration	Test all other untested entities Input Space: - map containing all entities untested	Start a newgame using the given map	Call saveGame and load game from new controller	All entities should be in new game
Integration	In depth Load Player Data using player load methods	Create the mock JSON and create a new player	Call loadPlayer and loadInventory	Player should have all fields updated

[1] AD: Attack Damage

	Input Space: - Mock JSON of playerData including built items and potions in the queue			
System	Test controller method AllGames is correct Input Space: - Save game directory with test file	SaveGame directory should have a test file	Call dmc.allGames	Response should contain test file
System	Test correct error is thrown for loadGame with invalid name Input Space: - String name that does not exist in gameSave directory	SaveGame directory should not have a file with the given name	Call dmc.loadGame with the given name	throws IllegalArgument Exception
Integration	Test milestone 3 items behave correctly for saves Input Space: - map with new items from milestone 3 and materials to construct buildables	Create new game and collect materials and build items	Save and load game from new controller	DungeonRespon se should be the same

[1] AD: Attack Damage

MIDNIGHT ARMOUR

Test Type	Test	PreCondition	Action	PostCondition
Integration	Successful crafting with no zombies Input Space: • JSON with midnight armour amterials	DungeonController exists with newGame called on a valid map	Collect materials and build armour	Midnight armour added to inventory
Integration	Test build with zombies on map Input space: - JSON wth materials and zombie	DungeonController exists with newGame called on a valid map	Collect materials and try to build armour	InvalidActionExcepti on
Integration	Test battle calculations with midnight armour equipped Input space: - JSON wth armour materials	DungeonController exists with newGame called on a valid map	Collect materials, build armour and battle entity	The player should have additional defence and attack in battle

SCEPTRE

Test Type	Test	PreCondition	Action	PostCondition
Integration	Crafting with all possible recipes 1 sceptre = (1 wood OR 2 arrows) + (1 key OR 1 treasure) + (1 sunstone) 1 sceptre + 1 (retained) sunstone = (1 wood OR 2 arrows)	DungeonController exists with newGame called on a valid map	Collect materials and build	Sceptre added to inventory and appropriate items removed

[1] AD: Attack Damage

	+ (1 sunstone) + (1 sunstone) Input Space: • JSON with materials			
Integration	Mind control Input space: - JSON wth materials and mercenary and assassin	DungeonController exists with newGame called on a valid map	Collect materials, build sceptre and mind control assassin and mercenary	Assassin/mercenary should be mind controlled for the given duration

SUNSTONE

Test Type	Test	PreCondition	Action	PostCondition
Integration	Collecting sunstone and opening door Input Space: • JSON with sunstone and locked door	DungeonController exists with newGame called on a valid map	Collect sunstone and open door	Door should now be opened
Integration	Collecting sunstone and building shield Input Space: • JSON with sunstone and 2 wood	DungeonController exists with newGame called on a valid map	Collect materials, build shield	Shield should be added to inventory
Integration	Collecting sunstone, should count towards treasure goal Input space - JSON with sunstone and treasure with treasure goal	DungeonController exists with newGame called on a valid map	Collect sunstone and treasure	Sunstone collected should count towards treasure goal

[1] AD: Attack Damage

Time Travel

Test Type	Test	PreCondition	Action	PostCondition
Unit	Collecting time turner Input Space: • JSON with time turner	DungeonController exists with newGame called on a valid map	Collect time turner	Time turner added to inventory
System	Rewind exceptions	DungeonController exists with newGame called on a valid map	Rewind with negative ticks and unreached ticks	IllegalArgumentException
Integration	Rewind 1 and 5 ticks Input space - JSON with time turner	DungeonController exists with newGame called on a valid map	Collect time turner and rewind 1 and 5 ticks	The time turner should reappear on map but still be in player inventory, old player should appear on map and current player should not move
Integration	Time travelling through portal with less than 30 ticks	DungeonController exists with newGame called on a valid map	Let a less than 30 ticks pass and go through time travelling portal	- game state is same as initial game state - current player is on same square as portal
Integration	Time travelling through portal with 30 ticks	DungeonController exists with newGame called on a valid map	Let a more than 30 ticks pass and go through time travelling portal	- game is not initial state - game is correct state - current player on same square as portal

[1] AD: Attack Damage

Random Dungeon Generation Test

Test Type	Test	PreCondition	Action	PostCondition
Integration	Test player and exit exist in randomly generated dungeon Input Space: • Coordinates with simple config	Create a new dungeon controller and then generate multiple dungeons under the same coordinates	None	Generated mazes should have a player and exit
System	Test there is a boundary wall around the maze Input Space: • Coordinates with simple config	Create a new dungeon controller and then generate multiple dungeons of varying size	None	Generated mazes should have a boundary wall around them

Hydra (3 POINTS)

Test Type	Test	PreCondition	Action	PostCondition
Unit	Test creation of Hydra Input Space: - dungeon JSON with Hydra	New Dungeon-Controller	Dungeon is loaded	Assert there is a Hydra in entity list
Integration	Battle Hydra, increasing health Input Space: - Config with hydra HP increase set to 1	Milestone 2 works	Player walks to hydra, hydra is boxed in so it can only move to fight player.	Assert the round response shows hydra health increase.

[1] AD: Attack Damage

Integration	Battle Hydra, decreasing health Input Space: - Config with hydra HP increase set to 0	Milestone 2 works	Player walks to hydra, hydra is boxed in so it can only move to fight player.	Assert the round response shows hydra health decrease.
Integration	Battle Hydra, random increase/decrease health Input Space: - Config with hydra HP increase set to 0.5 - Low attack damage	Milestone 2 works	Player walks to hydra, hydra is boxed in so it can only move to fight player.	Run a long battle Assert the round response shows a hydra health decrease. Assert the round response shows a hydra health increase.

Assassin (3 POINTS)

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Integration	Bribe Assassin, bribe fail 1 Input Space: - Config with bribe fail 1	Milestone 2 works	Player walks to assassin, he is boxed in so it can only move to fight player.	Assert gold is wasted. Assert a battle occurs.
Integration	Bribe Assassin, bribe fail 0 Input Space: - Config with bribe fail 0	Milestone 2 works	Player walks to assassin, he is boxed in so it can only move to player.	Assert gold is used. Assert a battle does not occur.

[1] AD: Attack Damage

Integration	Bribe Assassin, bribe fail 0, Fight another enemy Input Space: - Config with bribe fail 0	Milestone 2 works	Player walks to assassin, he is boxed in so it can only move to player. Walk to another enemy.	Assert gold is used. Assert a battle does not occur with assassin. Assert ally calculations in battle with another enemy.

Logic (5 POINTS) [Did not plan for all sections]

Test Type	**Test**	**PreCondition**	**Action**	**PostCondition**
Integration	Test bulb with switch Input Space: - dungeon JSON with boulder, switch, bulb	Milestone 2 works	Push boulder on switch	Assert the light bulb type is changed to light_bulb_on
Integration	Test bulb with switch step off Input Space: - dungeon JSON with boulder, switch, bulb	Milestone 2 works	Push boulder on switch, push boulder off switch	Assert the light bulb type is changed to light_bulb_on Assert the light bulb type is changed to light_bulb_off when the boulder is pushed off
Integration	Test bulb with wire and switch step off Input Space: - dungeon JSON with boulder, switch, bulb, wire between bulb and switch	Milestone 2 works	Push boulder on switch, push boulder off switch	Assert the light bulb type is changed to light_bulb_on Assert the light bulb type is changed to light_bulb_off when the boulder is pushed off

[1] AD: Attack Damage

Integration	Test switch door with switch Input Space: - dungeon JSON with boulder, switch, switch door	Milestone 2 works	Push boulder on switch	Assert the player can walk through door without key
Integration	Test switch door with wire and switch step off Input Space: - dungeon JSON with boulder, switch, switch door, wire between door and switch	Milestone 2 works	Push boulder on switch, push boulder off switch	Assert the player can walk through door without key After boulder is pushed off assert player cannot walk through door
Integration	Test switch door with key Input Space: - dungeon JSON with boulder, key, switch door	Milestone 2 works	Walk to switch door	Assert the player can walk through door with key

[1] AD: Attack Damage