Test Plan 2

Testing phase 2 - testing the new version

Traceability matrix:

The table below shows how our tests relate to the requirements of the assessment. This table should be maintained throughout testing. Each test is identified based on what testing phase it is, which overall test it is (e.g. TimerSystemTests, and which individual test it is (e.g. 1.1.3 is the third test in the TimerSystemClass for testing phase 1). Test/requirement overlaps are valid - this helps improve our overall coverage during testing.

Requirement identifiers	FR_TIMER	FR_EVENTS	FR_MAP	FR_MOVEMENT	FR_SCORE_CALC	FR_RESET	FR_RESUME	FR_INVARIANTS	FR_SOUND
2.1.1-9	X								
2.2.1-8				X					
2.3.1-5			X						
2.4.1-2					X				
2.5.x					X				
2.5.y					X				
2.6.1-2							X		
2.7.1-6								X	
2.8.1-4		X							
2.9.1-3		X							
2.10.1									

2.1 Requirement: FR_TIMER - The game shall implement a timer to track the players play and escape time throughout the game.

2.1.1 **testConstructor()**
Results: pass
2.1.2 **testAdd()**
Results: pass
2.1.3 **testAddGradually()**
Results: pass
2.1.4 **testGetTimeLeft()**
Results: pass
2.1.5 **testGetTimeLeftAtZero()**
Results: pass
2.1.6 **testGetTimeLeftNegative()**
Results: pass
2.1.7 **testGetClockDisplay()**
Results: pass
2.1.8 **testGetClockDisplayWithLeadingZero()**
Results: pass
2.1.9 **testMultipleAddCalls()**
Results: pass

2.2 Requirement: FR_MOVEMENT - The game shall allow the player avatar to navigate the map using standard directional keyboard inputs.

2.2.1 **testUpMovement()**
Results: pass
2.2.2 **testDownMovement()**
Results: pass
2.2.3 **testLeftMovement()**
Results: pass
2.2.4 **testRightMovement()**
Results: pass
2.2.5 **testFailureMovementCases()**
Results: pass
2.2.6 **testMovementReturnsCorrectDistance()**
Results: pass
2.2.7 **testPlayerInitialisation()**
Results: pass
2.2.8 **testPlayerSpeed()**
Results: pass

2.3 Requirement: FR_MAP- The map will be a hardcoded 2D maze and visible at all times with clear boundaries.

2.3.1 **testInit()**
Results: pass

2.3.2 **testInitNamedObjects()**
Results: pass
2.3.3 **testSafeToMove()**
Results: pass
2.3.4 **testNotSafeToMove()**
Results: pass
2.3.5 **testRemoveCollision()**
Results: pass

2.4 Requirement: FR_SCORE_CALC- The score constantly updates and is shown on screen so the player can always see how they are progressing.

testCalculateScore_noBonus()
Result: Pass

testCalculateScore_withLongboiBonus()
Result: Pass

2.5 Requirement: FR_RESET- The game shall rest/restart appropriately if player either wins/falls/quits the game

Property being tested: Functional suitability

2.5.1 RESET_2 -
Steps to be followed:

- Start the game
- Let the timer run out so the user fails the game (by running into the dean multiple times)
- Restart the game

Expected output:
The game should seamlessly restart to its original state after failing the game

Actual output:
The game can be restarted or the user can exit and restart the game

Status: Pass

2.6 Requirement: FR_RESUME- The game shall resume seamlessly from pause with no background AI or physics persisting behind pause.

2.6.1
Test ID: RESUME_1 - Pause stops game correctly
Steps to be followed:

- Start the game, move the player a bit so you see the timer counting and dean moving.
- Press P for pausing
- While paused, wait in real time for a while and observe any change

Category: Pause test
Expected output /actual output:
Status: Pass

2.6.2
Test ID: RESUME_2 - Resume continues game correctly
Steps to be followed:

- Start the game, let the timer reach a value and then pause the game
- Resume the game and observe

Category: Resume test
Expected output /actual output:
Status: Pass

2.6.3
Test ID: RESUME_3 - Repeated pausing does not break the game
Steps to be followed:

- Pause -> resume -> pause -> resume
- Play the game for a while after resuming and observe

Category: Robustness test
Expected output/actual output:
Status: Pass

2.7 Requirement: FR_INVARIANTS- The game should always have means to win(escape), the timer should always track player time, player avatar should always respond to player input.

2.7.1 Game always loads in with the correct items to overcome the obstacles:

E.g. the exit door and key,
the spikes and the lever,
The book and the sliding bookshelves.

Items do not disappear when the player is put into detention.
Items can be retrieved in any order preferred.

2.7.2 **testGameStateAlwaysValid()**
Test for if the game states, 0 to 4, are always within that range.
Not started, playing, paused, won, lost.
They can only be within one of these states.

2.7.3 **testScoreFormulaAlwaysHolds()**
Check that for when the player gets a bonus the score abides by the given formula

- When the player finishes at a range of time, do they get the correct points?

2.7.4 **testFlagsAreIndepentent()**
Check that the flags for the different items are separate from one another thus not affected by each other unless by means of the player.

This includes the exits, the chests, the spikes being lowered.

2.7.5 **testItemCollectionPersists()**
Tests that once the items are collected, the flag in the game code remains true and isn't changed as they are not able to lose the item when playing the game.

2.7.6 **testScoreCalcDeterministic()**
The score will be calculated the same no matter what state the game is in currently.
And when in the same state and no other variables have changed, the game should give the player the same score every time.

2.8 Requirement: FR_EVENTS - The game will detect and trigger events based on players interaction/collisions accordingly

2.8.1 **BobEventTest.java**
Tests for whether Bob can be found if the player has not got the piInteracted flag set to true
It checks for if Bob is found then the bobBonus is applied to the score.

Interaction has set piInteracted flag to true -> bonus points can be rewarded.

2.8.2 **BookshelfTest.java**
a.) Tests for if the bookshelf can be opened by placing the book when the player has not yet obtained it. This checks for if the associated flags are set correctly
b.) checks if the player can pick up the book if they have it already. Checks if they can place it once they have the book.replicating what the order of the aims in the game would be.

Book obtained -> bookshelf interaction -> bookshelf should move

2.8.3 **DoorTest.java**
a.) checks for if the door to the chest room can be opened in the event the player has not got the key for it. This will try it for both when the player has and hasn't got the key.

Player without key -> door cannot be opened
Player has acquired key -> door opened for the player

b.) it will also test for the key being acquired and then trying to open the exit door of the building. Creating a dummy player and then setting the suitable flags to test.

Same as prior.

2.8.4 **LeverTest.java**
Tests that when the lever is activated, the spikes are lowered so they can enter the room. Sets bool to false then calls the function the lever interaction would.

Lever interacted with -> spikes are lowered -> player can access area previously blocked

2.9 Requirement: FR_DEAN

2.9.1 **testReachPlayerVisible()** and **testReachPlayerInvisible()**
Can the dean spot/reach the player when they are either visible or invisible?
Initialise new dummy dean and player objects and set their hitboxes to overlap

We can assert that when they do overlap if they are NOT invisible, they will be caught.

In the event they have the invisibility potion and thus the flag for invisibility is set, they will be able to sneak past because the dean.canreach() is false.

2.9.2 **testRestartPath()**
Tests that the path of the dean can be properly restarted at the end of the cycle.

Checks for the current tile in the path being the first in the list thus the dean is at the start of the loop again

It also checks that the next tile is a tile's width away as the dean will move in increments less than a tile.

2.9.3 **testGetReachRectangle()**
tests that the reach rectangle can be returned properly. This allows us to debug the dean, but is also called in different parts of the game relating to collisions.

2.10.1
Test ID: INPUTS - Pressing all input keys, excluding movement keys, E for event interaction and P key for pause (these have been tested with automated tests), and testing that they function correctly.
Steps to be followed:

- Start the game
- Press the F11 key
- Press the space key
- Press F2
- Press E on an interactable event
- Press the esc key

Category: InputSystem test
Expected output:

- The game should seamlessly transition from a small window to a full screen window
- The game should start when the space key is pressed

- The game should show collisions when F2 is pressed
- The game should close when the esc key is pressed

Actual output:

- The game screen turned full screen
- The game started
- The game showed collisions
- The game closed

Status: Pass