Test Plan 1

Testing phase 1 - testing the original 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
1.1.1-9	X							
1.2.1-8				X				
1.3.1-5			X					
1.4.1-2					X			
1.5.1						X		
1.6.1-3							X	
1.7.1								X
1.8		X						
1.9.1								

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

Code Tested: Timer

1.1.1 **testConstructor()**

Description: Verifies that a new TimerSystem is initialised with 0s elapsed time.

Results: pass

1.1.2 **testAdd()**

Description: Tests the add method by adding a fixed amount of time, and checking for it being updated correctly.

Results: pass

1.1.3 **testAddGradually()**

Description: Tests the add method by checking it adds time over multiple calls.

Results: pass

1.1.4 **testGetTimeLeft()**

Description: Tests the getTimeLeft method by checking for the correct time left in 3 different examples when various times have elapsed.

Results: pass

1.1.5 **testGetTimeLeftAtZero()**

Description: Tests the getTimeLeft method for time when at 0, and when at the maximum 300s.

Results: pass

1.1.6 **testGetTimeLeftNegative()**

Description: Tests the getTimeLeft method by setting time to over the maximum, and negates accordingly to a valid time.

Results: pass

1.1.7 **testGetClockDisplay()**

Description: Tests the clock (timer) display and checks for correct output.

Results: pass

1.1.8 **testGetClockDisplayWithLeadingZero()**

Description: Tests the format of the timer by checking for the correct time when there is a leading zero in seconds.

Results: pass

1.1.9 **testMultipleAddCalls()**

Description: Tests when multiple calls are made to the timer, that the correct total time is added.

Results: pass

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

Code tested: InputSystem (for the player)

1.2.1 **testUpMovement()**

Description: Tests that the UP movement functionality works correctly, given the player isn’t frozen.

Results: pass

1.2.2 **testDownMovement()**

Description: Tests that the DOWN movement functionality works correctly, given the player isn't frozen.

Results: pass

1.2.3 **testLeftMovement()**

Description: Tests that the LEFT movement functionality works correctly, given the player isn't frozen.

Results: pass

1.2.4 **testRightMovement()**

Description: Tests that the RIGHT movement functionality works correctly, given the player isn't frozen.

Results: pass

1.2.5 **testFailureMovementCases()**

Description: Tests to check that no movement will occur when no valid input is given.

Results: pass

1.2.6 **testMovementReturnsCorrectDistance()**

Description: Tests that valid movement will return the correct distance moved by the player.

Results: pass

1.2.7 **testPlayerInitialisation()**

Description: Tests that the player is initialised to the correct position for future movement.

Results: pass

1.2.8 **testPlayerSpeed()**

Description: Tests the players default speed and this speed when updated.

Results: pass

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

Code tested: CollisionSystem

Testing the boundaries of the map:

1.3.1 **testInit()**

Description: Tests the init() will correctly create collision rectangles from map objects.

Results: pass

1.3.2 **testInitNamedObjects()**

Description: Tests that objects with and without names are initialized correctly

Results: pass

1.3.3 **testSafeToMove()**

Description: Test that safeToMove() returns true when there is no collision overlap, and verifies that distant collision objects do not block movement.

Results: pass

1.3.4 **testNotSafeToMove()**

Description: Tests that safeToMove() returns false when there is a collision overlap

Results: pass

1.3.5 **testRemoveCollision()**

Description: Tests removeCollisionByName() by creating a named rectangle, and attempting to remove a rectangle of a different name. The collision list should still contain just the original rectangle.

Result: pass

1.4 Requirement: FR_SCORE_CALC -

Code being tested: Main
Tests:
1.4.1 **testCalculateScore_noBonus()**
Result: pass
Description: Set time to be a small amount remaining, 5”. And ignores longboi bonus
Asserts that the score is dependent solely on time.

1.4.2 **testCalculateScore_withLongboiBonus()**
Result: pass
Description: Accounts for longboi bonus in score field
Sets bonus to be non-zero
Sets time to be small amount remaining
Assert score should be dependent on both longboi and time left.

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

1.5.1 RESET -
Steps to be followed:

- Start the game
- Let the timer run out so the user fails the game
- Restart the game

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

Actual output:
The game cannot be restarted after the user fails
(Additionally the user can choose to replay if they win).

Status: Pass

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

Property being tested: Functional suitability
(Note: Pressing P pauses and resumes gameplay)

1.6.1
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:
-When paused timer does not change and no background actions happen
-Player and dean does not move
Status: Pass

1.6.2
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:
-Timer shows the same value as before pausing and starts counting down when resumed
-Dean and player resumes movement smoothly
Status: Pass

1.6.3
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:
-No freezing and glitching when repeatedly paused and resumed
-Timer, player and dean all continue normally
Status: Pass

1.7 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.

1.7.1 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.

1.7.1 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.

1.7.2 **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.

1.7.3 **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?

1.7.4 **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.

1.7.5 **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.

1.7.6 **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.

1.7.7 **testScoreREmainsReasonable()**
The score will not ever be outside of the bounds of 0, when the player is at the end with no bonus, or the largest with the full timer bonus.

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

Events tested via unit tests as well as manual tests.

1.8.1 **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.

1.8.2 **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

1.8.3 **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.

1.8.4 **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

1.9 Requirement: General functionality when trigger buttons are pressed
1.9.1
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