Testing

Cohort 2 Team 1

Ahmet Abdulhamit
Zoey Ahmed
Tomisin Bankole
Alanah Bell
Sasha Heer
Oscar Meadowcroft
Alric Thilak

Testing Approaches Used:

We combined automated unit testing, manual gameplay testing and system integration testing to ensure comprehensive coverage of the game “Grep the Exit”.

1 - Unit Testing with JUnit 5:

We implemented JUnit 5 as our primary unit testing framework, focussing on core game logic (inventory, collision, scoring), data models (Exam, Locker, NPC) and utility functions (game state validations and mathematical calculations). JUnit 5 was appropriate as LibGDX’s separation of rendering/graphics concerns from game logic allowed for isolated testing of business components without graphical contexts.

2 - Automated Testing Suite:

We developed a comprehensive automated test suite covering:

Data Model Testing:

- Verification of game object states and behaviours
- Locker item distribution and opening logic
- Inventory slot management and item usage
- Exam question/answer storage and completion tracking
- NPC dialogue systems and interaction logic

Algorithm Testing:

- Mathematical verification of game mechanics
- Collision detection formulas and future position prediction
- Tiled map coordinate scaling (pixel to world units)
- Distance calculations for interaction ranges
- Movement pattern algorithms for entities

State Machine Testing:

- Validation of object lifecycles
- Locker state transition (closed -> opened)
- Exam completion persistence
- Guard movement pattern consistency
- Player respawn and checkpoint systems

This approach was appropriate as it validated core logic independently whilst avoiding complex mocking of OpenGL/rendering.

3 - Manual Testing Protocol

We conducted systematic manual testing for UI/UX verification, screen transitions, graphics rendering and real-time gameplay feel. These tests were important as certain game aspects such as visual feedback, button hover effects and smooth animations require human judgement that automated tests can’t provide. Manual testing validates the player experience, ensuring the game meets its accessibility requirements and provides an engaging, family-friendly experience as specified in UR_UX.

4 - Integration Testing

We performed integration testing focussing on screen navigation flows and map-transition systems. This testing approach was crucial as it ensured connected components worked together in a game with multiple interactive screens.

Test Results and Statistics:

Automated Testing:

Module	Total Tests	Passes	Failed	Coverage
InventorySystem	5	5	0	Total: 70% Inventory class: 83%, InventoryItem class: 100%
CollisionChecker	5	5	0	Total: 75% Maths logic: 100%, Core collision algorithms: 95%, Exception handling: 100%
Exam object	5	5	0	Total: 65% Business logic and rules: 95%, Answer validation algorithm: 100%, Gameplay mechanics: 90%
Locker object	5	5	0	Total: 75% Special case: 100%, Business rules: 100%, State management: 90%, Probability and distribution logic: 95%
NPC	3	3	0	Total: 45% Interaction rules logic: 85%, Distance calculation: 100%
Guard	4	4	0	Total: 65% Guard logic: 90%, Movement maths: 100%, Sprite selection: 100%
Player	5	5	0	Total: 75% Player logic: 95%, Spawn/respawn system: 100%, Sprite direction logic: 100%, Movement maths: 100%

Manual Testing:

Test ID	Test Type	Expected Result	Result
T_WASD	Input handling	WASD movement should correspond to the right direction	Pass
T_E_KEY	Input handling	E key is the interaction key for NPCs/exams/lockers	Pass
T_NO_LAG	Input handling	T key teleports the player through portals to different maps	Pass
T_T_KEY	Input handling	Verify there is no input lag or stuck keys	Pass
T_EDGE_CASES	Input handling	Edge cases don’t cause errors or lag	Pass
T_MAIN_FLOW	Gameplay flow	Gameplay flow follows common conventions	Pass
T_DEATH_RESPAWN	Gameplay flow	Player being caught triggers a respawn	Pass
T_LOCKER_FLOW	Gameplay flow	Locker interaction flow follows common conventions	Pass
T_EXAM_FLOW	Gameplay flow	Exam interaction flow follows common conventions	Pass

T_PICKUP_ITEM	Inventory system	Picking up an item from a locker adds the item to the player's inventory	Pass
T_USE_ITEM	Inventory system	Pressing 1,2 or 3 to use the corresponding inventory slot's item uses the item and applies the correct effect to the player	Pass
T_S_CARD_TO_ESCAPE	Inventory system	Once the user has collected their student card they are able to go back through the portal	Fail
T_TRY_USE_EMPTY	Inventory system	Trying to use an empty inventory slot doesn't do anything	Pass
T__FILL_INVENTORY	Inventory system	Inventory can be filled and items get added in the first available slot	Pass
T_LEADERBOARD	Leaderboard	The leaderboard functions as expected	Pass
T_PLAYER_HITS_WALLS	Collision and physics	A player can not collide with any walls when walking in any given direction	Pass
T_PLAYER_COLLIDE_GUARD	Collision and physics	When a player collides with a guard they respawn at the last spawnpoint	Pass
T_GUARD_PATHS	Collision and physics	Guards can travel horizontally and vertically	Pass
T_GUARD_HITS_WALL	Collision and physics	Guards cannot collide with walls and when they meet a wall they turn to go the other way	Pass
T_CORRECT_EXAM	Collision and physics	When a player completes an exam question correctly they receive brief guard immunity	Pass
T_INCORRECT_EXAM	Collision and physics	When a player completes an exam question incorrectly the guards speed up briefly	Pass
T_NO_GUARD_OVERLAP	Collision and physics	Guards in a close area do not overlap and stay to their given paths	Pass
T_LOCKER8_ITEM	Collision and physics	Locker 8 always contains a student card	Pass
T_OPEN_10_LOCKERS	Probability	Item probability of lockers spawning with different items functions as expected	Pass
T_RANDOM_LOCKER_ITEM	Randomness	For every game the lockers spawn with random items each time	Pass
T_EXAM_SYSTEM	State machine	The exam interaction functions fully as expected	Pass
T_NPC_DIALOGUE	State machine	The NPC interaction functions fully as expected	Pass
T_WALK_OFF_MAP	Boundary and edge case	The player cannot walk off either of the two maps	Pass
T_VALID_RESPAWN_POINTS	Boundary and	All respawn points are in a valid map location that is	Pass

	edge case	on the map and accessible	
T_MAX_INVENTORY	Boundary and edge case	No additional items can be added to a full inventory	Pass
T_ALL_GUARDS_ACTIVE	Performance	Multiple guards can be active at the same time	Pass
T_ALL_OBJECTS_VISIBLE	Performance	Picking up items has a visual display in the player's inventory	Pass
T_EVENT_COUNTER_UPDATE	UI/UX	When interacting with events the correct corresponding event tracker is updated	Pass
T_SCORES_DISPLAYED	UI/UX	The win screen correctly displays the players score with any bonus points from achievements	Pass
T_BUTTONS_ALL_WORK	UI/UX	All menu buttons work	Fail
T_NO_MAP_LOADING_LAG	UI/UX	Changing map renders correctly and has no lag	Pass
T_SCREEN_LOADING	UI/UX	All screens have clear visuals	Pass

Automated Tests Report: Tests | Grep The Exit under "Test Results" or directly at Test results - Test Summary under "io.maze"
Manual Test Descriptions: Tests | Grep The Exit

Failed Tests Analysis:

Manual Test Failures:
T_S_CARD_TO_ESCAPE

- *Issue 1*: Once the player has obtained the student card and the item has been added to the player's inventory, if the player uses that inventory item before exiting the hidden room the player is then unable to exit the room as they no longer have their student card - leaving the player trapped indefinitely
- Cause: the Inventory.useItem() destroys the student card once it is used but hasStudentCard() checks if the student card is currently in the inventory
- Fix: modify the student card in Inventory.useItem() to be non-consumable
- *Issue 2*: If the player's inventory is full, when the player opens special locker8 containing the student card, no student card is added to the player's inventory and the game does not register that the card has been found meaning the player cannot escape from the hidden room - again leaving the player trapped
- Cause: Inventory.addItem() returns false if the player's inventory is full but the student card is still removed from the locker and locker.open() is marked as opened, meaning its contents are cleared
- Fix: alter GameScreen to always accept a found student card by having a designated additional inventory slot just for the student card to go in

T_BUTTONS_ALL_WORK

- Issue: no settings screen has been implemented so the settings button can be pressed however no result occurs from the button press
- Cause: no additional features requiring a settings menu were implemented into the game therefore there was no need for a settings menu
- Fix: delete settings button from MainMenu or implement features requiring settings