Testing

Cohort 2 Team 5

Harry Beaumont-Smith
Tom Nolan
Will Punt
Ruth Russell
Mimi Shorthouse
Lottie Silverton
Stanley Thompson

This document shows our approach to Software testing for our software development project. The purpose of this document is to show that our 47 requirements have been tested and validated through systematic testing, as well as ensuring the system operates correctly. Throughout this document, we will describe the testing methodology used, explain the chosen approaches of testing and the verified results which are provided via structured tables showing traceability between our tests and requirements.

Our testing approach was to use an iterative approach early on in which it evolved further in development. We adopted a double approach to testing which included using a balance of both automated unit testing and manual testing. Throughout development, we aimed to make our code testable, so when encountering code that was not compatible with testing, we refactored that code. This included separate logic from the libGDX framework to be able to do more automated testing. For certain components containing more complex logic, such as calculations , we incorporated test-driven development principles. In these cases for example the score manager class and the achievements, we wrote JUnit tests before adding the functionality to the classes, to help us define how we expect the class(s) to behave. For components which needed user judgment, we utilised manual testing to validate gameplay and navigation via user testing.

When testing, we primarily focused on two approaches, which were chosen in regards to what was most appropriate regarding the aspect it affects within the system. The first approach is automated unit testing. We aimed to prioritize this approach due to its efficiency, scaleability, immediate results, code design and reliability. For this we used the Junit framework which has a good compatibility of our codebase and its simplicity of application. Our automated testing was especially useful when working with data structures, file handling and logic. However, there are limitations to automated testing therefore a different approach called manual testing was used. We found that manual testing was essential in several areas to be able to fully fulfil our requirements.This is because we needed user judgment and interaction. This could be because components need an input, a user needs to see or hear an output and user intuition is needed to be able to validate a test. Some examples include assessing the navigation, audio, graphics, object boundaries and system feedback. As well, components with dependencies on the libGDX framework needed manual interaction due complex integration and runtime behaviour.

Overall, we conducted 38 automated unit tests and 18 manual tests, totalling 56 tests. This covers 45 out of our 47 requirements. The two requirements which failed in some aspects in testing were UR_FAST and FR_Performance_Display. UR_FAST involved real time scoring which utilised runtime which is deeply integrated in the libGDX, therefore only manual testing was used so it was only partially tested. FR_Performance_Display this requirement was not met due to it not being integrated within the final code, this was due to a group agreement that it was no longer considered important as it had no use for the user. In regard to the two requirements regarding assets, we made sure before hand that all third party assets used were sourced and verified. As well for our testing evidence, we have provided a separate document which contains the descriptions and evidence of the tests.

Manual Testing Results			
Requirement ID(s)	Manual test ID	Description	Pass /Fail
UR_Player, FR_Player_Displayed	MT_PlayerTest	Tests that the game contains a player character which can move using (WASD)	Pass
UR_Pause, FR_Pause, FR_Pause_Menu	MT_Pause	Tests the pause menu which pauses the game by pressing on a key on the keyboard	Pass
UR_Tutorial, FR_Tutorial	MT_Tutorial	Tests that the game teaches the user how to play the game by giving instructions on the settings page	Pass
UR_Map_Limits,FR_ Map_Limits	MT_Map	Tests that there is a clear boundary visible to the player, and blocks the player from going there	Pass
UR_Dean_Object, FR_Chasing_Dean	MT_Dean	Tests there is a dean entity, which chases the player and interacts with the player	Pass
UR_Accessibility, NFR_Accessibility	MT_Accessibilit y	Shows that the game visually is readable and easily comprehensible	Pass
UR_Score, FR_Score_Counter, FR_Fast	MT_Score	Tests the score is clearly presented at the end of the game and works correctly	Pass
UR_Consistent_Maze , UR_University	MT_Maze	Test to see that the maze should be constant and university themed	Pass
UR_Main_Menu, FR_Main_Menu	MT_MainMenu	Tests to see if there is a interactable main menu screen which contains appropriate buttons	Pass
UR_Settings, FR_Settings	MT_Settings	Sees if there is a visible screen which allowed user to update the audio through user input	Pass
FR_Audio, UR_Audio	MT_Audio	See if the game gives feedback to music and sound effects.	Pass
FR_Performance_Dis play	MT_Performan ceDisplay	Tests there is a visible frame per second which can be switched off	Fail
FR_Data_Storage, UR_Achievements, UR_Leaderboards	MT_Achieveme nts	Tests to see if achievements and leaderboards are visible to the user	Pass
FR_Timer, FR_Losing_Time, UR_Time_Tracker, UR_Lose_Condition	MT_Timer	Tests there is a timer visible during the game showing the time left in the game	Pass
UR_Desktop_Inputs	MT_DesktopIn puts	Sees if the user should be able to interact and play the game with a mouse and keyboard	Pass
NFR_Compatibility,	MT_Compatibili	Sees if the user is able to play the game on a device with average	Pass

UR_Standard_Computer, NFR_Sys_Reqs	ty	hardware and is compatible with the user's computer.	
UR_Invisible_Events, FR_Event_Counter, UR_Negative, UR_Positive	MT_Event	Tests if there is a visible event counter and hidden events are made discoverable and positive and negative events work and are visible.	Pass
NFR_Usability, UR_Audience	MT_Usability	Sees of the game is fast and easy and understandable for casual gameplay	Pass

Automated Testing Results			
Requirement ID(s)	Automated Test ID	Description	Pass/ Fail
UR_Achievements	AC_CheckConstructorAchievement	Tests that the AchievementManager constructor correctly initialises with the set settings	Pass
	AC_AddAchievement	Tests that addAchievement, when increasing the achievements it correctly increases the total count and sets the achievement to true	Pass
	AC_RemoveAchievement	Tests that removeAchievement, when removing a specified achievement which is currently set to “true” would now be set to false and the total decreases	Pass
	AC_ResetAchievement	Tests that reset restarts the achievements to their default state appropriately	Pass
UR_Leaderboards, UR_Data_Storage	FM_ReadFileMap	Tests that readFile correctly parses a specified CSV file into a hashmap with the correct pairings of name and score	Pass
	FM_ReadOverrides	Tests that readfile can handle duplicate names appropriately by keeping the most updated score	Pass
	FM_EmptyMap	Tests that readFile returns empty hashmap when the file is empty	Pass
	FM_ReadStringFirstLine	Tests the readFileString returns only the first line of a selected file	Pass
	FM_ReadStringEmpty	Tests that readFileString returns null when reading an empty file	Pass
	FM_WriteMaxFive	Test that write file contains a boundary output to a maximum of 5 entries	Pass
	FM_WritesScoresCorrect	Tests that write file writes all entries correctly	Pass
UR_ScoreManager, UR_Score,UR_Fast	SM_calcFinalScore	Tests that calculate Final Score calculates a correct score based on events and time	Pass
	SM_increaseScore	Tests increase score adds to a current store correctly	Pass

	SM_Reset	Tests that reset correctly resets total to zero	Pass
	SM_Duplicate	Tests to see if it correctly sees if it handles duplicates correctly	Pass
UR_Time_Tracker, FR_Timer, NFR_Time_Constaint	TM_SetCorrect	Tests Timer constructor initialises the time correctly	Pass
	TM_DecreaseTime	Tests that update decreases the time left appropriately	Pass
	TM_IncreaseTime	Tests that update increases the elapsed time appropriately	Pass
	TM_isFinishedIsNeg	Tests that time left never goes below the boundary zero and isFinished is set to true when it reaches zero	Pass
	TM_hasReached	Tests that has reached returns true when reached selected amount	Pass
	TM_setFrozen	Tests set frozen that it prevents time from updating when frozen	Pass
	TM_isFinishedBool	Tests is finished returns true when elapsed time reaches the end	Pass
	TM_FrozenTimeContinues	Tests that time continues to update normally after being frozen	Pass
	TM_Reset	Tests reset restores the timer to the default state	Pass
UR_Negative, UR_Positive, UR_Invisible_Events, FR_Event_Counter	EV_SetCorrect	Tests constructor is initialized correctly	Pass
	EV_EnterRoomNeg	Tests that on enter room with a negative event calls a negative start event	Pass
	EV_EnterRoomPos	Tests that on enter room with a positive event calls a positive start event	Pass
	EV_EnterRoomHidden	Tests that on enter room with a hidden event calls hidden start event	Pass
	EV_ExitRoom	Tests on Exit room clears any active event so it cannot be called	Pass
	EV_drawWorld	Tests that draw is called and active from draw world	Pass
	EV_registerEvent	Test that when event type is none it does not increment the event counter	Pass
	EV_ValidCount	Tests that when get triggered contains the correct amount of events	Pass
	EV_BacktoZero	Tests that reset all event counts are set back to zero	Pass

Testing method	Number of tests	Total Passed	Requirements met	Pass percentage
Automated Tests	38	37	19	97.4%
Manual Tests	18	17	30	94.4%
Total	56	54	47	96.4%