Software Testing Report

Team 1:
Megan Forrow
Jonathan Broster
Sophie Bailey
Alastair Johnston
Luiz Oliveria
Alina Merkulova
Tameem Abuhaqab

Testing Methods and Approaches

As our game is small, the most efficient approach is to use small tests so that as few resources are used, the tests can run faster, and we can test over a mostly narrow scope so that if the test fails there are less lines to check.
We aim to have our tests follow the “pyramid” shape, such that roughly 80% of our testing are “unit” tests which test small parts of the game like individual classes and methods, roughly 15% of our testing is “integration” testing, which tests interactions between components, and around 5% of our testing will be “end to end” testing, which covers multiple parts of a system and things that are harder to test in isolation which involves testing one part by replacing it with a “test double” so that there is no influence from other parts of the system.
These test doubles can be “fakes”, which are lightweight copies of the original part, and should pass the same tests as the original to ensure fidelity, or test doubles can be “stubs” which are hardcoded results of the original. We will be using mainly black box testing, which involves testing based purely off the requirements, with some white box testing, which involves testing based on the code itself.
The testing will be automated for the majority as it is both time efficient and resource efficient but will have some manual testing for areas that require human intuition. We also aim to have an acceptable coverage of our tests, so that each block of code being tested has every possible test case be accounted for, including failure scenarios.
Our tests won’t be brittle and so will use public interfaces, and our tests will have clarity so that automated tests won’t have any irrelevant information, but contain everything the reader needs to understand what the test does.
The tests themselves will be JUnit tests and will consist of its own function using the format where “XTest” is a function that tests “X”, which will return void and use the @Test annotation. They will also use a “given-when-then” structure and use assert statements to verify expectations.

Report

1.1: testKeyDisappears - testing correctness (key should disappear when player interacts with it), tested to ensure the player knows they have picked up the key, tested on Jan 6th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

1.2: testCrateDisappearsWithKey - testing correctness (crate should disappear when key has been picked up), tested so it is possible for the player to progress once they have unlocked the crate, tested on Jan 6th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

1.3: testBridgeCollapse - testing correctness (bridge collapses when interacted with the player), tested to ensure another obstacle for the player to overcome, tested on Jan 6th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

1.4: bridgeRepairTest - testing correctness (bridge should be repaired after collapsing), tested so that the player can still use the bridge after the collapse event, tested on Jan 6th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

2.1: winTest - testing correctness (if the player stands on the WinCondition tile it should trigger victory), tested so that the player can win the game, tested on Jan 4th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

2.2: loseTest - testing correctness (if the timer runs out and the player hasn't won yet, the game ends), tested so that the player can lose the game, tested on Jan 4th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

2.3: testScoreCalculation - testing correctness (score calculated should match expected calculation), tested so the player's score is accurate to what it should be, tested on Jan 7th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

3.1: testTimerStartPauseResume - testing correctness (timer should be running/not running at the appropriate time), tested so that the timer runs normally but isn't running while the game is paused, tested on Jan7th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

3.2: testTimePauses - testing correctness (timer shouldn't decrease while paused), tested so that the player's score doesn't change while the game is paused, tested on Jan 7th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

3.3: timerCountsDownToZero - testing correctness (timer should stay counting down to 0), tested so while the game is playing the timer doesn't stop, tested on Jan 7th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

4.1: testTileWithObstacleIsNotSafe - testing correctness (whether the tile is safe), tested to ensure whether the tile is safe, tested on ?, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

5.1: testMovementRightChangesPosition - testing correctness (the player should move right when the correct key is pressed), tested so that the player is able to move right, tested on Dec 14th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

5.2: testMovementRightSetsDirection - testing correctness (the player direction should change when the player has moved right), tested so that the player's direction matches the direction of movement, tested on Dec 14th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

6.1: rightCode - testing correctness (when the right code is inputted, event2 should increment), tested so that the event counter increases when the player successfully completes an event, tested on Jan 5th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED
6.2: wrongCodeTest - testing correctness (when the wrong code is inputted, the dean should spawn), tested so that the dean spawns to chase the player when they input the wrong code, tested on Jan 8th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

6.3: testStoneRemoval - testing correctness (when the right code is entered, the stone should disappear and once all stones are removed, the stone barrier should be disabled and event2 should reflect that), tested so that the player can progress past the stone barrier, tested on Jan 8th, tested in group meeting, written by Luiz, executed by Luiz, analysed by Alastair, test PASSED

Traceability Matrix

	Tests	1.1	1.2	1.3	1.4	2.1	2.2	2.3	3.1	3.2	3.3	4.1	5.1	5.2	6.1	6.2	6.3
Requirements																	
UR_MOVEMENT													X	X			
UR_DEAN																X	
UR_POSITVE_EVENTS		X	X												X		X
UR_NEGATIVE_EVENTS				X												X	
UR_HIDDEN_EVENTS				X	X												
UR_EVENT_COUNTER		X	X	X	X										X	X	
UR_ACHIEVEMENTS								X								X	
UR_LEADERBOARD						X		X									
FR_TIMER							X		X	X	X						
FR_PLAYER_MOVEMENT													X	X			
FR_INTERACT_EVENTS		X	X	X								X			X	X	X
FR_DEAN_AI																X	
FR_ACHIEVEMENTS																X	

URLs

Below is a link to the website - the testing is found under the testing expandable button on the main page.

- *https://mforrow310.github.io/glitchgobblers.github.io/*