CSCI265 — Testing Plan

Team Badass

Abstract

This document contains the testing plan for the All Who Wander project, the Team Badass final project for CSCI265 Software Development. Contained within this document is a written summary, for clarity and posterity, of the streamlined testing procedures to be employed to ensure functionality of the project.

These procedures will be divided by category (domain of function) and then subdivided into unitary subsections. Each test will outline the initial state required for the test, the sequence of actions to be performed by the developer to perform the test, and the expected output.

Player Controls

Player controls are the scripts which translate input from the user into action by the player entity within the gameworld. Actions include movement, the execution of attacks using various weapons, and the triggering of an “interaction” event with other gameworld objects. Many such actions trigger corresponding animation sequences in addition to their primary effects.

Thus, to “test” a player control means ensuring an input is properly triggering some primary action and its accompanying animation.

Some actions have conditions placed upon them (another object must be in range, etc.) and some actions can only be performed whilst the player in a specific state (a player cannot attack with two weapons at once).

Therefore, some additional measures are required to ensure that input only maps to actions when those actions are valid.

Movement

Movement occurs when the player uses the joystick or arrow keys on the keyboard. Movement causes the player’s position to change, and the camera to follow the player. While moving, the player enters a “walking” animation state, and is displayed accordingly. Movement cannot occur if the player is an attacking state. Though the player has 360 degrees of freedom in which to move, they can only face in one of the four cardinal directions at once.

Initial Conditions
Idle in the center of an empty space.

Actions and Observations
Press and hold for 1 second each of the four arrow keys, in sequence. The player should move without encumberment in each of the corresponding directions. The player should enter into a walking animation state for each, facing the right direction.

Press and hold for 1 second each pair of arrow keys, in sequence. If the two arrows correspond to diagonal movement, the player should move diagonally. When moving diagonally, they should face in the corresponding x-axis direction. If the two arrows are in opposition, then the player should not move and be in an idle animation state.

Press and hold for 1 second each triplet of arrow keys. In any triplet of keys, to keys should be in opposition. Thus, the player should move and be facing the unopposed direction.

Press and hold for 1 second all four arrow keys. The player should not move and should be in the idle animation state.

Movement with Shield

The player may walk whilst holding out their shield. The mechanics of this are identical to movement without a shield, except movement speed should be reduced.

Initial Conditions
Idle in the center of an empty space.

Actions and Observations
Perform each of the tests outlined in the subsection above ("Movement") whilst also holding either of the "shift" keys. The results should be the same as in the tests above, except that the player's shield should be visible.

(Not) Walking through Solid Objects

Most objects as well as many environmental features in the game are impassable. When movement in any direction would cause the player to collider with such an impassable entity, the player's and their movement through the world should cease. The player's animation state should remain in walking-state and facing the appropriate direction.

An effect known as "sliding" occurs when the player has both x- and y- axis movement at once, and movement is incumbered on one of these two axes. When this occurs, movement in the unincumbered direction is executed, and movement in the other direction is ignored. Animation should remain in walking state as though it were unincumbered.

Initial Conditions
Idle in the center of a square of impassable objects such as walls (exactly five tiles wide and tall).

Actions and Observations
Move in each of the four directions in sequence. The player should move and be animated accordingly in each direction. When the player encounters the wall their movement should end, though their walking-state animation should continue as normal.

Move back to the center of the square.

Move directly up until the player encounters the wall. Then, move both up and to the left. The player should move to the left, sliding along the wall. Repeat this process for each of the diagonal directions.

Attack Execution

The player can attack with four different weapons: a sword, a spear, a bow and arrow, and a magic spell. The player can also "interact" with another object, which invokes similar machinery though without an animation. Each action is mapped to a specific and constant key.

Once triggered, each attack executes a 1-second animation. Attacks can be triggered in (can face in) each of the four directions. Attacks can be triggered whilst walking with or without a shield but cannot be triggered whilst another action is being executed.

Initial Conditions
Idle in the center of an empty space.

Actions and Observations
Whilst idle, strike each action key (giving each enough time for one to finish before striking the next). An animation should play for each. For spells and arrows, a projectile should be fired.

Add a "Debug.Log" statement to the method associated with each action key, within the conditional to indicate that the action is being executed. Strike each pair of actions in quick succession. The first action should fire and be visible in the console, but the second should not. If this is the case, then actions are not executing whilst other actions are in progress.

Projectiles

Projectiles are independent game objects spawned by long-range player attacks. These objects move in a straight line until they encounter another solid object and are then destroyed. If the

solid object is an enemy or the player (if the projectile was spawned by the player or an enemy, respectively) then damage is dealt to the collided with entity.
Magic spells have the secondary effect of triggering a flashy animation if they collider with an enemy.

Initial Conditions
Idle in the center of a square of impassable objects such as walls (exactly five tiles wide and tall).

Actions and Observations
Face in each direction and strike the bow-and-arrow key for each. An arrow should shoot from the player, collide with the wall, and vanish. Do the same for the magic-spell and observe the same results.

Remove the walls, and instead, place an enemy directly in-front of the player with its motion toggled off.

Fire an arrow at the enemy. It should enter the recoiled animation state, indicating that damage has been dealt.

Fire a spell at the enemy. A secondary, flash animation should engulf the enemy for 1 second, and then disappear. The enemy should then enter the recoil animation state.

Enemy and NPC AI

Enemies and NPCs are controlled by an artificial intelligence which commands their movement, their tracking towards the player, and their attacks. NPCs and enemies in their “idle” states will choose a random direction to move in, move for 1 second, and then stop and idle for 1 second (then repeat this sequence). When the player gets within a certain range of an enemy, the enemy will attempt to move towards the player and then attack. If the player leaves this range, the enemy will return to its idle state.

Enemy and NPC Patrolling

Enemies and NPCs engage in patrolling behaviour. When patrolling, these entities will choose a random cardinal direction and move in that direction for 1 second. They will then stand idle for 1 second, before repeating this cycle.

Initial Conditions
Spawn an enemy directly in the center of an empty space.

Actions and Observations

Observe the entity. The entity should move in some direction, stop, and then move again in some direction.

Chasing

Enemies will, when the player gets within chase range, attempt to move towards the player and then attack the player. Since enemies must face the player direction along the x-axis, enemies will actually move towards the space directly in front or behind the player (on the x-axis).

Initial Conditions
Spawn the player and an enemy across from each other on the x-axis, with the player to the left of the enemy and with a reasonable amount of space between then. Toggle off roaming behaviour.

Actions and Observations
Walk the player directly towards the enemy. The enemy should walk directly towards the player and then stop when it is just in-front of the player. If the player leaves chase range, the enemy should return to its roaming state.

Respawn the enemy and the player in the appropriate positions as to perform the same test as above, but with the enemy approaching from each of the 7 other directions (3 cardinal, 4 diagonal). In each test, the enemy should start moving towards the player as the player enters chase range, stop when they are directly in front of the player on the x-axis, and then attack.

Reset the player and enemy to the first test state. Move the player to within chase and range and observe that the enemy begins to chase the player. Quickly move away from the enemy and observe that the enemy stops chasing.

Registering Attacks and Damage

Damage is dealt from one party to another when an attack is triggered by one party, and another party is within attack range (and is directly in front of the attacker). Once one party has been attacked, it is invulnerable for 1 second. The player does not have a recoil animation, but enemies do. When an enemy is dealt damage, it enters into its recoil animation and is invulnerable for 1 second. The enemy then returns to either its idle, chasing, or attacking state.

Player Attacking Enemies

Initial Conditions
Spawn an enemy directly in front of the player, and toggle off its attack behaviour.

Actions and Observations

Repeatedly strike the player's various attacks. For each attack, the enemy should enter into its recoil state. After 1 second, the enemy should exit its recoil state. Continue attacking, repeating this cycle. Eventually, the enemy should despawn, having been destroyed.

Enemy Attacking Player

Enemies attacking the player works exactly like the opposite, except enemies have a 1 second cooldown after their attack animation ends.

Initial Conditions
Spawn an enemy directly in front of the player, and toggle on its attack behaviour.

Actions and Observations
Stand idle, allowing the enemy to repeatedly attack the player. Enemies should attack, stand idle for 1 second, and then attack again. Eventually, the player should despawn, having been destroyed.

Spawning

The spawning system is responsible for systematically repopulating the world with enemies. The spawning system is embodied by "spawn points" placed manually throughout the world. Each spawn point will, at the start of the game, spawn an enemy near to it. Whenever the enemy is destroyed, the spawn point registers this and becomes primed to spawn a new enemy. Once the spawn point is further than a minimum distance from the player (is off screen, more or less) it will respawn its enemy.

Initial Conditions
Place a spawn point somewhere on the screen. Also, place the player somewhere on the screen.

Actions and Observations
Start the game. An enemy should appear near the spawn point. Defeat the enemy. Walk in any direction until the spawn point is off screen. Walk back to the spawn point. A new enemy should have appeared.

Items

In game items are GameObjects with unique sprites and interactions based on their specific role within the game. Items are tested in the pocket versions of the game with the items placed "by hand" in the situations listed below. An items functionality and expected outputs are compared using in game interfaces, as well as console logs. An item is first tested under normal situations, once this is passed further tests are completed. Items can be interacted with using two different methods: player collision or user hotkey - these are referenced as trigger styles below.

1. Item placed alone and interacted with.
2. Two identical items place on top of each other.
3. Two different items with the same trigger style placed on top of each other.
4. Two items with different trigger styles placed on top of each other.
5. Items spawn at Player position.
6. Items spawn at Enemy position.

Looting

The loot system relies on a GameObject which holds references to different loot items in the game. When an enemy dies it instantiates one of these items at random using the product of a random number generator and the enemy's level.

Initial testing includes controlled situations of enemy death and observing the item output. Additionally, the value of the "loot roll" is displayed within the console so it can be checked against the expected output. Once this is confirmed additional testing follows.

Simulation of large amounts of enemy deaths are done within Unity's game world, with large quantities of enemies dying after a set amount of time. The appropriate rate of loot dropped is checked against expected results.

Leveling

A player level is gained when the player has collected experience points up to the next milestone amount. Once a level is gained, permanent changes happen to the players stats and a new milestone amount is set.
To test the levelling system intractable objects are placed in a line in front of the player object evenly spaced out. Each item gives enough experience points to progress the player to the next level in the system. Once interacted with the items destroy themselves.

Initial testing follows the sequential order of items, checking at each one if the player level is gained and if the appropriate stat changes are implemented using Unity's built-in object inspector. Additional testing takes uses the items in a random order to determine if the same level is reached as the initial test, and if any changes are seen in the player stats at each level.

User Interfaces

The general outline of each user interface contained in all who wander can be tested upon the initial startup of a new game even if it is the first execution of the build. Specific functionality may require minimal setup. Because of the graphical nature of the user interfaces, only manual testing will be performed. Very literal 'visual verification' will be performed for most of the UI testing where success is determined by physically observing the outcome of the actions of interactable UI elements and UI rendering.

Start Screen

The first screen displayed when executing a build of the game. Composed of a general background with some buttons with text.

Initial Conditions

Open the start screen by running an executable build of the game.

Actions and Observations

First visually verify all UI elements have been rendered properly (shape, colour, size, pivot/reference to other elements). Then click each button and verify that its callback function works as intended. Each button is responsible for a view change, visually inspect to make sure the correct view was rendered after selecting a button.

Settings Screen

Accessible from two different UI screens, the pause screen and start screen.

Initial Conditions

Test rendering from both the start screen and pause screen.

Actions and Observations

First visually verify all UI elements have been rendered properly (shape, colour, size, pivot/reference to other elements). Then once the settings screen is visible, click each button and verify that its callback function works as intended. The call backs will be related to actual game settings like sound, resolution, and difficulty (time dependent), and one callback will be to close the settings screen. These settings will all have to be verified manually by the user that is testing (Play sound and see if the volume slider functions, change resolution and see if the screen changes, etc).

Overlay Screen

Displayed concurrently with the active game. Always present when the game is in active run-time.

Initial Conditions

A game save that has 10 items spawned in front of the player, and attached scripts to the UI canvas (game object containing all UI Elements) that increment/decrement health points and experience points.

Actions and Observations

First Visually verify all UI elements of the overlay have been rendered properly (shape, colour, size, pivot/reference to other elements). This includes the HP Bar, the XP Bar, the Hotbar, and the pause screen button.

Then increment and decrement health points, from minimum to maximum (ensure slider changes with each increment decrement). At minimum, which is zero health points, the player's death animation should activate and a "game over" prompt will show. Visually verify this happens correctly.

Similarly increment and decrement the experience points and visually verify that the "experience bar" responds correctly.

Pickup an item and press its assigned hotkey (visible in hotbar now), make sure its animation activates properly. Now fill the entire hotbar with 10 items and verify all slots have the same outcome as with one item.

Run an in game timer that increments each second. Press the pause screen button and visually verify that the view changes to the paused screen. Close the screen and verify that the game actually paused by looking at the timer.

Pause Screen

Initial Conditions

Activated from the overlay screen while in active run-time.

Actions and Observations

Rendering is verified in the testing for the overlay. Click all buttons and verify their call back functions work as intended.

Inventory Screen

A visual representation of a list of items the character currently possesses.

Initial Conditions

Load a game with 100 items.

Actions and Observations

First Visually verify that the inventory appears when the hotkey ‘i’ is entered and subsequently will disappear when ‘i’ is hit again. Then verify all UI elements of the inventory (once opened and visible) have been rendered properly (shape, colour, size, pivot/reference to other elements).

Pickup less than the maximum amount of items possible (For initial testing n=30 but subject to change). Verify that items picked up are presented correctly in the inventory, that their sprite appears in the grid. Then try moving items around the grid and verify that inventory manipulation works as intended and that the current grid that is active is highlighted.

Hover over an item (should be highlighted then) and assign it a hotkey. First verify that the item was successfully assigned the hotkey, the hotkey should be present in the top left corner of the item image. Close the inventory and verify that the newly assigned item is present in the overlay hotbar, and further verify that upon activation of the hotkey the item animation will run correctly. Do this for 10 items to verify when the hotbar is full. Make sure when full that items can still be added (a current hotbar item will be replaced by the newly assigned one).

Dialogue

A testing strategy will be devised (likely very similar to other UI) for the Dialogue Prompts if we have enough time to implement that feature.