Compile a document that records the following information:

1. A list of bug reports or issues uncovered during testing.

This can be in a spreadsheet, or exported in Word or PDF from bug tracking software.

## Results from play testing sessions and user evaluation.

*You must run at least one playtesting session, with at least two play-testers.*

### Play-tester 1: Michael Burford (2023 AIE programming student - WebGL)

Enjoyed the concept, thinks it would be fun to play, should lend itself well to touchscreen. Suggested I should add a noise and message when the user combines ingredients which fail to produce a potion (**suggestion taken up and added**). Suggested maybe making the controls compass directions rather than relative to the character’s facing (**not taken up due to A) complexity of recalibrating the user inputs to synchronise with the animations and b) I personally prefer that the controls are relative to the facing**).

### Play-tester 2: Mitchell Bracken (personal friend, self-taught programmer - tablet)

Loved the use of music and sound effects (Mitchell is an audio engineer), enjoyed the click-to-move navigation. Found a bug where only one audio track played rather than randomly choosing the next track (**fixed – bug was due to me converting a .wav audio track to .mp3 and then failing to re-import it to the project**).

### Play-tester 3: Shaun Duggan (personal friend, gamer - laptop)

### Play-tester 4: Andrew Ellis (personal friend, gamer – WebGL through mobile phone)

### Play-tester 5: Cliff Sharif (2023 AIE programming lecturer)

## Run a performance profiler and analyse the results. Write at least 200 words analysing the performance of your prototype.

1. Which parts of your game are the most processor intensive/least optimized

A

1. Is the prototype optimized for the target platform(s)

A

1. What practical implications might exist from your analysis

A

Profiler memory management in a typical moment:A screenshot of a computer

Description automatically generated

## Describe (at least 100 words) how the results of testing, debugging and profiling influenced development or informed changes to the prototype’s design or implementation.

Iterative testing was the principal mechanism for determining my code implementations. Being new to Unity, I regularly wrote a handful of public variables or functions, observed the way they looked and might be used in the Inspector while the game was running. I would then return to the source code to decide whether or not it made sense for them to remain public, whether names were fitting, whether it made sense for an implementation to be built with code or built with in-engine tools. Live testing the game for myself was my principal means of determining the location and size of UI elements, whether buttons, sliders, menus, text, icons, colours, contrast, brightness or other elements.

I learnt that debugging in Unity can be incredibly easy, or incredibly difficult – some error codes and warnings are so descriptive that the problem is obvious, others are written such that they are completely intractable. C# and Unity are refreshingly error-free due to a reduction in complex memory management, but mathematical logic errors to do with rotation and translation remain very challenging, especially when coupled with animations, which I know nothing about and got working only through extensive reading and extensive trial and error.

Profiling did not affect implementation at the point of submission because, in full honesty, I had omitted it until after the project was ‘complete’ enough for submission, as optimisation was not at the forefront of my mind due to my starting the project 3-4 weeks late due to competing priorities with other assessments. My learnings from the profiler have been captured above in my analysis of my prototype, which I plan to implement in the future as this program can be used as a profile piece.