FIT3162 – Computer Science Project 2

Test Report (10%)

Due 11:55 pm Friday 11th October 2024

Bryan Daverel 32028644
Zacari Harrison 33119635
Hao-I (Benson) Lin 33049386
Charles Nierves 33120447
Emily Ung 33115958

Introduction

Our project involves developing an application that calculates the intersection over union metric of 2D convex polygon shapes, development of graphical user interface (GUI), random 2D convex shapes generation, and user self-defined shapes.

Throughout the development process, rigorous testing was performed continuously to ensure the correctness, robustness and usability of the application. We implemented a comprehensive testing plan that incorporated white box & black box encompassing unit testing, integration testing, and usability testing. This approach aimed to identify and address potential issues at every stage of the development process, from individual to the overall system integration.

In white box testing, it delves into the system's internal structure, ensuring that new code changes do not negatively affect the existing functionality, internal structure, and code logic.
The testing in the white-box section also encompasses a portion of the unit-tests within the application source code. Where we tested individual components of our application in isolation to identify and address bugs in every stage of the development cycle.

On the other hand, black box testing targets the external behaviour of the application, evaluating its functionality and usability from the user's perspective. Where we accessed how our system responded to a variety of input scenarios, ensuring the new updates don't break the application's user-visible functionality.

Integration testing verified that different components of our system function together seamlessly. We placed particular focus on the link between the front-end and the back-end of our application. By simulating user interactions, such as drawing shapes, verifying IoU values, etc. Confirming the integrated system met the functional requirements.

Usability testing adopts a user-centric approach by testing the software from end users' perspective. This was carried out with both guided and non-guided users, friends of ours, allowing us to understand and assess how intuitive and user-friendly our application was under real-world conditions and scenarios.

The following sections of this report will provide detailed description of the test approaches we used. This includes the use of Unity Test Framework (UTF) for unit testing as well as manual testing methods for integration, and usability testing. We will also outline the specific tests conducted and discuss the outcomes of our testing.

Testing Framework

The integrated development environment we used for development, a combination of the Unity Game Engine (and Development Toolkit) and Visual Studio, support Unity's proprietary testing framework (called Unity Test Framework, or UTF). This framework allows for the testing of code in both 'Edit Mode' and 'Play Mode'. Edit Mode allows for basic function testing outside the confines of other systems, simply compiling individual files and running tests on them rather than the whole system in an integrated setting. Play Mode instead simulates a sandbox environment wherein the conditions for using a given function are generated.

All unit tests at this current point in development use the Edit Mode as functionality purity identified as a point of great importance in the calculations we were testing.
Play Mode could be used to automate the Integration Testing and UI Testing process, however this comes with various complications due to the professional nature of the toolkit, and as such we have chosen not to pursue learning such a technology at this time, instead relying on manual testing for the relevant elements.

White Box Testing

Test #1 - IoU Test

This test focuses on ensuring all elements of the IoU pipeline are communicating correctly.
These elements are as follows

- Correct Function returns (return on invalid shape etc.)
- Polygon Area Calculation
- The calculation of the Intersection over Union metric

The CalculateIoU function() is harnessed by the testing framework for this approach, with the following inputs used to test elements (these are all sets of points so they shall be explained in an abstracted form)

1. Two shapes which fully overlap (they are the same shape at the same position)
2. Two Shapes which do not overlap
3. Two Shapes (A Triangle inside a rectangle of the same width/height) that partially overlap
4. Two invalid shapes

The results from the testing framework were as follows

1. As Expected: An IoU value of 1
2. As Expected: An IoU value of 0
3. As Expected: An IoU value of 0.5
4. Unexpected: A crash

For the fourth test we have noted that the function can not in fact handle invalid shapes. In this circumstance it was chosen not to refactor the function into being able to handle such shapes, but to instead throw an error when invalid shapes were provided as inputs. This is as invalid shapes should never be an element of the calculation pipeline if previous elements are working correctly, and as such handling them fully may mask errors elsewhere.

Test #2 - Intersection Testing

This test focuses on ensuring the size of the intersection calculated by the GetIntersectionPoints() function is correct.
This test is performed using two rectangles of varying size/location
The first rectangle stays in a static position, while the second rectangle moves closer to the first until their initial points overlap. The length and width of these rectangles varies to ensure that the intersection is not affected by extraneous factors.
The provided function is tested against calculated area values, which use basic geometry to identify the area of intersection of rectangles rather than the generalised 'Intersection Points' model we use.
The results of these tests were as expected, with all tests passing within a reasonable margin. On the first run a large number of tests were failing, however this was due to the strictness of the floating point comparison which was being run at the end of the test, something which has since been relaxed to account for floating point error.
No problems were encountered through this test.

Black Box Testing

****NOTE: Testing was completed on a version of the UI which has since been revised. Testing results stay consistent, button locations have simply moved***

Test #1 - UI Buttons

The objective of this test is to validate the functionality of the user interface (UI) buttons to ensure a smooth user experience.

The method involves having two quadrilateral shapes:

- One shape is generated using the Quadrilateral Shape Generator (triggered by clicking the square icon).
- The other shape is manually created by clicking four points on the canvas and connecting the first and last points to close the shape.

After generating the shapes, we will apply different button actions to validate the expected functionalities.

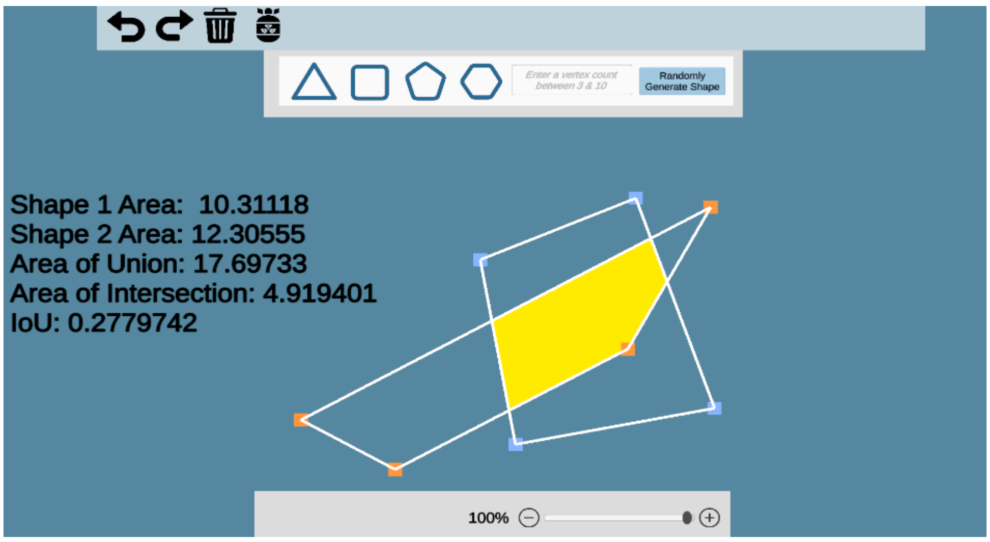

Shape 1 Area: 10.31118
Shape 2 Area: 12.30555
Area of Union: 17.69733
Area of Intersection: 4.919401
IoU: 0.2779742

Figure 1: Base Shape Generation

Undo Button

When clicked, the Undo button removes the most recently created edge or point, allowing the user to go back one step in shape creation. For example, if the user creates four points, clicking Undo will remove the last one and allow for further edits.

- Input: Click Undo Button
- Expected Output: Last edge or point is removed
- Actual output: The last edge was removed successfully

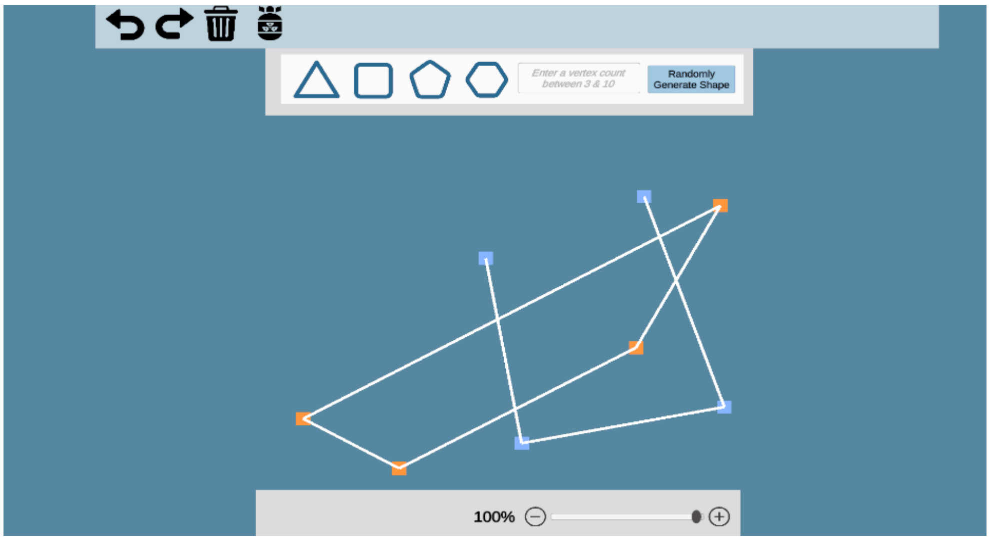


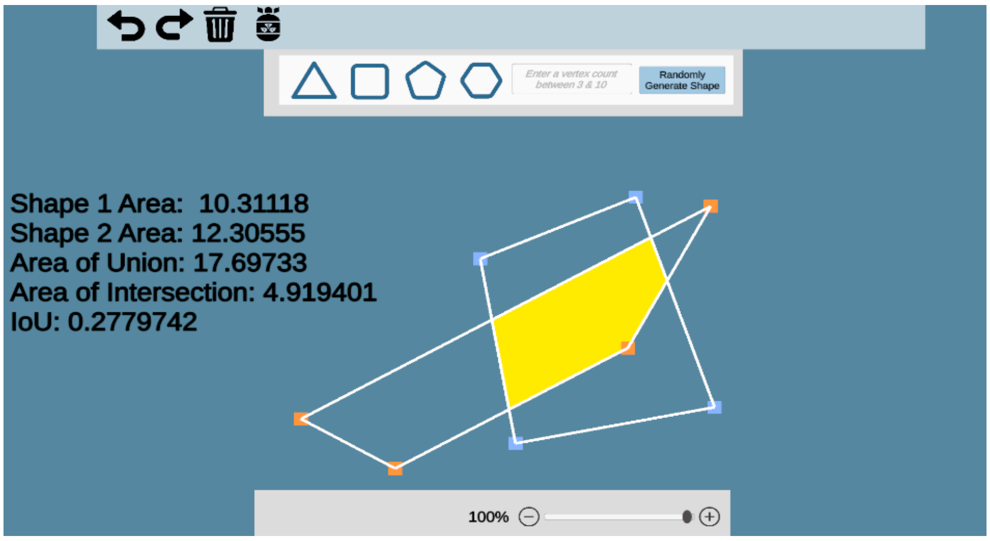
Figure 2: Undo Button has been Pressed

Redo Button

When clicked, the Redo Button will restore the most recent undone action.

- Input: Click Redo Button
- Expected Output: The previously undone edge or point should be restored

Actual Output: The shape was restored successfully after Undo



Shape 1 Area: 10.31118
Shape 2 Area: 12.30555
Area of Union: 17.69733
Area of Intersection: 4.919401
IoU: 0.2779742

Figure 3: Redo Button has been Pressed

Delete Button

The Delete button removes the most recent added shape from the canvas

- Input: Click Delete Button
- Expected Output: The latest shape should be removed from the canvas
- Actual Output: The shape was removed successfully

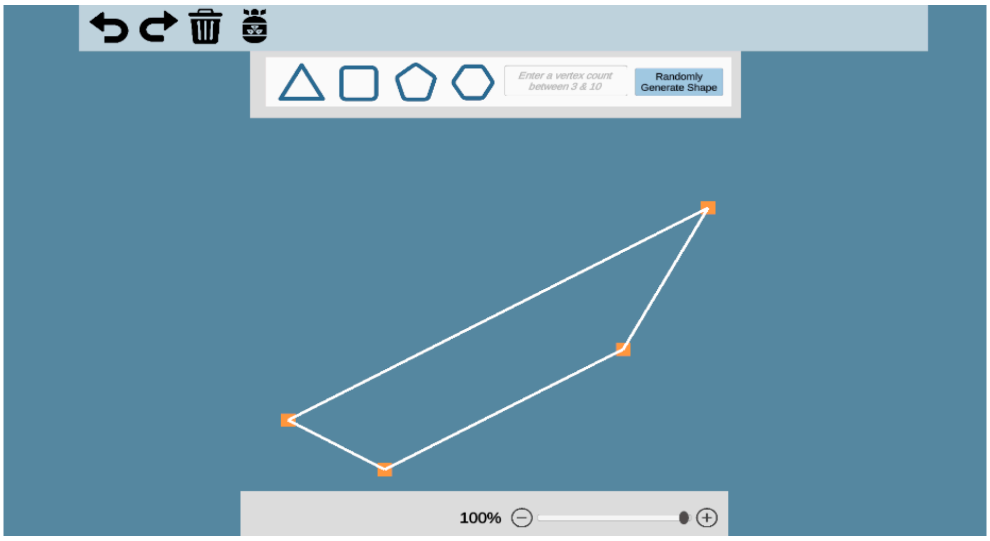


Figure 4: Delete Button has been Pressed

Reset Button

The Reset button represented by a bomb icon clears all shapes from the canvas, resetting the canvas to its original state.

- Input: Click Reset Button
- Expected Output: The entire canvas is cleared and no shape remains
- Actual Output: All shapes were cleared

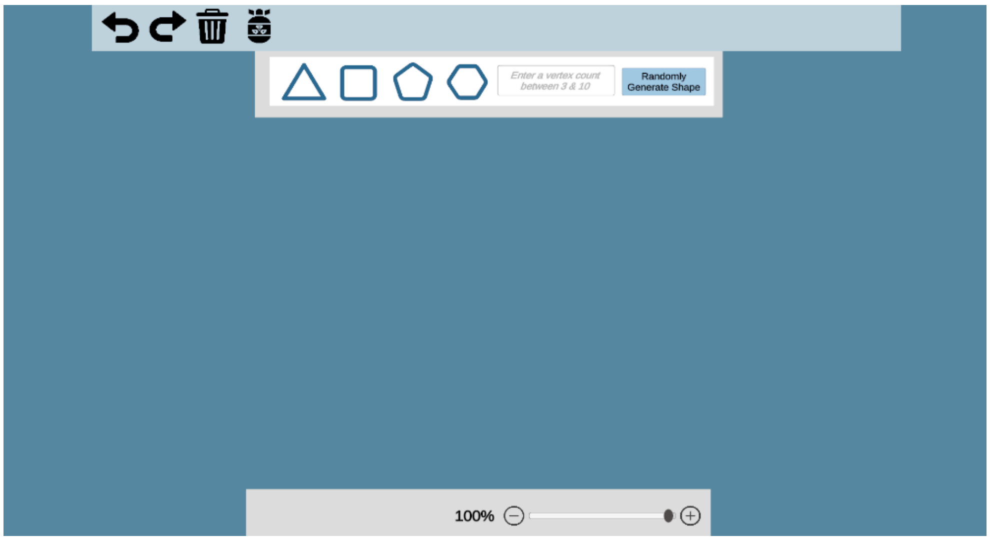


Figure 5: Reset Button has been Pressed

Random Shape Generator

When any shape icon is clicked, a shape with the specified number of vertices will be randomly generated on the canvas. Users can also input a number between 3 and 10 in the provided input box to generate a shape with a custom number of vertices.

- Input: Click on a shape icon or input a number of vertices between 3-10
- Expected Output: A shape with the selected number of vertices is generated.
- Actual Output: Shapes were generated as specified by the selected vertex count.

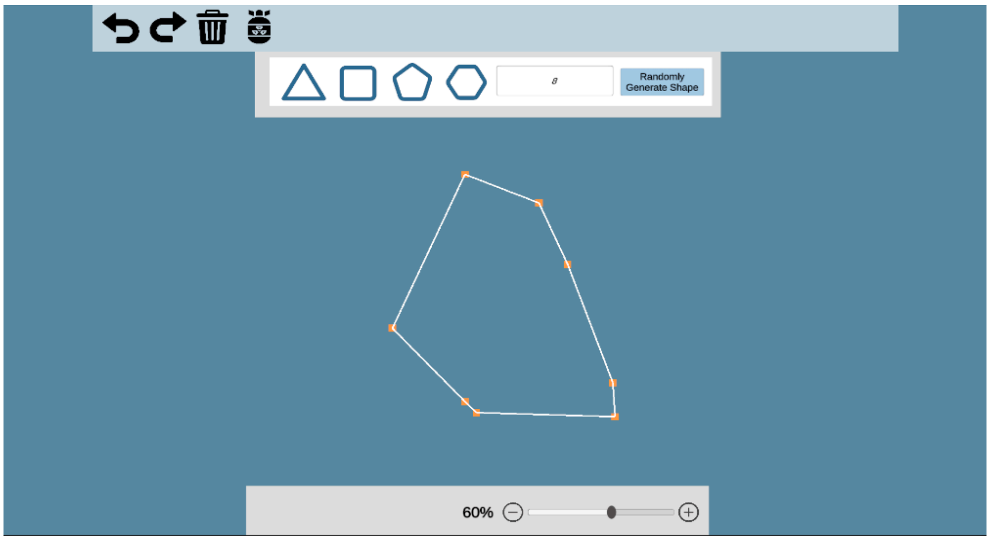
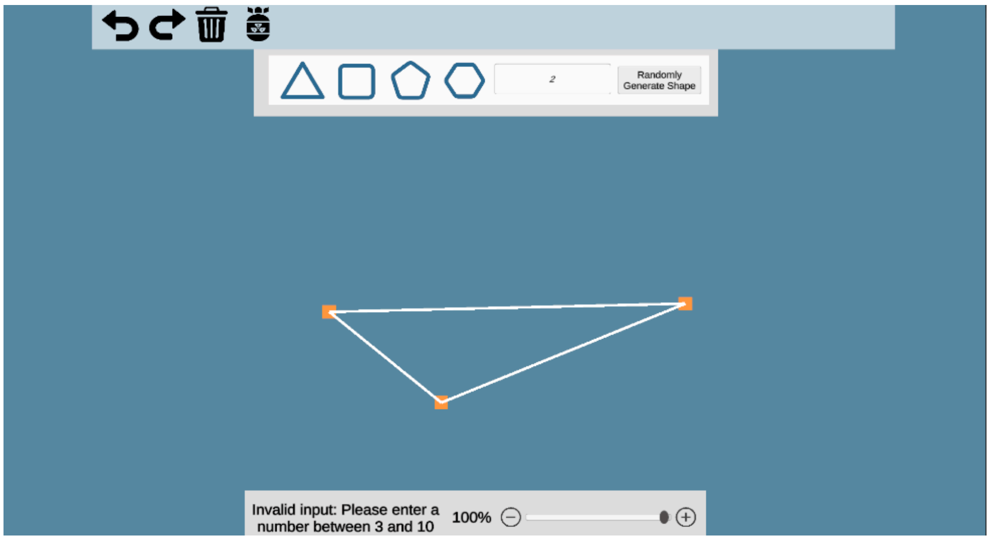


Figure 6: Random Shape Generation Button has been Pressed with input 8

Input Validation

If a user inputs a number less than 3 or greater than 10, the system should display an error prompt, informing the user that the value is invalid. The input box should only allow numeric input and ignore any other characters.

- Input: Any Character
- Expected Output: Validation prompt is displayed, only numbers between 3 & 10 are accepted.
- Actual Output: Validation worked correctly only allowing valid numeric input and the error prompt was displayed for invalid entries.



Invalid input: Please enter a number between 3 and 10

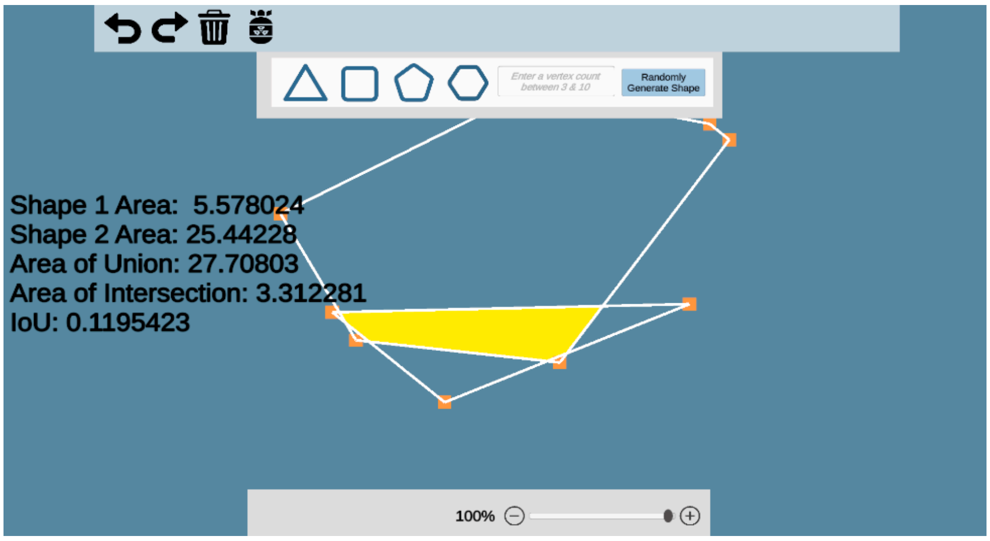
Figure 7: Random Shape Generation Button has been Pressed with invalid input 2

Zoom Functionality

The Zoom bar allows users to zoom in and out of the canvas. Users can interact with this functionality by:

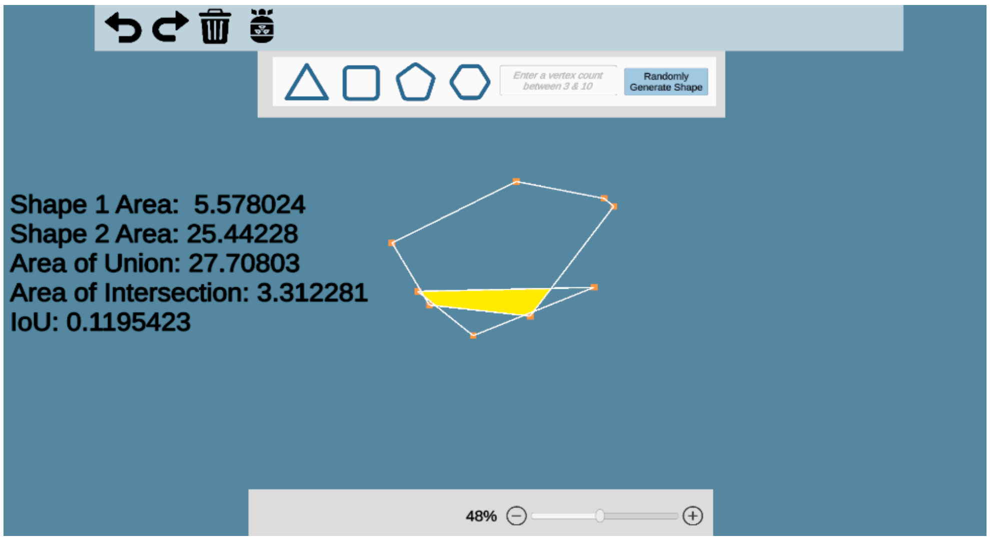
1. Scroll
2. Clicking and Holding the zoom slider to drag it to the desired zoom level
3. Clicking the '+' or '-' buttons to zoom in or out in 10% increment

- Input: Scroll or Click '+' or '-' or Drag Slider
- Expected Output: Zooming in enlarges the shapes, while zooming out reduces their size
- Actual Output: The zoom worked as expected



Shape 1 Area: 5.578024
Shape 2 Area: 25.44228
Area of Union: 27.70803
Area of Intersection: 3.312281
IoU: 0.1195423

Figure 8: Canvas before Zoom



Shape 1 Area: 5.578024
Shape 2 Area: 25.44228
Area of Union: 27.70803
Area of Intersection: 3.312281
IoU: 0.1195423

Figure 9: Canvas after Zoom Slider has been moved

Conclusion

The UI buttons and functionalities all worked according to expectation. All interaction yielded the correct results which improves user experience and ensures a reliable control over shape creation.

Test #2 - Drawing Shapes

The objective of this test is to validate that users are able to draw two convex shapes by clicking on the canvas and dragging any vertex or point while ensuring that the shape remains convex. The test will also validate the real-time update of Intersection over Union (IoU) calculations as points are moved on the canvas.

Testing method we will be using is that users will draw two convex shapes on an empty canvas. Each shape must have a minimum of 3 vertices and a maximum of 10 vertices. The system should ensure that each shape remains convex during the drawing process and while dragging any vertex. Additionally, IoU calculations will be updated in real-time based on shape interaction.

Shape Drawing

Users can click on any part of the canvas to place points which will form the vertices of the shape. Once the user clicks back on the first point, the system will recognise the shape as complete. Also a minimum of 3 and a maximum of 10 vertices will be enforced in the system.

- Input: Click on the canvas to place 3-10 dots
- Expected Output: The system will place dots and create a shape
- Actual Output: The convex shape is successfully created

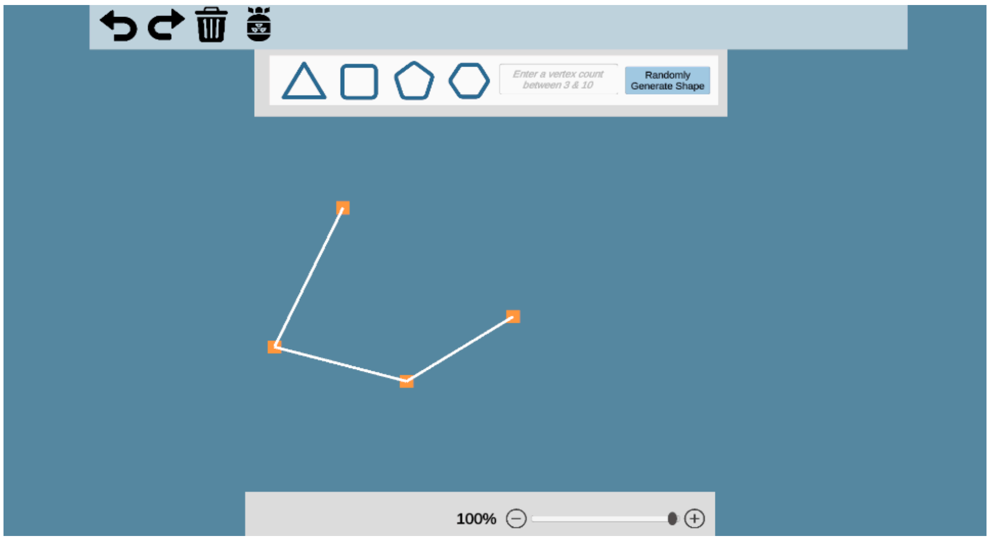


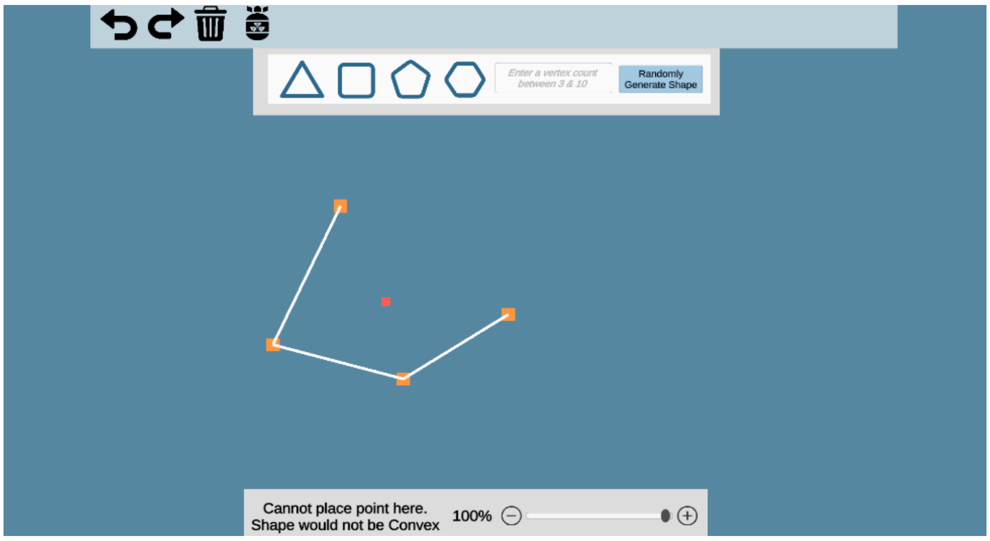
Figure 10: Shape is being drawn with Pen Tool

Convex Enforcement

During shape creation, the system continuously checks that the vertices being placed will result in a convex shape. Otherwise it will:

1. Display a warning message: “Shape would not be Convex.” & Point turns red

- Input: Click points that would result in a concave shape
- Expected Output: Warning prompt and red invalid point when non-convex shape is placed
- Actual Output: Warning prompt and red invalid point is displayed correctly



Cannot place point here.
Shape would not be Convex

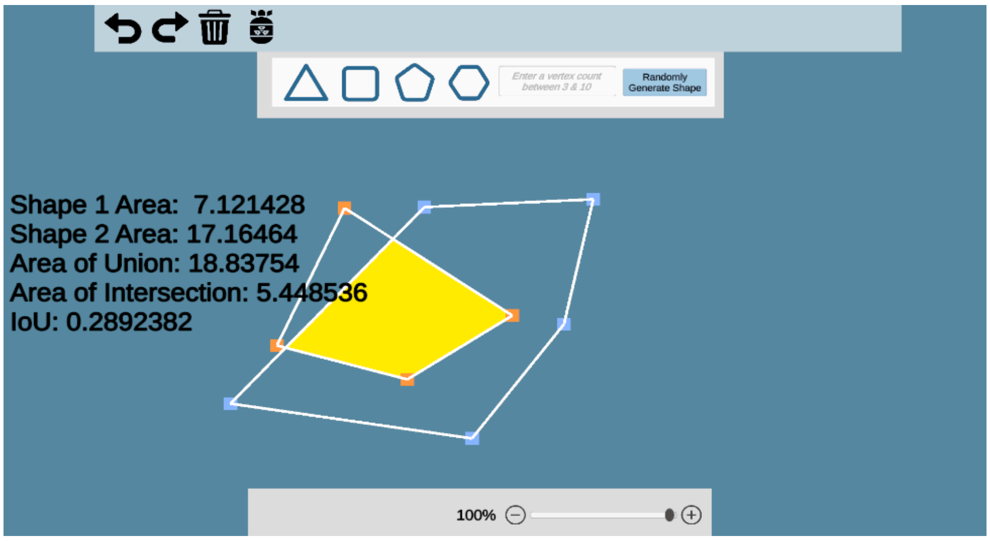
Figure 11: Shape is being drawn with Pen Tool, invalid point selected

Real-Time Information Update

When 2 shapes are drawn on the canvas, the system will display the necessary information such as:

1. Area of Each Shape, Union & Intersection
2. Intersection over Union (IoU)

- Input: Draw two shapes
- Expected Output: Real-time update of area, union, intersection and IoU
- Actual Output: Information updated in real-time



Shape 1 Area: 7.121428
Shape 2 Area: 17.16464
Area of Union: 18.83754
Area of Intersection: 5.448536
IoU: 0.2892382

Figure 12: Real Time Information Display

Point Dragging

Users can drag any vertex of the shape to modify it. The system ensures that the shape remains convex throughout this process. As a point is dragged, the IoU and shape information is updated in real-time, so users can see the impact of their changes without having to redraw the shape.

- Input: Drag any vertex
- Expected Output: Drag the selected vertex and updates information in real-time
- Actual Output: Drags the selected vertex to a new location and displays updated information in real-time

Convex While Dragging

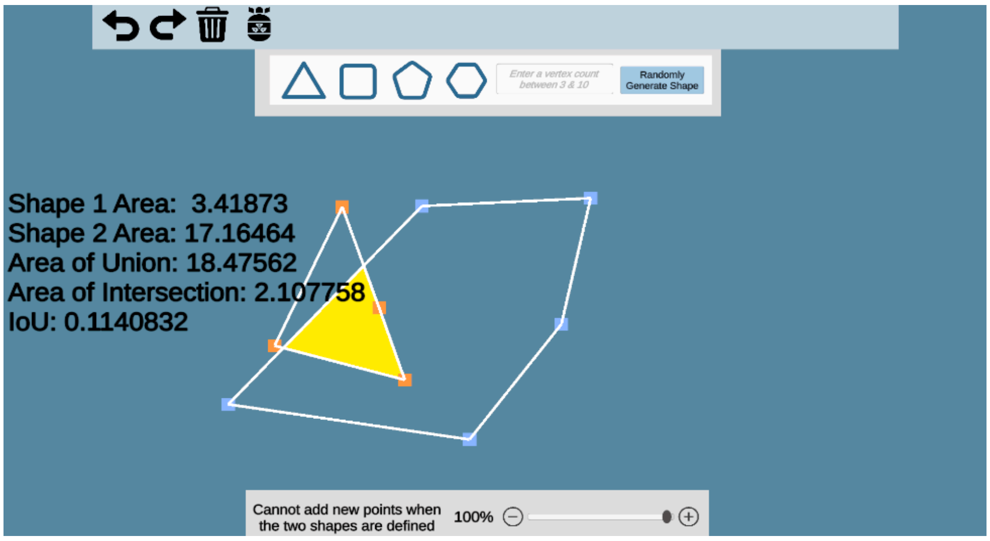
While dragging a point, if moving the vertex would cause the shape to become non-convex, the system will restrict the drag and prevent further movement in that direction.

- Input: Drag a vertex in such a way that it would make the shape concave
- Expected Output: Restrictions while dragging a vertex to maintain convex shape
- Actual Output: Drag restriction worked correctly

Shape Limit

Once two shapes have been drawn on the canvas, the system will prompt the user with a message indicating that only two shapes are allowed at a time.

- Input: Attempt to put another vertex after creating two shapes
- Expected Output: Users will not able to place another vertex and a prompt appears that tells the user only two shapes are allowed
- Actual Output: Users are not able to place another vertex and a prompt appears as expected



Shape 1 Area: 3.41873
Shape 2 Area: 17.16464
Area of Union: 18.47562
Area of Intersection: 2.107758
IoU: 0.1140832

Cannot add new points when the two shapes are defined

Figure 13: Attempting to add new shape after limit has been reached

Conclusion

All test cases related to drawing shapes and maintaining convexity were successful. The system responded as expected when users attempted to draw shapes, drag vertices and modify existing shapes. The real-time information updates were accurate and provided a seamless experience for users. Convexity was enforced during both shape creation and vertex dragging.

Integration Testing

The basic premise of intersection testing our application relied on drawing predefined shapes with the UI, allowing those shapes to calculate, and then identifying whether the shape areas and IoU value presented to the user were as expected if calculated by hand.

The test combinations used were as follows:

1. Two squares of the same size, located on top of each other
2. Two squares of the same size, located apart from each other
3. One square and one rectangle twice its size, located on top of each other
4. A triangle located within a square of the exact same width and height

All test combinations resulted as expected. This served to prove that the front-end and back-end were correctly linked through both the Unity Editor and Unity build process.

Usability Testing

Usability testing is a fundamental technique to use in user-centred interaction design that aims to evaluate the application by testing with real end users. In our case, we conducted usability with individuals, friends of ours, who had no prior exposure to our application. Providing us with valuable undiscovered insights into the design intuitiveness, user-friendliness of our design, and technical side of usability.

*****NOTE: UI Testing was completed with a design which has since been revised based on user feedback***

Testing Methodology - A/B Testing

A/B testing also known as split testing is an approach that aims to compare under 2 different versions, which are identical except one variation that might impact the user's behaviour.
We employed this method with the following 1 key variation difference and various settings .

Key Variation:

1. The presence of the user guide

2 Versions:

1. GroupA: Individuals in this group were tested **with** user-guide provided.
2. GroupB: Individuals in this group were tested **without** user-guide provided**.**

Testing Conditions and Settings :

1. All usability testing was conducted in person on campus.
2. Both groups were all under the condition of "No prior exposure to our application".
3. We gave all the users 10 to 15 mins to play around with the application without any interruption. We observed their reactions to identify any usability issues, and a 10 to 15 mins Q&A based on the reaction each participant gave on the testing day.
4. In each group, there existed 1 Mac User and 2 Windows User ratio for compatibility purposes.

Number of participants:

1. Each group consists of 3 users.
2. Total number of 6 participants.

Main testing difference:

1. The sole variable between the 2 groups was the **existence of the user guide**. Which allowed us to assess the impact of documentation on user experience.

Outcome of testing

GroupA (With the guide)

1. **Ease of Navigation:**
   The participants who were given the guide were all navigate the application without any significant issue, due to the completeness of the user guide.

   Participant Quote:
   - “The user guide helps a lot [!]” - from all participants in GroupA
   - “It was clear to me that Nuke Icon does mean delete everything” - from 1 participant in GroupA
   - “It’s easier to navigate the app than new moodle” - from 1 participant in GroupA

➔ This made it obvious to us that the amount of difference a completed user guide can provide is significant to UX, helping us to understand the crucial role of a completed guide.

2. **Smooth Installation Process:**
   Participants found the installation process was straightforward and easy to set up with the presence of a user guide.

   Participant Quote:
   - “Installation [was] easy and no issue” - from all participants.

➔ This indicates the installation process is easy to follow for everyone as long as the user guide is provided with. Hence, another confirmation on the vital role of user guide.

GroupB (Without the guide)

1. **Unclear Functionality of The Nuke Icon Button:**
   The initial participants' responses from GroupB were mostly identical in regards to the confusion on the purpose of the "nuke" icon button.

   Participant Quote:
   - "What does the nuke/bomb-like button do ?" - from all participants in GroupA

   - "I thought [the nuke button] might Alt + F4 the app or do something weird" - from one of the participants.

➔ This indicates, although we as the developers of the application thought the nuke icon is obvious for people to associate with "Delete everything" as Nuke generally refers to "destroy everything" at least this was agreed upon by us. But it was an assumption that people might not resonate with which caused hesitation on trying or pressing. Hence, a clearer icon choice or hover-over label on icons are needed to provide a better UX.

2. **Unaware of Draggable Shapes:**
   1 user was not able to realise that the defined shapes and generated shapes are all draggable. Other 2 users were able to realise the draggable function but through random clicking and dragging, moving the shapes by accident.

   The shape can be dragged through the vertex point of the shape as shown in the red circle (this was not told to groupB during testing):

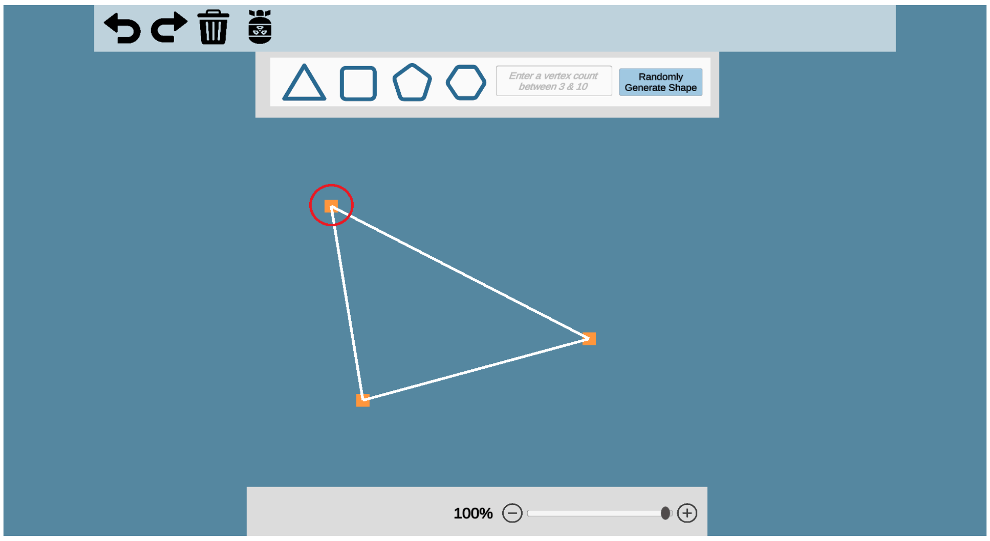


Figure 14: Example of a vertex to drag

   Participant Quote:
   - "Damn I didn't realise the shapes are movable until I [accidently] clicked and moved the shape on the [vertex] point"

➔ This showed that It was not obvious to the individual that the shapes being generated were draggable due to lack of information on the UI. A need for clearer explanation for interactive elements is also needed.

Common Responses and Observation from Both Groups

1. **UI Design:**
   Participants from both groups expressed the common feedback on the cleaness, layout, and intuitiveness of the application overall. Common symbols like undo, redo, ,trashcan, and zoom in / out were appreciated for their clarity.

   <u>Participant Quote:</u>
   - "Can immediately tell the what all the buttons do except the nuke icon button" - from all participants in groupB

   - "The UI is very simple yet effective and not over complex" - from 2 other participants from both groups

2. **Cross-Platform Compatibility:**
   Participants never ran into any issue from both OS, Mac and Windows. This was never brought up from participants but through our observations. It was nicely integrated in both platforms without any sign of lagging and crashing.

3. **Effective User Feedback:**
   Participants expressed the responsiveness and the clear feedback provided during interaction, such as confirmation messages and visual indicators.

   <u>Participant Quote:</u>
   - "The feedback helps me to understand what my action is doing" - from 2 participants in groupB

   - "It feels good to have an app provides feedback while some don't" - from 1 participant in groupA

Conclusion

Our usability testing revealed plenty of undiscovered insights into what the issues end users may run into while the developers, us, weren't aware of this during the development cycle. Especially with the "nuke" icon and unaware that the shapes are draggable, indicating a need for clearer visual cues. Meanwhile, it also revealed that the user guide significantly enhanced the experience for those who were given groupA, highlighting the importance of accessible and clear documentation.

Limitations of the Testing Process

At the current moment no automated UI tests are being run by our test suite. All UI tests have instead been relegated to manual testing by both end-users and our development team. This results in great time loss as the tester has to wait for the application to build, and then complete individual tests themselves rather than being able to build and batch the tests to an automated system.

The Unity Testing Framework does in fact support automated UI testing, however due to development constraints we shall not be implementing this functionality at the current time. Given an increased scope and development time, however, this would be one of the first considerations.

As none of our team is heavily mathematically inclined, we are not able to identify geometric edge cases that experts in that specific field may be able to. As such our tests are designed around fairly basic cases, solely to test whether our functions work- rather than having higher-level ambitions of testing the speed and efficiency of our algorithms with complex inputs.

Our application does work at a usable speed, however this is due to certain artificial limitations we have imposed (the most notable being the limit of random shape generation to 10 vertices). Given an updated testing framework focused on also analysing functional efficiency and speed, we may be able to replace these functions with faster counterparts to provide the end-user more freedom in random generation.

Finally, usability testing was only able to be completed on a small sample size (our development team, alongside a small group of technologically inclined individuals. If we had more resources this testing would be broadened to a wider variety of individuals, including some who do not regularly use complex GUI applications on desktop/laptop machines. This would help us to identify further revisions to be made in order to ease the clarity of the User-Interface.