SOFTWARE DESIGN SPECIFICATION

CONTENTS

1. INTRODUCTION

1.1 OVERVIEW – PURPOSE OF SYSTEM

The purpose of our project is to develop at least three puzzle games for the Microsoft HoloLens. Each interactive puzzle will vary in complexity and design. Our proposed puzzle games are Jenga, Tower of Hanoi, and Chess. The games should be entertaining, responsive, and accessible to all users.

1.2 SCOPE

The scope of our project is to deliver three puzzle games written in C#. Our application has been constructed to allow for the easy addition of new games. There will be a puzzle selection screen, allowing the user to choose which game they would like to play. Our system will anchor the selected puzzle to a flat surface of the user's choice. Once an appropriate surface is recognized, the puzzle will appear, and the user can begin to play. Some puzzles will more interactive than others with varying levels of difficulty. Due to our lack of experience, we opted to familiarise ourselves with the coding environment by developing a puzzle game each. While this had been working successfully, five puzzle games were beyond the scope of our project. Instead, we decided to implement paired programming and attempt to make one game multiplayer if possible, within the time constraints. However, this may be unlikely.

1.3 DEFINITIONS, ABBREVIATIONS

Augmented/mixed reality: The enhancement of a real-world environment through the use of computer-generated imagery.

HoloLens: Microsoft's augmented reality headset.

HoloLens 1: 1st generation Microsoft augmented reality headset

HoloLens 2: 2nd generation Microsoft augmented reality headset

Unity: A software development environment.

Mixed reality toolkit (MRTK): A Unity library for mixed reality development.

1.4 REFERENCES

https://github.com/microsoft/MixedRealityToolkit-Unity

https://www.microsoft.com/en-us/hololens

https://unity.com/how-to/programming-unity

2. SYSTEM DESIGN

2.1 DESIGN OVERVIEW

Our design, as specified by the client, will consist of three puzzle games to be displayed as hologram through Microsoft's HoloLens 1. The three games will be selectable by a menu screen and will be interactive for the user. They will use the HoloLens' hand recognition and its ability to anchor to flat surfaces such as walls and tables

2.1.1 HIGH-LEVEL OVERVIEW

High-level overview of how the system is implemented, what tools, frameworks and languages are used etc.

Firstly, we are using Git for version control with a repository set up on GitHub.com and all the team members set up as collaborators.

Our puzzle games are being produced in Unity. This is a game engine which allows for development in virtual and augmented reality. There are several reasons as to why we chose to develop our application in unity, examples of which are as followed

- It was recommended by our client, as it is compatible with the HoloLens and previous projects.
- It is easy to use in combination with other text editors such as visual studio code.
- Very good for rendering 3D images

We are using Microsoft's Mixed Reality Toolkit to work in augmented reality. Such libraries from the toolkit which are vital to our implementation are spatial mapping and hand recognition. Spatial mapping allows us to anchor our games to tabletops and walls. While hand recognition enables the function to interact with objects in our games.

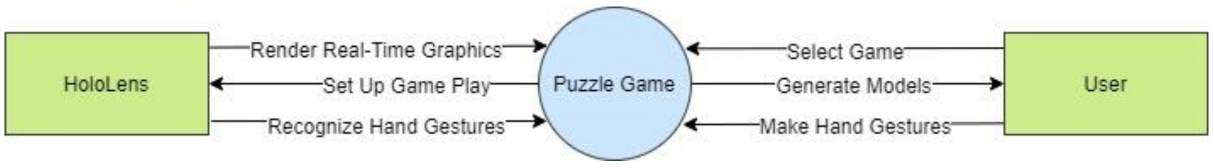
The scripts we are writing to add function to our games are written in C# which is a programming language often used in game development. Our C# scripts are written in the Visual Studios, the IDE we opted to use for our development.

Coinciding with Unity we are building unique objects for our games in Blender. This is a free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, and computer games. It is optimal for our development as it easy to export projects into unity and unity caters for these imports.

After development completion in Unity, scenes are deployed into an emulator. The two emulators we are using is the HoloLens emulator and the MRTK emulator. Finally, we will meet Microsoft to deploy our end product through the HoloLens.

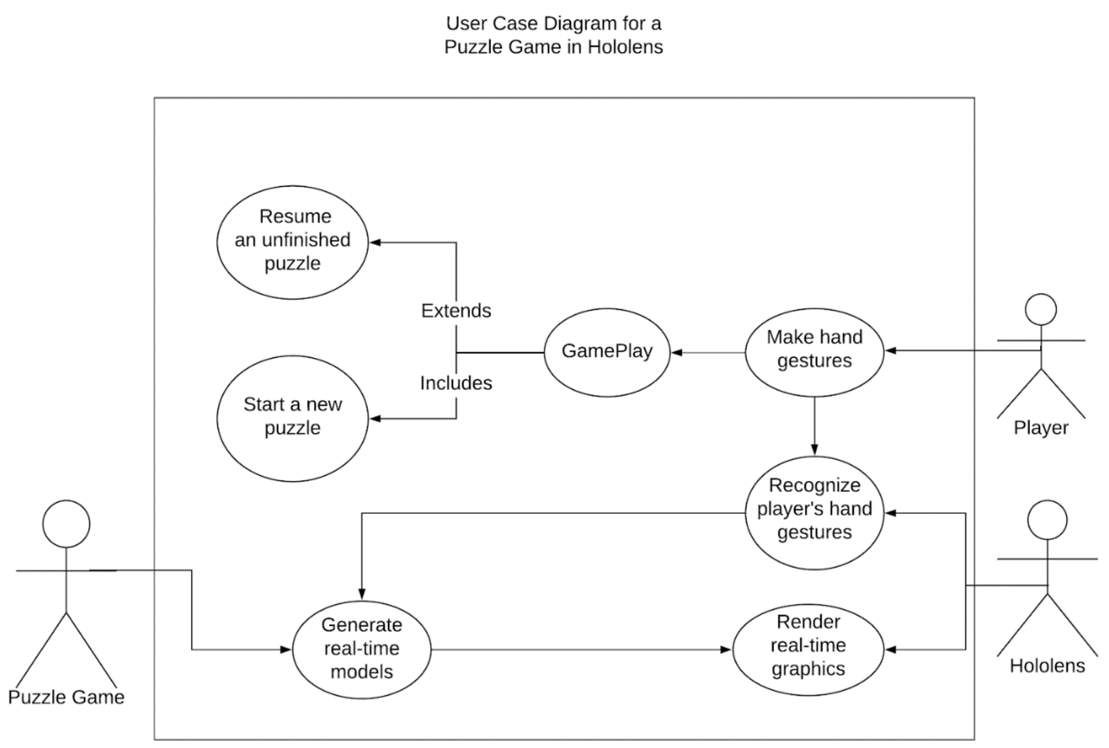
2.2 SYSTEM DESIGN MODELS

2.2.1 SYSTEM CONTEXT

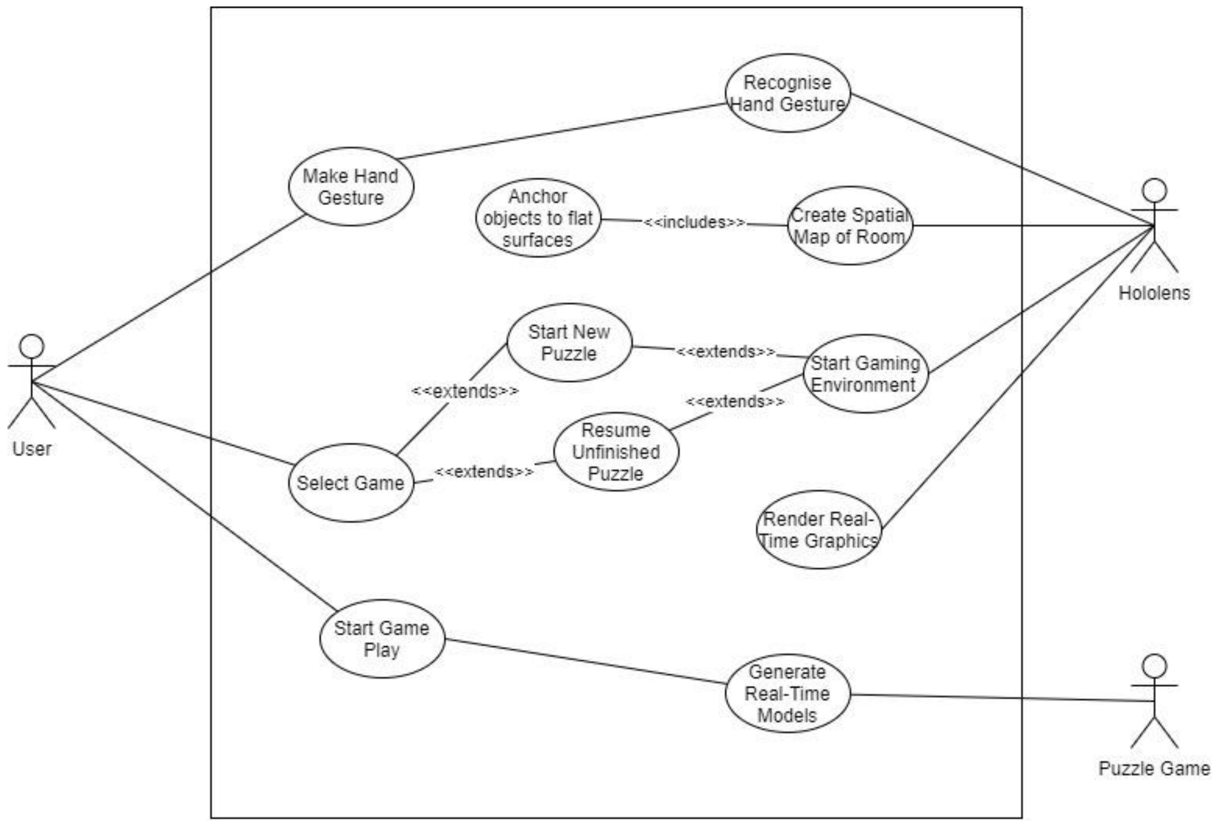


The above diagram outlines how the puzzle game interacts with the HoloLens and the user. It provides a high-level view of the system. As can be seen above, the user selects the puzzle game and make hand gestures, in response the puzzle game will generate the corresponding models. The HoloLens will then render real-time graphics and recognize the hand gestures, while the puzzle game will set up game play.

2.2.2 USE CASES


User Case Diagram for a Puzzle Game in Hololens
Resume an unfinished puzzle
Extends
GamePlay
Make hand gestures
Player
Includes
Start a new puzzle
Recognize player's hand gestures
Hololens
Puzzle Game
Generate real-time models
Render real-time graphics

This was our original use case. When we presented our initial requirements document, we worked from this diagram and developed our prototype from there. However, as we got further into the development process, we realised this no longer fit our design and we had to adjust our diagram accordingly. The following use case better represents the flow of our application.

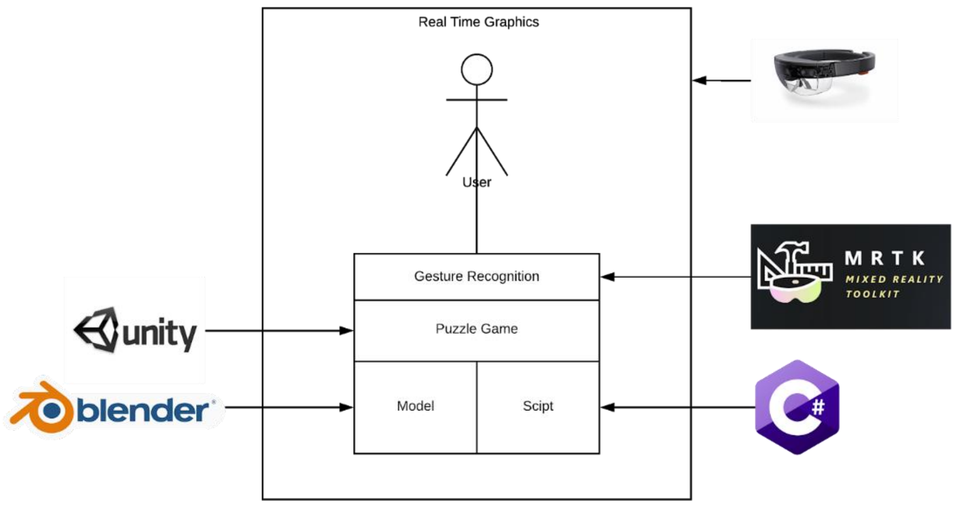


TEXT DESCRIPTIONS

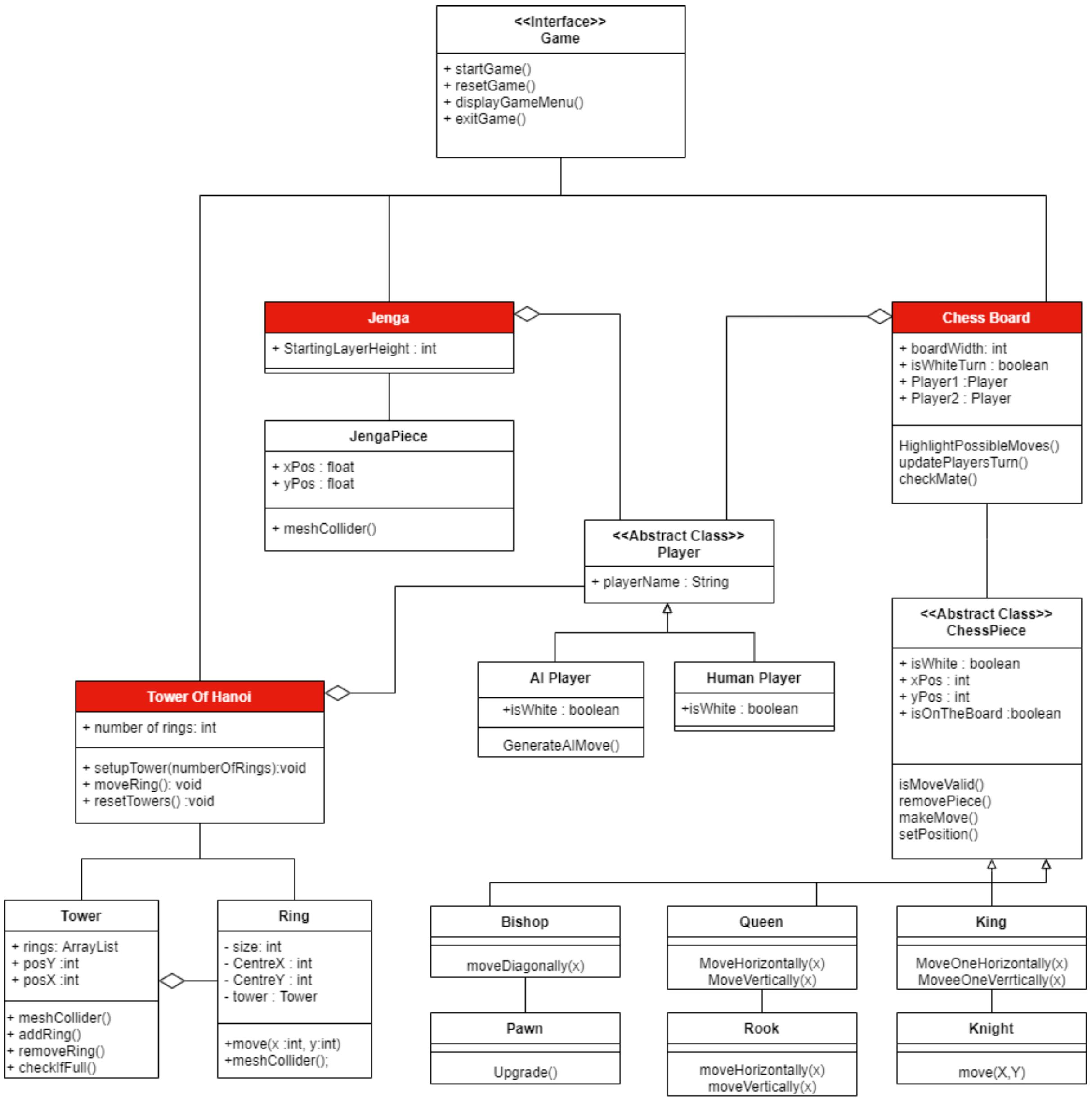
Name: Make Hand Gesture **Participating actors:** User **Entry condition:** User is correctly set up with the HoloLens **Exit condition:** User has successfully made a recognized hand gesture **Normal scenario:** **1.** User starts up the HoloLens **2.** User makes the desired hand gesture **3.** HoloLens accepts hand gesture and directs user to their desired result **Error scenario:** User makes invalid hand gesture	**Name:** Recognize Hand Gesture **Participating actors:** HoloLens **Entry condition:** HoloLens is correctly set up **Exit condition:** HoloLens recognizes valid hand gesture **Normal scenario:** **1.** User starts up the HoloLens **2.** User makes the desired hand gesture **3.** HoloLens recognizes and accepts hand gesture **Error scenario:** HoloLens does not recognize or accept the hand gesture	**Name:** Create Spatial Map of Room **Participating actors:** HoloLens **Entry condition:** User is correctly set up with the HoloLens and in an acceptable location **Exit condition:** HoloLens successfully mapped the area and anchored objects to flat surfaces **Normal scenario:** **1.** User starts up the HoloLens **2.** HoloLens scans the room, creating a mesh **3.** HoloLens finds an acceptable surface to anchor objects **Error scenario:** No valid surfaces are identified, user is in an unacceptable area
Name: Select Game **Participating actors:** User **Entry condition:** User is correctly set up with the HoloLens **Exit condition:** User has successfully chosen a game (either start from the beginning or resume a puzzle) **Normal scenario:** **1.** User selects the game they want to play **2.** Option pops up to allow user to continue a puzzle (if available) or start a new game **3.** User selects desired option **Error scenario:** option does not pop up to give user a choice, user does not select a game, user selects incorrect option	**Name:** Start Gaming Environment **Participating actors:** HoloLens **Entry condition:** User is correctly set up with the HoloLens and selected a game **Exit condition:** Game set up in accordance with user's selection **Normal scenario:** **1.** User selects desired game **2.** HoloLens sets up the correct gaming environment **Error scenario:** Incorrect gaming environment displayed	**Name:** Render real-time graphics **Participating actors:** HoloLens **Entry condition:** User is correctly set up with the HoloLens **Exit condition:** correct real-time graphics rendered **Normal scenario:** **1.** HoloLens sets up the correct gaming environment **2.** Accurate real-time graphics displayed throughout user experience **Error scenario:** No (or not all) graphics displayed
Name: Start Game Play **Participating actors:** User **Entry condition:** User is correctly set up with the HoloLens and selects to start the game **Exit condition:** User is playing the game **Normal scenario:** **1.** User selects their desired game **2.** HoloLens starts the gaming environment and renders real-time graphics **3.** User selects start game **Error scenario:** the game does not start	**Name:** Generate real-time models **Participating actors:** Puzzle game **Entry condition:** User is correctly set up with the HoloLens **Exit condition:** The correct models are displayed in accordance with game rules **Normal scenario:** **1.** User selects desired game **2.** HoloLens starts the gaming environment **3.** Puzzle game generates accurate real-time models and allows user to begin playing **Error scenario:** Incorrect models displayed or none at all	

2.2.3 SYSTEM ARCHITECTURE

The system architecture diagram outlines how our project has been made. We used Unity to develop our puzzle games while writing our scripts in C# to control the rules of the game. In addition, we are using Blender to build unique objects for our puzzle games. Finally, we use Microsoft's MRTK to recognize the hand gestures made by the users, allowing for seamless game play.

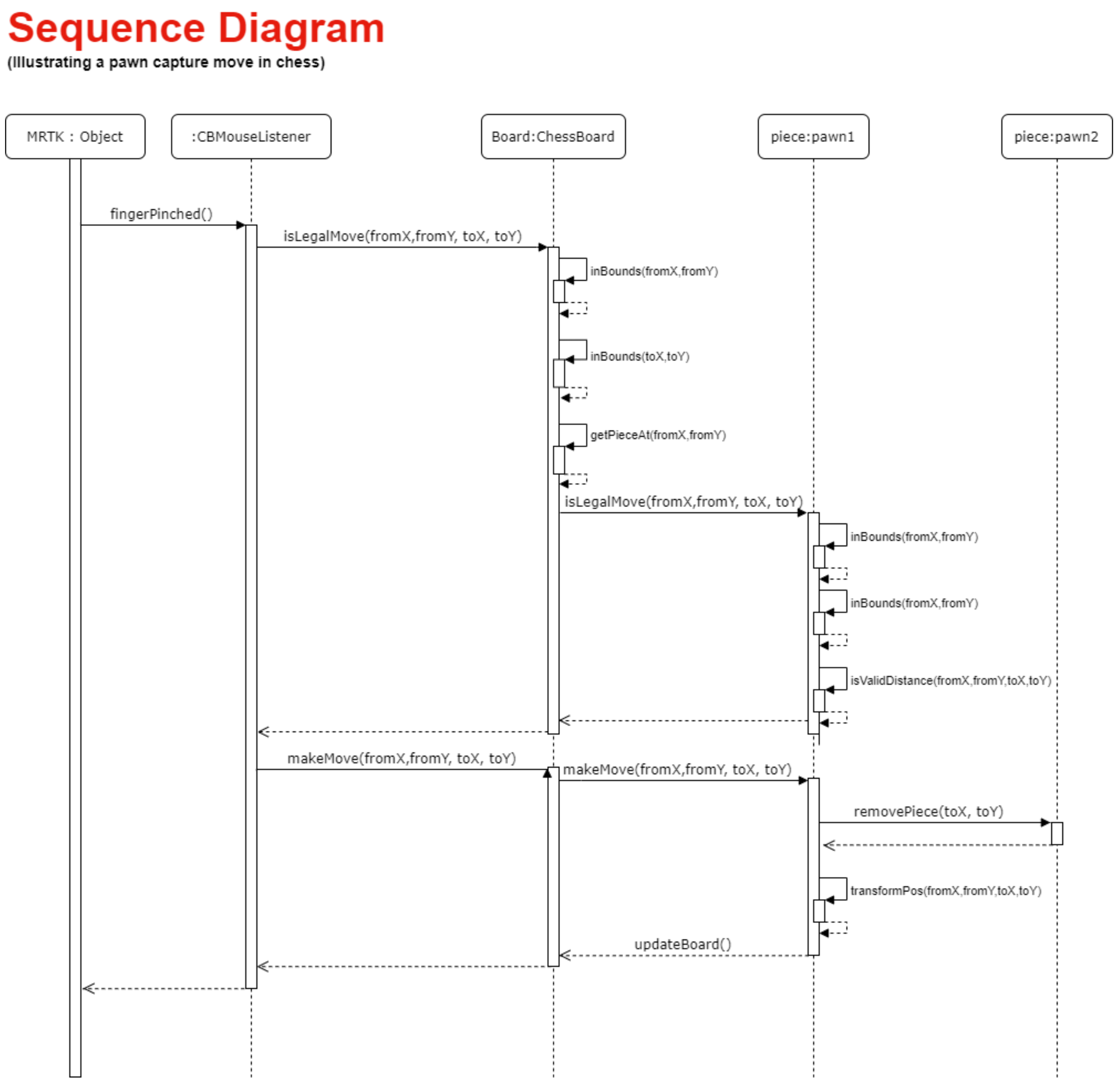


2.2.4 CLASS DIAGRAMS



The above diagram outlines how each aspect of our project interact with each other.

2.2.5 SEQUENCE DIAGRAMS

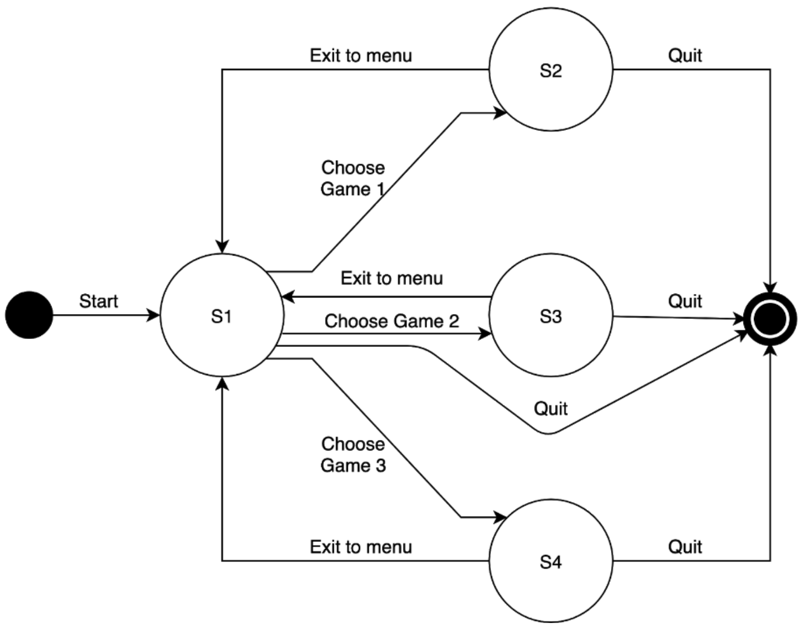

Sequence Diagram
(Illustrating a pawn capture move in chess)
MRTK : Object
:CBMouseListener
Board:ChessBoard
piece:pawn1
piece:pawn2
fingerPinched()
isLegalMove(fromX,fromY, toX, toY)
inBounds(fromX,fromY)
inBounds(toX,toY)
getPieceAt(fromX,fromY)
isLegalMove(fromX,fromY, toX, toY)
inBounds(fromX,fromY)
inBounds(fromX,fromY)
isValidDistance(fromX,fromY,toX,toY)
makeMove(fromX,fromY, toX, toY)
makeMove(fromX,fromY, toX, toY)
removePiece(toX, toY)
transformPos(fromX,fromY,toX,toY)
updateBoard()

The above diagram illustrates the happy case of a pawn capture move in our chess game. When the user makes the pinching hand gesture to pick up and move a piece, our application will ensure it is a valid move. It checks the coordinates of both the current location and the desired location, i.e. is the location on the board, and can that specific piece move there. If it is legal, then the application allows the pawn to move from their current location to the desired location and updates the board accordingly.

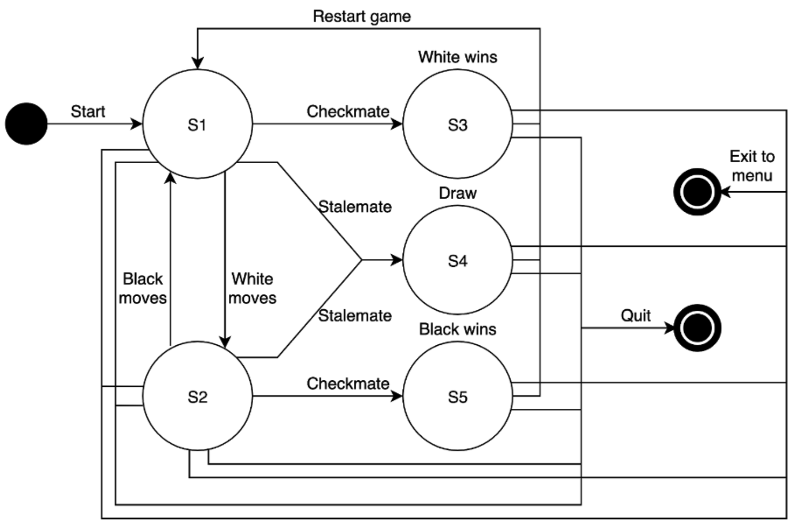
2.2.6 STATE DIAGRAMS

The following diagrams describe the behaviour of the main system as well as the chess game.

Main

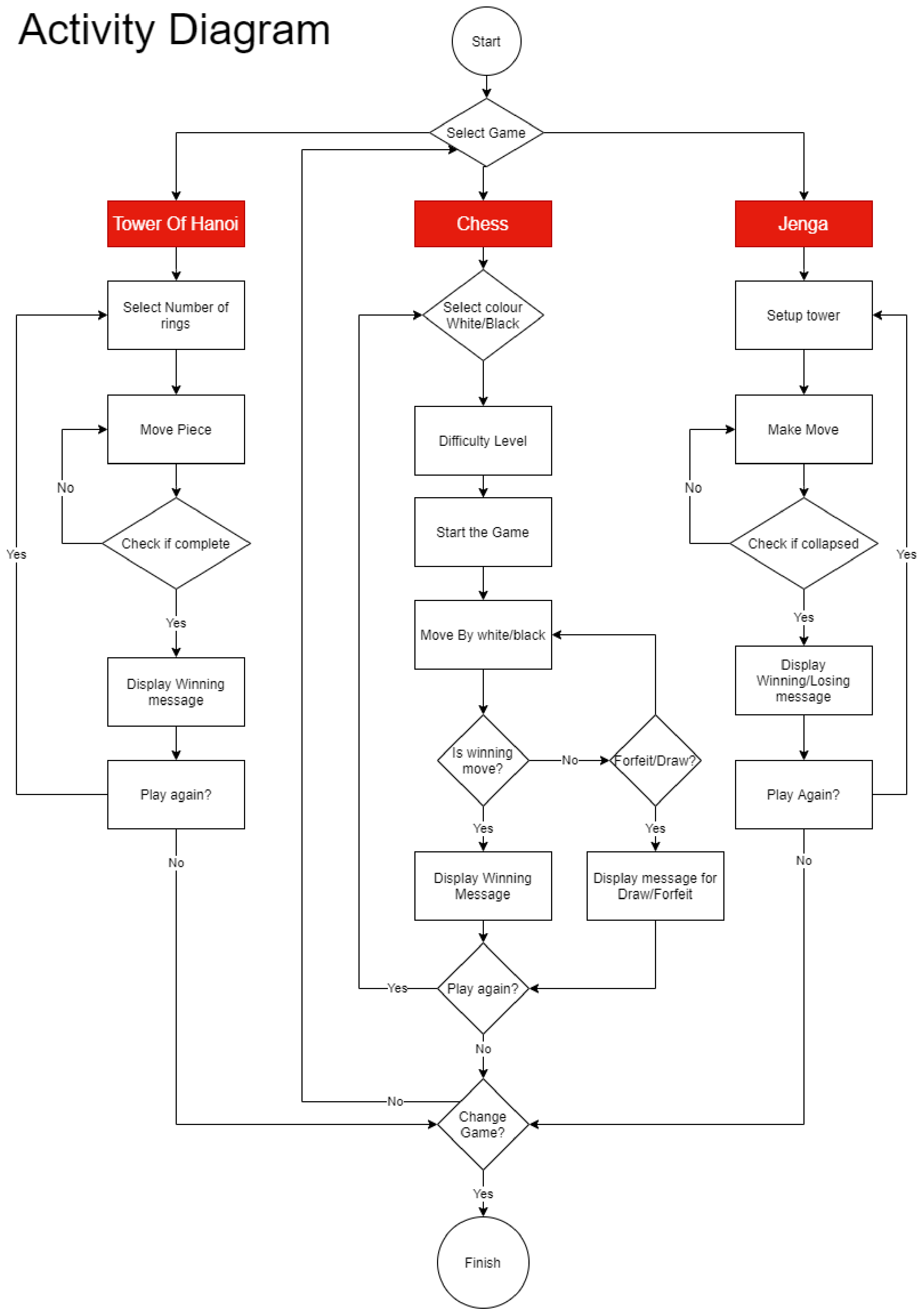


CHESS



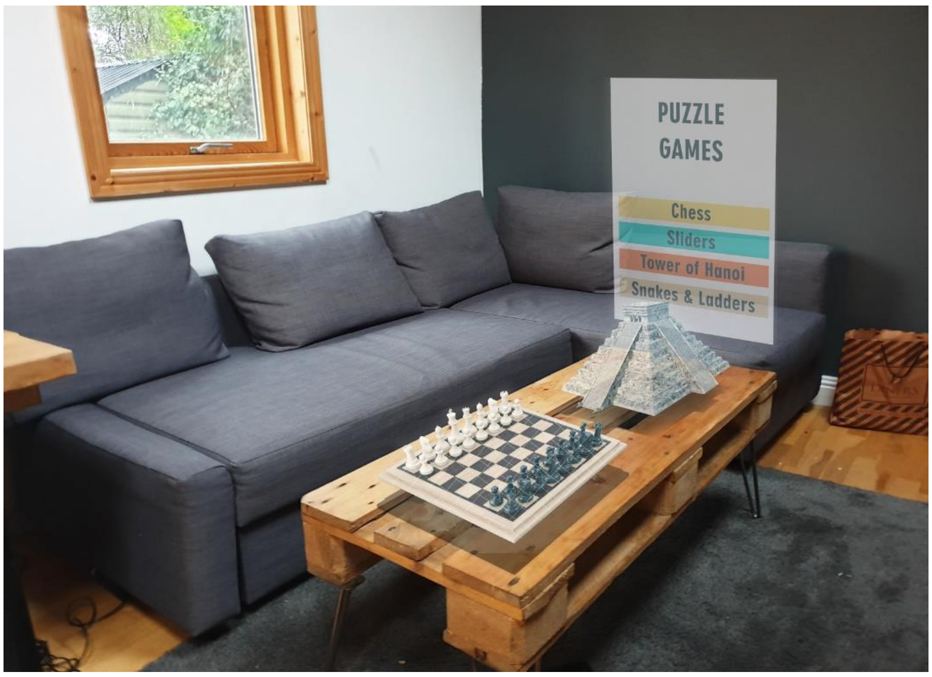
2.2.7 OTHER RELEVANT MODELS

ACTIVITY DIAGRAM



The above activity diagram depicts the behaviour of our system. First the user must select the game they want to play. If for example, the user decides to play Tower of Hanoi, they can choose their difficulty level by selecting the number of rings they'd like to play with. Each time they move a piece, the application will check if the game has been won. If the user wins, a winning message is displayed to congratulate the user, then asking if they want to play again. If yes, they can change difficulty, if no, they have the option to choose another game or exit the application all together.

UI MOCK-UP



This mock-up shows a standard living room environment with a table and couch. Our application will map out the room using spatial mapping and will register where the table is. It will then place our boardgames on top of this table to ensure easy use of the games for the user. The game menu is shown in the background, on a wall, making it easy to change game or to start a new game.