**Overview**

The application is a prototype of a Microsoft HoloLens augmented reality play space with the purpose of facilitating interaction and play between its users. It aims to provide an augmented reality play space which will allow autistic users to play and interact more easily with others. Children with autism can find it hard to interact and socialise with their peers, our augmented reality play space aims to provide an environment which allows users to more easily overcome these challenges.

The application must be suitable for our target audience and have a clear targeted problem to solve. The play space needs to incorporate augmented reality technology available with the Microsoft HoloLens using Unity as a development environment. The prototype will need to run on an Android mobile device and providing similar functionality to the HoloLens to adequately demonstrate our play space.

**System**

We will be producing an AR assistive technology game application aimed at primary school children with Autism, to help improve their social skills. Our end goal is a fun, helpful and educational app which will significantly benefit young people with autism. Using an emerging and growing technology like Augmented Reality, will provide a fun and educational experience, augmented reality is familiar to the youth with apps like Pokémon Go, Nintendo 3D’s AR Games and even Snapchat’s filters are of a form of AR.

The app will be developed with the Unity 3D game engine, with help from AR libraries such as the open source AR Toolkit which offers a free plugin for Unity, Vuforia Augmented Reality SDK and documentation and libraries from Google’s Android AR Core. Each type of ‘social skill’ is split in its level and section. We are taking inspiration from many popular brain training applications. Which helps improve things like ‘memory’ by such challenging the player to memorise a certain pattern, which will disappear after a certain time, and will ask the player to input the previous pattern, challenging the player’s memory, and attention. Our application will take great inspiration and do things like this.

For example, we can test the player’s eye contact by using the player’s phone camera, detecting the facial movement, and make them look at an AR avatar for a certain period, and we could mark them down if they look away for too long, or even stare at the avatar too long. Moreover, we can make it more intensive and expand on the game, by adding distractions, noise, time limits, ratings to add more replay value, fun and challenge to provide a fun and educational experience. We can greatly build upon and produce fun educational for all types of social scenarios.

We can also delve further, using the player data by generating statistics from it and feeding this information back to the teacher who could help greatly in the social development of the player and even gather more research into Autism.

We are developing an AR educational game for young people with autism that helps develop their social skills in a fun and even challenging way.

**Key risks**

**Deadlines**  
Missing agreed, or already defined deadlines could cause elements of the project to slip and create a backlog further down the line. Missing key milestones will likely mean that planned functionality will have to be removed closer to the final deadline to ensure working software can be delivered.

**Underestimating Deliverables**  
Projecting that a planned deliverable will take a lot less time than it takes, particularly if it is a high priority feature, will mean less time to work on other areas of the software, reducing the overall quality.

**Commitment & Effort**  
Team members putting in less effort than required to work on their assigned jobs will either mean incomplete or low-quality features being implemented, or less work being done developing other features to bring this work to a reasonable standard.

**Poor Testing**   
If the testing strategy is poor, the end-product will have a significantly higher number of bugs. Testing will need to be consistent and thorough across all areas of the application. Similarly, not allocating enough time to the testing side of things will result in a big drop in overall quality.

**Poor Implementation**  
Implementing features in a poor or inefficient way can cause the software to be of low quality. The code also needs to be clear and maintainable, or too much time will be spent on trying to interpret code rather than adding to or fixing it.

**Not Understanding the Tools**  
The team will have to learn the tools being used for the project (e.g. Unity, GitHub) or risk the final product being hastily put together poorly. Avoiding this will mean individual team members learning the chosen tools and understanding Unity before the development work begins.

**Project Management**

**People being told unexpectedly at the end that their work is rubbish:**  
By ensuring that each member of the team has their progress frequently peer-reviewed and monitored using periodic stand-up meetings and pair programming. This will allow each member to be provided with supportive feedback and constructive criticism on the task that they are assigned to.

**People not having enough/having too much to do or not being involved:**  
By employing the use of a Kanban board that can show the level of work being assigned to each member. This allows for the entire team to be aware of what each other is expected to do. We will also have some time in each of our meetings for any issues with the workload to be discussed with each other if any members of the team feel they are doing too much or too little.

**The group falling behind without noticing:**  
We will have regular meetings where we will talk about each other’s progress and what we need to do to be more efficient in terms of time taken to do specific tasks. If we fall behind in a task due to difficulty, we will assign another group member to the same task to speed the process up. We will use strict deadlines to ensure all work is done in time for us to complete the project and if there are problems with tasks not being completed, we will have a record of what has not been done, and of who was assigned to it. Team members must learn basic features of the software that we will use during the project (Unity, Git Hub etc).

**Everyone interpreting an agreement differently:**  
All agreements made about how we will design and implement the project will be written in the most detailed way. These guidelines must be followed exactly to prevent problems. If group members have questions about an agreement previously made, they can use one of our meetings our Microsoft teams to discuss with the rest of the group.

**Not having a demonstrable system by the deadline:**  
Features implemented must be tested thoroughly by more than one member of the team. When implementing a major part, we will save the previous working version in case the added data breaks the program. This way if we choose to add something complex before the deadline or we have fallen behind due to the difficulty of implementation, we have a working application to fall back on for demonstration.

**No one knowing where the latest version of the software is:**  
To ensure that everyone knows where the latest version of the software is, we will be using GitHub as a repository for all our work. By doing this we can see the current version of the software clearly and quickly and be able to edit it at any point, it also allows us to see who has recently made a change known as a ‘commit’ and what they have changed. This will mean that everyone will always have access to the latest version of the program and if there are any issues with an update. It also allows us to quickly revert to a previous working version so that we do not lose any work.

**A disk crash destroying all copies of the software:**  
To ensure that a disk crash does not destroy all copies of the software we will be keeping an extensive record of all versions of the software on GitHub. This means that if we have any personal issues with drive crashes, we will only lose work that hasn’t been uploaded, to try and minimize this we will be uploaded all finished work at the end of the day to ensure we have backups on GitHub for anyone to access in case they lose their copies. Between the online version and personal copies, it will be doubtful that we will lose any work due to driving crashes, we can trust that the online version on GitHub will have no issues due to it being a very reputable and professionally used site.

**Someone changing the interface between components without agreement:**If someone changes the interface between components without the group's agreement, we will discuss it during our team meetings and ask the person why. If they do not have a good reason, we will look at the change and potentially roll it back to a previous version and re-implement it as a team if the consensus is that it is a good change. If this was to keep happening, we could look at limiting that person’s access to edit the current version on GitHub directly. Strict version control would ensure that everyone knows what each aspect of the program does. If someone is changing things without approval, it could end up causing many problems down the line when new changes end up conflicting with old ones.