An Educational game that helps medical students to learn faster about various diseases in a fun and engaging way

Disease Diagnosis

A Serious Game to Diagnose Diseases

GitLab: https://cseegit.essex.ac.uk/21-22-ce901-su/21-22_CE901-SU_palamthodi_ajaykiran

Jira: https://cseejira.essex.ac.uk/browse/B901105-1

Ajay Kiran P
Dr. Aikaterini Bourazeri
30th August 2022

ABSTRACT

Over the past decade, usage of serious games for education has risen. People seem to be able to learn and remember better when they engage themselves with them rather than just reading a book or listening to a lecture. In this research project I have built a game where medical students would be able to engage themselves and try to diagnose different diseases. The main objective of the game is to help them improve their knowledge about various diseases, their symptoms, and how to treat them in a fun and engaging way. In the game the players take up the role of a doctor and try to diagnose a patient by collecting clues about the symptoms and what tests could be done to figure out the disease. Once this is done the player provides their diagnosis and prescribe the treatment, all in a risk-free environment where mistakes are not fatal. The player can try as much as they want to figure out the disease they are up against. Unlike to real life there is no actual risk, and the students can work in low pressure environment and enjoy the process of learning. After the game was developed a study was conducted with 17 participants who played the game. Most of the participants found the game to be engaging and fun. All the participant agreed the game would be helpful for students in medical school. The reviews were mostly positive; some suggestions were also offered. The project was made better taking into consideration these suggestions.

TABLE OF CONTENTS

INTRODUCTION

Video game industry is bigger than both the global box office and the music industry combined. It is highly impressive how massive this industry is and how it has evolved over the last decade. Games can be used for educational purposes and these kinds of games are called serious games. The raise of serious games can be traced back in history even as far as the 1700s. In the 1700s there was something called Craig Spell which is the German word for Wargames and in this board game which had units on each side was used to train soldiers strategies of warfare. Serious games have along and rich history.

The 1980's saw the breakthrough of 2 very popular serious games.

[Where in the World is Carmen Sandiego?] - This is a series of several video Edu-games that teach geography. The game is about tracking Carmen's Villains around the world and arrest them. The player begins by going to the place where the crime took place and collecting hints about where the thief would go next. This leads to a chase around the world to find the thief before the time runs out [1].

[Oregon Trail] – In this game the player controls a group journeying down the Oregon Trail from independence, Missouri Willamette Valley, Oregon, in 1948. The player can choose a profession from banker, farmer or carpenter which has certain advantages and disadvantages. The game was designed to teach children history as they went through the process [2].

Our education system, even in organizations, companies and business has typically been a linear model, but games can give us non-linear experiences. We don't have to always go in order, we have room to think about strategies differently, we have the freedom to fail etc., and all these are benefits of serious games.

Positive Impact of Video Games

Social Activity – Social games involve playing with others and these can be cooperative or competitive [3]. The social activity would involve interacting with each other or with a Non-Playable Character (NPC). Video games involving these have shown to be positive for the psychological aspects of life. An online survey suggests that player who played MMORPGs in moderation, had significantly low levels of psychological symptoms like stress, depression, and anxiety [4].

Communication – Most cooperative games require player to work in parties of 5 or more to bring down a raid boss or clear dungeons. This requires and motivates players to have excellent communication skills as a minor mistake by any one member would mean that the whole party has to retry the level again. This forces people with social anxiety to step up and face their fears all in a fun and positive environment.

Prosocial Behaviour – A study conducted at University College Dublin shows a positive significant relationship between prosocial video game use and the dependent variables: cooperation and sharing, the tendency to maintain positive and affective relationships and empathy [5].

Physical impact – A Research conducted by Doctor Renjie Li and colleagues from the University of Rochester in New York and Tel Aviv University in Israel says that a

person's night-time vision gets better after playing action games. The observational study looked at contract sensitivity function (CSF) an element of vision which is the ability to see things that do not stand out from the background. Video games helps in improving vision, such as the ability to track fast moving objects or several objects at the same time [6].

Exergames – These are the types of games that requires the player to physically interact with the game to control a character on the screen. These are often played on consoles that have a sensor to detect movement like Wii or XBOX Kinect. Just Dance and Wii Fit are some examples of such games. Studies show that engaging in an exergame regime significantly improves balance, flexibility, braking force, lower limb muscle strength, maximal oxygen levels, and heart rate [7].

History of Serious Games

Serious games can be looked on as games which can convey a message or educate the player rather than purely entertain them [8]. Clark Act was the first author who used this term in 1970. To him, serious games were effective teaching and training methods that students of all ages can use in many situations because they are highly motivating, and they communicate very efficiently the concepts and facts of many subjects. Some examples of serious games involve advertising, education, training, and simulation. Serious games are the perfect tools for any environment and getting the feel of almost actually being in there and doing the work, without having any aftermaths or consequences. Serious games can help students learn in a risk-free environment where mistakes are not fatal, and they can replay as many times as they want without any serious repercussions.

Michael et al. (2009) divides the history of serious games into 4 periods [8]. First with the arrival of learning machines and Pressey's Drum Tutor in 1924 the learner became responsible for their own learning. With the MIT Whirlwind Project, simulation was introduced in 1946. This allowed military airline pilots to be trained in a controlled environment. Using a trial-and-error method learning was achieved. Democratization of games made the simulations available to the public. Since early 2000s simulation games have gradually become professional. Some of the milestones in the history of serious games is shown in [Table 1].

Year	Serious game	Application
1970	Serious Games book by C. Abt	Academic book
1972	Magnavox Odyssey	Education
1973	The Oregon Trail	Education
1980	BattleZone	Training
1981	The Bradley Trainer	Training
1982/1983	Pole Position/Atari VCS 2600 console	Training
1996	Marine Doom	Military
2002	America's Army	Military
2003	DARWARS	Military
2005	VBS1	Military
2006	BiLAT	Interpersonal communication
2009	VBS2/Game After Ambush	Military
2012	X-Plane 10	Training

Table 1: Milestones in the history of serious games.

Features of a Serious Game

Literature about serious games and its effect on behaviour and attitudes of the player always highlights interrelated elements of immersion, identity, agency of control, interactivity, narrative, challenge, and feedback (Hays, 2005; h ompson et al., 2008; Bryant & Fondren, 2009; Klimmt, 2009; Ritterfeld et al., 2009; Annetta, 2010; Charsky, 2010) [9].

Immersion – This refers to player's sense of presence in the game. As per Tamborini and Skalski (2006) there are 3 kinds of presence: spatial, social, and self. Spatial presence refers to the players' sense of physical embodiment in the game. Social presence refers to players' interaction with NPC's and other players. Self-presence refers to how much the persona that the player takes up in the game is related to their actual self. Other sense of presence involves the different perspectives they can assume like First person vs. Third Person, nature of game as competitive or cooperative, sound, and visual effects, musical scores etc.

Identity - Often, creating an avatar or identity are as important as finishing the missions in a game. Game designers and researchers view this as having the ability to engage player and sustain gameplay. In educational games identifications points to attention of detail to the content to be learned. Falloon (2010) did an investigation on educational importance of using avatar-based games (MARVIN), with social studies and language based among 11–13-year-olds. The program involved creating appropriate characters for their classroom project. It revealed that students were able to present the project better because their avatar presented it. Students who were initially disengaged from classroom activities showed greater engagement. Teamwork and collaboration were also enhanced.

Interactivity – This refers to players ability to start and get feedback for their actions in a game. In the case of serious games, the result of this communication should be to foster players' learning (Moreno & Mayer, 2007).

Agency of Control – The ability to control the flow of game such as having ability to change the storyline depending on actions of the gamer is seen as a central part of gameplay.

Challenge - This refers to different levels of complexities that are allowed in a game. Just as with other elements of a game, the corelation of the challenge posed by the game and the players' skills have importance for inducing flow (Sherry, 2004; Weber, Tamborini, Westcott-Baker, & Kantor, 2009).

Narrative – Players of serious games see narrative as a hook for playing the game. Papastergiou (2009) found that adolescents' feedback on improvements they would make to a serious game to be embellishment of the plot.

Feedback – Feedback provides details about the efficacy of the game from players' point of view. IBM's innova8 incorporates real-time feedbacks that involves investigating discovering and optimizing pain points in a business process (e.g. – points in time where customers are eager to buy a product). Successful process results in positive feedback from characters in game like department managers, usually present in real organizational settings as well. This helps player know their strengths and weaknesses and gets in-game performance evaluations, test solutions and strategies that are ultimately expected to be exported to real business settings [9].

Some Iconic Serious Games

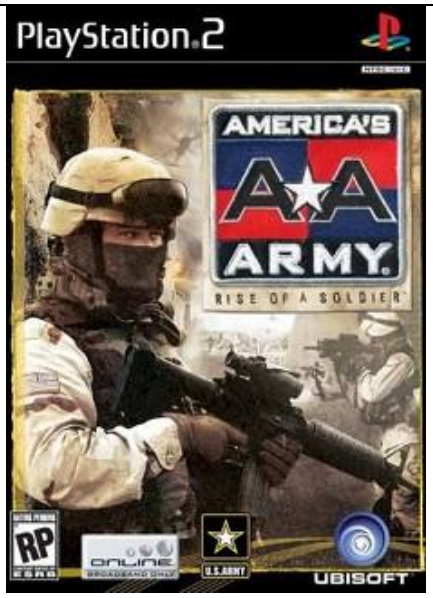


Fig 1: America's Army

America's Army - A first person shooter developed by the US army was distributed over the internet for free. The serious game movement was largely made better in 2002 by the commercial success of this game, which was meant as a recruitment tool for the US army. The game was branded as a strategic communication tool to make it possible for the Americans to explore the army at their own pace. It also intended to raise interest among youngsters between 16-24 years of age to join the army. It is one of the 1st successful and well-executed serious game that attained public recognition.

 *Fig 2: Re-Mission*	**Re-Mission** – This cancer fighting game was designed by HopeLabs. The game aims to impact various psychological and behavioural outcomes associated with successful cancer treatment by involving cancer patients in entertaining gameplay. It is free-to-play for young people with cancer and their families as well as oncology and healthcare workers around the world. This was the first video game to have scientifically proven to improve conditions of young cancer-patients.
 *Fig 3: Fold it*	**Fold-it** - A research project developed at the University of Washington, Centre for Game Science. The aim is to fold the structure of proteins as perfectly as possible using means provided in the game. The solutions with highest scores are analysed by researchers to find whether there is a native structural configuration that can be applied to relevant proteins in real world. Scientists can use these solutions to target and eradicate diseases and create new biological innovations. Foldit's 57000+ players at the time were credited for providing useful results that matched or outperformed algorithmically computed solutions by *Nature*, a 2010 paper in science journal.
 *Fig 4: 1979 Revolution – Black Friday*	**Revolution-Black Friday** - A game where the player takes up the role of a photojournalist in 1979 Iran. It is called the historical documentary game. The game was directed by Navid Khonsari, who was a child in Iran at the time of revolution and designed it with intention of making the player realize the moral ambiguity of the situation. The plot of the game is based on real events and eye-witness testimonies.
 *Fig 5: Kerbal Space Program*	**Kerbal** - A space flight simulation video game, in which the players direct a nascent space program and the game features realistic physics engines, allowing real life orbital manoeuvres like Hohmann transfer orbits and orbital rendezvous. This game has crossed over into the scientific industry with scientists and members of space industry showing interest in the game including NASA, ESA, SpaceX etc.

RELATED WORKS

Training and education of medical staff is a major prerequisite for patient safety. As gamers, medical students can test their skills and simultaneously have freedom to learn by failing until mastering the concepts they have to. In this section we present some serious games in the field of medicine, grouped by application area.

- ## Surgery

  Total Knee Arthroplasty – is a game that describes orthopaedic surgical procedures to residents outside the operating room with multiplayer modality. The aim is to find out if a serious gaming approach will enhance complex surgical skill acquisition. The player must complete the TKA procedure having a sequence of steps to perform, while minimizing time and maximizing score [11]. The games begin inside the operating room with a first-person viewpoint. The gamer must know the correct order of steps for the procedure. The player is corrected by an angry AI assistant. Each step starts with selecting the appropriate tools. If the player chooses the correct tool and performs correct step, they geta multiple-choice question to test their knowledge about this step. At the end the player gets their score.

  Blood Management – [12] puts forth the importance of blood management especially in Orthopaedic surgery where bleeding is common. The game has 3 parts the first and second being "Stop the Fountain" which focuses on teaching trainees blood management skill by trying to stop water fountain loss using virtual tools first on a plane and then on a curved surface. These 2 tasks help trainees adapt to 3D environment and improve their hand-eye coordination through haptic interface. In the 3rd part of the game, players must complete the correct operation and hints are given on the technique. There is no time limit in the training mode. Whilst on time-attack mode the operation must be done within a given time or the patient will die. There is also collaboration mode where several players can work together in a network to complete the task.
  To evaluate the training process the authors asked the trainees to perform vessel sealing session. Completion-time and off-target contact errors were the 2 parameters considered for determining the final score. In a 2nd experiment 21 undergraduate students were divided into2 groups Group 1 with 11 students completed the entire game training of "Stop the Fountains" and Group 2 containing 10 students did not. They were both then asked to do the 3rd part of the game and the mean execution time of those who went through the training process seemed to be lower than those who did not.

- ## Odontology

  Dental implant training simulation - This game developed for training dental students provides a simulation for diagnosing, decision making and protocols for improved treatment outcomes. It allows interaction with patient in a 3D virtual environment. BreakAway's "Pulse" proprietary technology powered the game [13].

Dental health awareness in Children - This game made to evaluate the dissemination of public awareness on preschool children's oral health through a serious game. A study was conducted by parents, teachers, and dentists. The study had 2 parts – the first being developed to evaluate the applicability of technology and also opinions on the effectiveness of the game. In the 2nd part an evaluation questionnaire was given to the same participants and 80% vouched that the game is useful for education and dental awareness [14].

- ## Nursing

Virtual Pain Manager – This online serious game simulates an analgesia machine controlled by the patient. The player must control and reduce the pain of patient within 48 game hours. After this time if the patient is okay, the PCA must be withdrawn and replaced with oral analgesia. Then the player must try and control the patient's pain within next 72 game hours. If the pain remains high after this period, the player fails, and negligence can cause complications and possibly death [15].

VA Critical Thinking – This game allows nurses to practice assessment, prevention, and treatment of illness the patient has related to skin integrity and pressure ulcers. The player interviews patients and has a virtual inspection of patient's skin to mark areas at risk for pressure ulcers. After this, the player gets a result showing which of their answers were correct, incorrect, or missed [16].

- ## Cardiology

Virtual ECG – This game allows an online platform for ECGs accurate recording. The player places electrodes on a virtual patient, connects ECG machine leads and records ECG. Data of real patients ECG scans are used in the simulation to get ECG corresponding to the way player has configured the electrodes. The electrode position can be adjusted by the player until satisfied. This ECG is then presented and overlay on ECG recordings by experts to give a direct visual comparison with the correct recording [17].

- ## First Aid

Code Orange – This is a serious game, where the player works in collaboration with the first-aid staff in a hospital to save people injured by a weapon of mass destruction. It provides a virtual hospital, and the user should complethe hospital procedures to deal with mass casualty situations [18, 19].

Burn Centre - This is a web-based triage game built with Macromedia Flash CS3, that allows users to play in 2 different modes. In triage mode the player should correctly stabilize, sort, tag, and transport burn victims during a mass casualty event in a busy theme park [20]. In burn care provider mode, the player must satisfy the needs of multiple burn victims using simulated hospital equipment, during 36-hour resuscitation period. At the end of each sessions the player gets a report on thier performance.

Dietitian and Diabetes

Affaire Burman – In this serious game the player grows skills in dietary terms, insulin injection and physics activity. The game was meant for children with type I diabetes and multiple daily injection routine [21].

Meli- ´ Melo Glucidique – This is a quiz game focused on carbohydrate knowledge of the patient [22].

InsuOnline – This game was designed to teach insulin therapy in adults with diabetes mellitus to primary care physicians because physician's lack of knowledge is one of the causes of poor glycaemic control in adults with diabetes. For the design and development of the game a team of clinical endocrinologists, experts in medical education and game designers were formed. The player is a substitute physician and treats diabetic patients. The aim of the game is to have each player correctly initiate and adjust insulin therapy for each patient. Right decision leads to next level. Each new level has a different patient and increasing complexity. To test the effectiveness of the game a trial with 128 physicians was conducted. They were divided into 2 groups of 64. One group played the game whereas the other underwent traditional learning. A multiple-choice web-based test was done before, immediately after and 6 months later and the study showed that the game can be an attractive option for large-scale continuous education [23].

Psychology

Treasure Hunt – This game aims to support psychotherapeutic treatment of children. It is the first game focused on principles of cognitive behaviour change and aimed at 8–12-year-olds in cognitive behavioural treatment for different disorders. The game comes in the form of therapeutic concepts and attractive HomeWorks that encourages children to rehearse basic psychoeducational concepts [24].

iSpectrum – In this web game, the goal is to improve the social interaction ability for patients suffering from autism or Asperger's syndrome. The game provides the player an environment much like the real world where they can practice and improve their social interaction abilities and use them in real life. The player is projected into interview situations coming in 3 phases. In the first phase the player meets their employment advisor who explains the different jobs available and gives general employment advice. The player is then given the opportunity to choose from 3 different jobs. After this there is a job interview with the new boss where the player is put in different situations depending on the chosen job and is asked to complete different tasks. These tasks are developed in a strict collaboration with experts. After completion the player can go back and start with a different job [25].

Others

Bio Inc. – This game gives you the role of a doctor and with the duty of saving the life of the patient. The patient has different symptoms somewhere between 4 to 12 including coughing, fever, leg pain etc. The player needs to figure out where all the symptoms overlap and do tests to narrow down the disease. As the patient presents symptoms, you

can click on them and know which diseases shows these specific symptoms, The player can also conduct different tests like blood, MRI etc, to finalize the disease. Bio Inc Redemption has also a mode where you can actively try to kill the patient by infecting them with deadly diseases. Playing against a friend one person can try to save the patient while the other tries to kill them leading to a cat and mouse mechanic which makes the game more fun [26]. The game includes over 600 actual diseases, symptoms diagnostic tests, treatments, and other medical conditions [27].

PROBLEM STATEMENT

[10] admits that serious games developed for healthcare are not as widespread as expected. It also suggests that players practiced on serious games show better results than those who are trained with traditional means such as books or listening to a lecture. Serious games have the potential to outperform these old school approaches of learning. Nothing helps in learning more than doing things by oneself, failing, and learning from mistakes, not reading, not listening, and not watching someone else doing things.

In fields like healthcare there is no room for mistakes. There are no test runs. Lives could be at stake and mistakes could be mortal. Gamification in medical education can be largely useful because it could provide medical students a risk-free environment to learn from, whilst keeping them motivated, whereas a wrong move in real world could cost lives. This could help them gain experience and have a more engaging and fun approach to learning.

Even though the interest in serious games is not new, there are not many that are up to date, and really engaging for the players. With the rise that the game development technology is at right now, there are lots of features that could be implemented today which were unthought of previously and still leave room for improvement.

Even though games like Bio Inc provides a plethora of knowledge in terms of an educational game, it lacks in engaging the player as it mostly includes a selection from a bunch of options, which I think gets boring quickly. Most other games are to narrow and only focus on one application area. There is a lack of games which takes a broader approach to the medical field. I also found most of the games in this field to be question and answer games where the player is presented a set of options to choose from. I think this holds back the fun aspect of the game and make the gameplay boring.

GAME DESIGN

The game design was initiated with the idea of having a simulation game with the player taking up the role of a doctor and trying to diagnose the condition of the patient using symptoms presented to him. However, I did not want this to be a question-and-answer type of game. As a gamer myself, I find these types of games quite boring. I wanted the player to have an on-screen avatar who can interact with the patient and the environment in some way. Another thing I wanted was the game to be 3D, mostly because these are the kind of games I am drawn to and because with where unity and other technological requirements are at right now, I find making 3D games as much time-consuming as making a 2D game. In terms of performance even though 2D games would have been easier on the processor, I think that making the characters and environment to be low poly helped keep the performance complexities minimal.
The first step of game design was to have an idea of a clear beginning, middle and an end for the game. Because the time limit for finishing the project was limited, we chose to have a game loop that can repeat itself like an endless runner game. The game begins with the patient being assigned - the player finds symptoms and diagnoses patient – another day begins, and the same pattern is repeated but with a different disease.
After this was made clear we wanted to have story to engage and hook the players in. I believe in games that have some sort of fantasy element to it. I think it is what draws players to it as the primary reason people would like to play games serious or not is to get away from the complexities of the real world. So, I decided to give the doctor the ability to shrink down in size and interact with the patient. In the game the doctor can travel inside the body of the patient and find clues which gives information about symptoms, prerequisites, if the patient has any hereditary diseases and suggestions for tests to be conducted.
So, we started off with an introduction where the player is eased into the game with instructions about how to play the game. Then we started working on designs for the characters and the environment. The game has 4 characters – The doctor (Player), the Nurse (who helps in diagnosing the diseases and runs medical tests), the patient and invading microbes (enemy characters). There are 2 scenes – The hospital – this is where the game begins with the doctor standing in front of the building. This scene has also a nurse and a patient. Randomly generated dungeon (inside the body of patient) has randomly generated microbe invaders that the player is supposed to kill.
I also wanted to add some fighting elements to the game as I have always been a fan of action games like Tekken, Little Fighter, Mortal Kombat etc. I believe that even though this is not a necessity for a serious game, it would only add to the fun and engaging elements of the game. So, I decided to have microbe invaders that cause the disease as enemy characters so that the player fights and kills these characters for revealing clues. Each disease has a pre-set number of clues ranging from 2 to 5 and they are spawned when the enemy character dies. There is a small but still existent possibility that player collects the same clue more than once they are unlucky. At first this was completely random, then I decided to reduce the chances of the player collecting an already collected clue to 30% to not exhaust the player.

Even though the game was a rinse and repeat model, I wanted the gamers to have a feel of being within a different environment every time. So, I decided to design the level inside the patient body to be like a dungeon. This level uses several rooms that are randomly connected each time the level is loaded. The enemy characters (invading microbes) are also spawned randomly. The player can also collect weapons which comprise of different medical tools that increases the damage they cause while attacking enemies.

The enemy characters also fight back and chases the player once the player is in their reach. Both the player and the enemy character have health which when exhausted either of them dies. If the player's health is depleted, he has the option of restarting the level but loses all the weapons and clues he has collected so far. I have also added some health pickup items which regenerated the health of the player.

Another thing I wanted to do was to make the game accessible both on mobile and pc. So, I designed UI buttons and touch-field that can work on a mobile device the same way as keyboard and mouse work on a pc. I found that creating a mobile build along with the pc build to be easier than first developing the pc build and then trying to port it to mobile as the core features and the interaction within the game remain the same.

After finding all the clues, we needed to do something to trigger the ending of one successful round of the game. So, I had the idea of having a dialogue system in place which helps the player to interact with the nurse. The player always has a clue which suggests which test to run. Once the player has started talking with the nurse, she presents different options for the player to choose from, one of which turns out to be if the player needs to run any test. A pop up having different options for tests is presented and the player must choose the one according to the clue he has collected. On doing so, the disease is being revealed to the player. One of the other options the nurse presents to the player is if he is ready to diagnose the disease. Selecting this option, a different pop up which has a text box appears, in which the player must type in the name of the disease. The player can do this without choosing to do the test if he is confident in his abilities just by referring to the clues he collected. Once the player does this a message is displayed showing if the diagnosis was correct or not. After this a description and treatment plan for the disease is displayed.

I also have a level up progression in the game, which increases the level of difficulty as the player levels up. The player gains experience and levels up by killing enemies, conducting correct tests, and diagnosing diseases correctly. Players can also lose experience and levels if they die, do a wrong test or diagnoses incorrectly. The player is allowed to unlock different tests depending on their level.

Once a complete round of the game is done, the game resets and the player inventory is cleared so the cycle repeats with a randomly selected and different disease. The game has a total of 74 diseases, each having its own set of clues and tests.

The game also has a saving system in place which allows the player to save their progress locally. The player experience and level, health, inventory items (clues and weapons) and position comprise the data that is saved. The player has the option to save the data at any point in the game. However, there is only one save file and the data in it gets overwritten each time save happens. The data is also automatically saved each time the level begins.

I believe the game covers the educational aspect through the clues, medical tests and descriptions of diseases and treatments. It then combines this with the fun part of playing an action-adventure role playing game through its fights, character interaction, progression, and story.

PROPOSED SOLUTION

A simulation game where the player takes the role of a doctor. Patients with random diseases approach the doctor and they must now figure out the disease using the symptoms, conducting tests, and having the knowledge of patients and their immediate family medical history.

To make the game more fun and engaging, the difficulty and complexity of the disease gets higher over time, depending on how good the doctor is performing. The doctor also gets access to different tests as he progresses in the game.

[10] tells that it is undeniable that participants in serious games have better results than those who studied through traditional methods. Serious games provide a useful training method for health professions and health-related studies. I believe I have developed a game which is engaging, fun and most importantly educational for medical students.

Even though the game follows a rinse and repeat pattern, there is randomness as every time the player travels inside the patient body the rooms are connected differently and there is always a sense of something new. The diseases that the patient has are selected randomly depending on the player's level. Moreover, I believe giving the player an avatar who can interact with other characters and the environment engages the player and leaves him wanting to play more.

BUILDING THE GAME

- ## Characters/Hospital modelling (Low poly)

The game contains 4 characters – The doctor, the nurse, the patient, and the microbe invaders. All these characters are low poly and made using Blender. Polygon models can have any number of sides in theory but are generally broken down into triangles for display. Using a low poly model can make the models less detailed but makes them much less computationally intensive to display.

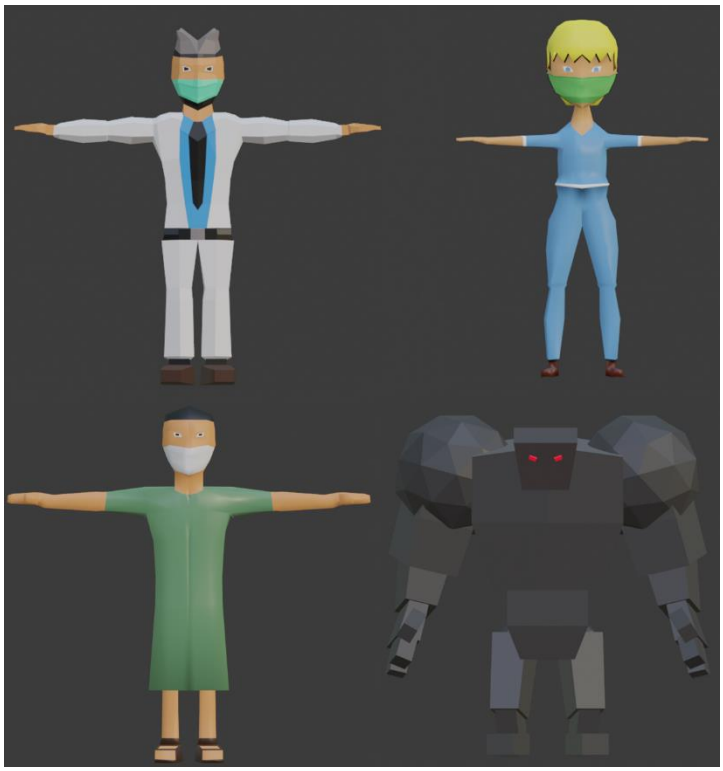


Fig 6: Characters in game

I have also designed a low poly Hospital environment for the game. The game starts with the doctor standing in front of the hospital. When he talks to the nurse, he is informed that the patient is waiting in the lab inside. The hospital design contains, a TV screen, 3 lockers, 4 sinks, 5 cupboards, a table, a desk a hospital bed and many more items.

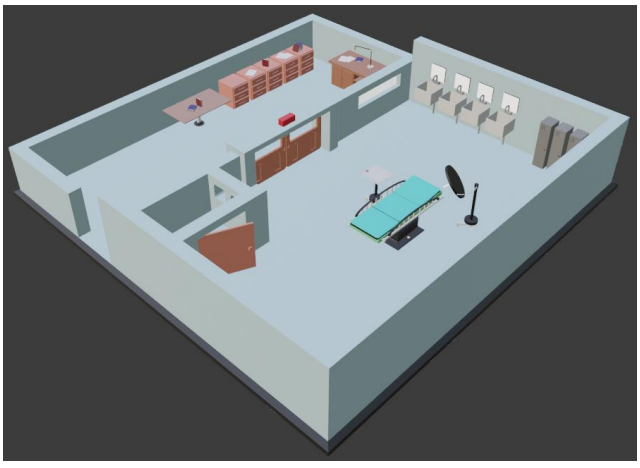


Fig 7: Hospital design

I have also designed some items for health, weapon and clues pickups. I have used low-poly medical instruments as weapons, these however come from a free asset pack [34, 35].

Movement System

The core of the player movement system is built on free standard assets pack [36]. I have however used only 2 scripts from the pack – **Third Person User Controller** and **Third Person Character**. These cover basic actions of the player like speed while running, jump force, angular speed while taking a turn, crouching (not used in the game) etc. I have also used a third person animator controller that comes with the asset pack, which made the job of animating different characters much easier. The animator controller contains 3 main states – Grounded, which includes idle, walk, run and turn animations, Air born – which includes jump animation and Crouch – which includes crouch idle, crouch movement etc. I have made animator override controller with this third person animator controller as the base for all the 4 characters. An animator override controller is used when different characters share some common animations. In this case the animations for character movement are the same for all characters. The only animation that the Nurse and Patient character use is the idle animation.

I have then used Joystick Pack [37] which provides an onscreen joystick for the game to be playable on android devices.

```
control.m_Jump = Input.GetKey(jumpKeboardButton) || jumpButton.Pressed;
control.hInput = Input.GetAxis("Horizontal") + fixedJoystick.Horizontal;
control.vInput = Input.GetAxis("Vertical") + fixedJoystick.Vertical;
```

Fig 8: code for movement and jump

I then created a **Player Controller** script**;** this script takes in Joystick and keyboard inputs (Horizontal and Vertical) and makes the character move in Z and X direction respectively. I also created a **Fixed Button** script; this is a simple script that uses unity engine's **IPointerUpHandler** and **IPointerDownHandler** to set a variable pressed to true or false. This works for both mouse clicks and onscreen button press. A Button of this type is then used for Jump action to be available when played on mobile.

After this the next step was to make the camera follow the player. The camera needs to be always behind the player looking down on him at a certain angle. I did this by using **Quaternion.AngleAxis** to update the camera position in the **Update Method** of the **Player Controller** script.

I then used **Fixed Touch Field** [38] to set up a region covering almost the entire screen that can be used for look-rotation i.e., moving the camera around the player to get a view of what is around. The basics of how touch field works is that screen has x and y coordinates, when you click on the screen, the coordinates where you clicked are stored and as you drag the curser or your fingers on a touch screen the camera position is being updated with an offset amount of the coordinates where the curser is at that moment. Finally, when you stop dragging, the position of the camera stops updating.

```
cameraAngle += touchField.TouchDist.x * cameraSpeed;
camera.transform.position = transform.position + Quaternion.AngleAxis(cameraAngle, Vector3.up) * cameraOffset;
camera.transform.rotation = Quaternion.LookRotation(transform.position + Vector3.up * rotOffset -
    camera.transform.position, Vector3.up);
if (touchField.TouchDist.x == 0)
{ cameraOffset.y -= touchField.TouchDist.y * touchRate; }
```

Fig 9: code for camera movement and touch field

Enemy AI

The Enemy Character mainly works with 3 scripts, **Enemy Controller, Enemy Mover, and Enemy Fighter**. The enemy always targets the player and calculates the distance between it and the player GameObject. There are 2 default float variables on the Enemy Controller, the chasingDistance which represents the enemy that has seen the player but needs to move towards them to attack and attackingDistance represents the enemy in range to make an attack. We constantly check if the player is within chasing or attacking range of the enemy. If the player is within the chasing range, we call a method on the Enemy Mover script to make the enemy move towards the player. As the enemy moves towards the player, it comes within the attacking range of the player. At this point the enemy stops moving (move is set to false on Enemy Mover) and a method in Enemy Fighter calls the Attack Behaviour. This makes the enemy start attacking the player. The enemy has 3 attacks, and each attack has its own cooldown timer. If all attacks are available, one of the attacks is chosen at random. Enemies are initially set to idle and start attacking or chasing the player once they come in range and as soon as the player gets out of range enemies return back to idle.

```
if(GetDistance(target)>chasingDistance)
{
    em.SetMove(false);
}
else
{
    if (GetDistance(target) <= attackingDistance)
    {
        em.SetMove(false);
        ef.AttackBehaviour();
    }
    else
    {
        em.SetMove(true);
    }
}
```

Fig 10: Enemy Controller

There is also a Field of View script on the Enemy which uses **Mesh Filter** to join a circle around the enemy. This is used to indicate the range within which it is unsafe for the player and the enemy will start chasing them.

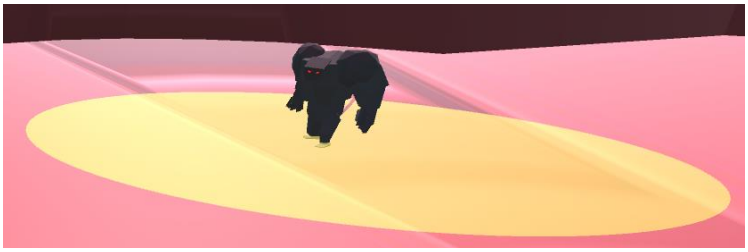


Fig 11: Enemy Field of View

The Clue Dropper script on the enemy is used to drop clue items. This script has a list of uncollected clues and is used to drop clues that the player has not already collected 70 % of the time.

- ## Interaction

All the main interaction the player has with other characters and the environment is covered by the **Interact** script on the player game object. There are 4 main interactions in this way. I have used Raycasting to detect objects that have specific scripts on them. The way it works is by shooting invisible rays from a point in a specific direction to detect colliders laying in the path of the ray. This ray starts from the camera and moves in the direction where the curser was when we clicked on the screen. Then it checks if the GameObject to which the collider belongs to has specific scripts on them. I have used a script called **Door** and if the ray hits an object with this script, it opens the door. I have used this only at one place in the game where the doctor must click on the door to open it and enter the lab to see the patient. I have used a script **TV** and when the ray hits the GameObject a new scene is started, and this scene plays a recorded video of one round of gameplay to help the player understand how the game works. There is a button that can be used to return to the hospital scene at any point. A script **called AI Conversant** is attached to the Nurse GameObject. The interact script uses this to trigger a dialogue when ray hits the nurse game object. A script called **pickup** is used on both weapon and clues pickup which are used to send these items to the player inventory.

- ## UI and Controls (PC and Android)

I have built the game to work both on a pc and android devices. I have key binded all controls for the PC and they also have their on-screen buttons as counterparts.

The on-screen controls mainly contain the joystick, fixed buttons, touch field and normal buttons from unity.

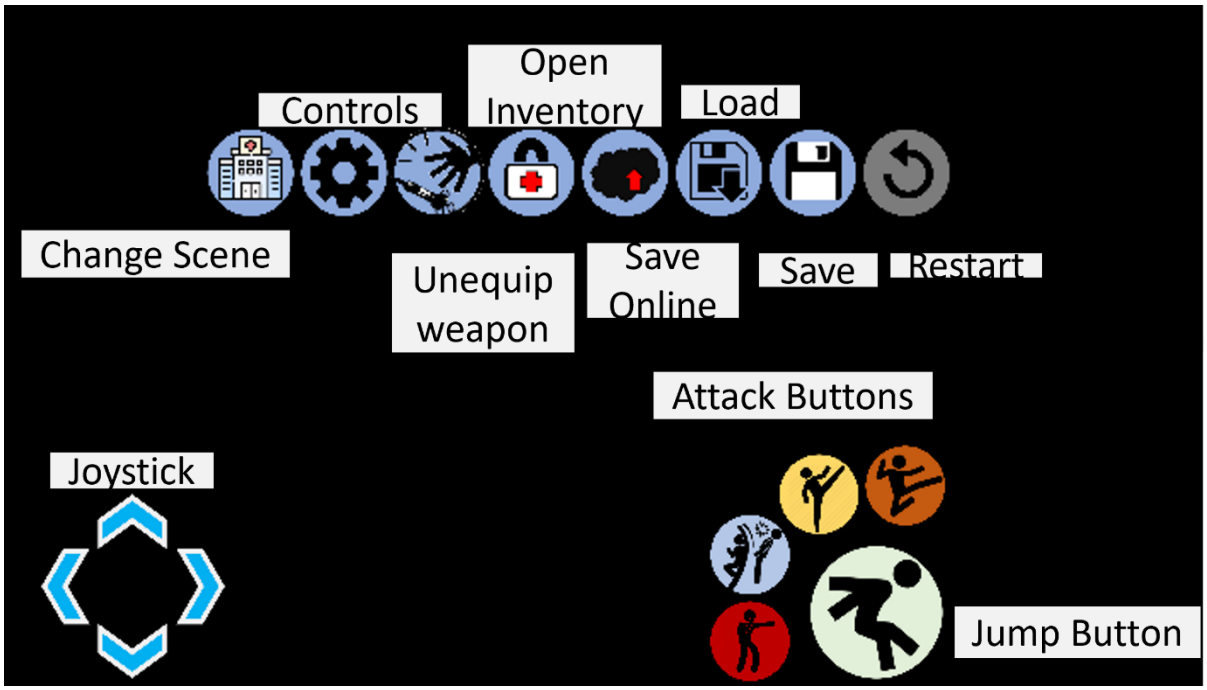


Fig 12: UI Controls

Keys	Use
W A S D | Arrow Keys	Movement
Left Click and drag	Look around
Space	Jump
Back Space	Restart
I	Inventory Open
U	Unequip current Weapon
G	Open Controls tab
F	Change Scene
C	Shrink
1 2 3 4	Attacks
S	Save
L	Load
Delete	Delete Save File

Table 2: PC Controls

I have attached a script **UI Assigner** on the player GameObject, which finds all the Fixed buttons on the screen by their name and adds them to an array in a specific order. These buttons can be then used from other scripts by calling the Get Fixed Buttons method with the index of the array where the button we need is at.

```
    foreach (FixedButton f in fixedButtons)
    {
        if (f.name == "JumpButton")
        {
            fixedButtonsList[0] = f;
        }
        if (f.name == "InventoryButton")
        {
            fixedButtonsList[1] = f;
        }
        if (f.name == "UnequipButton")
        {
            fixedButtonsList[2] = f;
        }
        if (f.name == "SaveButton")
        {
            fixedButtonsList[3] = f;
        }
        if (f.name == "LoadButton")
        {
            fixedButtonsList[4] = f;
        }
        if( f.name == "ControlsButton")
        {
            fixedButtonsList[5] = f;
        }
        if (f.name == "Attack01")
        {
            fixedButtonsList[6] = f;
        }
        if (f.name == "Attack02")
        {
            fixedButtonsList[7] = f;
        }
        if (f.name == "Attack03")
        {
            fixedButtonsList[8] = f;
        }
        if (f.name == "Attack04")
        {
            fixedButtonsList[9] = f;
        }

    }
}
public FixedButton[] GetFixedButtons()
{
    return fixedButtonsList;
}
```

Fig 13: UI Assigner

The UI also includes text fields to display health, level, experience to level up, kill counts and number of diseases diagnosed correctly per total number of diseases diagnosed.

Procedural Dungeon Generator

The second scene of the game is built like a maze, and I have created a procedural dungeon generator to spawn dungeon rooms and connect them randomly at each time. Each of the rooms has 2 or 3 connection points. The dungeon system consists of 8 rooms and 5 hallways.

First, I used the Connector script and draw the connectors using **OnDrawGizmos** to visualize the connection points. I then placed them inside each room and hallways depending on the number of openings the room has.

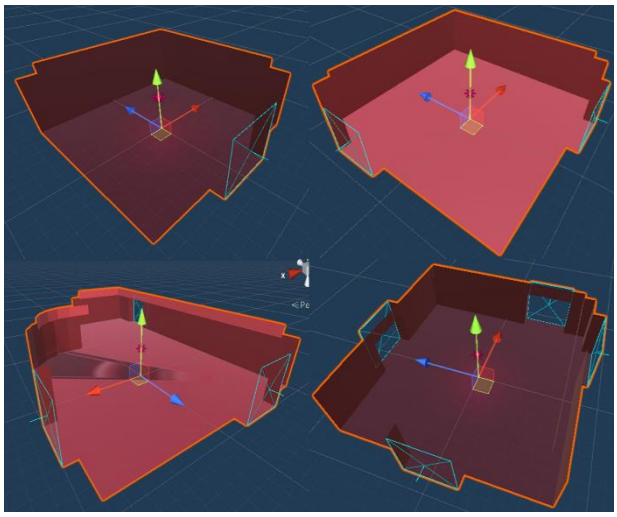


Fig 14: Rooms with 1, 2, 3 & 4 connectors

I then created a **Dungeon Generator** Script which controls most parts of generating random dungeon rooms and connecting them. This script has 3 arrays of room prefabs. One of the arrays startPrefab is filled with rooms having only 1 connection point as I thought they will be perfect for starting rooms. This is where the player will be spawned at the beginning of the scene. The 2nd array – tilePrfab contains all the other rooms and hallways except for the starting rooms. The 3rd array – blockPrefabs is used to block all connectors that are not connected even after generating the main path and all the branches.

I have set up a coroutine **DungeonBuild** to handle the dungeon generation. This works by first building a main path and then creating branches at places where the connectors of the main path are not connected. At first the tile root is set to the starting tile. I also have 2 fields **tileTo** and **tileFrom** which indicate the connectors from and to each tile. tileTo is set up with the starting room. Numbers are set for the length of the main path and length of branches and number of branches. Different empty **GameObjects** act as the containers for the main path and different branches.

A method **CreateStartTile** handles the instantiation of the starting room. The rotation of the room is also set here to be 0, 90, 180 or 360 on the y axis randomly. This method returns the transform of the created room.

```
Transform CreateStartTile()
{
    int index = Random.Range(0, startPrefabs.Length);
    // Instantiating the starting room
    GameObject goTile = Instantiate(startPrefabs[index], Vector3.zero,
        Quaternion.identity, container) as GameObject;
    goTile.name = "Start Room";
    // Setting the y rotation of the room to be 0, 90, 270 or 360
    float yRot = Random.Range(0, 4) * 90f;
    goTile.transform.Rotate(0, yRot, 0);
    // Add to generatedTiles
    generatedTiles.Add(new Tile(goTile.transform, null));
    return goTile.transform;
}
```

Fig 15: Creating Start tile.

A method **CreateTile** handles the instantiation of other rooms. Connecting 2 tiles is then controlled by the **ConnectTiles** and **GetRandomConnector** methods.

```
Transform CreateTile()
{
    int index = Random.Range(0, tilePrefabs.Length);
    GameObject goTile = Instantiate(tilePrefabs[index],
        Vector3.zero, Quaternion.identity, container) as GameObject;
    goTile.name = tilePrefabs[index].name;
    // Add to generatedTiles
    Transform origin = generatedTiles[generatedTiles.
        FindIndex(x => x.tile == tileFrom)].tile;
    generatedTiles.Add(new Tile(goTile.transform, origin));
    return goTile.transform;
}
```

Fig 16: Creating other tiles.

Inside the **ConnectTiles** method I call **GetRandomConnector** twice to set up connectors from and to by passing in tilefrom and tileTo as parameter. **GetRandomConnector** goes through all the connectors on the tile and returns the transform of a connector on the tile which is not connected yet. Then connectFrom is set as parent of connectTo and connectTo is set as parent of tileTo. I then set the local position and rotation of connectTo to zeros. Rotate connectTo on the y axis at 180 degrees and reset tileTo's parent to be the container.

After creating the starting room, I loop through many times as I want the size of the main path to be setting up tileFrom with tileTo (so initially tileFrom will be the starting room). Then I call the Create Tile method to set up the new tileTo. Then call the Connect Tile to connect the 2 tiles. On the next run of the loop the newly created tile, which is still in tileTo now becomes tileFrom and Create Tile is called again.

After the main path is build, I search for all the connectors which have not been connected to any room and add them to a list. Then I loop through as many times as I want the number of branches to be and do a similar loop to the one creating the main path inside this to set up branches. The length of this loop being the length of each branch.

After the main path and the branches are generated, I use the method **BlockPassages** to go through all the connectors in the whole dungeon and block the ones which are not yet connected to anything.

Object Pooling

Object Pooling is a good technique used widely by game developers to improve the performance of a game. In cases where we need to instantiate and destroy a certain object, it is better to use Object pooling. What is done here is instead of instantiating an object onto the scene every time the player levels up we are just going to have a particle effect object already available on the scene when the game begins to be inactive. Then we just set it to active and set the position to wherever you want the object to be, when you need it, instead of instantiating a new one. Then Instead of destroying the object we just set it to inactive and put it back in the queue, so it can be used later.

Here I have used object pooling for enemy characters, health and weapon pickups and health regeneration and level up particle effects. A **Pool Manager** script is attached to an empty game object. This script takes in a list of prefab **GameObjects** to be pooled along the number of each GameObject.

Combat System

The combat system mainly consists of 2 scripts; **Fighter,** which handles the player's combat system and **EnemyFighter,** which handles the enemy's combat system. The player and the enemy both have 4 attack animations which are controlled by the **PlayerAnimator** and the **EnemyAnimator** respectively**.** Both these are animation override controllers with **ThirdPersonAnimator** as the control. The enemy always targets the GameObject with the tag **"Player".** The player has a list of all GameObjects with tag **"Enemy".** Then from this list the enemy which is the closest to the player is the target. Both the player and the enemy have their respective attacking range. Both the player and the enemy have a **Health** script attached to them and when the animation event is triggered, it calls a method **Hit** on their respective combat scripts which checks if their counterpart is in attacking distance and if yes, it calls a method on the health script called **Take Damage,** which reduces the health of their target. If their health reaches zero, they die and an animation for death is played. Player then has the option of restarting the level, but he loses some experience and all items he had collected.

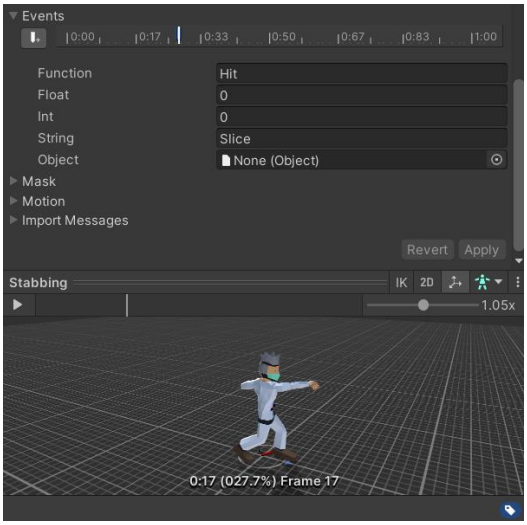


*Fig 17: Example of **Hit** event on the Doctor*

When a player presses an attack button while not having an enemy within the attacking range, the animation still plays but the **Take Damage** method is not called. In case of the enemy, attack behaviour only happens when the player is within the attacking range. One of the 4 attack animations are selected at random. All the attacks have a cooldown timer, which means that once an attack is used the character must wait a certain amount of time to be able to use it again. A text field overlaps the attack button image on the screen indicating the time after which the Player can use the same attack. This is done using the **Cooldown Timer** script. This script has 2 dictionaries of key-value **<string, int>**. The key being the Attack number, "Attack01", "Attack02" etc. In the first dictionary cooldownTime the values are populated by a fixed number indicating the wait period between the same attack. The values of the 2nd dictionary nextAttackTime is populated with 0s initially. When an attack happens the nextAttackTime is updated to using *formula 1*. Then a check condition before initiating the attack makes sure that the cooldown time has passed when using the same attack again.

Next Attack Time = Cooldown Time + Time.time

formula 1: Updating next attack time.

Inventory, Items and Equipment

The player inventory is a pop-up screen which opens when the player hits a button. The inventory shows all the clues and weapons the player has collected. The clues are spawned when an enemy character dies and the weapons are spawned randomly at the beginning of the 2nd level. The Player uses the Interact script to click on these items and add them to their inventory.

The Inventory UI consist of a canvas as the root parent. This contains a panel which holds a text field for description of clue items and a scroll view. The description text fields take up half of the Inventory panel. The other half has the scroll view that holds the inventory. The slots in the inventory are managed using a grid layout group. A script called **PlayerInventory** attached to the player determines the number of slots in the Inventory. I have created a prefab of the inventory slot and the **InventoryUI** script on the inventory populates it with slots using the number from **PlayerInventory**. This is done by calling a **Redraw** method to recreate the inventory. An **Event Action** on the **PlayerInventory** script is used to subscribe to this method and whenever this action occurs Redraw method is called.

The **Inventory Slot** prefab is basically an Image which acts as holder for items. This has another Image **Inventory Item** attached to it. The Image component here is disabled, so that when the player picks up an item, we set the image of the **Inventory Item** of the slot the item goes into with the sprite that the item holds and enable its Image component. This is done through a **SetItem** method on the **InventoryItem** script attached to **Inventory Item**. This method takes in an Item and an int (number of items) as parameters. If item is null, we disable the image, else we set the image of inventory item to image sprite from the item by calling **GetIcon** method in the Item Script and

enable the image component of the Inventory Item. The Inventory Item has also a text field attached to it. If the number of items is greater than 1, we update the text field with this number.

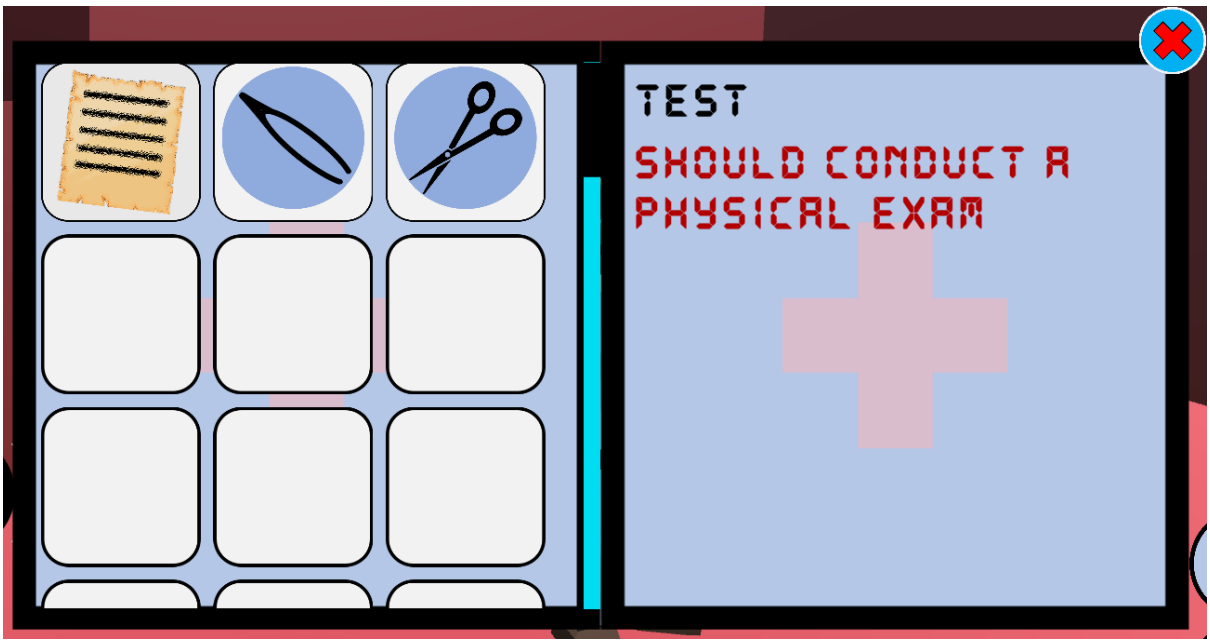


Fig 18: Inventory UI

There are 2 types of things that can fill up an inventory slot, an item and an equipment. Items are the clues that the player collects, and equipment is the weapons that the player can use in battle. Items just stay in the inventory and shows a description when clicked on, whereas equipment can be attached to the player when clicked on and visually be shown like the player holding a syringe. These are controlled by 2 scriptable objects **Item** and **Equipment**. Equipment inherits from Item and therefore has all the fields and methods that are available to Item. The Item class has fields and getters for Item unique Id, which are generated using **System.Guid.NewGuid(),** display name of the item, its description and sprite which acts as the icon for the item in the inventory. Some Items can be stackable, for example same clues are stackable in the inventory, so when the same clue is picked up twice, we just update a text filed in the inventory slot indicating the number of clues. I also created an enum indicating if the item was a clue or a weapon.

Equipment has fields for an Animator Override Controller. The player's animator is replaced with this when a player is holding a weapon to change the animations when he attacks. It also has a float called **damageModifier** which indicates the increase in damage when the player has a weapon. I created an enum which tells the position where we want the equipment to be attached. Right now, we only have weapons, so they always go in the arms of the player. However, in future if I want to add, say armor and want a helmet to get attached to the player's head, this could be useful. Then I attached the weapons to the player's hand in Unity Editor and made prefabs saving the position and rotation of the weapon when it is held by the player. The Equipment scriptable

object also has a GameObject field which uses this equipped prefab. Then Scriptable Objects for each clue and weapon were made.

When a player tries to pick up an item by clicking on it, the **PlayerInventory** script on them has a method **AddToFirstEmptySlot** and if an empty slot is found, it adds the item to the inventory by calling **Redraw**. It also has a method **AddItemToSlot** which takes in as parameter the Item, number of items and the slot number to add the item in. This is used in cases where we want to drag and drop items in a different slot or swap 2 items' position in the inventory. Again, this is accomplished by calling **Redraw**.

An **ItemManager** script is responsible for what happens when the player clicks on an item in the inventory. It first checks if the item is a clue or a weapon. If it is a clue **DisplayDetails** method is called which displays the item display name and description. Initially the player holds a default weapon, which is just an Equipment scriptable object with an empty GameObject for the quipped weapon. 2 fields in the ItemManager holds this – defaultEquipment and zeroEquipment. When the item the player clicked on from inventory is a weapon, **Equip** method in **ItemManager** is called. Here we check if the player had already a weapon, and if he does, the weapon he holds is sent to the inventory and he will equip the new weapon. The instance of the weapon from the scene is destroyed and an inventoryItem representing the item is added to the first empty slot in the inventory. The defaultEquipment field is also replaced with this new weapon. When saving, we always save the defaultEquipment, so that when we load back, we can spawn the same weapon in the player's hands. The zeroEquipment remains the same and is used to spawn an empty GameObject when the player unequips all weapons.

Fig 19: player holding scissors.

The items in the inventory UI can be dragged from one slot to another. If the 2nd slot does not have any item, simply drop is done otherwise the 2 items are swapped. This is handled by **DragItem** script which inherits from **IBeginDragHandler**, **IDragHandler** and **IEndDragHandler** interfaces. This script has field parentCanvas, which refers to the root InventoryUI canvas and source which refers to the inventory slot where the

dragging was initiated. **IBeginDragHandler** requires the implementation of the method **OnBeginDrag** which basically reparents the **inventoryItem** to the root canvas instead of the source, so we can drag it around. **IDragHandler** in the method **OnDrag** uses its parameter eventData to get details like where the mouse pointer is hovering over on the screen and updates the position of the item every frame. **IEndDragHandler** requires the implementation of the method **OnEndDrag**. Here we check if the position where the dragging ended was an inventory slot or not. If not the image of the component of the source slot is enabled with the same item's sprite. If the drag ended on an inventory slot, that slot becomes the destination, and we call the method **DropItemIntoContainer**. In this case we check if the destination slot already has an item. If not, we call the method **AttemptSimpleTransfer** which calls the methods **RemoveItem** to remove the item from the source slot and **AddItems** to add the item to the destination slot. In case there is already an item present in the destination slot, we save the items on both the slots to local variables. Then remove items from both the slots using the **RemoveItem** method and call **AddItems** method to add the item previously in the source slot to destination slot and vice-versa.

- ## Dialogue System

The dialogue system is a pop-up that appears on the screen indicating a conversation between the Player and the Nurse where the nurse provides different options for the player to chose form. These options include, conducting a medical test, diagnosing the disease etc. The core of the dialogue system is based on the **Dialogue** Scriptable Object which represents a node diagram of the dialogue. I created a dialogue editor for this purpose. This was done by creating a Script DialogueEditor, inheriting from the **EditorWindow** class. In this script I have a call-back to open the window from menu items using **GetWindow**. **MonoBehaviour.OnGUI** method was used to draw things in the editor window. **OnGUI** is called for rendering and handling GUI events. **EditorGUILayout** was used to lay out these fields in the editor window automatically. I created a method **ProcessEvents** to handle dragging of nodes. First when the mouse button is down and we do not have a node selected previously, but there is a node at position of mouse click. This node is selected, and we will be able to drag this node around in the editor. If the position on screen where the mouse click happened doesn't have a node, then we just scroll in the canvas. Finally, when the mouse is up, we set dragging node to null and stop dragging on the canvas.

```
private void ProcessEvents()
{
    // When we click on the mouse while no node is seleced
    if(Event.current.type == EventType.MouseDown && draggingNode==null)
    {
        // Search for nodes at point of mouse click
        draggingNode = GetNodeAtPoint(Event.current.mousePosition+scrollPosition);

        // If a node was found at point of mouse click
        if(draggingNode!=null)
        {
            // Setting dragging offser
            draggingOffset = draggingNode.GetRect().position - Event.current.mousePosition;

            // Setting Active Objeect as dragging node
            Selection.activeObject = draggingNode;
        }
        // if a node was not found
        else
        {
            // dragging canvas is set true
            draggingCanvas = true;
            // Setting dragging canvas offset
            draggingCanvasOffset = Event.current.mousePosition + scrollPosition;
            // Setting active object to selected dialogue
            Selection.activeObject = selectedDialogue;
            // We basically scroll in the editor window
        }
    }
    // if we are dragging and dragging node is not null
    else if(Event.current.type == EventType.MouseDrag && draggingNode!=null)
    {
        // setting the position of dragging node to mouse position + offset
        draggingNode.SetPosition(Event.current.mousePosition + draggingOffset);
        GUI.changed = true;
    }
    // if we are dragging and dragging node is null
    else if (Event.current.type == EventType.MouseDrag && draggingCanvas)
    {
        // setting the scroll position to offset - mouse position
        scrollPosition = draggingCanvasOffset - Event.current.mousePosition;
        GUI.changed = true;
    }
    // when mouse up and dragging node is not null we set dragging node to null
    else if(Event.current.type == EventType.MouseUp && draggingNode!=null)
    {
        draggingNode = null;
    }
    // when mouse up and dragging canvas is true we set dragging canvas to false
    else if (Event.current.type == EventType.MouseUp && draggingCanvas)
    {
        draggingCanvas = false;
    }
}
```

Fig 20: ProcessEvents

Every **DialogueNode** has a **Textfield**, for the dialogue itself, an Add button, to add a new node and child to it, a delete button to delete the node, and a link button. The link button when clicked turns the link buttons on all other nodes to Child, which allows making other nodes child of the selected node. All these are drawn in the Editor window and laid out using **GUILayout.** I used EditorGUIUtility to style the node differently for the AI and the player.

```
private void DrawNode(DialogueNode node)
{
    GUIStyle style = nodeStyle;
    if(node.IsPlayerSpeaking())
    {
        style = playerNodeStyles;
    }
    GUILayout.BeginArea(node.GetRect(), style);

    node.SetText(EditorGUILayout.TextField(node.GetText()));
    GUILayout.BeginHorizontal();
    if (GUILayout.Button("Add"))
    {
        creatingNode = node;
    }
    DwarLinkButtons(node);
    if (GUILayout.Button("Delete"))
    {
        deleatingNode = node;
    }
    GUILayout.EndHorizontal();
    GUILayout.EndArea();
}
```

Fig 21: DrawNode

I created the method DrawConnection to draw the link between a child and a parent node.

Links are drawn using Bazier curves from the xMax on x axis and center point in y axis of RectTransform of parent node to the xMin on x axis and center point in y axis of RectTransform of child node.

```
private void DrawConnections(DialogueNode node)
{
    Vector3 startPosition = new Vector2(node.GetRect().xMax, node.GetRect().center.y);
    foreach (DialogueNode childNode in selectedDialogue.GetAllChildren(node))
    {
        Vector3 endPosition = new Vector2(childNode.GetRect().xMin, childNode.GetRect().center.y);
        Vector3 controlPointOffset = endPosition - startPosition;
        controlPointOffset.y = 0;
        controlPointOffset.x *= 0.8f;
        Handles.DrawBezier(startPosition, endPosition,
            startPosition + controlPointOffset,  endPosition - controlPointOffset,
            Color.white, null, 4f);
    }
}
```

Fig 22: Drawing connections

Creating a dialogue editor made it easier to create several choices for the player to chose from and change the conversation according to the choice. Each node also has a list to store all its child nodes, a Boolean to check if it is the Player's dialogue or the AI's and 2 string fields **onEnterTrigger** and **onExitTrigger** to trigger events when entering or exiting a node. In the game, the exit triggers are used to show the corresponding pop-up windows when the player wants to run a test or is ready to diagnose the disease.

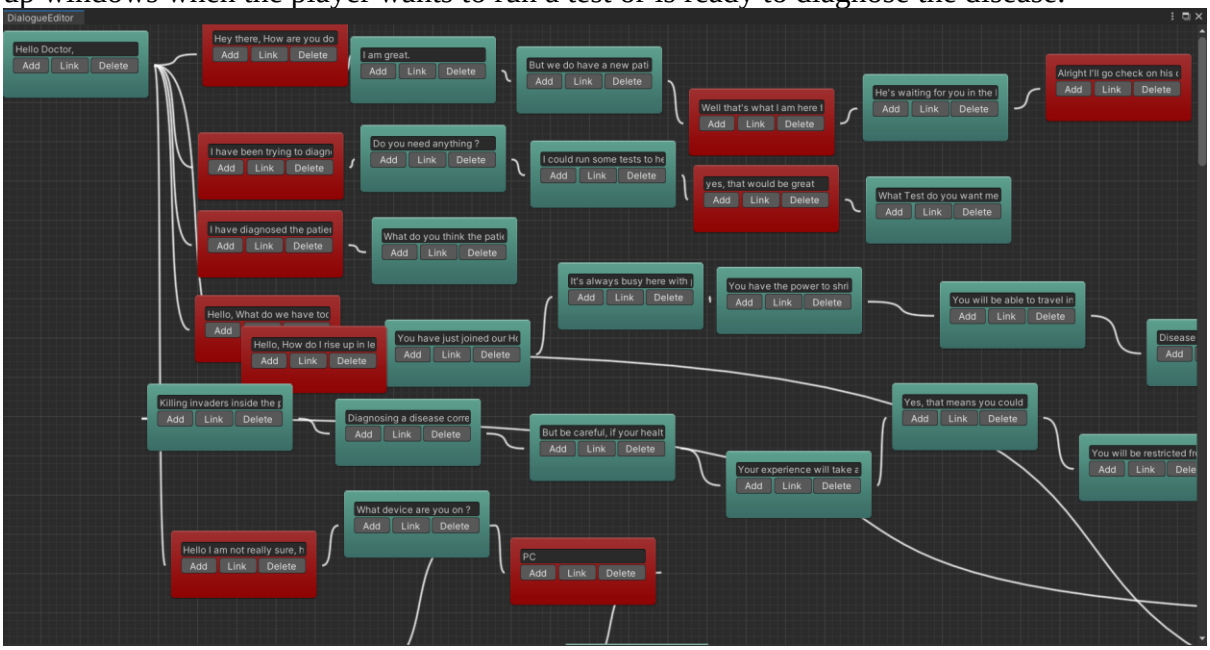


Fig 23: Dialogue Editor.

The Dialogue UI consists of a **TextField** to keep track of the current speaker. An empty GameObject called AIResponse deals with all the dialogues the AI has. This has a Textfield to display the dialogue and a next button to go to the next dialogue.

The dialogues the player has are managed by a choices GameObject, which is filled with Button Prefabs that get populated with the different responses that the Player can select from.

Fig 24: AI Dialogue

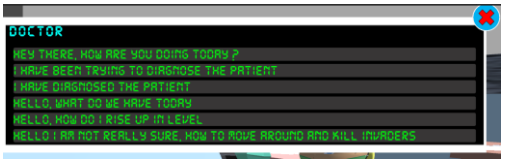


Fig 25: Player Dialogue

Saving System

The game uses 2 saving systems one to save the data locally and the other using the firebase database to store progress online. Both saving systems are available for the player to use at any point in the game. However, to save the data in the Firebase database the player must create an account first.

Saving the Data Locally

The saving system mainly contains 2 scripts, **SavingSystem** script which controls all the core part of saving and a **SavingWrapper** which involves how the game uses saving system, for example invoking save or load when a button is pressed.

I have used serialization to save the data locally. Serialization is the process of taking all the important states in the game and turning it into binary. I then save this into a file and when I need to load the data I deserialize it and fill the fields they represent. The data saved include – player health, items in inventory slots, kill count, number of diseases diagnosed correctly, total number of diseases diagnosed etc.

The location where the file is saved depends on the device the game is played however, I used **Application.persistentDataPath** to get this location. Also, by using **Path.Combine** method we can format the delimiters in the path depending on the device.

I created a **SaveFile** method in the **SavingSystem** script which opens the file using the path, serializes the data into binary and saves it using **BinaryFormatter**. A **LoadFile** method on the **SavingSystem** is responsible for reading from the file deserializing it and updating the fields.

Now to save the data, the SavingSystem does not know what to save. I created a Script called **SaveableEntity** which is attached to all **GameObjects** that have something to be saved. I then created an **ISaveable** Interface which is inherited by all the scripts that need to save something, for example health value in the health script. This interface has 2 methods, **CaptureState** and **RestoreState**. So, when health value needs to be saved, the **CaptureState** returns the health value and when it needs to be loaded **RestoreState**, has a parameter that sets up the value health on the health script. The return type of **CaptureState** and the datatype of parameter in **RestoreState** is an **object**. The **SaveabaleEntity** has **CaptureState** and **RestoreState** methods which checks for all Components on a **GameObject** that inherit from **ISaveable.** Then adds them to a Dictionary to be serialized and saved. The Key to the Dictionary being the type of the class the saveable field belongs to, converted to a string, e.g., **Health** is the class when we are trying to save **health points**. The value is the field to be saved. The **SavingSystem** also has its own **CaptureState** and **RestoreSatate** methods. In the **CaptureState** here I used a dictionary to save the dictionary from each of the saveable entities. The key is unique Id which is generated in the **SaveableEntity** script. This method is called from the **SaveFile** method to get the data to be Saved.

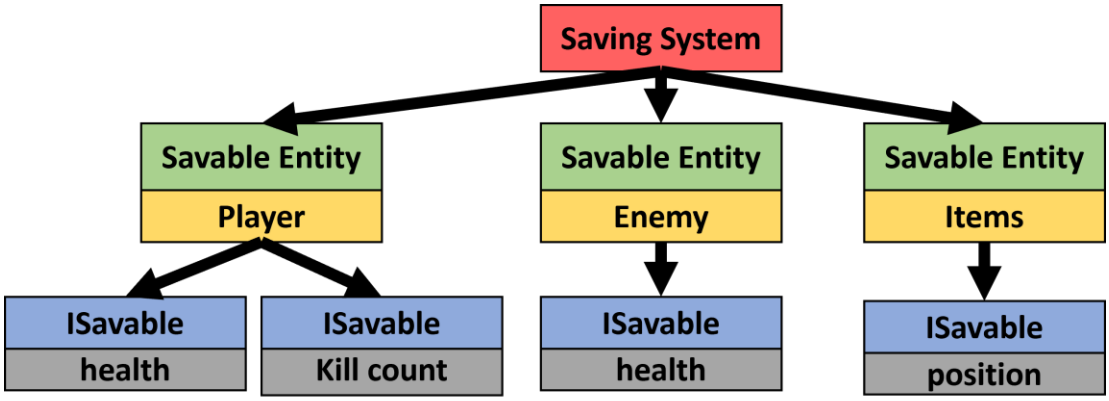


Fig 26: Saving System.

When loading data, deserialization happens in the **LoadFile** method and then **RestoreState** from **SavingSystem** is called, passing in the dictionary saved earlier as a parameter. Here I loop through all the **SaveableEntities** in the scene, use the dictionary to find the unique ID that matches the current SaveableEntity's unique ID and call the **RestoreSate** method on that **SaveableEntity** and passes in the Dictionary which contains details about all the Saveable's of that Entity. Here we loop through all the **ISaveables** on the **GameObject** that use the dictionary key to match class type and call **RestoreState** on that Script and passes in the value that was earlier captured. This then updates the value on that script to complete the loading process.

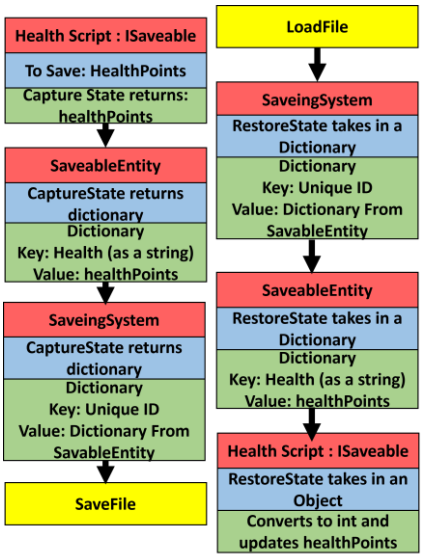


Fig 27: Working of Saving System.

Firebase for saving

The player can open a pop-up window by clicking on a button which allows to save the game to firebase real time data base [43]. The first time the player opens this window, it has 2 input fields for email and password, a button to register and a text field which shows different messages through the process. Since the first time the player has no account with firebase, the message lets the player know he needs to create an account to save data online. For this the player a must enter an email and a password, then hit the register button. The email must be in the correct format (a@b.com) and the password must be at least 6 digits indicated by a placeholder on the input fields.

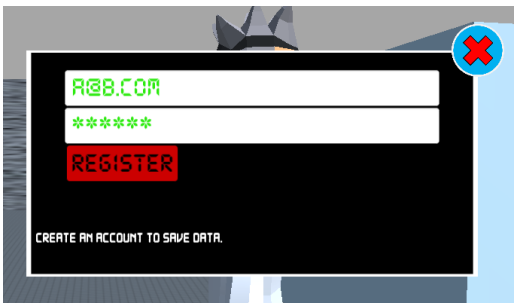


Fig 28: Create an account *Fig 29: Saving and Loading*

Once the player has registered the register button is replaced by 2 other buttons Save and Load. The input fields are still there, but the message text has changed. Now the player can save or load the data by entering the email and password they used to register.

I have set up a project in Firebase which allows email and password authentication. Then I created a real time database to store these data. When the Register button is pressed, the **Register** method of the **OnlineSaveLoadManager** script is called. Here we first establish that the email and password fields are not empty, then subscribe to a **FirebaseAuth** method called **CreateUserWithEmailAndPasswordAsync** which takes in the email and password. Exception handling is used to display any possible error in the message text field. If everything works fine the user is created and we can get a userid from **task.result**. This is a unique id that firebase creates for each user. Now we need to save the data at that point to the database. On completed, I call a method **SaveData** which takes in the newly created userid and email as parameters.

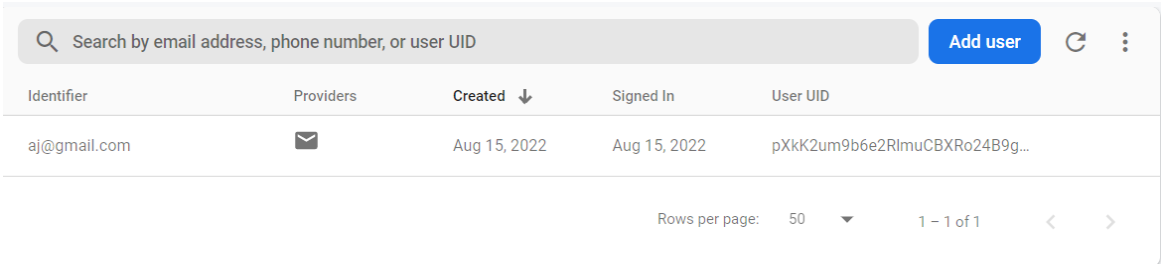


Fig 30: User Authenticated

We can get a reference of the database by using **FirebaseDatabase.DefaultInstance.RootReference.** I have created the class **PlayerData** which contains all the fields that needs to be saved online and a constructor to set these values. The fields include the userid, useremail, current experience, max experience to level up, kill count, diseases diagnosed correctly, and total number of diseases diagnosed. These are then converted to a Jason string and passed to the database to be saved using the reference. After this the user is logged out from Firebase. I have used this method to be less intensive on performance as the player does not need to be connected to the firebase once the saving is done.

```
public void  SaveData(string userid, string usermail)
{
    reference = FirebaseDatabase.DefaultInstance.RootReference;
    data = new PlayerData(userid, usermail, (int) experience.GetExperience(), (int) experience.GetMaxExperience(),
        acheiver.GetDiagnosisSucceeded(), acheiver.GetTotalDiseasesDiagnose(), acheiver.GetKillCount());
    string jsonData = JsonUtility.ToJson(data);
    reference.Child("Player_" + userid).SetRawJsonValueAsync(jsonData);
    messageString = "Save Success";
    FirebaseAuth.DefaultInstance.SignOut();
}
```

Fig 31: Save Method

If the player now hits the Save or Load button, we first check if the FirebaseAuth current user is null, which ideally should be the case as we are logging out each time we save or load the data. Then a **Login** method is called which takes in a Boolean parameter save. This is true if the player had hit the save button and false if the player had hit the load button. In this method we again establish that email and password fields are not empty, then subscribe to a **FirebaseAuth** method called **SignInWithEmailAndPasswordAsync**. This method takes in the email and password and checks to find a match in Firebase. Exception handling is used to handle errors and update the message textfield depending on it. If all goes well, the player is logged in. If the Boolean save is true, we just call the same **SaveData** method with the userid and email as before to save the date.

Fig 32: Data in Firebase real time database.

However, if the player had hit the Load button, we call a different method the **LoadData,** where we again use the database reference, now subscribing to a method **GetValuesAsync** to get all the values in the form of a **DataSnapshot**. With this we convert the data to a Jason string using **GetRawJsonValue** method and then update the **PlayerData** class by calling its constructor. After that we use the PlayerData to update the different fields we have saved by calling their setters. Finally, we sign out from firebase.

```
public void LoadData(string userid)
{
    reference = FirebaseDatabase.DefaultInstance.RootReference;
    reference.GetValueAsync().ContinueWith(
        task =>
        {
            if (task.IsCanceled) { message.text = "Loading data canceled"; return; }
            if (task.IsFaulted) { message.text=task.Exception.Flatten().InnerExceptions[0].Message; }
            if (task.IsCompleted)
            {
                DataSnapshot data = task.Result;
                string playerData = data.Child("Player_" + userid).GetRawJsonValue();
                PlayerData pd = JsonUtility.FromJson<PlayerData>(playerData);

                acheiver.SetKillCount(pd._totalKillCount);
                acheiver.SetDiagnosisSucceeded(pd._diagnosisSeucceeded);
                acheiver.SetTotalDiseasesDiagnosed(pd._totalDiseasesDiagnosed);
                experience.SetExperience(pd._experiece);
                experience.SetMaxExperience(pd._maxExperience);
                messageString = "Load Success";
                FirebaseAuth.DefaultInstance.SignOut();
            }
        }
        );
}
```

Fig 33: Loading the data.

Both times when we save or load the data a call to **Save** method on **Saving Wrapper** is done to save the data locally as well.

Disease Library and Diagnosis

The disease library contains 74 different diseases. Their symptoms, prerequisites for the disease, whether it is hereditary or not and suggested tests are used clue items. I have also used a description of the disease and treatments suggested for the disease. The diseases library was built by collecting data from sites like [40]. I also collected data from playing other serious games like bio inc. [41].

I have a **Disease** Script which has 4 dictionaries; first has disease and description as key value pair, second has disease and test required as key value pair. Third has tests and level required as key value pair, and finally a dictionary with disease and experience gained for diagnosing the disease as key value pair.

Running Test

When the player wants to run a test for a disease (clicks the option for test when talking to the nurse), a pop with buttons for all the tests is displayed. When clicked on a button, a method on **RunTest** script called **OnButtonPressed** is called. This method takes in a string (name of the test) coming from the button onClick event. It first checks if player has collected all the clues. If yes, checks the Diseases script to see if this test is accessible to the Player. If yes, the test is matched with the disease the patient has used the dictionary mentioned above. If it is the correct test the disease is revealed. If any of the cases fails an appropriate error message is displayed and player loses experience.

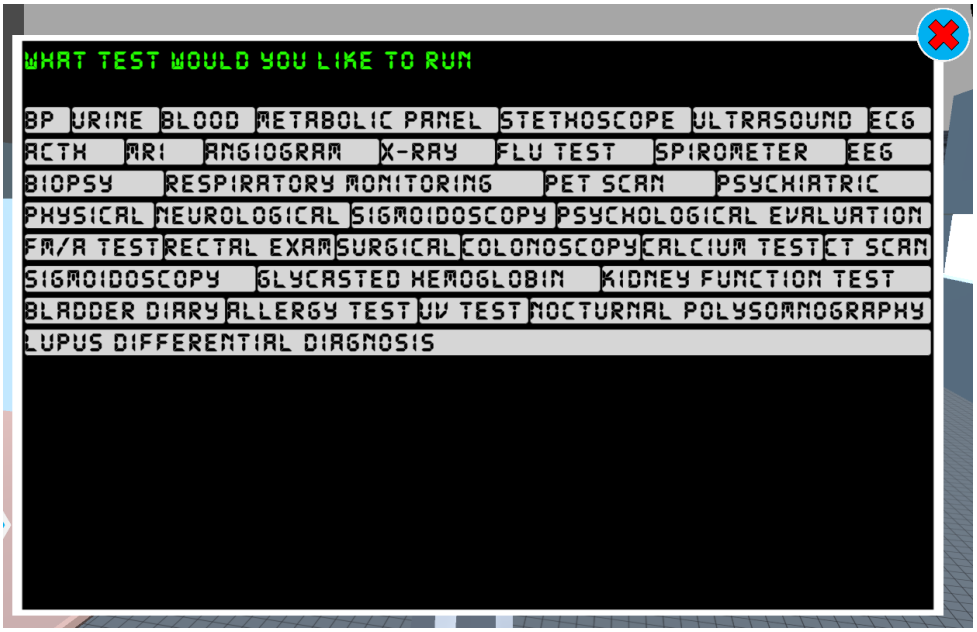


Fig 34: Run Test

Diagnosing

When the player wants to diagnose the disease (clicks the option for diagnosing when talking to the nurse), a pop with a text field to enter the disease is displayed. The player must enter the name of the disease and click enter button. This calls a method **OnEnterDisaesePressed** on the **Diagnosis** script. We check if the player has collected all the clues. If yes, we start a coroutine to diagnose the disease. Here we clear the player's inventory and get him ready for the next round. We update the total diseases diagnosed and check if the disease the player entered matches the disease the patient had. A message is displayed indicating if the diagnosis was right or wrong and the disease description and treatment is revealed. This comes from the dictionary mentioned earlier.

Then on closing the pop up a new round begins.

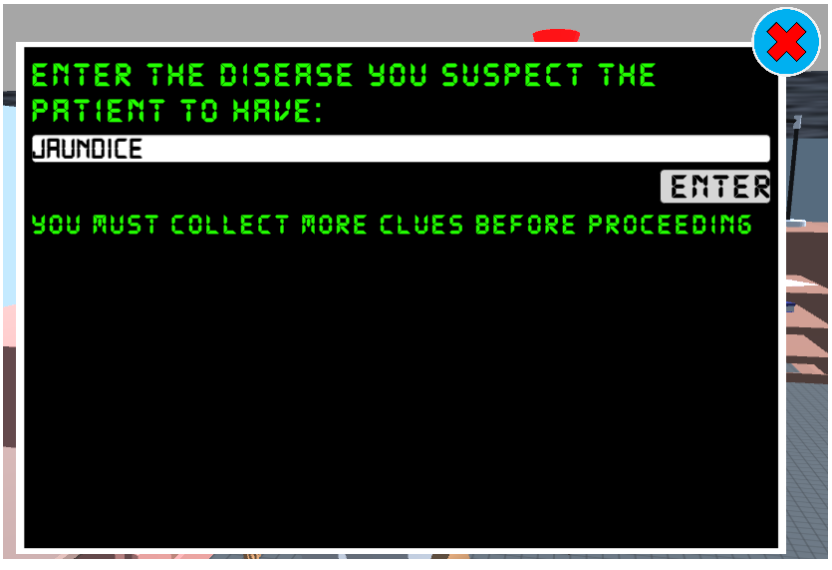


Fig 35: Provide Diagnosis

Levelling up and Progression

The player gains experience by killing enemies and diagnosing diseases correctly. The player can also lose experience by dying or diagnosing incorrectly. Each level has an experience ceiling and when the player passes this, they level up. Player can also lose experience and level down. As the player goes up level, more diseases from the disease library become accessible and player also gets to use more tests.

I have an enum **CharacterClass** which represents the type of character. In the game we have Player, Enemy and NPC. Another enum **Stats** has the values that we want to change as the character levels up, for example health, experience required to reach next level etc.

Two Scripts **Experience** and **CharacterStats** are attached to each character. **Experience** has fields for character's current experience and maximum experience that the character requires to level up. A method **GainExperience** is used to update the current experience when the player kills an enemy or diagnoses a disease correctly. **CharacterStats** tells the type of character by using a field of type Enum **CharacterClass**. It also has fields for character's current level and a Progression Scriptable Object which acts like a lookup table. It has each of the character classes and under each character class we have different stats we want to progress over gameplay. Under each of these stats we have a mapping between all the levels the character has and its corresponding stats value.

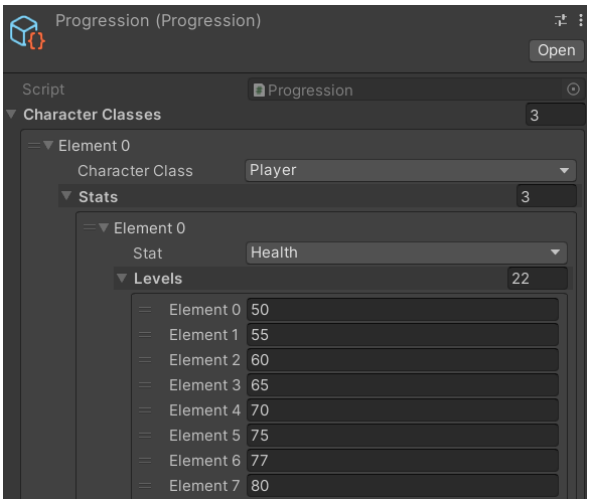


Fig 36: HealthPoints of the Player for different levels

So, the CharacterStats uses the class type and the current level fields on it to get the different stats from the Progression scriptable Object.

The progression scriptable Object has 2 classes **ProgressionStat** which has Stats type and an array representing the levels and **ProgressionCharacterClass** that has the character class an array **ProgressionStat** of all the stats that need progression over gameplay. Progression has a field of type **ProgressionCharacterClass.** Then I used a method **BuildLookup** which first loops over all the **ProgressionCharacterClass** elements. Inside this we loop through all the **ProgressionStat** in each

ProgressionCharacterClass and add them to a dictionary called **statLookupTable**, with the stat type as key and level as value. Then for each **ProgressionCharacterClass** we have another dictionary **lookupTable** which takes character type as key and its **statLookupTable** as value.

```
[System.Serializable]
class ProgressionCharacterClass
{
    public CharacterClass characterClass;

    public ProgressionStat[] stats;
}
[System.Serializable]
class ProgressionStat
{
    public Stats stat;
    public float[] levels;
}
```

Fig 37: Progression Scriptable Object is built on these 2 classes

```
private void BuildLookup()
{
    if (lookupTable != null) return;
    lookupTable = new Dictionary<CharacterClass, Dictionary<Stats, float[]>>();
    foreach (ProgressionCharacterClass progressionClass in characterClasses)
    {
        var statLookupTable = new Dictionary<Stats, float[]>();
        foreach (ProgressionStat progressionStat in progressionClass.stats)
        {
            statLookupTable[progressionStat.stat] = progressionStat.levels;
        }
        lookupTable[progressionClass.characterClass] = statLookupTable;
    }
}
```

Fig 38: BuildLooup

Particle Effects

Particle effects are not widely required for the game as it did not require much. However, some places where particle effects are used are near the head of the patient where the VFX acts like a portal to the next scene, at the portal where the 2nd scene begins. There is a particle effect being used for when the player levels up and when the player picks up a health item. I have used some of the particles from Cartoon FX free asset pack [39].

Sound Effects and Music

Sound effects and Music has been widely used in the game. I have used 2 music scores from Sound cloud [31] for the different scenes. Most of the sound effects I used come from [32]. Here are some regions where I have used sound effects.

- When the player attacks and enemy is not in range
- When the player attacks and hits enemy
- When the enemy attacks the player
- Opening door

- Starting a dialogue with nurse
- Player or Enemy taking damage
- Player or enemy dying

Adding a background music and different sound effects made the game more engaging. The game would have been rather blunt without music. I have used an **Audio Manager** script to control the music and the sound effects. This works almost the same as Object Pooling. First, I created empty GameObjects and attached the audio source for each sound effects and made prefabs out of it. Then I added them to the pool. Here the size of each pool needs to be 1 as there is no point in having 2 of the same sound effect at the same time. The Audio Manager script spawns all these prefabs when the game begins and sets them inactive. When the specific sound effect needs to be used, we set them to active, set the position to where the sound needs to be played from and call the function **Play** on the **Audio Source** attached to it and is set to inactive again.

PROJECT MANAGEMENT

The time allotted to for the project was about 3 months. We started by the end of May. Weekly meetings were held to make sure that the flow of the project was smooth. My Supervisor Katerina was as enthusiastic and excited about the project as I was and helped me every step of the way providing me with creative suggestions and ideas to develop the game. She never tried to force any of her visions about game development on me but encouraged me to develop the game in my way and provided a helping hand to make the project better. Play testing was done at every step of the way, to identify and fix bugs while developing the game. Here is a Table and Gantt Chart of the different stages of development of the game. Details about the project management has been updated in Jira [44].

Tasks	Start Date	Days to Complete
Creating Player Models - Player, Nurse, Patient	18-May	6
Implementing Player Movement, Jump and Shrink	24-May	2
Implementing Camera Follow	26-May	2
Implementing Joystick, Fixed Buttons and Touch Field	28-May	3
Brainstorming Ideas for game story	31-May	6
Creating a Hospital model	31-May	8
Creating Intro Sequence	08-Jun	5
Creating a Procedural Random Dungeon Generator	13-Jun	7
Portal to and from 2nd scene	16-Jun	4
Creating an Inventory & Item Pickup	20-Jun	5
Creating a clue item	22-Jun	3
Gathering Data about various Disease	25-Jun	30
Creating Character Model for Enemy	25-Jun	2
Creating Player Combat System	27-Jun	3
Creating Enemy Combat System & Item Dropper	30-Jun	3
Creating a local Saving System	03-Jul	7
Polishing the intro Sequence and improving story	03-Jul	4
Enemy movement and chasing Player	10-Jul	2
Adding Weapon item and health pickups	12-Jul	7
Adding Dialogue System	19-Jul	7
Running Test and Diagnosis	26-Jul	4
Clue Collector and Diseases to manage current disease and clues	30-Jul	4
Adding Music, SFX and VFX	04-Aug	4
Completing 1 full round of gameplay	04-Aug	1
Adding an Online Saving System	05-Aug	2
Creating a survey and testing with people	07-Aug	15
Improving the game based on feedback	07-Aug	13
Writing the Report & Proof reading	01-Aug	20
Play Testing	24-May	75
Finishing the report & Submission	27-Aug	1

Table 3: Project Management

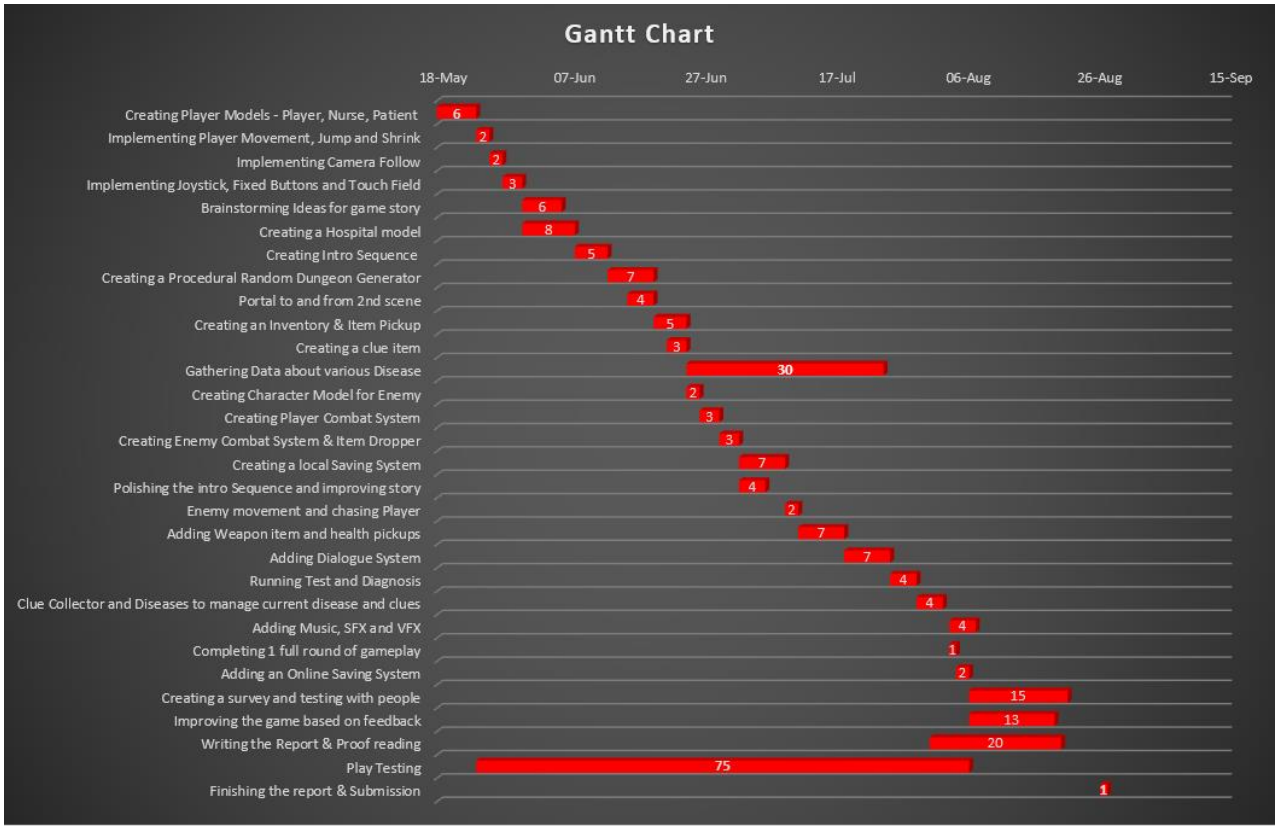


Fig 39: Gantt Chart

EVALUATION

Even though the game is aimed at medical students, we were not able to reach many medical students in time for the submission of the project. However, we have tested the game on a group of selected people who were willing to play the game. The gameplay of 1 complete round takes only up to 3-5 minutes.

After playing the game the participants were asked to complete an one-minute survey which had 10 questions and the reception was mostly positive. All participants agreed that serious games could make the learning process more fun. Most of the participants found the game to be interesting and fun to play and agreed that the game can be beneficial to medical students. Some of the participants had great suggestions which I think could improve the gameplay, which I would love to add in the future but cannot at this point because of the time constraint of the project.

The following shows the questions and replies from the survey:

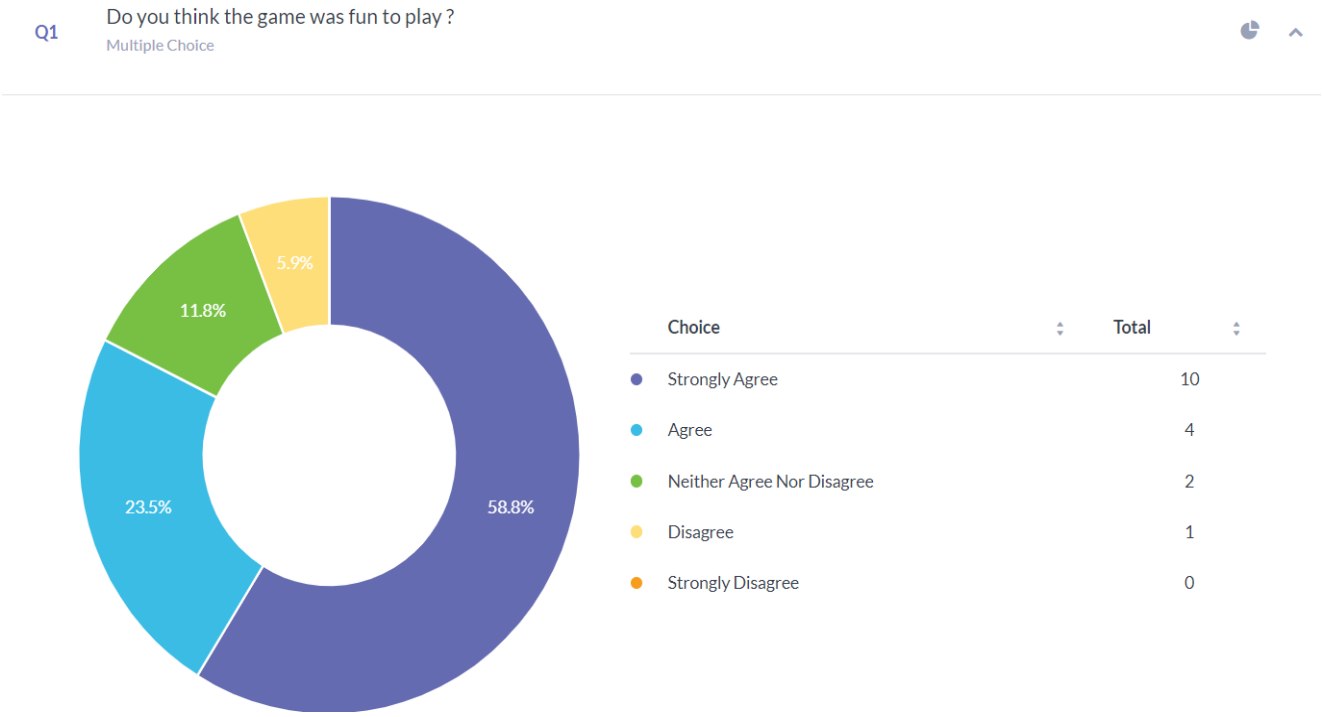


Fig 40: Survey Question 1

More than 82 % of the people who played the game thought it was fun to play. The small portion of the audience who did not like the game said that they found it difficult to understand the working of the game in the beginning. However, the addition of the map and the gameplay tutorial would help the players perform better.

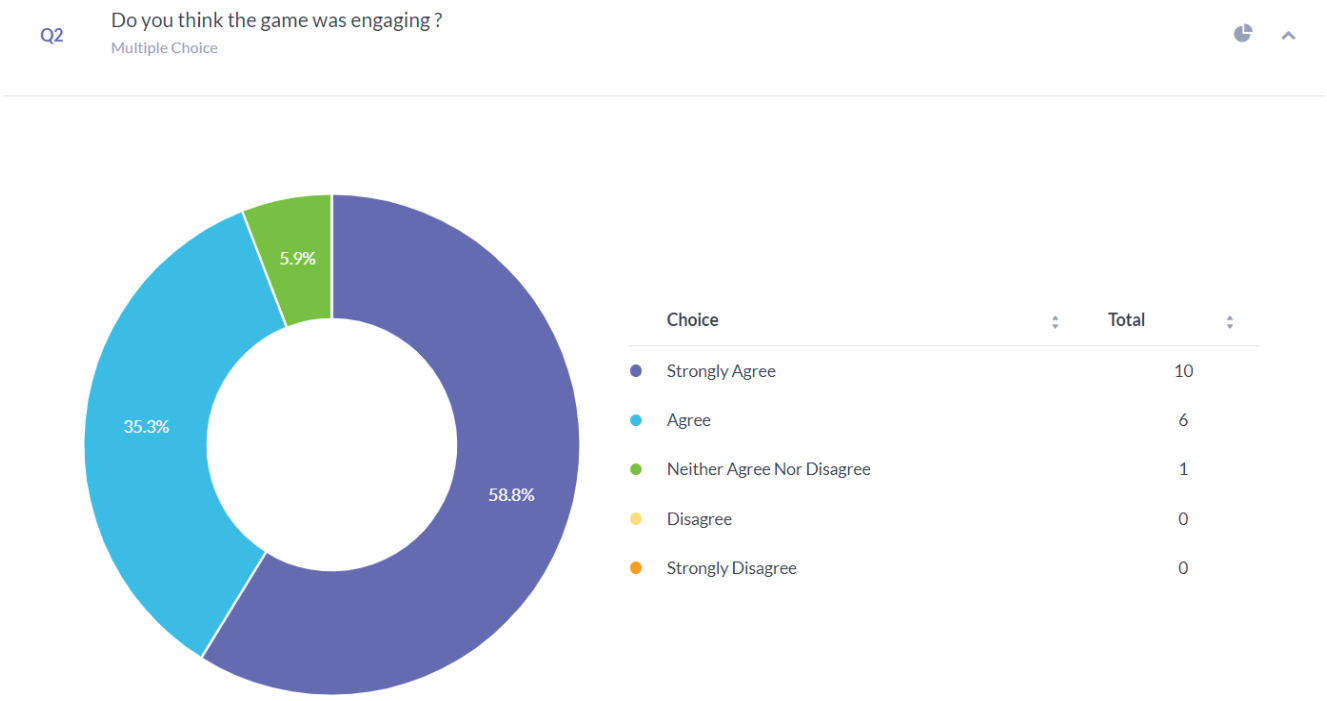


Fig 41: Survey Question 2

Almost all the players thought the game was engaging. This was a relief as making the game engaging was one of the main goals when setting out the game.

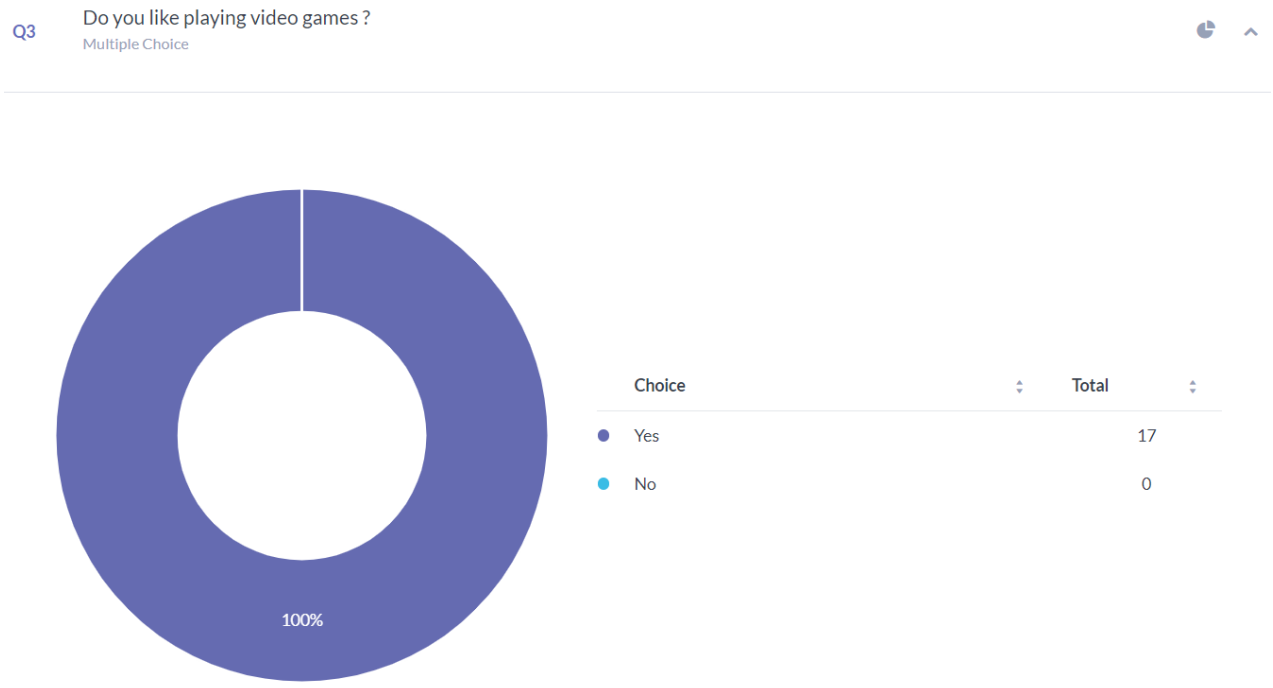


Fig 42: Survey Question 3

All the participants in the survey liked playing videogames.

Q4 If you like playing video games, what kind of games do you like ?

Essay

Date	Answers
8 hours ago	Action
6 days ago	Puzzle games/ adventure games
Aug 11	FPS, Platform
Aug 10	Sims Call of duty Dragon ball Z Madagascar Tetris Mario Kart
Aug 10	Sports
Aug 10	Clash of clans,nfs,pubg
Aug 9	TPP action
Aug 9	All
Aug 9	RPG
Aug 9	First person or third person shooter games.
Aug 9	Fighting, Racing, football
Aug 9	Adventure
Aug 9	Adventure, Strategy, Sports
Aug 9	Action, Strategy
Aug 9	RPG
Aug 9	MMORPG

Fig 43: Survey Question 4

This question helped to understand the different genres the participants liked playing. The majority of the participants loved playing action, adventure, strategy and RPG games. As our game can be included in all these genres, I believe it would be a success when released to a larger audience.

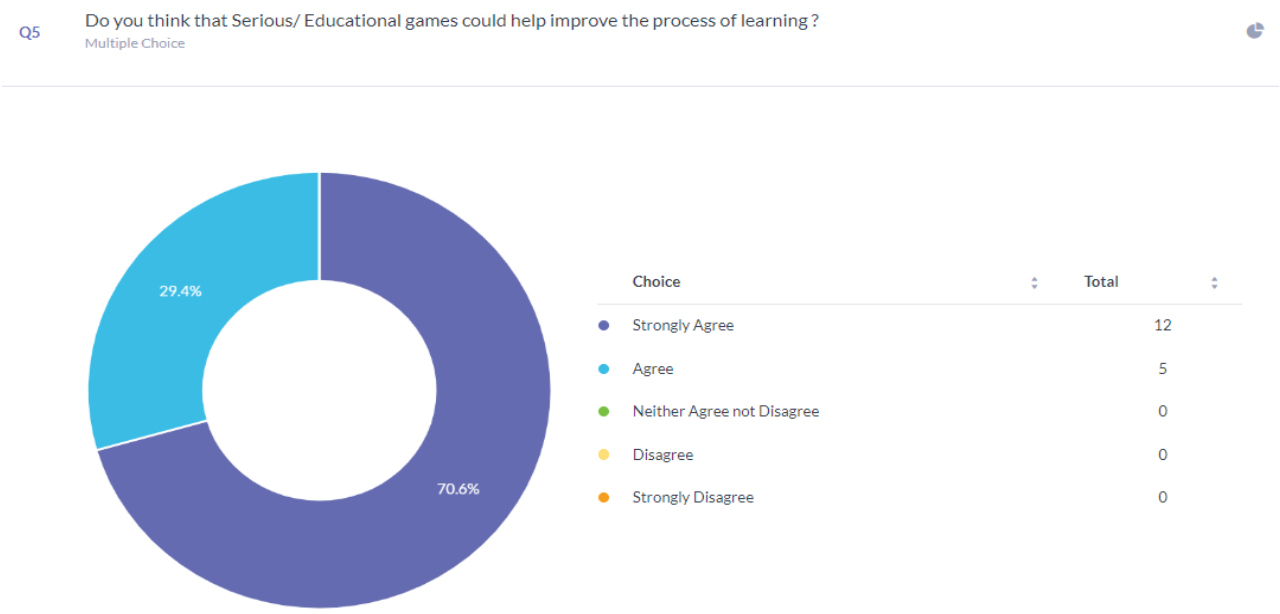


Fig 44: Survey Question 5

All the participants agreed that educational games can help improving the process of learning.

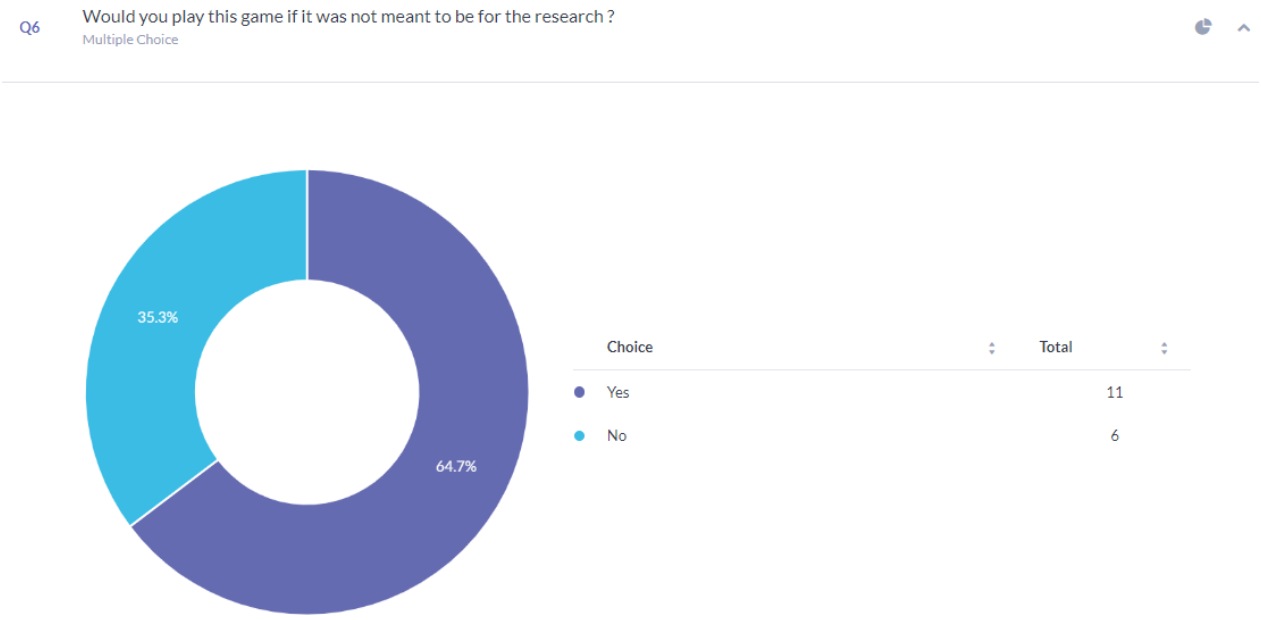


Fig 45: Survey Question 6

Almost 65 % of the participants said they would keep playing the game if it wasn't for research. This is an absolute plus and shows that the game was a success.

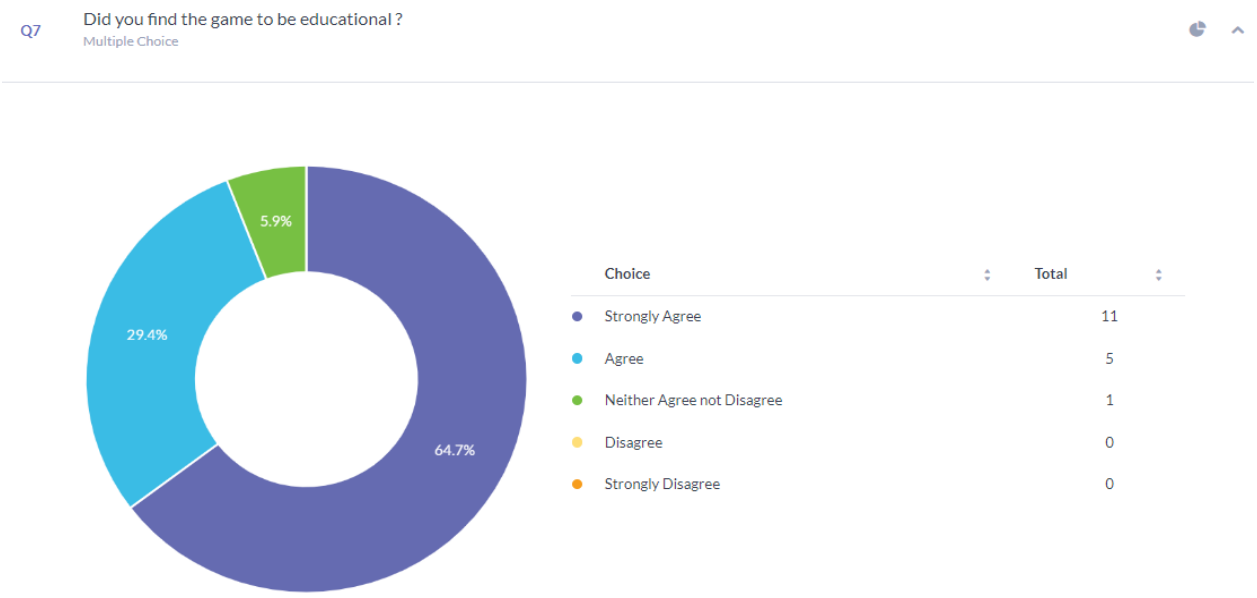


Fig 46: Survey Question 7

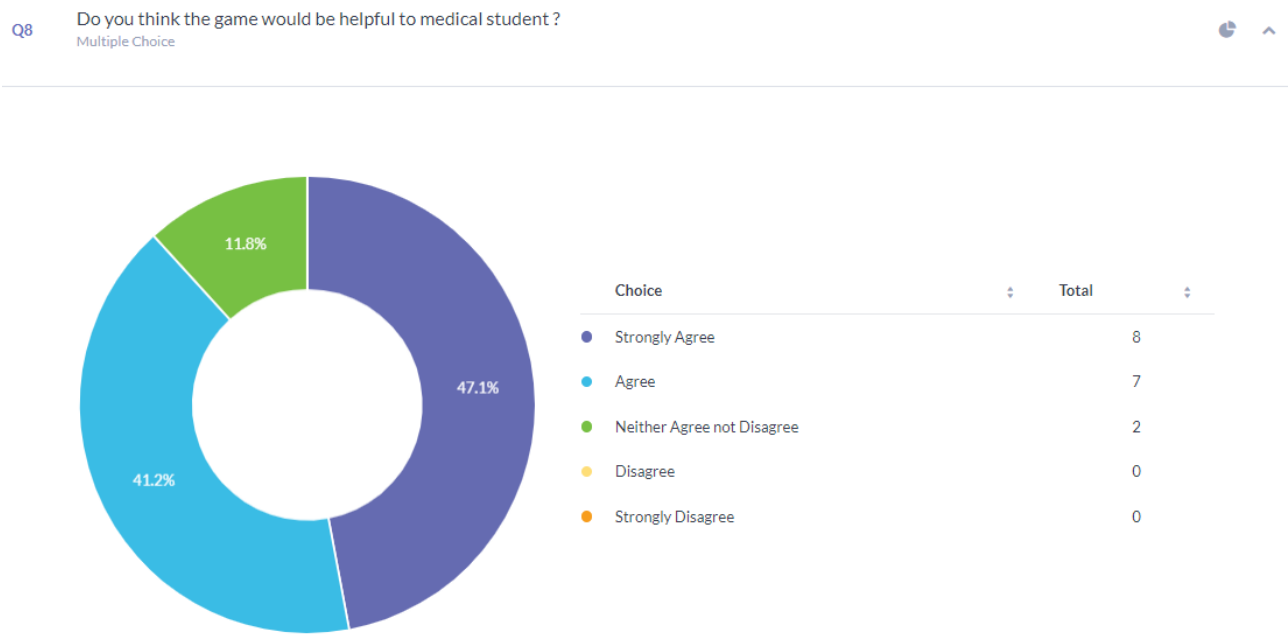
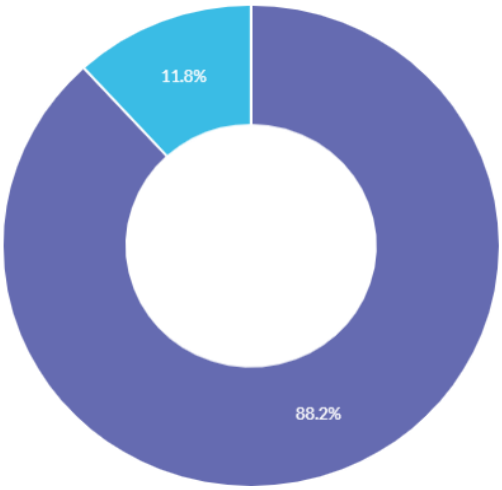


Fig 47: Survey Question 8

94% of the participants thought the game was educational and 88% thought it would be helpful to medical students, which suggests that the game was a success when it comes to the serious aspect of it.

Q9 Do you think education can be made fun through serious games ?

Multiple Choice



Choice	Total
Strongly Agree	15
Agree	2
Neither Agree not Disagree	0
Disagree	0
Strongly Disagree	0

Fig 48: Survey Question 9

All the participant agreed that education can be made fun through serious games. As most of the participants were between the ages of 20-30, it shows that they have an open mind and youngsters today will welcome serious games as a means to improve the learning process.

Q10 What aspects of the game do you think needs improvement ?

Essay

Date	Answers
8 hours ago	Multiplayer game should be more fun and more educational
6 days ago	Topic engagement and graphics
Aug 11	In the first place difficult to understand gameplay. It is needed more clues regarding which tasks should be done next. Map can be added.
Aug 10	Graphics and background music
Aug 9	Some pop-ups giving instructions for the initial level world have been nice, including how to proceed, which keys to press, directions to the hints and so, I also believe there is an issue with rendering and on a smaller note background music is not suiting the game.
Aug 9	Level generator, graphics, story, tutorial
Aug 9	Could add more levels
Aug 9	Having more features or unlockable items
Aug 9	No it was best, the person try really hard to make the game better
Aug 9	The camera movement, Character interaction and movement
Aug 9	Add a Map
Aug 9	The 3D animation could be more immersive
Aug 9	Sound Effects

Fig 49: Survey Question 10

I have taken into consideration and added some features to make the gameplay more fun. One of the suggestions in the survey was to add sound effects, this has been done. Also, the survey

suggested that the background music does not fit the theme of the game. So, I have changed the background music on the 2nd scene to one which gives more mysterious, and an urgency feel. I have added a button which when clicked opens a pop-up window which shows all the controls. I have added a cylindrical arrow on top of all the **GameObject** that the player can interact with in the Hospital scene. Further I have added a new scene which plays a video of 1 whole round of gameplay when entered. This scene is entered by clicking on the TV in the Hospital scene. I believe all these would make the game easier to play.

I however restrained from adding a map. The whole purpose of the 2nd level is to be like a maze, and I believe adding a map would take away from the gamer's experience of exploring the level by themselves and finding where the enemies are by themselves. However, I have added a mini map that would help the player navigate better. This was done by using a **Raw Image** in the UI and adding a new camera game object as a child of the Player **GameObject**, with the camera always facing top down on the player. The view of the camera then fills the raw image. I have also added a slider which updates the camera's orthographic size and helps zoom in and out on the mini map. There is a button used to open the mini map as well as one to collapse and hide it, in case the player thinks it is crowding the screen.

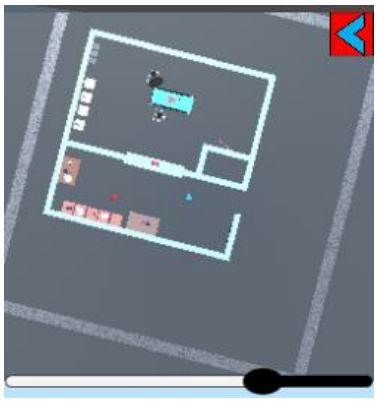


Fig 50: Mini map

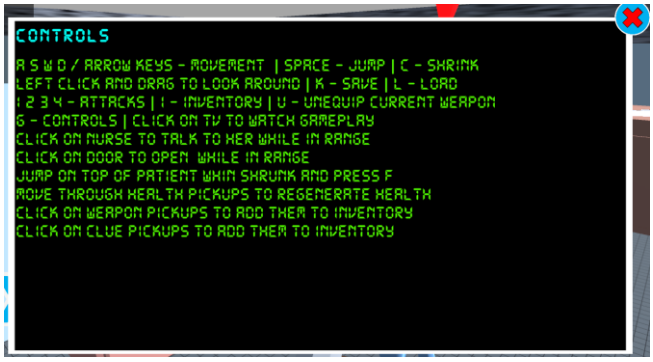


Fig 51: Controls Menu

I also do not wish to add on to the graphics, as the main intention behind making the game models, low poly was to have it easy on the display. If I create more detailed characters, I will obviously add to the graphics but at the cost of being more computationally intensive. This would cause lag and malfunction on mobiles and devices with low processing speeds.

I have added a controls pop-up for the player to know what the different keys do. I have also added a video of gameplay which plays when player clicks on the TV. One of the options the nurse gives while having a conversation with the nurse also explains what to do in the game and what different buttons do.

TECHNOLOGY AND RESOURCES USED

The most important resource that the University of Essex provides for the project is my supervisor. In this project I am going to be working under the guidance of Dr. Katerina Bourazeri who has well know contribution in serious games. There will be regular weekly meetings with the professor to assess progress, guide the research and evaluate results.

- ## Unity

  Unity is a cross-platform game engine developed by unity technologies. The engine can be used to create 2D as well as 3D games [28]. Unity has been playing an active role in the field of game development and with the passage of time the effectiveness and efficiency of the engine has magnified. Unity provides a primary scripting API in C# using Mono. It also provides Bolt which is a visual scripting asset which allows the user to create logic without writing a single line of code.

  Unity provided a chance to build the 3D Hospital environment, setting up the storyline and dialogue system, levelling up the character, adding particle effects and setting up the 2D UI for the game. I also made use of some free assets available in the unity asset store which helped in the development of the game. All the sprites for UI elements like joystick and buttons were created by me using just MS PowerPoint. Some of the top games made with Unity Engine include Kerbal, Space Program, Valheim etc.

Fig 52: Valheim

  Unity has both Free and Paid licence options. The free option is for personal use or smaller companies generating less than $200,000 annually. It also supports building games for more than 19 different platforms including Android, ISO, PC (Win, mac, Linux) and Web platform WebGL. For this project I have used the free licensing and focusing to build game for Android and Windows.

- ## Blender

  Blender is an open-source 3D application developed by both Blender Foundation and its community. In recent years with its upgrades, it has become a formidable alternative to any paid 3D modelling software. I have used blender to create the required characters, medical instruments and machines required by a hospital set up.

  I used a low-poly approach for the models as this provides a faster load-time. Low-poly mesh is a polygon mesh in 3D modelling that requires lesser number of polygons as opposed to High-poly models. They aren't as demanding as high-poly models and

wouldn't require high end computers to run it [29]. They are faster to create and often do not need texturing.

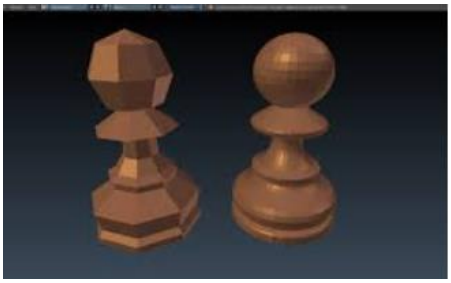


Fig 53: (left) Low-Poly, (right) High-Poly

- ## Mixamo

  Mixamo is a web-based service for animating 3D humanoid character models. It also allows to upload 3D model to online rigging services and rig them. Mixamo uses automatic model rigging service called Auto-Rigger, which applies machine learning to understand where the limbs of a 3D model are and to insert a skeleton/ rig and calculating skinning weights. This only takes up to 2 minutes. It also provides a wide variety of free to use animations which can be applied to the said model and downloaded easily [30].

- ## Music

  Music is one of the most latent features of a good video game which are usually not considered important when it comes to serious games. But I think that music really brings to the table the engaging factor that other elements of the game cannot. I have used some royalty-free music from [31].

- ## Audacity

  Sound not only has a big impact on the player experience, but research conducted [33] suggested that some of the levels in games are not explored possibly because the game sound was not balanced. I have used some Royalty free sound effects from [32] and used audio editor software Audacity for editing them as required.

- ## Firebase

  Firebase is a platform developed by google for mobile and web applications. Firebase's 1st product was its real-time database, an API that synchronizes data across IOS, Android and Web Devices and stores it in Firebase Cloud. [12] It is a cloud hosted No-SQL database and it ships with Mobile and Web SDKs, so you do not need to set up a server for saving the data.

  The Realtime database integrates with Firebase Authentication to provide simple and intuitive authentication. It allows the use of declarative security model to give access based on user identity or pattern matching.

  Firebase allows unlimited read and write operation to the real-time database and the spark plan allows the usage of almost all the services that we require for free up to 1GiB of stored data.

With the help of Firebase, I was able to save values like player experience, maximum experience of the level player is at, kill counts, diseases diagnosed correctly, and total number of diseases diagnosed.

All the resources required for doing the project are Freely and easily available. This will help the development of the game to be a smooth process.

CONCLUSION

Serious games are being considered as a formidable medium for education, especially in the past decade. Rather than just looking at games as a source of entertainment, people have started looking at it in this new light. I think that games are the best way to keep the users engaged and motivated to learn. Also, with the advancement in technology, right now we have the means to do it and it would be absurd not to use it.

In this research proposal, I presented a serious game for medical students. Especially in a field of health care, mistakes can be fatal. But in a game, we get to make as many mistakes as we want and learn from them and better ourselves for the real world, while keeping us motivated through engaging gameplay.

I believe the game to be a success and as inferred from the survey most of the people found the game to be fun and engaging. Most people also had positive responses when asked whether they found the game to be educational. I believe that the game can certainly help improve the learning of medical students. Also, from the survey a majority of the participants answered that they would still play the game if there was no research involved. This shows that the game had a positive impact on the gamers who tested it out.

DISCUSSION AND FUTURE SCOPE

Due to the time limit for the project, I was not able to create more levels or add other functions to the project, and there is lots of room for improvement in the project. Right now, we already have an online database in firebase. We could store a leader board to rank each player depending on their kill counts or the fraction of diseases diagnosed correctly per total number of diseases diagnosed.

Another way to improve the game would be to add a multiplayer system, where the players compete against each other on the time taken to diagnose the same disease. This can very well be accomplished using Photon multiplayer asset [42].

We can also add more models for playable characters or even set up character customization for the player to feel more involved in the game as they get to make the character they are going to play with.

All the data is saved locally on the player's device. If the player somehow loses the device or wants to play the game from a different device there is no way. However, building an online cloud data base could certainly solve this problem. This can certainly be done using Firebase. The player will be able to authenticate themselves with username and password. And log in to their account on the firebase Realtime database to store details like their level, health etc.

We could add more levels and add more compelling storylines for the player to play. This can add different dungeon environments to represent diseases that affect different body systems like respiratory, renal, digestive etc. We could add more enemies' characters (different models) representing different microbes like bacteria, virus etc.

Moreover, the game should be tested by healthcare professionals and their opinions would help improve the game more.

REFERENCES

[1] https://serious.gameclassification.com/EN/games/1213-Where-in-the-World-Is-Carmen-Sandiego/index.html

[2] https://www.visitoregon.com/the-oregon-trail-game-online/

[3] Halbrook, Yemaya & O'Donnell, Aisling & Msetfi, Rachel. (2019). When and How Video Games Can Be Good: A Review of the Positive Effects of Video Games on Well-Being. Perspectives on Psychological Science. 14. 1096-1104. 10.1177/1745691619863807.

[4] Longman H, O'Connor E, Obst P. The effect of social support derived from World of Warcraft on negative psychological symptoms. Cyberpsychol Behav. 2009 Oct;12(5):563-6. doi: 10.1089/cpb.2009.0001. PMID: 19817567.

[5] Video games as virtual teachers: Prosocial video game use by children and adolescents from different socioeconomic groups is associated with increased empathy and prosocial behaviour – Brian Harrington, Michael O'Connell - School of Psychology, University College Dublin, Newman Building, Belfield, Dublin 4, Ireland - Computers in Human Behaviour.

[6] https://www.nicswell.co.uk/health-news/does-gaming-improve-vision

[7] Maillot P, Perrot A, Hartley A, Do MC. The braking force in walking: age-related differences and improvement in older adults with exergame training. J Aging Phys Act. 2014 Oct;22(4):518-26. doi: 10.1123/japa.2013-0001. Epub 2013 Nov 13. PMID: 24231655.

[8] Peña-Miguel Noemí, Sedano Hoyuelos Máximo - Educational Games for Learning - Universal Journal of Educational Research 2(3): 230-238, 2014

[9] Serious games chapter 2013 in Handbook of Media Psychology - Fran Blumberg

[10] Sun, Hanqiu, Ricciardi, Francesco, De Paolis, Lucio Tommaso – 2014 - 2014/08/04 - A Comprehensive Review of Serious Games in Health Professions – 787968 – 2014

[11] S. H. Park, Y. S. Yoon, L. H. Kim, S. H. Lee, and M. Han, "Virtual knee joint replacement surgery system," in Proceedings of the International Conference on Geometric Modeling and Imaging (GMAI '07), pp. 79–84, Zurich, Switzerland, July 2007.

[12] J. Qin, Y. Chui, W. Pang, K. Choi, and P. Heng, "Learning blood management in orthopedic surgery through gameplay," IEEE Computer Graphics and Applications, vol. 30, no. 2, pp. 45–57, 2010.

[13] BreakAway, "Dental implant training simulation," http://www.breakawaygames.com/serious-games/solutions/healthcare

[14] C. Ito, A. V. Marinho Filho, M. Ito, M. M. Azevedo, and M. A. de Almeida, "Preliminary evaluation of a serious game for the dissemination and public awareness on preschool childrens oral health," Studies in Health Technology and Informatics, vol. 192, p. 1034, 2013.

[15] Barry Richards University of South Wales, Gareth Parsons, Virtual pain manager, http://vpm.glam.ac.uk/.

[16] Virtual Heroes, Va critical thinking, http://virtualheroes.com/ projects/va-critical-thinking.

[17] B. Richards, University of Glamorgan, Treforest, UK. Virtual ECG, http://ecg.glam.ac.uk/.

[18] K. Howell, "Games for health conference 2004: issues, trends, and needs unique to games for health," Cyberpsychology and Behavior, vol. 8, no. 2, pp. 103–109, 2005.

[19] BreakAway, "Code orange," http://www.breakawaygames.com/serious-games/solutions/homeland.

[20] S. N. Kurenov, W. W. Cance, B. Noel, and D. W. Mozingo, “Game-based mass casualty burn training,” Studies in Health Technology and Informatics, vol. 142, Article ID 142144, pp. 142– 144, 2009.
[21] A. Guillaume, Affaire birman, http://www.zippyware.com/.
[22] A. Guillaume, “Meli-m ´ elo glucique,” ´ http://www.zippyware .com/.
[23] L. A. Diehl, R. M. Souza, J. B. Alves et al., “Insuonline, a serious game to teach insulin therapy to primary care physicians: design of the game and a randomized controlled trial for educational validation,” Journal of Medical Internet Research, vol. 15, no. 1, article e5, 2013.
[24] V. Brezinka, “Treasure hunt—a serious game to support psychotherapeutic treatment of children,” Studies in Health Technology and Informatics, vol. 136, pp. 71–76, 2008.
[25] Imaginary SRL, ispectrum, http://www.i-maginary.it/
[26] https://gamingtrend.com/feature/reviews/in-need-of-life-support-bio-inc-redemption-review/
[27] http://playbioinc.com/
[28] Hussain, Afzal & Shakeel, Haad & Hussain, Faizan & Uddin, Nasir & Ghouri, Turab. (2020). Unity Game Development Engine: A Technical Survey. University of Sindh Journal of Information and Communication Technology. 4.
[29] https://joshocaoimh.com/advantages-and-disadvantages-of-low-poly-in-game-design#:~:text=The%20approach%20provides%20faster%20load,often%20no%20need%20for%20texturing.
[30] https://www.mixamo.com/#/
[31] https://soundcloud.com/royalty-free-audio-loops
[32] https://pixabay.com/
[33] Nael Ningalei Sebastian Wöhrman - The Impact of Sound on Player Experience - Faculty of Technology & Society Computer Science -Malmo University
[34] "SURGICAL INSTRUMENT" (https://skfb.ly/6xrEM) by ferofluid is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).
[35]https://assetstore.unity.com/packages/3d/props/horror-starter-pack-free-178413#publisher
[36]https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351#description
[37] https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631
[38] https://gist.github.com/ditzel/0257d74a7a04626efce7bd1f7a6cfaa0#file-fixedtouchfield-cs
[39] https://assetstore.unity.com/packages/vfx/particles/cartoon-fx-free-109565
[40] https://www.nhs.uk/
[41] https://bio-inc.fandom.com/wiki/Diseases
[42] https://assetstore.unity.com/packages/tools/network/pun-2-free-119922
[43] https://firebase.google.com/
[44] https://cseejira.essex.ac.uk/browse/B901105-1