



---

## D6.1

# PLATFORM IMPLEMENTATION PLAN

---

<b>Grant Agreement nr</b>	<b>856998</b>
<b>Project title</b>	Personalised recovery through a multi-user environment: Virtual Reality for Rehabilitation
<b>Project Acronym</b>	PRIME-VR2
<b>Start day of project (dur.)</b>	October 1 <sup>st</sup> 2019 (3 years)
<b>Document Reference</b>	PRIME-VR2_D_WP6_KRL_D6.1-Platform Implementation Plan
<b>Type of Report</b>	PU
<b>Document due date</b>	31/03/2020
<b>Actual date of delivery</b>	15/10/2020
<b>Leader</b>	KRL
<b>Responsible</b>	Gábor Vadász (KRL)
<b>Additional main contributors (Name, Partner)</b>	Robert Sarosi, KRL Robert Mooijman, CPD Anthony Demanuele, FSG
<b>Document status</b>	Final [Reviewed by Georgi Georgiev (UOO) and Yazan Barhoush (UOO)]



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 856998

This document is shared under the following Creative Commons License



## Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material

Under the following terms:

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- NonCommercial — You may not use the material for commercial purposes.
- ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Full terms can be found at <https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>

## Table of contents

<b>EXECUTIVE SUMMARY</b>	<b>9</b>
<b>1 INTRODUCTION</b>	<b>22</b>
1.1. TERMINOLOGY	22
<b>2 TECH STACK</b>	<b>25</b>
2.1. PROGRAMMING LANGUAGE	25
<b>3 UNITY SDK</b>	<b>26</b>
<b>3.1. INTRODUCTION</b>	<b>26</b>
3.1.1. PACKAGE	26
3.1.2. UNITY VERSION	27
3.1.3. PIPELINE COMPATIBILITY	27
3.1.4. SUBMODULES	27
<b>3.2. DOMAINS</b>	<b>27</b>
3.2.1. WEB	27
3.2.2. VR ECOSYSTEM	28
3.2.3. UNITY SDK	28
3.2.4. DISTRIBUTION PLATFORM	28
<b>3.3. FRAMEWORK</b>	<b>28</b>
3.3.1. BACK END	29
3.3.2. WEB PORTAL	29
3.3.3. PRIME-VR2 UNITY SDK	30
3.3.4. PLATFORM LOADER	30
3.3.5. VR USER INTERFACE	30
3.3.6. GAME	30
3.3.7. OPENXR	30
3.3.8. CONTROLLER	30
3.3.9. CUSTOM INPUT	31
3.3.10. INPUT MAPPING	31
<b>3.4. REQUIREMENTS</b>	<b>31</b>
3.4.1. SUPPORTED DEVICES	31
3.4.2. UNITY PLATFORM CONFIGURATION	31
3.4.3. BUILD TARGET ENVIRONMENTS	31
3.4.4. AUTOMATIC CONFIGURATION	31
<b>3.5. MODULES</b>	<b>32</b>
3.5.1. PRIMEVR2_SERVICE	32
3.5.2. PRIMEVR2_INPUT	32
3.5.3. PRIMEVR2_GAME	32
3.5.4. PRIMEVR2_UI	32
<b>3.6. API FUNCTIONS</b>	<b>32</b>
3.6.1. SERVICE	33
3.6.2. AUTHENTICATION	33
<b>3.7. SAMPLE SCENES</b>	<b>34</b>
3.7.1. SAMPLE SCENE INDEX	34
<b>3.8. PLATFORM LOADER</b>	<b>34</b>
<b>3.9. REALTIME ASSISTANCE</b>	<b>34</b>
<b>3.10. GAME STRUCTURE</b>	<b>35</b>

3.10.1.	GAME	35
3.10.2.	ENVIRONMENT	35
3.10.3.	EXERCISE	35
3.10.4.	EXERCISE SESSION	35
3.10.5.	GAME SESSION	35
3.10.6.	ACTIVITY	35
<b>3.11.</b>	<b>SCORE SYSTEM</b>	<b>35</b>
3.11.1.	EXERCISE METRICS	35
3.11.2.	SCORE CALCULATION	36
3.11.3.	USER CALIBRATION	36
<b>3.12.</b>	<b>LEADERBOARDS</b>	<b>36</b>
<b>3.13.</b>	<b>ACHIEVEMENTS</b>	<b>36</b>
<b>3.14.</b>	<b>ANALYTICS</b>	<b>37</b>
<b>3.15.</b>	<b>ACCESSIBILITY TOOLS (VR)</b>	<b>37</b>
3.15.1.	MOTION COMPENSATION	37
<b>4</b>	<b>WEB PLATFORM</b>	<b>38</b>
<b>4.1.</b>	<b>INTRODUCTION</b>	<b>38</b>
<b>4.2.</b>	<b>ARCHITECTURE</b>	<b>38</b>
<b>4.3.</b>	<b>USER TYPES</b>	<b>40</b>
4.3.1.	PERMISSIONS	40
<b>4.4.</b>	<b>USER MANAGEMENT</b>	<b>43</b>
4.4.1.	USER LIST	43
4.4.2.	CREATE USER	44
4.4.3.	NEW USER FLOW	46
4.4.4.	PATIENT OVERVIEW	46
4.4.5.	EDIT USER	47
4.4.6.	DISABLE/ENABLE USER	47
4.4.7.	DELETE USER	48
4.4.8.	EDIT OWN USER PROFILE	48
<b>4.5.</b>	<b>LOGIN</b>	<b>49</b>
4.5.1.	FIRST LOGIN	49
4.5.2.	FORGOTTEN PASSWORD	50
<b>4.6.</b>	<b>HOSPITAL MANAGEMENT</b>	<b>50</b>
4.6.1.	HOSPITAL LIST	50
4.6.2.	CREATE HOSPITAL	51
4.6.3.	MANAGE HOSPITAL USERS	51
4.6.4.	EDIT HOSPITAL	51
4.6.5.	DISABLE/ENABLE HOSPITAL	51
4.6.6.	EDIT OWN HOSPITAL PROFILE	52
<b>4.7.</b>	<b>GAMES</b>	<b>52</b>
4.7.1.	GAMES LIST	52
4.7.2.	CREATE GAME	53
4.7.3.	EDIT GAME	54
4.7.4.	DISABLE/ENABLE GAME GLOBALLY	54
<b>4.8.</b>	<b>PATIENT OVERVIEW</b>	<b>54</b>
4.8.1.	LIST OF GAMES	55
4.8.2.	GOAL REVIEW	55
4.8.3.	ACTIVITY FEED	56
4.8.4.	GAME DEFINITION	56
4.8.5.	CREATE SESSION	57
4.8.6.	EDIT SESSION	57
4.8.7.	DELETE SESSION	57
4.8.8.	LEADERBOARD	58



<b>4.9. MESSAGING</b>	<b>58</b>
4.9.1. MESSAGE ICON	58
4.9.2. MESSAGE THREAD VIEW	58
4.9.3. REAL-TIME DELIVERY	58
4.9.4. COMMENTING ON GAMEPLAY SESSIONS	59
<b>4.10. NOTIFICATIONS</b>	<b>60</b>
<b>5 GAMEPLAY API</b>	<b>61</b>
5.1. AUTHENTICATION	61
5.2. CONFIGURATION ENDPOINT	61
5.3. SESSION ENDPOINT	61
<b>6 ACHIEVEMENTS</b>	<b>62</b>
<b>7 LEADERBOARDS AND CHARTS</b>	<b>63</b>
<b>8 PERFORMANCE METRICS</b>	<b>67</b>
<b>9 VALIDATION CRITERIA</b>	<b>67</b>
<b>10 CI/CD</b>	<b>68</b>
<b>11 CONCLUSION</b>	<b>68</b>
<b>APPENDIX 1.</b>	<b>70</b>
<b>12 USER INTERFACE MOCK-UPS</b>	<b>70</b>
<b>APPENDIX 2.</b>	<b>75</b>
<b>13 CODING GUIDELINES</b>	<b>75</b>
13.1. INTRODUCTION	75
13.2. GUIDING PRINCIPLES	75
13.3. THE RUNDOWN	75
13.4. GENERAL GUIDELINES	75
13.4.1. FILE LAYOUT	75
13.4.2. USING DIRECTIVES	76
13.4.3. DECLARING TYPES	77
13.5. MEMBER DECLARATIONS	78
13.5.1. METHODS	79
13.5.2. PROPERTIES	79
13.5.3. TYPE INFERENCE	80
13.5.4. OBJECT AND COLLECTION INITIALIZERS	80
13.5.5. INDENTATION	81
13.5.6. WHERE TO PUT SPACES[1]	81
13.5.7. WHERE TO PUT BRACES	83
13.5.8. LONG ARGUMENT LISTS	87
13.5.9. CASING	87
13.5.10. INSTANCE FIELDS	88
13.5.11. <i>THIS</i>	88
<b>14 CODE COMMENTING GUIDELINES</b>	<b>91</b>
14.1. SIMPLE COMMENTS	91
14.2. MULTILINE COMMENTS	91

<b>14.3. COMMENTING OUT CODE</b>	<b>92</b>
<b>15 NAMING GUIDELINES</b>	<b>92</b>
<b>16 LAMBDA</b>	<b>100</b>
<b>17 COMMIT MESSAGE CONVENTIONS</b>	<b>103</b>
17.1. GOALS	103
17.2. GENERATING CHANGELOG.MD	103
17.2.1. RECOGNIZING UNIMPORTANT COMMITS	103
17.2.2. PROVIDE MORE INFORMATION WHEN BROWSING THE HISTORY	103
17.3. FORMAT OF THE COMMIT MESSAGE	104
17.3.1. SUBJECT LINE	104
17.3.2. MESSAGE BODY	104
17.3.3. MESSAGE FOOTER	105
17.4. EXAMPLES	106
<b>APPENDIX 3.</b>	<b>108</b>
<b>ENVIRONMENTS CHECKLIST</b>	<b>108</b>
<b>APPENDIX 4</b>	<b>110</b>
<b>PRIMEVR2 ENVIRONMENTS</b>	<b>110</b>
<b>1. GARDEN/ GREENHOUSE ENVIRONMENT</b>	<b>111</b>
<b>2. SUPERMARKET ENVIRONMENT</b>	<b>127</b>
<b>3. CRAFTS WORKSHOP ENVIRONMENT</b>	<b>139</b>
17.5. VISUAL	140
17.6. MODALITY	141
17.7. ASSISTANCE	141
17.8. SOCIAL	141
17.9. SUPPORTED ACTIVITIES	142
<b>4. UNDERWATER ENVIRONMENT</b>	<b>143</b>
17.10. VISUAL	143
17.11. MODALITY	144
17.12. ASSISTANCE	144
17.13. SOCIAL	144
17.14. SUPPORTED ACTIVITIES	144
<b>5. SPACE ENGINEER ENVIRONMENT</b>	<b>146</b>
17.15. VISUAL	146
17.16. MODALITY	147
17.17. ASSISTANCE	149
17.18. SOCIAL	149
17.19. SUPPORTED ACTIVITIES	150
<b>6. COOKING SHOW ENVIRONMENT</b>	<b>151</b>
17.20. VISUAL	151
17.21. MODALITY	152

17.22. ASSISTANCE	152
17.23. SOCIAL	152
17.24. SUPPORTED ACTIVITIES	153
<b><u>7. CAR ENVIRONMENT</u></b>	<b><u>154</u></b>
<b><u>8. MUSIC ROOM ENVIRONMENT</u></b>	<b><u>166</u></b>
17.25. INSTRUMENT BREAKDOWN	170
17.26.	172
17.27. GAME REFERENCES	174
<b><u>9. OFFICE ENVIRONMENT</u></b>	<b><u>175</u></b>
<b><u>10. BAR ENVIRONMENT</u></b>	<b><u>179</u></b>
<b><u>APPENDIX 5</u></b>	<b><u>183</u></b>
<b><u>CONTROLLER INTERACTION MATRIX</u></b>	<b><u>183</u></b>

## VERSION HISTORY

<b>COMMENTS</b>	<b>RESPONSIBLE</b>	<b>VERSION</b>	<b>DATE</b>
<b>First draft</b>	Robert Sarosi	0.8	20/02/2020
<b>Second draft</b>	Robert Sarosi	0.9	09/03/2020
<b>Peer reviewed</b>	Georgi Georgiev and Yazan Barhoush	1.0	23/03/2020
<b>Updates based on Peer review Extended API documentation for Web Platform Backend</b>	Robert Sarosi	1.1	26/03/2020
<b>Removing high-risk security information + add notice</b>	Gabor Vadasz	1.2	14/04/2020
<b>Added more information about environments, games and selection protocols with added references to D2.1.</b>	Anthony Demanuele	1.3	15/10/2021
<b>Added more information about web-portal with references to D2.1 and CI/CD, system architecture section</b>	Gabor Vadasz	1.4	28/10/2021
<b>Peer reviewed</b>	Georgi Georgiev and Yazan Barhoush	2.0	26/10/2020
<b>Updated based on peer review</b>	Gabor Vadasz	2.1	28/10/2021
<b>Released CC version</b>	Gabor Vadasz	2.1	31/10/2021

## **EXECUTIVE SUMMARY**

The Implementation Plan contains all the technical and functional information, and documentation what is needed to implement the VR-HABIT platform. The rules and guidelines that are included in the implementation plan guarantees that the environment and the quality stays consistent in the whole project. Because technology is likely to change during the project, this document will be continuously updated.

The main goal of the Platform Implementation Plan is to identify all the needed business logics and the functionalities of the VR-HABIT Platform. The deliverable is the result of the discussion of all the technical challenges of the project made by WP6 members. Discussions were extremely important to define the initial list of business and technical requirements for the overall Platform, and for each single component. Three Platform components have been identified and detailed in the document: Web Platform, VR Ecosystem, and Rehabilitation Games.

### ***Web Platform***

The web Platform consists of a client portal and back end that allows patients and therapists to log and review their data and progression statistics within any of the PRIME-VR2 implemented games. WP6 used questionnaire and 1 on 1 interview methodology to explore the needed functionality and organized 7 focus group meetings and interviews to collect the required information. All these data have been collected and organized by the KRL team and turned it into a functional and a brief technical documentation. This documentation is one of the main parts of the Platform Implementation Plan.

### ***VR Ecosystem***

The VR ecosystem consists of hardware and software components included in the VRHAB-IT platform; three types of custom-made controllers are recognized within an existing commercial virtual reality hardware ecosystem. Sensory input will be mapped to existing controllers or functionality will be extended via additional drivers. The implementation of VR ecosystem modules (like the games) will be based on the PRIME-VR2 Unity SDK. The functionality has been abstracted into a set of separate modules. All modules will include code samples and example scenes in order to define a technological standard.

### ***Rehabilitation Games***

The game implementation is based on the Unity game engine and the PRIME-VR2 Unity SDK. Unity is a popular game engine that integrates with all popular VR hardware platforms and is also the main development tool used by key partners in the project, with valuable accumulated expertise. The current version in use within the project is 2019.4.5f1 but this is bound to be updated frequently over the course of the project. Each game will be built and uploaded as a separate executable so once submitted and reviewed, the latest version of each game will be made available through a game loader interface made available as part of the PRIME-VR2 Unity SDK.

The games are made up of 3 dimensional environments built with tools such as Autodesk Maya, Autodesk 3Ds Max and textured with industry standard tools such as Substance painter and Adobe Photoshop. During the last few months, the team has had many discussions with other Consortium members to help outline the most sought-after environments that would encourage patients to stay focused on their goal. A pattern quickly emerged during these brainstorming sessions. The chosen environments can be categorised into two main categories: **Everyday life environments** and **Aspiration or fantastic environments**.

<b>Everyday life</b>	<b>Aspiration or fantastic</b>
<ol style="list-style-type: none"> <li>1. Crafts Workshop</li> <li>2. Greenhouse</li> <li>3. Cooking show</li> <li>4. Music room</li> <li>5. Office</li> <li>6. Bar</li> <li>7. Supermarket</li> <li>8. Gym</li> <li>9. Tennis court</li> <li>10. Football pitch</li> </ol>	<ol style="list-style-type: none"> <li>A. Space travel</li> <li>B. Underwater exploration</li> <li>C. Peaceful natural environment</li> <li>D. River rafting</li> <li>E. Mountain climbing</li> </ol>

The first environments to be prototyped were the real-life environments that patients will automatically be familiar with. Different level of detail and styles were used during concepting stage so these varied designs can be used during shortlisting stage as part of the established protocol for the selection of the final three to five environments. In addition to this, user testing and iterative design will determine the most successful environments that will make it to the end of the project. The protocol for selecting the final environments is outlined in more detail in D7.1.

The action items in each of the environments will have high contrast with backgrounds to make call to actions as clear as possible. In the **everyday life environments**, we try to mimic simplified real-life scenarios in contrast with the **aspiration or fantastic environments** where we create experiences that are impossible or very hard to access in real-life. Based on the requirements of D2.1, it is important to engage people in activities what are meaningful and valued and help collaboration and competition with other people.

D7.2 Persona Report highlights the three different personae targeted by VRHAB-IT; Hyperkinetic movement disorders in Young People (GDIH), Musculoskeletal injury (STJH) and Stroke (KNRC). The target audience for each group tends to vary from young kids for GDIH to seniors at KNRC and anything in between for STJH. For this reason, the shortlisted list of ten environments tries to encompass each of these requirements into reusable environments which cater for at least one or more of these target groups.

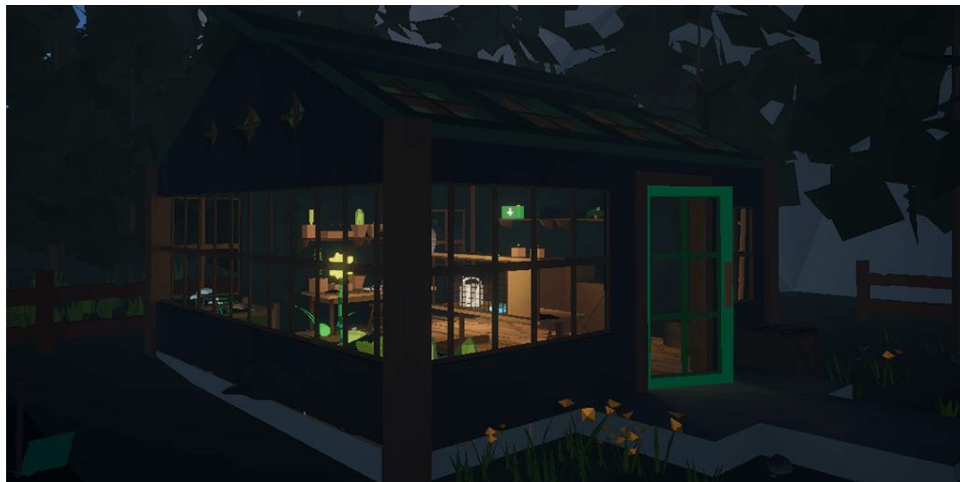
Whilst the final selection process is still not in progress, we envisage that the younger age groups will prefer aspirational or fantastic environments such as Space Travel or Underwater exploration and some of the real life environments which allow for freedom of expression such



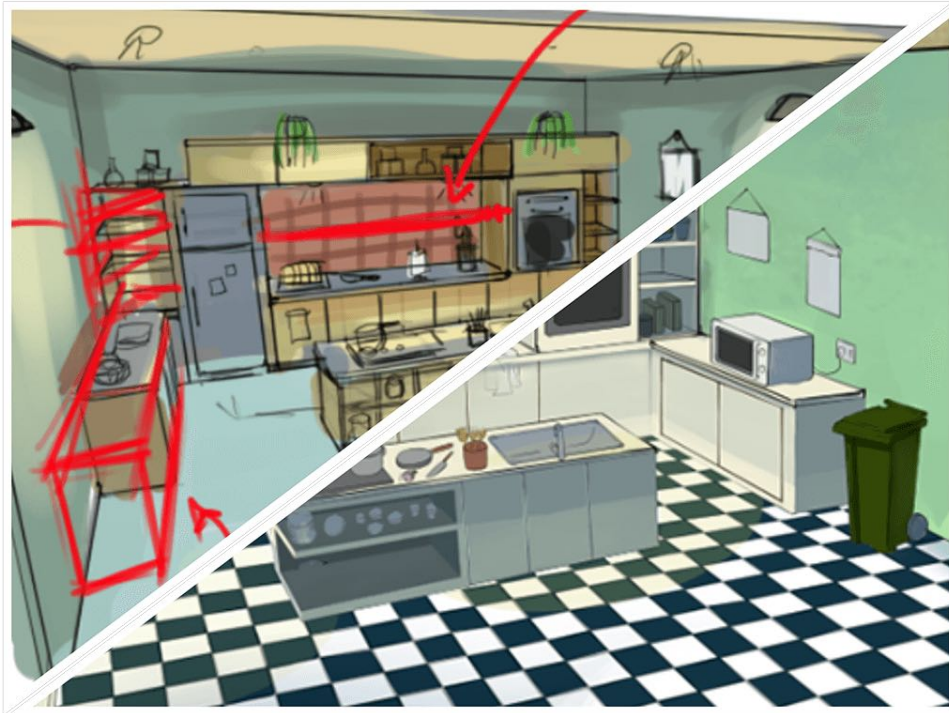




## 2. Greenhouse concept and 3D environment



### 3. Kitchen/Cooking show concept and 3D environment



#### 4. Music room/theatre concept and 3D environment



#### 5. Office concept and 3D environment







## 7. Supermarket concept

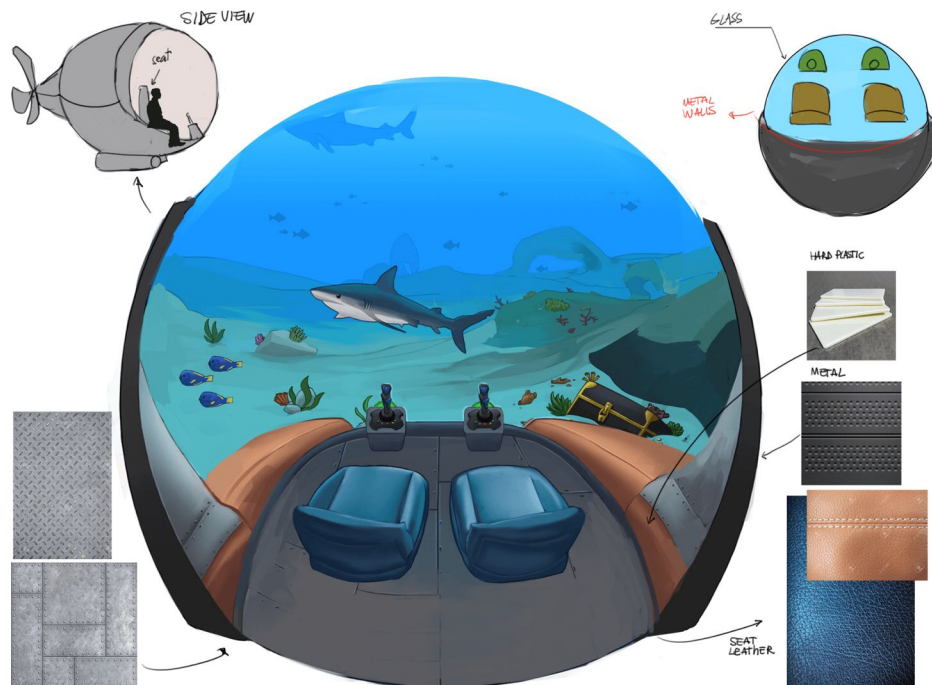


### A. Horticulture room in Space concept and 3D environment





## B. Underwater concept



For each of the selected environments, the team ran through the checklist to design an environment that fits the requirements of VRHAB-IT. The full environment specifications can be found in **Appendix 4 (PrimeVR2 environments)**. For each of these we outlined 5 aspects:

1. **Visuals:** This section establishes a theme and sets the scene for various objects and other virtual entities around the Player. These include static environment objects as well as interactive props. Each layout is designed to provide a space where players can feel safe and in-control while reducing unnecessary movement which can be challenging considering the spectrum of patients that will inhabit such space. Environments range from common day-to-day locales to fictional ones as we believe it gives us greater flexibility to reach a larger number of patients.
2. **Modality:** Provides a definition on how the environments interface with the various activity stations at play and for each it lists how the patient will be positioned to play.



Given the physical impairments of some of the patient classes we have opted where possible to support the sitting down position. This will give us the possibility of re-using the same environment across the whole spectrum of patients in VRHAB-IT.

3. **Assistance:** Patients may be unable to perform specific tasks. For this reason, we have listed several possibilities within the game space of how these tasks can be still be overcome with the assistance of a third-party.
4. **Social:** Different activities provide different social interaction potential. In this section we outline any potential ways in which the patient, their actions or even their creations can be used to interact socially with other players.
5. **Supported Activities:** Most importantly in this section we give a sample of the activities that can be performed by the patient to drive the gameplay. Here we define each activity while clearly outlining how the patient is supposed to interact it and thus establishing the requirements for the custom controller. These activities together with the aspects are designed to engage the patient in a relaxing yet challenging, environment to drive the rehabilitation process.

Once the three to five environments are selected, an array of games will be designed in detail. The expected outcome for each of these games includes but is not limited to exercises that enhance reach, range of movement, consistent movement, accuracy, strength and stamina. We will briefly describe each of these exercises and then follow up with a few examples of how these could be used in some of the environments mentioned above while making sure they make best use of the custom controllers being built as part of this project and confirming to the ICF framework.

**Appendix 5 (Controller interaction matrix)** sets out the language between WP2, WP3, WP4, WP5 and WP6 in terms of the core definitions of what the controller is required to do and how the custom controller capabilities will be used within the games.

The International Classification of Functioning, Disability and Health (ICF) framework developed by the WHO defines a simplified framework to approach patient care with a focus on the positive abilities of each individual patient. The ICF is made up of three components:

1. Body functions and structures
2. Activities
3. Participation

**Appendix 4 (PrimeVR2 environments)** sets the groundwork for the game design work in each of the ten concepted environments and how they map to the ICF framework. Over and above these general guidelines, D2.1 outlines additional requirements for each of the developed games including:

1. Competitive/collaborative
2. Reflect real activities
3. Difficulty levels
4. Challenging but maintain motivation

5. Enjoyable
6. Engaging
7. Therapeutic goals
8. Bonding/ team spirit
9. Exercise

A strict protocol is also being designed on how to score each game or task activity in a session. WP2 and WP7 recommended the Melbourne Assessment scoring methodology. This is made up of 3 broad categories; range of motion, fluency and strength. **Reach and Range** of movement applies to all patients and the parameter in this game would be how far a hand can twist, or how far an arm can reach. **Consistency** and **accuracy** games are targeted especially for dystonia patients where we monitor their ability to achieve a correct result consistently and accurately over time. **Strength** and **stamina** games apply to sports and stroke patients trying to rebuild their muscle tissue by doing an activity with forced feedback with varying durations.

We will use the first set of games to explain how such an activity could be modelled differently according to the environment with limited changes. Let's assume a range of movement game for a patient with tennis elbow where the user is asked to try and get to stretch and increase the elbow movement. At the gym, this movement could take shape in the form of the player sitting on a bench while lifting a dumbbell. At the river rafting environment, we could re-use a similar player position and movement but this time the player is manning the rows to navigate safely down the river. Similarly, the player may be asked to pick up heavy objects from the seabed while underwater diving.

Each of these games will have to be validated before it is used on patients. The first validation pass will be done internally within WP6 to ensure correct input and output behaviour. This will in turn be reviewed by the living labs that are the experts in the field. With ethics approved and patients willing to test the games, we will also strive to get the games validated and fine-tuned at this level as well.

Anonymized metrics will also be added to different key events within each game so we can monitor and fine tune different parts of the games once these are being actively used. These metrics will include but are not limited to frequency of use, fail rate, results progression over time and session times. Through these data points we can do statistical analysis and find common pain points and address them accordingly.

WP6 has also identified and considered major data privacy and security threats. These questions mainly focused on GDPR compliance and highly-sensitive information about health status. After a thorough discussion WP6 members came up with the following solution: the system will keep anonymized data only in the Web Portal, and all the additional data will remain in the Clinic/Hospital database. Any direct connection between the Web Platform and the Clinic/Hospital systems and databases are not planned, because every Clinic/Hospital system is different and has its own datasets and tools. WP6 members concluded that creating such a close integration of the two systems would introduce high risk to the project. This is why WP6 decided to use only the Patient ID as a unique identification number and a nickname to identify the patient. The only personal ID that will be stored is the email address, because the system needs to communicate with the patient.



## **Results achieved in WP6 at M12**

- Submission of deliverable D6.1 Platform Implementation Plan
- Communication and management software and routes were discussed and agreed on
- Organization of 7 focus group meetings to collect users' stories
- Definition of the requirements of the PRIME-VR2 Web Platform
- Specification of the PRIME-VR2 Web Platform and its functionalities
- Definition of the PRIME-VR2 Web Platform design mock-ups, including the modules and communications, with the definition of use cases and scenarios of interaction among the components
- Organization of 22 WP6 meetings for discussing the platform implementation plan
- Creation of Web Platform functional and technical documentation
- Introduction of coding Guidelines to the team to make proposed guidelines for the coding
- The Unity SDK functionality has been abstracted into a set of separate modules that will be used as technological standard (platform loader, service module, game module, controller input module)
- Discussion and filing of Project Risk Assessments

Notice:

**Due some high-risk security information, we changed these to “obscured” expression.**

# 1 INTRODUCTION

## 1.1. Terminology

---

Term	Meaning
Game Engine	A game engine is a software-development environment designed for people to build video games. Developers use game engines to construct games for consoles, mobile devices, and personal computers.
Unity3D	Unity is a cross-platform game engine.
Unity SDK	A software development kit (SDK) is a collection of software development tools in one installable package. They ease creation of applications by having compiler, debugger and perhaps a software framework.
Web Portal	Web Portal is a web application that coordinates the activity of hospital supervisors, doctors and patients in order to achieve successful VR therapy outcomes.
API	An Application Programming Interface (aka API) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. An API may be for a web-based system, operating system, database system, computer hardware, or software library.
API Key	A special string that is functioning as a secret password between the API and the API caller. This secures the API so unauthorized calls wouldn't be processed and can help to prevent data leakage.
REST	Representational State Transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services or API. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet.
JSON	JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is

	<p>easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.</p>
User	<p>User is a person who has some level of access in the system. The User interacts with the system and get results based on permissions.</p>
User Account / Profile	<p>The User Account / Profile contains personal (eg.: Name, Email) and non-personal information (eg.: UserID, Group, Permissions) about the User.</p>
Group	<p>A Group is a logical set of Users. It has a Group Name and a Group can have a special set of permissions what defaults to all Users who are included in the Group.</p>
Permission	<p>All available operations in the system and whether different user types can perform them.</p>
Authentication	<p>The Authentication is the process when the system checks the User's credentials. This is a gatekeeper to secure the system from unwanted access. This is mostly happening when the User logs in or getting back to the system after a while.</p>
Authorization	<p>The Authorization is the process when the system checks the User's permissions. This is a gatekeeper to secure the system from unwanted access. This happening at every API call.</p>
JWT	<p>JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties (mostly between client and backend or backend and backend communications). Built on top of JSON.</p>

C#	C# is a general-purpose, multi-paradigm programming language encompassing strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented, and component-oriented programming disciplines. Developed and maintained by Microsoft. Developers using C# to develop a Unity3D application.
NodeJS	Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.
Container	A Container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
Docker	Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.
Kubernetes	Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management. It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation.
Version Control System	A component of software configuration management, version control, also known as revision control or source control is the management of changes to documents, computer programs, large web sites, and other collections of information.

## 2 TECH STACK

### 2.1. Programming Language

---

There is a wide variety of programming language available today. Older and more robust ones like C / C++ and newer ones like C#, NodeJS, Go which provides better solutions for today's coding problems (async, multi-threading, heavy network communications). There is no good or bad programming language, but some are better than others when facing specific challenges. With a monolithic approach, the developer chooses only one programming language which is used exclusively in the whole software package. Another hallmark of monolithic approaches is that all is contained in one software package. In polyglot programming the developer has the freedom to write different parts of the software in different languages. This way the developer can leverage most of the benefits of each language. A danger of polyglot approaches is that the codebase becomes fragmented in many different languages and requires a variety of skillsets to maintain.

In the Prime-VR2 project we choose polyglot approach but limit the number of languages to a small set: C#, NodeJS. For certain heavy data and network traffic handling, using the language Go language is proposed.

C# is developed and maintained by Microsoft. Because it's wide adoption in education it's easy to find a C# developer in any seniority level. Also the chosen game engine, Unity3D using C# as its main programming language.

NodeJS is a free, open-source language built on top of Javascript. One of the ideas behind the language is to reuse the front-end web knowledge when developing the backend. Most of the web programmers know how to code in Javascript. This makes it easier to find developers and maintain the code base. Javascript also has a huge online community, which makes finding answers and solutions easy.

Go is an open source, compiled and statically typed programming language developed and maintained by Google. The main goal is to reach the C++ performance but without the difficulties what C++ could mean. Go is only a few years old language but it is gaining popularity. It is offering a simple syntax, a fast compiler and supports concurrent programming from the very beginning.

## 3 UNITY SDK

### 3.1. Introduction

This section describes specifications for content and application development made for PRIME-VR2 by using the PRIME-VR2 Unity software development kit (SDK), within the Unity 3D game development engine (Unity).

This section provides guidelines for application development within the PRIME-VR2 ecosystem and elaborates on all functionality within the framework capabilities.

This document aims to be a complete overview of the SDK. Please consider this document as a work in progress as not all details within the project's deliverables have been defined yet and new functionality will evolve during development.

#### 3.1.1. Package

The SDK is provided in the format of a Unity Package. After importing, developers can see the following directories:

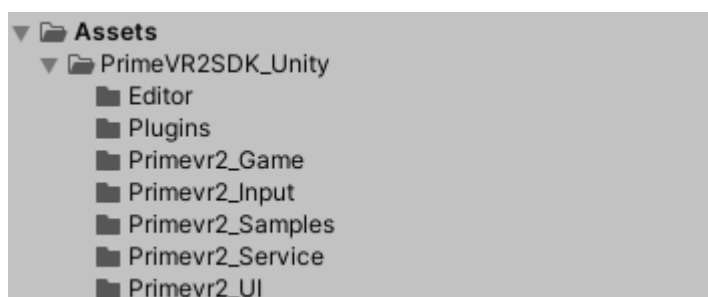


Figure 3.1 Directory structure of the SDK plugin

Each sub-directory under Assets>PrimeVR2SDK\_Unity corresponds to a function in the SDK. The PrimeVR2\_Samples directory provides reference scenes to display practical use of each of the SDK functionalities.

- *Editor* contains scripts that extend functionality of the Unity editor required by components within the SDK.
- *Plugins* contains scripts needed for low level communication with external platforms.
- *Primevr2\_Game* contains game logic related functions.
- *Primevr2\_Input* contains controller input related functions.
- *Primevr2\_Samples* provides a selection of sample scenes to display the use of the SDK
- *Primevr2\_Service* contains any service-related functionality, like authorization and data communication with the PRIME VR2 back end.
- *Primevr2\_UI* contains all user interface related functionality.

### 3.1.2. Unity version

The SDK is being developed using the latest release of Unity (2019.4.5f1 as of writing) and will be updated when future stable versions have been released.

### 3.1.3. Pipeline compatibility

The SDK is built using the standard render pipeline within Unity and is not dependent on a scriptable render pipeline like the Universal or High Definition Render Pipeline; however, it aims to be fully extendable to implementation in any of these described pipelines, depending on the developer's needs.

### 3.1.4. Submodules

Dependencies and the requirements of submodules or packages will be described here unless remaining non-existent.

## 3.2. Domains

---

Developing content for PRIME-VR2 requires an understanding of the PRIME-VR2 ecosystem. The system consists of several domains that each serve a different purpose. Figure 1.2 displays the differentiation between these domains; Web, Unity, VR and Distribution:-

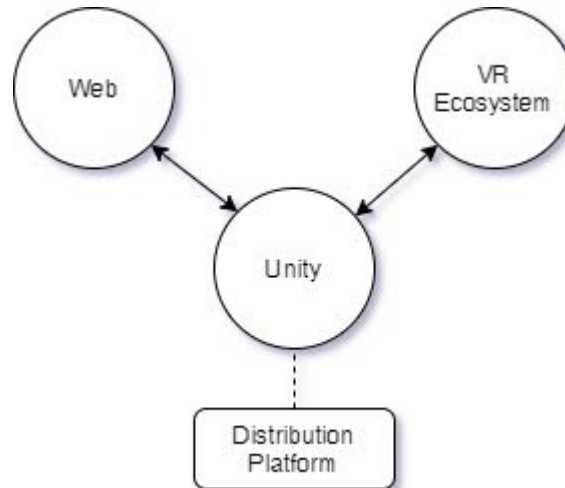


Figure 3.2 Framework domains

### 3.2.1. Web

The web domain consists of a client portal and back end that allows patients and therapists to log in and review their data and progression statistics within any of the PRIME-VR2 implemented games.

### **3.2.2. VR Ecosystem**

Three types of custom-made controllers are recognized within an existing commercial virtual reality hardware ecosystem, such as HTC Vive. Sensory input will be mapped to existing controllers or functionality will be extended via additional drivers.

### **3.2.3. Unity SDK**

Development of new game implementations is done using the Unity game engine and the PRIME-VR2 Unity SDK. The resulting applications, once submitted and reviewed, will be made available through a game loader interface.

### **3.2.4. Distribution Platform**

Since our main application is in fact a 'game loader', it in itself serves as a means to distribute games that have been created for the PRIME-VR2 Platform.

We are currently looking into alternatives of hosting the initial platform download.

## **3.3. Framework**

---

To understand the functionality of the SDK we should first look at the surrounding components, as the SDK serves as a mediator between these different end points. This section describes these components.



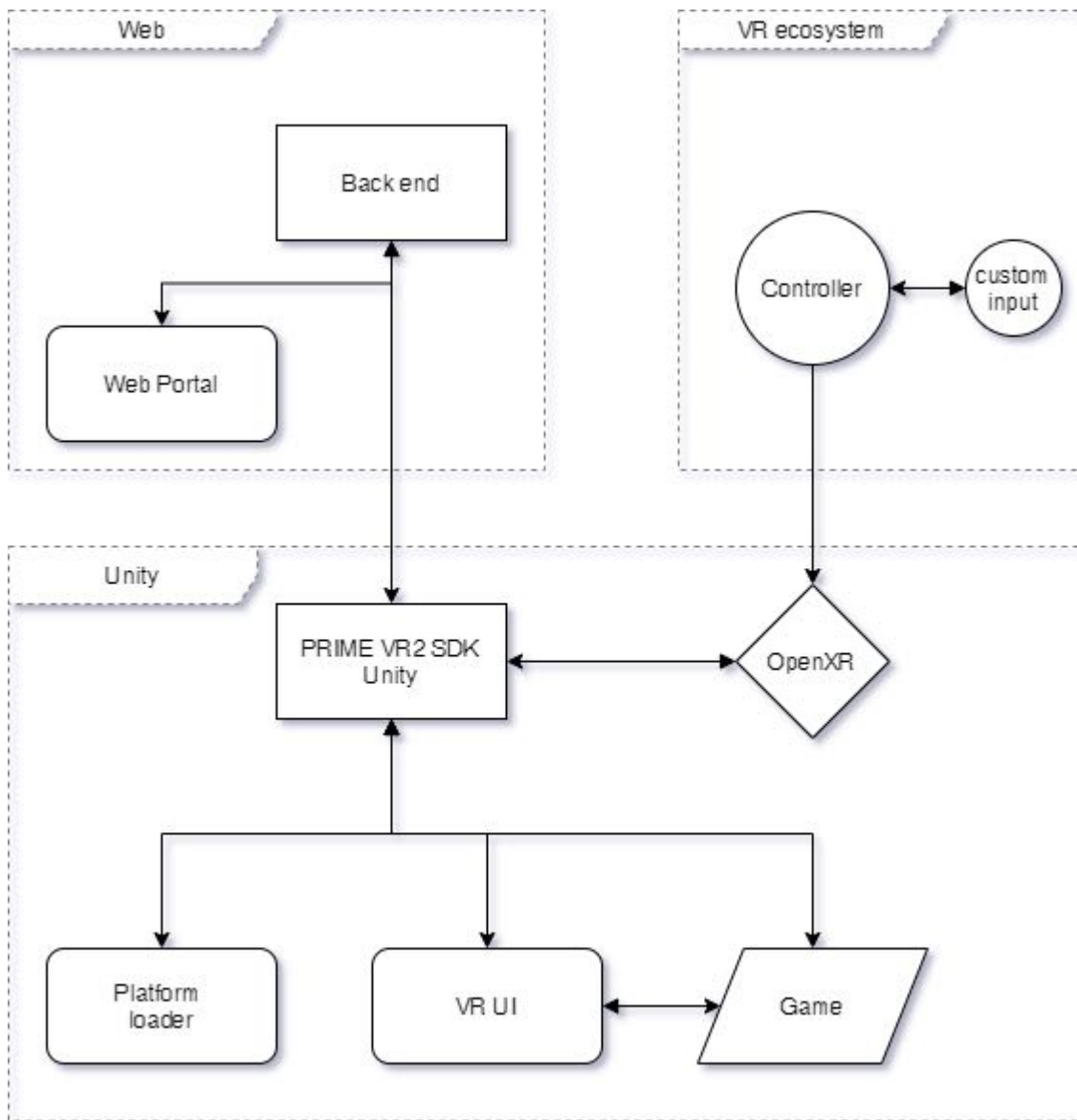


Figure 1.3 PRIME-VR2 high level framework

### 3.3.1. Back end

The back end serves a database and provides an API to handle various interactions consisting of but not limited to: user management, login authentication, storing and retrieving game related statistics and serving patient/therapist related data.

### 3.3.2. Web Portal

A front-end web portal for both patients and therapists to serve user profile settings, progress and statistics.

### **3.3.3. PRIME-VR2 Unity SDK**

The SDK implemented into Unity provides sample components and building blocks to establish communication between back end, user input and real-time game progression statistics. The SDK is the starting point for developers desiring to create new content for the PRIME-VR2 platform.

### **3.3.4. Platform loader**

A desktop entry point for any user willing to download and start new game.

The loader will provide the user with a login screen. From within this environment, the user can choose which game to start before they enter the complexity of the virtual environment.

### **3.3.5. VR User Interface**

The visual user interface in virtual reality with which a user will interact with further game related statistics and progression. The SDK will provide template examples displaying best practice interaction design made with the limited physical capabilities of the patient in mind.

### **3.3.6. Game**

Implementation of game mechanics and visual representation of all game related components, making use of the PRIME-VR API as a bridge between user, controller input and game statistic related back end.

### **3.3.7. OpenXR**

OpenXR is an open, royalty-free standard for access to virtual reality platforms and devices.

XR is an umbrella term covering a wide span of different virtual practices like augmented reality, virtual reality and mixed reality.

OpenVR is a software development kit and application programming interface developed by Valve for supporting the SteamVR (HTC Vive) and other virtual reality headset devices. The SteamVR platform uses it as the default application programming interface and runtime. It serves as the interface between the VR hardware and software and is implemented by SteamVR.

Note that as of Unity 2020.1 the OpenVR plugin will be deprecated within Unity and replaced by the new standard called Unity XR.

### **3.3.8. Controller**

During the course of the PRIME-VR2 project, three types of custom controller hardware will be developed. All of which will be mapped and integrated into the VR ecosystem.

### 3.3.9. Custom Input

Depending on the outcome of the various user profiles defined within the PRIME-VR2 project, a multitude of sensors will be added to the controllers to measure a range of user input activity.

### 3.3.10. Input mapping

Custom input will be mapped to existing hardware using Unity Input System v1.0, unless the capabilities of the hardware is found insufficient, in which case the additional hardware will be made available via a controller driver extension or replacement.

## 3.4. Requirements

---

This section describes system and environmental setup and requirements mandatory to be able to use the SDK.

### 3.4.1. Supported devices

Valve Index  
PRIME-VR2 custom controllers

### 3.4.2. Unity platform configuration

Table 3.4 provides an overview of the required Unity platform configuration. These will be expanded with additional packages and plugins that will be used over the course of development.

Unity Version	2019.1 - latest
API Compatibility Level	.Net Standard 4.0
Scripting Backend	Mono
Active Input Handling	Input System Package

*Table 3.4*

### 3.4.3. Build target environment

The target environment is Windows 10.

### 3.4.4. Automatic Configuration

Settings that are mandatory for a proper functioning of the SDK will be presented through an additional editor menu, allowing a quick setup for developers.

## 3.5. Modules

---

The SDK currently defines four main modules:

- A Service module, which makes the server-side API accessible to developers and wraps complex systems like user authentication and data validation.
- An Input module, which embeds all complexities considering the custom controller connectivity states and input logic.
- A Game module, which provides building blocks for creating games.
- A UI module, which provides building blocks for creating an accessible user interface, to enforce a consistent mechanism and interaction design that is default to the PRIME-VR2 platform integrations.

The following list describes the functionality of each of the modules.

### 3.5.1. Primevr2\_Service

- Establishing a server connection
- User Authentication
- Automatic sign in
- Retrieving and storing user profile data
- Retrieving and storing game progress related data
- Retrieving and storing game resulting statistics
- Data caching and synchronization
- Data serialization

### 3.5.2. Primevr2\_Input

- Retrieving controller status
- Wrapping complex input signal algorithms
- Calibration
- Measuring pressure, velocity, speed, inertia

### 3.5.3. Primevr2\_Game

- Exercise metrics
- Analytics
- Achievements
- Usability Tools

### 3.5.4. Primevr2\_UI

- Event pipeline for displaying data updates
- Handling user input
- Mock-up interface providing basic interaction guidelines

## 3.6. API functions

---

This section provides an overview of the available API calls within the Unity SDK. Each submodule has their own set of calls. Due to work package related dependencies, i.e. controller functionality and calibration, currently only the calls for the authentication process have been defined. A full overview of the implementation will be made available after all functionalities have been defined.

### 3.6.1. Service

Function name: public static string GetSDKVersion()

Functions: Get SDK version number

Parameter: none

Return value: SDK version number

Method of calling: Primevr2\_UnitySDK.Service.GetSDKVersion()

### 3.6.2. Authentication

Handles user authentication and login.

Initial user login will take place in the Application Loader. Data will be stored inside Unities PlayerPrefs. At the launch of the game we can automatically login the user with the same credentials.

Function name: public static void LoginWithEmailAddress

Functions: Signs in an existing PRIMEVR2 user

Parameters: LoginWithEmailAddressRequest

Action<LoginResult> resultCallback

Action<LoginError> errorCallback

Return value: none

Method of calling: Primevr2\_UnitySDK.Service.LoginWithEmailAdress()

Function name: public static void LogOut

Functions: Logs out an active user

Parameters: none

Return value: none

Method of calling: Primevr2\_UnitySDK.Service.LogOut()

Function name: public static IsClientLoggedIn

Functions: Retrieves login state of user

Parameters: none

Return value: bool

Method of calling: Primevr2\_UnitySDK.Service.IsClientLoggedIn()

Function name: public static ForgetAllCredentials()

Functions: Clears the Client SessionToken which allows this Client to call API calls requiring login.

Parameters: none

Return value: none

Method of calling: Primevr2\_UnitySDK.Service.ForgetAllCredentials()

## **3.7. Sample Scenes**

---

After the implementation of described functionality, sample scenes will be made available as a reference for future game implementations. A full list of sample scenes providing setup, calibration details and mock-up gameplay, will be described here.

### **3.7.1. Sample Scene index**

- Calibration
- Achievements Example
- Analytics
- Exercise metrics
- Realtime assistance
- Other

## **3.8. Platform Loader**

---

The Platform Loader is a desktop application that can be downloaded and installed as the main application for the project. When a patient wants to launch a game, he will use this application to login with his account, and select one of the games that have been assigned by a therapist using the web portal. The application will give general information about each game and manages the download and installation of each game.

When a game is launched using the Platform Loader a user session is created and will be delegated to the games, so the games won't require an additional login authentication. The current functionality of the Platform Loader will only consist of managing and launching games.

## **3.9. Realtime Assistance**

---

When a game is launched, it will provide a 2d dashboard on a connected desktop monitor. This dashboard can be used to allow non-VR interactions with the game and its settings. A caretaker or therapist can view what a patient is seeing in VR, or move their own flythrough camera to get a good overview of what a patient is doing.

Using microphone input, a therapist can give directions to a patient that is wearing a headset. The dashboards expose the game settings, which would solve problematic interactions with the menu and setting from within VR.

The current feature does not provide networked communication and is focused solely as an on-premise solution to a tethered headset.

The functionality of the dashboard can be extended to meet more active requirements, such as world / game interaction, through 'click and select' mechanisms.

## **3.10. Game Structure**

---

### **3.10.1. Game**

One of the Prime VR2 games installations, each having their own thematic 3d surrounding and set of exercises. Each game implements the Unity SDK and will run as a separate installment. example: 'Greenhouse Gardening'

### **3.10.2. Environment**

A thematic 3d environment in which a game is taking place. Most environments will have multiple areas to explore, some will be focused on a single position. example: 'Greenhouse'

### **3.10.3. Exercise**

Exercises are mini-games located within the game environments. They have a start, an ending and a measured result. Exercises will focus on improving a specific condition of the patient, isolating a specific muscle group or movement, or can be used as a platform to train exposure or gain confidence in performing specific tasks. Note that in the context of the game environment, an exercise is more than simply the training of a specific motion. They should be fun and challenging, hence the term mini-game. example: 'Watering the plants'

### **3.10.4. Exercise Session**

A single activity where a user starts, performs and completes a specific exercise. An exercise usually consists of multiple sets and repetitions of the same tasks.

### **3.10.5. Game Session**

A single gameplay session where a user starts a game, plays a various range of exercise tasks, and closes the game.

### **3.10.6. Activity**

Activity is the registration of both game and exercise sessions. See platform implementation plan for more information.

## **3.11. Score System**

---

### **3.11.1. Exercise Metrics**

Exercise metrics is a set of data collected from exercises.

This data is used to measure the rehabilitative process of a patient.

After each exercise session, various motoric conditions will be rated in their own relative score values. these values will then be stored and made visible on the web platform. The function of these metrics is to provide insight into a patient's progression. See a full list and description of the dissected measurable components below.

- Accuracy
  - A patient's capability to perform a task with precision.
- Force

- A patient's capability to apply a specific force to a motion.
- Pressure
  - A patient's capability to apply a certain grip.
- Fluency of motion
  - A patient's ability to make a motion gracefully without too much jitter.
- Range of motion (distance / angle)
  - A patient's ability to reach the hand or rotate the wrist.
- Reaction time
  - How quick does a patient respond from the moment a command is given to the moment the objective is completed.
- Speed of motion
  - The patient's ability to move with speed.
- Relaxation
  - The patient's ability to relax its muscle tension.

### 3.11.2. Score Calculation

A score is composed of both **qualitative** and **quantitative** measurements. The quantitative measurements are derived from the number of completed sets and/or repetitions. The qualitative measurements are derived from each exercise metric. Every exercise has an internal setup to calculate the final score by giving weights to each metric. I.e. when the goal is to water a plant, the *range* (angle) and *fluency of motion* would be the biggest determinants for the final score.

The score is also adapted to the patient's abilities to prevent a low starting score or a very slow progressing score over time. I.e. when a patient's ability is so limited that it can hardly rotate the wrist, he/she should still get a fair score by reaching all the way to its calibrated limits.

### 3.11.3. User Calibration

To understand the patient's limitations and capabilities, a short calibration process should be designed to gain insight into the different exercise metric values at the current state of the patient's rehabilitation. I.e. by giving the simple task like moving the hand from one place to another, we can determine most of the exercise metrics and suggest a compensational factor to be applied to the patient's motions.

## 3.12. Leaderboards

---

A leaderboard will primarily be visible on the web platform. A leaderboard will display a global score results surrounding specific exercises. A function of a leaderboard is to stimulate competitiveness amongst players by giving them insight in how others are doing in the exercise they are currently focused on.

## 3.13. Achievements

---

Achievements are unlockable rewards to mark certain progressive game events. The function of achievements is to both give a player an idea of what can be achieved (visible but yet locked achievements) and to rewards a player for something, whether it's completing a set of



exercises or discovering a new area within the environment. Every game has their own set of achievements.

### **3.14. Analytics**

---

Anonymized analytics collects large data about the types of patients, the total amount of games played etc., in order to track how the platform is being implemented and used.

### **3.15. Accessibility Tools (VR)**

---

To cover a wide range of patients with different disabilities and motoric or visual limitations, the following utilities **could** be developed as part of the SDK's accessibility toolset. Each tool's specific settings could be adjusted at runtime by a caretaker/therapist.

- Smoothing out movements (dystonia)
  - This could be a smoothing operation to support patients with excessive motoric disturbances in performing difficult tasks.
- Delayed or slowed down movements (dystonia)
- Over exaggerate range of movements (extending motoric limits)
- Magnification of objects of interest (visually impaired)
- Use of captions during spoken voices
- Gaze tracking mechanism to interact with UI (stroke, dystonia)
- Controlling font sizes. (visually impaired)

#### **3.15.1. Motion Compensation**

There are a number of ways the virtual representation of the patient's arm and motion can be used to support the completion of exercises or help the patient gain confidence in completing tasks.

In one case, a patient's motoric limitations are too severe to complete a motion like rotating the wrist to rotate a watering can. By calibrating the maximum range of motion of the patient, the rotation of the watering can be exaggerated in order to complete the task. In another case the patient's fluency of motion is so jittery, it can hardly perform the required motion, resulting in a loss of confidence. This effect can be compensated by enabling a smoothing pass on the patient's motion. As there are multiple scenarios to be explored and tested in both functionality and results, above stated mechanisms are not yet finalized and incorporated into the development plan.

- Smoothing out movements (dystonia)
  - This could be a smoothing operation to support patients with excessive motoric disturbances in performing difficult tasks.
- Delayed or slowed down movements (dystonia)
- Over exaggerate range of movements (extending motoric limits)

## 4 WEB PLATFORM

### 4.1. Introduction

The PRIME-VR2 Web Portal is a web application that coordinates the activity of hospital supervisors, doctors and patients in order to achieve successful VR therapy outcomes.

The system consists of two main functional units:

- **Website:** A human-friendly user interface where users can interact with the system.
- **Gameplay API:** A machine-friendly API that games can use to query configuration options and upload gameplay data.

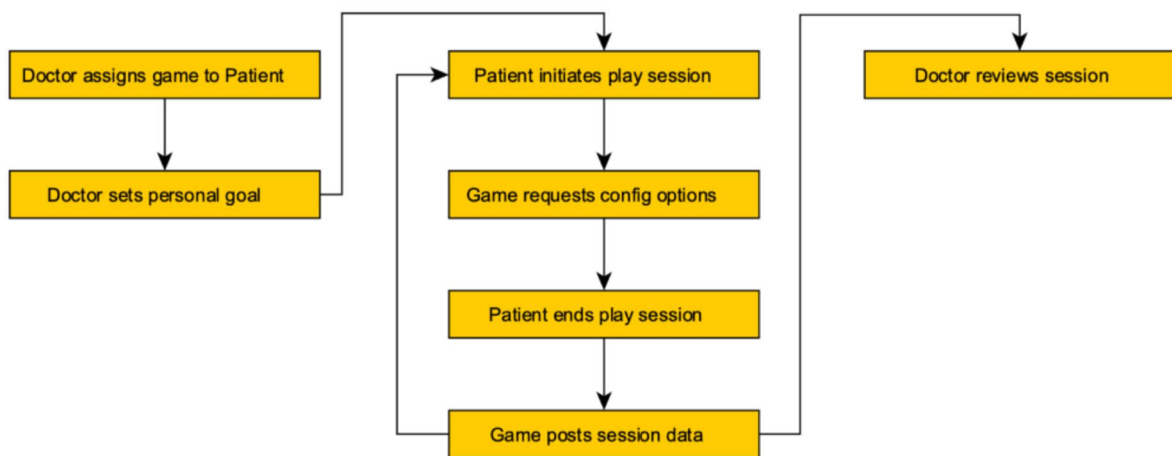


Fig. 4.1. User Type Hierarchy

### 4.2. Architecture

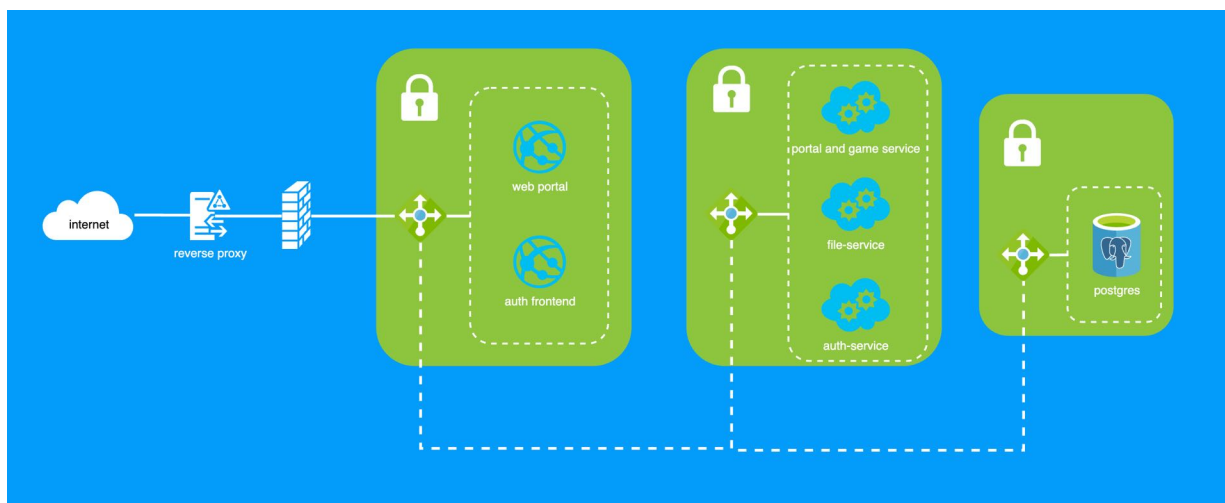


Fig. 4.2. Prime-VR2 Web Portal architecture diagram

We designed our system with microservice architecture in mind. All our services are running as independent containers. For container orchestration we're using Kubernetes, which provides us both resiliency and flexibility in the operation tasks.

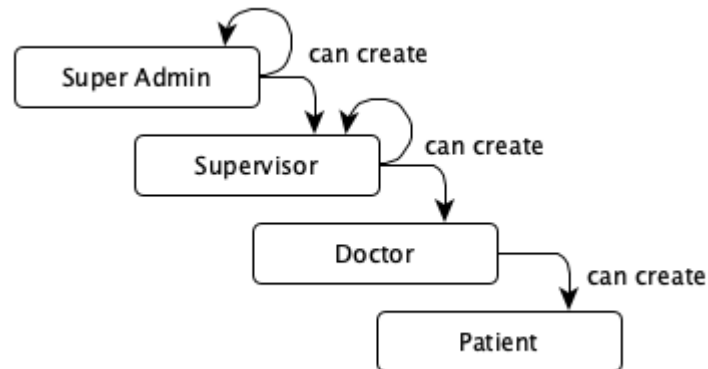
For authentication and authorization, we're using our own authentication service to ensure none of the services are accessible for non-authorized persons.

## Key components

- **Web Portal** – an Angular application which is the central hub for both patients and therapists. They can manage and understand the metrics that are coming from the games. It contains the login screen and all management screens, also the overviews. The users with the right permissions can use the Portal to add and edit other users, create, edit and delete hospitals and games. The end users can use the Web Portal to follow their progress and analyze their performance in different exercises.
- **Auth frontend** – a strictly administrative UI to manage roles and permissions. An administrator can use this special and secure UI to interact with the Auth service. Admins can manage users, roles and permissions through this.
- **Portal and game service** – this is the main service which serves the Web Portal and also provides an API to games to be able to send session data and access game details. The games communicate with this API through the HTTP REST standard. The API endpoints are secured with the use of JWT technology and firewall.
- **File service** – Basically this is the Prime VR2 blob storage. The users or the games can upload files and it will be stored as objects / blobs on a decentralized storage place. The UI or the games are not interacting with the physical storage directly. The File service is a secured abstraction level built on top of the physical storage to achieve custom business logics.
- **Auth service** – The whole user management is implemented in this component. It stores the list of users with all the details. Using advanced industry standard solutions to securely store passwords. The Auth service also manages the login / logout and the forgot password processes. This is the central component who gives JWT tickets for the users if they are successfully logged in. Later using this secure JWT the users are able to use different APIs if they have the right permissions to do that.
- **Database** – A relational database (PostgreSQL) is used for the persistent data storage for all of the services. This is the only component, which is not run in a container, as an industrial standard it's an independent database server, with daily backups.

## 4.3. User Types

---



*Fig. 4.2. User Type Hierarchy*

As in D2.1 proposed as requirements, the platform supports four different user types that each have a different set of security permissions - thereby they can access a different set of functionalities and see different parts of the user interface.

### **Super Admin**

Responsible for tasks concerning the entire portal – adding new Supervisors, Doctors and Patients, and disabling/enabling them. They can access a read-only view of all patient data in the system.

### **Supervisor**

Responsible for managing a hospital in the system. They can create Doctors and Patients, disable and enable them. They can access a read-only view of all patient data in their respective hospital.

### **Doctor**

Responsible for administering therapy to the patients. They can create new Patients, disable and enable them, set which games are available to each of them and edit their configuration options, and set personal goals for each game and user. They can add, edit and delete game sessions and view the leaderboards for all games.

### **Patient**

Can play games and review their progress. They can view leaderboards for each game they are assigned.

#### **4.3.1. Permissions**

The following table lists all available operations in the system and whether different user types can perform them.

Some table headers have been abbreviated:

- SA = Super Admin
- SPV = Supervisor
- D = Doctor
- P = Patient

Operation	SA	SPV	D	P
List All Users	●			
List Users in Own Hospital		●		
List Own Patients			●	
Create Super Admin	●			
Create Supervisor	●	●		
Create Doctor	●	●		
Create Patient	●	●	●	
View Patient Overview for Any Patient	●			
View Patient Overview for Any Patient in Own Hospital		●		
View Patient Overview for Own Patient			●	
View Own Patient Overview				●
Edit Super Admin	●			
Edit Any Supervisor	●			
Edit Any Doctor	●			
Edit Any Patient	●			
Edit Any Supervisor in Own Hospital		●		
Edit Any Doctor in Own Hospital		●		
Edit Any Patient in Own Hospital		●		
Edit Own Patient			●	
Disable/Enable Super Admin	●			

<b>Disable/Enable Any Supervisor</b>	<input checked="" type="radio"/>			
<b>Disable/Enable Any Doctor</b>	<input checked="" type="radio"/>			
<b>Disable/Enable Any Patient</b>	<input checked="" type="radio"/>			
<b>Disable/Enable Any Supervisor in Own Hospital</b>		<input checked="" type="radio"/>		
<b>Disable/Enable Any Doctor in Own Hospital</b>		<input checked="" type="radio"/>		
<b>Disable/Enable Any Patient in Own Hospital</b>		<input checked="" type="radio"/>		
<b>Disable/Enable Own Patient</b>			<input checked="" type="radio"/>	
<b>Delete Super Admin</b>	<input checked="" type="radio"/>			
<b>Delete Any Supervisor</b>	<input checked="" type="radio"/>			
<b>Delete Any Doctor</b>	<input checked="" type="radio"/>			
<b>Delete Any Patient</b>	<input checked="" type="radio"/>			
<b>Delete Any Supervisor in Own Hospital</b>		<input checked="" type="radio"/>		
<b>Delete Any Doctor in Own Hospital</b>		<input checked="" type="radio"/>		
<b>Delete Any Patient in Own Hospital</b>		<input checked="" type="radio"/>		
<b>Delete Own Patient</b>			<input checked="" type="radio"/>	
<b>Edit Own Profile</b>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
<b>List Hospitals</b>	<input checked="" type="radio"/>			
<b>Create Hospital</b>	<input checked="" type="radio"/>			
<b>Edit Hospital</b>	<input checked="" type="radio"/>			
<b>Disable/Enable Hospital</b>	<input checked="" type="radio"/>			
<b>Edit Own Hospital Profile</b>		<input checked="" type="radio"/>		
<b>List Games</b>	<input checked="" type="radio"/>			
<b>Create Game</b>	<input checked="" type="radio"/>			

<b>Edit Game</b>	<input checked="" type="radio"/>			
<b>Disable/Enable Game Globally</b>	<input checked="" type="radio"/>			
<b>Disable/Enable Game for Patient</b>			<input checked="" type="radio"/>	
<b>Configure Game</b>			<input checked="" type="radio"/>	
<b>Set Personal Goal</b>			<input checked="" type="radio"/>	
<b>Create Session</b>			<input checked="" type="radio"/>	
<b>Edit Session</b>			<input checked="" type="radio"/>	
<b>Delete Session</b>			<input checked="" type="radio"/>	
<b>View Leaderboard</b>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
<b>Send Message to Own Patient</b>			<input checked="" type="radio"/>	
<b>Send Message to Own Doctor</b>				<input checked="" type="radio"/>

## 4.4. User Management

---

### 4.4.1. User List

Please find the information of user lists, user control, user profile, and login requests as follows, based on the requirements D2.1.

The administrators, that is super admins, supervisors, and doctors should have the abilities to manage the users on the system as following:

A sortable, filterable, paginated list of users in the system. By default:

- Super Admins see a list of all users in the system
- Supervisors see a list of all users in their respective hospital
- Doctors see a list of all Patients assigned to them
- Patients cannot access the list.

List items have the following fields:

- Email
- User Type - Super Admin/Supervisor/Doctor/Patient

- Hospital Name - empty for Super Admins
- Full Name
- Nickname - empty for everyone except patients
- Number of unread messages from this user - only for Doctors

Users can sort the list by email, hospital name, full name or nickname, in alphabetical or reverse alphabetical order, by clicking on the corresponding list header field.

Users can filter the list by:

- Email - free text
- User Type - select from list (not available to Doctors who can only see Patients):
  - Super Admin - option only available to Super Admins
  - Supervisor
  - Doctor
  - Patient
- Hospital Name - select from list; only available to Super Admins
- Full Name - free text
- Nickname - free text
- Has Unread Messages - true/false

The following operations are available on the User List page:

- Create User
- Patient Details
- Edit User
- Disable/Enable User
- Delete User
- Open Message Thread View (for Doctors)

The next five sections describe these in detail. Message Thread View is described in the Messaging section.

#### **4.4.2. Create User**

This is a page Super Admins, Supervisors and Doctors can use to create new Users.

Super Admins can create:

- Super Admins
- Supervisors
- Doctors
- Patients



Supervisors can create (only in the same hospital):

- Supervisors
- Doctors
- Patients

Doctors can create (only in the same hospital; assigned to themselves):

- Patients

When creating any user, the following fields must be specified:

- Email (must be of valid format and must not exist in the system)
- User Type
- Full Name (must have at least three non-whitespace characters)

Additionally, the Super Admin must select the Hospital when creating Supervisors, Doctors or Patients. Supervisors and Doctors can only create users for their respective Hospitals.

When creating Patients, the user must select the Doctor they are assigned to. Supervisors can only assign Patients to Doctors in their own Hospital. Patients created by Doctors are automatically assigned to them.

### 4.4.3. New User Flow

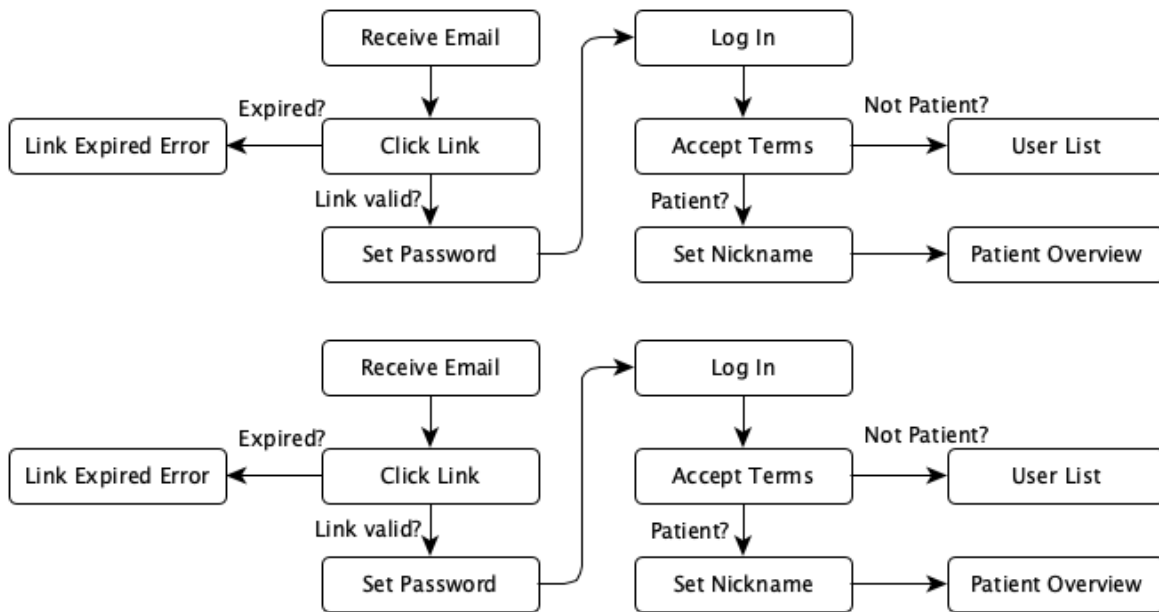


Fig 4.3. New User Flow

1. Newly created users get an email that contains a unique, single-use link. This link expires in one week.
2. By clicking the link, they can access a web page where they can set a password for themselves. The password must be confirmed by typing it again in a separate field to prevent typos; the two fields must match for the registration process to continue. The password must be at least 8 characters long.
3. After entering the code on the website, they are taken to the login page where can now access the system by logging in.

### 4.4.4. Patient Overview

The Patient details page is available to Super Admins, Supervisors and Patients. It shows the following basic data about a Patient:

- Email
- Hospital Name
- Full Name
- Nickname

This page also shows game-specific data that will be discussed in a later section.

#### 4.4.5. Edit User

This page allows editing the following user profile fields:

- Email
  - Free text
  - Must be of valid format
- Hospital
  - Select from list
  - Only for Supervisors, Doctors and Patients
  - Only available to Super Admins
- Assigned Doctor
  - Select from list
  - Only for Patients
  - Only available to Super Admins and Supervisors
  - Super Admins can only choose from Doctors in the selected Hospital
  - Supervisors can only choose from Doctors in their own Hospital
- Full Name
  - Free text
  - Must be at least 3 non-whitespace characters
- Nickname
  - Free text
  - Must only contains letters, numbers and underscores
  - Only for Patients

The type of a user cannot be changed.

#### 4.4.6. Disable/Enable User

This function allows users to disable another user.

Disabling a user means they can no longer log in to the system. Enabling them means undoing this.

Super Admins can disable and enable:

- other Super Admins (not themselves)
- Supervisors
- Doctors and
- Patients.

Supervisors can disable and enable:

- other Supervisors in their own Hospital (not themselves)
- Doctors in their own Hospital and
- Patients in their own Hospital.

Doctors can disable and enable:

- Patients assigned to them.

#### **4.4.7. Delete User**

According to EU regulations (colloquially, the “Right to be Forgotten”), a user of a software system must be able to request the permanent removal of all of their data from that system.

Deleting a user will remove all of their associated data: their profile, Hospital, Doctor and Game assignments, personal goals, high scores from leaderboards and their entire gameplay history.

Super Admins can delete:

- other Super Admins (not themselves)
- Supervisors
- Doctors and
- Patients.

Supervisors can delete:

- other Supervisors in their own Hospital (not themselves)
- Doctors in their own Hospital and
- Patients in their own Hospital.

Doctors can delete:

- Patients assigned to them.

Users cannot delete themselves in order to prevent accidental data loss. They must request the removal of their data on a communication channel inside or outside the system (by using the built-in messaging function, emailing or calling another user who can delete them).

#### **4.4.8. Edit Own User Profile**

All users can access a page where they can edit their own:

- Email,
- Full Name and
- Password

The password must be confirmed by typing it again in a separate field to prevent typos; the two fields must match exactly. The password must be at least 8 characters long.

If the password fields are left empty, the user’s password remains unchanged; only the other fields are updated.

Patients can also edit their Nickname.

## 4.5. Login

A user can log in to the system by correctly entering their:

- Email
- Password

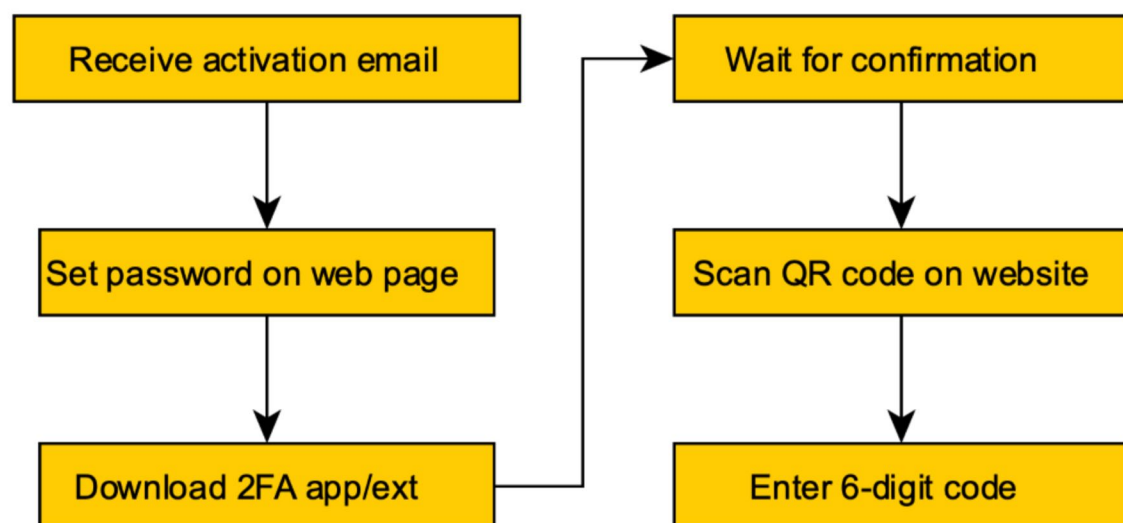
The login page also has a “Forgot your password?” link that points to the Forgotten Password page.

If any of the above are incorrect, the user is presented with a generic “Invalid login data” message. Due to security concerns, we cannot provide more detailed feedback on which of the fields contain incorrect data.

On successful login, Super Admins, Supervisors and Doctors are redirected to the User List page.

Patients are redirected to the Game Overview page.

### 4.5.1. First Login



*Fig 4.4. Login sequence*

On first login, after successfully entering their email and password, users are redirected to a Profile Setup page where:

- All users must accept the terms and conditions by checking a checkbox.
  - The document can be read by clicking on a link - it opens on a new browser tab.
  - If the user declines, a confirmation prompt appears: “Are you sure you want to decline? This will delete your profile from the system.”

- On clicking “No”, the confirmation disappears.
- On clicking “Yes”, the user profile is deleted from the system and the browser is redirected to the main page.
- Patients must enter a Nickname.
  - Free text
  - Must only contains letters, numbers and underscores
  - Must not exist in the system

After successfully completing the Profile Setup form, they are redirected to the page described in the previous section.

#### **4.5.2. Forgotten Password**

Upon clicking the “*Forgot your password?*” link on the login page, users can enter their email address in a text field and request a password reset link by clicking a button.

Due to security concerns, we cannot provide feedback to the user about the existence of the email entered. If the email is mistyped or does not exist, sending the link will silently fail.

If the email address specified exists in the system, a unique, single-use link is sent to the user. By clicking the link, they are presented with a page where they can set a new password for themselves. The password must be confirmed by typing it again in a separate field to prevent typos; the two fields must match for the registration process to continue. The password must be at least 8 characters long.

## **4.6. Hospital Management**

---

As D2.1 requires, the hospital management has to relate to how the individual hospitals utilising the web platform will manage patient data. WP6 considered the requirements and decided to implement it in the following way:

### **4.6.1. Hospital List**

Only Super Admins can access this page.

The page displays a sortable, filterable, paginated list of Hospitals in the system. For each Hospital, the following fields are shown:

- Hospital Name
- Country
- City

Users can sort the list by hospital name, country and city, in alphabetical or reverse alphabetical order, by clicking on the corresponding list header field.

Users can filter the list by:

- Hospital Name - free text
- Country - select from list
- City - select from list

The following operations are available on the Hospital List page:

- Create Hospital
- Manage Users
- Edit Hospital
- Disable/Enable Hospital

Hospitals cannot be deleted.

The next four sections describe these operations in detail.

#### **4.6.2. Create Hospital**

Only Super Admins can access this page.

To create a Hospital, the Super Admin must specify the following:

- Hospital Name - free text; at least 3 non-whitespace characters
- Country - select from list
- City - free text; at least 3 non-whitespace characters
- Street Address - free text; at least 3 non-whitespace characters
- Zip Code - free text; at least 2 non-whitespace characters
- Contact Name - free text; at least 3 non-whitespace characters
- Contact Email - free text; must be valid email address
- Contact Phone - free text; at least 3 non-whitespace characters

#### **4.6.3. Manage Hospital Users**

This is a link that points to the User List page, with the filter pre-set to the Hospital selected.

#### **4.6.4. Edit Hospital**

Only Super Admins can access this page. They can edit all attributes of a Hospital:

- Hospital Name - free text; at least 3 non-whitespace characters
- Country - select from list
- City - free text; at least 3 non-whitespace characters
- Street Address - free text; at least 3 non-whitespace characters
- Zip Code - free text; at least 2 non-whitespace characters
- Contact Name - free text; at least 3 non-whitespace characters
- Contact Email - free text; must be valid email address
- Contact Phone - free text; at least 3 non-whitespace characters

#### **4.6.5. Disable/Enable Hospital**

Only Super Admins can access this function.

Disabling a Hospital means that no user assigned to that Hospital can log in to the system. Enabling a Hospital means undoing this.

#### 4.6.6. Edit Own Hospital Profile

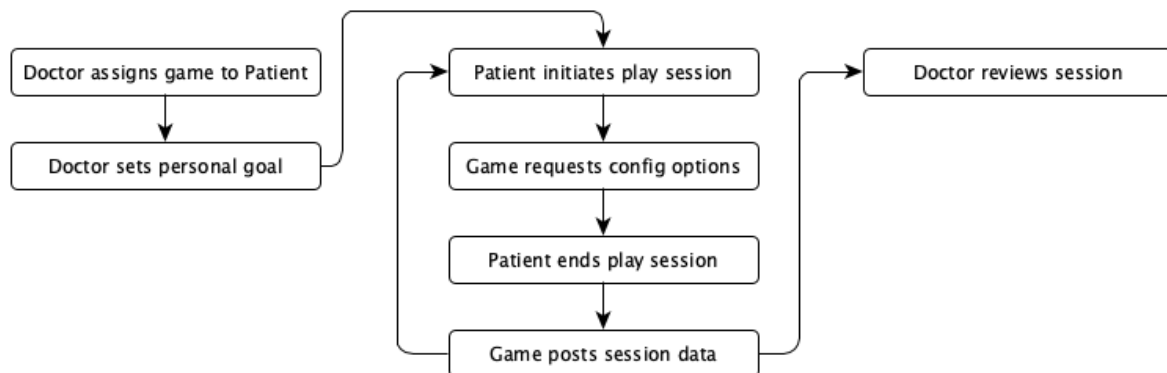
Only Supervisors can access this page. They can edit all attributes of a Hospital:

- Hospital Name - free text; at least 3 non-whitespace characters
- Country - select from list
- City - free text; at least 3 non-whitespace characters
- Street Address - free text; at least 3 non-whitespace characters
- Zip Code - free text; at least 2 non-whitespace characters
- Contact Name - free text; at least 3 non-whitespace characters
- Contact Email - free text; must be valid email address
- Contact Phone - free text; at least 3 non-whitespace characters

### 4.7. Games

---

Based on the requirements of the game management - how games should be accessed, edited, and administered to patients and therapists, because this is a critical part of successful therapy assignment -, we implemented the gameplay flow as follows:



*Fig. 4.6 Gameplay Flow*

#### 4.7.1. Games List

This page is only available to Super Admins.

A sortable, filterable, paginated list of Games in the system. List items have the following fields:

- Name
- Recommended gameplay time per day (if specified)



- Recommended sessions per week (if specified)

Users can sort the list by name, in alphabetical or reverse alphabetical order, by clicking on the corresponding list header field.

Users can filter the list by name (free text).

The following operations are available on the Game List page:

- Create Game
- Edit Game
- Disable/Enable Game Globally

Games cannot be deleted.

The next three sections describe these in detail.

#### **4.7.2. Create Game**

This page allows Super Admins to create new Games. The following fields must be specified:

- Name
  - Free text
  - At least 3 non-whitespace characters
- Icon
  - PNG format
  - 64X64 pixels
- Game preview image
  - PNG format
- Description
  - Free text
  - At least 3 non-whitespace characters
- Recommended Uses
  - Free text
  - At least 3 non-whitespace characters
- Intro Video
  - MP4 video
  - Max. 50 MB
- Executable game file
  - We don't know the extension yet.
- At least one of:
  - Recommended gameplay time per day
    - Number
    - Between 1-1440
  - Recommended sessions per week
    - Number
    - Between 1-35
- Configuration Option Definition

- JSON Schema file (.schema.json extension) provided by the game developer
  - See: <https://json-schema.org/>
  - All object properties in the schema must have a title attribute (these become form labels when the Doctor configures a Game)
  - The file must contain well-formed JSON
  - The file must conform to the core/validation JSON metaschema: <https://json-schema.org/draft/2019-09/schema>

Newly created games are automatically disabled. Only by enabling them will they become available to non-superadmin users.

### **4.7.3. Edit Game**

This page allows Super Admins to edit all game attributes listed in the previous section except the configuration options. Due to technical constraints, if the option definition of a game changes, a new Game must be created (e.g. by appending a version number to its name).

### **4.7.4. Disable/Enable Game Globally**

This function allows Super Admins to control the availability of Games across the entire system. By disabling a Game, it is no longer listed as available for Supervisors and Doctors. Patients can still review their gameplay history.

## **4.8. Patient Overview**

---

It was highly important to create the patient overview where the data is organised, accessed and displayed in the right way. As the different users should have access to different data, we created the platforms as follows:

This page is available to Super Admins, Supervisors, Doctors and Patients. It gives an overview of a Patient's profile data (discussed earlier in the User Management section), gameplay activity and goals achieved. Doctors can also access additional functions.

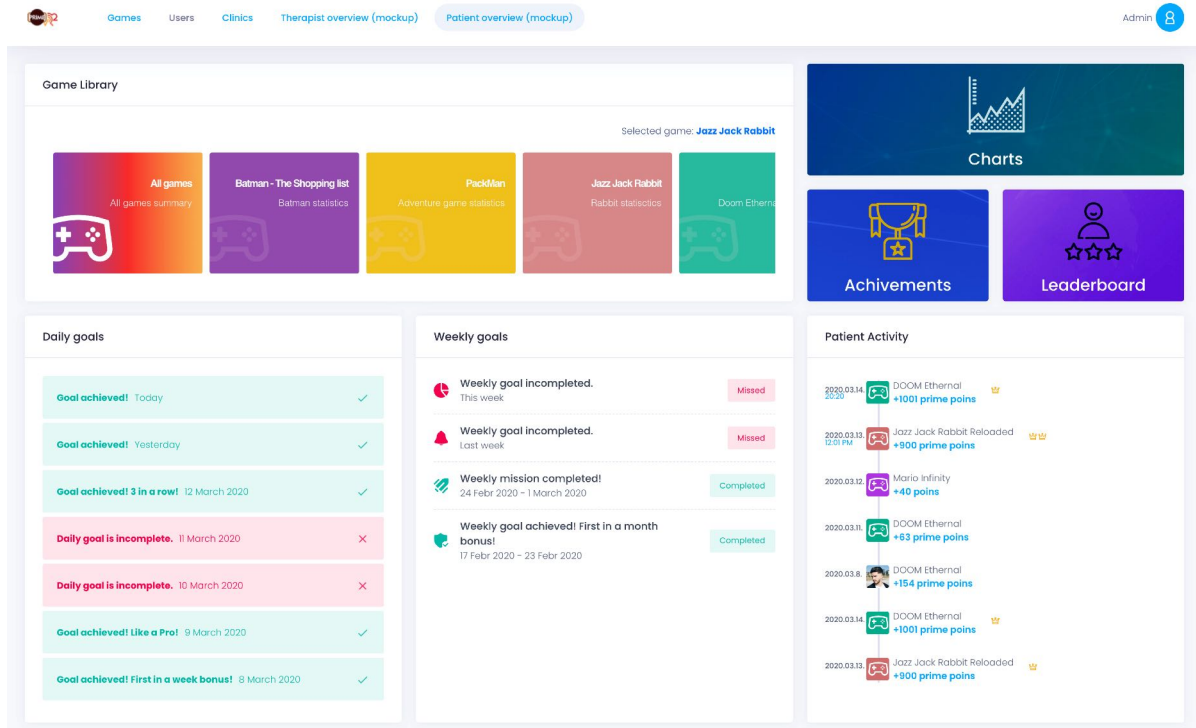


Fig. 4.8 User Interface for the Patient Overview page

The page consists of the following sections:

- Profile Data
- List of Enabled Games
- Goal Review
- Activity Feed

#### 4.8.1. List of Games

This is a list of globally enabled Games that are also enabled for the Patient. Only the icon and the game name are shown. By clicking the icon, the Game Details Popup appears.

Doctors can see all globally enabled Games, regardless of whether they are enabled for the Patient. Games that are disabled for the Patient have a saturated/grayed-out icon.

#### 4.8.2. Goal Review

This section is a visualization of the goals achieved by the Patient. A dropdown list allows selecting a single game or an aggregation of goal data from all games (an option titled “All Games”). By default, “All Games” is selected. A calendar widget shows, by color coding and other visual cues, whether the Patient achieved their goals for the days and weeks displayed.

### 4.8.3. Activity Feed

This is a reverse chronological, lazy-loaded, infinitely-scrolling list of the user's gameplay sessions. The following data are displayed for each session:

- Game Name
- Game Icon (small size)
- Date and Time
- Duration
- Score
- "Daily goal achieved" badge, where applicable
- "Weekly goal achieved" badge, where applicable

By clicking a list item, the configuration options are displayed in a pop-up window.

Doctors can also access the following functions for each list item:

- Edit Session
- Delete Session
- Comment on Session (see the Messaging section)

And also, a global Create Session function. These will be discussed in detail in later sections.

### 4.8.4. Game Definition

For a new Game to be created in the system, the following data must be specified:

- Game Name (free text)
- Icon (PNG file, 64×64 pixels)
- Intro Video (MP4 file with H264 encoding, max. 50 MB)
- Preview Image (PNG file, 1920×1080 pixels - "Full HD")
- Game File (.exe, .zip, etc.)
- Description (free text)
- Recommended Uses (free text)
- Recommended Goals
  - At least one of the following:
    - Number of minutes per day
    - Number of sessions per day
- Configuration Option Definition
  - JSON Schema file (.schema.json extension) provided by the game developer
    - See: <https://json-schema.org/>
    - All object properties in the schema must have a title attribute (these become form labels when the Doctor configures a Game)
    - The file must contain well-formed JSON



#### 4.8.8. Leaderboard

The Leaderboard is a page where Super Admins, Supervisors, Doctors and Patients can see the top 10 daily, monthly and all-time scores for any globally enabled Game (Patients can only see leaderboards for Games that are enabled for them).

The user can select from a list of Games and a list of time ranges (“Today”, “This Month”, “All Time”). Based on their selection, the 10 best scores and the nicknames of the Patients who have achieved them are displayed. By default, a placeholder screen is shown, the user must select both options to see any data.

### 4.9. Messaging

---

The system provides a real-time messaging system that allows Doctors and Patients to communicate effectively. Right now this is more a later development that is not needed in the prototype phase but it is nice to have when the Prime VR2 will achieve it’s commercialized phase. The Messaging feature is not going to be implemented now.

#### 4.9.1. Message icon

The website header has an icon that represents a message (e.g. an envelope). A red dot overlay appears over this icon if the user has any unread threads. Clicking this icon opens the User List filtered by the “Has Unread Messages” option (for Doctors) or the Message Thread View (for Patients).

The number of unread messages is **not** shown inside the indicator.

#### 4.9.2. Message Thread View

This page displays a chat conversation between a Patient and their Doctor. Messages are displayed in speech bubbles. The current user’s messages are aligned to the right, the other user’s messages are aligned to the left. The current user can send a new message by entering text into a text box on the bottom of the page and clicking the “Send” button next to it, or pressing Enter.

The system does **not** support sending:

- Emoticons
- Images
- Videos
- Any other media type except plain text

Opening the Message Thread View marks the thread as read and makes the red dot indicator disappear if there are no other unread threads.

#### 4.9.3. Real-Time Delivery

A web socket connection is established between the server and the browser after a Doctor or a Patient logs in. Incoming messages are received through the web socket which guarantees near-instantaneous delivery. If the current page is the Message Thread View, the message is displayed and the thread stays in the “read” state. If another page is active, the thread is marked as “unread” and the red dot indicator appears. The web socket connection is automatically disposed when the user closes the site.

#### **4.9.4. Commenting on Gameplay Sessions**

Doctors can post comments on gameplay sessions by clicking the “Comment on Session” button next to a session on the Patient Overview page. This:

- Redirects the Doctor’s browser to the Message Thread View for the Patient
- Pre-fills the text box with the following text:
  - “Commenting on your session: {game name} {session date}, {duration} minutes, {score} points” and a newline character

This way the Doctor can add their own comments to the message and send it by clicking “Send” or delete it from the text box to discard it.

## 4.10. Notifications

---

When certain events occur in the system, users receive email notifications. These events, the users affected and the data content of the messages are listed below.

Event	Recipient	Content
User Created	The user that has been created	Unique, single-use registration link
Password Reset Requested	The user that has requested the password reset link	Unique, single-use password reset link
Patient Assigned to Doctor	The Patient that has been assigned	The name of the Doctor
Patient Assigned to Doctor	The Doctor the Patient has been assigned to	The name of the Patient
Patient Assignment Deleted	The Doctor the Patient is no longer assigned to	The name of the Patient
Message Received	The recipient of the message (Doctor or Patient); only if not currently on-line	The name of the sender and the contents of the message
New Game Enabled for Patient	The Patient the Game is enabled for	The name of the game, the name of the Doctor who assigned it and the personal goals set
Personal Goals Changed	The Patient the goals are set for	Name of the Doctor and the new personal goals
Daily Personal Goal Met	The Patient whose goal is met	Name of the Game and the description of the goal
Weekly Personal Goal Met	The Patient whose goal is met	Name of the Game and the description of the goal
Game Disabled for Patient	The Patient the Game is disabled for	Name of the Doctor and the Game
Game Enabled Globally	All Super Admins, Supervisors and Doctors	Name of the Game
Game Disabled Globally	All Super Admins, Supervisors and Doctors; all Patients the Game is enabled for	Name of the Game



## 5 GAMEPLAY API

### 5.1. Authentication

---

The API uses JWT (JSON Web Token - <https://jwt.io/>) based authentication.

- The client authenticates with a third-party server (to be specified later) and receives a JWT - a cryptographically signed JSON object that contains the identity of the user.
- The client includes this token in the Authorization header in all requests made to the API in the following format: `Authorization: Bearer {token}`
- The API service validates the signature to verify that the information in the token is valid. After successful validation, it can assume that the identity data received is correct.
- The token has an expiration time that is stored in one of the JSON properties. After this time, the client needs to request a new token, otherwise authentication with the API will fail.

### 5.2. Configuration endpoint

---

The response JSON object returned by the Configuration endpoint will be extended with a new state property that contains the latest state the game posted to the Session endpoint. If no previous state exists, null is returned.

```
{
  "config": // configuration option values as defined in the JSON
            schema,
  "state" : {} // the previously stored state
}
```

### 5.3. Session endpoint

---

The JSON object in the body of the Session endpoint POST request will be extended with a new achiev

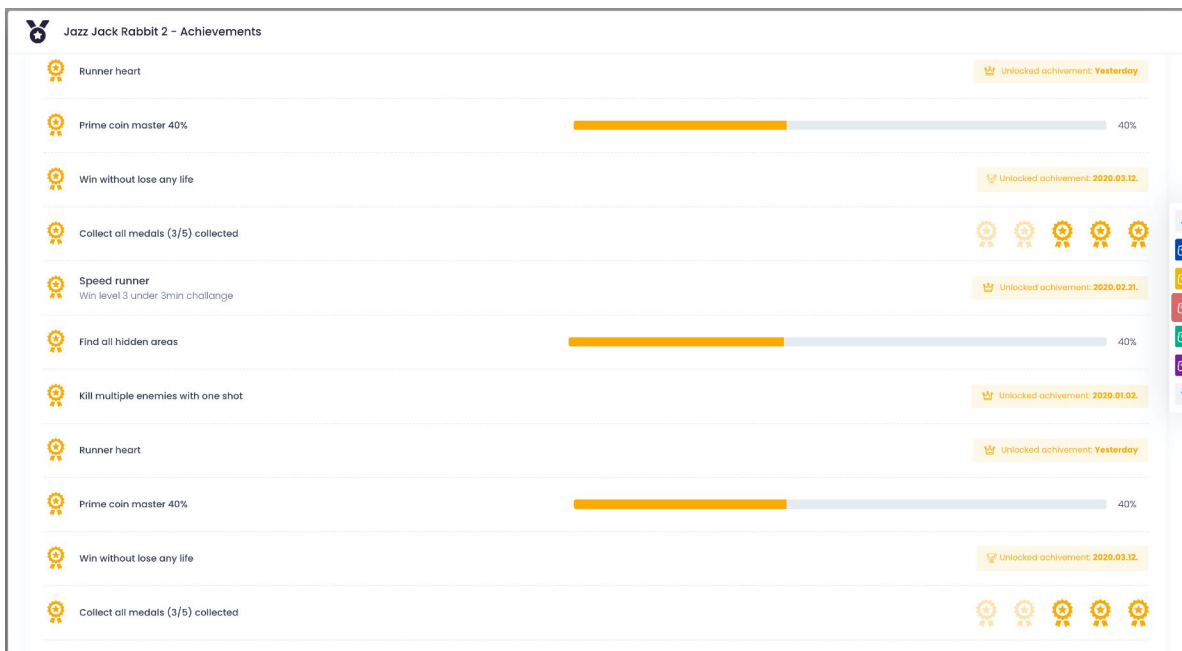
```
{
  "patientId": /* patient ID - string (UUID) */,
  "gameId": /* game ID - string (UUID) */,
  "startedAt": /* start date and time - ISO date and time string */,
  "duration": /* duration in minutes - positive integer */,
  "score": /* score - nonnegative integer */,
  "sessionData": /* custom JSON object, see below */,
  "achievements": ["100-coins", "discovered-hidden-room"],
  "variables": [{ name: "coins", value: 46 }]
  "state": { /* custom game state object */ }
}
```

- The **achievements** array contains all achievements unlocked in the session posted.
- **State** is a custom object where the game can store its current user-specific state in order to be able to calculate future achievements.

## 6 ACHIEVEMENTS

In the requirements of D7.1, it was emphasized that, the rehabilitation can be more effective (especially for sports participants), when the games are providing a competitive environment. It is important to challenge the patients to achieve more and more on their healing progress.

Visible and yet locked achievements can motivate the patient as well as give them rewards and recognition for their keen efforts. Furthermore, doctors easily can see how their patients are progressing, if the achievements are well described.



*Fig. 6 Achievement Pop-up*

Achievements are displayed for a specific User and Game in the Achievement Pop-up. This window shows a list of:

- Unlocked achievements in descending order of the time of unlocking
- Locked achievements for which the `visibleWhileLocked` property is set to `true`. The visual style indicates that these achievements are not yet unlocked. They appear after unlocked achievements and are sorted alphabetically.

Achievements that belong to the same group are shown as a single item in this list. The unlock time of such a grouped item is the latest unlock time in the group; this time is used when ordering items.

## 7 LEADERBOARDS AND CHARTS

Leaderboards help therapists to overview the progress of each patient from visualized data.

As in the requirements of D2.1 was described, the therapists can view the leaderboards for all game. Detailed data is showed from exercises, gameplay scores, and the patient's achievements.

Measurements show the current status and the achieved progress also, the results of the exercises and games are easy to manage and read. Another function of the leaderboard is to stimulate competitiveness amongst players, so these data can be motivating for patients to perform even more exercises.

Leaderboards and charts are calculated from the same data stream. The UI is custom-built for each game as the number of games will be low enough to make building a completely configurable system unnecessary.

Leaderboards and Charts will be built using variable values (key - value pairs) the game can send with each session.

Supported chart types with examples:

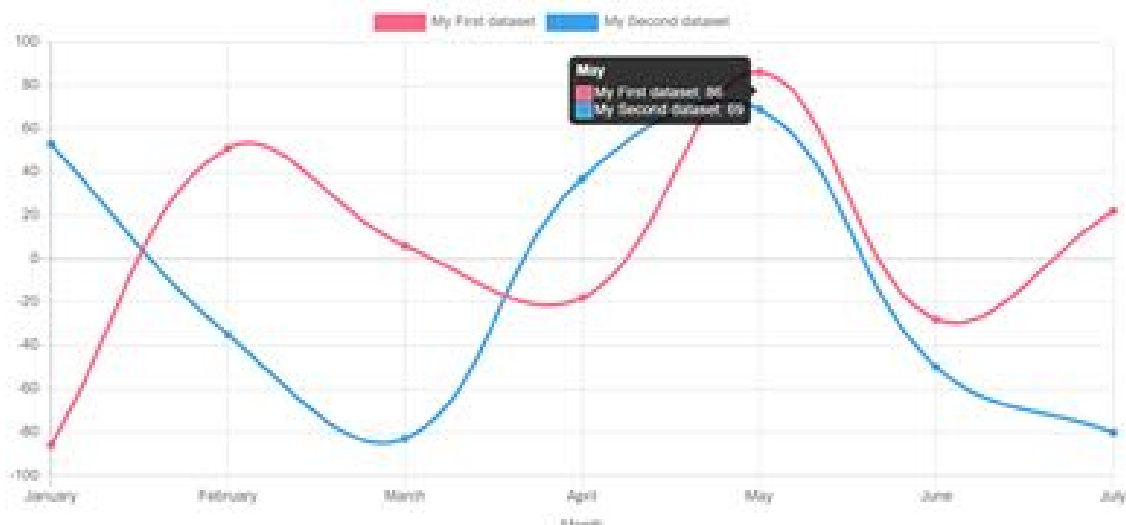


Fig. 7a Line Chart

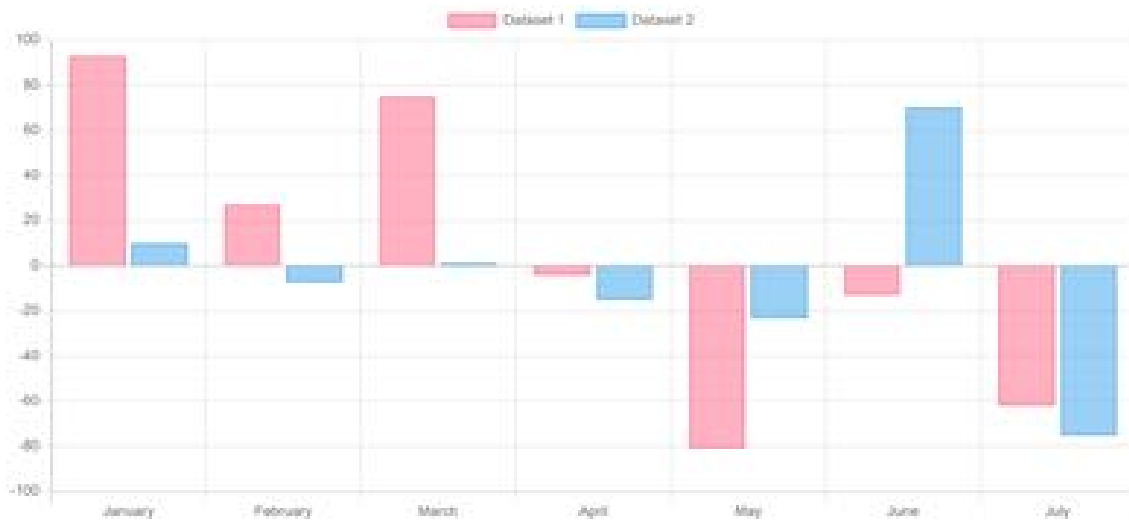


Fig. 7b Bar Chart

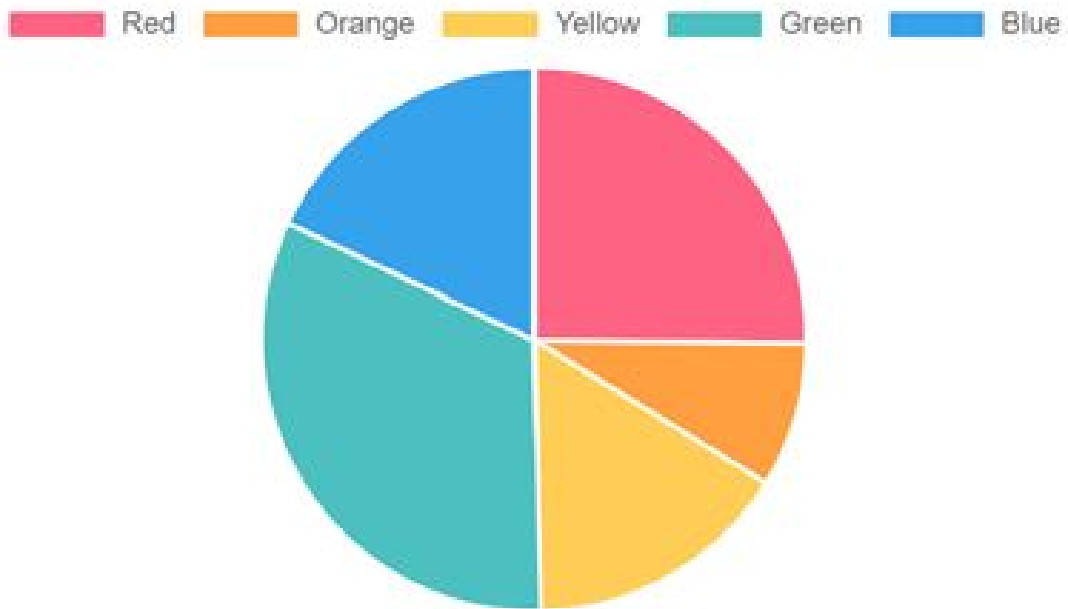


Fig. 7c Pie Chart

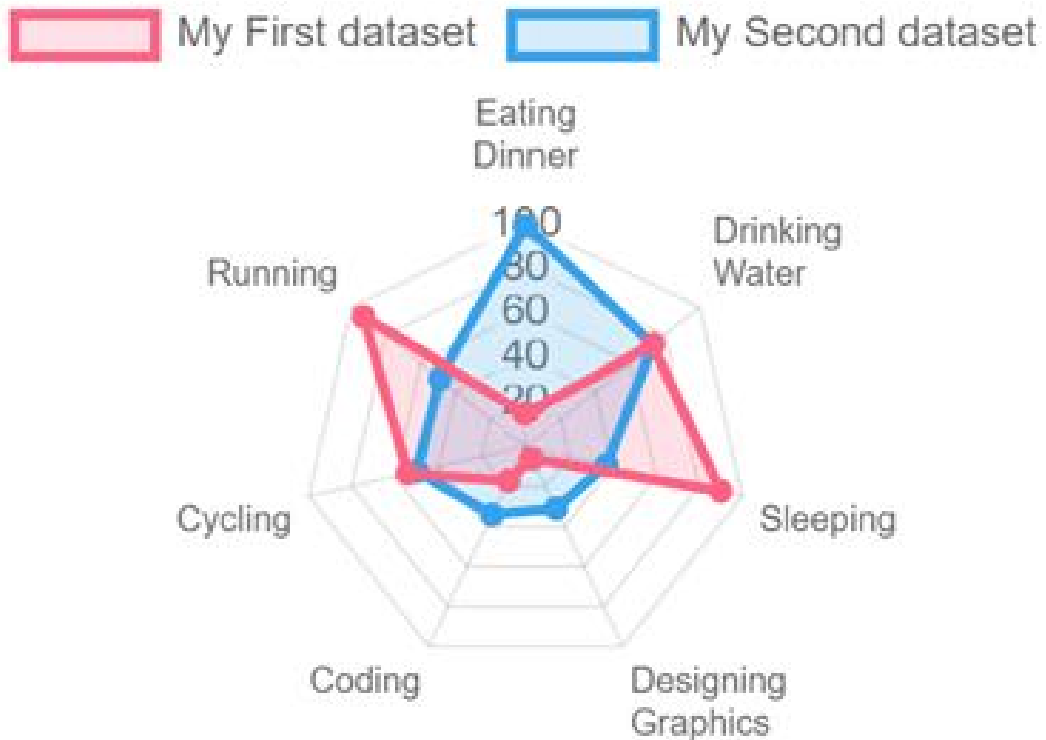


Fig. 7d Radar Chart

**NOTE:** the above example images represent the visual style used in the application but contain dummy data unrelated to the domain.

All 4 roles (Super Admin, Supervisor, Doctor and Patient) can view leaderboards on the web portal. There are no global leaderboards (there is no meaningful way the data can be aggregated), the user must always select a game first. Then they can choose which game-specific board (variable) they want to see and select a time range (this week, this month, all-time).

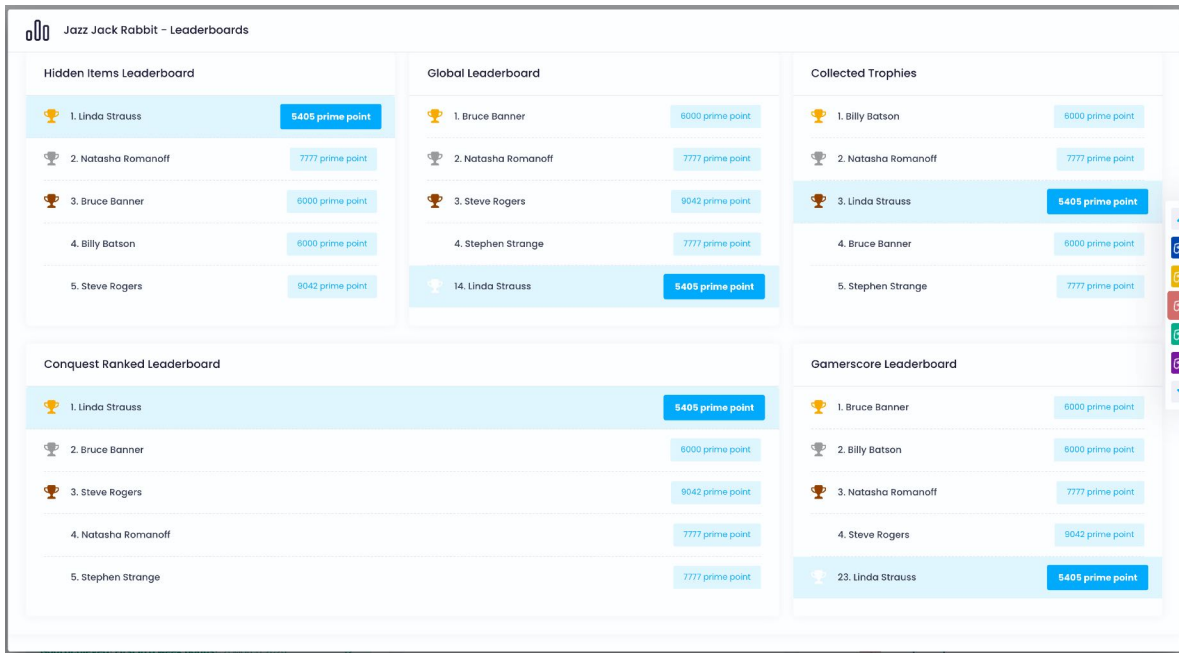


Fig. 7e Leaderboard Pop-up

All 4 roles (Super Admin, Supervisor, Doctor and Patient) can view charts on the web portal. Charts are always user- and game-specific. Super Admins, Supervisors and Doctors can open a User's charts for a specific game from the Game Details Pop-up. Patients have a separate Charts menu item.

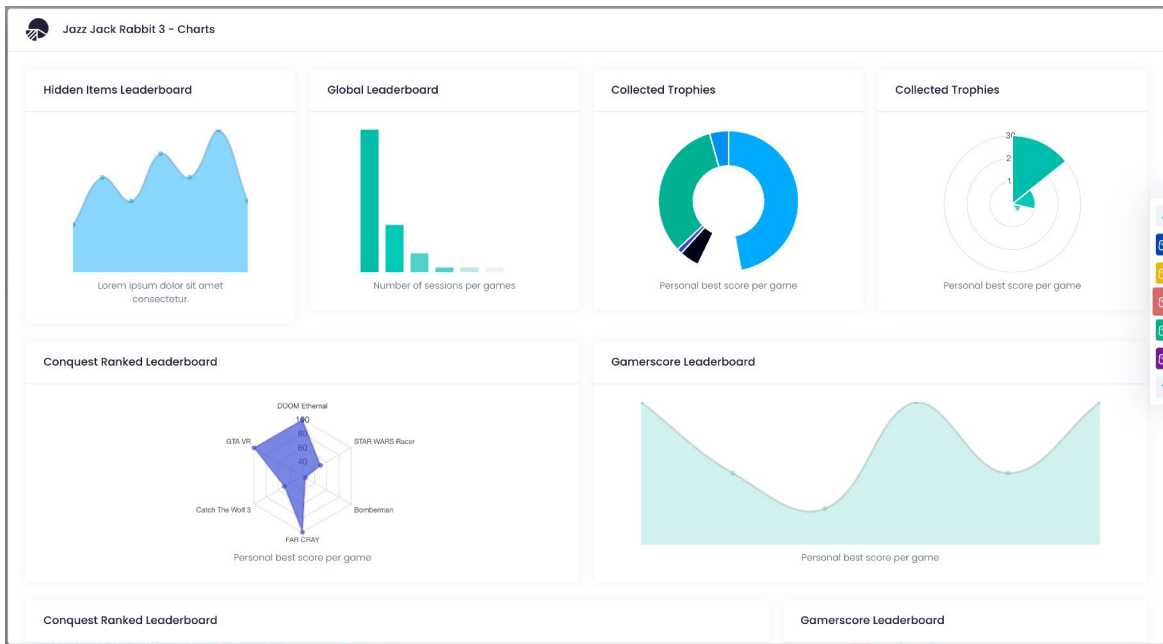


Fig. 7f Chart Pop-up

## 8 PERFORMANCE METRICS

WP6 recommended the initial KPIs to be used for games in the VRHAB-IT. The four-performance metrics that will be used to benchmark each game developed during this project will also be used to judge the quality of the games submitted by third-party developers on VR-HABIT platform. These are:

1. **Retention rate** – in this metric we look for how long patients keep coming back to the platform. There could be many reasons why patients walk-away from the platform so this is the top-most important metric to look for on a daily basis. It is defined as a percentage of all patients who come back after Day-1, Day-3, Day-7, Day-14 and Day-30.
2. **Number of daily sessions** – Building on the retention rate, in this metric, we also look for the number of sessions each player played per day. Please note that the target for this metric might be different for each user for each different game and is hence set by their therapist. This value could also be less than 1 if the prescription defines less than 1 session per day (e.g. 3 sessions per week)
3. **Session duration** - a session is the time from when the player starts playing to the moment when the activity is suspended. This metric will give us a better understanding of how players are doing when compared to traditional therapy methods and if the games are interesting enough to keep the players engaged for a longer period of time.
4. **Engagement score** - the engagement score is another way of measuring the user satisfaction. In this metric we'll try and see what the players are doing while playing the game. For instance, are they using the most basic feature only, are they exploring around menus or environments to look for additional content and do they replay the same game over and over again to get a better score? We define this metric in terms of areas traversed with points given to each one according to the difficulty of finding that particular place and summing the values up.

## 9 VALIDATION CRITERIA

**D2.1 VR Games and Activities** sub section '**Skill learning, performance measurement and Gameplay**' defines how each game evolves over time and how the difficulty is set in such a way so as to remain challenging for VRHAB-IT users. In order to support those requirements, the team is tracking some metrics to help understand the user behaviour and identify any choke points within the games and activities.

**Start, fail and completion metrics** – We measure these values for any given game. For every *start* event, we see whether the player *achieved* or *failed* their goals. This information will be valuable to the game developers to help adjust difficulty as well as to therapists who might adjust the targets for their patients.

At WP6, we have been discussing ways to validate each activity within the VR-HABIT platform. This is traditionally done through play testing in focus groups but this has 2 drawbacks – it is not accurate and it might not even be feasible in our domain due to the conditions of our patients. The conclusion we came up with is that each activity will be validated with the same conditions to understand how much it helps patients get better in their needs when compared to traditional approaches. This is very important in order to develop trust within the community.

To achieve this, we decided to build a flexible system that empowers the living labs to define the expected outcomes by allowing them to fine tune properties for each given task at a global level as well as at a personal level for each patient.

These properties include the following:

1. **Difficulty** – A float data type between 0 and 1 where 1 is the hardest. *This is interpreted differently for each game.*
2. **Session Duration** – An integer data type that defines the target session length in seconds.
3. **Number of sessions** – A float data type that defines the number of sessions a patient should strive for in a week.

These data values are compared with the real values recorded from player activity to help validate the real value for each given game.

## **10 CI/CD**

To ensure code quality and make the daily DevOps tasks easier, the project is using both continuous integration and continuous delivery.

Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied.

Continuous Delivery (CD) is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

The cloud build is connected to the Bitbucket repositories and after a successful build the artifact is ready to be deployed as a container. In production system the CD step obviously will be disabled.

During the prototype phase there is no need of a full fledged CI/CD process to be in place. This will be detailed and implemented after the commercialization phase.

## **11 CONCLUSION**

The Platform Implementation Plan has outlined all the details about the Web Portal, PrimeVR2 Unity SDK and the VR Ecosystem (Rehabilitation Games). It also defined the terminology and coding standards to align the work between WP6 members.

The Web Portal, PrimeVR2 Unity SDK, and the Rehabilitation Games create an integrated environment. Within this ecosystem, the Doctors and their Patients can work seamlessly to achieve their rehabilitation goals. Because every data from the games are uploaded into the Web Portal real-time, the analysis is much easier even remotely.

The system remains extendable in the future with new games and metrics. Later goals can be to extend the system with multiplayer functions or third-party developers.

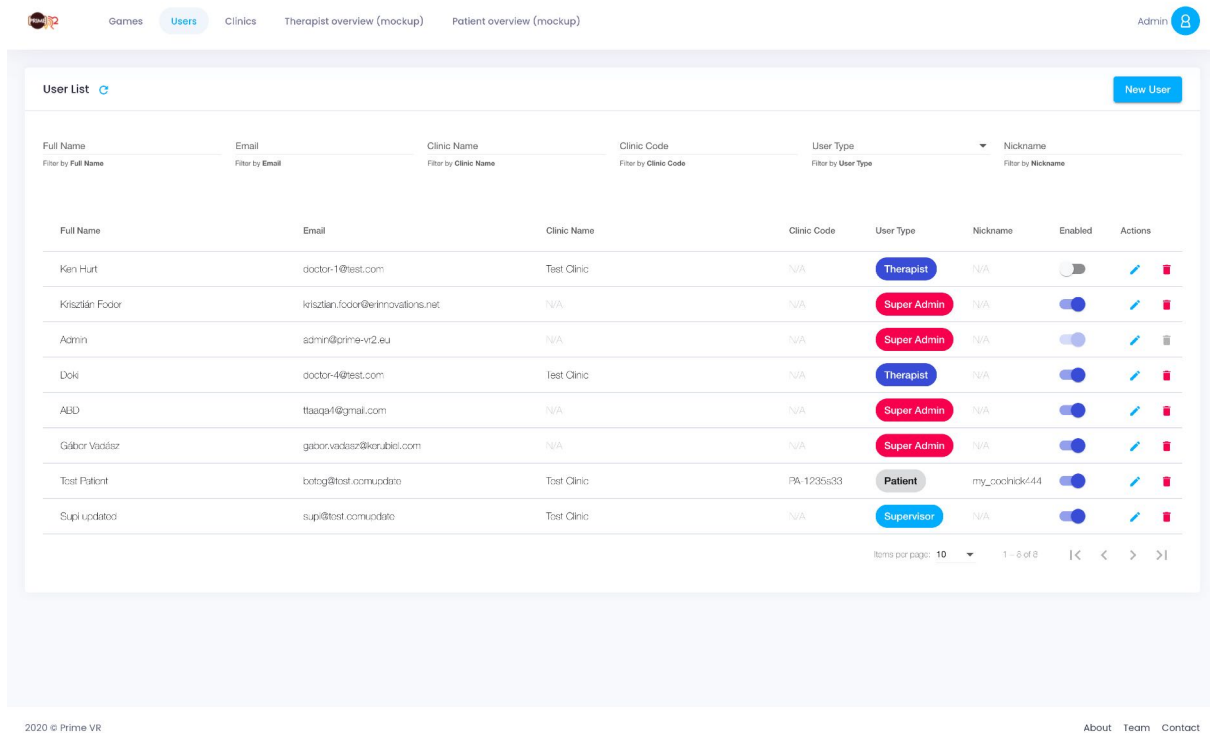
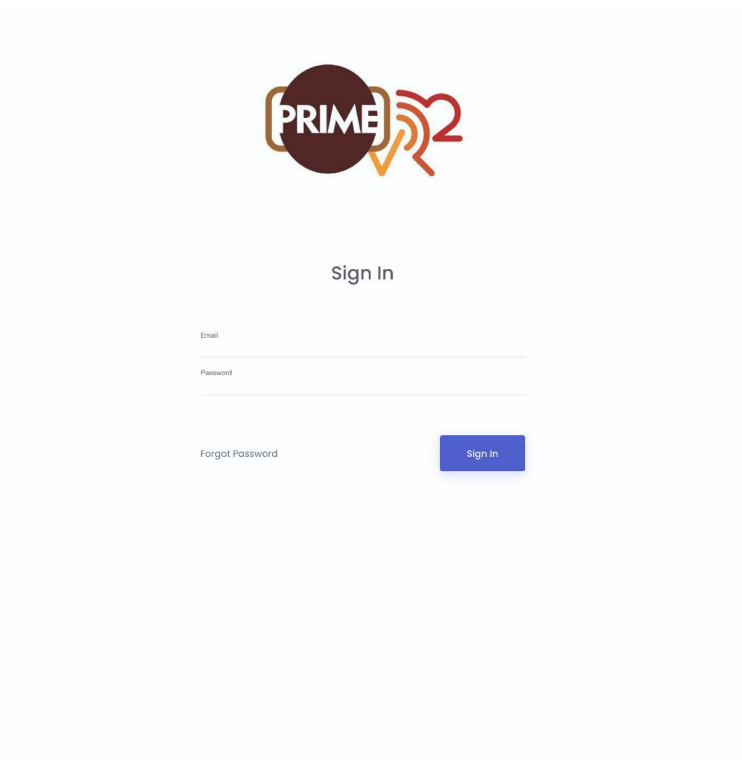
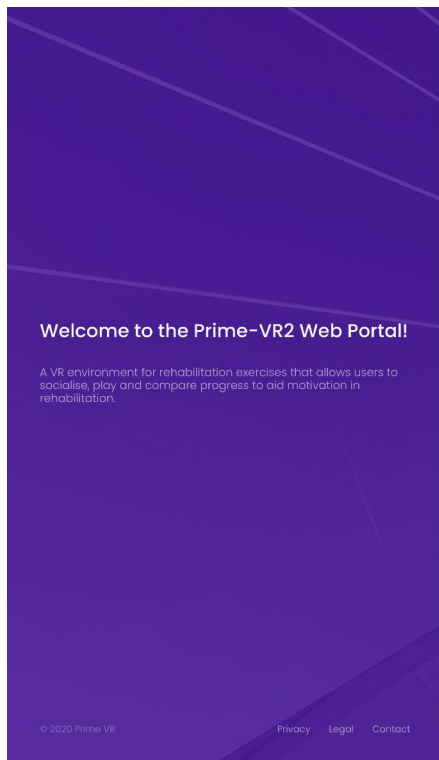


Keep in mind in the implementation phase there can be slight changes in the execution. This is mostly related to late information from other Work Packages or findings that were not previously known.

The next step is to execute the plan and implement all components of the described ecosystem.

# APPENDIX 1.

## 12 USER INTERFACE MOCK-UPS





### Game List

[New Game](#)

Search

Filter by Name

ID ↑	Name	Recommended playtime per day	Recommended sessions per week	Actions
1	Test Game	30	3	
2	Test Game2	10	5	

Items per page: 10 1 - 2 of 2 |&lt; &lt; &gt; &gt;|



### Clinics

[New Clinic](#)

Clinic name

Filter by Clinic Name

Country

Filter by Country

City

Filter by City

Name	Country	City	Actions
Test Clinic	Hungary	Budapest	

Items per page: 10 1 - 1 of 1 |&lt; &lt; &gt; &gt;|

Games Users Clinics **Therapist overview (mockup)** Patient overview (mockup) Admin

**John Wick**  
The Boogeyman  
email | baba\_yaga\_ofcl@gmail.com  
hospital | Princeton-Plainsboro Hospital

Interaction Game Details Configure Game

Selected game: **Jazz Jack Rabbit**

**Game Library**

All games: All games summary  
 Batman - The Shopping list: Batman statistics  
 PacMan: Adventure game statistics  
 Jazz Jack Rabbit: Rabbit statistics  
 Doom Eternal: Doom Eternal

**Activity Summary**  
Lorem ipsum dolor sit amet consectetur.

**Most played games**  
Number of sessions per games

**Best Scores**  
Personal best score per game

**Best Scores II.**  
Personal best score per game

**Visited hidden areas**

**Collected items**

https://app-staging-prime-vr2.eu/therapist/overview

Games Users Clinics **Therapist overview (mockup)** Patient overview (mockup) Admin

**Game Library**

Selected game: **Jazz Jack Rabbit**

All games: All games summary  
 Batman - The Shopping list: Batman statistics  
 PacMan: Adventure game statistics  
 Jazz Jack Rabbit: Rabbit statistics  
 Doom Eternal: Doom Eternal

**Charts**

**Achievements**

**Leaderboard**

**Daily goals**

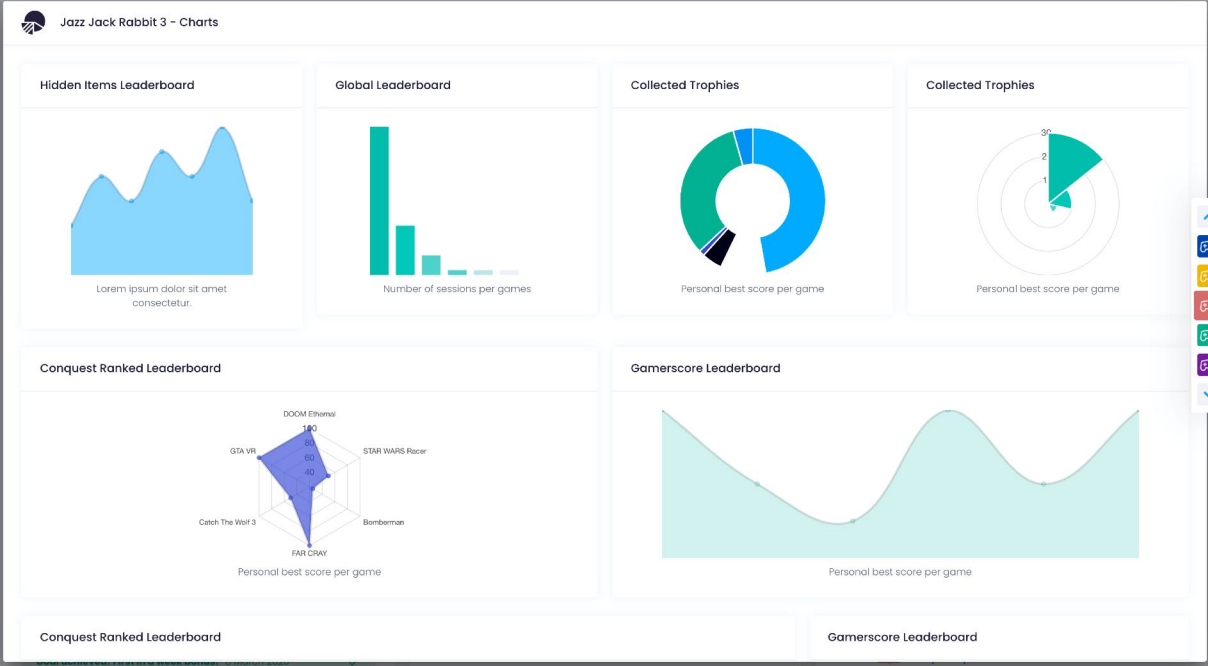
- Goal achieved! Today ✓
- Goal achieved! Yesterday ✓
- Goal achieved! 3 in a row! 12 March 2020 ✓
- Daily goal is incomplete. 11 March 2020 ✗
- Daily goal is incomplete. 10 March 2020 ✗
- Goal achieved! Like a Pro! 9 March 2020 ✓
- Goal achieved! First in a week bonus! 8 March 2020 ✓

**Weekly goals**

- Weekly goal incomplete. This week Missed
- Weekly goal incomplete. Last week Missed
- Weekly mission completed! 24 Febr 2020 - 1 March 2020 Completed
- Weekly goal achieved! First in a month bonus! 17 Febr 2020 - 23 Febr 2020 Completed

**Patient Activity**

- 2020.03.14 2020 DOOM Eternal +1001 prime points
- 2020.03.13 12:21 PM Jazz Jack Rabbit Reloaded +900 prime points
- 2020.03.12 Mario Infinity +40 points
- 2020.03.11 DOOM Eternal +63 prime points
- 2020.03.8 DOOM Eternal +154 prime points
- 2020.03.14 DOOM Eternal +1001 prime points
- 2020.03.13 Jazz Jack Rabbit Reloaded +900 prime points
















**Jazz Jack Rabbit - Leaderboards**

Hidden Items Leaderboard	Global Leaderboard	Collected Trophies
1. Linda Strauss <b>5405 prime point</b>	1. Bruce Banner <b>6000 prime point</b>	1. Billy Batson <b>6000 prime point</b>
2. Natasha Romanoff <b>7777 prime point</b>	2. Natasha Romanoff <b>7777 prime point</b>	2. Natasha Romanoff <b>7777 prime point</b>
3. Bruce Banner <b>6000 prime point</b>	3. Steve Rogers <b>9042 prime point</b>	3. Linda Strauss <b>5405 prime point</b>
4. Billy Batson <b>6000 prime point</b>	4. Stephen Strange <b>7777 prime point</b>	4. Bruce Banner <b>6000 prime point</b>
5. Steve Rogers <b>9042 prime point</b>	14. Linda Strauss <b>5405 prime point</b>	5. Stephen Strange <b>7777 prime point</b>

Conquest Ranked Leaderboard	Gamerscore Leaderboard
1. Linda Strauss <b>5405 prime point</b>	1. Bruce Banner <b>6000 prime point</b>
2. Bruce Banner <b>6000 prime point</b>	2. Billy Batson <b>6000 prime point</b>
3. Steve Rogers <b>9042 prime point</b>	3. Natasha Romanoff <b>7777 prime point</b>
4. Natasha Romanoff <b>7777 prime point</b>	4. Steve Rogers <b>9042 prime point</b>
5. Stephen Strange <b>7777 prime point</b>	23. Linda Strauss <b>5405 prime point</b>

**Jazz Jack Rabbit 2 - Achievements**

 Runner heart	Unlocked achievement: Yesterday
 Prime coin master 40%	40%
 Win without lose any life	Unlocked achievement: 2020.03.12.
 Collect all medats (3/5) collected	
 Speed runner Win level 3 under 3min challenge	Unlocked achievement: 2020.02.21.
 Find all hidden areas	40%
 Kill multiple enemies with one shot	Unlocked achievement: 2020.01.02.
 Runner heart	Unlocked achievement: Yesterday
 Prime coin master 40%	40%
 Win without lose any life	Unlocked achievement: 2020.03.12.
 Collect all medats (3/5) collected	

## **APPENDIX 2.**

### **13 CODING GUIDELINES**

#### **13.1. Introduction**

---

These documents contain guidelines for writing consistent, lucid, enticing, modern C#.

If you take issue with anything here, please open a pull request with your recommended changes and include an argument for *and against* their adoption; explain the benefits of your proposed change, and also any drawbacks.

#### **13.2. Guiding Principles**

---

- Be consistent.
- Don't rewrite existing code to follow this guide.
- Don't violate a guideline without a good reason.
- A reason is good when you can convince a teammate, not just when you like it.
- Assume your reader knows C# and English.
- Prefer clarity to 'performance'.
- Prefer clarity to .NET dogma.
- Write comments that people want to read, with correct spelling and grammar.

#### **13.3. The Rundown**

---

- Indent with tabs.
- Max line length is 100 columns.
- Use spaces and empty lines precisely.
- Braces generally go on their own lines.
- Never put a space before `[`.
- Always put a space before `{`.
- Always put a space before `(` (except for method invocations or when following another `(`).

#### **13.4. General Guidelines**

---

##### **13.4.1. File Layout**

Layout your `.cs` files like this:

File Header

Using Directives

Namespace Declaration

Type Declaration

- Constants
- Static Fields
- Static Auto-Properties
- Static Delegates
- Static Events
- Static Enums
- Static Constructors
- Static Complex Properties
- Static Methods
- Static Structs
- Static Interfaces
- Static Classes
- Fields
- Auto-Properties
- Delegates
- Events
- Enums
- Constructors
- Finalizers (Destructors)
- Complex Properties
- Methods
- Structs
- Interfaces
- Classes

Within each of these groups order by access:

- public
- internal
- protected
- private

An exception to this layout is manual properties with a backing field used exclusively via the property; these members should occur in the file together in the properties section. If your backing field is accessed anywhere other than inside the property definition, stick to normal layout rules.

```
string name;
public string Name {
    get { return name; }
    set { name = value; }
}
```

### 13.4.2. using Directives

Group using directives by common prefix, with shorter namespaces coming before longer ones, creating neat clusters of statements separated by single empty lines.



Namespaces should be ordered in increasing order of platform specificity, with .NET namespaces first, then library or component namespaces, then Xamarin namespaces, then application namespaces:

```
// Beautiful:
using System;
using System.Linq;
using System.Collections.Generic;

using MyLib;
using MyLib.Extensions;

using MonoTouch.UIKit;
using MonoTouch.Foundation;

using MyApp;

// Disaster:
using MyLib.Extensions;
using MonoTouch.Foundation;
using System.Collections.Generic;
using System;
using System.Linq;
using MonoTouch.UIKit;
using MyLib;
```

Prune redundant namespaces aggressively.

### 13.4.3. Declaring Types

Leave an empty line between every type definition:

```
// Perfect.
namespace MyApp
{
    enum Direction { Left, Right }

    class ImportantThing
    {
        ...
    }
}

// Wrong - missing and empty line between type definitions.
namespace MyApp
{
    enum Direction { Left, Right }
    class ImportantThing
```

```

    {
        ...
    }
}

// Wrong - more than one empty line.
namespace MyApp
{
    enum Direction { Left, Right }

    class ImportantThing
    {
        ...
    }
}

```

Put a space before and after `:` when listing base classes and interfaces.

```

// Perfect.
class MyClass : BaseClass, IDoesThis
{
}

```

```

// Wrong.
class MyClass: BaseClass, IDoesThis
{
}

```

### Enums

Simple enums may be defined on a single line:

```
enum Edge { Left, Right, Bottom, Top }
```

Larger enums should list entries on separate lines and always end in a comma:

```
enum StringSplitOptions
{
    None = 0,
    RemoveEmptyEntries = 1,
}

```

## 13.5. Member Declarations

---

Leave an empty line before every method, property, indexer, constructor, and destructor:

```
class Person
{
    string name;
}

```

```

    public Person(string name)
    {
        this.name = name;
    }
}

```

Automatic properties don't need to be preceded by an empty line:

```

class Person
{
    string Name { get; set; }
    int Age { get; set; }

    ...
}

```

### 13.5.1. Methods

```

public async Task<string[]> Query<TDatabase>(User user, TDatabase database
, Role role = Role.Admin)
    : where TDatabase : IDatabase
{
}

```

### 13.5.2. Properties

Declare automatic properties on a single line with the exact spacing shown below:

```

// Perfect.
string Name { get; set; }

```

Simple properties may define *get* and *set* on a single line each, with *get* first:

```

// Perfect.
string Name {
    get { return name; }
    set { name = value; }
}

```

Also note the single spaces before and after *{*, and the space before *}*.

Complex properties go like this:

```

// Perfect.
string Name {
    get {
        return name;
    }
    set {
        name = value;
    }
}

```

```
}  
}
```

### 13.5.3. Type Inference

Use it. Less typing is almost always better than more typing, with some important exceptions.

Use *var* when the type is repeated on the right-hand side of the assignment:

```
// Perfect!  
var users = new Dictionary<UserId, User>();
```

```
// Bloated.  
Dictionary<UserId, User> users = new Dictionary<UserId, User>();
```

Don't use *var* for capturing the return type of a method or property when the type is not evident:

```
// Horrendous.  
var things = Interpret(data);
```

```
// Much better.  
HashMap<Thing> things = Interpret(data);
```

```
// Even better.  
var things = InterpretAs<Thing>(data);
```

Omit the type when using array initializers:

```
// Could be better:  
database.UpdateUserIds(new int[] { 1, 2, 3 });
```

```
// Better:  
database.UpdateUserIds(new [] { 1, 2, 3 });
```

### 13.5.4. Object and Collection Initializers

Use them.

For simple initializers, you may do a one-liner:

```
// Perfect.  
var person = new Person("Vinny") { Age = 50 };
```

```
// Acceptable.  
var person = new Person("Vinny") {  
    Age = 50,  
};
```

Omit the () when using parameterless constructors:

```
// Perfect.  
var person = new Person { Name = "Bob", Age = 75 };
```

```
// Wrong.  
var person = new Person() { Name = "Bob", Age = 75 };
```

In general, each expression should be on a separate line, and every line should end with a comma ,:

```
// Very nice collection initializer.  
var entries = new Dictionary<string, int> {  
    { "key1", 1 },  
    { "key2", 2 },  
};
```

```
// Very nice object initializer.  
var contact = new Person {  
    Name = "David Siegel",  
    SocialSecurityNumber = 123456789,  
    Address = "1234 Montgomery Circle Drive East",  
};
```

```
// Bad collection initializer - multiple entries on one line.  
var entries = new Dictionary<string, int> {  
    { "key1", 1 }, { "key2", 2 },  
};
```

### 13.5.5. Indentation

*switch* statements have the case at the same indentation as the *switch*:

```
switch (x) {  
case 'a':  
    ...  
case 'b':  
    ...  
}
```

### 13.5.6. Where to put spaces[1]

We prefer to put a space before an open parenthesis only in control flow statements, but not in normal method/delegate/lambda calls, or expressions. This makes method invocations stand out from simple logical groupings. For example, this is good:

```
// Flow control...  
if (awesome) ...  
foreach (var foo in foos) ...  
while (hazMonkeys) ...
```

```

// Logical grouping...
var result = b * (4 + i);

// Method invocation.
Foo(database);
Debug.Assert(5 + (3 * 4) && "laws of math are failing me");

// Consider
A = result ?? (int) compute (foo (b + 1));

// At first glance it looks very similar to:
A = result ?? (int) compute (foo) (b + 1);

// Whereas:
A = result ?? (int) compute(foo(b + 1));

// Looks more immediately distinct from
A = result ?? (int) compute(foo)(b + 1);

```

The reason for doing this is not completely arbitrary. This style makes control flow operators stand out more, and makes expressions flow better. The function call operator binds very tightly as a postfix operator. In some cases, such as when C# is embedded in Razor markup, inserting a space before an opening parenthesis will cause compilation to fail.

[1] Adapted from <http://llvm.org/docs/CodingStandards.html#spaces-before-parentheses>

Do not put a space before the left angle bracket in a generic type:

```

// Perfect.
var scores = new List<int>();

// Incorrect.
var scores = new List <int>();

```

Do not put spaces inside parentheses, square brackets, or angle brackets:

```

// Wrong - spaces inside.
Initialize( database );
products[ i ];
new List< int >();

```

Separate type parameters to generic types by a space:

```

// Excellent.
var users = new Dictionary<UserId, User>();

// Worthless.
var users = new Dictionary<UserId,User>();

```

Put a space between the type and the identifier what casting:

```
// Great.  
var person = (Person) sender;
```

```
// Bad.  
var person = (Person)sender;
```

### 13.5.7. Where to put braces

Inside a code block, put the opening brace on the same line as the statement:

```
// Lovely.  
if (you.Love(someone)) {  
    someone.SetFree();  
}
```

```
// Wrong.  
if (you.Love(someone))  
{  
    someone.SetFree();  
}
```

Omitting braces for single line if statements is fine, however braces are always acceptable:

```
// Lovely.  
if (you.Like(it))  
    it.PutOn(ring);
```

```
// Acceptable.  
if (you.Like(it)) {  
    it.PutOn(ring);  
}
```

Very short statements may be one-liners, especially when the body is a *return*:

```
// Lovely.  
if (condition) return;
```

```
// Acceptable, but a little complex for a one-liner.  
if (people.All(p => p.IsAdmin)) return new AdminPage();
```

```
// Wrong - too complex for a single line:  
if (people.Where(p => p.IsAdmin).Average(p => p.Age) > 21) return DrinkDis  
penser.FireWater;
```

Always use braces with nested or multi-line conditions:

```
// Perfect.  
if (a) {
```

```

    if (b) {
        code();
    }
}

```

*// Acceptable.*

```

if (a) {
    if (b)
        code();
}

```

*// Wrong.*

```

if (a)
    if (b)
        code ();

```

When defining a method, put the opening brace on its own line:

*// Correct.*

```

void LaunchRockets()
{
}

```

*// Wrong.*

```

void LaunchRockets() {
}

```

When defining a property, keep the opening brace on the same line:

*// Perfect.*

```

double AverageAge {
    get {
        return people.Average (p => p.Age);
    }
}

```

*// Wrong.*

```

double AverageAge
{
    get {
        return people.Average(p => p.Age);
    }
}

```

Notice how *get* keeps its brace on the same line.

For very small properties, you can compress things:



```
// Preferred.
int Property {
    get { return value; }
    set { x = value; }
}
```

```
// Acceptable.
int Property {
    get {
        return value;
    }
    set {
        x = value;
    }
}
```

Empty methods should have the body of code using two lines, in consistency with the rest:

```
// Good.
void EmptyMethod()
{
}
```

```
// These are wrong.
void EmptyMethod() {}

void EmptyMethod()
{}
```

Generic method type parameter constraints are on separate lines, one line per type parameter, indented once:

```
static bool TryParse<TEnum>(string value, out TEnum result)
    where TEnum : struct
{
    ...
}
```

If statements with else clauses are formatted like this:

good:

```
if (dingus) {
    ...
} else {
    ...
}
```

bad:

```

if (dingus)
{
    ...
}
else
{
    ...
}

```

bad:

```

if (dingus) {
    ...
}
else {
    ...
}

```

Namespaces, types, and methods all put braces on their own line:

```

// Correct.
namespace MyApp
{
    class FluxCapacitor
    {
        ...
    }
}

```

```

// Wrong - opening braces are not on their own lines.
namespace MyApp {
    class FluxCapacitor {
        ...
    }
}

```

To summarize:

Statement	Brace position
Namespace	new line
Type	new line
Methods	new line
Constructors	new line
Destructors	new line
Properties	same line
Control blocks (if, for...)	same line

Anonymous types and methods same line

### 13.5.8. Long Argument Lists

When your argument list grows too long, split your method invocation across multiple lines, with the first argument on a new line after the opening parenthesis of the method invocation, the closing parenthesis of the invocation on its own line at the same indentation level as the line with the opening parenthesis. This style works especially well for methods with named parameters.

```
// Lovely.
Console.WriteLine(
    "Connect to {0} via {1} with extra data: {2} {3}",
    database.Address,
    database.ConnectionMethod.Description,
    data.FirstPart,
    data.SecondPart
);
```

It's also acceptable to put multiple arguments on a single line when they belong together:

```
// Acceptable.
Console.WriteLine(
    "Connect to {0} via {1} with extra data: {2} {3}",
    database.Address,
    database.ConnectionMethod.Description,
    data.FirstPart, data.SecondPart
);
```

When chaining method calls, each method call in the chain should be on a separate line indented once:

```
void M() {
    IEnumerable<int> items = Enumerable.Range(0, 100)
        .Select(e => e * 2);
}
```

Use single spaces in expressions liberally:

good:

```
// Good.
if (a + 5 > method(blah()) + 4))
```

// Bad.

```
if (a+5>method(blah()+4))
```

### 13.5.9. Casing

Argument names should use the camel casing for identifiers, like this:

good:

```
// Good.  
void Method(string myArgument)
```

```
// Bad.  
void Method(string lpstrArgument)  
void Method(string my_string)
```

### 13.5.10. Instance Fields

Don't use `m_` or `_` as prefixes for instance fields. Just use normal parameter naming conventions:

```
// Perfect.  
class Person  
{  
    string name;  
}
```

```
// Wrong.  
class Person  
{  
    string m_name;  
}
```

Don't write `private` for private members, as this is the default visibility in C#:

```
// Perfect.  
class Person  
{  
    string name;  
}
```

```
// Wrong.  
class Person  
{  
    private string name;  
}
```

An exception to this rule is serializable classes. In this case, if we desire to have our serialized data be compatible with Microsoft's, we must use the same field name.

### 13.5.11. `this`

The use of "this." as a prefix in code is discouraged, it is mostly redundant. In general, since internal variables are lowercase and anything that becomes public starts with an uppercase letter, there is no ambiguity between what the "Foo" and "foo" are. The first is a public property or field, the second is internal property or field.

Good:

```
class Foo
{
    int bar;

    void Update(int newValue)
    {
        bar = newValue;
    }

    void Clear()
    {
        Update();
    }
}
```

Bad:

```
class Foo
{
    int bar;

    void Update(int newValue)
    {
        this.bar = newValue;
    }

    void Clear()
    {
        this.Update();
    }
}
```

An exception is made for *this* when the parameter name is the same as an instance variable, this happens sometimes in constructors or if naming is difficult:

Good:

```
class Message
{
    char text;

    public Message(string text)
    {
        this.text = text;
    }
}
```



## 14 CODE COMMENTING GUIDELINES

### 14.1. Simple Comments

---

Comments begin with `//` followed by a single space, use sentence casing, and exhibit proper spelling and grammar.

```
// Great:  
// Verify that the client and server states are consistent.
```

```
// Bad - missing space:  
//Verify that the client and server states are consistent.
```

```
// Bad - not a sentence:  
// verify client server states
```

If your comment just paraphrases code, remove it:

```
// Bad  
// Makes the window key and orders it front.  
window.MakeKeyAndOrderFront ();
```

### 14.2. Multiline comments

---

Long comments tend to grow from smaller ones, so it's simpler to always use `//` than to switch to `/* ... */` when a comment becomes "long".

```
// Good:  
  
// Sartorial leggings ennuï before they sold out banjo, lo-fi Truffaut  
// Shoreditch sustainable Godard skateboard next level iPhone. Locavore to  
usled  
// meh fingerstache DIY church-key keytar, Vice pug quinoa seitan. Blog ph  
oto  
// booth Pinterest Letterpress kogi leggings aesthetic irony.
```

```
// Bad:
```

```
/*  
* Sartorial leggings ennuï before they sold out banjo, lo-fi Truffaut  
* Shoreditch sustainable Godard skateboard next level iPhone. Locavore to  
usled  
* meh fingerstache DIY church-key keytar, Vice pug quinoa seitan. Blog ph  
oto  
* booth Pinterest Letterpress kogi leggings aesthetic irony.  
*/
```

## 14.3. Commenting Out Code

---

The only recommended use of `/* ... */`-style comments is for commenting out code. Please do not comment out multiple lines of code with `//`.

*// Good:*

```
/*
for (int i = 0; i < int.MaxValue; i++)
    Console.WriteLine (i);
*/
```

*// Bad:*

```
// for (int i = 0; i < int.MaxValue; i++)
//     Console.WriteLine (i);
```

You should avoid commenting out code anyway, preferring version control or other methods.

## 15 NAMING GUIDELINES

Object Name	Notation	Length	Plural	Prefix	Suffix	Abbreviation	Char Mask	Underscores
Class name	PascalCase	128	No	No	Yes	No	[A-z][0-9]	No
Constructor name	PascalCase	128	No	No	Yes	No	[A-z][0-9]	No
Method name	PascalCase	128	Yes	No	No	No	[A-z][0-9]	No
Method arguments	camelCase	128	Yes	No	No	Yes	[A-z][0-9]	No
Local variables	camelCase	50	Yes	No	No	Yes	[A-z][0-9]	No
Constants name	PascalCase	50	No	No	No	No	[A-z][0-9]	No
Field name	camelCase	50	Yes	No	No	Yes	[A-z][0-9]	Yes
Properties name	PascalCase	50	Yes	No	No	Yes	[A-z][0-9]	No
Delegate name	PascalCase	128	No	No	Yes	Yes	[A-z]	No
Enum type name	PascalCase	128	Yes	No	No	No	[A-z]	No

1. Do use *PascalCasing* for class names and method names:

```
public class ClientActivity
{
    public void ClearStatistics()
    {
        //...
    }
    public void CalculateStatistics()
    {
        //...
    }
}
```



```
}  
}
```

**Why: consistent with the Microsoft's .NET Framework and easy to read.**

2. Do use camelCasing for method arguments and local variables:

```
public class UserLog  
{  
    public void Add(LogEvent logEvent)  
    {  
        int itemCount = logEvent.Items.Count;  
        // ...  
    }  
}
```

**Why: consistent with the Microsoft's .NET Framework and easy to read.**

3. Do not use Hungarian notation or any other type identification in identifiers

```
// Correct  
int counter;  
string name;  
// Avoid  
int iCounter;  
string strName;
```

**Why: consistent with the Microsoft's .NET Framework and Visual Studio IDE makes determining types very easy (via tooltips). In general you want to avoid type indicators in any identifier.**

4. Do not use Screaming Caps for constants or readonly variables:

```
// Correct  
public const string ShippingType = "DropShip";  
// Avoid  
public const string SHIPPINGTYPE = "DropShip";
```

**Why: consistent with the Microsoft's .NET Framework. Caps grab too much attention.**

5. Use meaningful names for variables. The following example uses *seattleCustomers* for customers who are located in Seattle:

```
var seattleCustomers = from customer in customers  
    where customer.City == "Seattle"  
    select customer.Name;
```

**Why: consistent with the Microsoft's .NET Framework and easy to read.**

6. Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as *Id*, *Xml*, *Ftp*, *Uri*.

```
// Correct  
UserGroup userGroup;  
Assignment employeeAssignment;
```

```
// Avoid
UserGroup usrGrp;
Assignment empAssignment;
// Exceptions
CustomerId customerId;
XmlDocument xmlDocument;
FtpHelper ftpHelper;
UriPart uriPart;
```

**Why: consistent with the Microsoft's .NET Framework and prevents inconsistent abbreviations.**

*7. Do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase):*

```
HtmlHelper htmlHelper;
FtpTransfer ftpTransfer;
UIControl uiControl;
```

**Why: consistent with the Microsoft's .NET Framework. Caps would grab visually too much attention.**

*8. Do not use Underscores in identifiers. Exception: you can prefix private fields with an underscore:*

```
// Correct
public DateTime clientAppointment;
public TimeSpan timeLeft;
// Avoid
public DateTime client_Appointment;
public TimeSpan time_Left;
// Exception (Class field)
private DateTime _registrationDate;
```

**Why: consistent with the Microsoft's .NET Framework and makes code more natural to read (without 'slur'). Also avoids underline stress (inability to see underline).**

*9. Do use predefined type names (C# aliases) like int, float, string for local, parameter and member declarations. Do use .NET Framework names like Int32, Single, String when accessing the type's static members like Int32.TryParse or String.Join.*

```
// Correct
string firstName;
int lastIndex;
bool isSaved;
string commaSeparatedNames = String.Join(", ", names);
int index = Int32.Parse(input);
// Avoid
String firstName;
Int32 lastIndex;
Boolean isSaved;
```

```
string commaSeparatedNames = string.Join(", ", names);
int index = int.Parse(input);
```

Why: consistent with the Microsoft's .NET Framework and makes code more natural to read.

*10. Do use implicit type var for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.*

```
var stream = File.Create(path);
var customers = new Dictionary();
// Exceptions
int index = 100;
string timeSheet;
bool isCompleted;
```

**Why: removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.**

*11. Do use noun or noun phrases to name a class.*

```
public class Employee
{
}
public class BusinessLocation
{
}
public class DocumentCollection
{
}
```

**Why: consistent with the Microsoft's .NET Framework and easy to remember.**

*12. Do prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.*

```
public interface IShape
{
}
public interface IShapeCollection
{
}
public interface IGroupable
{
}
```

**Why: consistent with the Microsoft's .NET Framework.**

*13. Do name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.*

```
// Located in Task.cs
public partial class Task
{
}
```

```
// Located in Task.generated.cs
public partial class Task
{
}
```

**Why: consistent with the Microsoft practices. Files are alphabetically sorted and partial classes remain adjacent.**

14. Do organize namespaces with a clearly defined structure:

```
// Examples
namespace Company.Product.Module.SubModule
{
}
namespace Product.Module.Component
{
}
namespace Product.Layer.Module.Group
{
}
```

**Why: consistent with the Microsoft's .NET Framework. Maintains good organization of your code base.**

15. Do vertically align curly brackets:

```
// Correct
class Program
{
    static void Main(string[] args)
    {
        //...
    }
}
```

**Why: Microsoft has a different standard, but developers have overwhelmingly preferred vertically aligned brackets.**

16. Do declare all member variables at the top of a class, with static variables at the very top.

```
// Correct
public class Account
{
    public static string BankName;
    public static decimal Reserves;
    public string Number { get; set; }
    public DateTime DateOpened { get; set; }
    public DateTime DateClosed { get; set; }
    public decimal Balance { get; set; }
    // Constructor
    public Account()
    {
```

```

    // ...
}
}

```

**Why: generally accepted practice that prevents the need to hunt for variable declarations.**

*17. Do use singular names for enums. Exception: bit field enums.*

*// Correct*

```

public enum Color
{
    Red,
    Green,
    Blue,
    Yellow,
    Magenta,
    Cyan
}

```

*// Exception*

```

[Flags]
public enum Dockings
{
    None = 0,
    Top = 1,
    Right = 2,
    Bottom = 4,
    Left = 8
}

```

**Why: consistent with the Microsoft's .NET Framework and makes the code more natural to read. Plural flags because enum can hold multiple values (using bitwise 'OR').**

*18. Do not explicitly specify a type of an enum or values of enums (except bit fields):*

*// Don't*

```

public enum Direction : long
{
    North = 1,
    East = 2,
    South = 3,
    West = 4
}

```

*// Correct*

```

public enum Direction
{
    North,
    East,
    South,
}

```

```
    West  
}
```

**Why: can create confusion when relying on actual types and values.**

19. Do not use an “Enum” suffix in enum type names:

```
// Don't  
public enum CoinEnum  
{  
    Penny,  
    Nickel,  
    Dime,  
    Quarter,  
    Dollar  
}
```

```
// Correct  
public enum Coin  
{  
    Penny,  
    Nickel,  
    Dime,  
    Quarter,  
    Dollar  
}
```

**Why: consistent with the Microsoft’s .NET Framework and consistent with prior rule of no type indicators in identifiers.**

20. Do not use “Flag” or “Flags” suffixes in enum type names:

```
// Don't  
[Flags]  
public enum DockingsFlags  
{  
    None = 0,  
    Top = 1,  
    Right = 2,  
    Bottom = 4,  
    Left = 8  
}
```

```
// Correct  
[Flags]  
public enum Dockings  
{  
    None = 0,  
    Top = 1,  
    Right = 2,  
    Bottom = 4,  
}
```

```
    Left = 8  
}
```

**Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.**

21. Do use suffix EventArgs at creation of the new classes comprising the information on event:

*// Correct*

```
public class BarcodeReadEventArgs : System.EventArgs  
{  
}
```

**Why: consistent with the Microsoft's .NET Framework and easy to read.**

22. Do name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:

```
public delegate void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e);
```

**Why: consistent with the Microsoft's .NET Framework and easy to read.**

23. Do not create names of parameters in methods (or constructors) which differ only by the register:

*// Avoid*

```
private void MyFunction(string name, string Name)  
{  
    //...  
}
```

**Why: consistent with the Microsoft's .NET Framework and easy to read, and also excludes possibility of occurrence of conflict situations.**

24. DO use two parameters named sender and e in event handlers. The sender parameter represents the object that raised the event. The sender parameter is typically of type object, even if it is possible to employ a more specific type.

```
public void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e)  
{  
    //...  
}
```

**Why: consistent with the Microsoft's .NET Framework**

**Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.**

25. Do use suffix *Exception* at creation of the new classes comprising the information on exception:

*// Correct*

```
public class BarcodeReadException : System.Exception
{
}
```

**Why: consistent with the Microsoft's .NET Framework and easy to read.**

26. Do use suffix *Any, Is, Have* or similar keywords for boolean identifier :

*// Correct*

```
public static bool IsNullOrEmpty(string value) {
    return (value == null || value.Length == 0);
}
```

## 16 LAMBIDAS

Lambdas are written with a single space before and after the =>:

*// Great.*

```
Func<int, int> square = i => i * i;
```

*// Terrible.*

```
Func<int, int> square = i=>i * i;
```

If your lambda takes a single argument, omit the parentheses around the argument list:

*// Great!*

```
var admins = Users.Select (user => user.IsAdministrator);
```

*// Silly.*

```
var admins = Users.Select ((user) => user.IsAdministrator);
```

Whenever possible, omit types from lambda argument lists, and use simple names:

*// Great:*

```
list.OnScroll += (sender, e) => {
    ...
};
```

*// Passé:*

```
list.OnScroll += (object sender, EventArgs e) => {
    ...
};
```

*// No! Parameter name is needlessly complex:*

```
sqlDatabaseAdaptors.Select (sqlDatabaseAdaptor => sqlDatabaseAdaptor.Id);
```

*// Much better. We have enough context from the larger identifier to know*



*what 'adaptor' is:*

```
sqlDatabaseAdaptors.Select (adaptor => adaptor.Id);
```

When the body of a lambda is a simple statement or expression, don't use a block:

```
// Excellent!
```

```
var averageSalary = employees.Average (employee => employee.Salary);
```

```
// Inconceivable!
```

```
var averageSalary = employees.Average (employee => { return employee.Salary; });
```

When the body of the lambda is a block, put the opening brace on the same line as the =>, indent the body of the block, and close the block at the same level of indentation as the line containing the opening brace:

```
// Ideal:
```

```
people.ForEach (person => {  
    person.BrushTeeth ();  
    person.CallMom ();  
    person.RegisterToVote ();  
});
```

```
// No! Improperly positioned opening brace:
```

```
people.ForEach (person =>  
{  
    person.BrushTeeth ();  
    person.CallMom ();  
    person.RegisterToVote ();  
});
```

```
// No! Improperly positioned closing brace:
```

```
people.ForEach (person => {  
    person.BrushTeeth ();  
    person.CallMom ();  
    person.RegisterToVote ();  
}  
);
```

```
// No! Bad indentation:
```

```
people.ForEach (person => { person.BrushTeeth ();  
                             person.CallMom ();  
                             person.RegisterToVote ();  
});
```

Always prefer lambdas, *Func<>*, and *Action<>* types to *delegate*. The only recommended use of *delegate* is when the body of your anonymous method doesn't reference any of its arguments:

```
thing.EventWithSenderAndEventArgs += delegate {  
    Console.WriteLine ("EventWithSenderAndEventArgs raised.");  
};
```

It is acceptable to use single-character argument names in lambdas if the receiver is an *IEnumerable* and is named in such a way as to make the lambda argument obvious, and the lambda argument name is the first character of the receiver's identifier:

*// Acceptable:*

```
var averageSalary = employees.Average (e => e.Salary);
```

*// Acceptable:*

```
var averageSalary = employees.Average (employee => employee.Salary);
```

*// Wrong - parameter name doesn't correspond to collection name:*

```
var averageSalary = employees.Average (x => x.Salary);
```

## 17 COMMIT MESSAGE CONVENTIONS

These rules are adopted from [the AngularJS commit conventions](#).

### 17.1. Goals

---

- allow generating CHANGELOG.md by script
- allow ignoring commits by git bisect (not important commits like formatting)
- provide better information when browsing the history

### 17.2. Generating CHANGELOG.md

---

We use these three sections in changelog: new features, bug fixes, breaking changes. This list could be generated by script when doing a release. Along with links to related commits. Of course you can edit this change log before actual release, but it could generate the skeleton.

List of all subjects (first lines in commit message) since last release:

```
git log <last tag> HEAD --pretty=format:%s
```

New features in this release

```
git log <last release> HEAD --grep feature
```

#### 17.2.1. Recognizing unimportant commits

These are formatting changes (adding/removing spaces/empty lines, indentation), missing semi colons, comments. So when you are looking for some change, you can ignore these commits - no logic change inside this commit.

When bisecting, you can ignore these by:

```
git bisect skip $(git rev-list --grep irrelevant <good place> HEAD)
```

#### 17.2.2. Provide more information when browsing the history

This would add kinda “context” information. Look at these messages (taken from last few angular’s commits): \* Fix small typo in docs widget (tutorial instructions) \* Fix test for scenario.Application - should remove old iframe \* docs - various doc fixes \* docs - stripping extra new lines \* Replaced double line break with single when text is fetched from Google \* Added support for properties in documentation

All of these messages try to specify where is the change. But they don’t share any convention...

Look at these messages: \* fix comment stripping \* fixing broken links \* Bit of refactoring \* Check whether links do exist and throw exception \* Fix sitemap include (to work on case sensitive linux)

Are you able to guess what’s inside ? These messages miss place specification... So maybe something like parts of the code: docs, docs-parser, compiler, scenario-runner, ...

I know, you can find this information by checking which files had been changed, but that's slow. And when looking in git history I can see all of us tries to specify the place, only missing the convention.

---

## 17.3. Format of the commit message

---

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier to read on github as well as in various git tools.

### 17.3.1. Subject line

Subject line contains succinct description of the change.

*Allowed <type>*

- feat (feature)
- fix (bug fix)
- docs (documentation)
- style (formatting, missing semi colons, ...)
- refactor
- test (when adding missing tests)
- chore (maintain)

*Allowed <scope>*

Scope could be anything specifying place of the commit change. For example \$location, \$browser, \$compile, \$rootScope, ngHref, ngClick, ngView, etc...

*<subject> text*

- use imperative, present tense: “change” not “changed” nor “changes”
- don't capitalize first letter
- no dot (.) at the end

### 17.3.2. Message body

- just as in use imperative, present tense: “change” not “changed” nor “changes”
- includes motivation for the change and contrasts with previous behavior

<http://365git.tumblr.com/post/3308646748/writing-git-commit-messages>

<http://tbagery.com/2008/04/19/a-note-about-git-commit-messages.html>

### 17.3.3. Message footer

#### *Breaking changes*

All breaking changes have to be mentioned in footer with the description of the change, justification and migration notes

BREAKING CHANGE: isolate scope bindings definition has changed and the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

```
scope: {
  myAttr: 'attribute',
  myBind: 'bind',
  myExpression: 'expression',
  myEval: 'evaluate',
  myAccessor: 'accessor'
}
```

After:

```
scope: {
  myAttr: '@',
  myBind: '@',
  myExpression: '&',
  // myEval - usually not useful, but in cases where the expression is
  // assignable, you can use '='
  myAccessor: '=' // in directive's template change myAccessor() to my
  Accessor
}
```

The removed `inject` wasn't generally useful for directives so there should be no code using it.

#### *Referencing issues*

Closed bugs should be listed on a separate line in the footer prefixed with "Closes" keyword like this:

Closes #234

or in case of multiple issues:

Closes #123, #245, #992

## 17.4. Examples

---

feat(\$browser): onUrlChange event (popstate/hashchange/polling)

Added new event to \$browser:

- forward popstate event if available
- forward hashchange event if popstate not available
- do polling when neither popstate nor hashchange available

Breaks \$browser.onHashChange, which was removed (use onUrlChange instead)

fix(\$compile): couple of unit tests for IE9

Older IEs serialize html uppercased, but IE9 does not...

Would be better to expect case insensitive, unfortunately jasmine does not allow to user regexps for throw expectations.

Closes #392

Breaks foo.bar api, foo.baz should be used instead

feat.directive): ng:disabled, ng:checked, ng:multiple, ng:readonly, ng:selected

New directives for proper binding these attributes in older browsers (IE). Added corresponding description, live examples and e2e tests.

Closes #351

style(\$location): add couple of missing semi colons

docs(guide): updated fixed docs from Google Docs

Couple of typos fixed:

- indentation
- batchLogbatchLog -> batchLog
- start periodic checking
- missing brace

feat(\$compile): simplify isolate scope bindings

Changed the isolate scope binding options to:

- @attr - attribute binding (including interpolation)
- =model - by-directional model binding
- &expr - expression execution binding

This change simplifies the terminology as well as number of choices available to the developer. It also supports local name aliasing from the parent.

BREAKING CHANGE: isolate scope bindings definition has changed and the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

```
scope: {  
  myAttr: 'attribute',  
  myBind: 'bind',  
  myExpression: 'expression',  
  myEval: 'evaluate',  
  myAccessor: 'accessor'  
}
```

After:

```
scope: {  
  myAttr: '@',  
  myBind: '@',  
  myExpression: '&',  
  // myEval - usually not useful, but in cases where the expression is assignable, you can use '='  
  myAccessor: '=' // in directive's template change myAccessor() to myAccessor  
}
```

The removed `inject` wasn't generally useful for directives so there should be no code using it.

## **APPENDIX 3.**

### **ENVIRONMENTS CHECKLIST**

Please use the below template to ensure cohesion and consistency across all Environment definition documents.

#### **Visual**

- What does the player see around him/her?
- What object(s) can the player interact with
- What object(s) cannot the player interact with
  - Pick up
  - Aim
  - Cut/Trim
  - Put down
- How big is the environment?
- Define layout of the environment
- Define reasons for each of the above
- Can the environment support more than 1 person? (That is: someone to assist player)
- How could stress impact the patient in the game? How will it be monitored?
- How can the environments be inclusive?
  - Use captions?
  - How can it be inclusive for people with vision impairment? Colour arrangements in settings?
  - How are colorblind people going to see it? Include different color modes?
  - Use settings to customize the light/dark environment for each patient?
  - [Reference](#)

#### **Audio**

- Define background sounds
- Define sound effects
- Define character sounds
- Define movement sounds
- Define sound made by the interaction between player and object
- Define objects sounds
- Describe why these sounds are important in these cases
- How can the environment's sounds be inclusive?
  - Guided instructions
  - Description of sound events



- Illustrated instructions

## Physical Feedback

- What feedback can be given from the controller to help the patient?

## Patient-centric Analysis

*Questions related to movement and player for sport, stroke and dystonia patients.*

### **Case 1:** Sport injuries (*elbow, wrist, hand*)

This kind of patient can move around, the focus of the injuries is the arm, and they need to exercise it. This kind of patient is the least game constrained when it comes to in-game physical actions.

### **Case 2:** Stroke patients (*brain injury*)

Patient has lost the connection to the brain-muscle parts of the body, usually half of it is “frozen”. Thanks to exercise they can partially regain this connection; the majority of the patients are elderly people with little mobility. A game that has one direction or that allows to move their heads (max) is required.

### **Case 3:** Dystonia (*muscle problem*)

These kinds of patients have little control over their movements, and they require a simple environment where they can focus on one simple task.

For each case type consider these questions:

1. Can you move?
2. Do you play standing up or sitting down?
3. What movements do you need to do?
4. What parts of the body do you need to play? (Interaction necessities)
5. Do you need one or two hands?
6. How is the player represented in this environment?
7. How is workload on the body assessed?
8. What is the relationship between a patient's movements and the controller?
9. Will people with serious movement impairment (such as they need to lay down) be handled?

## APPENDIX 4

### PRIMEVR2 ENVIRONMENTS



*In this document we describe a total of 10 potential VR environments which we deem suitable for the PrimeVR-2 rehabilitation platform. For each of these environments we outline the following aspects:*

- **Visuals:** This section establishes a theme and sets the scene for various objects and other virtual entities around the Player. These include static environment objects as well as interactive props. Each layout is designed to provide a space where players can feel safe, in-control while reducing unnecessary movement which can be challenging considering the spectrum of patients that will inhabit such space. Environments range from common day-to-day locales to fictional ones as we believe it gives us greater flexibility to reach a larger number of patients.
- **Modality:** Provides a definition on how the environments interface with the various activity stations at play and for each it lists how the patient will be positioned to play. Given the physical impairments of some of the patient classes we have opted where possible to support the sitting down position. This will give us the possibility of re-using the same environment across the whole spectrum of patients in VRHAB-IT.
- **Assistance:** Patients may be unable to perform specific tasks so for this reason we have listed several possibilities within the game space of how these tasks can be still be overcome with the assistance of a third-party.
- **Social:** Different activities provide different social interaction potential. In this section we outline any potential ways in which the patient, his actions or even his creations can be used to interact socially with other players.
- **Supported Activities:** Most importantly in this section we give a sample of the activities that can be performed by the patient to drive the gameplay. Here we define each activity while clearly outlining how the patient is supposed to interact it and thus

establishing the requirements for the custom controller. These activities together with the aspects are designed to engage the patient in a relaxing, yet challenging, environment to drive the rehabilitation process.

## 1. GARDEN/ GREENHOUSE ENVIRONMENT

Controller Requirements / Activity	Plant Potting	Tools	Containers	Other Tools	Storage Handling/Transport	Note Taking
Finger Tracking	Nice to have	Nice to have		Nice to have		
Palm Open/Close	Required	Required	Required	Nice to have		Required
Wrist Movement	Required	Required	Required	Required	Required	Required
Elbow Movement	Required	Required	Required	Required	Required	Required
Wrist Resistance			Nice to have		Nice to have	
Elbow Resistance			Nice to have		Nice to have	



Table: Controller Requirement Matrix

A garden or a greenhouse environment is an all-rounder for most patients. The aim is to create a familiar, relaxing and stimulating environment that could change according to the players' action and does not bore them, making them come back more often. For example, the player could plant seeds and come back the next session to see them germinate and have a sprout. The weather and sound could also change according to the session even though this has no impact on the player's actions. It is also for this reason that we feel a roofed Greenhouse is preferred in this case. Gardening in VR can be a good tool to cheat the personal mental barriers of the brain for performing activities. It will create an artificial environment and show the patient a different set up than the hospital. The enthusiasm of the patient can push the patient towards a barrier free performance of actions which are targeted by the therapist.

Patients tend to be in a very fragile state after a stroke therefore offering them a familiar environment will facilitate their getting used to the VR environment. The garden is a small typical outdoor place surrounded by a fence/brick wall and with a lot of green

and props (such as fountains and other decorative items). On the other hand, the greenhouse is an indoor environment with a lot of potted plants and props.

In this environment, a carer can assist the player with some of the activities that take place. We will explain the assistance functions later this document.

### Areas challenged by **VR immersive environment**:



### **Description of the environment and sounds:**

Garden: birds chirping, wind moving tree's branches, insects flying, (bees, flies, grasshoppers, crickets, cicadas)

References: [Garden Sounds](#) , [Garden sound 2](#) , [Windy garden](#)

[Greenhouse sound](#) (Sounds are the same as the garden but with some reverb to give the feeling that player is sheltered) [Herbology greenhouse](#),

- **Character sounds**: Gasping when doing challenging tasks, exclamations when doing something good or happy noises to motivate the patients.
- **Movement sounds**: Clothes rub [Sound example](#)
- **Sound effects**: *Sound made by the interaction between player and object*: when picking up an object, when objects fall and when objects touch each other.

---

<sup>1</sup> <https://www.saebo.com/>

Sound is extremely important to grant the player a realistic experience because it helps the immersion in the virtual environment, the more accurate it is, the better the experience. Sound is a very powerful tool and can be used to ease the patient's therapy and accompany the experience as it can help reaching a relaxed state where the patient is encouraged to exercise and left with a positive feeling at the end of the session.

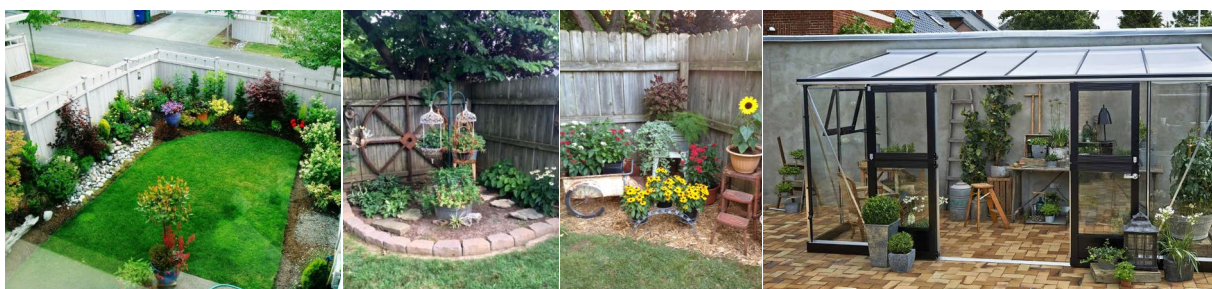
VR video games can be used to distract a patient, alleviating some of the experience of pain during tasks that can be excruciating. In this very case a bird chirping combined with summer spring or summer sounds and the relaxing activities can put the patient into a cheerful state which can push the patient to perform better.

As for effects, giving clothes and object sounds whenever they happen to move increases the realism. The sounds made by the "VR Player" when something is accomplished help the patient feel good and not give up. A cheer after a particularly difficult task, like putting a compost bag on the table, will make the patient feel more powerful therefore much likely to perform the same task again. Any sound concerning failure must be avoided because the patient will feel powerless and unable to do and will quickly lose interest in the game and the therapy. *See Positive feedback loops<sup>2</sup>.*

Audio and Video can be combined and reinforced by force feedback from the controller. In rehabilitation, force feedback could prove to be very useful, as it could provide feedback to a patient undergoing treatment by simulating the presence of solid objects (picking up compost bags/heavy pot/moving soil) in the virtual environment in order to sustain reproducing daily activities.

## **Visual**

Reference: [Greenhouse tour](#)



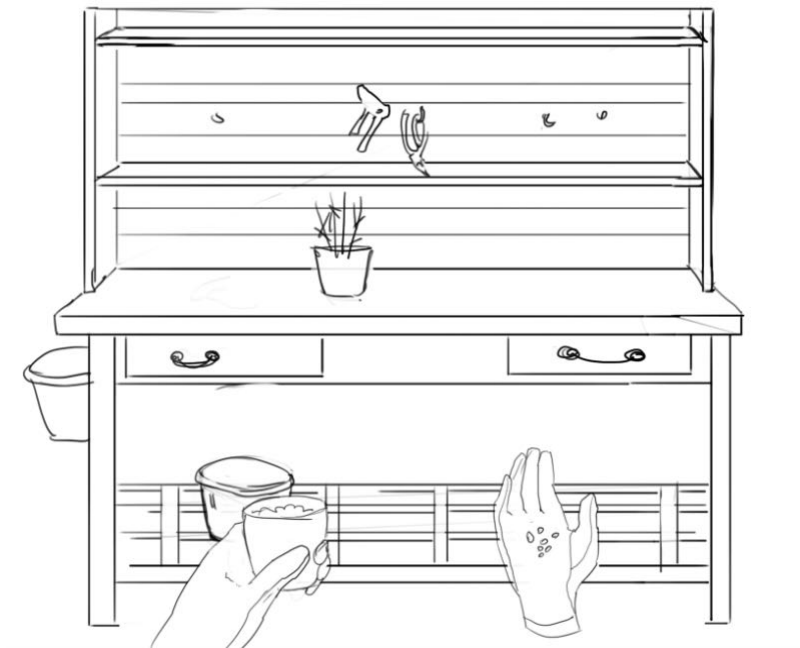
The garden is characterized by grass, plants, lots of flowers in order to make the scene pleasing to the eye and colorful. A fence that limits the perimeter of the player's area,

---

<sup>2</sup> <https://learn.canvas.net/courses/3/pages/level-4-dot-4-feedback-loops>

a fountain or water tap for actions and the player's "working area" characterised by a table or a stand with tools. The area around the player which is out of bounds (where they can't go or have interaction with objects) is characterized by plants, fence, grass to make it resemble a quiet grass lawn of a house or other blocking objects. Some of the patients will be able to move around (such as sport injury ones) but they will not be able to go far from the workstation that is the focal point of the whole game. The player in this game grows flowers and plants, harvests them from seeds and follows them through their growth. They will therefore check them, water them, fertilize, trim them and move them in the area to the best place for them according to how much light they need.

A working top can be found both in the garden and in the green house. Ideally the working station gives the feel of a workshop and the activities that the players can do should be straight forward. There is going to be a table-top made of wood or plastic where the player is going to perform most of the actions. This is also the place where the players will find the tools. Tools must be placed around this space and all must have easy access. They could be displayed as hanging in front of the players, or in a basket or pouch.

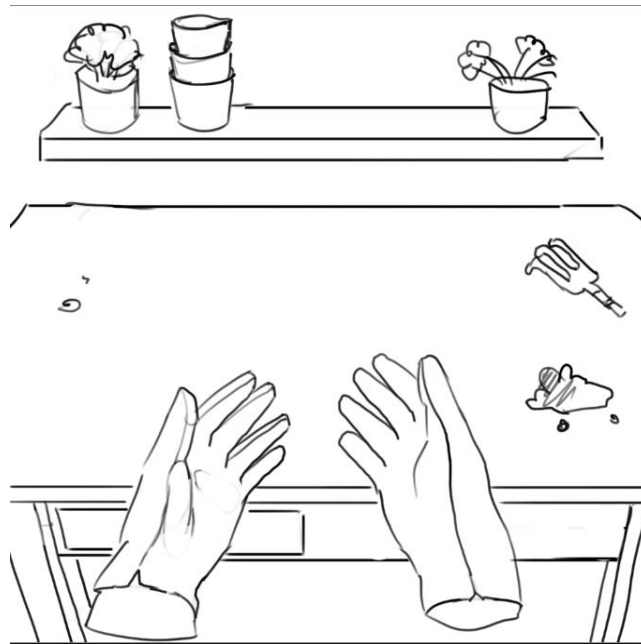


Concept: Workbench









Concept: Player gloves to match environment

### **Modality**

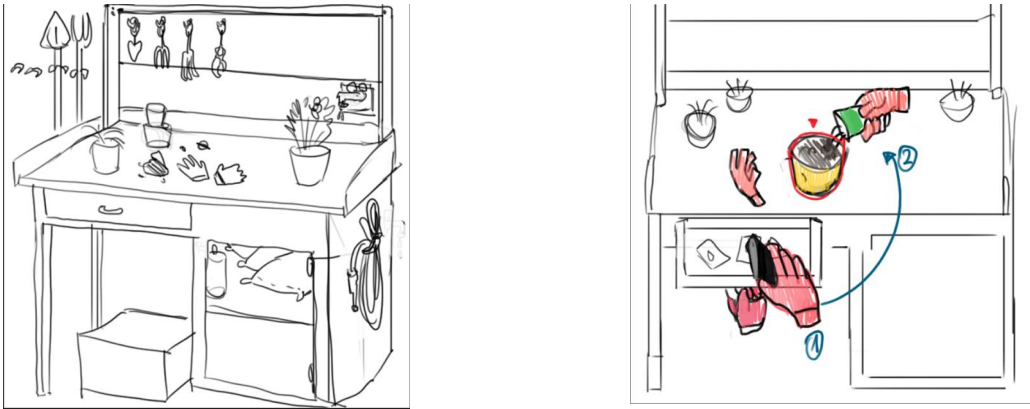
This game can be played sitting down or standing up. There always needs to be enough free space for the player's action, and for this reason the working station should be big enough. Once the players have finished their action (like planting seeds) they can decide to move it on the shelves (display teleport the plant option for stroke patients that cannot move too much) or can have some free shelves in front of the work top.

There could be a teleport function in case there are some areas that are not accessible to patients with limited mobility such as a spot in the garden where they can plant directly on the soil or the place where they access the water tap, a storage for tools etc. The movement settings can be customized per patient by the doctor according to the case. For example, a sports patient can walk around and pick up objects on their own, needing little assistance in the game by the doctor (who can focus on how the recovery movement is executed). This is not the case for a stroke patient that will not be able to move parts of their body and will require assistance for specific actions.

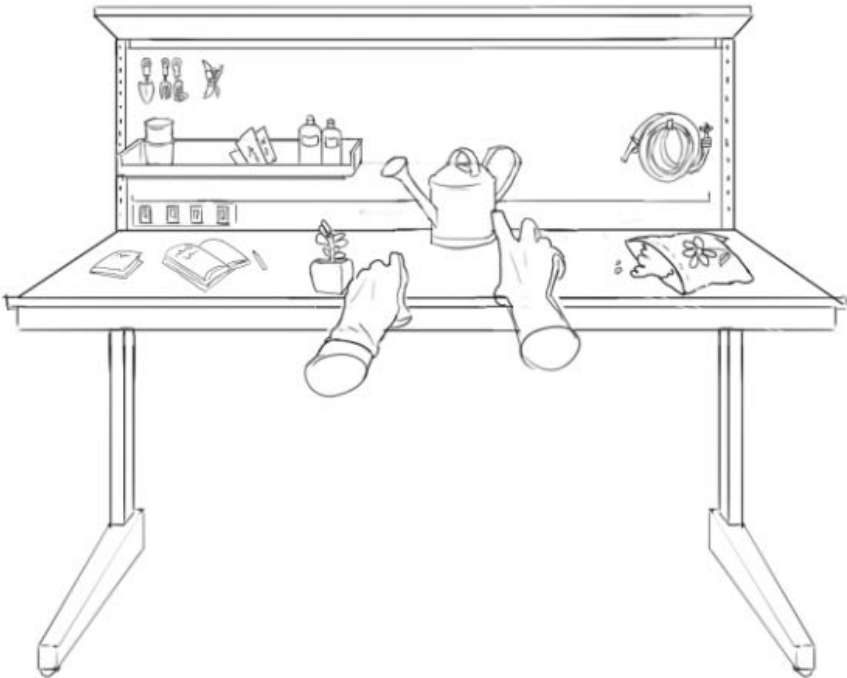
Players will require movements coming from the upper part of the body. Fairly simple actions such as moving the torso, using their hands, shoulders, arms, moving the neck, mouth to read out loud from instruction's books.

For certain patients (such as dystonia's patients) gardening actions will be further simplified allowing the player to make one thing at a time such as picking up a flower, or an object, pressing a button to open the garden hose, picking up a bottle of fertilizer, putting soil that is in their hand in the pot.

Players will focus on one movement specifically for one part of the body for example opening the garden hose they will press a button, they will have dirt in their hands and they will have to put it in the pot, put the seeds in the pot, pick up the gardening hand shovel.



Concept: Planting seeds in container



Concept: Workbench V2. All objects are more accessible to the player.

It must be considered that some of the patients will have extremely limited mobility therefore it would be ideal to have someone assisting them during the game in order

to avoid frustration and make them enjoy the therapy. This assistant can be included in the game as “spectator” or as a figure next to the patient.

## **Assistance**

The assistant may help picking up or storing objects for the player to make the experience doable. The assistant will also monitor the progress of the patients checking on the correctness of their movements. (In this case. the assistant may be their carer or a virtual teacher (bot) that the patient must try to imitate.)

## **Social**

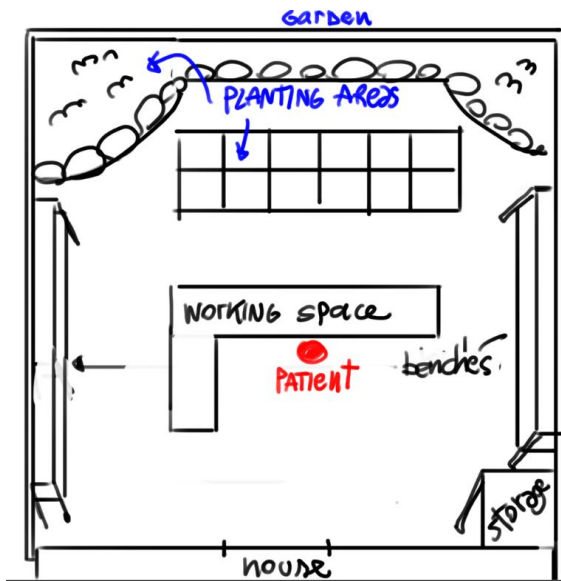
N/A

## **Supported Activities**

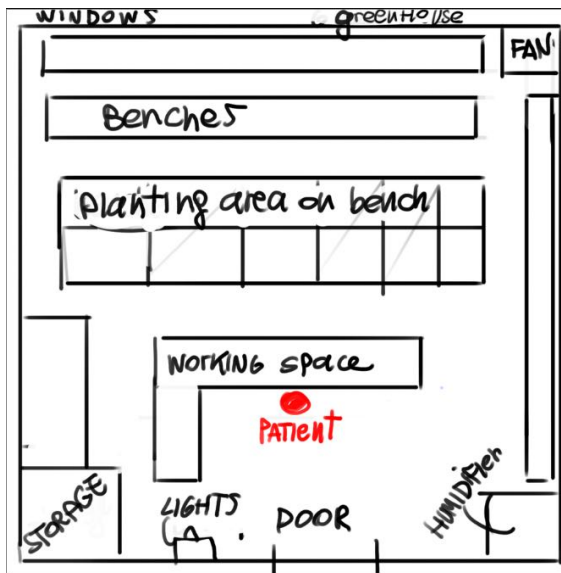
- Gardening tools include gardening hand trowel, digging shovel, bow rake, digging fork, garden hose, pruning shears, irrigation tools, fertilizers and compost bags.
- Greenhouse: glass windows, benches and shelves where the pots with plants are placed, water tap or fountain. Tools: humidifier or temperature tools, hand trowel, pruning shears, irrigation tools, fertilizers and compost bags, fans.



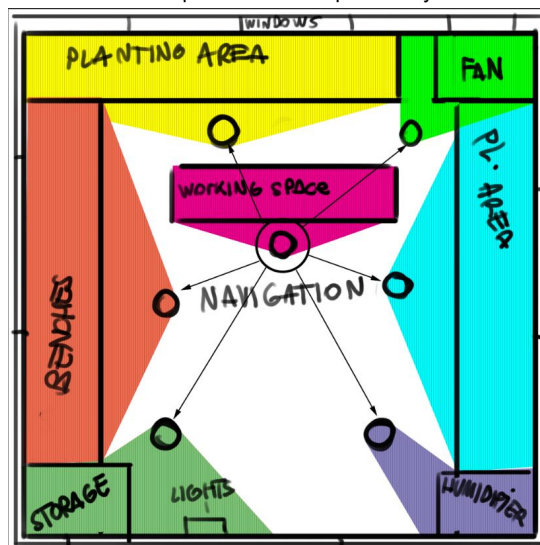
Ref: "Potioneer" (showing planting area)



Concept: Garden planimetry







Concept: Greenhouse planimetry











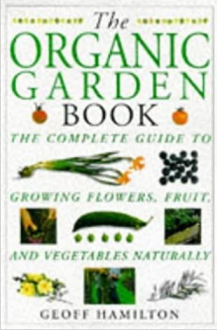

Concept: Greenhouse with improved navigation

## Interactive objects




Tool	Actions	How to interact with the object
<p>Hand trowel</p> 	<p>Pick it up Carry it Dig Put it down Take the dirt from the bag apply dirt in the pot Cover holes with dirt Smoot dirt surface</p>	<p>1 hand - Sitting down</p>
<p>Digging shovel(garden)</p> 	<p>Pick it up Dig a hole Place it away</p>	<p>2 hands -Standing up</p>
<p>Bow rake(garden)</p> 	<p>Pick it up Collect leaves and debris Loosen the soil Break up the soil Spread material(fertilizer) Collect grass Put it down</p>	<p>1 hand for small one - sitting down 2 hands for big one - standing up</p>
<p>Digging fork (garden)</p> 	<p>Pick it up Loosen the soil Lifting soil Put it down</p>	<p>1 hand for small one - sitting down 2 hands for big one - standing up</p>
<p>Garden hose</p>	<p>Pick it up Move it and aim Open the water tap Close the water tap</p>	<p>2 hands - used combined with the water tap</p>

	<p>Put it down</p>	
<p>Pruning shears</p> 	<p>Pick it up Aim Cut/Shape/Trim Put it down</p>	<p>1 hand, sitting down or standing up</p>
<p>Fertilizers</p> 	<p>Pick it up Read details Open the bottle Pour Put it down</p>	<p>2 hands, required to open it</p>
<p>Compost bag</p> 	<p>Pick it up Read details Open the package Pour Put it down</p>	<p>2 hands - needs to be opened and requires hands or garden shovel tool for filling the pots</p>

<p>Pots</p> 	<p>Pick up Fill Move</p>	<p>Empty 1 hand Full 2 hands Requires hands or shovel tool to be filled /finger</p>
<p>Lightening system(greenhouse )</p> 	<p>Switch on the light Switch off the light</p>	<p>1 hand/fingers</p>
<p>Humidifier</p>  	<p>Switch it on Switch it off set parameters</p>	<p>1 hand/fingers</p>
<p>Watering can</p> 	<p>Pick it up Fill it up Water the plants Put it down</p>	<p>2 hands, needs to be filled with water, use combined with the water tap</p>
<p>Flowers</p> 	<p>Change pot Pick flower up Cut flower Harvest seeds from flower</p>	<p>hands/fingers</p>
<p>Dirt</p>	<p>Move Smooth</p>	<p>2 hands/Tool</p>

	<p>loosen (garden) Lift it up</p>	
<p>Gloves</p> 	<p>Wear them when the games begin</p>	<p>1 hand</p>
<p>Gardening Books with instructions</p> 	<p>Pick the book up Change page with the hands Read out loud Put it down</p>	<p>2 hands</p>
<p>Notebook with pen</p> 	<p>Pick it up Write / draw Put it down</p>	<p>2 hands</p>
<p>Seeds package</p>	<p>Pick it up Open Plant the seeds Throw away</p>	<p>2 hands</p>



		
<p>Water tap</p> 	<p>Open close</p>	<p>1 hand</p>
<p>Fan</p> 	<p>Switch on Regulate Switch off</p>	<p>1 hand</p>

### Non interactive objects

Anything beyond the focus of this environment is placed there to complement the environment. There is a grey-area which we could use to allow different sets of interactive objects according to the player's abilities. For example, in sport injuries patients the players can look around, move in a moderate space and cannot interact with the background objects or change room. Stroke patients can move their head to see around but actions are restricted and cannot interact with anything that does not belong to the working space. On the other hand, dystonia patients will have full focus on the action that is required and will not be able to interact with anything else. In addition, elements outside of the action will be partially removed or blurred to help players focus on a single or a small subset of items.

### Environment size

Both garden and greenhouse environments should give the impression that the environment is realistically big and not make the player feel claustrophobic.



Reference picture from "garden flipper" showing the garden area.

Players with different abilities may have different movement restrictions.

### **Game references**

1. [Garden flipper](#)
2. [Potioneer](#)

## 2. SUPERMARKET ENVIRONMENT

Controller Requirements / Activity	Cart stocking	Weighing	Petanque	Darts (Sword fish)	Bowling (Detergents)	Hockey (Brooms tick)	Painting/ Writing	Basketball Hoops	Checkout
Finger Tracking	Required	Required		Required			Required		Required
Palm Open/Close	Required	Required	Required	Required	Required	Required	Required	Required	Required
Wrist Movement	Required	Required			Required		Required		Required
Elbow Movement	Required	Required	Required	Required	Required	Required	Required	Required	Required
Wrist Resistance	Nice to have	Nice to have	Nice to have		Nice to have				Nice to have
Elbow Resistance	Nice to have	Nice to have			Nice to have	Nice to have			Nice to have

Required
  Nice to have

Table: Controller Requirement Matrix



Reference: Supermarket VR. A good example of minigames a player could find within the game.



Reference: Shelfzone VR. Supermarket view from the perspective of the player and example of teleport in the different supermarket's departments.



Reference: Supermarket VR. In game view.

The supermarket location was selected because it is familiar to all. As mentioned earlier, being in a friendly and familiar environment plays an important role in the patient's psychology helping them adapt to VR. The location can be beneficial to the patients because it represents a common action, they might not be able to do anymore or not without external help and it offers them a glance of “normality”. Another reason why this environment is to be considered is that it offers a variety of different motions that can be beneficial to sport, stroke and dystonia patients. The supermarket leaves a certain level of freedom to both patients and developers. Developers can set a various number of activities involving the object in the supermarket beside the common action of buying food.

The difference between this and the garden/greenhouse environment described earlier can be found in the type of activity. Not everybody can be fond of gardening and passionate about it, but we certainly know that a supermarket is an extremely familiar place where everybody visited many times and is strictly linked with common day life activities.

The activity of buying and searching for goods does not only lead the player to perform a daily activity but also leads him to a sense of gratification. Elderly people for example, who are not so familiar with games and technology, might find in this a nice and calming activity that helps them familiarize with VR and have some good exercise.

The player is afforded a large amount of creative freedom in how they shop. They are also free to mess around with the various objects in their reach, such as throwing things in trash cans.

### **Audio**

- Background sounds [supermarket sounds](#), [supermarket sounds2](#),
- Sound FX: [machine blip](#), [cash register sound](#)
- Character's sounds: the character is cheerful and expresses emotions while doing activities, encouraging the players to keep up what they are doing.
- Other sounds present in the game would be: the sound of the various products hitting each other when being put in the shopping cart, the player might occasionally drop something on the floor, put it back on the shelf or throw it. The sound on the footsteps of the characters, the sounds of all the machines present in the environment, people's chatting, shopping cart bumping into each other...

These sounds are important because they help the player focus on the game and they give him an experience close to the real world. In this game there could also be sounds that do not exist in real life, for example a sound like: [arcade sound example](#), for when the player does actions that are part of the minigames. (throwing goods in the shopping cart like a basketball player or playing bowling with a stack of cans.) Players would be in part entertained by the sounds and stimulated to do more or repeat the action.

The aim is to achieve a good level of realism for those who want a calmer environment and at the same time make it more enjoyable to the people who want to play. Sounds would therefore change according to the type of gameplay the player has.



## Visual



Reference: Real life photographs of common big chain supermarkets.

What is this going to look like? The idea is to recreate an environment as close as possible to reality with colorful visuals where the players feel “at home” and safe, something that they immediately recognize and makes them relax. In this environment the players will immediately know what the main purpose of the game is, however, they will need explanation to understand the level of freedom they have.

This environment will be characterized by long isles that are filled with goods from various brands (either real or fictional) that the players will interact with in a different number of ways. Players will be able to pick up the goods and put them in the shopping cart, consult them (picking up the product and reading the tags), etc. The environment we are trying to reproduce will be divided into sections like a real supermarket:

- Vegetables and fruit section
- Raw meat section
- Fish section
- Cured meat section
- Bakery
- Detergents
- Beauty product section
- Shampoos section
- Household items
- Fry shop
- Cooking section
- Candy section
- Book section

As expected, this environment is much bigger than the greenhouse. Each section will have its own goods like a real supermarket and the player will have to search in the proper one for certain items.

## Modality

Due to its physical dimensions it might be difficult for certain players to move between different sections. A teleport system (see reference below) can be used; here players draw a path to a ground location and instantly teleport to it.

The use of a controller analog stick can also be used to allow for finer movement without requiring the player to physically move out of his current position.



Reference: From "Surviv3" the player is pointing where he wants to go using the controller.

As for checking the shelves the players would need the use of the head to move around, or an eye tracking system whenever the players have huge difficulties in moving. Eye tracking could be particularly helpful with stroke and dystonia patients.

In this game patients might need hands and fingers to interact with the controller plus arm movement and head to move around. This game can be played with one hand but given the possibility it can be used to exercise both hands and can be played both in a sitting position or standing up.

The doctor can customize activities according to the patients in order to match his physical capabilities (for example a stroke patient might be asked to pick up a certain good from a shelf). Dystonia patients will be asked to do one simple movement at a time, the focus will be on the action they have to do, and the background elements could be few or blurred in order to facilitate them.

The session begins by choosing a shopping cart (which defined the way you will move in the game: sitting down or standing up):



Reference: Shopping cart A



Reference: Shopping cart B



Reference: Shopping Cart C for wheelchair or people who play sitting down.

How are you going to play the game?

**Sitting/ Wheelchair**

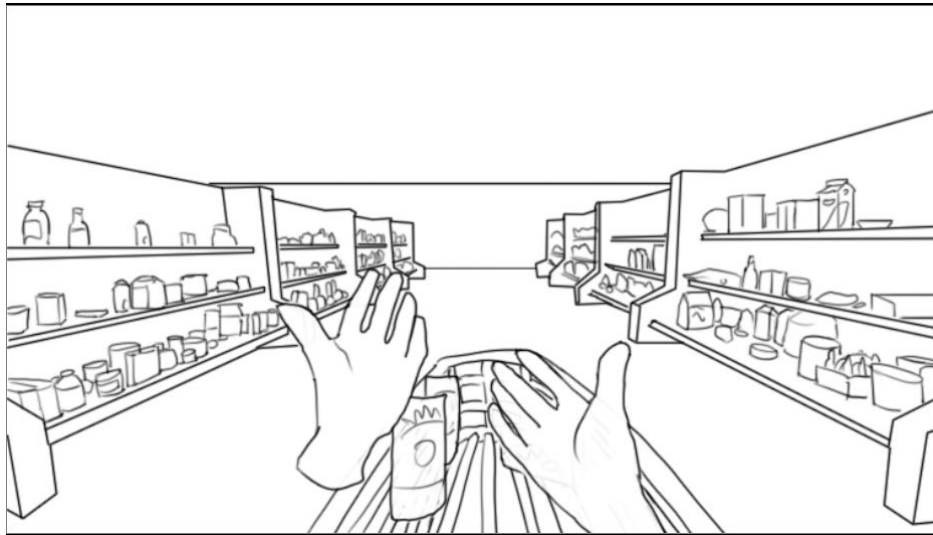
Standing up

The image shows a survey question with two options. The first option, 'Sitting/ Wheelchair', is highlighted with a green box, indicating it is the selected response. The second option, 'Standing up', is in a grey box. The background is a blurred image of shopping carts.

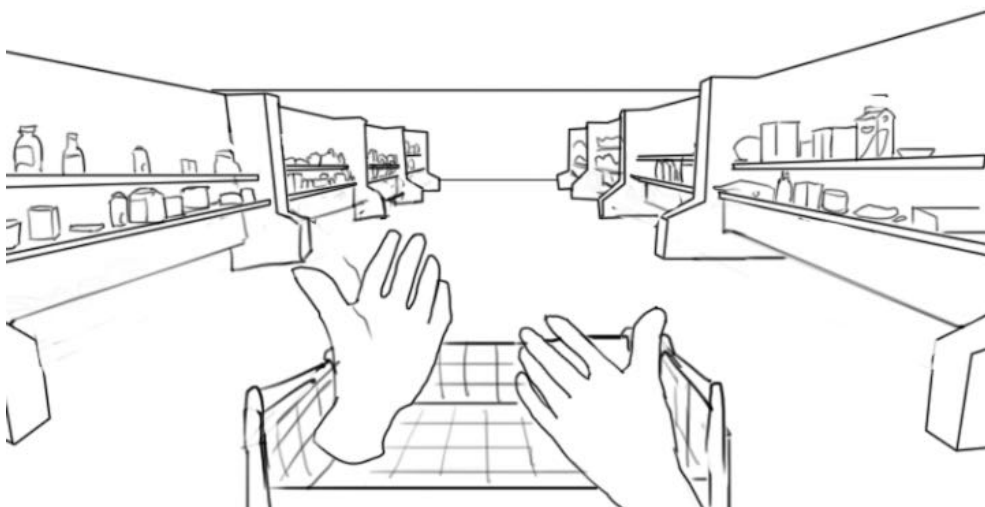
Concept: When the game begins players choose how they will be playing.



According to the player's choice, the quantity of items the cart will accommodate will also change. There is not going to be a limit for stacking, but items will eventually fall off the cart if it is too full.



Concept: First person camera, VR point of view.



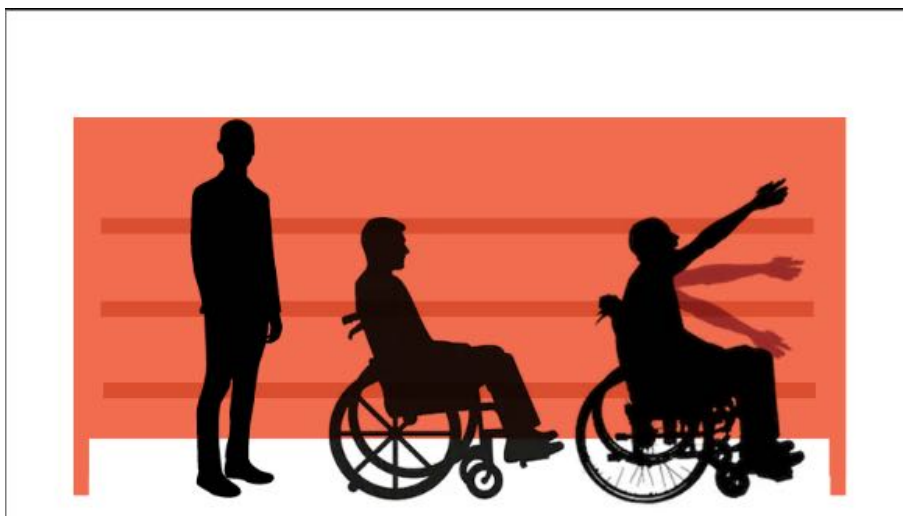
Concept: First person camera of a person sitting in the wheelchair.



Concept: Top down view of a player in a wheelchair.

For people who are forced to play the game in sitting position, there is going to be a shopping cart that attaches to the virtual wheelchair so that they can focus on the movements. In order to pick up items they just need to turn their heads or torso to face the shelves.

Players will walk the isles where they will have shelves filled with different goods. The isles are going to be wide to allow space for wheelchairs while granting them enough space to turn around. Goods will be limited; a limited number of items will be present on the shelves with some of them even having run out of goods entirely. Goods will not spawn endlessly. Every session is going to be different; goods and their availability can change. Players will be able to reach shelves from every position. The shelves will be placed at medium height so that it is reachable by both sitting and standing players. The shelves will not be very deep so everything a player can grab is also automatically within reach.



Concept: Height of the shelves. (1)



Concept: questioning the reach of the player sitting in the wheelchair. Shelves placed in a range where everything is easily accessible. (2)

In order to facilitate this movement for people whose movements are limited, there could be an extendable arm that increases the movement they do, reaching way more than the actual movement.



Concept: extendable arm.



Concept: Picking up items without an extendable arm.

In order to get goods players will do actions specific for the department they are in, such as weighting the vegetables, digitizing the number and printing the tag.

The players will be able to interact with all the goods (cans, food, products, boxes etc.) that are found on the shelves or on the ground. Players will not be able to interact with the shelves, manipulate them in any way, they will not be able to change anything that cannot be bought and is part of the supermarket layout and changes the environment. Most of the goods will be modeled with different textures and sizes to give the players the impression they are choosing between brands.

### **Assistance**

In addition, doctors could play a huge role in helping the patients: they could assist their movements and help them doing the action the patients are having difficulties with. They would also be able to monitor the improvement and see what actions the patient has more difficulties with and change the therapy according to what they feel it is necessary. Being an environment where the players are free to do anything can lead the doctor to challenge them to achieve goals through unexpected and possibly lighthearted goals (for example, for wrist injuries an activity that requires precision and movements would be stacking up a product that has a particular shape). Players will have total freedom of what goods to buy.


### **Social**

In the supermarket the player will not be alone, there will be other NPCs (non-player-characters) doing shopping because the environment of an empty supermarket might be unsettling and scary. The player will be able to interact with the NPC like the cashier and the butcher choosing what goods to buy. Also, upon bumping into other characters and their shopping carts the NPC will react with sentences such as “hey”, “be careful” etc. like in real life. When doing mini games, the np NPCs will ignore the player.  
Multiplayer option N/A

### **Supported activities**

In this environment there can be all kinds of different activities:

Vegetables and fruit section	Select fruits Weight fruits Label fruits Place fruits in the shopping cart Play petanque game with fruits
Raw meat section	Select meat Place meat in the shopping cart
Fish section	Select fish Take fish Place fish in the cart Play darts with small swordfishes Feed the fishes in the tank
Cured meat section	Select meat Place meat in the shopping cart Cut the meat yourself (2 hands)
Detergents	Buy detergents Make bubble with detergents Do piles with detergents (2 hands/ 1 hand) Play bowling with detergents
Beauty product section	Buy product Try product in the mirror
Shampoos section	Buy shampoos
Household items	Buy items Play hockey with brooms (2 hands)
Fry shop	Buy goods
Cooking section	Buy goods
Bakery	Choose goods Buy goods Make bread (2 hands) Follow recipes
Book section	Read book Buy book Paint

	Write
Candy section	Buy products
 <p>Shopping cart activities A&amp;B</p>	<p>Move the cart</p> <p>Place items in the cart</p> <p>Throw items in the cart from a distance</p> <p>Leave the cart</p>

### **Modes**

The players will begin their adventure in different way so that the game does not become monotonous.

- Unlimited cash
- Limited cash
- Follow shopping list
- Buy only discounted items
- Buy not discounted items
- Time challenge, shopping within given time
- Time challenge with any of the restrictions above
- Freedom mode

### **Game References**

1. [Pizza maker](#)
2. [Shooty fruity](#)
3. [Mister Mart](#)
4. [Job Simulator](#)
5. [Supermarket VR](#)
6. [Shelfzone VR](#)

### 3. CRAFTS WORKSHOP ENVIRONMENT

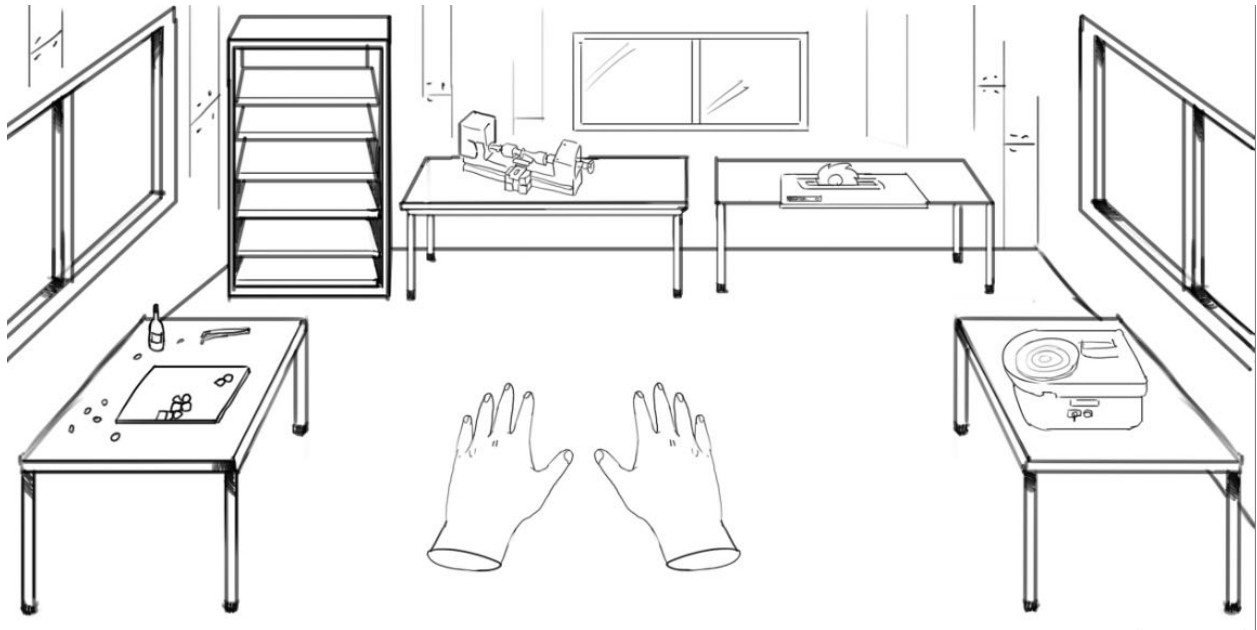
Enjoy a relaxing afternoon creating art pieces and hand crafts.

Controller Requirements / Activity	Woodworking	Pottery	Mosaic	Painting	Lathe	Morse Puzzle
Finger Tracking	Nice to have	Required	Nice to have	Nice to have		Required
Palm Open/Close	Required		Required	Required	Required	
Wrist Movement	Required		Required	Required	Required	Required
Elbow Movement	Required	Required	Required	Required	Required	
Wrist Resistance					Required	
Elbow Resistance	Nice to have					

 Required
  Nice to have

Table: Controller Requirement Matrix

## 17.5. Visual



Concept: player view inside the backyard shed.

- The Player finds himself inside his wooden backyard shed where various types of activities can be engaged separately. Those activities include (but not limited to) woodworking, pottery, mosaic, painting.
- The environment is the size of a normal to medium sized room where the player can clearly see different areas/worktops/machines where a craft is practiced.
- The workshop acts as a hub - by pointing to the selected area/worktop/machine the player is fixed again in the sub environment
- Once in a sub environment the gameplay modality changes according to the actions required by that section of the game.
- A central section may be required to illustrate the task at hand, such as a whiteboard with a checklist of things to be done (replacement for using on-HUD UI).





Concept: example of task board in the shed.

### **17.6. Modality**

---

- All activities can be done sitting or standing

### **17.7. Assistance**

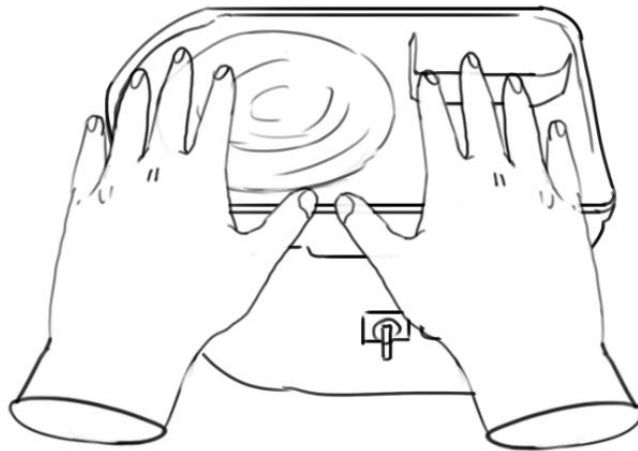
---

- The environment allows the presence of a 2nd player that can assist with the actions as if the first player were doing them.

### **17.8. Social**

---

- The end of each activity results in one or more artifacts which can be stored as discrete virtual objects. For example, the Player has created a mosaic painting of a simple portrait or a simple table.
- These objects can be put up for display, traded/gifted with other players, rated by other players (system of 'Likes'). (A new kind of leaderboard where others vote for your content)



Concept: example of pottery activity.

- **Woodworking:** tasks include creating simple objects like tables, chairs, stools, objects of simple geometry. This would mean sliding sheets of pre-marked wood across saws, placing them on pre-marked points to glue them together etc.... Considerably basic actions are required to perform the operation, assembly done in Ikea-like fashion.
- **Pottery:** use one hand or both to mold a piece of clay into a simplified pot of different shapes. Finally place in the oven/furnace to bake.
- **Mosaic:** given a shape or a description, cut a few pieces of shards and glue them on a canvas to produce the final work of art.
- **Painting:** dip your brush to change color and paint a picture on a virtual 3D canvas.
- **Lathe:** transform a block of wood into a usable piece of art by chipping away with a chisel (produce a goblet, stand, candlestick holder, handles, knobs).
- **Morse code:** potentially fun to have Morse-code build into the equipment for some puzzles. Easy to understand once you get used to the alphabet.

## 4. UNDERWATER ENVIRONMENT

Sit back and unwind as you explore this aquatic world full of wonder.

Controller Requirements / Activity	Sample Collection	Fish Capture	Waypoints	Treasure Hunting
Finger Tracking	Required	Nice to have	Nice to have	Nice to have
Palm Open/Close	Required	Required	Required	Required
Wrist Movement	Required	Required	Required	Required
Elbow Movement				
Wrist Resistance	Nice to have	Nice to have	Nice to have	Nice to have
Elbow Resistance				



Table: Controller Requirement Matrix

### 17.10. Visual

- The Player finds himself inside the cockpit of his small, trusty mechanical submarine that is free to explore the surrounding ocean.
- Most of the cockpit is made up of glass giving the player a good view of the surrounding ocean (extruded cockpit) and though small the sub is not that small as to induce claustrophobia.
- The player is confined to the sitting position in the sub and is not allowed to go anywhere else.
- Navigation of the environment is done by moving the submarine with a joystick (grabbing a virtual joystick) or pressing one of the virtual buttons on the dash.
- (UI) Elements of the activities supported by this are drawn on the sub's viewing sphere: such as a map or objective, counters, etc....

- A central section may be required to illustrate the task at hand.



Concept: Navigation via stick for Forward/Backward/Sideways. Additional Up/Down and rotational buttons illustrated.

---

### 17.11. Modality

- All activities involving the sub are sitting down.

---

### 17.12. Assistance

- The environment allows the presence of a 2nd player that can assist with the actions such as sub navigation. Alternatively, also taking over total control of the game where necessary.

---

### 17.13. Social

- N/A

---

### 17.14. Supported activities

- **Sample collection:** extend the sub's robotic arm to collect samples of sand, coral and other marine life.
- **Fish Capture:** swing your robotic arm equipped with a catch net to capture live aquatic fauna.
- **Waypoint Traveling:** navigate your sub around a set of waypoints for a scenic tour.
- **Hunting:** aim your net (harpoon? stun gun) to acquire the target marked by your HUD.

- **Codex:** complete this entry book of all the flora and fauna you have discovered while playing.
- **Treasure Hunting:** use sonar to find forgotten pieces of history.

## 5. SPACE ENGINEER ENVIRONMENT

*There is always something to fix on this floating heap of metal!*

Controller Requirements / Activity	Puzzles	Maintenance	Item Recovery
Finger Tracking			
Palm Open/Close			
Wrist Movement			
Elbow Movement			
Wrist Resistance			
Elbow Resistance			



Table: Controller Requirement Matrix

### 17.15. Visual

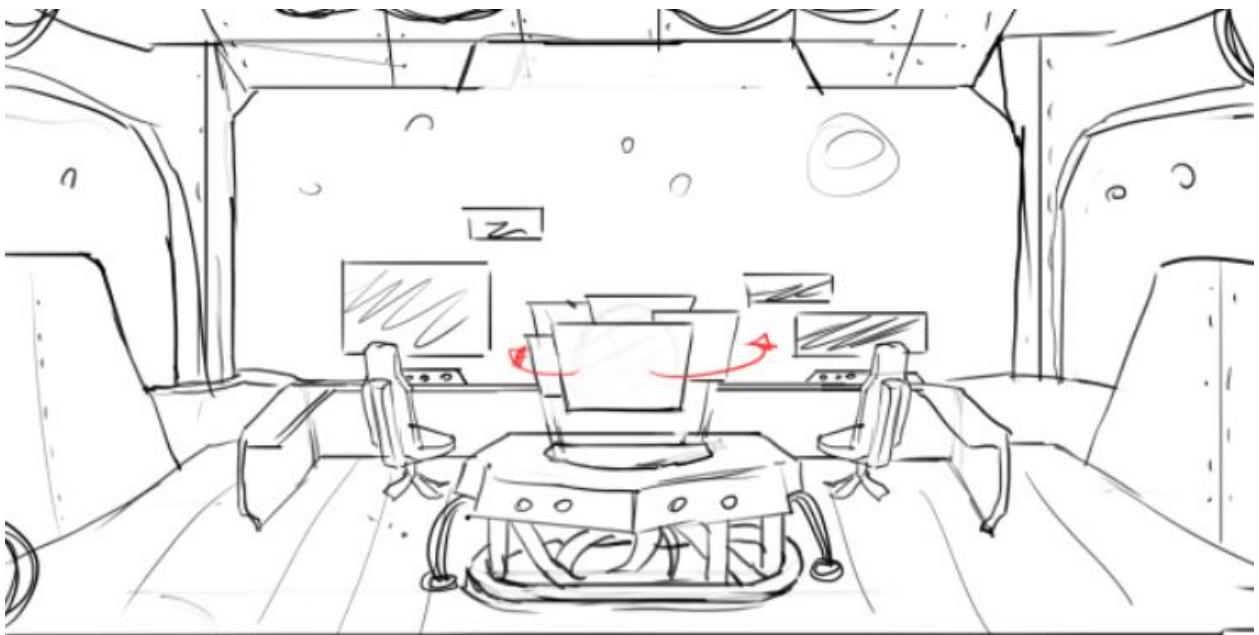
- The Player finds himself inside the most fault-prone space station known to mankind that is currently also orbiting Earth. The surroundings are that of a typical space station with large panes of glass displaying outer space. Feel should pivot to light-hearted/comical rather than realistic.
- A central monitor at the center/hub (or talking AI) of the station displays areas of the station that currently require maintenance (some inside the station, some outside). This information might need to be relayed via a watch-like device.
- The Player can look around and easily identify the location or where to go by in-environment highlights such as a luminous floor.
- The Hub connects to a series of exits or areas inside the ship. For example: Service Shaft A, B, Left/Right wing exit, etc....
- The Player can be required to fix something on the inside or on the outside where there is zero gravity.

- On leaving the station the Player is automatically put inside a suit and on looking down a virtual joystick for navigation can be found.

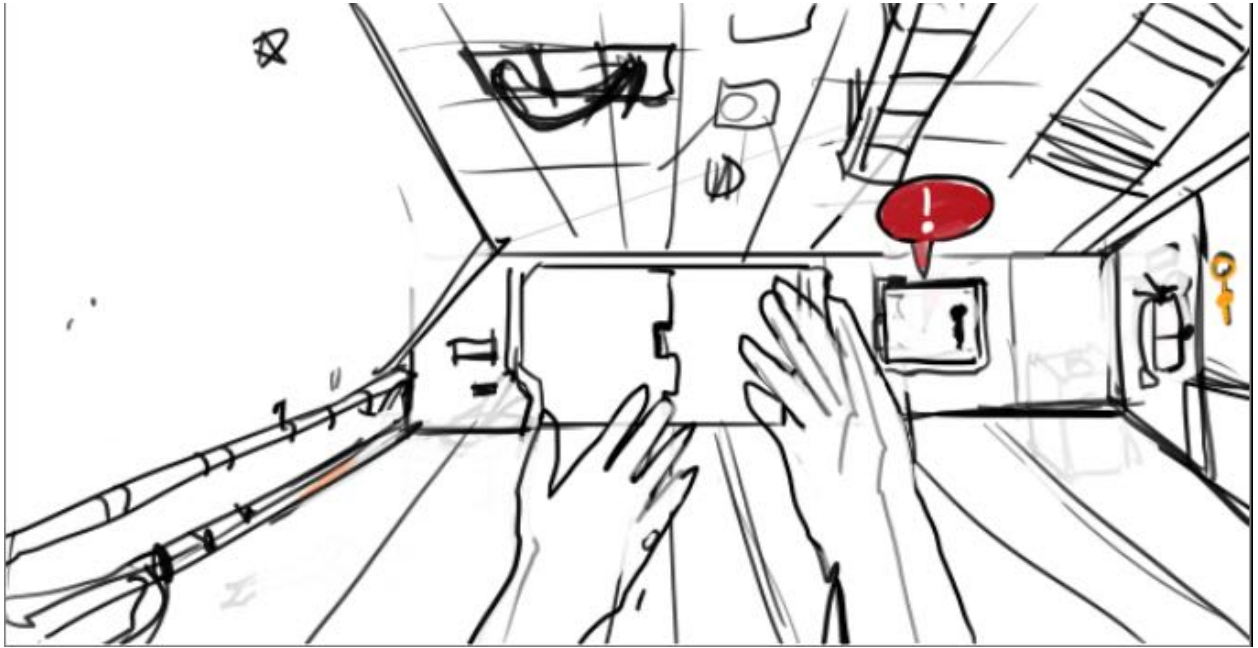
### 17.16. Modality

---

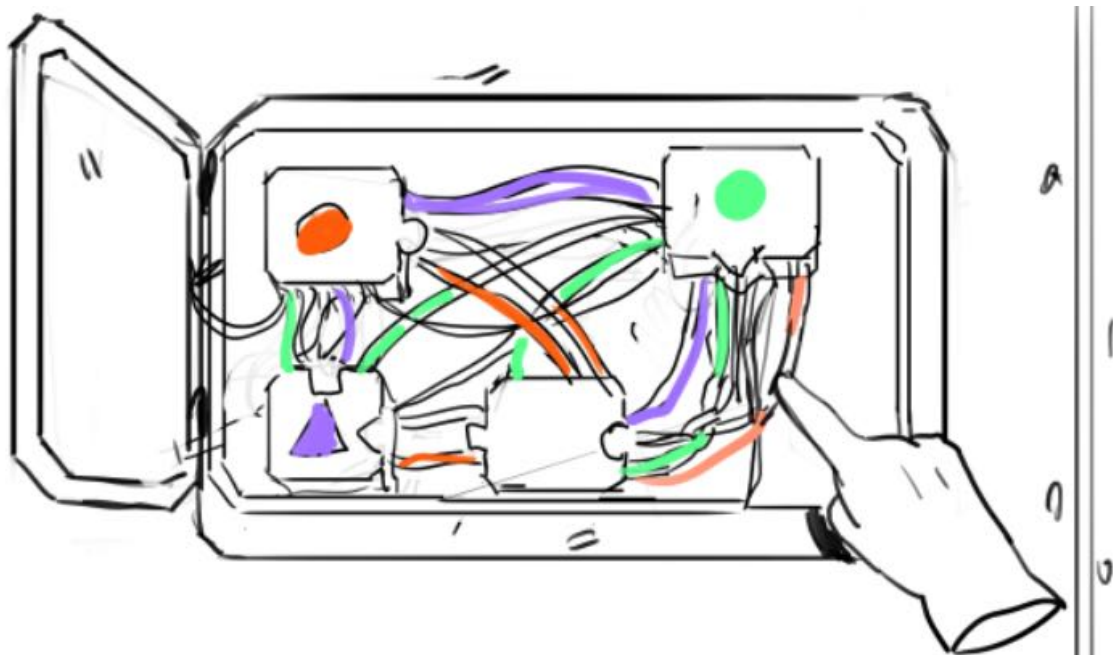
- All activities can be done sitting or standing.
- Movement inside the station is point-based; gravity is also present inside.
- Movement outside the station is done by grabbing a virtual joystick held chest-high and moving along one planar direction. The direction follows where the Player is currently looking.
- Fixing 'things' means going around the station and solving puzzles and/or beating the proverbial problem with a wrench (light-hearted approach).



Concept: first concept of the spaceship's main room.

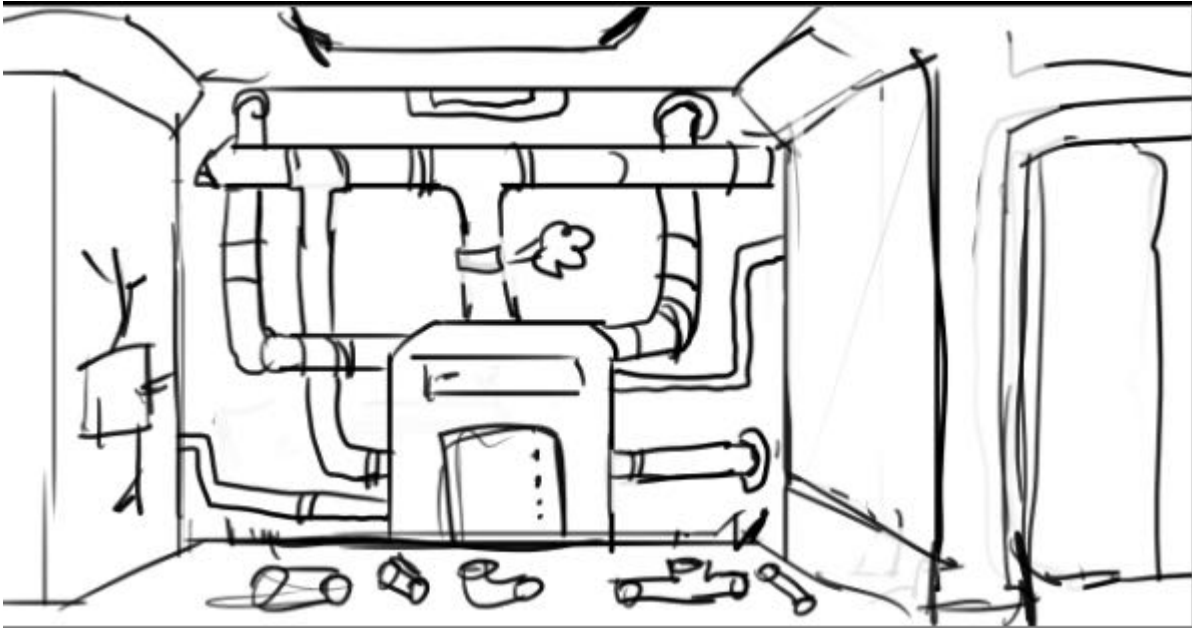


Concept: An example of a puzzle that can be found in the spaceship.

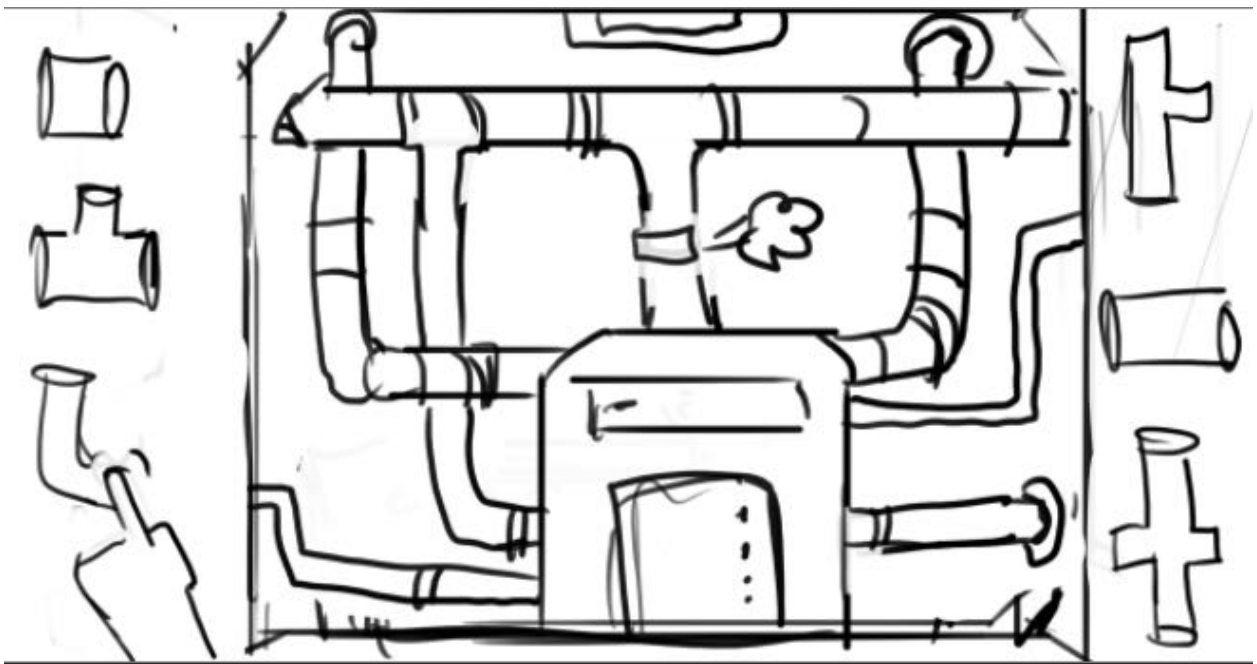


Concept: Example of how to solve the puzzle. Puzzles will be solved in real-time through the help of nearby objects





Concept: Another example of pipe-type puzzle.



Concept: How the puzzle is played and solved.

### 17.17. Assistance

---

- The environment allows the presence of a 2nd player that can override the control of the Player when necessary. This is especially necessary for the outside sections where navigation is an activity.

### 17.18. Social

---

- N/A

## 17.19. Supported Activities

---

- **Puzzle:** Restoring power by redirecting flow of current. Done by dragging or rotating flow (Pipe Dream<sup>3</sup>) of key pieces on a display board in front of the Player or by moving pieces of block circuits around (Sokoban<sup>4</sup>). Other minigames can be incorporated into this.
- **Hands-on Maintenance:** fixing power flow such as closing or turning valves, welding air leaks, patching hull. May involve grabbing pieces from a section of the station before leaving.
- **Item Recovery:** recover an item/rock/artifact from outside the station.

---

<sup>3</sup> [https://www.youtube.com/watch?v=Y\\_C-xzSKg-k](https://www.youtube.com/watch?v=Y_C-xzSKg-k)

<sup>4</sup> [https://youtu.be/dzlGmq\\_ef9U?t=36](https://youtu.be/dzlGmq_ef9U?t=36)

## 6. COOKING SHOW ENVIRONMENT

Having been called to fill in the shoes of the previous host, it is up to you to cook the most delicious dishes on the fly!

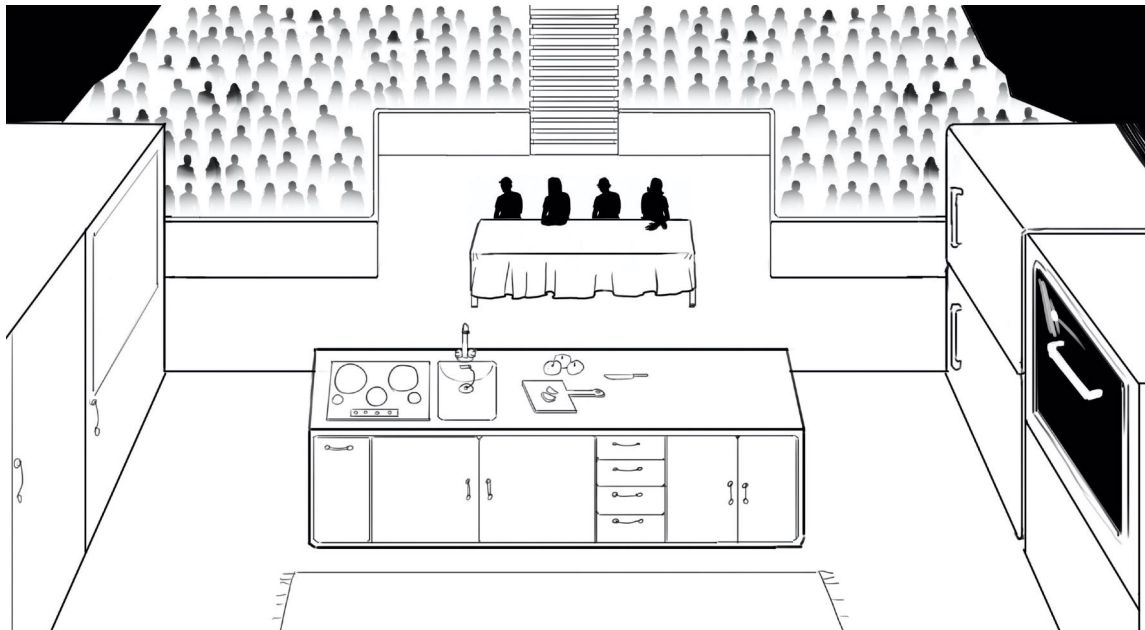
Controller Requirements / Activity	Appliances	Grilling	Chopping Veg.	Pouring	Stirring	Carrying
Finger Tracking	Nice to have					
Palm Open/Close	Required	Required	Required	Required	Required	Required
Wrist Movement	Required	Required	Required	Required		Required
Elbow Movement	Required	Required	Required	Required	Required	Required
Wrist Resistance	Nice to have					Nice to have
Elbow Resistance	Nice to have			Nice to have	Nice to have	Nice to have

Required
  Nice to have

Table: Controller Requirement Matrix

### 17.20. Visual

- This design can be split in 2 versions: one in which the Player is in a studio kitchen setup where facing him/her would be the audience/cameras and behind the real kitchen. The other version is where *the Player is in a real-world kitchen following recipes from a TV set*. This document will describe the first version.
- In front of the Player is a typical studio set composed of spectators, a couple of cameramen and 4 judges.
- Extending just in front of the Player and directly behind him is a fully-fledged kitchen: Fridge, oven, cupboards, etc....
- Just in front of the Player (and where most of the ingredient mixing and placing will take place) is the counter.



Concept: Cooking show environment example.

### 17.21. Modality

---

- All activities can be done sitting or standing.
- Movement is handled via a pointer across 3 or 4 preset locations to access points of interest such as taking objects out of a cupboard, putting a dish in the oven.
- Requires grabbing/manipulation of items in the world, at least one hand.

### 17.22. Assistance

---

- The environment allows the presence of a 2nd player that can override the control of the Player when necessary. (Co-op).
- Helpers can bake/cook along with the patient.
- Tweakable difficulty such as finding the items already on the counter.

### 17.23. Social

---

- Album-like structure that logs pictures of the original 3D model of the artifact produced, including rating.

## 17.24. Supported Activities

---

- **Cooking:** The Player receives a cooking 'job' at the start of the session. An announcer voice proceeds to dictate what you will be doing in the next steps. Direct instructions are also visible anytime when looking ahead to the oversized teleprompter. Ingredient quantities are mostly up to the Player having to judge accordingly (for example: you are making a spicy pizza and you can choose whatever spicy topping you want as long as it is spicy, but you must also not over-do it) The mood of the game is lighthearted - grading parameters are not very harsh. At the end, the produce is baked (if applicable) and the judges each grade the cooked food based on loose parameters such as recipe accuracy etc.
  
- **Actions:**
  - *Appliance usage:* Opening and closing of oven/microwave, turning timer dial
  - *Grilling:* flipping meat patty
  - *Cutting:* chopping vegetables such as carrots or coriander
  - *Opening/Closing:* containers housing liquids or powder
  - *Pouring:* pouring oil
  - *Stirring:* using a handle spoon to stir soup

## 7. CAR ENVIRONMENT

A car represents a common everyday tool that people constantly use to move from a place to another, to a person who had severe injuries it becomes something they can no longer take advantage of and they miss it deeply. For older patients being able to drive again even if just in VR can be an incentive to work hard and at the same time have fun.

The car/driving environment can be useful because it offers a large range of actions that involve fingers, arms, shoulders, head and torso. In the games involving the car the player will be required to play from a sitting position and move the upper part of the body, above all focusing on fingers, hands, wrist, elbow and neck.

Controller Requirements / Activity	Ignition	Open/ Close Door	Seatbelt	Hand brake	Radio	Signaling / Wipers	Gears
Finger Tracking	R				N	R	
Palm Open/Close	N	N	N	N		N	N
Wrist Movement	N		N			N	
Elbow Movement	N	N	N	N			N
Wrist Resistance	R	R					R
Elbow Resistance				N			



Table: Controller Requirement Matrix

### Audio

The background noise while driving the car changes according to the driving location. These are always slightly muted when windows are closed. You can also hear other sounds such as:

1. other cars while driving
2. Engine sound
3. Horns, ambulances or police cars if you are in a city environment

4. Braking sounds and various ambient sounds according to where the scene is taking place.

Sound References:

1. [City noises](#)
2. [traffic sounds](#)
3. [big city traffic sounds](#)
4. [car sound effects](#)

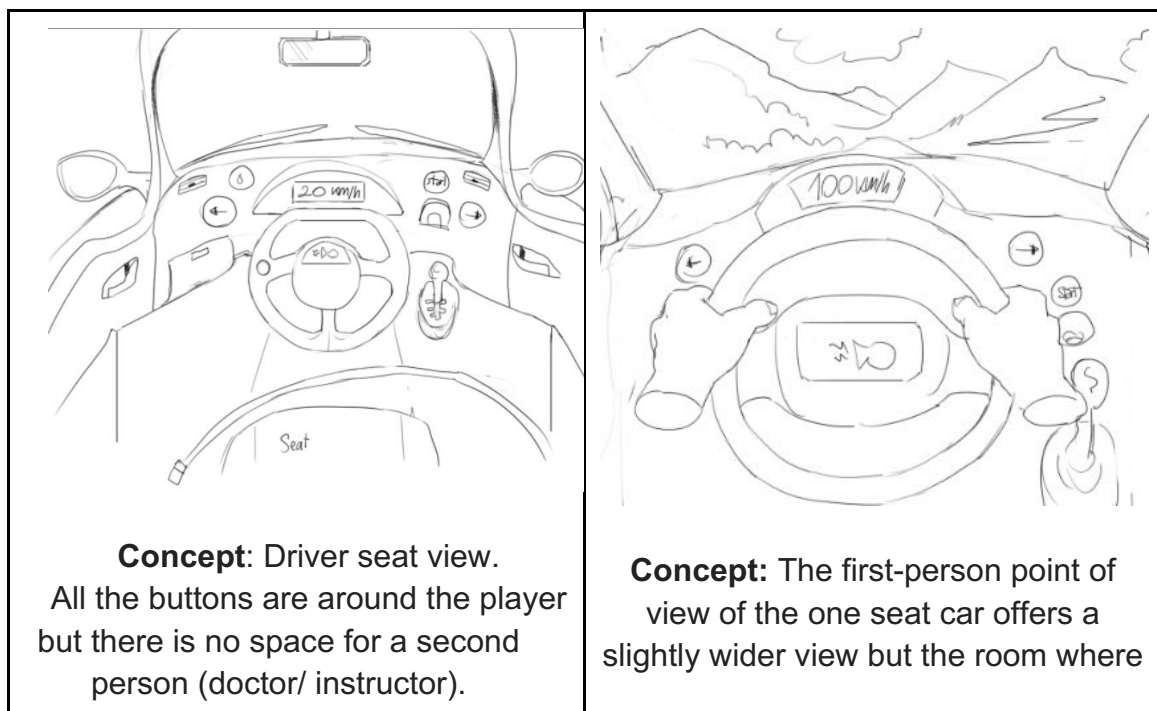
Sounds are important to give authenticity, and to accompany the players action in order to make him feel as close as possible to reality.

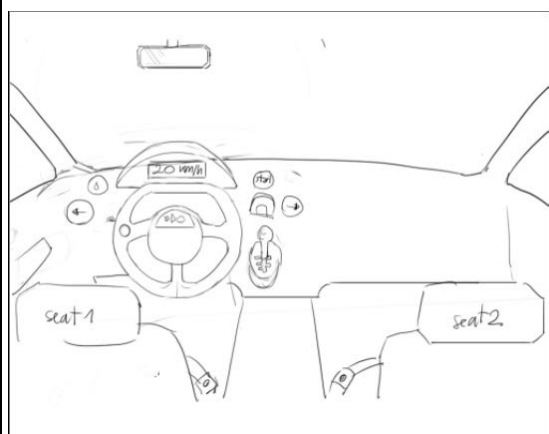
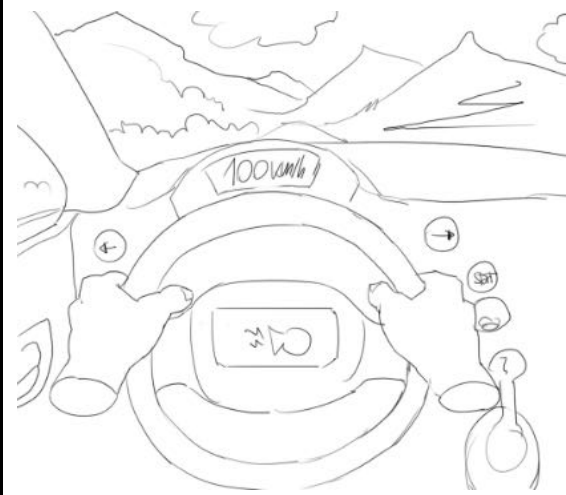
Possible scenarios where the patients can drive a car:

1. City landscape
2. Countryside
3. Seaside Town
4. Mountain landscape
5. Desert highway

## Visual

The player enters the car and sees a typical car interior with the following simplifications. We are striving to simplify the interior to make controls and gauges more accessible and visible in this game context. (See figure 1)



	<p>the player moves is less.</p> 
<p><b>Concept:</b> backseat view. All the buttons are like the one seat view, there is more space to add extra ones.</p>	<p><b>Concept:</b> First person point of view. The player is not facing the very center of the windshield but there is plenty of room to see outside.</p> <p>The single or two-driver point of view do not differ too much, but the latter allows for adding more controls while supporting the option to have a seat for the assistant.</p>

Concepts: Car representations

The player finds himself in a vehicle, big enough for the player to sit comfortable and not feel claustrophobic. The surroundings are composed by:

1. A centered steering wheels
2. The turning signals will be placed to the left and right of the steering wheel in the form of buttons or using the traditional switch type.
3. Other additional controls include fog lights, full beam headlights, hazard warning lights etc.)
4. The car engine is started through a big key that must be placed in its hole near the steering wheel and a "start" button.







Ref: Push to Start button

5. Big dashboard gauges so they can easily be read by visually impaired people.
6. The gear shifter is an optional accessory
7. The height of the seat can be adjusted according to the player necessities through the settings menu.
8. The player is confined to the seat and cannot go anywhere else.
9. The car will automatically move based on the instructions.

Objects the player can interact with:

 <p>Steering wheel &amp; horn</p>	<p>Can be used with 2 hands. Requires the movement of the elbow and shoulders.</p>
 <p>Car door handle</p>	<p>1 hand and use of fingers</p>
 <p>Car windows power</p>	<p>1 had and use of fingers</p>
 <p>Car door lock</p>	<p>1 hand and use of fingers</p>

	<p>1 hand, use of the fingers and movement of the wrist</p>
	<p>1 or 2 hands Head movement</p>
	<p>1 hand and elbow movement Head movement</p>
	<p>1 hand, wrist and elbow movement</p>



Turn signal & lights settings

Rear view mirror

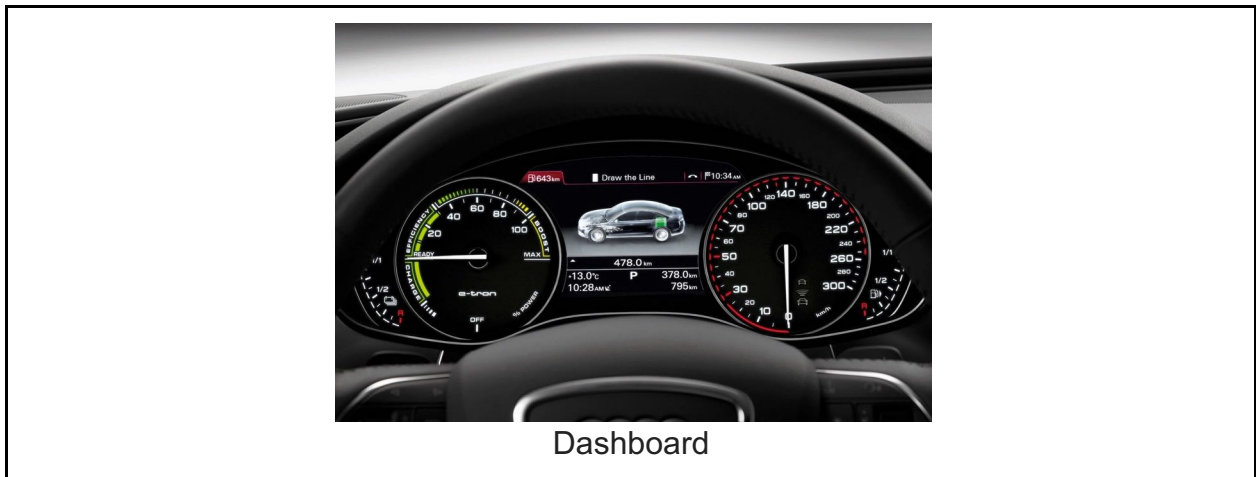
Side view mirror

Gear shift

 <p>Handbrake</p>	<p>1 hand, wrist and elbow movement</p>
 <p>Ventilation control and air vents</p>	<p>Use of the fingers</p>
 <p>Car radio</p>	<p>Use of the fingers</p>
 <p>Seat belt</p>	<p>1 or 2 hands, wrist and elbow</p>

 <p>Emergency flashers</p>	<p>Use of the fingers</p>
 <p>Visor</p>	<p>1 or 2 hands and elbow movement</p>

Objects player cannot interact with:





### **Modality**

- This environment requires 2-hand controls
- To avoid nausea, the car would move at slow speeds
- This environment requires player to be in seated position
- The doctor could take the second seat as “instructor”
- The player would be represented by a pair of hands and looking down he should see its legs and body. (One standard body for female and one for men)

### **Assistance**

- The player can be placed side by side with an instructor. (doctor/carer)
- Assuming there will be other cars the amount can be modified so players focus on the basics not traffic navigation.
- The number of curves, intersection, traffic lights and objects that increase difficulty can be tuned.
- Input from a player's actions can be modified according to needs (amplified or reduced).

### **Social**

- Being assisted by an instructor could also give you points according to how well you drive and a shared leaderboard among the players. (There is not going to be a global leaderboard, but it is divided per difficulty. People with similar abilities will be put in the same leaderboard.)



### **Supported Activities**

Common actions performed around a car are the following:

1. Upon entering the car, the patient takes out the **keys** and according to the model of the key we can have 2 movements: inserting the key in the keyhole and rotating the hand or pressing a button with the fingers to open the car from remote.
2. Patient **closes the door** with the movement of the wrist and applies strength on the handle. Movement would require the person to be able to grab, hold and pull toward himself.
3. Patient **opens the door** with the movement of the wrist and applies strength on the handle. Movement would require the person to grab, hold and push (from inside a car).
4. Patient would grab the **safety belt** with the use of the fingers, wrist and elbow motion to fasten it.
5. Patient would insert the keys to **start the engine**, this requires the use of the elbow and a rotation movement from the wrist plus holding the keys with fingers.
6. In order to start the car, the unlock the change and **change** gear, involving the movement of the elbow and applying strength.
7. Switching **Radio on/off** involves pressing with the fingers.
8. Showing **directional arrows**, turning on the **headlights** or using the **wiper blades** involve using the finger to set the proper command and activate it.
9. If the environment is set in a sunny day scenario there could be a movement to set the **sun visor** through the movement of hand and elbow.
10. In order to drive the patient will have to grab the steering wheel and hold it with his hands and move his arm/elbow to change direction.
11. To use the horn, patient will have to use one hand and press on the middle part of the steering wheel applying force with the hand/ fingers.
12. In the car the patient would have to change gear with the help of hand and elbow movement, placing the correct gear applying force (pulling, pushing and lateral movements).



13. Open the windows can be done using either a retro (recommended) rotating style window handle or a push-button powered electric window control.

	Example 1: windows with handle.
	Example 2: Push a button down/pull up to open/close the window.

14. Change the direction of the **air vent** which requires the use of the finger plus elbow motion toward a direction.
15. The patient could switch on/off the **radio** and change songs using the movement of the fingers and applying force to press a button.
16. The patient could also switch on/off the **air conditioner** through the same movements used for the radio.
17. In the car, the patient would have to change the angle of the **rear-view mirrors** adjusting it with the movement of the wrist.
18. The patient could open/close the **storage compartment** using fingers, the movement of the arm and applying force.
19. They could use the **emergency brake handle** holding it and pulling it up or pushing it down.
20. The patient would adjust the **side view mirror** using the hand and moving the elbow.
21. The patient would adjust their **seat** using the hand and fingers.



22. The patient could use the **windscreen wiper** to clean the windshield, activating it through a button he will push with the fingers.

## Concerns

- If this concept is chosen, we must find creative solutions to how the car accelerates and stops without user behavior and without making them feel nauseous.
- Will the controller support add-ons that resemble something like a steering wheel? Having 2 free hands might break the impressiveness.

## 8. MUSIC ROOM ENVIRONMENT

The music room could be a theme fitting all range of ages. Instruments are expressive tools that people like to practice with and music is a common theme employed in the videogames industry spawning the genre of Rhythm Games with titles such as “Guitar Hero” or “Beat Saber”. Music is beneficial to people because it can stimulate or calm down according to the feeling it evokes in the person listening to it. A person who was injured might feel comfort in playing a VR instrument that they find difficult in real life.

5

Controller Requirements / Activity	Guitar / Bass	Drums	Piano	Violin	Maracas	Xylophone	Tambourine
Finger Tracking	Nice to have		Required	Required			
Palm Open/Close	Required	Required			Required	Required	Required
Wrist Movement	Required	Required		Required	Required	Required	Required
Elbow Movement	Required	Required	Required	Required	Required		Required
Wrist Resistance							
Elbow Resistance							



Table: Controller Requirement Matrix

### Notes

- Possible instruments are: Piano, Xylophone, Maracas, Triangle, Tambourine, Cymbals and Acoustic Drums.
- It would be possible to include a large variety of instruments if the input is simplified.

<sup>5</sup>Study about dystonia in musicians. [Link](#)

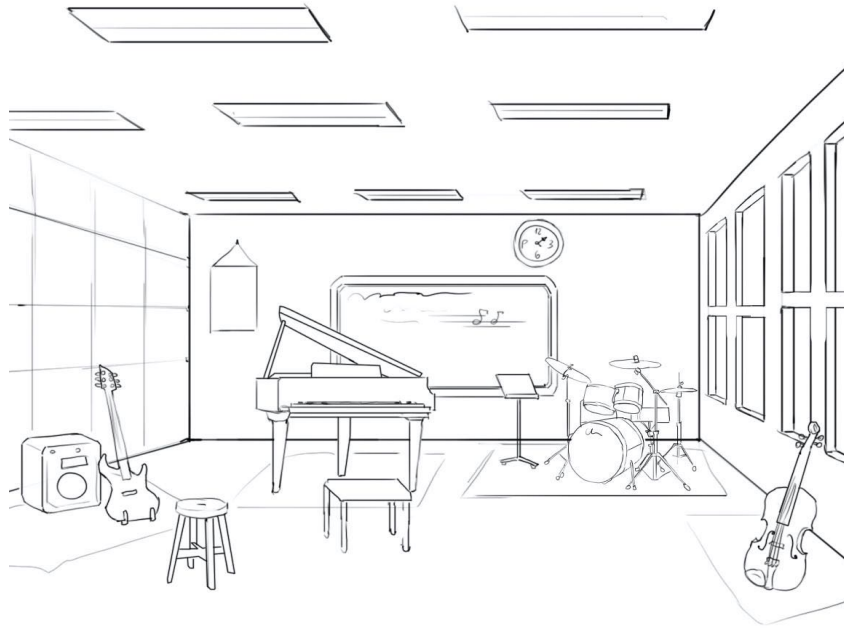


Example: Guitar Hero's guitar comes with 5 buttons instead of strings.

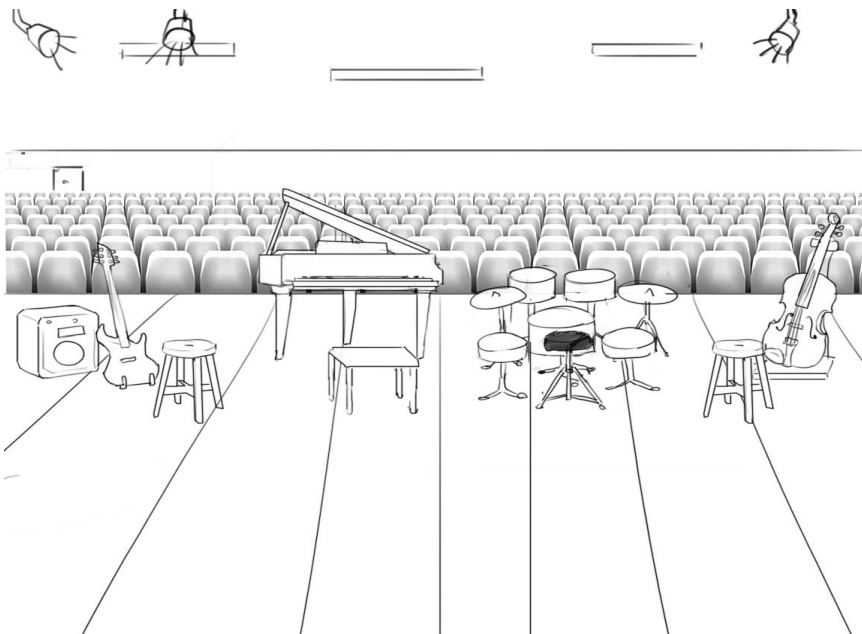
- The speed of the song and must calibrated to each game and level of difficulty (assuming there is going to be a game following the style of the example above)

## Visuals

- The player finds himself in a room with a couple of different instruments he can play with.
- Each instrument has a station and the stations are around the player.
- The player is represented by hands.
- The room the player is in is a big room whose walls are covered with sound absorption panels.
- The room is big and colorful with poster on the walls representing bands and so on, natural light fills it giving it a welcoming atmosphere
- The room could be in a school or in a private house
- The light comes from a window but there is also artificial light on the ceiling of the room (artificial LED of various colors might be an option to consider). Lightning is especially important in real life for performing music because it stimulates the player and creates ambience.
- Each station accesses a rhythm game related to the instrument. For example, the player accesses the piano area and each key or subset of keys of the piano corresponds to a note the player must press. [video example](#)



Concept: example of the room once the player enters the game, instruments are all around him and easy to reach. Each instrument defines a station



Concept: Second iteration of the music's room layout. The layout was changed because it was difficult to teleport for the player to teleport correctly in every station.

## Social

The music room is somewhere in real life musicians meet to play. In this case we could define the social aspect of the game like:

- Creating a system where players can share their music and have likes.
- Sharing their creations on social networks.
- A scoreboard that applies to existing songs (players can get points for accuracy)
- Playing the same song with another patient (tentative).
- Joining a music course with a teacher (assistant).
- There could be a system of trophies and reward the player must achieve as well as daily tasks.
- There could be a leaderboard with the songs for each song or a leaderboard with the best scores

## Modality

- The activities can be done both standing up or sitting down.
- 2 hands are required for most activities, but some can be simplified down to one.
- All instruments are placed around the player and they can be easily reached.
- The level of difficulty can be changed according to the patient's needs; for example, playing the piano could be done with patterns composed of just 2 keys and then scaled up with practice.

## Required movements

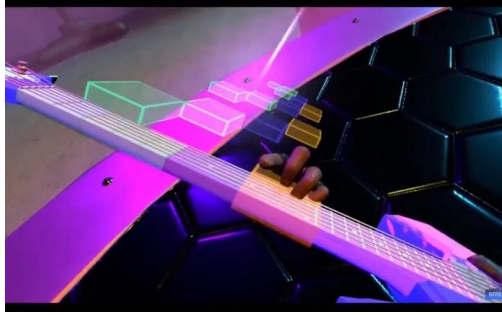
In this environment the movements required from the player are:

- Pressing the keys of the piano with the fingers
- Grab the drumsticks
- Hold the drumsticks
- Move the drumstick to a required location (the drum or xylophone)
- Movement of the shoulder-wrist to move the drumsticks
- Movement of elbow with the maracas

These movements are explained in more detail below.

## Concerns

If this concept is considered, we must find a way to represent instruments played with strings such as guitar or violin which are harder to represent on VR. The solution adapted in this document is to divide the string instrument into bigger segments that would act as discrete buttons (see [Riff VR](#)).



Riff VR partitions the guitar neck board and uses these as discrete inputs.

## 17.25. Instrument Breakdown

---

### **Guitar / Bass**

*Played:* Standing or sitting position

*Hands:* requires both hands.

Instrument can be picked up and locked on position on the chest as if it were strapped (req. Upper body tracking). Needs to be height adjustable. Support for right-handed possible.

*Simplification:*

The fretboard is divided into 5 parts, each part is roughly the width of a hand. The player can slide his left hand up and down these 5 parts. By clutching the fingers / pressing a trigger that part becomes 'held down' else it is 'held up'. The right-hand rests near the body of the guitar and can hit the strings with a downward strum.

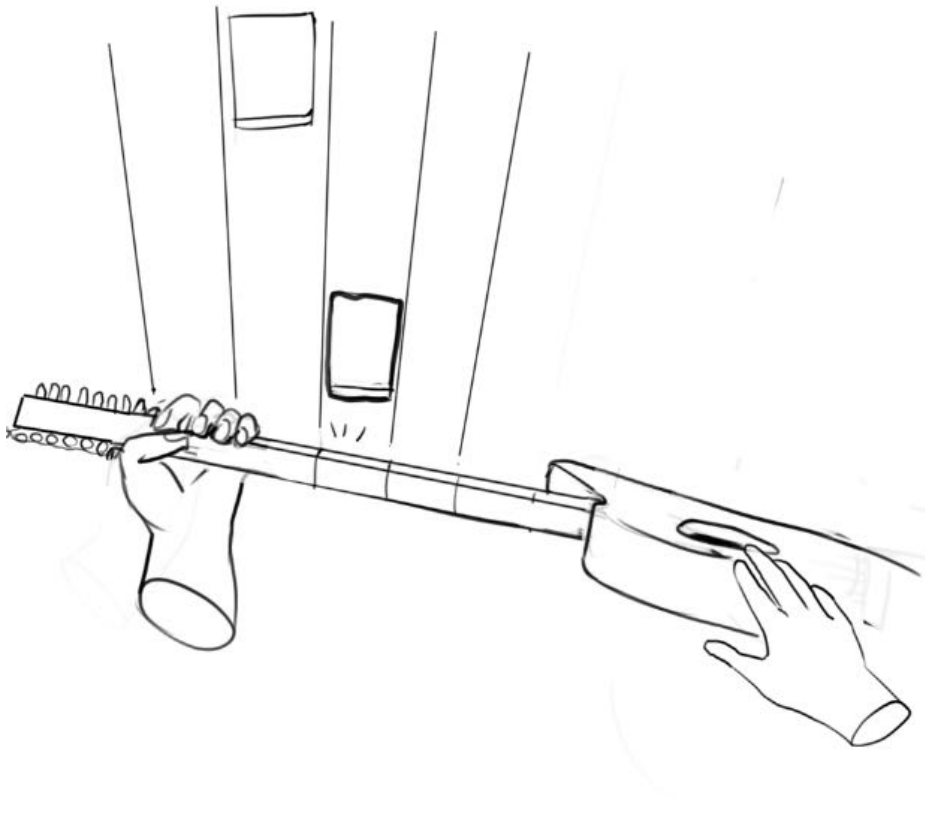
### *How to play:*

With the left hand clutching the correct part of the neck board the right hand can apply a downward or upward strumming motion to hit the strings. The timing counts when the right hand contacts the strings to generate the input.

After starting a song, an adjustable preview of the notes to be played is displayed (attached to the neck board). From here on the gameplay is like other genre titles.

### *Controller Requirements:*

- Finger Tracking (at least Open/Close of Palm for left hand)
- Wrist movement (right hand)
- Elbow movement (mostly left hand)



Concept: example how to play a guitar/bass

### **Acoustic Drum Kit**

*Played:* Sitting down

*Hands:* requires both hands

Instrument is mostly fixed in place however the possibility to slightly adjust range of some instruments can be considered. The parts will be: Hi-hats, snare drum, kick drum, 1x tom, 1 cymbal. The Player can pick up and hold the drumsticks.

### *Simplification:*

To hold the sticks the Player must clutch his fingers or use a trigger. For the most part the instruments will respond to being hit with the sticks. Hitting force can generate higher amplitude but this should not be considered for gameplay. The kick drum is a *concern* at this point because we are not tracking the feet nor using an auxiliary input device such as a pedal. A sensible option would be to add an additional drum that replicates this function/sound while keeping the existing kick drum for visual authenticity.

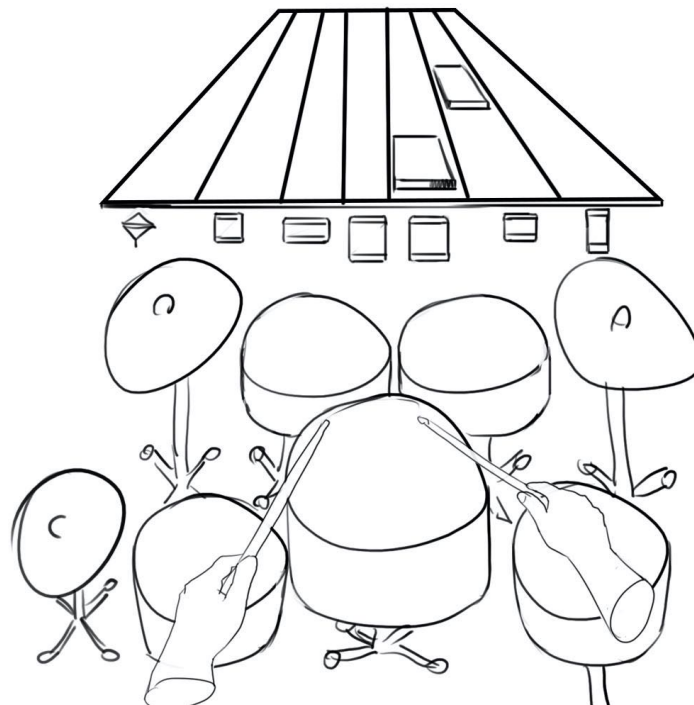
### *How to play:*

Playing this instrument is as straightforward as hitting the correct kit piece as defined by the timing. Timing counts when the instrument is hit with the stick.

After starting a song, an adjustable preview of the notes to be played is displayed (visible directly in front of the tom, notes sliding down). From here on the gameplay is like other genre titles.

### *Controller Requirements:*

- Palm Open/Close (both hands, hold sticks)
- Wrist movement (both hands)
- Elbow movement (both hands)



Concept: sample POV when playing the drums.

## **17.26.**

### **Piano**

*Played:* sitting down



*Hands:* requires both hands

*Simplification:*

This activity has 2 major concerns: how accurate the finger tracking determines whether the keys are similar in size to the real life counterparts and pressed individually or whether they are simplified in larger shapes such that they can be pressed with multiple fingers (less accurate option). This latter approach has an adverse effect on the Piano's visual authenticity but is perhaps the more sensible especially for people who are not familiar with the piano at all. Below we describe such an option.

The number of keys is significantly reduced to about 8, all keys will be at the same level (no flats or sharps). The Player is expected to use both hands, each hand covering the range 4 keys. Depending on how accurate tracking is these keys can be small such that they can be pressed with individual fingers or, alternatively larger keys can be used and the whole hand acts as the activator.

*How to play:*

Playing the piano would be as straightforward as hitting the correct input keys at the correct time as indicated by the preview.

*Controller Requirements:*

- Finger Tracking (both hands, pressing of keys)
- Elbow movement (both hands, seek keys)

Other instruments

Some short notes about other instruments that can be supported:

- *Violin:* Simplification on the neck of the instrument can be applied however finger tracking would be a requirement as the instrument's neck is quite small. The right hand holding the bow would need to be moving to sustain the note. Use of both hands required but can be played sitting down or standing.
- *Maracas:* Can be used with one hand but two hands offer a richer experience in terms of gameplay. Playing this instrument in VR would involve the Player hitting or keeping within the range of note blocks coming towards him/her from the horizon; like many existing VR beat games. Due to the motions required for this type of activity it is likely to be done in standing position only.
- *Xylophone:* Like the acoustic drum kit example however this instrument can be represented on a 1:1 basis depending on the accuracy of the tracking and the ability of the player.

- *Tambourine*: shares similarities with maracas with the differences that the motions followed can be different. For example, the technique hitting the tambourine with the palm of your hand can be incorporated into the motions to perform, it is not simply a case of meeting the end of the hand with the oncoming block or pattern.

## 17.27. Game References

---

1. Audioshield
2. Holodance
3. Audica
4. Dance Collider
5. Synth Riders
6. [Electronauts](#)<sup>6</sup>
7. Beat Saber
8. RIFF VR
9. Beats Fever
10. Airtone
11. Rock Band VR
12. Soundboxing
13. [Guitar hero](#)

---

<sup>6</sup> List from [VR game critic](#)

## 9. OFFICE ENVIRONMENT

Perform your day-to-day office duties as required - one day that Employee of the Month Award (and hefty bonus) will be yours!

Controller Requirements / Activity	Filing	Office Accessories	PC operation	Signing	Rubber Stamping	Paper Toss	Phone Answering
Finger Tracking							
Palm Open/Close							
Wrist Movement							
Elbow Movement							
Wrist Resistance							
Elbow Resistance							



Table: Controller Requirement Matrix

### Visual

- Player finds himself at the seat of an everyday L-shaped desk in a slightly cramped but luxurious (for his stature) small office.
- Front: Monitor + keyboard + tower of the era (depending on average age of patients this can be late 90-mid 2000s, i.e. CRT monitors)
- Other objects/interactables close at hand
  - Folder Rack
  - Desk Drawers
  - Filing Cabinets
  - Accessories: Stapler, Puncher, Pen, Pen holder, Radio, Rubber Stamp
  - Coffee Machine + Cup
  - Dart + Hanging target
  - Corded Phone

- Whiteboard / live display
- Behind player: relaxing garden view
- Infront player: view into corridor. Beyond the corridor is another garden with further view obscured by vegetation. Occasionally people can be seen walking in front and/or entering office from here.
- Lighthearted feel stylized aesthetics.
- Possible locations/roles: Law Firm, Online Retail Company, Storekeeper, Event Organizer, Bank.



Concept: example of an office layout for a VR game where everything is around the player.

### Modality

- All activities inside the office are performed sitting down.
- Nothing should be directly out of reach; we can imagine the Player as being on a swivel chair.

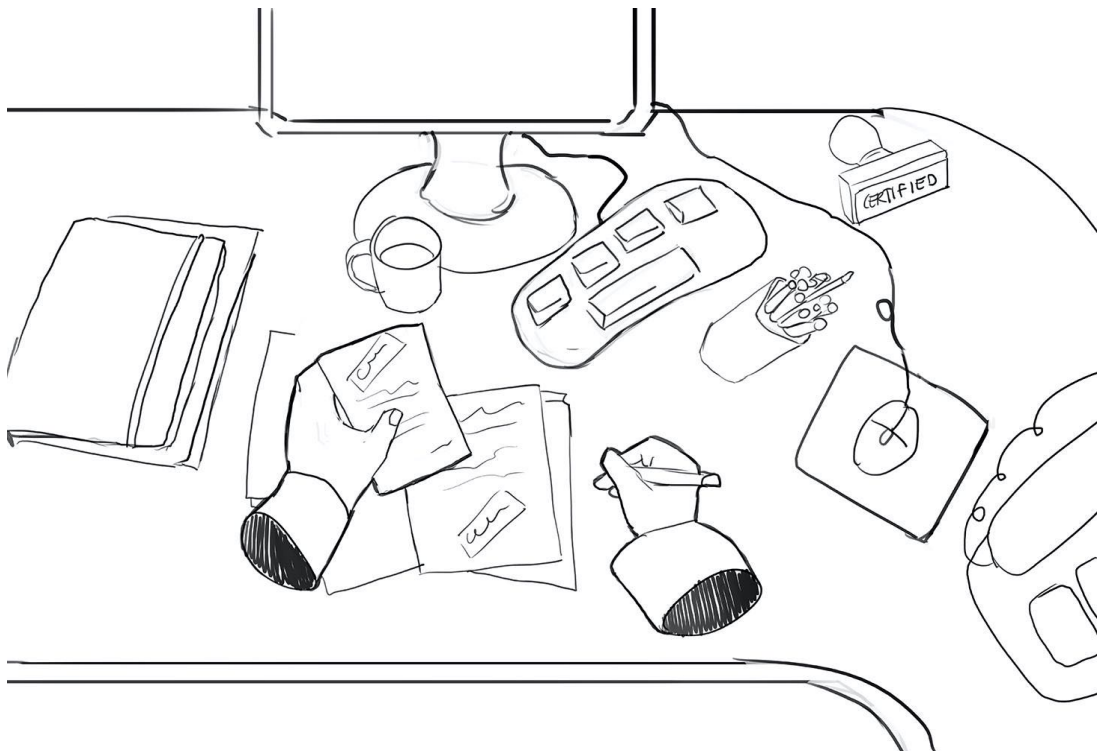
### Assistance

- Scalable difficulty: amount of actions required; number of jobs reduced, as necessary.
- Second player can drop in and act as a second pair of hands that can perform any action (as listed further below) the patient can.

## Social

- Assuming a deterministic flow of tasks in the game, players would be able to see their performance to achieving the Award against other players or against your previous efforts. This acts as a public leaderboard and is visible by any player.

## Supported Activities:



Concept: example of gameplay on the office's desk

- **Gameplay:** The Player starts the day and is given a set of tasks to do for that day. This is communicated via a board or live display in-environment just in front of the Player. There is no set time limit for completing the objectives however the run is timed, and this time affects the rating/score (shorter time -> better rating).

*Objectives* update in real time, completed objectives may disappear from the

board. Some objectives can be cryptic and are left to the player to figure out: for example “I hate my boss!” would require the player to pick up a playing dart he has casually lying on the desk and throw it at a picture of the boss on the wall. These can be optional objectives.

Additional objectives can occur people show up requiring objects or documents which you must find and hand in. Emails can come in adding a new objective.

At the end of the day the Player’s performance is graded and a rating is given of where he is stacked up with the other fictional AI employees that are all competing for the same award. This goes on until either all days are completed, and the Player has either won or lost the award.

- **Discrete activity types:**

- Filing: grabbing folders from the rack and putting them inside the correct filing cabinet drawer.
- Punching paper before sliding it into a folder.
- Paper clips/stapler: attaching mixed photos to their application forms (match the comic description to the picture).
- Use your computer: use the simplified keyboard (contains 4 large buttons) to match out a sequence on your PC. Use this to reply to emails, calendar events, play a simplistic minigame.
- Write with your pen: your signature on papers
- Rubber stamp forms: decide who gets taken as new employees based on some criteria.
- Paper toss: Hit specific objects/other employees (not so kind to you) by tossing a piece of paper, launch a paper plane for longest flight,
- Phone call forwarding

### **Day Example:**

Board contents:

- Re: Janice / Accounting

You have one email to process from Janice in Accounting.

(Keyboard entry mechanic / mash any button or pattern match)

- New Applicants
  - Your inbox rack has some new applicants.
  - Decide who to approve with your rubber stamp.
  - Do not approve inadequate candidates!
- Customer Support
  - You will receive some calls during the exercise.
  - Answer all calls but do not forward any of the angry sounding ones!
- “It wasn’t me”
  - Hit a passerby with any object from your office.

**Game References:**

- *Job Simulator*: general dynamics of play
- *Untitled Goose Game*: objective presentation

**10. BAR ENVIRONMENT**

*Manage your bar properly, keep everyone happy and watch those customer ratings soar!*

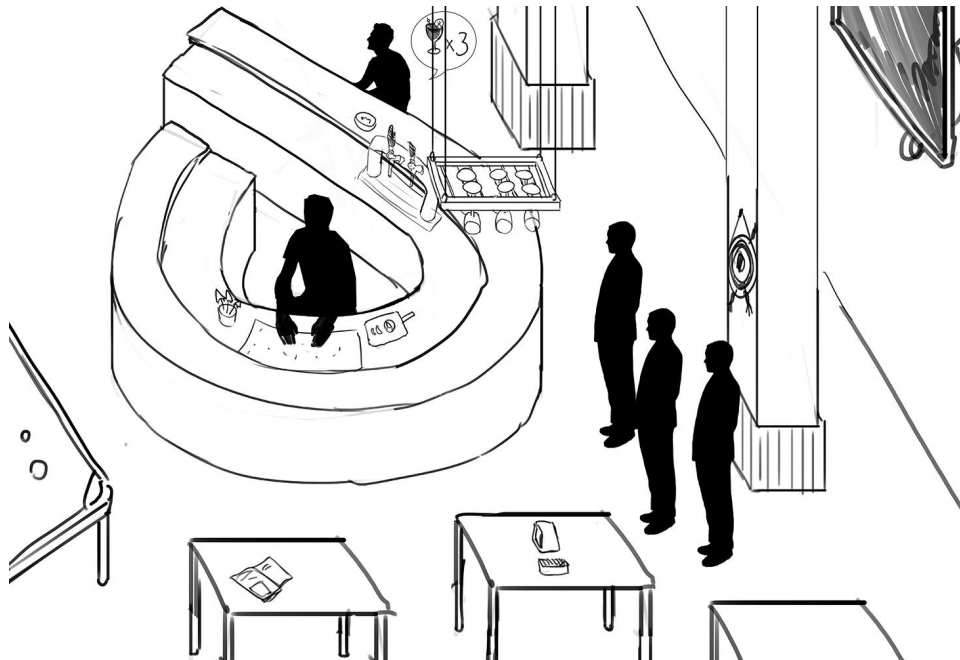
Controller Requirements / Activity	Mixing Cocktail	Pouring Pint	Operate Remote Control	Fly Swatting	Pint Bowling	Darts
Finger Tracking						
Palm Open/Close						
Wrist Movement						
Elbow Movement						
Wrist Resistance						
Elbow Resistance						

Required
  Nice to have

Table: Controller Requirement Matrix

## Visual

- Player finds himself in the role of a bartender: working the counter of a typical contemporary English pub.
- A selection of drinks is available behind the player as well as empty glasses/pitchers. Empty glasses can also be found under the top portion of the bar directly in front of the player.
- Other items in the immediate vicinity of the Player include the cash register, ice container, peanut container, darts, TV remote, reference tablet, opener.
- Typical evening, light-hearted mood. One can see other tables and patrons sitting enjoying their day.



Concept: an idea of how the bar layout might be, all objects are around the player who can have easy access to them.

## Modality

All activities carried out assume standing and stationary position though he will be required to turn around. This should be adaptable (by means of a high stool) to players unable to leave a sitting position.

## Assistance

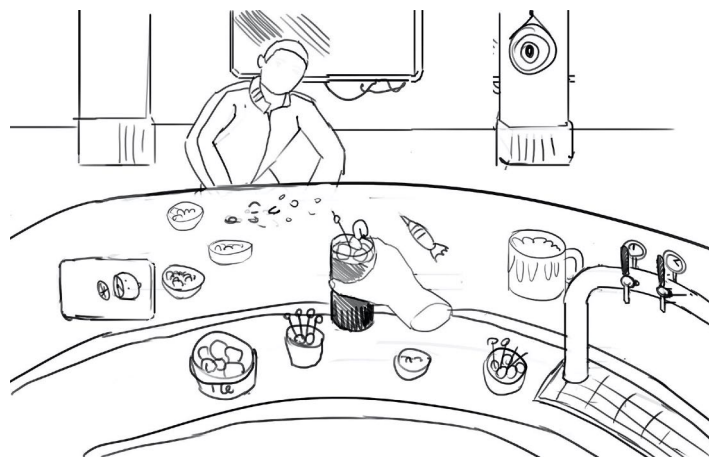


- **Adaptable difficulty:** actions required to perform action can be simplified. Opening a bottle may not require twisting/using an opener or mixing drinks may not require shaking at all. There is a limited amount of actions that can be omitted in favor of easier difficulty and some games would not be as scalable, if at all. Visual aids such as highlighting can alleviate difficulty.
- **Direct assistance:** in the case where the above method is not enough direct intervention from an assistant might be necessary.

## Social

- None but perhaps we can use progress markers (discovered types of drinks) or achievements sharing.

## Supported Activities:



Concept: first person view of player serving a cocktail to a customer.

- **Mixing a cocktail:** a *main activity* that would happen often is taking in an order for a cocktail or drink and having to make it on the fly. This would involve looking up a simple recipe, pouring and shaking and presenting (depending on difficulty) it. Occasionally you might be asked to add your own touch to it by adding additional ingredients. We can expand this to creating your own drinks, with some combinations creating hilarious effects (these can be standard orders too)
- **Pouring a Pint + Snack:** just about what it sounds. Can be made more interesting if the 'draught machine' delivering the beer does not always work as expected (slow/fast flows, refuses to work, dispenses something else instead of beer, etc.).

- **Changing TV station:** keep an eye on what is coming up on the television - make sure you switch to the correct football match to keep the patrons happy. (Using a remote control)
- **Fly swatting:** Pick up your trusty fly swatter and dispose of the annoying fly that will occasionally come and pester you.
- **Pint bowling:** Some patrons at the far end of the bar top will sometimes demand beer. Pour pints and slide them to the end of the counter without them dropping or hitting anything else on the way.
- **Play darts:** throw darts anytime or play a game against one of the patrons.
- **Other:** Takeaway handling, Handle Cash Register, Happy Hour bell tolling.

#### **Game References:**

- *Job Simulator:* general dynamics of play
- *VA-11 Hall-A:* objective presentation

## APPENDIX 5

### CONTROLLER INTERACTION MATRIX

Controller Requirements - Input	
Name	Definition
<b>1. Movement Tracking</b>	<i>Most games need to know the position of the hands, their movement speed, and their rotation</i>
A. Position	E.g. Pushing a button or painting with a paintbrush on a canvas
B. Rotation	E.g. Watering a plant
C. Velocity	E.g. Throwing darts
<b>2. Friction (force requirement)</b>	<i>Living labs mentioned a requirement where players must exert force to play the game. (Around 20kg of force)</i>
2A. Rotation movement (orientation)	E.g. Opening a tap of water
2B. Horizontal movement	E.g. Pulling the string horizontally while doing archery
2B. Vertical movement	E.g. Pulling a heavy sack in an upwards/downwards movement
<b>3. Finger tracking</b>	<i>In some cases, we need to track if palm is open/closed OR individual finger movement</i>
3A. Palm status	E.g. Using a stapler at the office
3B. Finger movement	E.g. Playing the piano
<b>4. Joysticks, triggers, and buttons</b>	<i>We have not outlined how many buttons and triggers we need yet but we will need at least 1 joystick, 1 trigger and 1 action button.</i>
4A. Joystick	E.g. Turning around in some environments
4B. Trigger (multistage)	E.g. Start action to pick something up
4C. Action buttons	E.g. Push a button (trigger could be used as well but best have other options)
Controller Requirements - Output (Feedback)	
Name	Definition
HD Haptic feedback	<i>Haptic feedback like commercial controllers that help simulate and give feedback on actions taking place in game.</i>