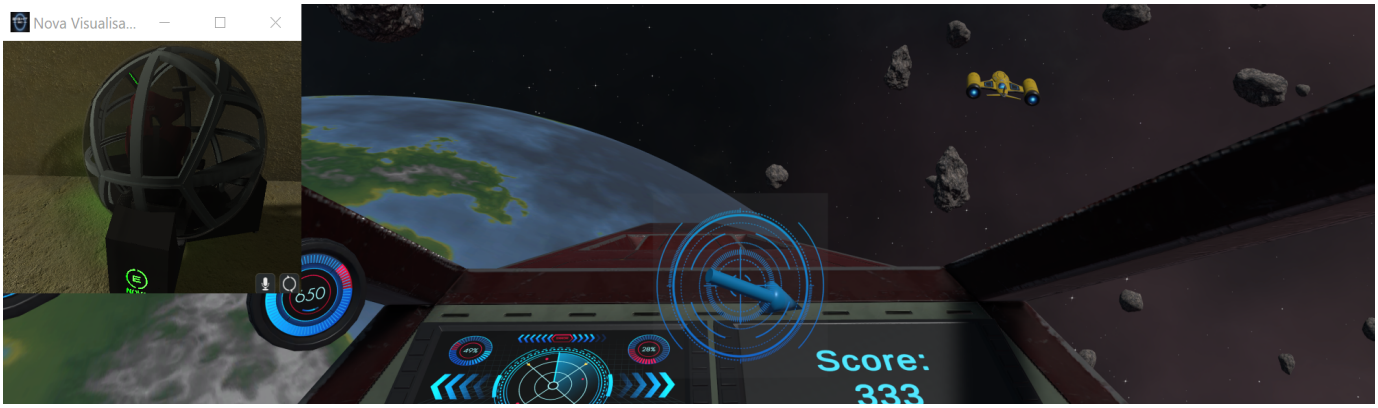
Understanding Sickness and Emotional Experiences in 360° VR Motion Simulators

Harley Welsby



***Abstract*—Simulating real-life scenarios for training and entertainment purposes in Virtual Reality (VR) is more realistic than ever before. Currently available VR head-mounted displays are unable to match movements in VR simulations to real physical movements effectively. Eight360's Nova device is a 360-degree VR motion simulator which physically moves its user in time with their virtual movement in VR simulations. The Nova brings to light new unknowns on whether 360-degree motion simulation adversely affects VR motion sickness (cybersickness). This project developed a spaceship flight simulation game for the Nova to evaluate motion sickness symptoms. A user study was conducted to evaluate the symptoms experienced and to better understand the causes and effects of these symptoms to inform future design of VR simulations and 360-degree motion simulators.**

***Index Terms*—Virtual reality, simulation, cybersickness, motion sickness.**

I. Introduction

THROUGHOUT the history of virtual reality (VR) technology, motion sickness (cybersickness) has been a critical issue at the forefront of device and simulation design [1]. The use of head-mounted VR headsets has proven to cause symptoms of motion sickness such as "significant disorientation" and rising heart rates [19]. Studies such as D. Saredakis et al. [20] have proven that video game and task-based content heightens the symptoms experienced, and can be a significant blocker stopping some people from using VR technology [18]–[20]. A primary cause of cybersickness is the difference in motion between one's physical and digital presences [19]. Theoretically, matching the movements of one's physical presence within VR content could reduce effects of cybersickness and potentially make VR content more accessible to the general public. With the development of full-body motion simulators such as Eight360's Nova device [21], matching virtual movements for VR has become technologically feasible. However, new technology inherently means new unknowns as the extent of the difference in motion sickness has yet to be fully understood. Two previous studies have been conducted assessing the difference between headset-only VR usage and Nova device usage [7], [8], but little is known about the effects of differing movements and Nova-specific variables such as movement speed and acceleration on users of 360 degree motion simulators. This project sought to better inform future designs of VR content and 360 degree motion simulators by assessing the effects of the Nova device via the use of a bespoke software application, in the form of a VR spaceship flight simulation game with support for the Nova [21]. A user study was then conducted using the software application developed, to assess the effects of the device on motion sickness and identify any links between symptoms experienced and rotational movement.

This project was supervised by Craig Anslow (primary), and Christopher Maymon.

II. Related Work

A. VR Motion Sickness and Evaluation Strategy

To understand the effects of the Nova device in regard to motion sickness, an evaluation method must be developed to determine an accurate reading of one's sickness levels. Previous studies have shown that the Simulator Sickness Questionnaire (SSQ) [1] is able to provide accurate readings of common motion sickness symptoms [1]–[3]. Symptoms are categorized by Nausea, Oculomotor (eye comfort) and are typically scored from 0-3, where 0 is no experience of the symptom and 3 is a severe experience [3]. The SSQ system has been standard in academia since it was developed in 1993 [4]. The primary limitation of the SSQ system however is that there are a lot of questions asked [1], [2], meaning it is not well suited to being continually answered verbally during an experiment. Another study has been done previously assessing an adaptation of SSQ called FastMS, which uses

short questions such as “How sick do you feel?” answered on a scale from 0 to 20 [2]. As this system has been proven to be accurate, it can stand in place of SSQ for during-experiment questions to participants, to keep things short and simple [2]. To properly evaluate the solution, metrics should also be given on the task presented to participants, and the usability of the software. Two key systems have been developed previously to quantify software usability, the NASA Task Load Index (TLX) [5] and the System Usability Scale (SUS) [6]. The TLX scale focuses on task achievement and stress experienced when undertaking a given task via 6 questions answered on 21-point scale, such as “How mentally demanding was the task?” and “How hurried or rushed was the pace of the task?” [5]. The SUS scale is oriented more to usability, with 10 statements such as “I found the controls too complex” and “I need support of a technical person” answered from “Strongly Disagree” to “Strongly Agree” [6]. Each are suited to assessing software and Nova device functionality but with different approaches, either task-focused or directly stated [5], [6]. These two questionnaires complement each other well, and are the standard in assessing usability.

B. VR Simulators and Previous Solutions

Two prior projects have been run to assess sickness levels of the Nova device, by C. Simmonds [7] and C. De Bruyn [8]. The first featured a roller coaster simulation built from scratch, where participants experienced a short roller coaster ride (See Fig. 1) with no control over the simulation [7]. The simulation had no player controls, and participants were run through a pre-programmed course [7]. This has limitations in that the simulation can only be used for the track provided and there are only a few variables that can be changed [7]. However, no controls means no variation in the experience, and participants would always be run through identical movements in the device which is more consistent for data collection [7]. The second project instead used a modification of an existing VR game, *Subnautica* [9], to support the Nova device, meaning the visuals were much more pleasing as they were developed by a professional game studio [8]. C. De Bruyn’s project gave controls to participants and had them move the Nova themselves using a handheld game controller [8]. The consistency issue was mitigated by the use of a pre-programmed checkpoint system, where all participants would navigate the same course in a submarine with little room for variation [8]. The biggest difference between the two prior solutions however is what the solution was built upon. C. Simmonds’ solution was built from scratch using Unity, which has benefits in that the developer has absolute freedom over what can be developed in the game and that any required changes to fit requirements would be feasible, at the cost of spending more time implementing basic functionality that would already exist if one were to modify an existing game [7]. C. De Bruyn’s solution was instead a modification of the existing VR-supported game *Subnautica* [9], to provide a checkpoint course and allow for control of the Nova device [8]. A modification could be quicker to develop as the game has already been built, and just requires an extra component to be added to allow for Nova device rotation [8], but has drawbacks in that the developer is limited to the modding API for the game. Some games are open-source and provide their code online, and some actively discourage modification, often to prevent cheating. The choice made here should reflect the project goals and requirements, and a solution of either method has the potential to meet the project requirements.

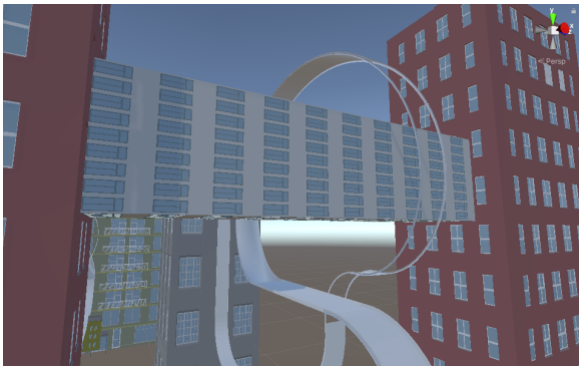


Fig. 1. A section of the roller coaster track from the previous solution [7].

Fig. 2. A screenshot of *Subnautica* [9] from C. De Bruyn’s solution [8].

III. Solution Design

A. Requirements

The requirements for this project are as follows:

Req1: The solution must be a playable VR experience.
Req2: Control of the Nova device must be supported.
Req3: The solution must involve first-person spaceship flight.
Req4: Participants must be able to rotate in every direction.
Req5: Experiences between participants must be consistent.
Req6: Data must be output on the Device’s rotation.
Req7: The solution should be expandable for future work.

The requirements given by Victoria University of Wellington upon beginning the project were minimal and open-ended, provided that the solution was fit for purpose in a user study with several constraints. These initial constraints advised by the project supervisors being that the solution is a VR video game involving first-person spaceship flight, in which the player controls. It was also specified that the Nova device must

be able to rotate in every direction. Building on these initial requirements (Requirements 1-4 below), the "fit for purpose in a user study" specification had to be resolved. As seen in previous work, an important part of this is consistency between subjects to retrieve accurate results. Therefore, Requirement 5 is defined in that experiences between participants must be consistent. Another difficult question to answer here is how the analysis will be run after the user study, and which data can be used to evaluate the symptoms effectively. While the user study planned to primarily evaluate symptoms, it cannot be ignored that participants will move in different ways as they have full control over the game. Therefore, Requirement 6 was defined that at all times, data must be output on the device's current rotation. This means that differing levels of rotation between participants can be easily analyzed and interpreted to factor in any major differences in symptom analysis which could act as causes of said symptoms. As the development of a VR game from-scratch is a large undertaking, it should be possible to use the software in future work. Effort should be made to design the software in a way which facilitates later expansion (Requirement 7).

B. Design Choices

Drawing from previous solutions, the biggest decision is whether to modify an existing VR game, or build one from scratch [7], [8]. After initial investigation, the project proposal stated that a modification would be developed for No Man's Sky [10], a VR-supported open-world spaceship flight game with an active modding community [11]. Upon speaking to members of the community and investigating the structure of the game's modding features, it was concluded that the game did not support real-time retrieval of data values such as ship rotation, meaning that control of the Nova device would be near impossible [11]. A possible option could be to use an in-memory editing software such as CheatEngine [12]. However, this software is known to contain viruses unless payment is given to an untrusted third-party and is commonly used as a hacking tool for cheating in competitive games [12]. The use of this tool would be ethically dubious and thus the option of modifying this game would not be feasible for this project. Since no other game could be found with adequate modding tools, it was decided that a game must be built from scratch. Several game engines exist with VR support, the most popular being Unity [13] and Unreal Engine [14]. C. Simmonds' project used Unity [13] for game development as the Nova device SDK is written as a plugin for Unity [7]. Using another engine would require the Nova device controller to be rewritten, therefore Unity was chosen for this solution as well. Since Unity has been chosen as a game engine for this project, the solution will follow Unity's conventions and framework. Unity projects are organized as a nested series of components, where a component is anything from an individual object to a code file. The highest level of this component architecture is the Scene. The Scene acts as the uppermost parent component of the solution as well as the visual 3D space for the game. The scene must contain a folder of checkpoint objects, laid out in the scene as specified by the checkpoint system design. There must also be a folder of obstacles and scenery containing asteroid/planet game objects, and a folder of enemies containing spaceship game objects. Finally, a Spaceship game object is required and should be placed at the top of the scene directory. Each of the folder directories acts as a bounded context for the inner components, where nothing outside of the folder component should interact with these objects.

C. Game Design

With the requirements of supporting consistency between subjects for data collection (Requirement 5) but encouraging movement (Requirement 4), a checkpoint course is a an ideal task as it gives control to the player but in a consistent manner and with as much movement as desired, depending on how the course is set up. This method has also been used previously in C. De Bruyn's [8] project, where it proved a good task for consistency which participants understood well. The task requires participants to follow a set of checkpoints through 4 "stages", where after each stage one variable changes for assessment on sickness effects in the user study. Each stage should be consistent in length at 10 checkpoints each and no more than 2 minutes in length, with an 11th checkpoint straight forward from the 10th to allow time for verbal questionnaires on symptoms.

The stages are organized in increasing difficulty as follows:

Stage 1: Basic movement and slowed controls.
Stage 2: Basic movement, regular controls.
Stage 3: Continual up and down movements.
Stage 4: Intensive obstacle course.

As well as the checkpoint system, a scored shooting system was added. This is because the initial checkpoint task while good for consistency, was often boring or too little in terms of interaction. The shooting system creates more difficulty and therefore requires more engagement from the player, keeping them immersed in the game and thinking more about the task than how they're feeling at any moment. Although it should add difficulty, the shooting aspect should not take away from the checkpoint course as the course is essential to keep the requirement of consistency (Requirement 5). However, there needs to be incentive for players to use the system. If the player were required only to shoot at static objects such as asteroids, it likely would not be engaging enough to gather interest. The solution devised is that during the checkpoint course, several enemy ships should fly through the player's view on a randomly generated path from their spawn point, which can be shot at for extra points. Each ship should be worth equal points each, and additional points should be given based on the speed in which participants reach each checkpoint. This adds some variance between subjects however, so the ships should at least spawn in the same locations to promote consistency. Checkpoints should be numbered individually and organized in a singly-linked list, where on game start each checkpoint searches for the next in the list until one has no next checkpoint value. The final checkpoint should be flagged as the last one

and should stop the game once reached. Each checkpoint consists of a visual circle indicating where the checkpoint is, an invisible trigger which activates the checkpoint only when a player has collided with it, and a C# script to handle the location of the next checkpoint and focus switching (See Fig. 3). The benefit of a checkpoint system which auto-detects its own path is that the checkpoints can be moved and remixed at any time. Should future studies take place using the software built in this project, building an entirely new checkpoint course or even a new scene would be simple due to the fact that checkpoints dynamically self-detect. This satisfies the requirement of keeping the software expandable (Requirement 7).

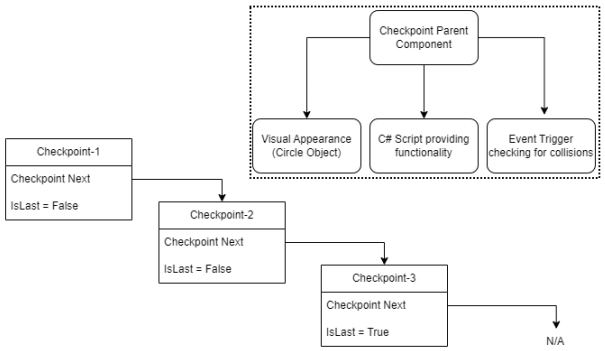


Fig. 3. Structure of the singly-linked list of checkpoints (Bottom Left), and structure of each individual checkpoint (Top Right).

In previous work, a loose handheld controller was used to control the movements of the Nova device [8]. However, the previous project saw no more than 45 degree vertical rotations, meaning that there was no danger to the participant should they let go of the controller [8]. This project has been given the requirement of allowing 360 degree vertical rotations (Requirement 4), meaning the use of a handheld controller inside the Nova is no longer a viable option. The Nova device comes equipped with fixed flight simulator controls (a forward and back throttle in one hand, and joystick in the other), so this is the only option for participants of this solution. The throttle should control movement of the ship forward and backwards, and all turning should be done with the flight stick. A trigger or button on the flight stick should activate the guns on the spaceship for the shooting feature. The most important object in the scene is the spaceship. This will be front and center during all gameplay and will act as a connector for other components, since it is the way in which the player will interact with all other objects in the game. The spaceship component must consist of a parent component acting as the visual representation of the ship, with inner components/scripts attached to the parent and not each other (See Fig. 5). One of these inner components must be the C# script provided by Eight360 to allow for control of the Nova device. The ship must also have a rigidbody component, which handles physics simulations for the ship and acts as the object for the Nova controller script to read rotations from. An OpenXR component [36] must also be added within the Spaceship component to act as the camera the player views from when wearing a VR headset. OpenXR components are physical camera components in the scene and therefore must be placed within the cockpit to ensure the player is seated correctly during gameplay [36]. Another C# script must be provided which handles reading of control inputs and rotation/acceleration changes on the rigidbody component. Finally, a C# script component must be added within the spaceship to handle logging of the rigidbody translation and rotation during gameplay, to provide data for the user study.

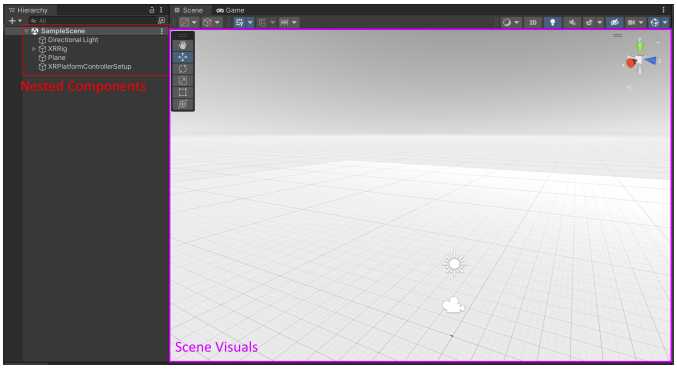


Fig. 4. A default Unity scene using Unity's VR Template. Highlighted in red is the list of nested components within the scene, and in pink is the visual display of the scene itself.

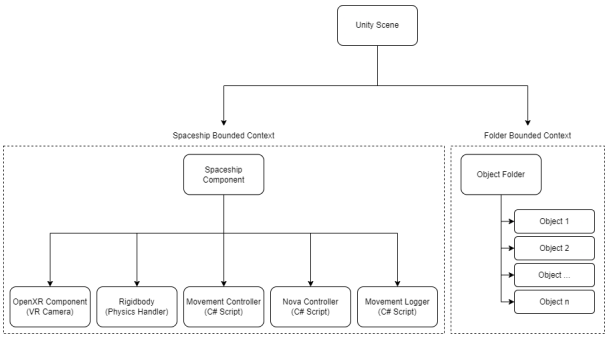


Fig. 5. Architecture of the scene, with each sub-component of the scene acting as an isolated parent for the inner components.

D. Visual Assets

The Unity Asset Store is accessible as a free source of visual assets [23]. Unity by default only includes simple colours and surfaces with cube, sphere and plane shapes for providing visuals. To meet the requirement of providing a spaceship flight simulation (Requirement 3), the 3D scene for the environment must visually appear like space, where the surroundings are space-themed and participants sit in a virtual spaceship. This means that a spaceship asset with a textured cockpit is required, for the player to sit inside. A space-themed background must also be provided, as Unity's default appears as a large white box. Any obstacles present should also fit into the general space theme, and be recognizable as objects that would be in space for the purpose of creating an immersive space environment. When findings fitting assets, several free

options were available on the Unity Asset Store [23] which were adequate for producing the game, therefore these were prioritized over paid options which would provide little more value to the project. Assets were chosen based on their theme fit and ability to adapt and modify the provided visuals. For the player spaceship, an asset with a detailed interior is required. The only free asset on the Unity Asset Store featuring a detailed spaceship interior at the time of selection was Ebal Studios' "*Hi-Rez Spaceships Creator Free Sample*" [23], [30], therefore this asset was chosen. For the enemy spaceships players would shoot at as part of the task, an asset was chosen from the same provider, "*Star Sparrow Modular Spaceship*" [31]. This was to keep consistency between models, as the assets have a noticeable style more similar to a cartoon than reality. Therefore, choosing a model from a different Unity Asset Store provider could be immersion-breaking or jarring for the player. While several free assets exist for distant planets, most appear to be 2D or only for background use. "*Earth like Planets*" features 3D visuals which are high enough in resolution to be used as close-up obstacles in the game [32]. Several free asteroid assets are available on the store [23], but are low-polygon or 2D. To fit the theme thus far, just one asset "*Asteroids Pack*" actually appeared similar enough to the ships and planets to fit into the game effectively [33]. Many space backgrounds are available on the Unity Asset Store, so there was much more freedom of choice on which background to use [23]. However, most of these assets have poor reviews, indicating there could be issues with them [23]. Therefore, the space background was selected based on the item with the best reviews to ensure that a visually pleasing experience is provided, with as little hassle in implementation as possible. This asset is Pulsar Bytes' "*Starfield Skybox*" [34].

E. Data Collection and Analysis

1) In-Game Data Output: For the user study, data had to be output every frame containing the current time and the translation and rotation data of the spaceship, to fully understand each participant's movements. This can be done from a C# script component attached to the spaceship in the Unity editor. Unity's API by default uses two main functions, Start and Update. Start runs the first time the component is rendered, e.g. when the game starts. Update is called once per frame. The Start function for the ship logging component should determine an output file using a random GUID for differentiation between subjects, and log to that file that the game has started at the current time. The update function must continually log the position and rotation of the spaceship's rigidbody component, which would be a child component under the same parent as the logging script (the spaceship component). The output file should be in CSV format to facilitate easy parsing of the data in the analysis tools.

2) Analysis Tools: While the checkpoint course is able to provide a guideline for consistency in the study phase, everyone plays video games differently. When evaluating the results of the study, it is important that inconsistencies can be quickly recognized and assessed in participant's experiences. In C. De Bruyn's project, a visualization tool was built in Python [8] using the Plotly low-code data visualization library [15]. The tool tracked a given participant's path in 3D from a data file, giving insight on any inconsistencies that could occur [8]. The major drawback of this system though is that using Python means that the data visualizations must be downloaded and run locally, with setup required in installing and setting up Python. This could pose an issue if any future work were to be done by a non-technical person, such as collecting further data from this solution with modifications. To resolve this drawback, a user interface should be developed which could be used by a wider range of people with less understanding of the underlying code. This could be done in Python, but using a JavaScript web application, which Plotly supports [15], would also mean that the tool is readily available with no setup. Therefore, a web-based analysis tool would be the preferred option.

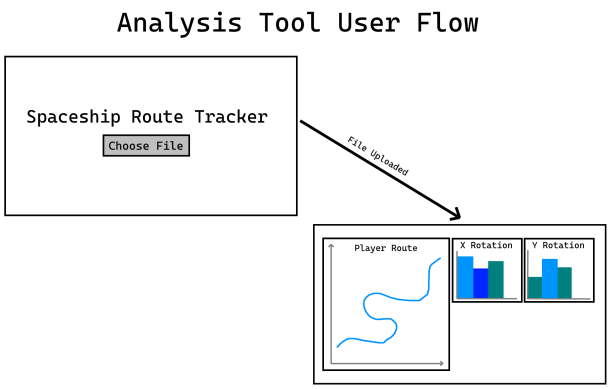


Fig. 6. User flow of the Analysis Tool web application when uploading a log file from the game.

Since this tool is not the primary solution for this project and only an additional tool for later analysis, development should be quick and simple. The React framework features a one-command setup "*Create React App*", which would suit this requirement well as it builds a website template using React and bypasses any initial setup required [16]. React also supports use of TypeScript, which allows strong typing of variables for important calculations, preventing programmer errors. By utilizing a user interface, data could be uploaded by dragging and dropping onto a file picker HTML component. This requires minimal effort from the user and is generally an understood concept as many other modern websites make use of drag and drop file choosing, such as Facebook's Profile Picture upload system [17]. To promote usability, the screen should be as simple as possible meaning that no other features are required than the drag and drop file picker. Once a file has been uploaded, the page should update to display a 3D diagram of a given player's route in the game using Plotly [15], and optionally some additional density plots displaying rotation over time on the x and y axis, to assist in understanding whether one participant rotated more than another.

F. Sustainability

As the project requires no hardware components other than the Nova device [21], the primary sustainability consideration is the use of power for operating the device, and running the produced solution. The more complex operations required in the software, the more power will be required to run it. As the project was run in New Zealand using the New Zealand national power grid, up to 87% of the electricity used to develop and run the project was produced by renewable sources [22]. To reduce power usage, the solution designed uses only operations directly required to fulfill the project requirements, and avoids unnecessarily complex logic to reduce its power usage and increase sustainability. Due to the use of Unity, there is little that can be done to promote sustainability more so than limiting unnecessary calculations in code and their use of electricity. Primarily, conventional standards should be adhered to in writing any scripts to prevent extra memory causing power usage such as unused or incorrectly scoped variables. However, this is not a major concern due to the environmental stability of New Zealand's power network [22].

IV. Solution Implementation

A. Checkpoints

Checkpoints have been implemented as a game object within the scene, under the "Checkpoint Course" context as specified in the design. Each checkpoint object consists of 4 inner components; the Mesh Renderer, Sphere Collider, Functionality Script and rigidbody (See Fig. 7). The Mesh Renderer is included on a new game object by default and handles the display of the visual representation of the object, in this case a green sphere. The Sphere Collider component is visible in the Unity editor as a green wireframe around the checkpoint mesh from the Mesh Renderer, which has been placed over the object and sized correctly around the checkpoint. This component has a flag enabled titled "*Is Trigger*", which allows the use of the OnTriggerEnter() function in C# scripts attached to the same parent object. Therefore, this Sphere Collider acts as collision detection between the checkpoint and other game objects. This is required to correctly detect when players have reached a checkpoint so that it can be disabled and the next one enabled to continue the course. The rigidbody component also helps with this functionality, as it is required to provide physics to the checkpoint object. Although the checkpoint will be stationary, this rigidbody component in Unity lets the checkpoint know where it is and what should and shouldn't be a collision with the Sphere Collider. Finally, there's the functionality script component. This component is a C# code file which determines what the checkpoint should do under certain conditions such as game start, update and Sphere Collider overlap with another game object. Public fields in these C# components are represented as variables on the Checkpoint object in the Unity editor interface, meaning that they can be freely toggled and modified without editing any code. This is an incredibly useful feature for maintainability, which has been utilized on the checkpoints in the form of boolean flags. On game start, a checkpoint will first scan for a Player Ship object, as knowing which object is the player from game start helps keep interactivity simple during runtime and prevent intensive searching of components during the game. Next they will individually determine their assigned checkpoint number, and locate a checkpoint in the game with the number after theirs e.g., if one checkpoint is assigned "Checkpoint-3", it will locate and store "Checkpoint-4" as its next value in a singly-linked list. Should a checkpoint not have a "next" value, it will flag itself as the final checkpoint and trigger the end-game screen once reached instead of unlocking the next checkpoint.

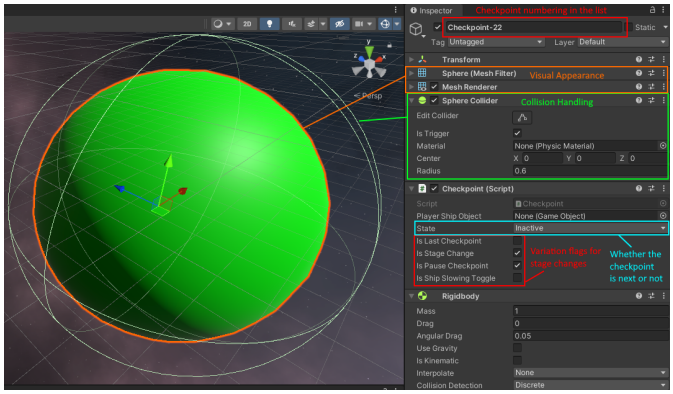


Fig. 7. A dissection of the inner components on a Checkpoint object.

All individual checkpoints act as state machines in two different states, *Active* and *Inactive*. All checkpoints begin as inactive unless they are numbered as "Checkpoint-1". While this could have been done with a single boolean "IsActive" field, this approach was chosen as in future work, more states could easily be added and handled for each checkpoint. Once a checkpoint senses a collision has been made, it will confirm that this collision is with the player's spaceship and if so, set itself as inactive and activate the following checkpoint in the list. When performing this check, the checkpoints will also enact certain functions given the boolean flags set in the Unity editor as mentioned prior. Each flag is a separate functionality, determining whether a checkpoint should be overridden as the final checkpoint, or whether it is a stage change. If a checkpoint is a stage change, then the game should be paused so that participants can be questioned on their symptoms thus far.

B. Checkpoint Task Layout

As per design, the checkpoints have been organized into 4 stages (See Fig. 8). Each stage consists of 11 checkpoints taking roughly two minutes to complete, with one changing variable between each stage. At the beginning of the game and after every stage, there are checkpoints flagged to pause the game for symptom questioning. The changing variables are determined both by checkpoint flags and course design. The first stage consists of just basic left and right movement, with minimal vertical changes in the course. During this stage and from the beginning of the game, the participant's flight stick controls for rotating themselves are slowed to one fifth of their regular speed, via the "*ShipControl*" C# component on

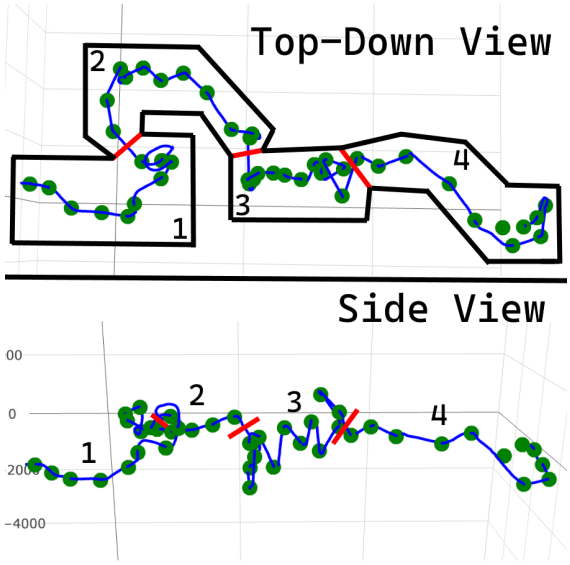


Fig. 8. The checkpoint course layout split by stage, displayed via the 3D visualization in the analysis web application.

the player's spaceship game object. Once stage 1 is completed by reaching Checkpoint 11, the checkpoint will increase the player's flight stick sensitivity to its regular level. This is the variable change between the first and second stages, aiming to determine by the end of the second stage whether the speed in which the player rotates in the Nova device adversely affects their symptoms or sickness levels. Stage 2 consists only of basic movement similar to stage 1, as to not create any unexpected variance outside of the control sensitivity change. Once a player reaches stage 3, they are met with continual up and down movements, forcing them to be nearly upside down or suspended in the Nova device for most of the section. Finally, stage 4 acts as an intensive obstacle course using asteroids as obstacles directly in front of the checkpoints at times as to provide a more mentally demanding task as the final variable.

C. Enemies and Shooting

Enemies were implemented as object-orbiting spaceships. Their implementation differs from the design in that dynamically spawning the enemies around the player proved difficult, and did not fit the requirement of consistency where possible (Requirement 5). Instead, enemy spaceships orbit around a fixed obstacle such as a checkpoint or asteroid, at various points in the game. It was also found that the collision detection provided by the imported visuals was extremely difficult to hit with the shooting implementation, so a wider collision detection box was required so that most participants could easily engage in the game's shooting feature (See Fig. 9). Since shooting is a player-controlled action, it was implemented in the same C# script as the movement controls for consistency. On pressing the designated shooting button as mapped in the Unity Input Manager [29], a pre-defined bullet object is created, which contains a sub-component C# script perpetually moving it forward and eventually destroying itself if it collides with another object or travels for too long. When implementing this feature, it seemed fitting that shooting the obstacle asteroids should also have an effect, as stray bullets missing targets buts hitting the asteroids and doing nothing did not initially provide an engaging experience. Therefore, asteroids explode when shot based on a health variable attached to them in a C# script sub-component. For consistency, this was implemented for any object hitting asteroids. This means that should the player crash into an asteroid, the asteroid would explode. This proved a fun experience, and also allowed for added difficulty in that on explosion, the spaceship shakes itself side to side, rotating the player in the Nova device. For the obstacle course in Stage 4, this effect can be used to move the player in ways they may not expect or do themselves.

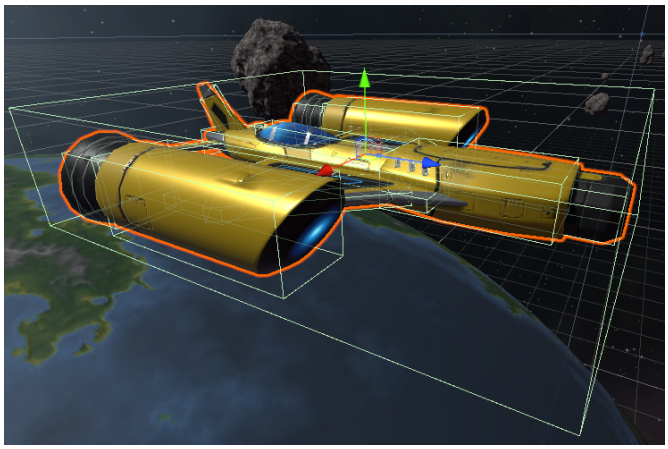


Fig. 9. An enemy ship in the Unity Editor, where the green wireframe cube is its collision detection.

D. Nova and VR Integration

VR integration was completed via Unity's OpenXR plugin [36]. Unity provides game templates when creating a project with basic setup for various use cases, one of those being VR with OpenXR. OpenXR is also an ideal VR plugin, due to its multi-platform support of VR devices [36]. The game template used provided just a "*XRRig*" component acting as the VR camera, and some directional light behind it so that the player can always see in front of them. This XRRig component has been moved into the ship visually in the scene editor, as well as placed as a sub-component of the spaceship game object, meaning that it will act as part of the ship and move along with it. During development, an Oculus Rift S [24] and HTC Vive Pro 2 [25] were used to test VR functionality, however the Nova device uses an HP Reverb G2 [26]. All three of these headsets are notable as each one runs on a different VR framework (Oculus [24], OpenXR [25], and Windows Mixed Reality (WMR) [26]). This quickly became an issue as it was unknown whether the game would work in a Nova device since it had only been tested with other frameworks directly supported by Unity. Near the end of development, an opportunity arose to test the game with an HP Reverb G2

[26], to which it was discovered the game did not support it by default. An extra WMR plugin exists for the OpenXR for Unity plugin, which allowed for use of the Reverb G2, meaning that the game is functional on the Nova device. Integration with the Nova device [21] was completed via the use of the Nova Software Development Kit (SDK) provided by Eight360 upon beginning the project implementation.

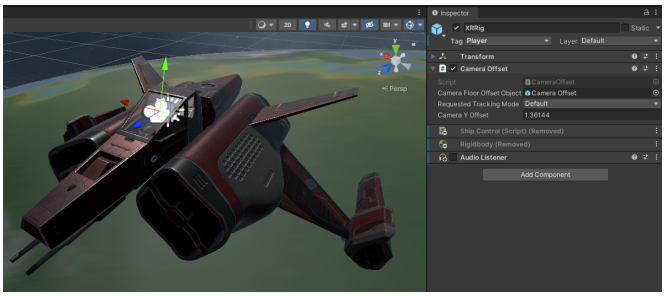


Fig. 10. The XRRig component and its placement in the player's spaceship.

Fig. 11. The Nova Emulator running locally, as provided by Eight360 [21]. This emulator was used to test the affect of the game in controlling the Nova.

The SDK is a Unity plugin which consists of two scripts; one to read rotations from an in-game rigidbody in the scene, and another to actually rotate the device accordingly. The SDK also came provided with a Nova emulator, which can simulate the movements of the device virtually to test whether the game is correctly connecting to and rotating the device. From the SDK files, the script "*NovaController.cs*" was added as a sub-component to the player's spaceship, along with a rigidbody component to handle physics simulations. This allowed the emulator to move, proving that the device was compatible during development and before testing on a real Nova device [21]. Once the game was ready enough to test on a real device, an issue was found in that the IP address and port for the device are hard-coded into the game as public fields in the Nova Controller component. This meant that to debug issues with the device or switch between use of the emulator and real device, the game had to be rebuilt entirely from the Unity editor. To resolve this, modifications were made to the pre-provided C# script to pass the IP and Port fields from a configuration file. This configuration file was stored in Unity's pre-defined "*StreamingAssets*" folder as this is persistent between build and editor versions and can easily be accessed and modified.

E. Player and Investigator Controls

Implementation of the flight stick controls was difficult due to the lack of accessibility to said controls, which could not be emulated the same as the Nova device. Therefore, this was one of the final implementations from the design before user testing began. The Nova device comes equipped with a Thrustmaster Hands-On Flight Stick (HOTAS) Warthog flight simulation controller [28], which features both a throttle and flight stick, where the user would hold the throttle in their left hand and the flight stick on the right.

Fig. 12. Player controls using the Thrustmaster Warthog [28]. This diagram was adapted from PB Tech's listing of the simulation controls [27].

According to the design of using bolted-down flight simulator controls, the throttle should handle forward and back spaceship movement and the flight stick should handle spaceship rotation. On the back of the HOTAS Warthog [28] is a trigger, which can also be used for the shooting system implementation. The input of the throttle and flight stick were implemented via the use of the Unity Input Manager [29], which can read controller input and translate it into a variable in the form of an 'axis', which is a scale reading where the controller is currently located. Once the Input Manager knows to read input from flight simulator controls (labelled as Joysticks in the Unity Editor), the value of the controller's axis can be read in code as "*Input.GetAxis("ControllerAxis")*" [29]. This is then used to translate the axis to movement, by multiplying the throttle's axis value by the maximum speed and adding it to the velocity of the spaceship's rigidbody component. This rigidbody component is monitored for increased acceleration by the Nova Controller script, and will move the

Nova device backward slightly to simulate acceleration of the vehicle. The spaceship object's rotation is modified by the flight stick, by multiplying Unity's default rotation axes by the horizontal and vertical axes input by the HOTAS Warthog [28], multiplied by the sensitivity setting which is used to control the slowing of the controls as specified in the study design for Stage 1. Joystick controls can be read from multiple joysticks at once in Unity, meaning that the HOTAS Warthog controls were mapped the same as those for a traditional video game controller [28]. Therefore for the investigator, a handheld controller is able to control every aspect of the game as a participant would. This adds a level of safety as if at any time a participant is uncomfortable in the position they are in and unable to control the ball themselves, the investigator would be able to help orient them.

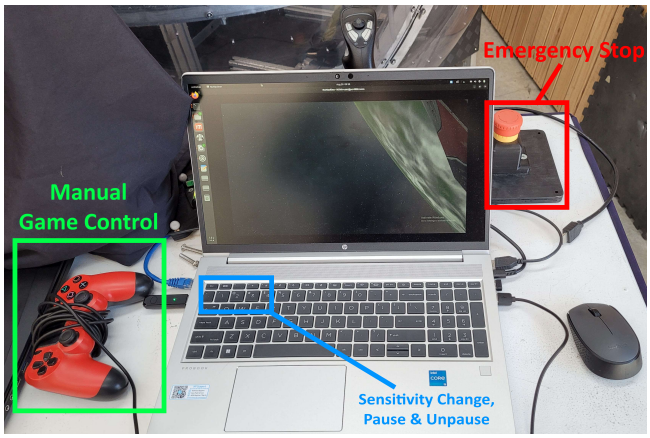


Fig. 13. The investigator controls for the Nova device.

F. Spaceship Dashboard

In development, the task proved more difficult than expected as the next checkpoint location is not always immediately obvious. If participants are unaware of the next checkpoint's location, there would be more variation in them searching for it by rotating to look around. To resolve this, a compass system was implemented on the spaceship's dashboard. The compass is its own game object which is a sub-component of the spaceship, powered by a C# script. The script identifies which checkpoint is currently active, and rotates the arrow to point toward that checkpoint (see Fig. 14).

G. Data Analysis and Tooling

As discussed for the game's configuration file, Unity has a designated folder "*StreamingAssets*" for storing persistent files between game builds and the editor. Therefore, this can not only be used to provide configuration for the game but also for outputs from the game that must be retrieved later, such as logging of the ship's location and rotation. This logging is done in the C# script "*ShipLogger.cs*" which is a sub-component of the spaceship. Logs are formatted as a JSON as the analysis tool is designed to be a JavaScript web application, meaning JSON files would be easily readable as JavaScript objects in-code,

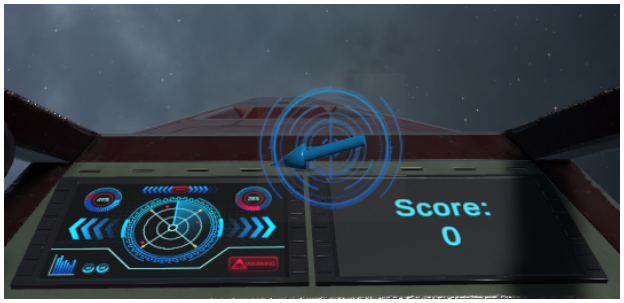


Fig. 14. The spaceship dashboard, featuring a score counter and the in-ship compass.

compared to CSV files which would require extra parsing on import. To ingest the output data, the developed analysis tool is a single-page web application, featuring a file chooser which once a file has been uploaded, instead displays graphs depicting the route of the player whose data was uploaded, and density plots on their rotation during the game (See Fig. 15). The page state is determined by a set of React states, detecting whether a file has been uploaded, processed, or is unable to be processed. Depending on these checks, either the upload screen, and error screen or the graphs will display. The graphs have been built using Plotly's JavaScript implementation [15], using a 3D scatter plot. Checkpoint markers have also been added via an array of X, Y and Z coordinates (See Fig. 15). Checkpoints were not added dynamically due to time constraints, and the fact that in this case the checkpoints will never move. In future work, it would be ideal for checkpoints to be uploaded to this tool dynamically so that it could be used for multiple courses or simulations.

Fig. 15. The two primary screens of the analysis web application; file upload and route view.

V. User Study

A user study was conducted to evaluate the symptoms experienced in the Nova device [21] under varying conditions suspected to correlate to feelings of motion sickness.

A. Study Design

As previous projects have already assessed differences in sickness between VR headset and Nova device usage [7], [8], this project primarily focused on whether differences

in types of motion in the Nova device affects the user's symptoms of motion sickness and to what degree. To collect data using the software produced, a within-subjects study was designed featuring 4 stages of gameplay in the Nova device, with one clearly defined variable changing between each, as defined in the game design in Section 3C. Before beginning, between each stage, and after completion, the investigator undertaking the study asked participants about their symptoms. The questionnaires used for this study were devised from the SSQ [1] and FastMS methods [2]. Since the participant will be in-game during questioning, questions must be fast and simple. Therefore, the FastMS method is ideal since it is proven, robust and requires only 1-2 questions on a 0-20 scale [2]. These questions should be "How sick do you feel?" and "How nauseous do you feel?". While this will provide adequate data on sickness levels at each stage, it does not directly evaluate the software's effectiveness. Therefore, an extra question "How immersed do you feel?" should be asked as well, to determine whether participants feel that the simulation is working. This should be consistent with the previous questions for familiarity, and thus use a 0-20 scale as well. As FastMS does not directly ask for symptoms [2] and only an estimate level of sickness, it would be ideal to ask additional symptom questions after participation is complete on symptoms and task effectiveness for evaluating the software. These questions were dervied from the SSQ [1], NASA TLX [5] and SUS [6] questionnaires.

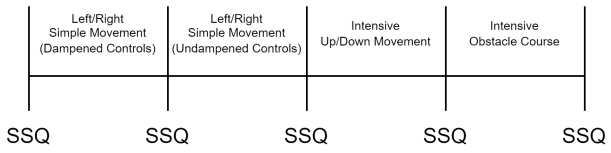


Fig. 16. The study design by stage with SSQ questions in-between.

As well as the sickness levels, due to the software logging spaceship translation and rotation over time, extra variables to be considered included the participant's amount of rotation, time to complete and whether they were able to complete the study or became so sick they had to stop. This data was then aggregated and analyzed for any noticeable correlations, such as a significant number of participants becoming sick due to a specific variable change in the simulation, whether those who rotated more than others saw more symptoms, or if those who were less familiar with VR saw greater symptoms.

B. Participants

Participation was offered via a sign up form, where participants were then to be randomly selected. In total, 10 participants signed up and 10 were required, meaning all participants who signed up were selected. In the sign up form, data was collected from each participant on their history of motion sickness, VR usage and whether they actively play video games. The participant criteria looked for any person who had no history of seizures, heart conditions or migraines which could be aggravated by the use of VR. The project aimed to run the experiment with some participants who had VR and video game experience, and some who did not, as well as some with a history of motion sickness on vehicles such as cars, planes, and boats, and those who had not experienced motion sickness in other settings. The 10 people who participated in the study met these specifications in that there were 7 male and 3 female participants, where 5 participants had experienced motion sickness in VR or vehicles prior, 4 had not and 1 was not sure (See Table 1). 5 participants had either used VR irregularly or regularly prior to participating, and 5 had only used it once or twice or not at all. In terms of experience with video games, 4 participants did not play video games and 6 did actively play video games, for at least 1 hour per week.

TABLE I
PARTICIPANTS BY ID, GENDER, HISTORY OF VR OR VEHICLE SICKNESS, AND PRIOR USE OF VR

ID	Gender	Sickness History	VR History
1	M	No	Yes, Once or Twice
2	M	Yes	Yes, Irregularly
3	M	No	Yes, Every week
4	F	No	Yes, Irregularly
5	M	Yes	Never used VR
6	M	No	Yes, Once or Twice
7	F	Unsure	Yes, Once or Twice
8	M	Yes	Yes, Once or Twice
9	M	Yes	Yes, Irregularly
10	F	Yes	Yes, Irregularly

C. Procedure

Each session lasted up to 1 hour. In this session, participants were brought into the Eight360 office [21] and sat in their staff room. Participants would then be given time to read the relevant paperwork before beginning and confirm that they fit the criteria via a series of verbal statements asked by the investigator. They were then informed about the game they are about to play, and what the objective is. The investigator then walked them to the Nova device and sat them inside with the VR headset on and the game open, but no physical movement. Participants then verbally completed the modified FastMS questionnaire [2] for the first time as a baseline for their current sickness levels in the headset, discounting any movement. After the questionnaire was completed, the game was started using the built-in pause and un-pause functionality and participants were left to complete the course. After each stage was completed, the game automatically paused so that the investigator could re-ask the FastMS questions [2]. On game completion, the questions were answered again for the final stage, and participants then completed a written questionnaire on their symptoms, featuring SSQ [1], NASA TLX [5] and SUS [6] questions. In total, each participant was asked the FastMS [2] inspired questionnaire 5 times, once before the game, three times during the game and once after the game.

VI. Results

A. Sickness and Symptoms

To evaluate whether the changing variable of one stage saw significant differences in sickness to the other stages, a Repeated-Measures ANOVA was completed on the data retrieved from the FastMS questionnaire [2], using the statistical software Jamovi [35]. The data was organized by question and stage where each stage contains 3 variables, sickness, nausea and immersion. Immersion was recorded in order to evaluate the software itself, whereas the sickness and immersion values can be used to determine the symptoms experienced throughout the experiment. Unfortunately from the 10 participants who undertook the experiment, 1 participant had a technical issue with the Nova which invalidated their data, and 3 participants did not complete the course as they were too sick. While this shows that symptoms of sickness do occur for some people, the nature of the ANOVA method of analyzing the data means that their results cannot be used, since they are incomplete. Therefore, the results produced were only produced using the data from the remaining 6 participants.

Within Subjects Effects

	Sum of Squares	df	Mean Square	F	p	η^2_p
Time	62.3	3	20.78	4.15	0.025	0.453
Residual	75.2	15	5.01			

Note. Type 3 Sums of Squares

Within Subjects Effects

	Sum of Squares	df	Mean Square	F	p	η^2_p
Time	114.8	3	38.28	7.49	0.003	0.600
Residual	76.7	15	5.11			

Note. Type 3 Sums of Squares

Fig. 17. The Repeated-Measures ANOVAs on feelings of sickness (top) and nausea (bottom) between stages, produced using Jamovi [35].

One Repeated-Measures ANOVA was completed on each FastMS [2] variable (sickness, nausea and immersion), using Jamovi [35] (See Fig. 17). The feeling of immersion has been excluded from the figures provided as it does not factor into motion sickness symptom levels. The sickness and nausea ANOVAs saw a significant P-value for the difference in each value between stages at 0.025 and 0.003. This means that at least one variable change between stages saw a significant difference in its affect on the participant's symptoms of both sickness and nausea. In the same ANOVAs, a partial eta squared test was run to quantify the size of the effect. A value above 0.14 is considered a large effect, and the sickness and nausea data saw values of 0.453 and 0.6 respectively, meaning the difference had a significant area of effect. Post-Hoc tests were conducted on the data to find any errors present in the data that would otherwise go unnoticed using only the ANOVAs, by comparing each stage to all stages that come after it. This produced some interesting results in that the levels of sickness were a false positive, and have no significant difference between stages. This is because no adjusted P-value under the Tukey Test for sickness saw a value below 0.05, the lowest being for between Stage 1 and Stage 2 at 0.176 (See Fig. 18).

Post Hoc Comparisons - Time

Comparison			Mean Difference	SE	df	t	p	P_{tukey}
Time		Time						
Stage 1	-	Stage 2	−2.833	1.138	5.00	−2.490	0.055	0.176
	-	Stage 3	−3.667	1.647	5.00	−2.227	0.076	0.235
	-	Stage 4	−4.167	1.778	5.00	−2.344	0.066	0.207
Stage 2	-	Stage 3	−0.833	0.833	5.00	−1.000	0.363	0.757
	-	Stage 4	−1.333	1.229	5.00	−1.085	0.328	0.713
Stage 3	-	Stage 4	−0.500	0.806	5.00	−0.620	0.562	0.921

Post Hoc Comparisons - Time

Comparison			Mean Difference	SE	df	t	P_{tukey}
Time		Time					
Stage 1	-	Stage 2	−4.500	0.922	5.00	−4.881	0.017
	-	Stage 3	−5.167	1.352	5.00	−3.822	0.044
	-	Stage 4	−5.333	1.687	5.00	−3.162	0.086
Stage 2	-	Stage 3	−0.667	1.054	5.00	−0.632	0.917
	-	Stage 4	−0.833	1.682	5.00	−0.496	0.957
Stage 3	-	Stage 4	−0.167	0.872	5.00	−0.191	0.997

Fig. 18. The Post-Hoc Tests run on the sickness data (top) and nausea data (bottom), produced using Jamovi [35].

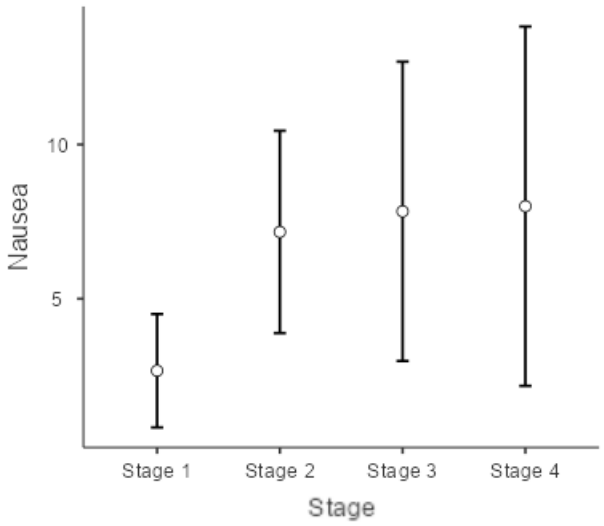


Fig. 19. The estimated marginal means of nausea values provided at each experiment stage, produced using Jamovi [35].

However, the same tests run on the nausea data show that there is a significant difference between Stage 1 and Stage 2, with an adjusted P-value of 0.017, which is below the significance boundary of 0.05 (See Fig. 19). This confirms that there is a clear and significant difference in each stage's effect on nausea, between Stage 1 and Stage 2. However, all other stages were too similar and saw no difference in the symptoms experienced. In terms of the changing variables between stages, this means that the raised sensitivity of the controls directly correlates to a more prominent feeling of nausea. By plotting the marginal means of the FastMS [2] results for nausea on a scale between 0-21, the significance

of Stage 1 compared to the remaining stages is clear (See Fig. 19). Participants saw far less nausea in Stage 1 with the lowered control sensitivity, than they did in further stages. It is important to note that there could be interference in that Stage 1 was first, and the time participants were in the device could be a factor in the final result.

TABLE II
PARTICIPANTS BY APPROXIMATE TIME, FILTERED BY PARTICIPANTS WHO COMPLETED THE COURSE

ID	Stage 1 Time	Full Course Time
1	12 minutes	26 minutes
3	5 minutes	23 minutes
4	4 minutes	14 minutes
5	4 minutes	18 minutes
6	16 minutes	55 minutes
10	4 minutes	16 minutes

However, the game has been designed to output participants' rotation and translation data with timestamps, and provides a metric on the times when each checkpoint and stage were reached. This data can be used to view the time spent in Stage 1 compared to other stages, and whether this is likely to be a factor in the result (See Table 2). Participants had vastly varying times depending on how they controlled the game and what parts of the task they prioritized. Some participants engaged with the shooting feature while others ignored it, and some participants went off course whereas others followed it strictly. As visible in Table 2, two participants saw a significantly higher time spent in Stage 1 at 12 and 16 minutes. While the sample size is small which could cause unexpected variance, this suggests that the time spent in VR was not affecting the symptom of nausea alongside the change in control sensitivity.

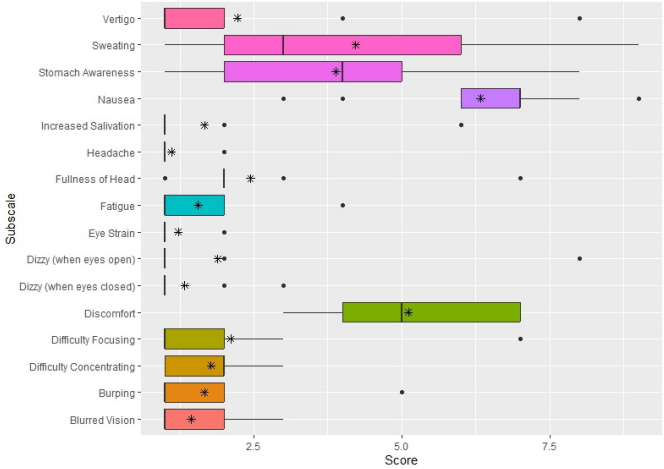


Fig. 20. SSQ sub-scale results across participants, produced using R [37].

The SSQ [1] questions asked after the experiment showed more accurate results on the symptoms felt across participants (See Fig. 20). Notably, the symptoms of salivation, headache, fullness of head, eye strain, and dizziness were spread widely between participants. While the mean for each of these symptoms is on the lower end of the scale, there are many outliers. The sample size of participants is likely influencing this, so more participants would be needed to gain an accurate result. The most common symptoms were nausea, stomach awareness and discomfort (See Fig. 20). Sweating, while having a low mean overall, appears to have a larger variance with many people experiencing it at varying levels. The highest scoring symptom overall was nausea, with just two outliers scoring this symptom lower than 5/10.

B. Software Usability

Using the NASA TLX [5] and SUS [6] results for all participants (excluding the one with technical issues who did not complete the task), the usability of the produced software and Nova device can be evaluated. The raw NASA TLX [5] data is scored between 0 and 20. The standard way of calculating this is to multiply it by 5 to reach a number out of 100 (See Table 3). Overall, participants found that completing the experiment saw a reasonable 45/100 in demand mentally, and 35/100 physically. The pacing for the checkpoint task scored a low 25/100 (asked as "How hurried/rushed was the task?" [5]), and most participants found the difficulty fair at 55/100 (See Table 3).

TABLE III
NASA TLX RATING AVERAGES

Factor	Mean (Raw)	Mean (Calculated)
Mental Demand	9	45
Physical Demand	7	35
Temporal Demand	5	25
Performance	13	65
Effort	11	55
Frustration	6	30

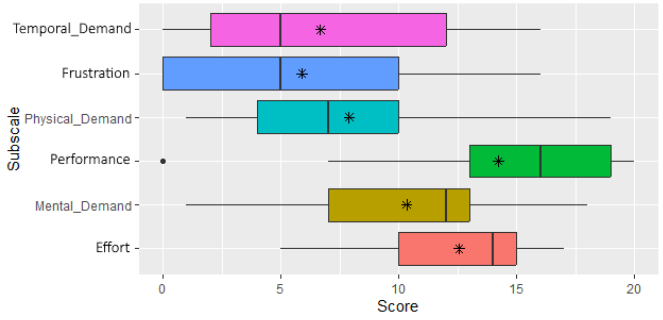


Fig. 21. Results of the NASA TLX scale across participants, produced using R [37].

The SUS usability score [6] can be derived by converting the "Strongly Disagree" to "Strongly Agree" statements into a scale from 1-5 (See Table 4). Few participants found the controls too complex with a mean score of 1.55/5. Most people found the Nova device very easy to use and learn how to use with scores of 4.09 and 4.18 (See Table 4). while confidence in one's ability to use the software scored a 3.91/5, a score of 2.55/5 was given for the software being difficult to use. Most

people stated they require the support of a technical person to use the Nova device [21], which makes sense as operation of the device required 2 people.

TABLE IV
SUS RATING AVERAGES ON A 1-5 SCALE, WHERE 1 = STRONGLY DISAGREE AND 5 = STRONGLY AGREE

Statement	Mean Score
I would use this simulation frequently	2.72
The controls were too complex	1.55
The device was easy to use	4.09
I need technical support for the device	3.45
The device functions were well integrated	4.00
There was too much inconsistency	2.00
Learning to use the device is easy	4.18
The software was awkward/difficult to use	2.55
I felt confident using the software	3.91
I need to learn more to use the software	2.64

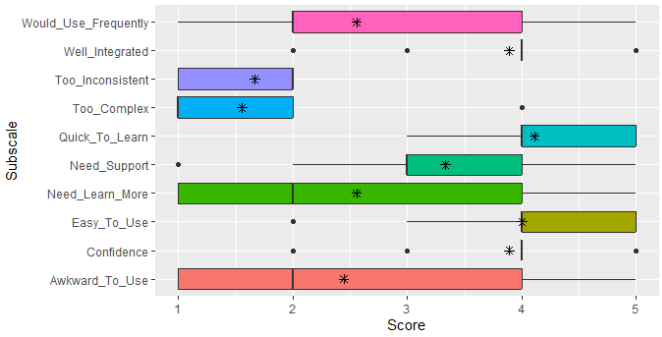


Fig. 22. Results of the SUS scale across participants, produced using R [37]. Note that the questions were answered from "Strongly Disagree" to "Strongly Agree" on a 5-point scale, which has been converted to a corresponding value from 1-5.

C. Discussion

The produced software met all 7 requirements. It is a playable VR experience in which the player controls the Nova device, presented as first-person spaceship flight (Requirements 1-3). Players also have the control to rotate themselves 360 degrees (Requirement 4). Experiences between participants were kept consistent via a checkpoint course task, during which the game logs the translation and rotation data of the player (Requirements 5 and 6). The game has been designed with future expansion in mind, with checkpoints being easily rearrangeable (Requirement 7). Participants were asked in the post-experiment questionnaire whether they enjoyed the experience, which resulted in an average score of 8/10. This is exceptional and proves that the game and device could draw more participants for future work, as most people find the experience very entertaining. In terms of purpose fit, most participants found the controls understandable (1.55/5 in complexity, see Table 4) and feel moderately confident using the simulation (3.91/5, see Table 4). The simulation was understandable by participants and was very usable. While the software achieved a level of consistency via a checkpoint course however, the course did have flaws in that participants varied in time significantly to complete the same section (see Table 2). While in some cases this is due to participants intentionally veering off course to explore the game, variations more than a minute per stage could definitely cause inconsistencies in the symptoms experienced. This is difficult to resolve with the requirement of allowing the player to control the experience, as the investigator has no power over their choices outside of encouraging them to complete the course. It is worth noting that while the user study was conducted well and produced good results, the sample size was very small in that only 10 participants took part, and only 6 could be included in the ANOVAs. In future work, it would be beneficial to confirm the results of the study with a larger pool of participants.

VII. CONCLUSIONS AND FUTURE WORK

Use of modern head-mounted VR equipment often causes symptoms of nausea and motion sickness for some people [1], [19]. This is thought to be due to the difference in movement when a user appears to move physically in the digital space, while not moving in real life [19]. The Nova device matches user's physical movements to what they see in VR, which has the potential to offset these feelings of motion sickness [21]. This project developed a bespoke software application for the purpose of assessing sickness symptoms in the Nova device [21], in the form of a spaceship flight simulation game where players traverse a set of checkpoints laid out strategically to assess varying levels of movement. A within-subjects user study was conducted using this software with 10 participants of varying backgrounds, in which a link was found between the speed of player-controlled movement and feelings of nausea (See Fig. 19). Other common symptoms included sweating, stomach awareness, and discomfort, however the study was unable to link these symptoms to a particular variable within the study (See Fig. 20).

The software was deemed fit for purpose in the study and scored well in the usability analysis (See Fig. 21 and Fig. 22). Overall, symptoms of sickness are still experienced in the Nova device [21], but can be dampened by slowing rotational movement (See Fig. 19). In future work, it would be ideal to include a larger sample of participants to confirm that the results shown generalize to a wider population. The study was also limited in that during the experiment, only feelings of general sickness and nausea were recorded at each stage. Using the newfound knowledge of the most common symptoms experienced in the Nova [21] (See Fig. 20), a future study could ask more specific questions in the FastMS [2] format, which could explain more about the primary cause of these symptoms. Also, this study was limited in that it initially planned to measure physiological data via health monitoring vests, which did not happen due to a lack of time and equipment. Future work could utilize health monitoring

equipment to gain more accurate insights into the symptoms. In the NASA TLX [5] scoring, participants also suggested that the task pacing was slightly too slow (See Fig. 21). In future, the checkpoint task system could be expanded upon or replaced with a more intensive task to maintain engagement. This could also alter the results in that people who are more engaged in the task may think less about their symptoms while completing the task, and should be investigated.

REFERENCES

[1] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal, "Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness," The International Journal of Aviation Psychology, vol. 3, no. 3, pp. 203–220, 1993. doi:10.1207/s15327108ijap0303_3

[2] B. Keshavarz and H. Hecht, "Validating an efficient method to quantify motion sickness," Human Factors: The Journal of the Human Factors and Ergonomics Society, vol. 53, no. 4, pp. 415–426, 2011. doi:10.1177/0018720811403736

[3] P. Bimberg, T. Weissker, and A. Kulik, "*On the usage of the simulator sickness questionnaire for Virtual Reality Research,*" 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2020. doi:10.1109/vrw50115.2020.00098

[4] S.A. Balk, M.A. Bertola and V.W. Inman, "*Simulator Sickness Questionnaire: Twenty Years Later*", Driving Assessment Conference, 2017, pp 257-263. doi: 10.17077/drivingassessment.1498.

[5] NASA, "*NASA Task Load Index*," https://humansystems.arc.nasa.gov/groups/tlx/downloads/TLXScale.pdf (accessed Aug. 28, 2023).

[6] Usability.gov, "*System usability scale (SUS),*" https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (accessed Aug. 28, 2023).

[7] C. Simmonds, "*Understanding the Effectiveness of 360 Motion in VR Simulators*," 2021.

[8] C. de Bruyn, "*Understanding Sickness and Emotional Experiences in Virtual Reality 360 Degree Motion Simulators*," 2022.

[9] UnknownWorlds, "*Subnautica*," Available: https://unknownworlds.com/subnautica/ (accessed Aug. 28, 2023).

[10] Hello Games, "*No Man's Sky*," https://www.nomanssky.com (accessed Aug. 28, 2023).

[11] H. Welsby, "*NMS Modding Discord Conversations*," https://gitlab.ecs.vuw.ac.nz/groups/coursework/project489/2023/welsbyharl/-/wikis/NMS-Modding-Discord-Converations (accessed Aug. 28, 2023).

[12] CheatEngine, "*CheatEngine,*" https://cheatengine.org/ (accessed Sep. 3, 2023).

[13] Unity, "*Unity.com*", Accessed 9 May, 2023. Available: https://unity.com/

[14] Epic Games, "*Unreal Engine*", https://www.unrealengine.com/en-US (accessed Aug. 30, 2023).

[15] Plotly, "*Plotly: Low-code data app development,*" https://plotly.com/ (accessed Sep. 3, 2023).

[16] Meta, "*Create react app,*" https://create-react-app.dev/ (accessed Sep. 3, 2023).

[17] Meta, "*Add or change your Facebook profile picture: Facebook help center,*" Add or change your Facebook profile picture — Facebook Help Center, https://www.facebook.com/help/163248423739693 (accessed Sep. 3, 2023).

[18] H. Oh and W. Son, "*Cybersickness and its severity arising from virtual reality content: A comprehensive study,*" Sensors, vol. 22, no. 4, p. 1314, 2022. doi:10.3390/s22041314

[19] Y. S. Kim, J. Won, S.-W. Jang, and J. Ko, "*Effects of cybersickness caused by head-mounted display–based virtual reality on physiological responses: Cross-sectional study,*" JMIR Serious Games, vol. 10, no. 4, 2022. doi:10.2196/37938

[20] D. Saredakis et al., "*Factors associated with virtual reality sickness in head-mounted displays: A systematic review and meta-analysis,*" Frontiers in Human Neuroscience, vol. 14, 2020. doi:10.3389/fnhum.2020.00096

[21] Eight360, "*Eight360 - NOVA Untethered Motion Simulator*," https://www.eight360.com/ (accessed Sep. 3, 2023).

[22] New Zealand Ministry of Business, Innovation and Employment (MBIE), "*Energy in New Zealand 2023*," https://www.mbie.govt.nz/dmsdocument/27344-energy-in-new-zealand-2023-pdf (accessed Sep. 3, 2023).

[23] Unity, "*Unity Asset Store,*" https://assetstore.unity.com/ (accessed Sep. 3, 2023).

[24] Meta, "*Oculus Rift S,*" https://www.oculus.com/rift-s/ (accessed Sep. 6, 2023).

[25] HTC, "*HTC Vive Pro 2 Overview*" https://www.vive.com/nz/product/vive-pro2/overview/ (accessed Sep. 6, 2023).

[26] Hewlett-Packard, "*HP Reverb G2 VR Headset*" https://www.hp.com/us-en/vr/reverb-g2-vr-headset.html (accessed Sep 6, 2023).

[27] PB Tech, "*Thrustmaster Hotas Warthog Joystick For PC, Official Replicas Of The Joystick, Throttle and Control Panel of the U.S. Air Force A10C Aircraft,*" https://www.pbtech.co.nz/product/GAMTTMH913771/Thrustmaster-Hotas-Warthog-Joystick-For-PC-Officia (accessed Sep. 7, 2023).

[28] Thrustmaster, "*HOTAS WARTHOG,*" https://www.thrustmaster.com/products/hotas-warthog/ (accessed Sep. 7, 2023).

[29] Unity, "*Unity Manual: Input Manager,*" https://docs.unity3d.com/Manual/class-InputManager.html (accessed Sep. 7, 2023).

[30] Ebal Studios, "*Hi-Rez Spaceships Creator Free Sample,*" https://assetstore.unity.com/packages/3d/vehicles/space/hi-rez-spaceships-creator-free-sample-153363 (accessed Sep. 8, 2023).

[31] Ebal Studios, "*Star Sparrow Modular Spaceship,*" https://assetstore.unity.com/packages/3d/vehicles/space/star-sparrow-modular-spaceship-73167 (accessed Sep. 8, 2023).

[32] A. Bielecki, "*Earth like Planets,*" https://assetstore.unity.com/packages/templates/packs/earth-like-planets-32790 (accessed Sep. 8, 2023).

[33] M. Dion, "*Asteroids Pack,*" https://assetstore.unity.com/packages/3d/environments/asteroids-pack-84988 (accessed Sep. 8, 2023).

[34] PULSAR BYTES, "*Starfield Skybox,*" https://assetstore.unity.com/packages/2d/textures-materials/sky/starfield-skybox-92717 (accessed Sep. 8, 2023).

[35] Jamovi, "*Jamovi - Open Statistical Software for the Desktop and Cloud,*" https://www.jamovi.org/ (accessed Sep. 23, 2023).

[36] Unity, "*OpenXR Plugin,*" https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.7/manual/index.html (accessed Sep. 27, 2023).

[37] The R Foundation, "*R: The R Project for Statistical Computing,*" https://www.r-project.org/ (accessed Oct. 1, 2023).