Universidade do Minho
Escola de Engenharia
Departamento de Informática

Alexandre Rzepecki Rodrigues

Exploring the Use of Serious Videogames in a Robotic Walker to Improve Ataxic Gait

January 2023

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Alexandre Rzepecki Rodrigues

Exploring the Use of Serious Videogames in a Robotic Walker to Improve Ataxic Gait

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Cristina Manuela Peixoto dos Santos

January 2023

ACKNOWLEDGEMENTS

My five-year academic journey ends today, six and a half years later.

First and foremost, I want to thank my parents, Monika and António, for being the most supportive people I have known. I may not always(never really) show it, but I love you and appreciate all you've done. And the rest of my family: my grandparents, my uncle and my cousins, who are prouder of me than I am of myself.

I want to thank my home boys, Pedro, Paulo, Rui and Bruno, for always being there for me right from the start. I hope we can keep this going for a lot longer.

The Academic Choir of the University of Minho, where I've had great experiences and met some amazing people. Thanks to them, I feel like I have really grown as an individual.

I am also grateful to my friends who wrote their theses at the same time as me. Knowing someone else is facing the same hardships as you really lightens the load.

Of course, I'd like to thank the folks over at the BiRDLAB, who have shown me great hospitality, and professor Cristina Peixoto for proposing the dissertation. And a special thanks to my "supervisor", João Lopes, whose guidance throughout this process was invaluable.

STATEMENT OF INTEGRITY

ABSTRACT

Ataxia is a neurological sign indicative of dysfunction in the cerebellum, a part of the brain that coordinates movement. The typical symptoms include lack of balance when walking or standing, loss of limb coordination, change in speech, and difficulty with fine motor tasks, strongly affecting the person's daily activities. Cerebellar ataxia can be inherited or caused by other disorders such as stroke, multiple sclerosis, or cerebral palsy. While there is no known cure for inherited ataxia, the symptoms can be managed through intensive and personalised rehabilitation, including physical, speech, and occupational therapy.

Physical therapy, also known as conventional therapy, is a standard practice in the rehabilitation of patients with ataxia. It focuses on performing different physical exercises repeatedly, where a therapist monitors the session and regulates the intensity. Although effective, the repetitiveness of conventional therapy can become monotonous, besides being a very time-consuming process, which can be cumbersome for the therapist. Without any additional stimulation and decreasing motivation, patients may experience stagnation or even drop out of therapy.

Robot-assisted Gait Training (RAGT) is being introduced in the rehabilitation of persons with motor disabilities. This technology allows personalised and intensity-adapted training, which could benefit the patient's recovery. However, this therapy can also become monotonous, so there is a need for more interactive and appealing strategies. A possible solution to this problem is the development of exergames (serious games) and their inclusion in robotic-assisted therapy. This can become advantageous since the robotic devices integrate several sensors that read the patient's movement and can be used as controllers in the game, allowing a more immersive experience.

Considering this, this dissertation proposes two serious videogames, which were integrated into a robotic walker, the WALKit Smart Walker, intended for gait ataxia rehabilitation. The first game is cognitive, whose goal is to react to a certain shape or sound as quickly as possible. With this game, it is intended to study the influence of dual tasking on motor rehabilitation. The second game is a dynamic one whose goal is to incite the patient's balance control. This game uses an algorithm for torso orientation estimation to control an avatar holding two buckets full of water. The main goal is to give patients biofeedback regarding their postural balance and to exercise their balance with specific events, or minigames, that encourage them to lean in a way that causes weight transfer.

Both games were validated with healthy volunteers, in terms of functionality and usability. The results allowed to conclude that both were fun to play and showed potential to aid rehabilitation. Moreover, the results emphasised the relevance of customisation, replayability, and aesthetics in serious games. Future work includes the thorough validation of both serious games with patients with cerebellar ataxia, to assess their effectiveness in rehabilitation along with WALKit Smart Walker.

KEYWORDS ataxia, balance, exergames, motivation, rehabilitation, serious games.

RESUMO

A ataxia é um sinal neurológico indicativo de disfunção no cerebelo, uma parte do cérebro que coordena o movimento. Os sintomas típicos incluem a falta de equilíbrio ao caminhar ou ficar de pé, perda de coordenação dos membros, alteração na fala e dificuldade em tarefas motoras finas, gravemente afetando as atividades diárias da pessoa. A ataxia cerebelar pode ser herdada ou causada por outros distúrbios, como acidentes vasculares cerebrais, esclerose múltipla ou paralisia cerebral. Embora não haja cura conhecida para a ataxia hereditária, os sintomas podem ser controlados através de reabilitação intensiva e personalizada, incluindo fisioterapia, terapia da fala e terapia ocupacional.

A fisioterapia, também conhecida como terapia convencional, é a norma na reabilitação de pacientes atáxicos. Foca-se em fazer diferentes exercícios físicos de forma repetida, juntamente com um terapeuta que regula a intensidade. Embora eficaz, a repetitividade da terapia convencional pode tornar-se monótona. Sem estímulo adicional e motivação decrescente, os pacientes podem sentir estagnação ou até mesmo abandar a terapia.

O treino de marcha assistida por robô está a ser introduzido na reabilitação de pessoas com deficiência motora. Esta tecnologia permite treino repetitivo e adaptado, o que pode beneficiar a recuperação do paciente. No entanto, esta terapia também se pode tornar monótona, pelo que são necessárias estratégias mais interativas e apelativas. Uma possível solução é o desenvolvimento de *exergames* (jogos sérios) e sua inclusão na terapia assistida por robôs. Pode ser vantajoso uma vez que os dispositivos robóticos integram vários sensores que leem o movimento do paciente e podem assim ser usados como controladores no jogo, permitindo uma experiência mais imersiva.

Esta dissertação propõe dois videojogos sérios, que foram integrados num andarilho robótico, o WALKit *Smart Walker*, criado para a reabilitação da ataxia de marcha. O primeiro jogo sério é um jogo cognitivo, cujo objetivo é reagir a uma determinada imagem ou som o mais rápido possível. Este jogo pretende estudar a influência do *dual tasking* na reabilitação motora. O segundo jogo é um jogo dinâmico, cujo objetivo é estimular o controlo do equilíbrio do paciente. Este jogo usa um algoritmo para estimar a orientação do tronco, para controlar um avatar que segura dois baldes cheios de água. O principal objetivo é caminhar sem entornar os baldes e, desta forma, dar aos pacientes *biofeedback* sobre seu equilíbrio postural e exercitar seu equilíbrio com eventos específicos, ou minijogos, que os estimulem a se inclinar de forma a causar transferência de peso.

Ambos os jogos foram validados com participantes saudáveis, em termos de funcionalidade e usabilidade. Os resultados permitiram concluir que ambos foram divertidos de jogar e mostraram potencial para ajudar na reabilitação. Os resultados enfatizaram a importância da personalização, *replay value* e estética em jogos sérios. O trabalho futuro inclui a validação exaustiva dos dois jogos sérios com pacientes com ataxia, de forma a avaliar a sua eficácia na reabilitação juntamente com o andarilho WALKit *Smart Walker*.

PALAVRAS-CHAVE ataxia, equilíbrio, exergames, jogos sérios, motivação, reabilitação.

CONTENTS

ACRONYMS

API Application Programming Interface. 24, 27, 28, 30, 33, 34, 38, 39, 41, 51

AR Augmented Reality. vi, 16, 17, 22

ASBGo Assistance and monitoring System Aid. 24

BB Balance Board. vi, 12–14, 21, 22, 25

BBS Berg Balance Scale. 14, 20

BDS Basic Dynamic Stabilometry. 14

CP Cerebral Palsy. 11, 13

CT Conventional Therapy. 1, 10, 20

DCD Developmental Coordination Disorder. 19

FPS Frames Per Second. 38

FRT Functional Reach Test. 11, 13

GUG Get Up and Go. 11

GUI Graphical User Interface. vi, 26, 27, 39

IMU Inertial Measurement Unit. viii, 26, 41, 66, 67, 71–73, 81

IO Input/Output. 66, 68

JS JavaScript. 29

JSON JavaScript Object Notation. 41, 53, 66

LBP Lower Back Pain. 15

LTS Long-term support. 31, 34

MCE Motor Control Exercise. 15

NPRS Numeric Pain Rating Scale. 15, 21

OS Operating System. 26–28, 39, 81

PSEQ Pain Self-Efficacy Questionnaire. 15, 21

PSFS Patient-Specific Functioning Scale. 15

QML Qt Modeling Language. 27, 29, 33, 39, 80–82

QoL Quality of Life. 1, 12, 14, 31, 78

QT Quick Toolkit. 26, 39

RAGT Robot-Assisted Gait Training. 1, 2
RGB Red Green Blue. 12
RGB-D RGB-Depth. vi, 2, 12, 21, 22
RMQ Roland Morris Questionnaire. 15
ROS Robot Operating System. 3, 26, 27, 30, 33, 34, 41, 66

SDS Sequential Dynamic Stabilometry. 14
SFX Sound Effects. 78
SG Serious Game. 2–6, 12–17, 19–24, 42, 51, 81
SUS System Usability Scale. vii, viii, 55, 56, 59, 73, 74, 80
SW Smart Walker. 2, 3

TCP Transmission Control Protocol. 39
TD Typically Developing. 19
TSK Tampa Scale for Kinesiophobia. 15
TUG Time Up and Go. 13, 14, 20

UE Unreal Engine. 32, 34
UI User Interface. 27, 31, 35, 59, 67, 68, 81
URF Ultrasonic Range Finder. 25

VR Virtual Reality. 16, 22

WS WebSocket. 40, 41, 46, 47, 53, 66

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This dissertation, entitled *Exploring the use of serious video games in a robotic walker to improve ataxic gait*, presents the work developed in the scope of the fifth year of the Integrated Masters in Informatics Engineering. This work was proposed by the Biomedical Robotic Devices Laboratory (BiRDLab) of the Center for Microelectromechanical Systems (CMEMS) of the University of Minho. The dissertation emerged from the WALKit Smart Walker project of BiRDLab, with the goal of developing serious video games within a practical clinical context to allow gait training of persons with balance disorders using a robotic walker. This first chapter presents the motivation and problem statement, followed by the outlined goals, research questions, and dissertation structure.

1.1 MOTIVATION AND PROBLEM STATEMENT

Cerebellar ataxia results from damage to the cerebellum, which is a part of the brain responsible for movement coordination [1]. Many symptoms of ataxia mimic those of being inebriated, such as slurred speech, stumbling, falling, and poor limb coordination [2]. This loss of balance and mobility can strongly affect the patient's Quality of Life (QoL), limiting their daily activities. Currently, there is no pharmacological or surgical solution that can restore the neurological damages, so motor rehabilitation through Conventional Therapy (CT) is vital to relieve symptoms and improve QoL [3, 4].

Conventional therapy consists of several strategies that encourage the breakdown of complex movements into simple single-joint exercises [5]. A therapist usually oversees and assists in performing these, repeating the process every session. These movements are usually complemented by visual and verbal cues, which aid the patient in maintaining a healthy walking step or stride length [6].

While physical therapy has proven to be effective [6, 7], patients can find themselves getting **less motivated, especially in the absence of clear feedback regarding their progress** [8]. This lack of motivation may influence the patient's recovery process, as they may become **less committed to the rehabilitative process**, or even **lose the will to continue** therapy.

The use of assistive and rehabilitation technologies, namely Robot-Assisted Gait Training (RAGT), is proposed as a complement to the work of physiotherapists [3, 9]. These technologies have shown their potential to decrease patients' dependence and increase their participation in therapy [10]. For instance, the literature encourages using robotic walkers and exoskeletons as means for rehabilitating persons with mobility disorders [11].

Walkers specifically have a high potential for cerebellar ataxia rehabilitation, as they increase the patient's base of support, giving them more confidence and balance, which allows for a more stable gait [12]. Repetitive and intensity-adapted exercises are crucial for the recovery of mobility disorders [3]. However, monotonous exercises or activities may reduce the patients' motivation in therapy [13]. Thus, **rehabilitation should be as interactive and appealing as possible.**

A possible solution to this problem is making the rehabilitative process fun and engaging through video games. These games are referred to as Serious Games (SGs) because their purpose is clinical rather than recreational. Despite having been researched for over two decades [14], SGs are still quite the novel and vast concept. These games present many variables that need to be considered, for example, the type of game, sensors used, and evaluation methods. Therefore, each game can vary wildly from one another. All these differences can make it hard to tell which game aspect is the best in rehabilitating a specific condition, which makes the creation of a new SG a challenge. Nevertheless, SGs make **therapy more engaging by capturing the patient's interest**, encouraging them to return for additional sessions. However, there are many challenges, namely: i) how to integrate these SGs with RAGT; ii) how these games should look like; iii) what feedback should they provide; and iv) what the goals/rewards would be.

This dissertation addresses these challenges by proposing two SG for WALKit Smart Walker (SW). WALKit is a personalised robotic walker proposed for the rehabilitation of patients with gait ataxia who exhibit loss of balance and poor limb coordination. The robotic walker is endowed with several sensors, namely RGB-Depth (RGB-D) cameras, load cells, inertial, and infrared sensors, which constantly analyse the user [15]. This data was used in the context of this dissertation to develop a serious game that may help ataxic patients regain balance and coordination.

A SG can benefit all parties involved in the rehabilitative process. For the patient, it is a **fun and engaging experience**, with instant feedback, which helps them understand if exercises get performed correctly. Additionally, a game can have scores, which make a patient's progress more apparent, motivating them to continue improving. For the supervising therapist, games offer all the data captured by the sensors, which they can use to re-adjust the game to the needs of each individual patient.

1.2 GOALS AND OBJECTIVES

Considering the statements of the previous section, the main goal of this dissertation was to design and develop a serious game that may improve the rehabilitation of patients with gait ataxia. This game was integrated into the walker's architecture by taking advantage of the existing framework and sensors. Additionally, another goal was to improve a cognitive SG and integrate it into the walker's architecture. In the future, this could help study the effect of multitasking on the rehabilitation of patients with ataxia. To achieve these goals, the following objectives (OB) were outlined:

OB. 1: Conduct a literature review of different serious games applied in the physical rehabilitation of persons with balance disorders, with attention to the following: i) the sensors used, ii) developed strategy, iii) patient feedback, and iv) game clinical evaluation.

OB. 2: Identify technical and clinical requirements for the development of the SG, brainstorming with technical and clinical personnel about the key aspects that should be in the game.

OB. 3: Design and conceptualise the SGs based on the literature review and key elements from Objective 2.

OB. 4: Develop the SG, using the best concept from Objective 3. The game should be implemented in the walker's architecture, based on Robot Operating System (ROS), using C++, Python or both.

OB. 5: Validate the developed game, considering experimental tests with end-users, to prove its functionality and effectiveness as a complement to balance rehabilitation..

1.3 RESEARCH QUESTIONS

The proposed objectives, namely the literature review, the survey of technical and clinical requirements, the design and conceptualisation of the serious video game, its implementation in the walker's architecture, and its validation with end-users, will allow answering the following research questions (RQ):

RQ. 1: *How can serious games be helpful in motor rehabilitation?* This research question is related to Objective 1.

RQ. 2: *How can serious games be included in WALKit Smart Walker?* This research question is related to Objectives 2 and 3.

RQ. 3: *How could the addition of SGs to WALKit improve the rehabilitation of ataxic patients?* This research question is related to Objective 5.

1.4 CONTRIBUTIONS

The main contributions of this dissertation are:

1. A literature review on the use of serious games in the field of balance rehabilitation, and medical rehabilitation as a whole.
2. An exploration of the challenges and complexity of developing serious games from scratch.
3. Two serious games built for possible improvement of gait ataxia, one cognitive game for studying the effects of multitasking and another dynamic game to control and exercise the patients' balance.
4. Several frameworks and tools for future game development on WALKit SW.

1.5 DISSERTATION OUTLINE

This document is organised into six chapters. A brief description of what is covered in each one is provided below:

- **Chapter 1** focuses on providing a contextualisation, motivation, and problem statement of this dissertation. The research questions are also addressed, along with the primary goals. Ultimately, it presents the paper structure in detail.
- **Chapter 2** presents the literature review on the use of SGs in the rehabilitation of patients with balance disorders. This chapter presents several studies that design serious games for rehabilitation, addressing their strategies, sensors used, and evaluation methods.
- **Chapter 3** covers the project's planning phase, which outlines the steps taken to develop the serious game. It includes a brief description of WALKit Smart Walker, the requirements, considered solutions, and justification for the chosen solution.
- **Chapter 4** describes the development of the Cognitive Game, including comparisons with the previous version, literature used, new game modes, the integration in WALKit's application, and results of its validation using a standard usability questionnaire.
- **Chapter 5** describes the design and development of the Dynamic Game, including its philosophy, key components, the integration in WALKit's application, and validation results and analysis.
- **Chapter 6** concludes the dissertation, summarising the work that was developed throughout this dissertation, briefly recapitulating the previous chapters, and answering the outlined research questions. Lastly, it concludes with suggestions for future research insights.

2

STATE OF THE ART

2.1 INTRODUCTION

Serious Gaming has been an ongoing field of study for over two decades [14, 16]. Most studies find that serious games improve motivation, to varying degrees [17, 18]. However, balance rehabilitation is a vast field of study and can apply to many conditions, from ataxia to old age. There are many differences between studies in literature, such as the metrics used to validate the game and assess the patient's performance, the choice of games and accessibility, and the sensors used to foster patient interaction with the game.

This chapter focuses on a detailed literature review that supports the main goals of this dissertation. First, it presents the relevance of SGs in rehabilitation. Second, it explains the research strategy, selection strategy, and data extraction. Lastly, the results are presented across four sections: i) the game types (section 2.5.1), ii) the sensors and strategies used (section 2.5.2), iii) the desirable properties of serious games (section 2.5.3), and iv) the validation strategy of the included studies (section 2.5.4).

2.2 SEARCH METHODOLOGY

This chapter presents a literature review of the use of serious games in the rehabilitation of balance disorders. The research was mostly conducted in the Scopus database, using keywords such as: **Dual Tasking**, **Therapy**, **Cognitive**, **Game**, **Rehabilitation**, **Balance**, **Posture**, and **Serious**. Logic operators "AND" and "OR" were used to combine these keywords. The search was limited to the articles' title, abstract, and keywords. A manual search was also conducted considering the bibliography of the selected articles. A total of four queries were performed, namely:

i) dual **AND** tasking **AND** therapy **AND** cognitive

ii) cognitive **AND** games **AND** rehabilitation **AND** (balance **OR** posture)

iii) serious **AND** games **AND** (posture **OR** gait **OR** balance) **AND** rehabilitation

iv) (serious **OR** health) **AND** games **AND** (balance **OR** posture) **AND** rehabilitation

2.3 SELECTION STRATEGY

The articles were included if they met the following criteria: i) written in English, ii) conducted after 2016, iii) related to balance, iv) full document available, and v) related to serious games. Additionally, the research included not only cerebellar ataxia but also other balance disorders, as their rehabilitative processes share similarities.

2.4 DATA EXTRACTION

Each article was evaluated using the following criteria: i) Game Types: Whether they are custom or commercial, ii) the type of mobility disorder (if applicable), iii) the controllers used in the game, and iv) the forms of validation and main conclusions.

2.5 RESULTS

From the methodology, and after deleting duplicate articles and studies that did not fit in the inclusion criteria, 19 articles were included in this literature review. Different sensors create different SGs, each with unique design strategies. According to the results, there are two types of SGs in the field of rehabilitative therapy: the custom games, made from scratch for rehabilitation, and the commercial games, made for healthy persons for entertainment and then later adapted for therapy.

Tables 1, 2, and 3 present the articles included in this literature review, addressing: i) the goal of the study, ii) sensors used, iii) how the game plays, and iv) the authors' main conclusions.

Table 1: Studies obtained through the literature review (part 1)

Study(year)	Goal	Participants	Sensors	Gameplay	Conclusions
Subramanian et al. (2020) [19]	How older and younger people feel about serious games	12 Healthy Young 10 Healthy Older	Kinect	Celestial Shower - The goal is to stretch and tilt the body to catch falling stars	(i) Younger participants want additional motivation from game. (ii) Older participants enjoyed knowing the health benefits. (iii) Both wanted a clear feedback system.
Moyà-Alcover et al. (2019) [20]	Create a serious game development framework. Create a background extraction algorithm.	Not Applicable (review)	Kinect	Wipe the screen with hand	(i) Better patient response when the games visuals are costumizable to the users liking. (ii) Game adaptability is import to fit to each patient. (iii) Data should be save in easily accessible formats.
Hung et al. (2017) [21]	Commercial vs Custom Games, relating to their effects.	43 Patients who had hemiplegic stroke	(i) Wii BB (ii) Tetrax BB	Tilt a virtual board through body tilt	Both games showed comparable improvements to CASI and BBS scores. However, commercial games showed better gain retainment.
Bonney et al. (2017) [22]	To compare the benefits of playing the same game repeatedly versus playing various games each session.	111 Children with DCD	Wii BB	Lean left or right to control the character.	(i) Variance is marginally better than Repetition. (ii) Some variance should always be included in SGs to reduce possible monotony.
Kachmar et al. (2021) [23]	Comparison between Commercial Wii Games vs Custom Games for balance rehabilitation	25 Children with CP	Wii BB	Lean left or right to control the game avatar	Custom game players showed significant improvements to both TCMS and DBT metrics.

Table 2: Studies obtained through the literature review (part 2)

Study(year)	Goal	Participants	Sensors	Gameplay	Conclusions
Hemmi et al. (2020) [24]	Lower limb rehabilitation with a projection based stepping game	19 Healthy	RGB Camera	Physically step on frog projected on the floor.	RGB cameras are fairly accurate for stepping games, and can even be set up at home.
Deutsch et al. (2019) [25]	Comparison between custom projection based game and kinect games	15 In the chronic phase post stroke	Kinect	Physically step on target projected on the floor.	Participants prefer the custom game due to its better costumization over the commercial one.
Matheve et al. (2018) [26]	Balance Rehabilitation through the use of back attatched controllers	10 Chronic nonspecific LBP and an underlying motor control impairment.	3 Inertial Sensors	(i)Lean left or right to control avatar (ii)Tilt body to tilt the in-game board	(i)The games received positive feedback on verious qualitative metrics. (ii)The setup process was described as tedious and de-motivating.
Amiri et al. (2018) [27]	Lower limb rehabilitation through projected serious games	Not clinically tested	Kinect	Physically step on target projected on the floor.	(i) Projecting games on the floor provides great feedback, as opposed to watching a screen. (ii) Games should be fine tuned to each patient
Bonnechère et al. (2017) [28]	Balance Rehabilitation through the use of RGB-d controlled games.	10 Children with CP	(i) Kinect (ii) Wii BB	(i) Tilt body to tilt game objects (ii) Wipe the screen	(i) Commercial games can be too physically demanding for some individuals. (ii) Their ease of setup and entertaiment helped with at-home compliance.
Soares et al. (2016) [29]	Balance rehabilitation in elderly persons through MR games	19 Older(65+) persons with Frailty Syndrome	Kinect	Control avatar with body Touch or avoid objects by moving limbs	(i) Significant improvements to TUG and FRT. Moderate improvements to upper and lower limb strength. (ii) Playable at home.
Terigi et al. (2015) [30]	Improved Control System through the use of a webcam	13 Persons with CP	RGB Camera	Place hand in in-game boxes	Good qualitative feedback. The technology can be used in many SGs as a controller.

Table 3: Studies obtained through the literature review (part 3)

Study(year)	Goal	Participants	Sensors	Gameplay	Conclusions
Lozano-Quilis et al. (2014) [31]	Balance Rehabilitation with full body Kinect use	11 Persons with MS	Kinect	Touch or drag in-game ball by placing hands on it	(i) Reported higher motivation. (ii) Improvements to BBS, weight transfer and tug metrics.
Wang et al. (2020) [32]	Balance rehabilitation through AR-based SGs	5 Patients with PD (Hoehn and Yahr stages I–III)	AR Headset	Touch or avoid objects by stepping on or over them	(i) AR induced fatigue in some patients. (ii) Stride length was slightly increased.
Charbonneau et al. (2017) [33]	Gait rehabilitation through VR and Avatar Embodying	10 Healthy Participants	RGB Camera	Step on tanks while walking on a treadmill	Avatar embodying showed positive results, as players underestimated time played.
Chen et al. (2021) [34]	Post stroke rehabilitation with the kinect and at-home sessions	30 Individuals with chronic stroke	Kinect	Reach leg forward in one of 8 cardinal directions	SGs faciliate telerehabilitation by recording detailed metrics or replays.
Ozdogar et al. (2020) [35]	Commercial video games on arm and cognitive rehabilitation for persons with MS	60 Persons With MS	Kinect	Multiple games from Kinect Sports Rivals	(i) Video-games improved with depression and working memory. (ii) CT was better at self-reported walking fatigue and quality of life.
Noveletto et al. (2018) [36]	Balance Rehabilitation	6 Hemiparetic Stroke Patients	Tilt BB	Attempt to either keep balanced or tilt towards a desired direction	Improvements to BBS, TUG, and overall stability.

2.5.1 *Game Types*

One of the first choices when deciding to use video games for balance rehabilitation is whether to adapt an existing game or create a new one. According to tables 1, 2, and 3, some studies presented in the literature adapted commercial games for therapy, while others developed new, custom games specially made for it. Both options come present advantages and disadvantages regarding, for instance, price, penalisation, and development time. The following subsections will detail both commercial and custom games presented in the literature, critically discussing their advantages and disadvantages.

2.5.1.1 *Commercial Games*

Commercial games are games that have been adapted for rehabilitation, usually exergames. These games involve some degree of physical activity, which is an important aspect of physical rehabilitation.

Made for the general consumer, they are usually affordable and easy to set up, even for home use. *Game polish* is the process of refining a game to make it more polished and consumer-ready, as defined in [37]. This process includes making the game more aesthetically pleasing through the use of shaders, better models, an intuitive user interface, and effective level design, among many other techniques.

All these steps mean that this process is also very time-consuming; it requires a large team of dedicated developers with a matching budget. Commercial exergames are often multi-million-dollar-budget games developed by big publishers. One example is Microsoft and its line of Kinect games, shown in Figure 1. As illustrated, the game is full of visual details, complex models, and sound design.

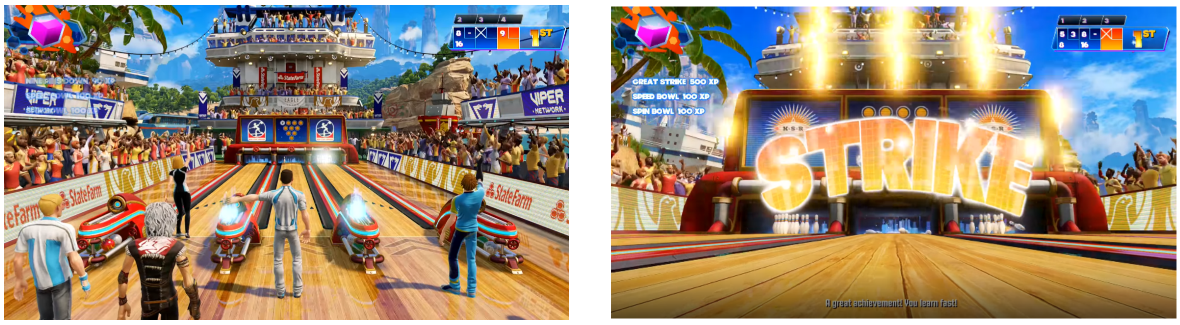


Figure 1: Instances of the Kinect Sports Rivals Bowling Game. Retrieved from the following video.

The main focus of these games is entertainment, which makes them highly motivating to play. However, they target the average consumer, with gameplay involving exercises such as jumps and fast movements. Examples include the studies [38, 39, 40].

These exercises are strenuous for someone with a balance disorders to perform [28]. Nevertheless, this does not mean commercial games have no place in rehabilitation. For instance, the authors of [41] consider commercial games to be beneficial, but as a complement to CT. In this study, 26 post-stroke patients performed 70 minutes of CT coupled with 20 minutes of playing mixed mini-games from several *Microsoft Kinect* games. These patients presented improved results over those who performed only the CT. The study reported an increase in the Tinetti gait test, the Get Up and Go (GUG) test, and the Functional Reach Test (FRT), leading to the conclusion that together with traditional therapy, Kinect can be considered an effective tool for improving balance and postural control.

2.5.1.2 *Custom Games*

Custom games are developed solely for rehabilitative purposes. Ranging from all-encompassing conditions such as ataxia to specific ones, e.g., Cerebral Palsy (CP) [23, 28, 30], the exercises are tailored to the condition to maximise rehabilitation and minimise pain. They are made with customisation in mind [20], which allows the therapist to adjust the game to better suit the patient's needs, unlike commercial games.

As noted in studies [36] and [20], custom games allow the therapist to review sessions later and evaluate the patient's performance more precisely than an arbitrary score, commonly seen in commercial games.

The largest drawback of custom games is often their lack of game polish. Throughout the literature review process, it became apparent that these games looked very different from their commercial counterparts. This problem likely stems from the fact that most custom games are developed for a specific study, often by a small team or even a single individual.

In [24], a simple game is projected on the floor. A camera reads the player's foot position. The goal is to step on the frog, as seen in Figure 2. The game is functional but basic-looking: static images, repeating images, and solid colour backgrounds.

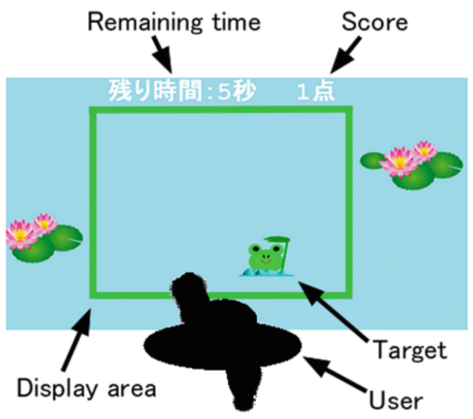


Figure 2: Frog stepping game. Retrieved from [24].

On the other hand, there are also Custom Games developed by larger entities whose goal is rehabilitation as well as profit. With a bigger budget, they are capable of developing more complete games. [42] These games are typically packaged with proprietary hardware, making them more expensive than their commercial counterparts; their target market are the medical institutions rather than the average consumer. Lastly, these devices may not be as straightforward to set up, at home or otherwise, as evidenced by [36], where patients felt less motivated having to go through a tedious setup process.

2.5.2 *Sensors and Strategies*

Sensors are physical devices that interface with, in this case, the game. Output sensors generally refer to speakers and monitors, so they are not as relevant to this study. Input sensors capture user information and can take many forms. From the literature review (see Table 1), these include cameras, inertial systems, and Balance Boards (BBs). The most used sensor in the literature is the RGB-D camera, as nine analysed studies used this type of sensor to create their SGs [19, 20, 25, 27, 28, 29, 31, 34, 35]. It is followed by the BB [21, 22, 23, 28, 36], with five studies, and Inertial Systems, with two studies [26, 36].

2.5.2.1 *RGB-D Cameras*

RGB-D cameras combine Red Green Blue (RGB) colour information with per pixel Depth. Despite using the same type of sensor, the games made with these can vary wildly. Some cameras also have built-in frameworks that facilitate their use.

The most popular of these is the *Microsof Kinect*. Originally made for at-home use with commercial games, it later became popular in the scientific field as a reliable and cheaper RGB-D camera [43]. Additionally, the Kinect contains a built-in framework, with algorithms such as skeleton detection and hand tracking, ready to use with minimal setup. In [44], authors used skeleton detection to create a game where an avatar mimics the player's movements by tracking the relative position of certain joints. For example, leg raises count each time the knee crosses a certain height threshold. Other exercises included: hip adduction, abduction, straight jumps, one-leg stances, hamstring curls, and squats. However, this paper studied the viability of Kinect in SGs with healthy persons rather than assessing if it effectively improved the QoL of persons with balance disorders. The authors concluded that the Kinect v1.0 sometimes had deviations between the predicted skeleton and the actual user's body. Still, the authors consider that Kinect v2.0 may improve this drawback point. Figure 3 displays the skeleton rig, as captured by a Kinect camera.

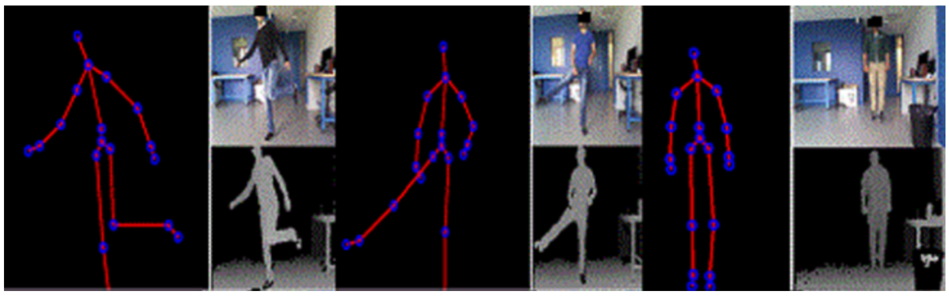


Figure 3: Skeleton rig (joints in blue), as captured by a Kinect RGB-D camera. Retrieved from [44].

In [29], the authors used a Kinect for balance rehabilitation in elderly persons in a study that included 24 participants of ages 65 and up. The equipment consisted of a Kinect, a computer, a screen projector, and a safety harness. Notably, the projector could be swapped for a monitor, reducing cost in exchange for a less immersive experience. The goal was to improve patients' balance and overall QoL by having patients control an avatar through body motion. The gameplay revolved around touching or avoiding different coloured objects for a set duration. Each interaction was accompanied by sound for better feedback. The results showed significant improvements in the Time Up and Go (TUG) and FRT tests, leading the authors to conclude that Kinect-driven motion-based SGs appear effective at balance rehabilitation. Figure 4 illustrates an example of the Kinect-driven game proposed in [29].

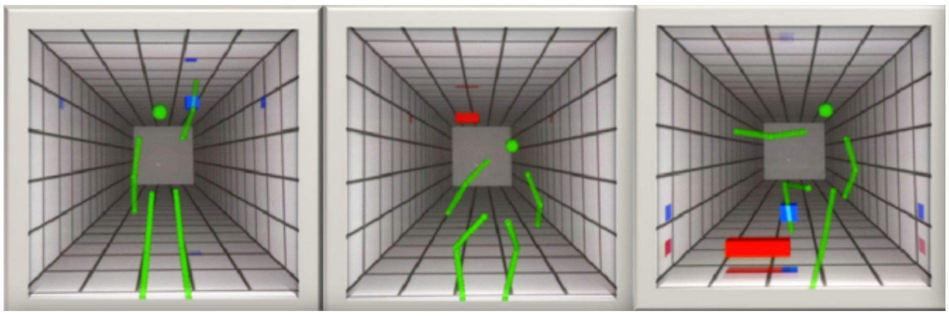


Figure 4: Kinect-driven game, where the avatar mimics a player's body motion. Retrieved from [29].

2.5.2.2 *Balance Boards*

The literature also highlights balance boards as another sensor for balance rehabilitation. These sensors read the weight distribution or tilt of the person standing on them, both useful metrics for perceiving balance or lack thereof. From the literature review, studies [21, 22, 23, 28] used them as the basis of their SG.

In [22], the commercially available *Wii Balance Board* was used in a study conducted with 20 children with CP. Throughout ten sessions, lasting 20 minutes each, the children played the commercial game *Nintendo Wii Fit*, where the player mimics positions shown on the screen. Figure 5, illustrates a healthy person playing Nintendo Wii Fit. In it, the person is mimicking the position shown on screen. Additionally, the game also highlights which body part is being exercised.

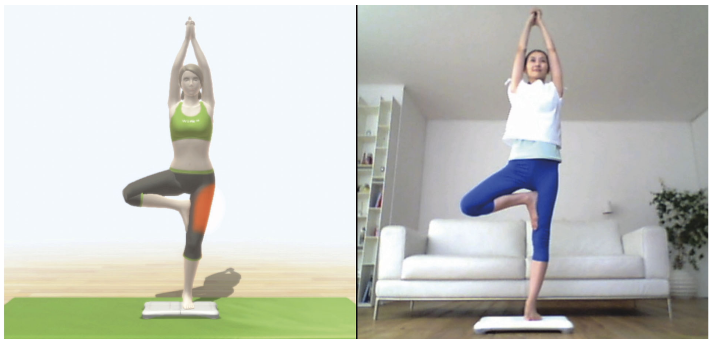


Figure 5: A healthy person playing Nintendo Wii Fit. Retrieved from DualSHOCKERS.

Another BB variant can be found in [36], where the focus is on tilt rather than weight distribution. In this article, the sensor consists of a circular wooden board fixed to a semicircular base. It has a tilt limit of 11 degrees, and patients are strapped to a harness during its use. While standing on the board, players attempt to keep the tilt at 0 degrees. Figure 6 displays the tilt-based BB and the respective system used to devise the SG.

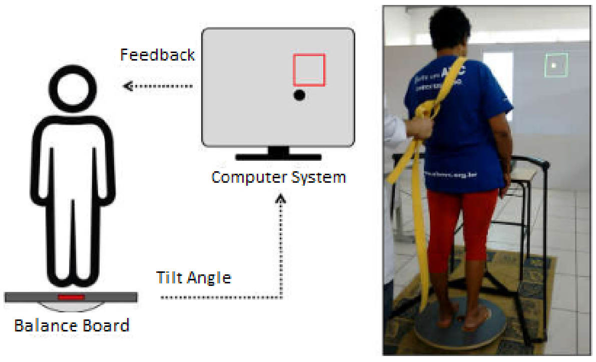


Figure 6: Tilt-based BB, and its system. Retrieved from [36].

The game is the visual representation of the player's tilt. If the player leans to the left, a yellow dot moves accordingly. The distance travelled indicates degrees. Additionally, a bounding box indicates when they have gone too far. The game is personalised to the user, as the therapist can configure the box size according to the patient's performance. This bounding box can be seen in Figures 7a and 7b.

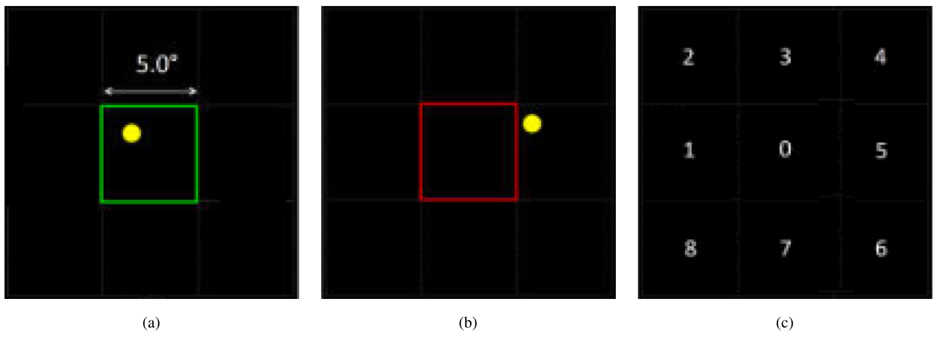


Figure 7: Game made possible with the Tilt BB; **(a)** Bounding box with a 5.0 degree limit; **(b)** Player (yellow dot) going farther than 5 degrees; **(c)** Possible box placements. Retrieved from [36].

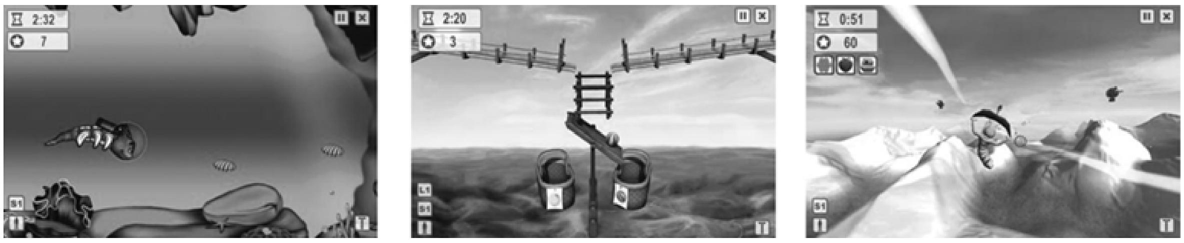
The authors of [36] reported an increase on all measured metrics, namely Berg Balance Scale (BBS) with a balance improvement of 12.1 $\pm$ 7.8%, and the functional mobility assessed by the TUG test showed improvements of 15.1 $\pm$ 7.4%. Patients also found their QoL perception had increased. Finally, a near perfect correlation between game score and Basic Dynamic Stabilometry (BDS) and Sequential Dynamic Stabilometry (SDS) tests was found, indicating that some game metrics can be as viable as evaluation methods.

Despite its frequency in the SG literature, this sensor is not as popular in clinics or hospitals, partly due to its elevated price. Furthermore, it is also impossible to use under dynamic conditions, i.e., it can not assess a patient's walking balance, which is an important component of rehabilitation.

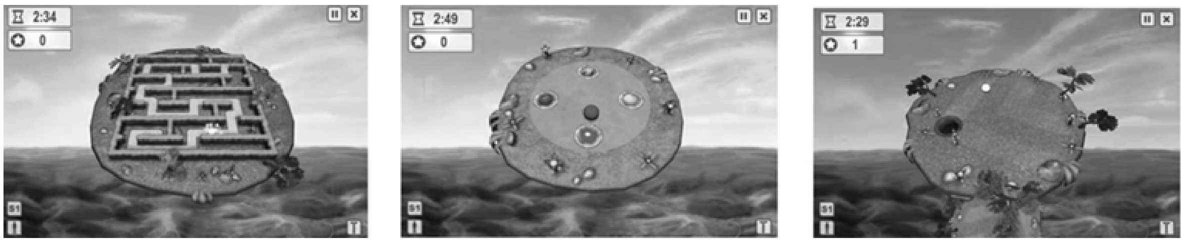
2.5.2.3 *Inertial Systems*

Another popular sensor in SGs for balance rehabilitation is inertial systems. They output angular velocity and acceleration and some can also sense the magnetic field intensity. An algorithm can use this information to obtain the user's position and, by using multiple of these, extrapolate the limb orientation.

In [26], ten patients with chronic nonspecific Lower Back Pain (LBP) were outfitted with three inertial sensors, one for calibration and two for controlling the games, positioned at the S1 and L1 vertebrae. Therapy sessions included two parts. First, patients played commercial games with the inertial sensors as a controller by moving on one or two planes of motion, as shown in Figure 8.



(a)

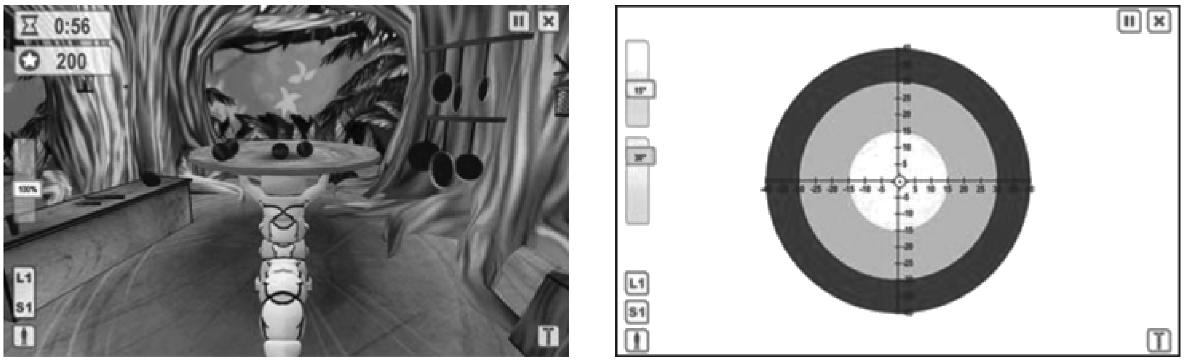


(b)

Figure 8: Examples of games controlled with inertial sensors. **(a)** Games controlled with a single plane of motion. **(b)** Games controlled with two planes of motion. Retrieved from [26].

The custom SGs were the primary drivers of rehabilitation. These games were made specifically for the inertial sensors, and throughout the gameplay session, they provide feedback to the user. For example, Figure 9 illustrates a game where the player has to hold a tray of coconuts by attempting to stay physically balanced. Furthermore, the games are adjustable, i.e., the therapist can configure how much real-life tilt translates to in-game tilt, allowing wider or narrower movements. Sessions lasted for around 45 minutes. On certain weeks patients took the sensors home and played three SGs and three Motor Control Exercises (MCEs). However, the authors found that, while helpful to the patients' rehabilitation, the process of setting up was too lengthy and demotivating.

Study [26] gathered only qualitative feedback, with overall positive results on Numeric Pain Rating Scale (NPRS), Pain Self-Efficacy Questionnaire (PSEQ), Patient-Specific Functioning Scale (PSFS), Roland Morris Questionnaire (RMQ), patient satisfaction with the treatment, and Tampa Scale for Kinesiophobia (TSK).



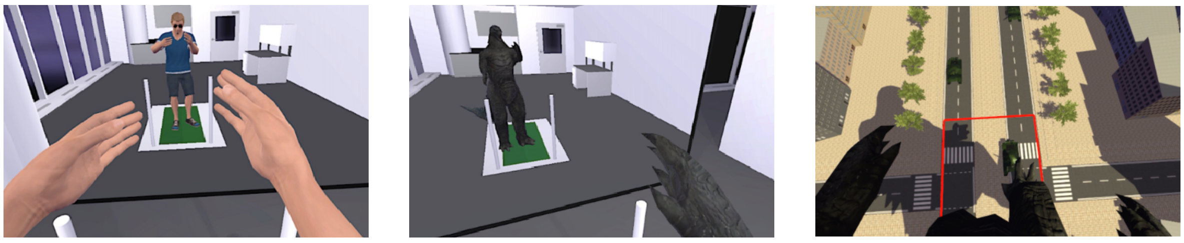
(a) (b)

Figure 9: Examples of custom-made inertial sensor games with feedback. **(a)** Coconut balancing game; **(b)** Dartboard game. Retrieved from [26].

2.5.2.4 *Virtual and Augmented Reality Headsets*

Another way to implement SG includes virtual or augmented reality glasses. Recently, Virtual Reality (VR) and Augmented Reality (AR) have become quite popular; researchers are starting to explore their applications in the medical field. These glasses, or goggles, have two main components. The first is a screen in front of each eye, and the second is an inertial sensor that tracks head orientation. The difference between VR and AR is that VR completely isolates the user from the real world by covering their eyes and ears. AR places objects on top of the world.

In [33], patients used a VR headset, along with a treadmill. Inside the game, the player avatar is a giant lizard monster whose goal is to crush as many tanks as possible by walking over them, as seen in Figure 10c.



(a) (b) (c)

Figure 10: Monster game; **(a)** Person avatar in a mirror; **(b)** Monster avatar in a mirror; **(c)** First Person view of the monster game. Retrieved from [33].

The research objective was to determine whether having a more immersive avatar would lead to more engaging play. Participants tried two avatars: a regular person, and the monster. When asked about their playtime estimates, participants replied, on average: +3% for the calibration phase with no avatar, -14% with the person, and -31% with the monster. That led to the conclusion that participants had more fun, on average, playing as the monster, which led to a more engaging experience, therefore, more motivation.

AR can be very powerful: placing objects on the floor, highlighting paths, and other things generally only achievable through projectors, which need certain conditions to operate, namely a dimly lit environment and a suitable projection surface.

In [32], the authors presented a SG with AR headsets, in which users are led around the room in a circuit, gaining points whenever they step the gold idols and losing when they touch a hole, as displayed in Figure 11. Additionally, it is possible to increase the difficulty by making the path longer and more winding.

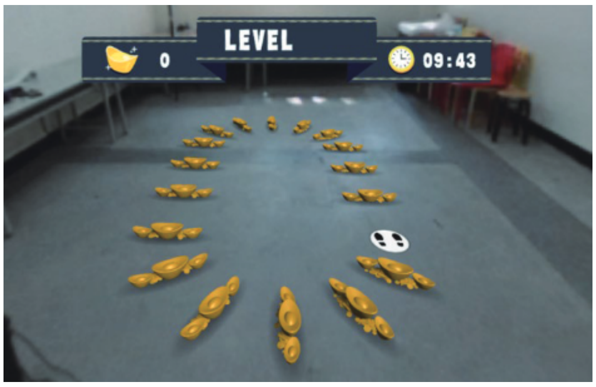


Figure 11: AR Game. Retrieved from [32].

Despite these qualities, there are downsides to these headsets, such as the inability to detect trunk position or balance. Thus, they are more suited as a complement to the previously mentioned sensors, such as the RGB-D cameras or balance boards.

2.5.2.5 *Miscellaneous Devices*

The four sensors mentioned so far were the most common among all the papers examined. However, the less common sensors/devices are still worth mentioning, especially the games they made possible. These are a full-body robotic exoskeleton [45], robotic orthoses [46], and a platform with buttons [47]. While some of these are not sensors themselves, they employ several sensors.

The first one is a full-body robotic exoskeleton, the Lokomat, designed for persons suffering from lower limb disabilities. It can support from 0 to 100% of body weight. Additionally, this exoskeleton allows the level of support provided to the patient to be adjusted: if the support level is at 100%, the exoskeleton imposes its reference trajectory; if the support level is at 0%, then the patient has complete control over the exoskeleton [45].

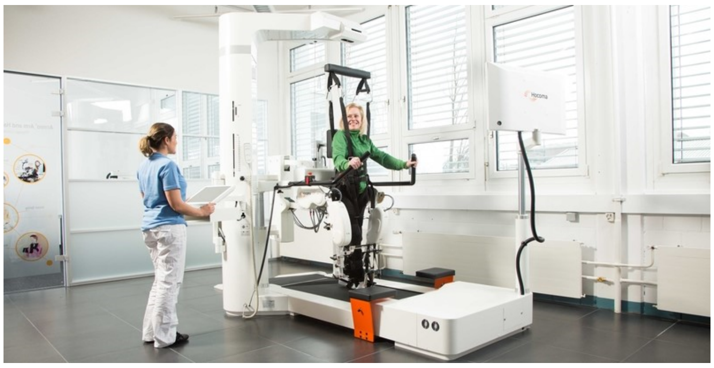


Figure 12: The Lokomat setup. Retrieved from hocoma.com.

In [45], the authors presented a game made for children to play with the Lokomat. According to their research, children need an extra stimulus beyond the benefits of rehabilitation itself [48]. The game is an endless runner, as seen in Figure 13, where the player must step over the obstacles in the dock. The avatar attempts to mimic the player's movement and gait speed to provide better immersion.

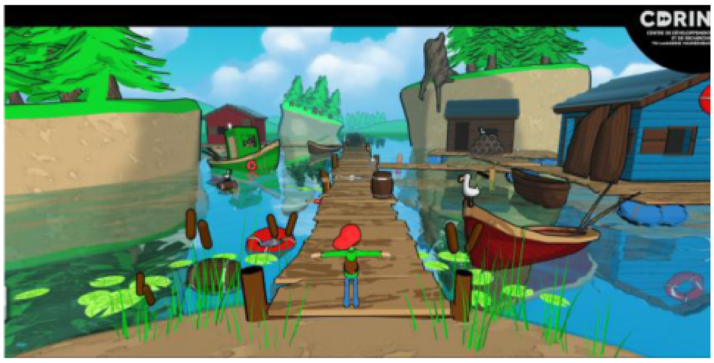


Figure 13: A game included in the Lokomat. Retrieved from [45].

The last sensor is the *Dividat Senso*, a platform with five buttons to be stepped on. It includes a series of games made specifically for balance rehabilitation. Figure 14 shows the *Divitat*, with its built-in monitor and safety railing.

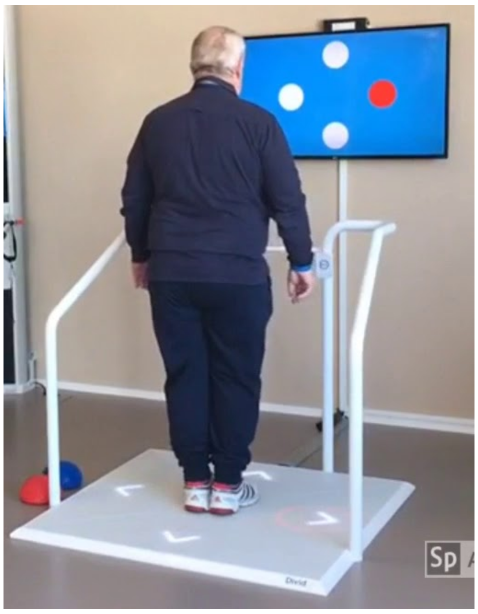


Figure 14: Dividat Senso. Retrieved from the following video.

In [47], 40 participants played the built-in games. The difficulty automatically adjusts according to the players' performance, i.e. the games speed up for quicker players and slow down for slower ones. The purpose of these adjustments is to provide a challenging, yet not too frustrating, game experience.

Participants performed 15-minute sessions where they played multiple games. The device provided visual, auditory, and somatosensory cues and feedback while playing the games. Participants described the game as positive, enjoyable, and motivating. The authors reported improvements in gait speed, sense of balance (with one participant no longer needing walking aids), flexibility, and overall physical fitness.

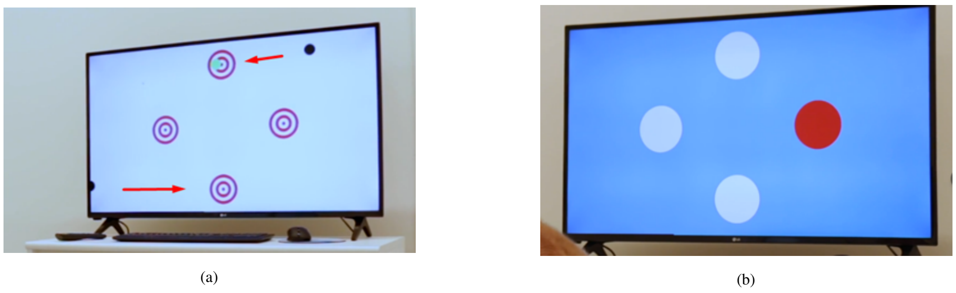


Figure 15: Two pre-packaged Dividat Senso Games; **(a)** Press button when ball is over circle; **(b)** Press the red button. Retrieved from [47].

2.5.3 *Desirable Properties of a Serious Game*

This subsection focuses on the practises that make SGs a viable form of therapy. These include design choices, such as bundling multiple games into one and game progression. Additionally, it presents tools that can be incorporated into the SGs to assist the therapist or research team.

2.5.3.1 *Variance or Repetition*

Variance means playing several games, while repetition focuses on playing the same game. In [22], the authors study which one is best for rehabilitative purposes. The trial lasted five weeks and had 111 child participants: 57 had Developmental Coordination Disorder (DCD), while the remaining 54 were Typically Developing (TD). Each group got further split: i) one half played a single game repeatedly (a ski slalom game with two difficulty levels), and ii) the other played several random games from ten possible options.

The skiing game is illustrated in Figure 16a. The main goal is to lean in the desired direction, which moves the character accordingly, as seen in Figure16b. The complete list of random games can be accessed in the following annex.

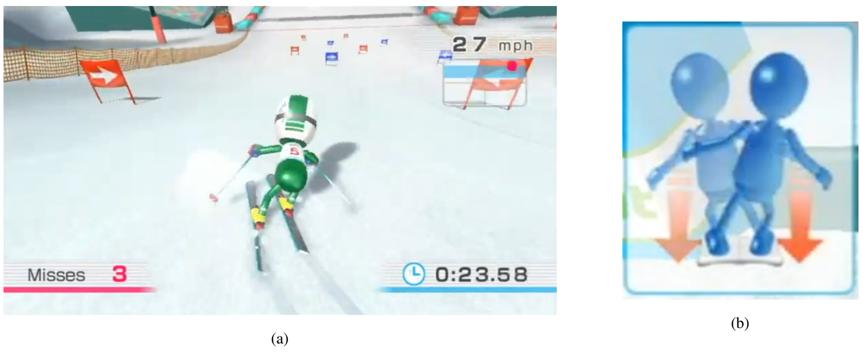


Figure 16: **(a)** Wii Fit - Balance Games - Ski Slalom; **(b)** Controls for the Ski Slalom game. Retrieved from the following video.

The authors found no significant difference between variance and repetition. They argue that, even within the repetition of the slalom game, there is still variance occurring each time, either from players approaching the same challenge in different ways or attempting to improve their scores.

2.5.3.2 *Sense of progression*

Motivation is one of the main objectives of SGs. Some people get motivated by the fact that they are improving their condition, while others feel like they need an extra incentive to therapy. In [19], the authors compared the opinions of two age groups after playing a SG. Participants were given basic information about the game's goal and control scheme before playing. Participants played the SG for 20 minutes, after a 2-minute warm-up.

The young majority (ages 23-28) felt that the game lacked a sense of reward or progression. Alternatively, the older party (ages 65-85) was generally satisfied with the underlying health benefits. Participants agreed that it was important for the game's difficulty to adapt to the player's skill level: too easy would make the game monotonous, and too hard would make it frustrating.

Additionally, they felt there should be a way to keep track of progress. One example was a profile for saving scores, which would benefit the patient and therapist. Lastly, some argued that the game lacked customisation. Being able to unlock aesthetic changes, such as different avatars, gives players a long-term goal to work towards, resulting in a more motivating experience. This sort of reward system is commonplace in commercial video games.

2.5.4 *Validation*

Validation has two main categories: quantitative and qualitative. Quantitative analysis pertains to the clinical outcomes of rehabilitation, i.e., whether a game can improve the person's balance condition. On the other hand, qualitative validation analyses patients' opinions during or after the trial.

2.5.4.1 *Quantitative*

Quantitative validation is crucial to any article regarding balance rehabilitation. It shows whether the game was able to improve the person's balance. First, patients are usually divided into two homogeneous groups: control and test [22], if possible. The control group is the reference point, usually performing CT. The test group plays the SG(s) that are being evaluated. Comparing the two groups reveals how the game performs against CT.

Patients are evaluated with a series of tests at one or more points throughout the trial, at which point studies in the literature diverge. Different articles use different metrics for validation. The most popular ones are the TUG [34, 36] and the BBS [36, 49].

The literature review done in [50] examine multiple articles regarding balance rehabilitation. The authors found that the metrics used in each one were so different that comparison was not possible [51]. For instance, an article might have considered a game a success based on TUG metrics, while another used BBS. However, although it is not possible to make a direct comparison, it is possible to see whether, in fact, serious games are important in rehabilitation.

2.5.4.2 *Qualitative*

Motivation is an intangible quality. The only way to assess it is by questioning the participants and extrapolating from their answers.

SGs generally receive favourable feedback. For instance, in [47], players reported feelings of well-being, being positively challenged, and a change of pace. In another study [26], patients reported positive feelings on several scales and questionnaires, such as NPRS, PSEQ. Lastly, in study [52], participants showed interest in continuing the trial, and professionals considered the game satisfactory. This study coupled that with quantitative feedback as well.

2.6 DISCUSSION

The main goal of this dissertation is to design and develop a serious game to assist with the rehabilitation of ataxic individuals. The literature review shows there are design choices to be made before development begins. The following section compares these and argues which ones best suit the project's goal.

In the literature, there were two types of games: commercial and custom. This dissertation's goal is to develop a serious game, which means creating a custom game. As such, the discussion is not about which is best but what aspects to take from each one.

When comparing commercial to custom games, custom games appear to be superior because they offer better tools for the therapist themselves to work with their patients. With full control of the variables in a game, a therapist can adjust it much better than, e.g., trying to choose the most appropriate level from a list.

Custom games offer customisation, detailed metrics, adjusted exercises, and tools for the therapist. This core concept should be in the planned game. Commercial games have better visuals, simple-to-understand metrics (scores), more gameplay, and rewards. The goal would be to implement these in the custom game, although realistically, not everything can be done, much less in the time allocated by this dissertation. As such, gameplay should be the top priority since the highest priority is motivation.

As for sensors, one can argue that RGB-D cameras, if sensitive enough, are ideal since they can detect the entire body and translate it into an avatar for better feedback for the player. The second best would be the inertial sensors, as these can more accurately interpret the position of a person's torso, allowing for greater granularity when configuring a game. Lastly, the BB, these only analyse weight placed on them, regardless of the cause, i.e., in what way is the person leaning and how are they standing. They leave much to interpretation and can make giving feedback more challenging.

The authors of [36] found that some participants felt unmotivated by the tedious setup process, which affected the rest of the session. That means that the process should be as streamlined as possible. For example, in commercial games, the entire process is just connecting the sensor to the console.

A game should also include some variance in its gameplay loop. For example, obstacles could appear in different places every time. This simple feature can make the game much less monotonous.

Additionally, a rewards system could encourage better play and, thus, better engagement in therapy. This system would not only incentivize a player to perform to the best of their abilities (without causing pain or stress) but also provide long-term goals, e.g., rewards that are only attainable through multiple play sessions.

Regarding game validation, quantitative and qualitative evaluations are not mutually exclusive; they can even complement each other. Qualitative feedback is ideal for gauging engagement and motivation with a SG. The disadvantage is the difficulty in comparing answers from different participants. Different people will inherently have different standards, experiences, and backgrounds, all of which affect their perception. Individuals feeling the same amount of pain could have different answers when asked to classify it on a scale from 1 to 10. This type of feedback is still valid; if most participants report the game as **"fun"** or "**engaging**" then it is safe to assume that it is. Quantitative feedback, on the other hand, shows concrete evidence of whether a game did or did not help further a patient's rehabilitation. The challenge with this method is deciding which metrics to track and understanding when a change is statistically significant or not. In summary, quantitative feedback indicates the game's therapeutic effectiveness, while qualitative feedback suggests how motivating it is to play, but both are equally important.

2.7 GENERAL CONCLUSIONS

For the past two decades, serious games have been a focus of therapy research. From the literature, the two main factors of SGs are: commercial or custom, which sensors to use, and how to validate them.

Commercial games are made for healthy individuals and can be adapted for rehabilitation. They are usually more fun to play, with the downside of not being as useful for the therapist in terms of customisation and saving metrics. Moreover, they are generally affordable and playable at home. On the other hand, custom games are developed to treat a specific condition. Contrary to commercial games, these usually look worse but offer better tools for the therapist. They also tend to have better results overall, at the cost of needing to be built from scratch. The majority of these are not playable at home since they may be hard to set up or require heavy or expensive pieces of hardware.

Regarding the sensors used, most of the literature involves four sensors (RGB-D cameras, BBs, inertial sensors, and AR/VR headsets), and the type of sensors dictates what sort of game can be developed. RGB-D cameras can translate the whole person (or less) into a game avatar, BBs can detect the balance distribution of a person, inertial sensors can read the torso or limbs' position and orientation, and VR or AR headsets are used together with other sensors to provide a more immersive experience by replacing the person's view with a game.

There are many ways to make a game more appealing. With extensive personalisation, the therapist can tailor the experience to the patient's needs. Gameplay variance also plays a vital role in reducing monotony. Lastly, providing the player with a sense of progression through either rewards or high scores can help keep patients engaged in the therapy for a longer time.

Lastly, SGs have two possible validation methods: quantitative and qualitative. The former compares several metrics taken throughout the therapy to confirm whether or not the patient progressed. The latter concerns to the patient's opinion on therapy and is equally important. The two are complementary, so any rehabilitative process, with or without SGs, should employ both if possible.

3

MATERIALS AND METHODS

3.1 INTRODUCTION

Before the games' development could begin, a small study had to be conducted, presented in this chapter in two steps. The first step is to understand the walker's full capabilities. The sensors dictate how the game controls, and the architecture dictates any programming languages or software choices during development. The second step is to explain the process behind the choice of the game development framework by analysing the most popular options in three distinct fields: i) low-level Application Programming Interfaces (APIs), ii) scripting languages, and iii) game engines. This framework will dictate how development progresses for most of the dissertation, thus choosing the most appropriate one is vital.

3.2 WALKIT SMART WALKER

WALKit Smart Walker is a robotic walker designed to assist with the physical rehabilitation of persons with balance disorders, namely cerebellar ataxia. The walker can support an entire person's weight for those with severe ataxia while ensuring proper posture, thanks to its design.

As a robotic walker, patients can control it with the handlebars. The therapist can configure the speed and turn angle through the embedded software with a comprehensive frontend. Its sensors and actuators work together to provide both an intuitive drive and continuous status monitoring. The walker has several algorithms that take advantage of these sensors: pose estimation, visual biofeedback of gait and posture, and balance analysis.

WALKit is the successor of another robotic walker, the Assistance and monitoring System Aid (ASBGo), from the same research department. This previous walker also had a prototype SG.

This dissertation's goal is twofold: create a new SG and optimise the prototype, totalling two games developed. Lastly, while the focus is on ataxia rehabilitation, the walker is also suitable for several balance-affected persons, including the elderly.

3.2.1 *Sensors*

The walker is fitted with multiple state-of-the-art sensors. Figure 17 illustrates the sensors and actuators that equip the device.

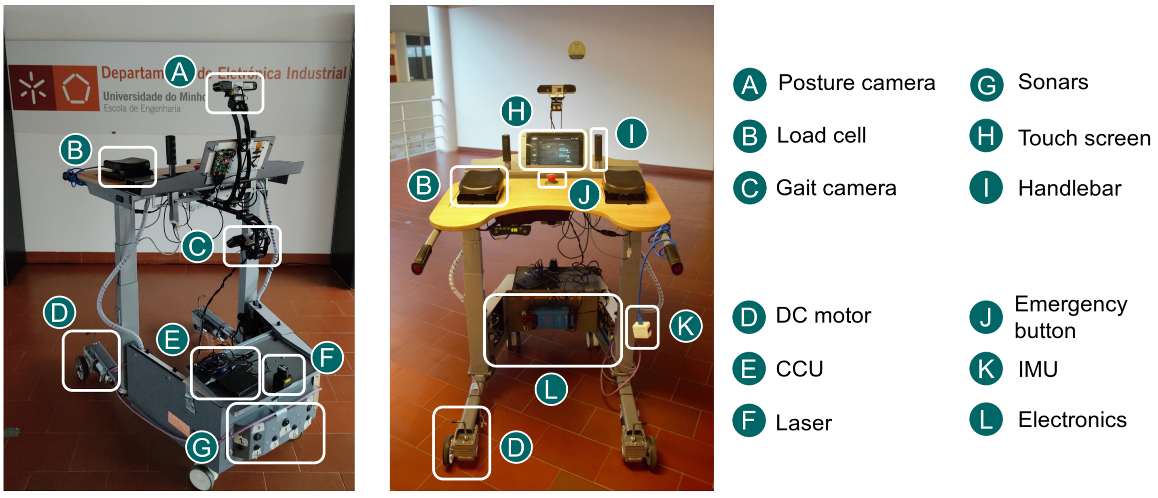


Figure 17: Detailed overview of all the WALKit components.

It has three sections. First, the input sensors read the user's information: tilt, weight, or even the whole user (image sensors, e.g., camera). Second, the control devices: motors and encoders. The encoders read angular velocity and can be used to deduce the walker's linear velocity. Finally, the electronics compartment, Wherein lies the batteries, drivers, and controllers. The main controller is a computer with the Ubuntu 18.04 operating system. Table 4 presents the sensors that compose WALKit Smart Walker.

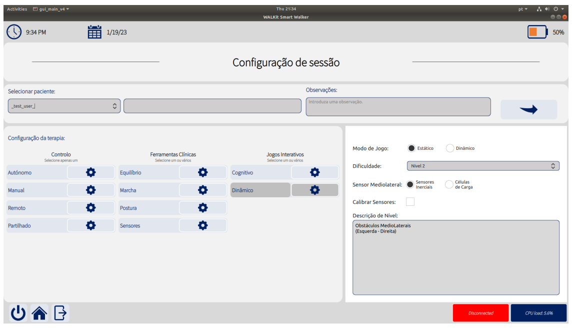
Table 4: WALKit sensors and additional information

Sensor Type	Sensor	Number	Measures	Range	Units
RGB	Orbbec Astra	2	3-channel image to assess gait and torso's posture.	0 – 255	8-bit Color
Depth	Orbbec Astra	2	Distance between the image plane and the object	0.6 – 8	m
Infrared	Sharp GP2Y0A21	1	User distance from the walker.	0.1 – 0.8	m
Load Cells	CZL635	2	Load the user is applying on the walker.	0 – 50	kg
Acceleration	MPU-6050	1	Torso acceleration in all axes.	$\pm$ 8	g
Gyroscope	MPU-6050	1	Torso angular velocity in all axes.	$\pm$ 2000	deg/s
Ultrasonic Range Finder	MB1210 XL-MaxSonar-EZ1	9	Frontal walker distance from obstacles.	0 – 7.25	m

Most sensors in Table 4, aside from the Ultrasonic Range Finder (URF) and load cells, could be found across games from the literature. The load cells act similar to a BB, measuring weight distribution. In WALKit, there are two load cells, one for each arm, that the patient leans on. The difference in weight between the two gets used to assess the user's posture.

3.2.2 *Architecture*

As previously mentioned, the main controller is a computer running the Ubuntu 18.04 Operating System (OS), connected to a touchscreen with an optional mouse and keyboard. The robotic walker includes an application, written in C++ and Quick Toolkit (QT), that interacts with the user and therapist, with different therapy configurations. This application also interfaces with the sensors, database, and algorithms. The program starts automatically when the walker gets powered on, hiding the underlying OS for a seamless experience. Figure 18 illustrates two examples of the applications' Graphical User Interface (GUI): the first is the configuration page, and the second is the runtime page, in which users and therapists can access all functionalities in real-time.



Selecionar paciente: _test_user_

Modo de Jogo: [✓ Estático] Dinâmico

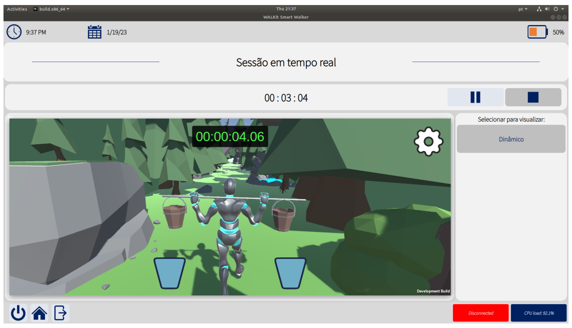
Dificuldade: Nível 2

Sensor Mediolateral: [✓ Sensores Inerciais] Células de Carga

Calibrar Sensores: ☐

Descrição de Nível: Obstáculos MedioLaterais (Esquerda - Direita)

(a)



(b)

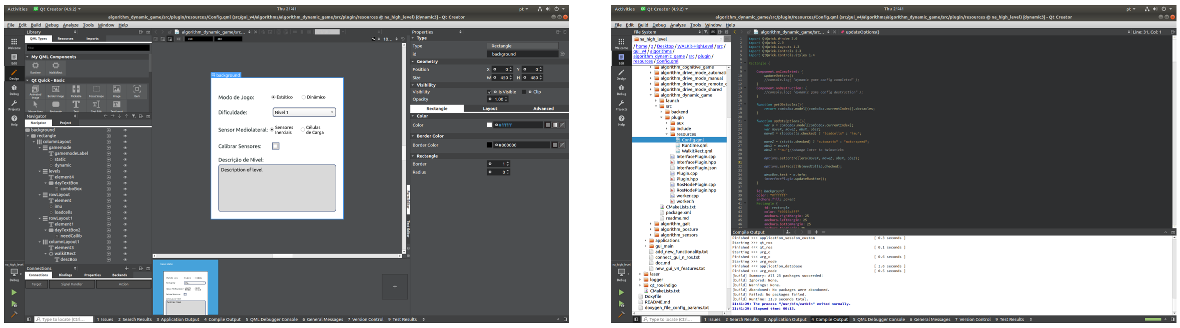
Figure 18: WALKit GUI: **(a)** Configuration Page and **(b)** Runtime Page.

Therapy with the walker involves four steps, as follows. First, the patient's profile should be selected, followed by any desired algorithm(s). Secondly, the therapist can start/stop the session by pressing the Play/Stop button. Halting a session will save various metrics to a database, under the patient's profile, for posterior review.

"Algorithm" is the internal name for the programs integrated into the walker's software. For instance, the "balance" algorithm calibrates, converts, and monitors the Inertial Measurement Unit (IMU) sensor. In turn, other algorithms can use this information through topics and services of ROS. The design philosophy is that each one is self-contained, and the therapist decides which ones are best suited considering the patient's needs. All parameters are configured directly on the application without having to open or edit any external files. Lastly, once a session starts, all selected algorithms launch at once. Each one can be hidden or shown by the user as needed in the runtime page.

There are three distinct categories. The first is the "Controls" and it dictates how the walker is physically controlled, including manual, autonomous, and remote control. The second category is "Clinical Tools". This category comprises algorithms that allow gait, posture, and balance analysis, as well as configure several different sensors. Lastly is the "Serious Games" category, where both games will go, as well as any future games.

Every algorithm has two parts, the frontend, and the backend. The former includes everything that the user can see and interact with. Contextually, this is the configuration page, which allows a user to further configure an algorithm to the patient's needs, and the runtime page, which handles all information displayed during an active session. It is written in Qt Modeling Language (QML), a language built on top of JavaScript. However, the bulk of the work occurs in QTCreator, a program made for the QML workflow, including a visual editor for the design of User Interface (UI) and a code editor for specific actions.



(a) (b)

Figure 19: The QTCreator program: **(a)** GUI editor and **(b)** Traditional Code Editor.

The backend handles the logic and data storage. In this case, it is written in the C++ low-level language. As such, it has the potential to be very efficient, especially with access to multi-threading frameworks. That is ideal for the backend, where some algorithms may do several thousands of calculations every second, sometimes concurrently. Besides, there are two main APIs that provide extra functionality. First is the QML's API: it offers functionalities like accessing information from the QML executable and communication with the frontend. Second is the WALKit's own API, which provides algorithm-specific utility. That includes access to patient profile information (only not confidential data), database, and, most importantly, sensor information, effectively a wrapper between the backend and ROS.

ROS is a free and open-source robotics middleware suite. It aims to provide standardisation between all robotic hardware. Its purpose is to communicate to and from robotic devices and the computer. For example, the walker's various sensors communicate with the GUI through it. ROS encourages the separation of code into small packages, called nodes, for easy reuse. As middleware, there is no actual ROS coding language; instead, it supports C++, Python, and Lisp.

3.3 GAME DEVELOPMENT FRAMEWORK

The two previous sections presented the project's work environment. However, they also raise some requirements for the games-to-be. First, the game must be compatible with the Ubuntu OS. Second, they should be able to read the sensor information published through ROS and integrate it into the existing WALKit application. More specifically, these are the requirements for the game's development framework. This choice is vital to the entire project since, once the work begins, it would be extremely difficult to change. Therefore, this section explains the plan followed to meet the main objectives outlined in this dissertation. For simplicity, three subsections are presented: low-level graphics APIs, script-based languages, or game engines.

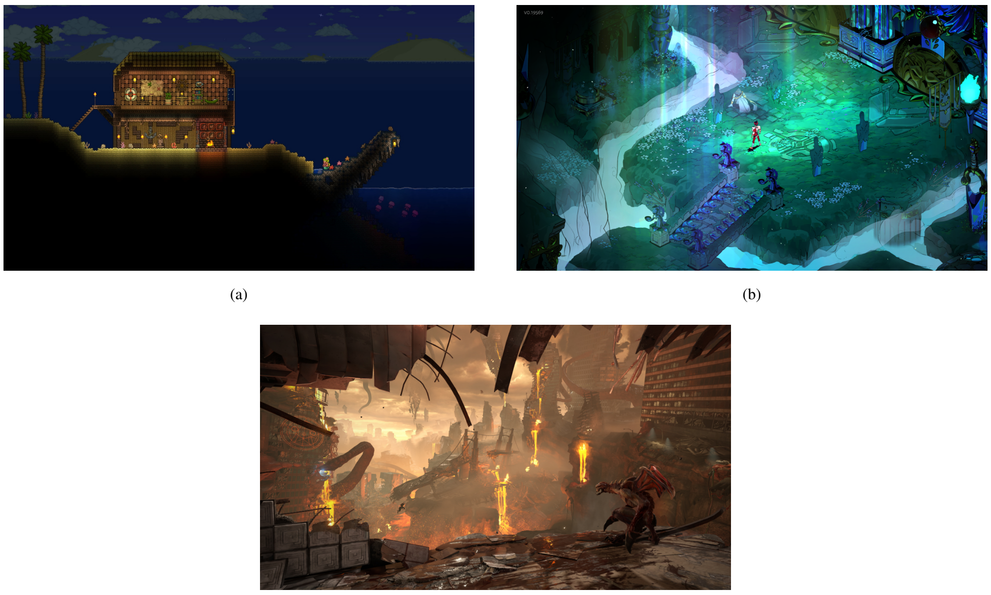
3.3.1 *Low-Level APIs*

Low-level APIs are programming frameworks that interface near-directly with the graphics render pipeline, with minimal overhead. The two most popular ones are DirectX and Vulkan. DirectX is currently in its twelfth version and under active development. It was created by Microsoft in 1995 and has since been used in Windows and Xbox-based game development. Vulkan on the other hand, is a relatively new API, built to succeed OpenGL. Unlike the proprietary DirectX family of APIs, Vulkan is open-source and cross-platform. However, both are written for low-level programming languages such as C or C++.

Low-level APIs are, theoretically, perfect for game development in non-standard environments such as these. They have little to no overhead, which allows for very efficient programs, in both performance and memory cost. Another benefit is their versatility; almost anything can be implemented, as their supported languages interact with the OS very closely.

However, these benefits come with expensive drawbacks. The absence of overhead means that most, if not all, game-related functionalities would need to be implemented. At their core, these APIs serve for all kinds of graphics work, not just game development. For this reason, games are rarely built directly with these. Instead, game frameworks are built on top of such APIs. For example, all the game engines listed below support, but are not limited to, the Vulkan library.

Figure 20 shows a variety of games made possible with the Vulkan API. It's worth mentioning that they were all built with their respective engines, built on top of Vulkan. The difference between these is the level of abstraction they provide. Regardless, it shows that the library is capable of various games.



(a) (b) (c)

Figure 20: Screenshots of various games built with engines that support the Vulkan API; **(a)** *Terraria*, made with the *XNA Framework* [53]; **(b)** *Hades*, made with *The Forge* [54]; **(c)** *DOOM Eternal*, made with *idTech 7* [55].

Additionally, a low-level language offers next to no safety nets. Resources require careful management because the smallest programming oversight can cause program, or even system, termination.

These reasons lead to their worse drawback: time. Game development is notoriously time-consuming, which would only be exacerbated by having to create any possible additional tools.

3.3.2 *Script-Based Languages*

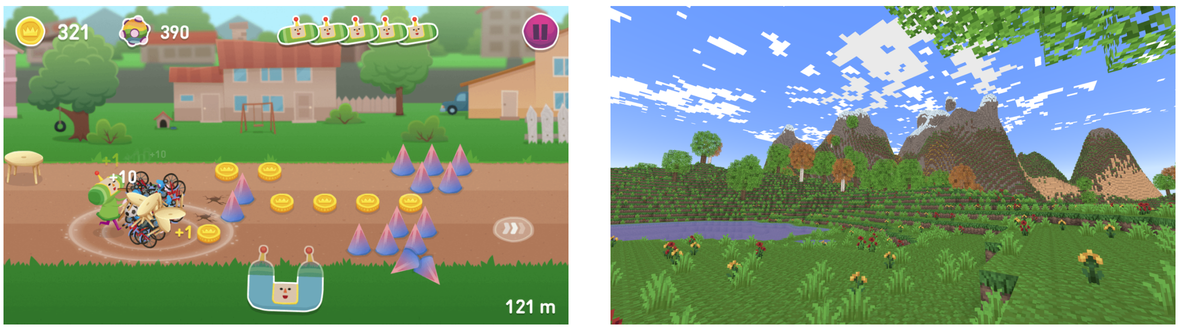
Scripting languages do not require compiling; the code is passed through an interpreter and executed at runtime. They tend to have better readability and portability compared to compiled languages.

As previously mentioned, QML's visual component is built on top of JavaScript, a web-oriented programming language. Another prevalent scripting language is Python, which focuses on desktop and server use. Python is known for its simple and clear syntax, which makes it easy to read and understand, even for non-programmers. Because of this it is popular in robotics, where some programming knowledge is required.

Both programming languages boast an impressive collection of additional resources, either in the form of JavaScript (JS)'s libraries or Python's modules. These include several packages oriented towards game development.

When considering a programming language for this project, JavaScript was a top candidate due to its compatibility with the QML language. It offers two great options for game development: Phaser.js for 2D games and Babylon.js for 3D games. Not only is JavaScript a widely used and easy-to-learn language, but it also has the added benefit of being type-safe. This means that most errors will not completely halt the game's execution, but simply cause it to fail silently. While this can prevent some errors from completely shutting down the game, it can also make development more challenging, as errors become harder to detect and may cause bigger issues down the line.

JavaScript game engines have become increasingly popular for developing both 2D and 3D games. To demonstrate this versatility, Figure 21 provides a comparison of two such engines, by showing screenshots of games developed using these engines, **(a)** "Katamari Damacy Mini" and **(b)** "VoxelSrv", respectively. The examples illustrate the capabilities of Phaser.js and Babylon.js in creating visually distinct and engaging games.

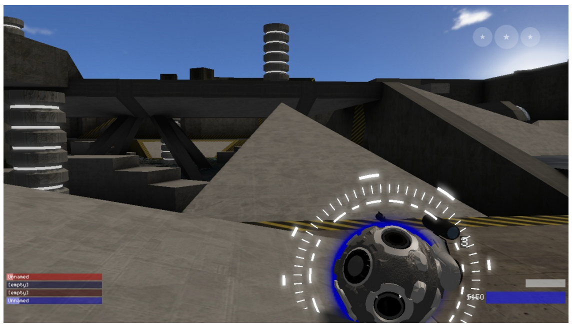


(a) (b)

Figure 21: JavaScript-based games, 2D and 3D; **(a)** Screenshot of *Katamari Damacy Mini*, made with *Phaser.js* [56]; **(b)** Screenshot of *VoxelSrv*, made with *Babylon.js* [57].

Python is similar to JavaScript, except with added features for ensuring code quality, such as type checking. When it comes to game development, Python offers the Pygame library for 2D games and the Panda3D framework for 3D games. Additionally, Python has a ROS API that allows for direct communication with the walker's sensors, thus making it a prime candidate. As illustrated in Figure 22, the Pygame library allows for the creation of 2D games such as "SpellCraft" (a), while the Panda3D framework enables the development of 3D games like "A3P" (b). These examples demonstrate the capabilities of Python in the field of game development and its potential as a choice for creating visually impressive and engaging games.

(a)

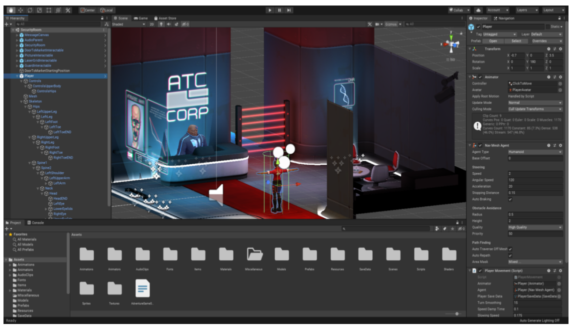


(b)

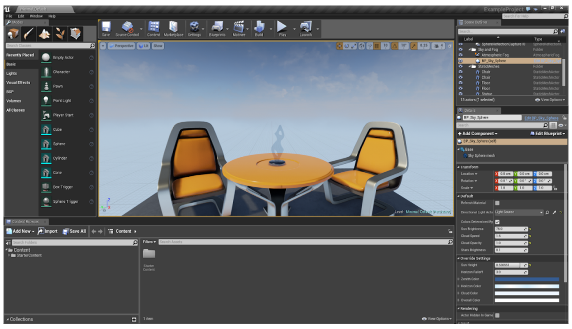
Figure 22: Examples of Python-based games; **(a)** *SpellCraft* [58] made with *Pygame*; **(b)** *A3P* [59] made with Panda3D.

3.3.3 *Game Engines*

Game engines, such as Unity and Unreal Engine, offer a visual editor that streamlines game development. As shown in Figure 23, objects can be easily placed and interacted with in the scene, allowing for instant previews of the game. This intuitive workflow is demonstrated in the **(a)** Unity editor and the **(b)** Unreal Engine editor respectively. These examples illustrate the time-saving benefits of using such an engine for game development and the flexibility it provides in designing and creating a game.



(a)



(b)

Figure 23: Game engines with their respective visual editors: **(a)** Unity [60] and **(b)** Unreal Engine [61].

Another benefit of game engines is the number of built-in functionalities. The graphical rendering pipeline is integrated, but can still be tweaked by the developers. The importing of assets, such as models, is also handled internally. Most difficult-to-implement systems, i.e., animation, physics, lighting, and UI, are basic features of these engines.

If a specific functionality is not in the base version, there are extensions available, either free or paid. Some engines even have their own store where these can be easily acquired, ranging from simple QoL features to massive game-specific development frameworks. Finally, most engines make the process of creating new custom extensions very straightforward.

There are a few game engines from which to choose. This section examines the three most popular, publicly available ones. All three have Long-term support (LTS) from their respective creators and thriving communities that offer vast learning resources and support.

Unity, released in 2005 by Unity Technologies, is known for its ease of learning and accessible licensing plan. This game engine has been used in a variety of projects, from indie games to more high-profile titles, as demonstrated in Figure 24. The screenshots showcase the diversity of games that can be developed using Unity, such as the 3D platformer **(a)** *Yooka-Laylee*, the digital card game **(b)** *Hearthstone* and the action-adventure game **(c)** *Ori and the Will of the Wisps*. These examples highlight the versatility of Unity as a game engine and its ability to support a wide range of game genres and styles.

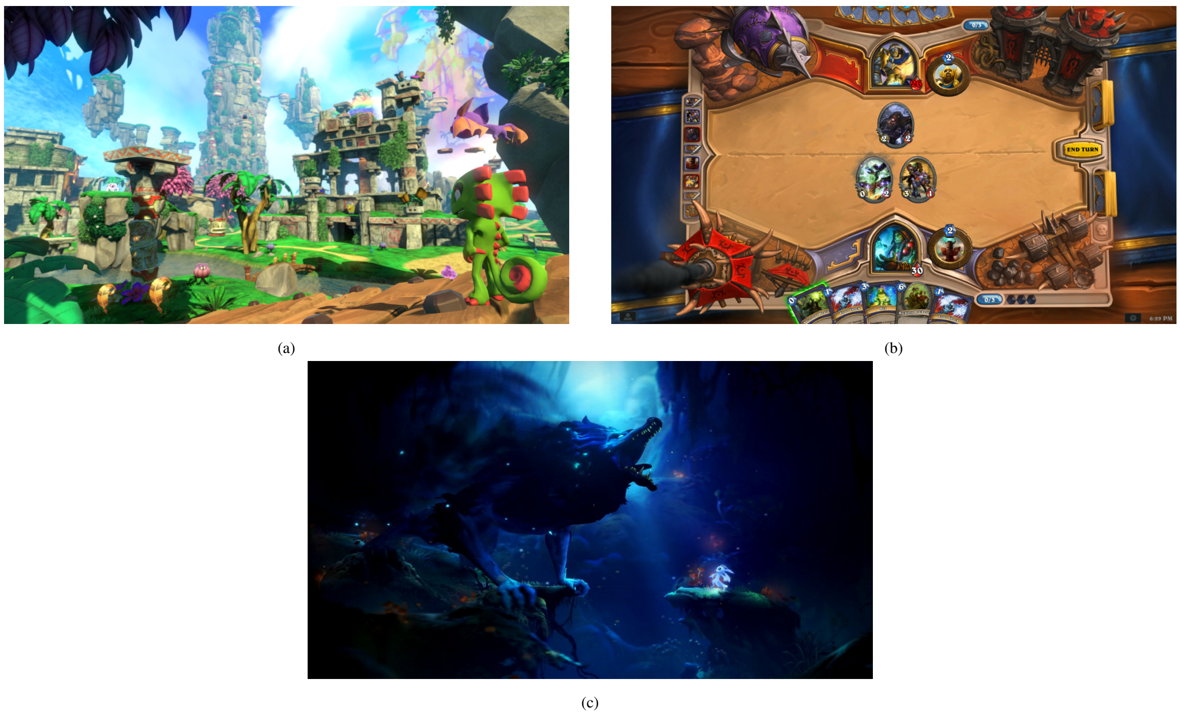


Figure 24: Examples of Unity powered products: **(a)** *Yooka-Laylee* [62]; **(b)** *Hearthstone* [63]; **(c)** *Ori and the Will of the Wisps* [64].

Unity's licensing plans scale cost-wise, according to team sizes and revenue, with additional free student licenses. The only features locked behind the paid plans are cloud-based services and technical support. Moreover, it supports both 2D and 3D. Most 2D games are in 3D behind-the-scenes, with camera adjustments. This way, any development knowledge translates nicely from one type to the other. While its original purpose is game development, Unity has been adapted for and slowly expanded into other fields, such as Robotics.

Unreal Engine (UE), currently in its fifth version, is widely considered the choice game engine for AAA[1] game development. As shown in Figure 25, it can achieve better visual fidelity out-of-the-box, compared to its competitors. The screenshots showcase the diversity of games and other products that can be developed using Unreal Engine, such as the action role-playing game **(a)** *Borderlands: The Pre-Sequel*, the horror game **(b)** *Little Nightmares II* and the TV show **(c)** *Mandalorian* which use UE for special effects. These examples highlight the versatility of UE as a game engine and its ability to support a wide range of game genres and styles and also its use in video rendering, taking part in large-budget movie and television projects.

(a)

(b)

(c)

Figure 25: Examples of Unreal Engine powered products; **(a)** *Borderlands: The Pre-Sequel* [65]; **(b)** *Little Nightmares 2* [66]; **(c)** The TV show *Mandalorian* [67].

Godot is a relatively new open-source game engine, first released in 2014. In recent years, it has gained popularity among game developers. However, due to its relatively recent introduction, it is not as well-equipped with tools and capabilities as the more established engines. The benefits of using Godot include its lightweight executable, and open-source nature. As shown in Figure 26, it is capable of producing a variety of games such as the puzzle-adventure game **(a)** *Primal Light*, the point-and-click adventure game **(b)** *Until Then* and the simulation game **(c)** *Hive Time*. These examples highlight the versatility of Godot as a game engine despite its limitations.

1 Games produced by large companies, with higher development and marketing budgets. Also referred to as *Triple-A*

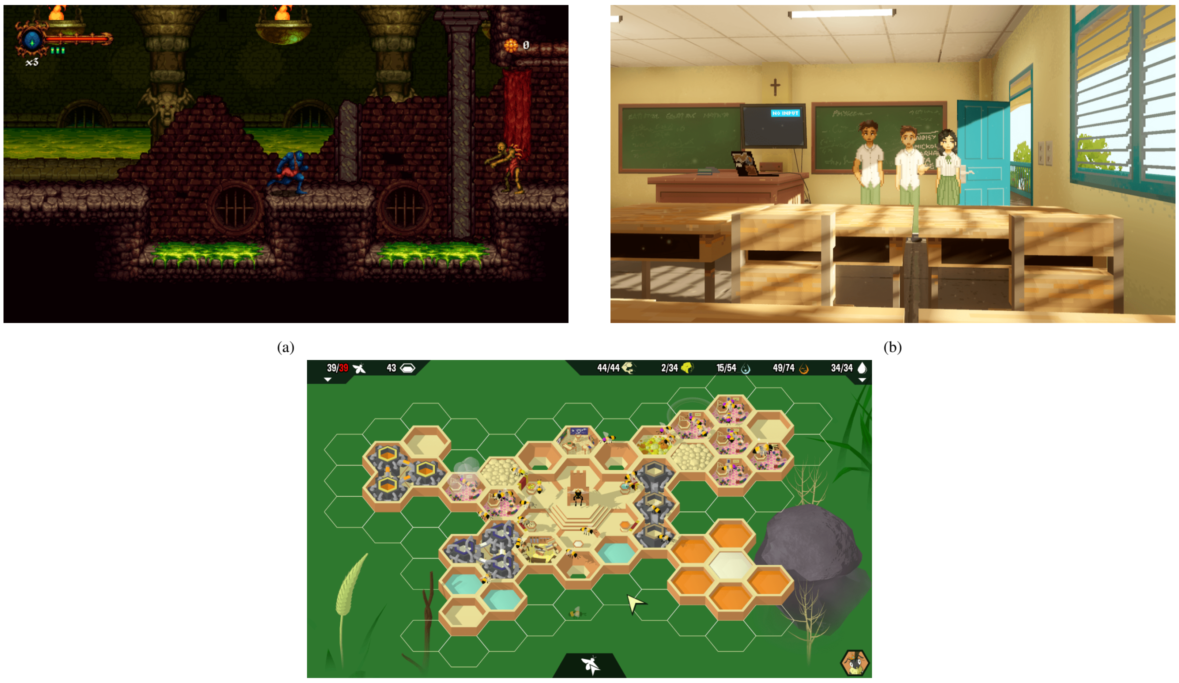


Figure 26: Examples of games made with Godot: **(a)** *Primal Light* [68]; **(b)** *Until Then* [69]; and **(c)** *Hive Time* [70].

3.3.4 *Discussion*

This subsection compares the options previously presented, attempting to select the one that best meets the project's requirements. First, the technical aspects: communicating with both ROS and the walker's application. Second, and less apparent, are the time and resource constraints of developing two games in less than a year.

Game Engines appear as the most convenient options, both time and capability-wise. Their worst aspect is the performance overhead that comes with so many features. They also produce stand-alone executables, which can not be integrated directly into the application.

Script-based languages are the middle ground between low-level APIs and game engines. Despite being built on top of JavaScript, QML does not support many of its libraries. This issue stems from how much it has diverged from its source. Unfortunately, phaser.js and babylon.js are both included in these.

On the other hand, Python does not integrate easily into the application nor provide any game-related features at the level of game engines. Its only benefit is the ability to communicate with ROS. However, that is not a strong enough benefit to offset the negatives. Development with both languages' libraries involves designing everything manually with code. That can be very time-consuming and, if not properly managed, it could devolve into large amounts of files. Another problem is that the code may not be easily readable, making it difficult to comprehend for new individuals coming into the project. For these reasons, other frameworks were discarded, leaving only the APIs and engines as viable options.

Low-level APIs can be extremely efficient, partly due to a lack of any additional features. Most of this efficiency relies on the programmer's knowledge of the language, operating system, and any other related components. The application's backend is written in C++, which could make the integration of an API-developed game feasible. Given enough time and manpower, it would arguably be the best choice. Unfortunately, these two resources are scarce in a project of this scale, thus excluding low-level APIs, leaving only game engines.

When choosing game engines, two issues need to be addressed: i) externally communicating with both ROS and ii) the application. Afterwards, it is a matter of which better fits the project's profile.

Godot's early development stage makes it a less secure choice in terms of LTS and the presence of potential bugs, leaving Unity and UE as possible options.

The Unreal Engine presents the same difficulties as the low-level APIs in terms of complexity and performance due to its high visual fidelity. Given the limited computational resources of the robotic walker, this may pose a challenge. On the other hand, the Unity game engine has already gained popularity in the field of robotics and is well-known in the BiRDLab, where this dissertation was conducted. Choosing Unity would be beneficial, as any modifications or additions to the game could be more easily implemented, given the familiarity with its workflow.

Table 5 condenses each game engine's advantages, allowing to compare them to each other, in which "Dev Speed" stands for "Development Speed", "Avg." stands for "Average", and "Diff. Libs" stands for "Different Libraries".

Table 5: A summary comparison of different game-dev frameworks.

Framework	Development Speed	Visual Potential	Integrates	Code or Visual	Performance	2D / 3D
APIs	Slow	High	Yes	Code	Higher	Both
JS	Avg.	Avg.	Unlikely	Code	Avg.	Diff. Libs
Python	Avg.	Avg.	No	Code	Avg.	Diff. Libs
Unity	Fast	Higher	No	Both	Lower	Both
UE	Fast	High	No	Both	Lower	3D
Godot	Fast	Higher	No	Both	Lower	Both

Considering Table 5 and the reasons aforementioned, it was concluded that Unity is the best choice. As a game engine, its visual editor and built-in features considerably speed up the workflow. Of all the game engines, Unity is lighter than UE and more complete than Godot, which offers a fair balance between performance and visual fidelity.

3.4 THE UNITY GAME ENGINE

This section introduces the Unity Game Engine, offering insight into the engine's development process and terminology. It starts by showcasing the visual editor, followed by an overview of core concepts, and ending with an in-depth analysis of scripting and Unity's core classes. These are ordered by scale, from most to least encompassing.

3.4.1 *Unity Editor*

The visual editor combines several tools in a single display. Figure 27 illustrates a typical Unity editor screen.

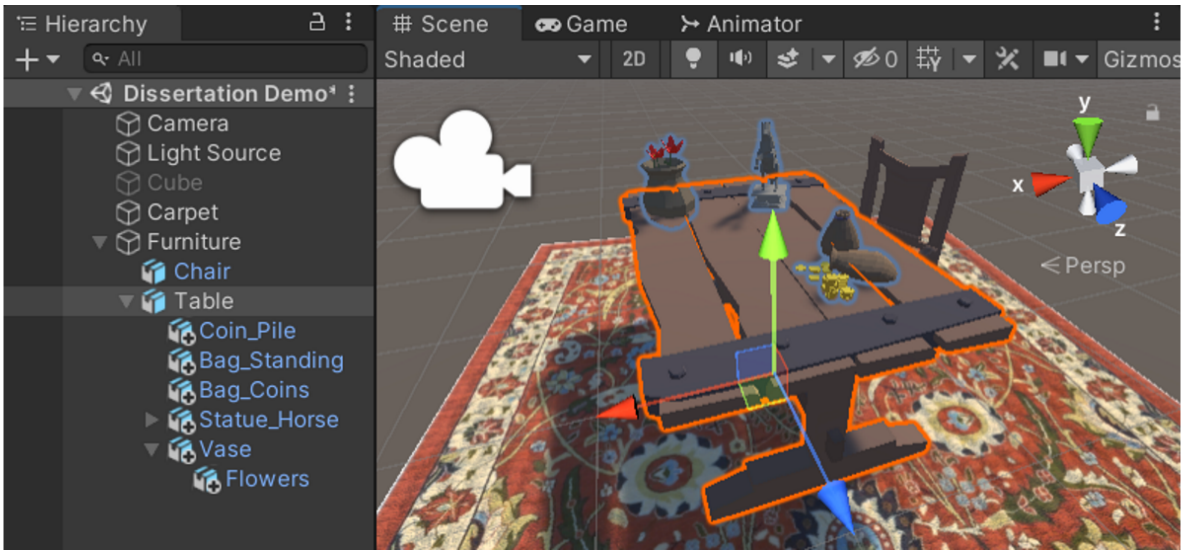


Figure 27: An example of the Unity Editor's Hierarchy Tab and Scene Viewer.

The Scene Editor is Unity's prime tool, aptly located at the centre of the editor. Figure 27 illustrates three objects in the scene: a cube, a light source, and the camera, all of which can be moved, scaled and rotated by clicking and dragging. Moreover, the editor has the ability to play the scene in real-time, enabling rapid testing and prototyping without repeatedly building the entire game.

3.4.2 *Scene*

In a Unity application, almost everything happens in a scene. They contain all or some part of the game. For example, a scene can comprise an entire game, or there can be multiple scenes, one for each level, with their own environment, obstacles, and UI.

Figure 27 highlights the hierarchy view, a list of all *GameObjects* in the current scene. The Scene Viewer shows the scene in its initial state. After the game starts, all changes made to a scene, unless specifically programmed otherwise, are temporary. Every time a scene is loaded, it has the same initial state, which means that every GameObject and their values are the same.

3.4.3 *GameObjects*

The *GameObjects* is one of the most vital concepts in Unity. In a scene, everything is a *GameObjects*, from lights to cameras to the player character. They can have a parent and multiple children, also *GameObjects*. Disabling, deleting, or transforming a parent object does the same to its children. Additionally, empty parents are often used to group *GameObjects* together. For instance, moving the table shown in Figure 43 will also move its contents accordingly.

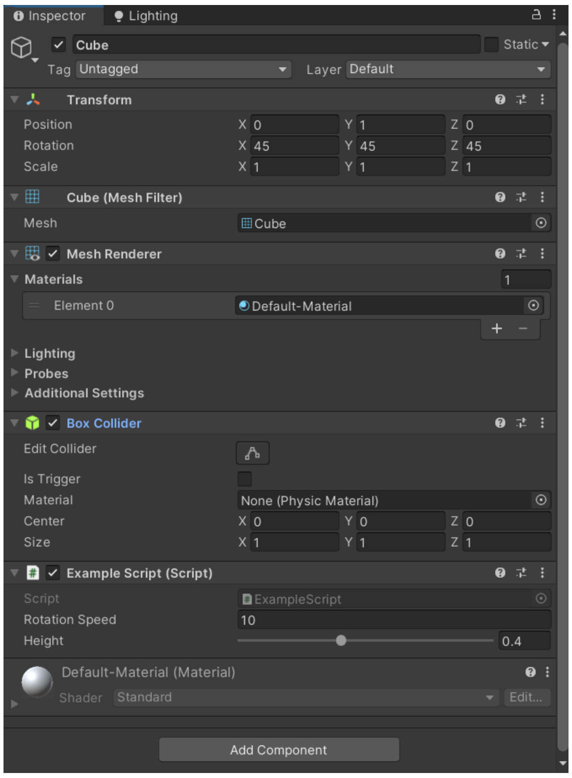
Figure 27 has a few notable aspects. On the left is the "Scene Hierarchy Tab", where all *GameObjects* are displayed. On the right is the "Scene View", where the same *GameObjects* appear as they would be in the game. Since the Cube *GameObjects* is deactivated, it does not appear in the Scene View. Then, there is an empty Furniture *GameObject*, which parents both the Table and the Chair. Additionally, the Table parents all the items placed on it. Moving this would move them, but not the Carpet underneath. Lastly, the Table was selected, as seen by the blue highlight in the "Hierarchy Tab", which highlights it, in orange, and any children in the "Scene View".

3.4.4 *Component*

In the Unity game engine, a *GameObject* is one of the building blocks for creating a scene or game. However, these objects do not have any behaviour or functionality on their own. For that, developers attach *Components* to the *GameObjects*. These components hold variables and define the object's behaviour. For example, a cube *GameObject* has a Component that defines the cube's 3D model and colour. The camera is a *GameObject*, with a *Camera Component*. This applies to everything in a Scene: lights, floors, trees, the player, and even the Game Manager.

Every *GameObject* has a *Transform* Component that defines the object's position within the scene. Additionally, some components require others to function properly. There are multitudes of different components in Unity, not to mention the many others available as external packages. While it would be impractical to cover all existing components, Figure 28 includes the most commonly used ones, alongside their descriptions.

- **Mesh Filter** - It defines the object model.
- **Mesh Renderer** - Together with the Mesh Filter (which it requires), it renders the model, applying Materials and defining how light affects the object, among others.
- **Material** - While not directly attached to a *GameObject*, the "Mesh Renderer" uses it to add colour to its "Mesh Filter" model. Colour includes flat colour, textures, and transparency, among others. Additionally, objects are not limited to a single "Material".
- **Collider** - It defines a volume around the object and detects collisions with other *GameObjects* with Colliders attached. Collision detection is present in many aspects of video games. For example, to prevent characters from walking through walls, detecting when projectiles have reached a target or setting a checkpoint once the player reaches a certain point on a level.



```
Inspector   Lighting
✓ Cube                                    Static
Tag Untagged                 Layer Default
Transform
Position         X 0      Y 1      Z 0
Rotation         X 45     Y 45     Z 45
Scale            X 1      Y 1      Z 1
Cube (Mesh Filter)
Mesh             Cube
✓ Mesh Renderer
Materials                          1
  Element 0      Default-Material
Lighting
Probes
Additional Settings
✓ Box Collider
Edit Collider
Is Trigger
Material         None (Physic Material)
Center           X 0      Y 0      Z 0
Size             X 1      Y 1      Z 1
✓ Example Script (Script)
Script           ExampleScript
Rotation Speed   10
Height                                 0.4
Default-Material (Material)
Shader  Standard                        Edit...
Add Component
```

Figure 28: An example of the Unity Editor's *GameObject* Inspector.

3.4.5 *Scripts*

Scripts are essential to any application made with Unity. They provide the flexibility needed to create unique game experiences and the specificity otherwise impossible with premade components. They can be as simple as moving an object or as complex as managing the entire state of the game.

Internally, the engine uses C and C++ but provides a C# abstraction, meaning all code is written in C#. Usually, each script holds a single class. Additionally, scripts can hold references to other components, which, for simplicity, can be considered scripts made by Unity. As scripts, they have class methods and properties that can be accessed and manipulated. The same applies to scripts that interact with other scripts. Some common script uses are: i) Moving a character in response to player input, ii) setting the properties of other components, such as the transform, or iii) defining what happens when two "Colliders" intersect.

3.4.5.1 *MonoBehaviour*

Every Unity script has a single class that inherits *MonoBehaviour* by default. Otherwise, they cannot be attached to a *GameObject*. This convention ensures that user-made scripts conform to Unity's game design philosophy and avoid potential issues.

Several of Unity's functionalities, such as the "Time" and "Mathf" class methods, are only available to *MonoBehaviours*. This changes how scripts are written, mainly through the two following methods:

- **Start** - This method gets called when the object is created and active. A typical use is to set variables that reference other components, as these only exist when the scene starts.
- **Update** - Called every frame[2]. Every script in every active *GameObject* in the current scene will call this method. The next frame can only happen after every *Update* has finished.

Finally, despite using multiple threads internally, the *MonoBehaviour* API guarantees no race conditions[3] will happen. Essentially, developers can treat the program as if it were single-threaded.

3.4.5.2 *Important Classes*

Unity has additional classes that provide functionality [71]. Virtually every script uses one or more of these. In addition to components, Unity's built-in classes provide additional functionality to scripting. Like with Components, there is a multitude of these, the more commonly used being:

- **Time** - it provides information regarding the time between frames and game startup.
- **Vector3** - it is a structure that holds x, y, and z float coordinates and provides basic vector operations, such as addition and multiplication.
- **Mathf** - it provides several math-related methods, including Vector-based ones, such as the dot product.
- **Transform** - it includes methods and properties related to the object's position, rotation, and scale, all of which are Vector3s.
- **Object** - it has methods like instancing and destroying other Objects. Every *MonoBehaviour* holds a reference to its own *GameObject* and *Transform*.

3.4.6 *Prefabs*

A Prefab is a type of asset that stores a *GameObject* with all of its components and properties. It is a way to reuse common game objects and their properties in a Unity project. For instance, a group of trees in a game with the same model and material can be managed by creating a single tree prefab and then instantiating multiple copies of it in the scene. This way, any changes to the tree prefab will also update its instances.

The Prefab Variant feature offers the ability to create variations of a prefab asset with different properties or components while still maintaining a link to the original prefab. This allows for the creation of different variations of the same prefab without the need for multiple copies of the asset. For example, a base enemy prefab can be a

2 A single still image or "snapshot" of the game's visual output. The number of Frames Per Second (FPS) determines the smoothness of the game's animation. Higher FPS will be smoother, while lower fps can seem choppy and stuttery.

3 Can occur when two programs access the same resource simultaneously, which leads to unpredictable behaviour.

starting point for the creation of many different variations, such as one with a different colour and another with another weapon. This way, a new prefab does not need to be created from scratch and changes and updates to the base prefab can be easily made and the variants will be updated as well.

Prefabs can be created by dragging a *GameObject* from the Scene View into the Assets folder.

3.5 ADDITIONAL FRAMEWORKS

This section covers two additional requirements for the game: integrating with the walker's application and accessing the walker's sensor information. Due to complications with the integration process, that section instead explores possible alternatives. For the second, i.e. the sensor information acquisition, it details the solution found.

3.5.1 *Integration*

Integration means that starting a session would run the games as any other algorithm: they could be shown and hidden, get dragged along with the window, and close once the session stops. However, for reasons further explained in this chapter, implementing something like this would be challenging and somewhat outside the project's scope. Therefore, an alternative that largely conforms to the requirements was devised.

The straightforward solution is making a native game with the GUI capabilities of QML. And while that approach could work for a 2D game, the walker's current QML version (4.9.2) does not support 3D[4].

Another issue is that creating a game using GUI tools can be quite complex, similar to trying to code a game without the help of a game engine. Performance would be another possible issue with using GUI elements. Animation is possible in QML for the occasional button click or slider, but games require constant refreshing and animating, which QML is not optimised for.

Another option would be to contain an external game inside the application window, in the space reserved for the algorithms.

However, this involves very low-level operations that require using the OS's window management API. Manipulating OS elements directly is challenging, and it can also be dangerous, with unintended side effects, so any possible alternatives are welcome.

The third option is to have the external game communicate with the app rather than integrate. If both programs communicate, the app can share its status with the game, including window position, being shown or hidden, etc. With this, it is possible to "integrate" the game, fulfilling all requirements.

3.5.2 *WebSocket Protocol*

WebSocket is a two-way communications protocol that uses Transmission Control Protocol (TCP) to send information packets. Figure 29 illustrates a diagram of the WebSocket Protocol. A working system includes one or

4 Qt 3D was first introduced in QT's version 5.5.

more clients and one server. The clients attempt to connect to the server, and if successful, the server returns a token indicating a successful connection. They can now communicate until one closes the connection.

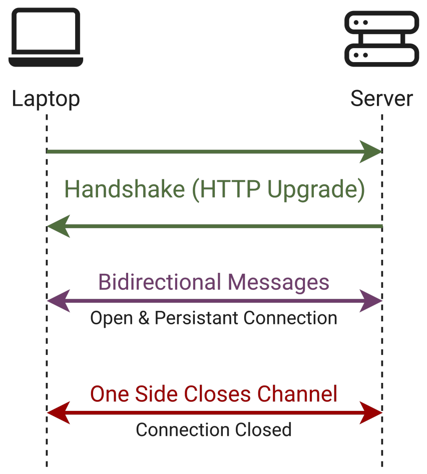


Figure 29: A diagram of the WebSocket Protocol.

Instead of having a direct pipeline for communication, the game will start a WebSocket (WS) server. When the game launches, the app's client will try to connect to the server at a rate of once per second, with a configurable timeout value of 30 seconds by default.

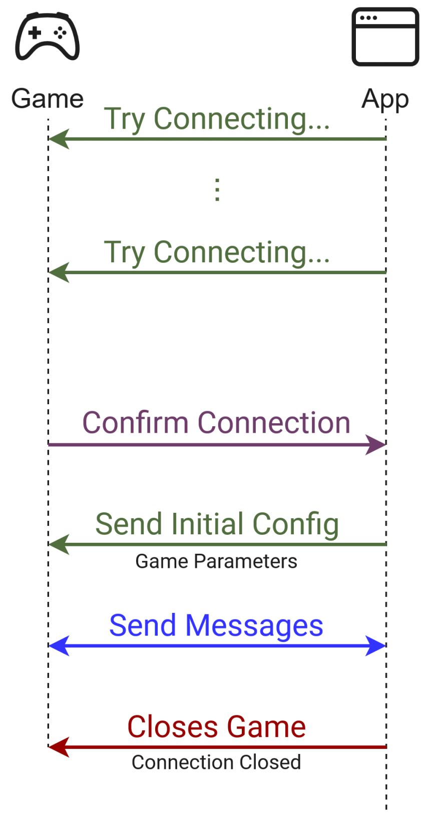


Figure 30: A diagram of how the Game and App communicate.

3.5.3 *ROSBridge*

To communicate directly with the WALKit's sensors, the ROS# library was used. Specifically, a version of the library with Unity support. Together with the external program ROSBridge, it made for an easy and straightforward solution.

ROSBridge handles all direct communication with the ROS API. In turn, it publishes all sensor information through a WebSocket server, conveniently converting it to the JavaScript Object Notation (JSON) format. ROS# provides ways to access this server, and its Unity variant includes *MonoBehaviours* that already implement this library, further simplifying the process. All that remained was creating custom classes suited to the walker's specific sensor configuration.

3.5.4 *Conclusion*

In conclusion, having proper integration was a very time-consuming solution, so an alternative was required and found. The solution uses WSs to communicate between the WALKit app and games to emulate integration as closely as possible. On the other hand, the communication with ROS was established through a combination of RosBridge and the library ROS#. Figure 31 illustrates a diagram of the proposed solution, explaining how the game, the WALKit's app, and RosBridge communicate.

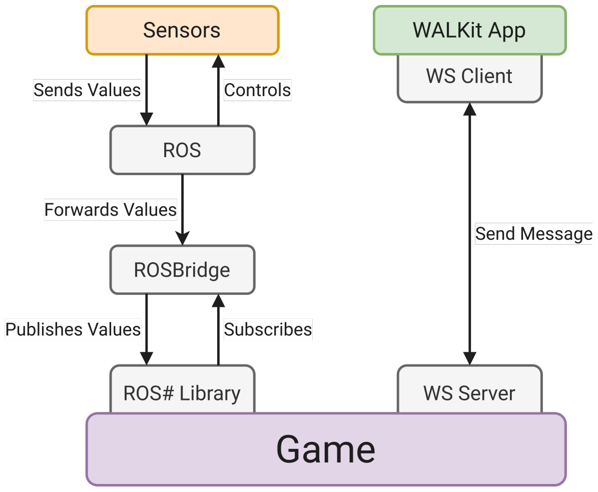


Figure 31: A diagram of how the Game and App and RosBridge communicate.

The diagram of Figure 31 summarises the additional frameworks established. There are some advantages to this solution: first, by not integrating into the walker, the games could theoretically get ported to other devices, providing they set up the client; second, while the WebSockets are used locally in this case, one could theoretically connect to a WS client and ROSBridge on another device entirely; for instance, the game could be played on a tablet while walking with an IMU.

4

GAME DEVELOPMENT: COGNITIVE GAME

4.1 INTRODUCTION

The Cognitive Game, also referred to as the dual-tasking game, was the first one developed in this dissertation. The main focus of this game will be to study whether a cognitive task is beneficial to gait rehabilitation. The name "dual-task" comes from how the patient has to focus on driving the robotic walker and playing the game. This SG was created and developed alongside the medical team, whose input is invaluable for rehabilitative games such as these. The Cognitive Game was originally on the WALKit's predecessor, the ASBGo.

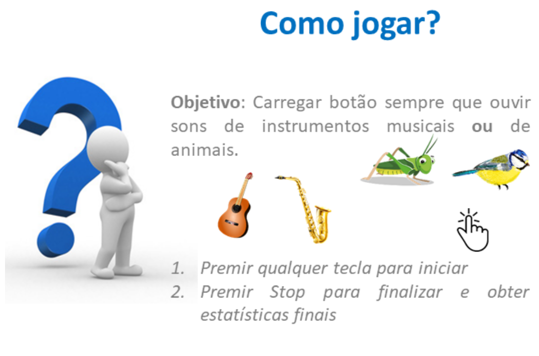
The ultimate goal for this game was to improve and integrate the ASBGo version into WALKit software, considering possible backend and frontend modifications. The original game code is written in C++ and organised by game levels. However, some levels were not yet fully implemented, and the ones that were were hard-coded. For this reason, creating new levels may be a complex, cumbersome task, and may exponentially increase the project's size and complexity. Therefore, the new version was recreated considering the methodology explained in Chapter 3.

The following sections present a small overview of the original game and the additions and modifications performed during the recreation process. Moreover, it details all the significant components that make up the new game, including the new algorithm, which serves to configure and launch the game.

4.2 PREVIOUS GAME VERSION

The original game is simple, with three modes: i) visual, ii) audio, and iii) dual. The visual mode has random shapes appearing at random intervals. The goal is for the user to press a button on WALKit whenever a specific shape appears. This shape could be a circle, a triangle, or a square in different colours. Similarly, the audio mode requires the user to press the button whenever they hear a specific sound (it could be a cricket or the sound of drums). Lastly, the dual mode encompasses the other two.

In all three modes, it is intended that levels become progressively more challenging, with more shapes or sounds to identify or conditioned levels, i.e., the users have to react to a specific shape/sound considering a previous one. To verify the progress of the user throughout therapy sessions, the game counts the user's reaction time to a given stimulus. Figure 32 presents one example of level instructions for both audio and dual game mode.

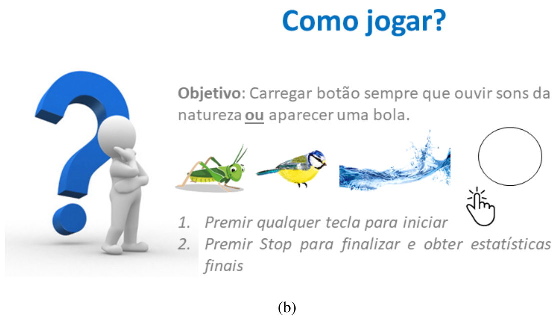


Como jogar?

Objetivo: Carregar botão sempre que ouvir sons de instrumentos musicais **ou** de animais.

1. *Premir qualquer tecla para iniciar*
2. *Premir Stop para finalizar e obter estatísticas finais*

(a)



Como jogar?

Objetivo: Carregar botão sempre que ouvir sons da natureza **ou** aparecer uma bola.

1. *Premir qualquer tecla para iniciar*
2. *Premir Stop para finalizar e obter estatísticas finais*

(b)

Figure 32: Level Instructions for (a) Audio and (b) Dual Game Modes (Old Version).

4.3 NEW GAME VERSION

As previously mentioned, the medical staff played a part in designing the original game with the technical team. This design included the instructions for most levels, even the ones that were not fully implemented at the time. Additionally, this version is written entirely in a single file, with over 2000 lines of code. Much of this is duplicated code, with minor tweaks for each game mode.

Levels are also hard-coded, which means adding or tweaking one requires a new build, and this is cumbersome to perform during a therapy session. Moreover, to make these changes, one would need coding knowledge. For these reasons, the entire game structure was re-designed. The following sections focus on this, divided into Level, Shape, and Game Loop. Figure 33 illustrates the new version of the Cognitive Game during play.

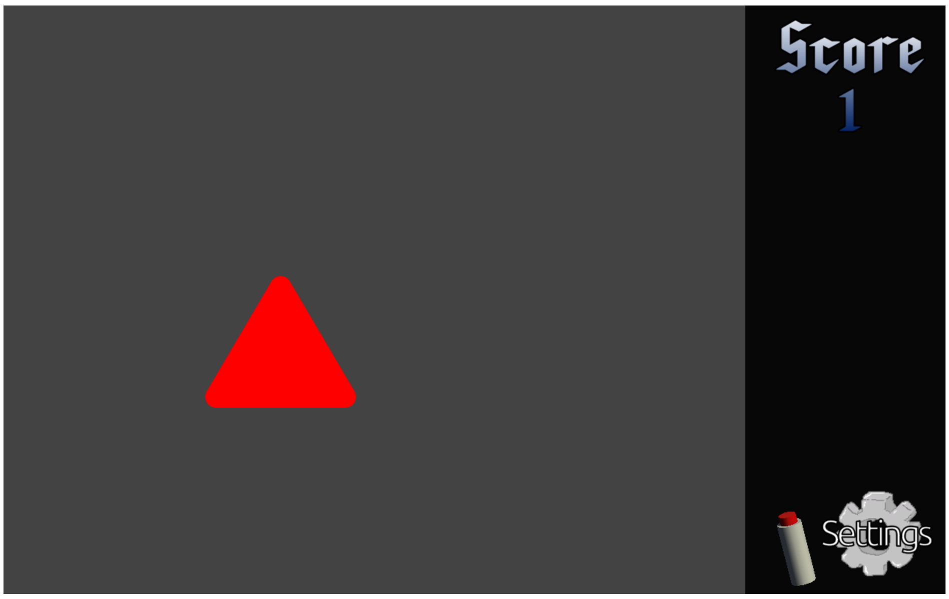


Figure 33: New Version of the Cognitive Game during play.

4.3.1 *Level Structure*

In this new version, the game itself does not contain any levels. Instead, it parses external "level" files. The files are in the JSON format, whose primary feature is readability. The medical staff should have no issues with these, at least for small changes. Furthermore, splitting each level into its own file allows for better customisation, e.g., unique background colours and shape timings. Moreover, it also solves the problem of having to re-build the game whenever a level gets changed.

The way levels work has also changed, most significantly, the removal of game modes. Now every level has shapes, each one able to have an image, sound, or both. This makes it possible to design levels where some shapes are strictly visual while others only emit sound. Further changes include better configuration granularity, exposing some variables, and added functionality.

The list that follows provides a complete overview of all the possible configurations for the levels. If any fields are missing or left blank, default values will be applied instead.

- **gameMode** - Whether the level is **Visual**, **Audio**, or **Dual**.
- **order** - The order in which the levels are ordered when shown in the app.
- **levelName** - The name of the level. To avoid having to name the files themselves.
- **description** - Gives a summary description of the level behaviour to the file's reader.
- **instructions** - The instructions intended for the user, which the WALKit app displays on-screen.
- **simultaneosSquares** - The number of squares displayed simultaneously every interval.
- **interval** - The time between shapes being shown.
- **timeOnScreen** - The time that shapes remain on screen.
- **shapes** - A list containing all the shapes in the level. Each one has its specific information.
- **forceCorrectEveryNth** - The nth image/sound will always be one of the correct ones.
- **background** - The playable area background; Supports RGB, HEX, and some named colours. Additionally supports local and web URLs for image backgrounds.
- **chains** - Defines the chains, if desired. Chains are explained in section 4.4. This will also override **simultaneosSquares**.

As previously stated, shapes have individual information. Every level has a list of shapes, each with the same chance to appear on-screen. That chance can be increased by adding duplicates or setting the count property.

4.3.2 *Shapes*

Shapes are the foundation of the game. They can have both visual and audio components. Like the level structure, shapes have also been implemented differently from the previous version. Now, each one is its own entity, and they all get instanced at the start of the level, being hidden or shown as needed. This improves performance, since creating new objects is more computationally intensive. When shown, each shape decides its position randomly, ensuring no overlap.

Just like the game configuration, each shape has its own properties. The list below briefly explains them, alongside an example of a level file.

- **Name** - The internal name assigned to the shape. It only sees use in the chain mode.
- **Width** - The shape's width in pixels.
- **Height** - The shape's height in pixels.
- **Shape** - What shape to show, e.g., square or diamond.
- **Image Source** - The file path to any local image.
- **Colour** - Applies colour to the shape, even if it has an image.
- **Correct** - Whether the shape is correct.
- **Audio** - Which sound to play, from a list of pre-compiled values.
- **Count** - The number of copies added to the pool, effectively increasing its chance of appearing.

4.3.3 *Game Loop*

The game starts by launching the executable with an argument that indicates the level's file path. The game also supports others for overriding ROS and WS addresses. Figure 34 illustrates a diagram of the cognitive game's starting operation.

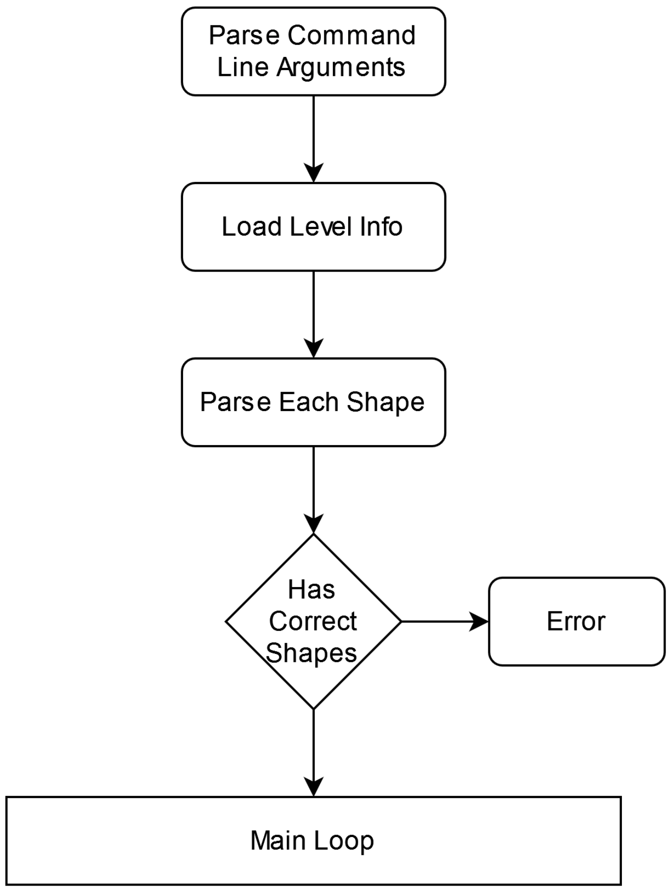


Figure 34: Diagram of the Cognitive Game's Starting Operation.

Figure 34 shows a regular game start-up flow. After launching, the game initialises the input/output systems, then parses the level file. At the same time, each shape is also parsing its own information. The level configuration file may have some invalid values, like a file path that does not exist. In most cases, this would cause the game to crash, but the game attempts to prevent that by using default variables instead.

Once everything is ready, the game subscribes to the WS to receive external messages, i.e., from the WALKit app. All messages get buffered by the WS Manager for posterior access. Then, the game moves to the Main Loop, shown in Figure 35, which repeats every frame.

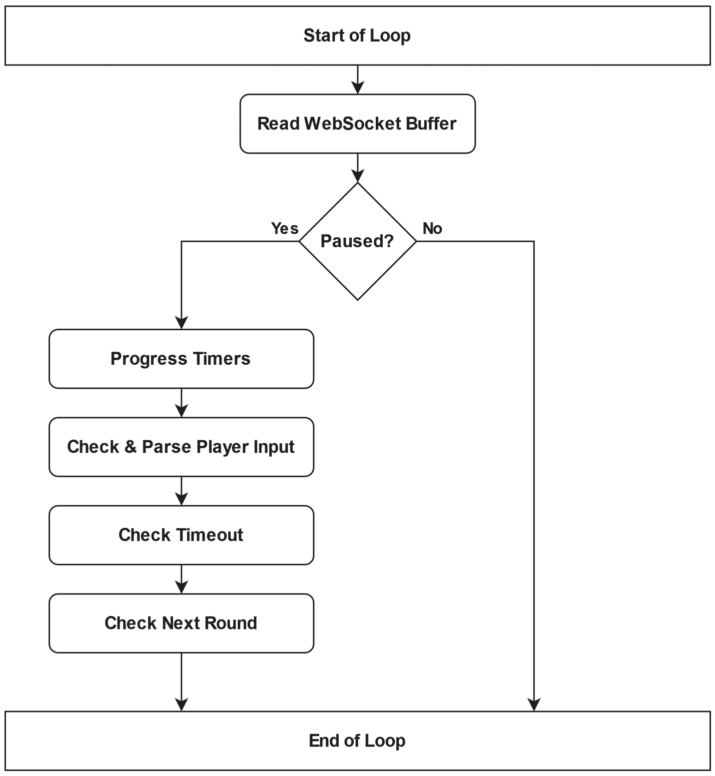


Figure 35: Diagram of the Cognitive Game's Main Loop.

This loop contains the bulk of the game. Its functions range from reading player input to keeping track of which shapes to show. As most programs do, it systematically goes through a list of steps.

The first step is to read from the WS Manager's buffer; in case there are incoming commands, such as to pause, quit, or resume. Next, if the game is paused, no further action is taken until the next frame. Otherwise, the game updates all time-related variables, including the time until the following shapes appear and the time until they disappear.

Afterwards, the game checks for player input, which can be: mouse clicks, screen presses, or the walker's button. Furthermore, pressing outside the playable area does not affect the game. Upon confirming a valid input, the game decides whether it was correct or not to do so, attributing the score and advancing the game. This input validation process is illustrated below in Figure 36.

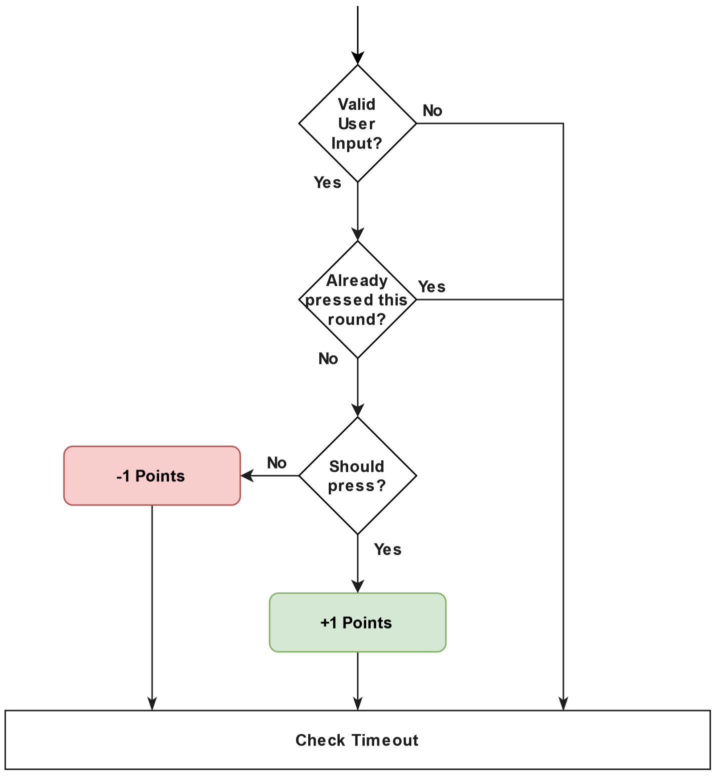


Figure 36: Diagram of the Cognitive Game's Player Input Process.

Afterwards, the game checks if the player timed out, likewise deciding whether that was correct or not. Lastly, after a certain amount of time (defined in the *interval* field of the level configuration file) has passed, most variables get reset to then show a new batch of shapes. This loops infinitely until the program closes. If this was a wrongful termination, there are safeguards against possible data loss. Primarily, the game should be able to catch most terminations and save successfully, but otherwise, there are still periodical backups.

4.4 CHAIN MODE

Chain Mode has already been referenced occasionally throughout this chapter. This name characterises a subset of unimplemented levels, where some shapes are only correct if preceded by certain shapes. Figure 37 exemplifies this: a red square is only correct if the previous shape was a red triangle. The chain here is red-triangle, red-square. This concept can be expanded further, with larger chains or even chains within one another.

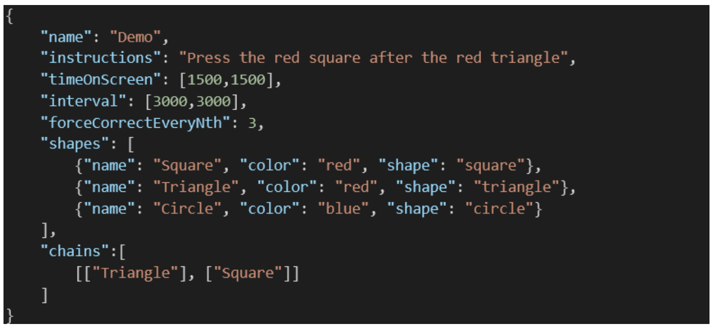

```
{
    "name": "Demo",
    "instructions": "Press the red square after the red triangle",
    "timeOnScreen": [1500,1500],
    "interval": [3000,3000],
    "forceCorrectEveryNth": 3,
    "shapes": [
        {"name": "Square", "color": "red", "shape": "square"},
        {"name": "Triangle", "color": "red", "shape": "triangle"},
        {"name": "Circle", "color": "blue", "shape": "circle"}
    ],
    "chains":[
        [["Triangle"], ["Square"]]
    ]
}
```

Figure 37: Example of Level Config File with a chain.

4.4.1 *Level File Structure*

Level files can have the **chains** property. If present and not empty, the game will recognise that there are chains and parse them accordingly, adjusting any necessary settings. In these files, a chain is a list of shape names. Consider the following example level structure: Four shapes, named A through D. Additionally, there are two chains in this file: A-B-D and A-C-D. In this case, the user is correct only when the first shape is A, the second B or C, and the third D. These two separate chains can be represented like so:

$$[A, B, D]$$
$$[A, C, D]$$

A shorthand is introduced to the level file for these types of chains. The two chains, A-B-D and A-C-D, can be simplified to a single line, taking into account that the only difference between them is the B and C shapes in the middle. This would now look like so:

$$[A, [B, C], D]$$

This method helps shorten the time spent writing levels and avoids large amounts of duplicates. It can also be easier to visualise these nested lists as matrices, as follows:

$$[A, [B, C], D] \quad \Rightarrow \quad \begin{bmatrix} A, \begin{bmatrix} B \\ C \end{bmatrix}, D \end{bmatrix} \quad \Rightarrow \quad \begin{matrix} [A, B, D] \\ [A, C, D] \end{matrix}$$

Lastly, since levels can have multiple chains, the chain property in the level configuration file is a list of lists of lists.

4.4.2 *The ChainTree Class (C#)*

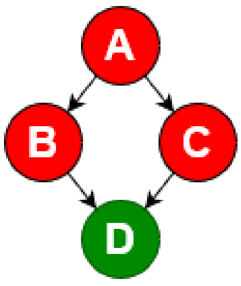
The Chain Tree Class is the game's internal representation of these chains, it stores them, and every time there is a new shape, it verifies whether it is correct or not.

The more intuitive solution would be to store each individual sequence in a list and then check each one every time there is a new shape. However, that would be inefficient, in terms of memory allocation and computational speed since, in the worst case, it would need to go through each of these lists.

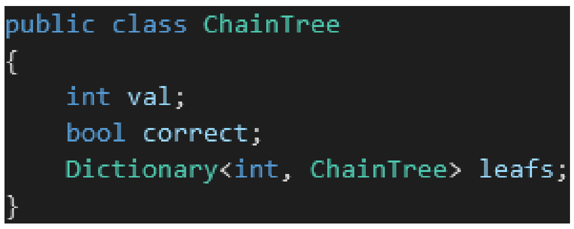
Instead, the ChainTree class implements a Rose Tree[1] structure. Specifically it has three components, shown in Figure 38:

- **Shape ID**: An integer that uniquely represents the shape. This made from the shape's name.
- **Correctness**: A boolean that indicates whether this shape is correct or not.
- **Next Shapes**: A Dictionary[2] that maps to the next shapes on the chain, based on their ID.

Figure 38a also illustrates the example used in the previous chapter as a diagram.



(a)



```
public class ChainTree
{
    int val;
    bool correct;
    Dictionary<int, ChainTree> leafs;
}
```

(b)

Figure 38: ChainTree: (a) a diagram example and (b) the definition of the ChainTree class.

Verifying whether a list of shapes is a chain goes as follows: the first element gets checked against the root; if there is an entry in its dictionary for that ID, then the process repeats. Either the list ends and the node indicates correctness, or the dictionary has no entry, meaning an invalid chain.

The number of rounds before a correct shape appears is defined in the level configuration file in the **forceCorrectEveryNth**. Normally, this is done by picking a correct shape, and showing when this Nth round is reached. However, when dealing with long chains, this becomes more challenging to enforce. The answer to this is presented bellow and illustrated in Figure 39:

1. The GameManager creates a list consisting of around 2 millions of shapes, chosen at random. This is large enough that a game could go for months without seeing the end of the list.
2. The TreeChain class instance, checks this list for any naturally occurring correct shapes, according to the chain, then computes the gap size between them.

1 A tree data structure with a variable number of branches per node.
2 The C# implementation of a HashMap

3. Where there are big enough gaps, the class fills with random chains, assuming they fit in this space.

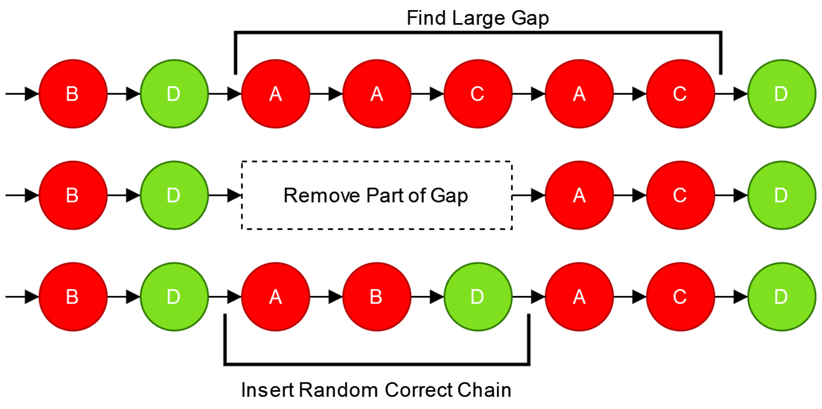


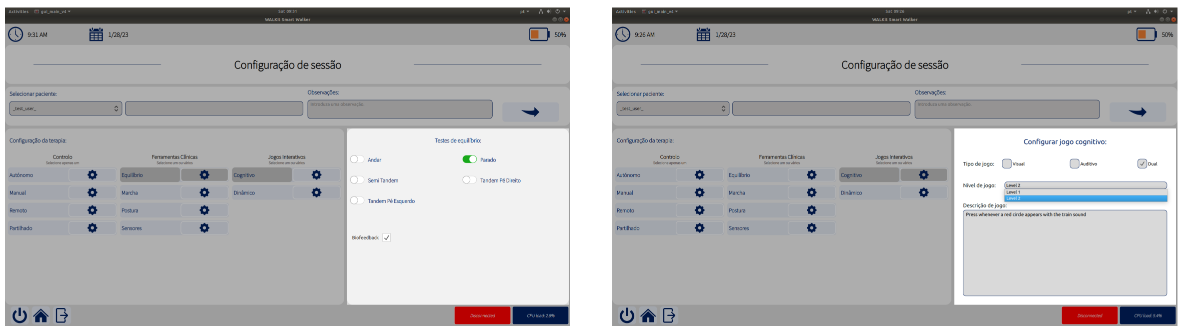
Figure 39: Diagram of ChainTree filling a gap with a chain.

This approach ensures that some shapes are truly random, and that there are not long gaps of incorrects, which could be demotivating. One downside of this approach is the overhead[3] incurred when creating the list and populating it with correct shapes. On the other hand, this method is faster at runtime, since the game only needs to check the generated list once per round.

4.5 WALKIT ALGORITHM

4.5.1 *Development*

The game itself is only half of the process of creating a SG for WALKit. The other half is developing the algorithm, which is used to configure the game to suit the patient's needs, and launch it, as seen in Figure 40a. Additionally, the WALKit's API gives them several functionalities, such as access to sensor data or patient ID. Fortunately, they did not have to be built from scratch but instead made based on other existing algorithms.



(a)

Sat 09:31 — 9:31 AM — 1/28/23 — 50%

Selecionar paciente: _test_user_

Observações: Introduza uma observação.

Testes de equilíbrio:

- Andar
- Parado ✓
- Semi Tandem
- Tandem Pé Direito
- Tandem Pé Esquerdo

Biofeedback ✓

CPU load: 2.8%

(b)

Sat 09:26 — 9:26 AM — 1/28/23 — 50%

Selecionar paciente: _test_user_

Observações: Introduza uma observação.

Configurar jogo cognitivo:

Tipo de jogo: Visual Auditivo [✓ Dual]

Nível de jogo: Level 2 (Level 1, Level 2)

Descrição de jogo: Press whenever a red circle appears with the train sound

CPU load: 5.4%

Figure 40: Examples of WALKit algorithms' configuration pages; (a) The balance algorithm; (b) The cognitive game algorithm.

3 This term refers to large amounts of computation done at the beginning of a task.

The Cognitive Game Algorithm, as illustrated in Figure 40b allows the user/therapist to choose the game mode and select a specific level within that mode. Additionally, the text box displays the currently selected level's instructions.

Internally, the algorithm accomplishes this by scanning the levels folder for any Cognitive Game level files, displaying them in the proper category. When the session starts, the algorithm launches the game with the parameter `--jsonpath <path>`, telling the game where the level file is. After verifying that the game successfully launched, the algorithm then resizes it to fit within the display box, exemplified in Figure 41.

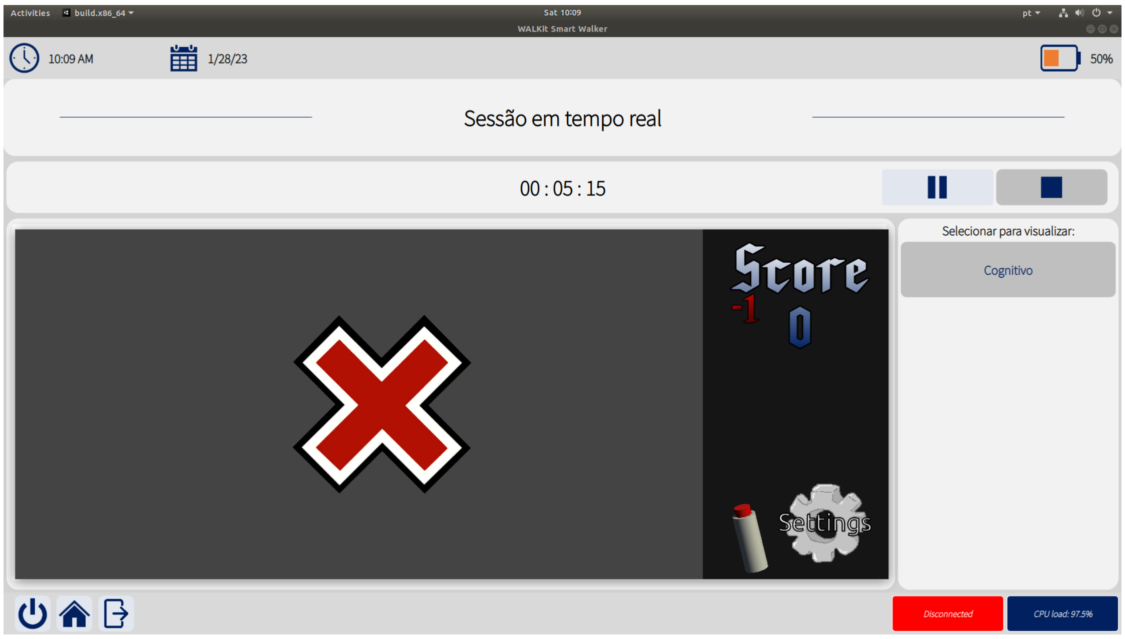


Figure 41: Instance of the Cognitive Game being played on the WALKit app.

4.5.2 *WS Library Choice*

The ability to communicate with the game through WebSockets is a crucial aspect of the game's algorithm, and there are multiple libraries available for this purpose. The *easywsclient* [72] library was chosen because it is easy to install and has no dependencies. While others had more features, they also required additional libraries and drivers to be installed, which would have made the environment needlessly more complex.

Afterwards, a wrapper[4] was built to provide game-specific functionality. This includes sending stop, resume and pause signals to the game, and buffering incoming messages. Furthermore, because the game does not start immediately, this class attempts to connect to it multiple times for over 30 seconds or until the algorithm gets stopped manually.

After the game is launched, its window needs to get fitted to the algorithm space, i.e., the rectangle where algorithms get automatically placed. Some external tools were required to do this, namely xdotool, wmctrl, and xprop. These are not traditional libraries but rather a collection of Linux commands. Instead of using these directly,

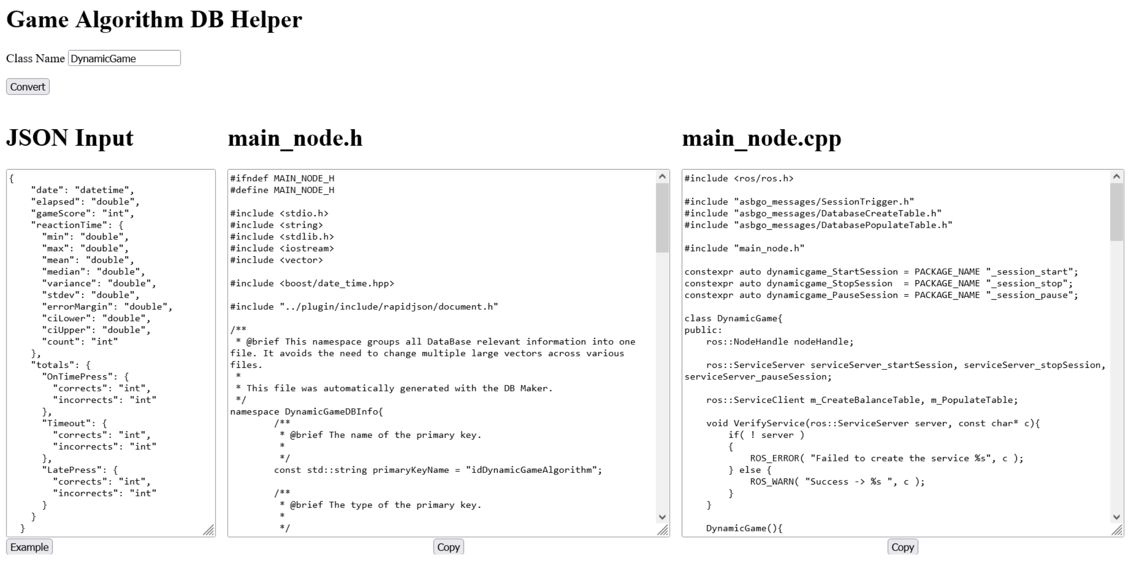
4 Abstractions that simplify or tailor code to a desired purpose.

two Bash scripts were created: one is responsible for resizing the window, and the other is responsible to bring the windows to the foreground in case the user presses outside the game. Furthermore, these Bash scripts only get called once the game successfully connects to the WS client.

4.5.3 *Database Integration*

As detailed in Section 4.2, the game counts the user's reaction time to a specific stimulus, considering the level it is playing and uses this information to calculate and save several metrics in the WALKit's database, associated with both patient and session ID. These metrics include total score, mean reaction time, standard deviation, maximum and minimum reaction time, and limits of the 95% confidence interval.

These metrics were exported by the game in JSON format. Thus, to save them in the WALKit's database, which consists of one MySQL database for structured data and one MongoDB database for non-structured data, a tool was designed to convert JSON data structure into a C++ header file with all variables organised. Figure 42 displays an example of the WALKit database's marker tool.



```
{
    "date": "datetime",
    "elapsed": "double",
    "gameScore": "int",
    "reactionTime": {
      "min": "double",
      "max": "double",
      "mean": "double",
      "median": "double",
      "variance": "double",
      "stdev": "double",
      "errorMargin": "double",
      "ciLower": "double",
      "ciUpper": "double",
      "count": "int"
    },
    "totals": {
      "OnTimePress": {
        "corrects": "int",
        "incorrects": "int"
      },
      "Timeout": {
        "corrects": "int",
        "incorrects": "int"
      },
      "LatePress": {
        "corrects": "int",
        "incorrects": "int"
      }
    }
  }
```

```
#ifndef MAIN_NODE_H
#define MAIN_NODE_H

#include <stdio.h>
#include <string>
#include <stdlib.h>
#include <iostream>
#include <vector>

#include <boost/date_time.hpp>

#include "../plugin/include/rapidjson/document.h"

/**
 * @brief This namespace groups all DataBase relevant information into one
file. It avoids the need to change multiple large vectors across various
files.
 *
 * This file was automatically generated with the DB Maker.
 */
namespace DynamicGameDBInfo{
        /**
         * @brief The name of the primary key.
         *
         */
        const std::string primaryKeyName = "idDynamicGameAlgorithm";

        /**
         * @brief The type of the primary key.
         *
         */
```

```
#include <ros/ros.h>

#include "asbgo_messages/SessionTrigger.h"
#include "asbgo_messages/DatabaseCreateTable.h"
#include "asbgo_messages/DatabasePopulateTable.h"

#include "main_node.h"

constexpr auto dynamicgame_StartSession = PACKAGE_NAME "_session_start";
constexpr auto dynamicgame_StopSession  = PACKAGE_NAME "_session_stop";
constexpr auto dynamicgame_PauseSession = PACKAGE_NAME "_session_pause";

class DynamicGame{
public:
    ros::NodeHandle nodeHandle;

    ros::ServiceServer serviceServer_startSession, serviceServer_stopSession,
serviceServer_pauseSession;

    ros::ServiceClient m_CreateBalanceTable, m_PopulateTable;

    void VerifyService(ros::ServiceServer server, const char* c){
        if( ! server )
        {
            ROS_ERROR( "Failed to create the service %s", c );
        } else {
            ROS_WARN( "Success -> %s ", c );
        }
    }

    DynamicGame(){
```

Figure 42: The *WALKit DB Maker* tool.

4.6 GAME VALIDATION

Validation is present in both software development and clinical rehabilitation. The first focuses on whether the software functions adequately and the intuitiveness of its user interaction experience, and the latter concerns the effects of the treatment on the patient, be it physical or mental. According to the literature, there are two forms of clinical validation: qualitative and quantitative. The former evaluates the physical advancements of patients through tests, while the latter emphasises the opinion and mental state throughout the trial.

Ideally, to verify the game's effectiveness, clinical tests would be conducted, which require two groups of patients with gait ataxia, one control group and one experimental group. Due to time constraints and the lack of ataxic patients, the tests were restricted to healthy participants, with one session each.

4.6.1 *Participants*

Twelve healthy volunteers (5 females and 7 males) participated in the validation of the proposed cognitive game. The validation was performed under the ethical procedures of the Ethics Committee in Life and Health Sciences (CEICVS 147/2021), following the Helsinki Declaration and the Oviedo Convention. All participants accepted voluntarily to be part of the study. Data were collected at the School of Engineering of the University of Minho.

4.6.2 *Validation Protocol*

The validation consisted in performing a 5-minute trial walking with the robotic walker at 2.0 km/h in a long corridor of the Department of Industrial Electronics. At the same time, the participants played the game at level 5, which consisted of reacting whenever a ball or the colour red appeared. As the hardest level to date, it felt appropriate when considering healthy participants, since the other levels may be overly easy and, therefore, monotonous. Figure 43 illustrates a participant walking with the robotic walker and playing the cognitive game.

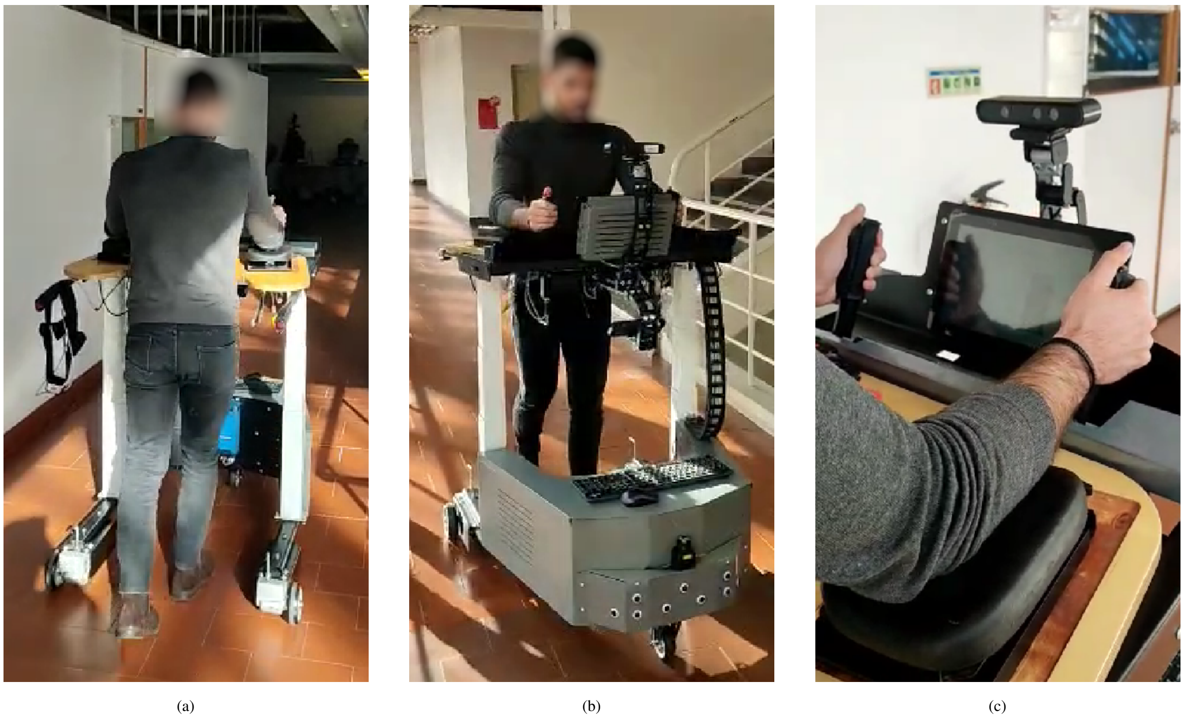


Figure 43: Cognitive Game validation: (a) and (b) Driving the walker, while playing the Cognitive Game; (c) Close up of arms while driving and controlling the walker.

4.6.3 *Validation Metrics*

After performing the 5-minute walking trial, each participant fulfilled a questionnaire, which was divided into three parts. The first part collected demographic information from the user, namely age, gender, and any possible impairments, although all participants were healthy. The second part consisted of a System Usability Scale (SUS), which was adapted to the context of the serious game, considering its European Portuguese version presented in [73]. The third and last part consisted of open-ended questions, encouraging participants to express their thoughts and opinions on which parts of the game were lacking and any suggestions on improving them.

The SUS is a dependable tool for assessing usability. It consists of a ten-item questionnaire with five response options, ranging from "strongly agree" to "strongly disagree". According to study [74], a system is considered above average at 68 points or more. Moreover, this same study also correlated adjectives to SUS score, as illustrated in Figure 44.

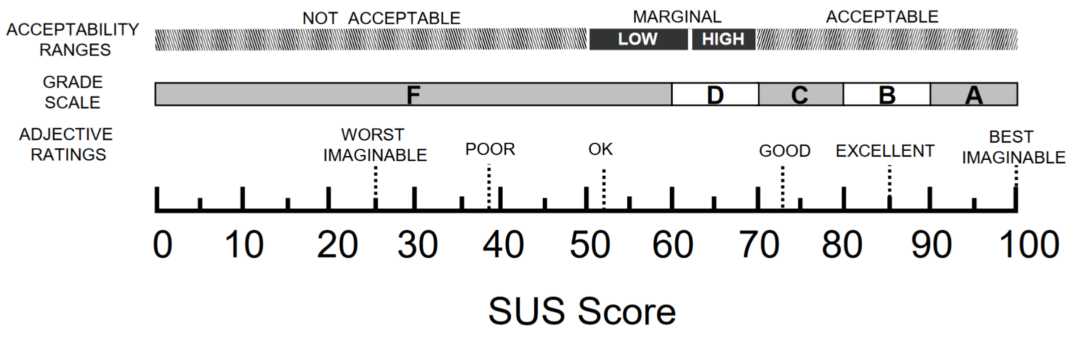
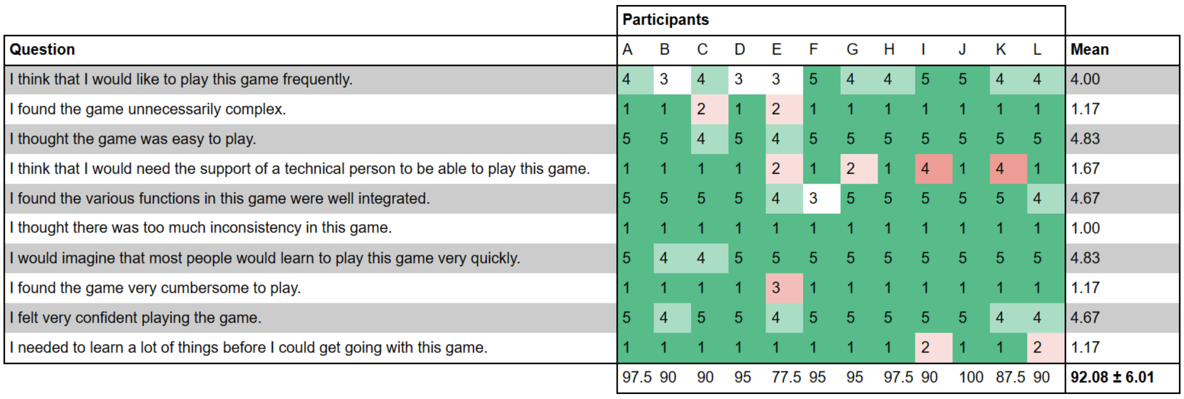


Figure 44: The correlation between adjectives and SUS score. Retrieved from [74].

4.7 RESULTS AND DISCUSSION

4.7.1 *System Usability Scale*

As mentioned in section 4.6.3, participants were asked to fill a SUS form. Figure 45 shows each respective answer, as well as the average value of each answer.



	Participants												
Question	A	B	C	D	E	F	G	H	I	J	K	L	**Mean**
I think that I would like to play this game frequently.	4	3	4	3	3	5	4	4	5	5	4	4	4.00
I found the game unnecessarily complex.	1	1	2	1	2	1	1	1	1	1	1	1	1.17
I thought the game was easy to play.	5	5	4	5	4	5	5	5	5	5	5	5	4.83
I think that I would need the support of a technical person to be able to play this game.	1	1	1	1	2	1	2	1	4	1	4	1	1.67
I found the various functions in this game were well integrated.	5	5	5	5	4	3	5	5	5	5	5	4	4.67
I thought there was too much inconsistency in this game.	1	1	1	1	1	1	1	1	1	1	1	1	1.00
I would imagine that most people would learn to play this game very quickly.	5	4	4	5	5	5	5	5	5	5	5	5	4.83
I found the game very cumbersome to play.	1	1	1	1	3	1	1	1	1	1	1	1	1.17
I felt very confident playing the game.	5	4	5	5	4	5	5	5	5	5	4	4	4.67
I needed to learn a lot of things before I could get going with this game.	1	1	1	1	1	1	1	1	2	1	1	2	1.17
	97.5	90	90	95	77.5	95	95	97.5	90	100	87.5	90	**92.08 ± 6.01**

Figure 45: SUS scores from the Cognitive Game.

According to Figure 45, the average SUS score was 92.08 ±6.01, which falls in the category of "Excellent" to "Best imaginable", according to Figure 44. This result is indicative of the program's intuitiveness. The lowest score attributed was 77.5, which still falls comfortably in between good and excellent, meaning that, at its worst, the game is still good.

4.7.2 *Open-ended Questions*

The last portion of the questionnaire focused on the game aspects of the game. The upside to these questions is how they allow participants to freely express what they thought was less than ideal and any possible solutions. Of course, the answers are open to interpretation, as characteristics such as fun and challenges are not perfectly quantifiable.

There were five questions (Q) in total (some of which had follow-up questions). These were: i) "Did you find the game to be fun?"; ii) "How challenging did you find the game?"; iii) "Did simultaneously walking and playing make the game harder?"; iv) "Do you think the game can be helpful for rehabilitative purposes?"; and v) "What could be improved about the game?". An analysis to the participants' answers is given bellow.

Q1: Did you find the game to be fun?

This question is simple yet vital; the onus of the dissertation rests on serious games being fun to play and therefore motivating patients to persist with therapy. Ten of twelve participants (which corresponds to **83%** of the total sample) found the game fun, so we can assume that the game is, at some level, motivating. If the majority thought otherwise, then it would potentially have to be re-examined and redesigned.

Q2: How challenging did you find the game?

There is a balance to be had with game difficulty: too little, and it becomes monotonous, while too much can be frustrating. Figure 46 shows that three participants found the game not challenging at all, while the other nine thought it was challenging yet fair.

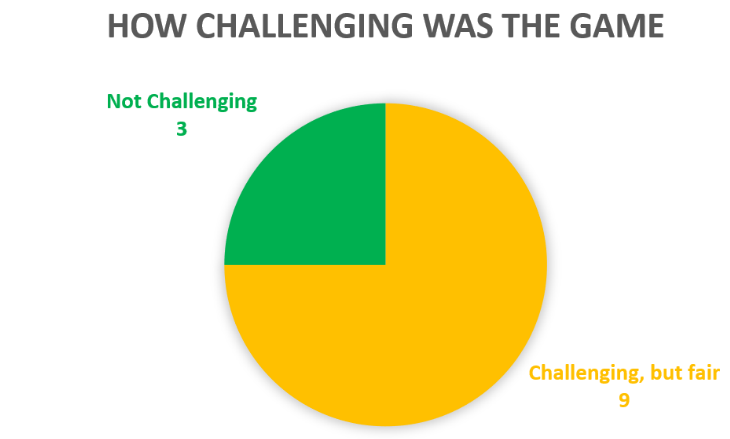


Figure 46: Pie Chart of "*How Challenging Did You Find the Game*".

Challenge is perceived differently by each individual. However, **75%** thought the game to be balanced, i.e., suitable in its current iteration. Additionally, minor difficulty adjustments are also possible. By creating new levels or configuring old ones, therapy sessions can get tailored to each patient.

When asked what specifically made the game challenging, participants had various reasons. Using these, the previous chart was refined down to the following Figure 47.

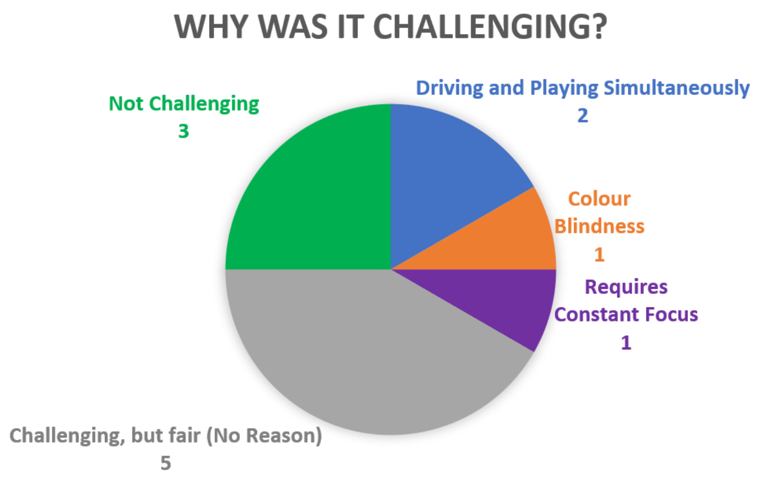


Figure 47: Pie Chart of "*Why was it challenging*", a follow-up question to "*How Challenging Did You Find the Game*".

Q3: Did simultaneously walking and playing make the game harder?

Around **55%** of players indicated that walking did not make the game harder to play but rather the opposite, where playing made walking more difficult. Another **18%** agreed with the original statement, while the rest felt no increase in difficulty.

While the specific reason might differ, the majority agrees that performing both together was more arduous than each one individually.

Q4: Do you think the game can be helpful for rehabilitative purposes?

The response was unanimously in favour of the rehabilitative potential of the serious game, with three different opinions on why. Some attributed it to the motivation of playing a fun game, while others identified dual-tasking as the driver of rehabilitative progress. Finally, some gave no further explanation.

Q5: What could be improved about the game?

Regarding this question, the participants suggested some points in which the cognitive game could be improved, namely:

- Have shapes appear at a faster rate. Three participants shared this sentiment.
- Support for persons with colour blindness.
- Have the game launch in full-screen.
- Background music, to improve immersion.
- Clarify that the Correct/Incorrect Indicators are not shapes.

These are valid suggestions that will be considered in the future iteration of the cognitive game. Regarding the first suggestion, this parameter can be adjusted by the clinician in charge. Regarding the second suggestion, the aim of this game is for the user to react to a certain visual or sound stimulus. Depending on the level chosen, the visual stimulus may include the reaction to certain colours which, as a user with colour blindness, may make it difficult for he/she to perceive colour. Therefore, the future iteration of the game may only include geometric shapes, rather than colours, for persons with colour blindness. Concerning the third suggestion, this was not considered during the development of the game because one of the requirements was to integrate the game into the walker's application and, therefore, only use the designated space for that. Still, the game could be played on full-screen by maximising the corresponding window. Regarding the fourth suggestion, indeed, the addition of music could make the experience more immersive. However, the sound could confuse the user, as there is a sound-only game mode. Finally, regarding the last suggestion, a green tick was associated with correct answers and a red cross for wrong answers. However, no indication was given to the user at that point. In this way, the next iteration of the cognitive game could indicate that information in the description of each level of the game.

4.8 GENERAL CONCLUSIONS

This chapter presented the design process and development of the first game of this dissertation, the cognitive game. The resulting game is functionally identical to the original; all levels can be replicated perfectly, except the newly added UI. However, the true benefit of this re-design lies in its extended functionality: highly customizable shapes, scores, and the new chain mode, to name a few. Additionally, all unimplemented levels can be made with no programming knowledge and can even be configured without the need to rebuild the game.

As shown in this section, most of the work on this algorithm revolved around constructing new tools. As such, the development of future game algorithms should be considerably faster. That extends to any other future algorithms that may need to integrate external applications besides games.

Finally, testing revealed that the game is fun and intuitive while successfully emulating the dual-tasking faced through everyday walking. Although the testing procedure was performed with healthy participants, this fact should not affect the SUS portion of the questionnaire since it is mainly about the game's intuitiveness rather than the game experience itself. However, healthy participants might perceive challenge and fun differently than those with balance impairments, that is, the game might be more difficult than initially perceived from these results. Due to this, future work should include trials with the target demographic.

5 GAME DEVELOPMENT: DYNAMIC GAME

5.1 INTRODUCTION

The dynamic game was the second game developed under this dissertation. The idea behind this game is to control an avatar while walking, mimicking the user's behaviour, with the goal of maintaining the user's posture correct. Moreover, different mini-challenges are proposed during the game, in which users have to perform specific exercises that incite their balance control.

In terms of complexity, this serious game supersedes the previous one, by presenting multiple levels, 3D shapes, and different types of controls. This chapter presents the conceptualisation of the game, including the game design, the development phase, and validation, with the respective results and discussion.

5.2 GAME DESIGN PHILOSOPHY

The development started with finding a concept to base the game around. The idea is that, without a motif, a game can feel overly clinical and not fun, whereas a simple theme can add to the immersion [33, 40]. These can be seen in almost every commercial game where, for example, players simulate the experience of playing sports.

The game's purpose has already been established, which is to improve the user's balance in two different ways: by helping them maintaining a correct posture and walking balance, and the second is to exercise their balance control by performing certain movements that cause weight transfer.

To build on that, it got decided that the game's motif is a farmer carrying a stick with two pails of water on their back. If the player leans too much in one direction, the respective bucket will slowly drain out, thus encouraging a balanced posture. The goal is to reach the end without running out of water in each bucket. Finally, there is a chance to encounter events while walking, which the user must overcome through active movement.

Those two aspects are the core of the game. However, to reduce the monotony of "walking" through the same scenario repeatedly, distinct levels were introduced, each with a specific theme. Currently, there are two themes, a forest and a desert, with the potential to develop more in future. Additionally, levels may diverge into multiple choices, e.g., after reaching the end of the forest, the player would be able to choose between the desert or a possible swamp level. That approach adds variety to the game since the only way to see both levels is to play at least twice. As the number of levels increases, so does the replay value, one of the most significant factors in therapy [22].

In order for the game to be expanded upon in the future, levels can be adjusted or even created with no code required. The game's design focuses on the visual editor, so it is only dragging and dropping. Additional features, such as new events, will not require in-depth knowledge of the game, only general programming. This is possible through class abstraction, where the parent class implements all functionality and component interoperability. In this way, anyone creating a new event can focus solely on the game-play aspect of it.

5.3 GAME MAIN COMPONENTS

Aside from the Main Menu, the game is comprised entirely of levels, e.g., forest and desert, with the possibility of more. They all follow the same structure: four essential GameObjects, each with specific tasks and degrees of persistence[1]. Additionally, each one is part of a larger system within the game, which are:

- **YBot** - the player's character model. Specifically, the "Player Character Controller" that moves and animates this model.
- **Assembler** - which uses the "Level Building" part of the game.
- **Persistent Loader** - It houses all the Input/Output components and settings menus.
- **GameManager** - which manages the current game session.

Figure 48 shows the Forest Level with each of GameObjects, and an additional Light Source.

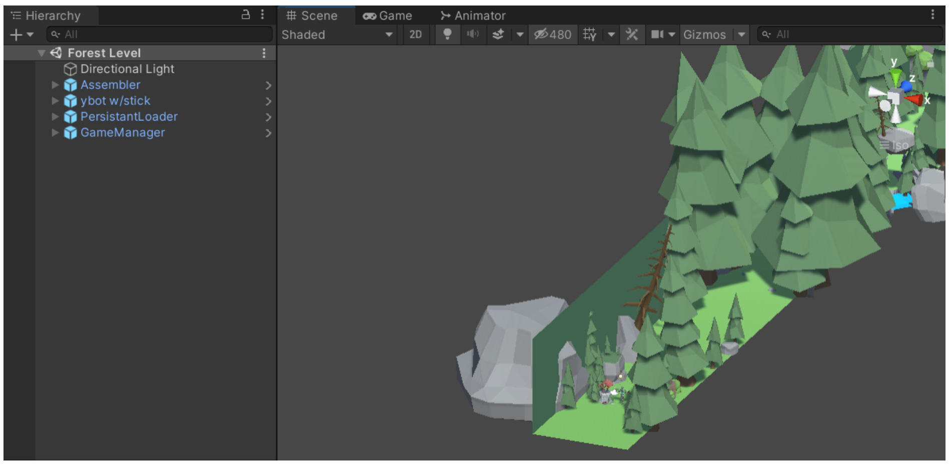


Figure 48: The Forest Level, featuring all main Components.

Finally, this section also details the "Events" system. Events are specific activities that exercise the user's balance control.

1 Whether objects get destroyed when their Scene unloads.

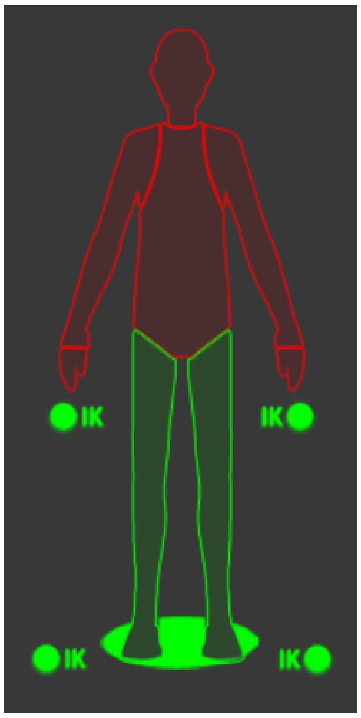
5.3.1 *Player Character Controller*

The first component is the **Player Controller**, which reads the user's input and animates the player character accordingly. In Unity, animating requires little to no code and offers features like Avatars and Blending. Avatars restrict animations to specific body parts. For example, Figure 49b shows the "Arms and Torso Avatar" that, when used with a walking animation, would result in a model that would only move and sway its arms.

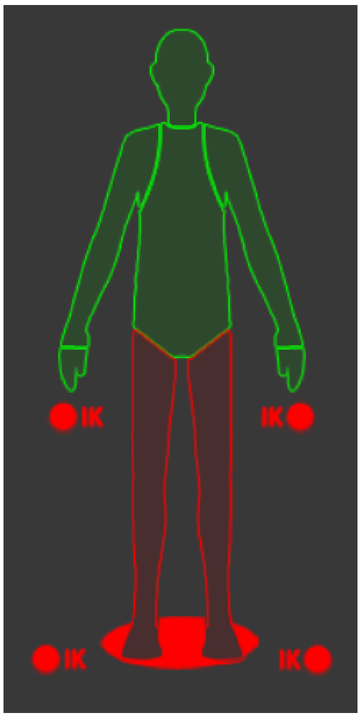
As shown in Figure 49c, the player is essentially divided into three parts:

"Legs": controls the walking animations. "Arms and torso": controls the leaning and object-holding animations. "Torso": used with the walking animations to introduce some sway to the upper body while walking.

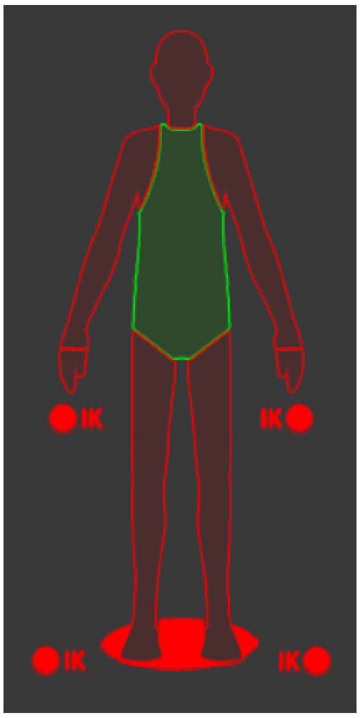
These are all layered together in the Editor, resulting in a player model walking while holding the buckets.



(a) Legs Avatar



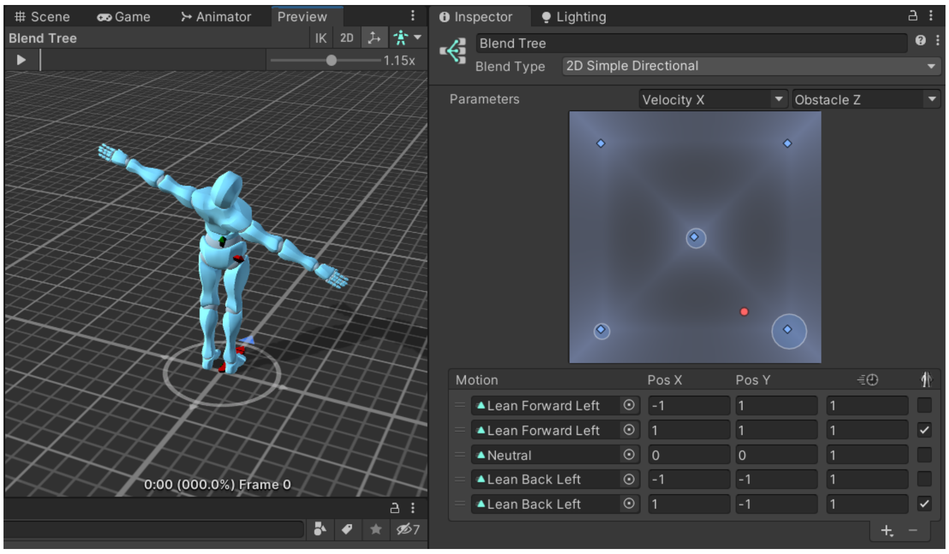
(b) Arms and Torso Avatar



(c) Torso Avatar

Figure 49: The different avatars used to animate the player.

Animation Blending, or **Lerping**, refers to when two animations are mixed based on one or more variables, significantly reducing time spent animating. For instance, the leaning animation results from blending a fully bent pose and an upright one using a float value. It can also smoothly transition between, for example, idling and walking.



Blend Tree
Blend Type: 2D Simple Directional
Parameters: Velocity X, Obstacle Z

Motion	Pos X	Pos Y		
Lean Forward Left	-1	1	1	
Lean Forward Left	1	1	1	✓
Neutral	0	0	1	
Lean Back Left	-1	-1	1	
Lean Back Left	1	-1	1	✓

Figure 50: An example of Animation Blending.

Figure 50 illustrates the character both leaning right and back by blending the "Neutral" animation with the "Lean Back Left" animation and it's mirrored version. Additionally, there is only one leaning animation, which the "Animation Controller" can mirror automatically, avoiding unnecessary duplicates.

5.3.2 *Level Building*

Half of the game consists of walking along a straight path, which can get monotonous quickly. To prevent this from happening, the levels get randomly generated each time, providing a slightly different experience in every session. There are two ways to do this. First is randomly placing props, like trees and rocks, along the path. This is faster to design but runs the risk of looking very unappealing. The second approach is Lego-like building blocks of terrain, all handcrafted. This is more time-consuming, but produces more cohesive and aesthetically pleasing levels. The second option was chosen since, ultimately, it is capable of producing a more visually appealing game, which is a strong selling point of commercial games.

5.3.2.1 *Nodes*

Nodes are the in-game implementation of the blocks of terrain, and they provide ways to connect to other Nodes. There are three types of Nodes: the Node itself, "Start", and "End". In terms of functionality, they are identical, except "Start" and "End" are Node subclasses. This distinction is primarily due to how the visual editor works; some variable slots can specifically request a start or end node, ensuring no invalid levels get accidentally made. Figure 51 exemplifies two Nodes: Figure 51a is a base Node and Figure 51b illustrates a fully decorated Node that is ready to be used in the game.

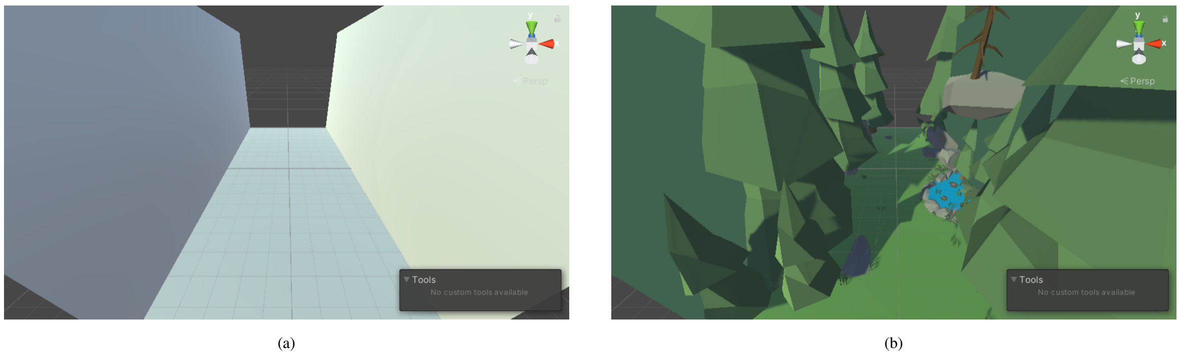


Figure 51: Two examples of a Node in the Dynamic Game: (a) the base Node and (b) a fully decorated Node.

Nodes come in different forms. There is the basic version that only has the necessary parts, and then there are variations for each level. Fully decorated Nodes are built on top of these variations. This is made possible using prefabs, and as mentioned in section 3.4.6, all variations will be affected by any changes made to the default version, which can be helpful if a group of Nodes needs to be adjusted. As an example, Figure 52 shows two variations of Nodes from different levels. The "Desert" Node variation, shown in Figure 52b, has no walls, so any Nodes based on it will have the same layout.

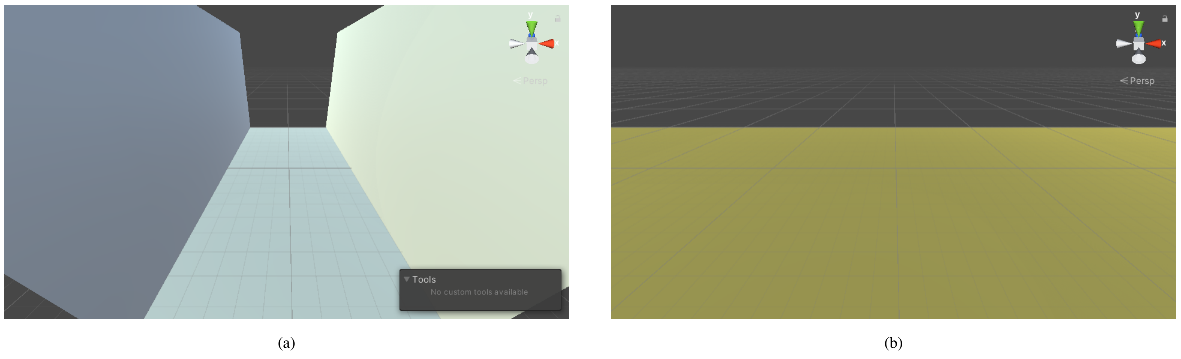


Figure 52: Two Nodes from different levels: (a) the root Node and (b) the default desert Node.

5.3.2.2 *Assembler*

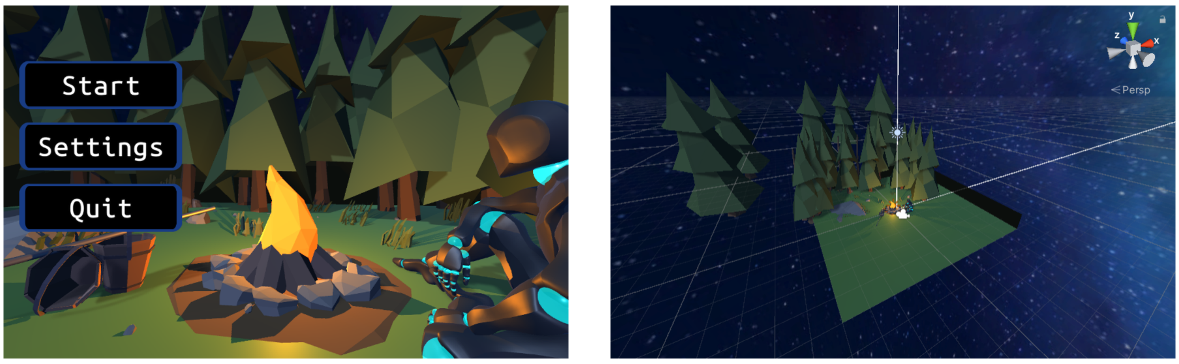
Every Node has a Transform called *AnchorPoint* that slots an *ObstacleAnchor*[2], ensuring the two models do not clip[3] into each other. The *Assembler*, as the name implies, uses Nodes and *ObstacleAnchors* to build levels by picking a random "Start", an "End", and filling the space between with other Nodes. Additionally, the Assembler Component builds the levels dynamically, with each Node being placed as the player progresses. Each Node can have a random event slotted in, also at random, depending on the event frequency previously assigned to the level. These events get picked from a list available to that level.

2 The term "Obstacle" is used internally to refer to Events
3 When two models, usually 3D, overlap with each other.

5.3.2.3 *Level Design Workflow*

As previously mentioned, designing the game considering its future growth is crucial, not just to maintain the current game, but to add new features, namely levels. For this reason, new levels are possible with no programming, although end-of-level animations would be missing or re-used. The following list details a typical workflow for creating new levels, excluding additional events.

1. Create Node variations for Start, Node, and End.
2. Decorate at least one Start and End, and create multiple Nodes:
   a) Ideally, all of these should be clearly distinct. More Nodes are always better as it decreases repetition.
   b) The End Nodes must have level transitions, otherwise the player cannot progress. It should merge this transition and the next level's environment to increase the user's immersion.
3. Create event variations for the level's theme.
4. Create a new scene from the template and adjust any variables needed. These variables can be, but are not limited to:
   a) The ambient lighting. For example the desert level has a yellow tint to it.
   b) The skybox[4] can also be changed, as seen in the main menu's starry sky, shown in Figure 53.



(a) (b)

Figure 53: An example of a skybox in Unity: (a) The starry skybox as seen in-game; (b) The starry skybox, as seen from the editor.

The process itself is straightforward; its only bottleneck is acquiring assets and designing new and diverse environments.

4 A method of enclosing a level in a cuboid, where distant terrain is drawn, e.g., skies, mountains, among others

5.3.3 *Persistent Loader*

The third part is the Persistent Loader. It houses every component that needs to remain active across scenes since, otherwise, every game object gets destroyed when the Scene unloads. They track values across multiple game sessions and, as such, must not be interrupted.

5.3.3.1 *Input/Output*

The **Input/Output (IO)** object contains the same WS and ROS Components from the Cognitive Game, with some minor alterations. It has an additional ROS subscriber that subscribes to the user's torso orientation, which is estimated with an IMU placed on the user's back through an existing algorithm that allows the monitoring of postural balance. This algorithm comprises the implementation of both complementary and Kalman filters, and estimates the user's torso orientation, in degrees, in both mediolateral and anterior-posterior axes.

5.3.3.2 *Options*

The Options class defines the structure and default values for the game options and configures how they get serialised to the JSON file format. It also has a Profile field, which stores the current user's ID. This way, multiple patients can use the walker without having to re-configure between sessions. Currently, the class is structured as such:

- **Audio and Graphics:** Standard game settings.
- **Game:** Game-specific settings such as showing or hiding the timer.
- **Controls:** Which controls are currently in use, and each controller's configuration, such as max and minimum values.
- **Misc:** Miscellaneous settings, like toggling the debug menu or choosing a preferred language.

There's also an additional class, the **OptionsManager** MonoBehavior, which manages an instance of Options and saves or loads any files. Given a profile name, the **OptionsManager** looks for the correct Options file. If there's no file yet, it gets created, else it's loaded into the game. Finally, these settings can be modified in-game through the in-game Settings Menu(see subsection 5.3.3.5), by directly altering the file or via the WS.

5.3.3.3 *Controls*

The purpose of Controls is to provide a single input interface for both conventional inputs, such as keyboards or mice, and the WALKit's sensors, like the IMU and load cells. Since most sensors output values in different units of measurement, the Controls class also normalises and returns these in the [-1, 1] range.

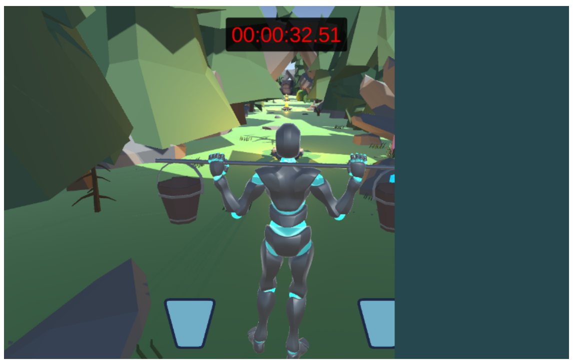
"Controls" works by reading conventional inputs from Unity's Input class and sensor information from ROS#. It then normalises the incoming values by looking to the Options class for configurations like the load cells maximum or minimum values.

This class is a central part of the game and, as such, is called virtually every frame, meaning it needs to be as efficient as possible. The solution is a HashMap that maps a device name to a function that returns the device's value, already normalised. In this way, callers only need to provide the device name to get input values. Adding a new controller is as simple as registering it in this HashMap and creating the corresponding function.

Finally, some sensors can have noticeable noise that, if left unchecked, can lead to stuttery animations, which detract from the gameplay experience. To prevent this, the Kalman Filter Lite [75], made for computational efficiency, was adapted to C# and applied over the incoming values.

5.3.3.4 *Transition Manager*

The **Transition Manager** handles changing from one scene to the next by transitioning into and out of a loading screen, where it waits for the next one to load. Figure 54 illustrates an example of transition, which covers the screen with a loading screen (Figure 54b) from right to left, as depicted in Figure 54a.



(a)

(b)

Figure 54: The Transition Manager, actively transitioning between scenes: (a) example of a transition, covering the screen from right to left, and (b) a loading screen.

5.3.3.5 *Settings Menu*

The settings menu provides an in-game UI to configure the game's options without editing code or files. This menu also allows users to configure sensors, such as the IMU and load cells, which have to be calibrated for "Controls" to normalise them properly. However, manually configuring them is repetitive and cumbersome, not to mention time-consuming. In light of this, some sensors have an "**Auto-Config**" feature, which takes the user through a series of steps, configuring them in seconds. Afterwards, the therapist can fine-tune these manually if necessary. Figure 55 illustrates an example of the Settings Menu.

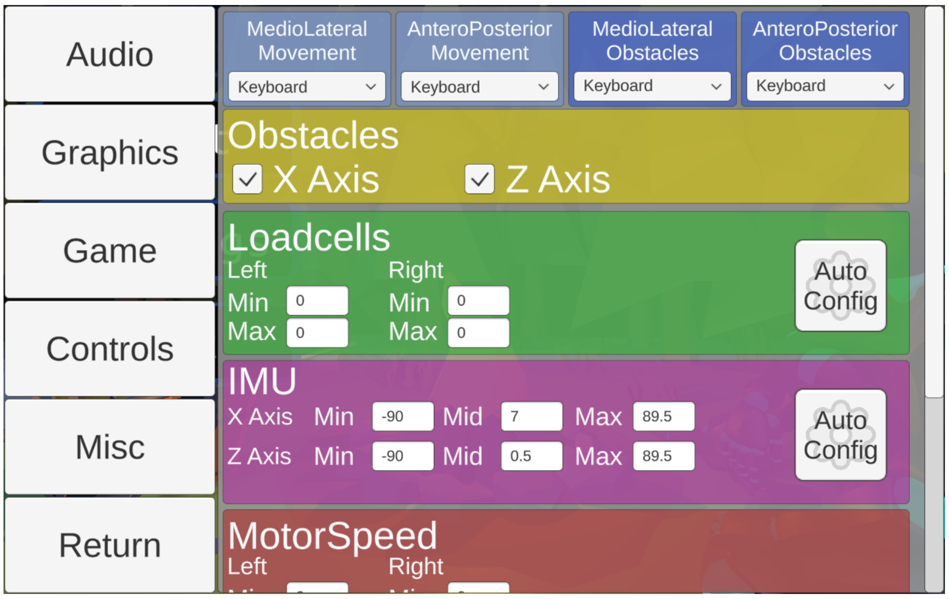


Figure 55: An exemplification of the Settings Menu.

5.3.3.6 *Debug*

Finally, **Debug** displays useful information otherwise only available in the editor. It shows whether both IO components successfully connected with the walker, the launch parameters, and the numerical input from Controls.

5.3.4 *Game Manager*

The **Game Manager** manages all components essential to the current game session, unloading when the last level ends. These include the timer, HUD[5], metrics, and other UI elements pre-loaded for other scripts' use, such as when choosing between two levels. It also acts as a central hub for information: it tracks and exposes several metrics like when a level starts and ends, if the player is in an event, among others. Additionally, those other classes can subscribe to the GameManager, and receive a signal whenever these changes happen, such as the MetricsManager, which subscribes to "EndEvent" to save the event results in the metrics.

5.3.5 *Events*

Events are mini-games the player runs into during the Movement Phase. When this happens, they get transported into the Event, where they perform specific tasks that they have to complete to continue the game. The main goal of these events is to incentive the player to lean for a few moments to exercise their balance control. It is noteworthy that these events are played with the walker at a standstill position, since leaning while walking can be potentially dangerous for the patient, who may exhibit a serious loss of balance. Figure 56 illustrates the

5 Often refers to the UI elements visible during active gameplay, e.g., health bars and mini-maps.

events that were developed under this dissertation. These include: i) "Whac-A-Mole" game, in which users have to drown a mole, by leaning to the right or left, depending on the mole's position; ii) a door event, in which users must lean to the right or left to fill a bucket with water to open the door; iii) the flying carpet event, which is the most difficult because users must lean forwards, backwards, to the right and left to drive a flying carpet and catch the golden rings; and iv) a fire event, in which users must extinguish the fire by leaning water of the bucket.

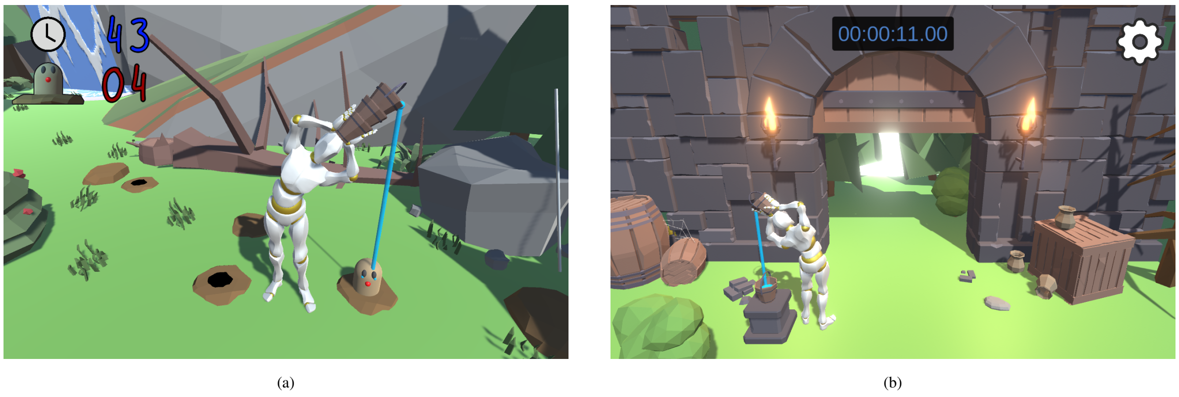


Figure 56: Screenshots of two events currently in the Game; (a) The Whac-A-Mole Event; (b) The Door Event.

Events have two parts: the anchor and the event itself. The anchor stands in the way of the player during the Movement Phase. It looks like a miniature event, so the player is aware and ready for it. They get randomly placed in a node, so every playthrough is different, and when the player touches them, they get teleported into the event.

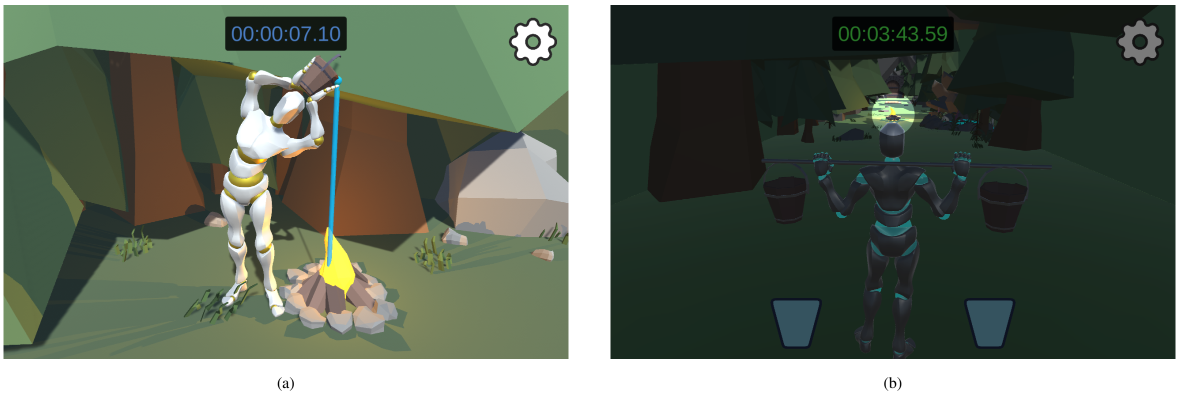
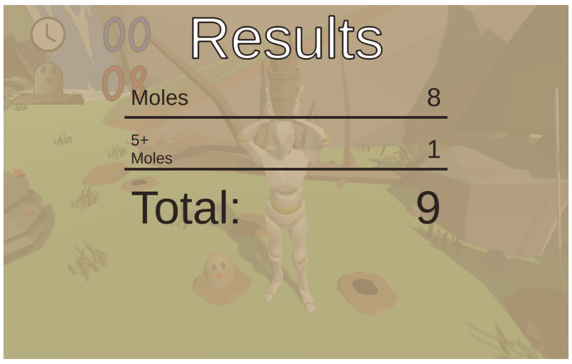
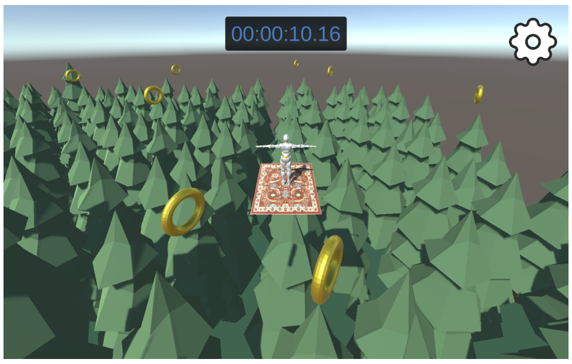


Figure 57: Both parts of the Fire Event; (a) Anchor; (b) Event.

Events translate the users' movements directly to the player character on screen. They typically involve completing a task as fast or as often as possible within a time frame. In the end, a tally of objectives gets shown alongside a final score, shown in Figure 58a.



(a)



(b)

Figure 58: (a) Score tally; (b) The Rug Event.

Internally, events are classes that inherit the *ObstacleEvent*, which implements the scoring system, and the *End()* method that indicates the Event has finished and returns the player to Movement Mode.

Events can be automatically mirrored by inverting the camera's projection matrix on the X-axis, automatically adding extra variety to the events.

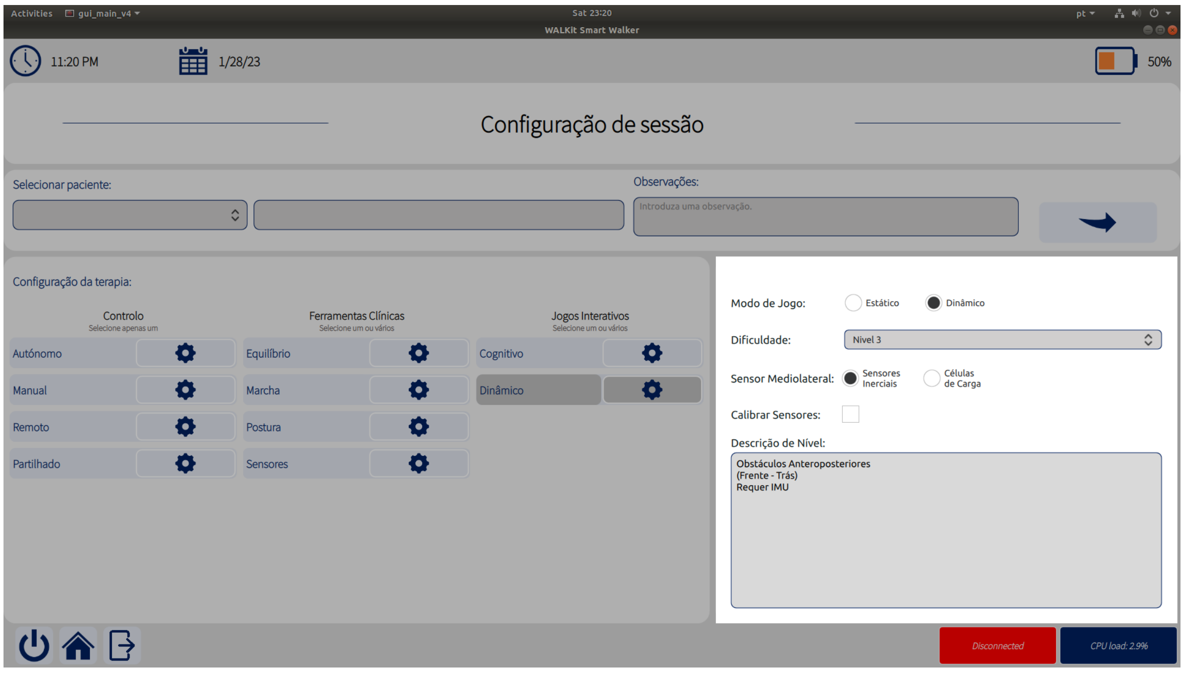
The process for developing a new event is as follows:

1. Conceptualise the game alongside a possible control scheme.
2. In a new scene, place everything required for the event like, for example, the player model.
3. Program it as a standalone mini-game.
4. Subclass it to *ObstacleEvent*, and include walker control support, and scoring.
5. Create the anchor without decorators; it should visually indicate the event it belongs to.

5.4 WALKIT ALGORITHM

This section covers the development of the Dynamic Game's WALKit algorithm, a modified version of the Cognitive Game algorithm. As such, the two algorithms are near-identical, except for some variables, an additional class, and the configuration and runtime pages.

The largest difference is the configuration page, which leads to all the others. This is because the Cognitive Game had several levels to choose from, while the Dynamic Game always starts on the forest level, with players deciding the next levels in-game. This leads to the configuration page shown in Figure 59.



Modo de Jogo: Estático [✓ Dinâmico]

Dificuldade: Nivel 3

Sensor Mediolateral: [✓ Sensores Inerciais] Células de Carga

Calibrar Sensores: ☐

Descrição de Nível:

Obstáculos Anteroposteriores
(Frente - Trás)
Requer IMU

Figure 59: The Dynamic Game Algorithm configuration page.

Figure 59 illustrates an entirely different configuration page from the Cognitive Algorithm. There are several possible configurations, which are, in order:

- The Game Mode. It can either be **Static** or **Dynamic**, dictating whether the game character moves according to the walker's speed (**Dynamic**) or not (**Static**).
- The difficulty decides the type of events that appear during gameplay. These are:
  - **Level 1**, which disables all events.
  - **Level 2** enables mediolateral-controlled events or side-to-side.
  - **Level 3** enables anteroposterior, or front-to-back, events.
  - **Level 4** enables all events, including those played on both axes.
- Mediolateral sensor. Currently, this can either be the IMU or the load cells.
- Calibrate Sensors, which, if ticked, will prompt the user to use the *Auto-Config* feature (see section 5.3.3.5) at the start of the game.
- Level description, which gives information about the current configuration, such as the events enabled, and sensors required.

Sending options to the game at launch involves a few steps. The Cognitive Game algorithm only needs to send the level file path, so it can be done through launch parameters. However, since there is a lot of information to be shared, WebSockets were used instead. This method has the added advantage of being easily expandable.

To help organise and manipulate this information, an additional "Options" class was created, which the configuration page uses to keep track of selected options. When a user starts the session, the current "Options" instance is serialised and sent to the game, changing its settings at the start.

5.5 GAME VALIDATION

5.5.1 *Participants*

Ten healthy volunteers (4 females and 6 males) participated in the validation of the proposed dynamic game. The validation was performed under the ethical procedures of the Ethics Committee in Life and Health Sciences (CEICVS 147/2021), following the Helsinki Declaration and the Oviedo Convention. All participants accepted voluntarily to be part of the study. Data were collected at the School of Engineering of University of Minho.

5.5.2 *Validation Protocol*

A 5-minute trial walking with the robotic walker at 2.0 km/h was performed as part of the validation in a long corridor of the Department of Industrial Electronics. However, participants did not drive the walker themselves. Instead, a third party remotely controlled the walker while they held the back handles and walked alongside it.

Before playing, the game's settings got tailored to each participant, like the IMU's maximum and minimum angles. Figure 60a shows one of these participants doing this by leaning as far as possible. Furthermore, gameovers were disabled, as they would take up more time off the test itself. On that note, sometimes testing was prolonged (by 1 or 2 minutes) so that participants had a chance to experience each event so that they could provide feedback on them.

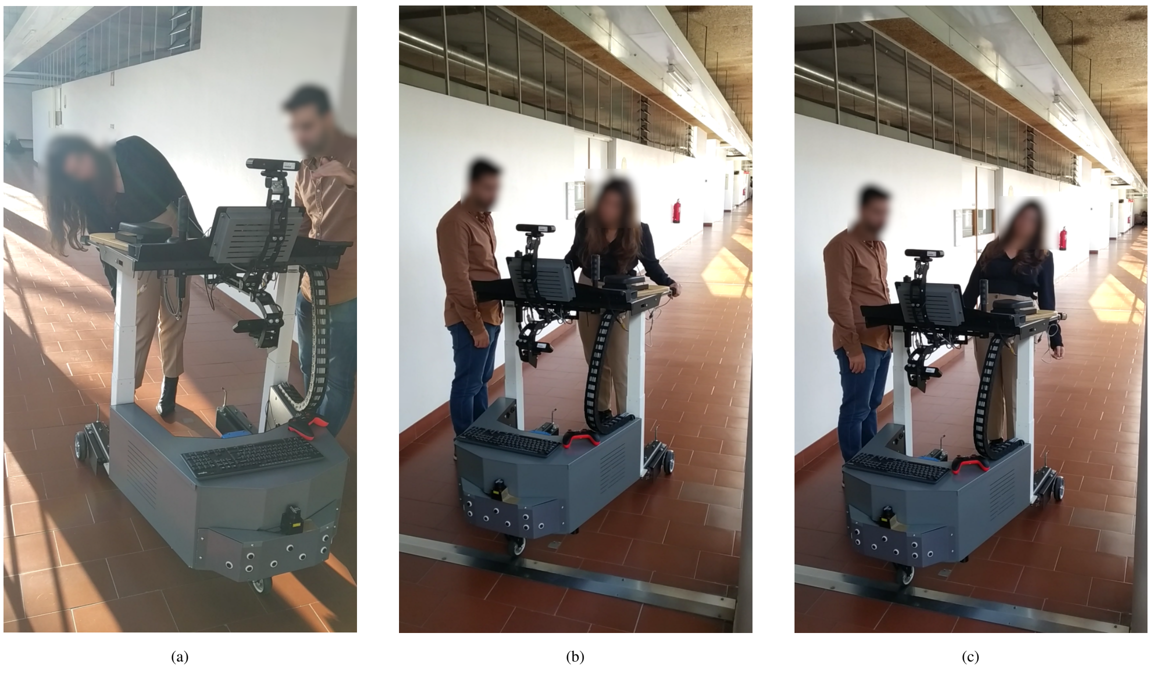


Figure 60: Dynamic Game Test Participant; (a) Configuring IMU, (b) Stopping for Rug Event, (c) Playing Rug Event.

5.5.3 *Validation Metrics*

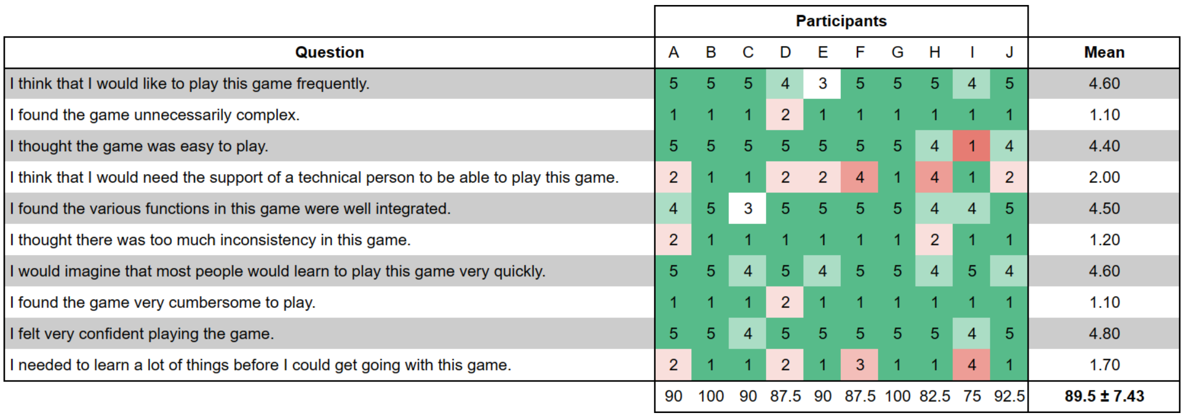
A questionnaire consisting of four parts was given to each participant after completing the 5-minute walking trial. The first three parts of the questionnaire were identical to those found in the previous chapter's section 4.6.3. The first part collected demographic information from the participants, all of whom reported to be healthy. The second part consisted of a Portuguese version of the SUS questionnaire [73], adapted to the context of a game. The third part contained open-ended questions about the experience of playing a game on the walker, e.g., the challenges of walking and playing simultaneously. The fourth and final part of the questionnaire focused on the participants' opinions, specifically regarding the Dynamic Game, including their thoughts on events, character feedback, and expectations for future development.

5.6 RESULTS AND DISCUSSION

This section covers the validation results. In a similar fashion to the Cognitive Game, this section starts by explaining the various questionnaire parts, before presenting an analysis of the results.

5.6.1 *System Usability Score*

A SUS form was completed by the participants, as mentioned in section 5.5.3. The average value of each answer, as well as each individual answer, is displayed in Figure 61.



	Participants										
Question	A	B	C	D	E	F	G	H	I	J	**Mean**
I think that I would like to play this game frequently.	5	5	5	4	3	5	5	5	4	5	4.60
I found the game unnecessarily complex.	1	1	1	2	1	1	1	1	1	1	1.10
I thought the game was easy to play.	5	5	5	5	5	5	5	4	1	4	4.40
I think that I would need the support of a technical person to be able to play this game.	2	1	1	2	2	4	1	4	1	2	2.00
I found the various functions in this game were well integrated.	4	5	3	5	5	5	5	4	4	5	4.50
I thought there was too much inconsistency in this game.	2	1	1	1	1	1	1	2	1	1	1.20
I would imagine that most people would learn to play this game very quickly.	5	5	4	5	4	5	5	4	5	4	4.60
I found the game very cumbersome to play.	1	1	1	2	1	1	1	1	1	1	1.10
I felt very confident playing the game.	5	5	4	5	5	5	5	5	4	5	4.80
I needed to learn a lot of things before I could get going with this game.	2	1	1	2	1	3	1	1	4	1	1.70
	90	100	90	87.5	90	87.5	100	82.5	75	92.5	**89.5 ± 7.43**

Figure 61: SUS scores from the Dynamic Game.

With an average score of 89.5±7.4, the dynamic game ranks slightly lower than the cognitive game, which is not unexpected, as the game is far more complex. However, this score is still quite high, falling comfortably in the "Excellent" category, as shown in Figure 44 from section 4.2, or at worse better than "Good". The lowest individual score attributed was 75, which is still acceptable and within the "Good" adjective.

It is worth noting that the first question, i.e., "I think that I would like to play this game frequently.", saw an increase of 0.6 points from the Cognitive Game, which indicates that the game might be more fun to play. On the other hand, questions like "I thought the game was easy to play." saw a decrease of over 0.4 points, which is indicative of the complexity of this second game.

5.6.2 *General Questions*

This section analyses the results from the third part of the questionnaire, which asked multiple questions about the overall game experience, such as challenges and difficulty in walking and playing. Eleven questions were considered in this questionnaire, as follows: i) "Was the game fun?", ii) "How challenging was the game?", iii) "What could be improved?", iv) "Did walking and playing simultaneously make the game harder?", v) "In your opinion, do you think the game can be useful for rehabilitative purposes?", vi) "Which one (event) was your favourite?", vii) "Did you consider the minigames to be: easy, adequate, or hard?", viii) "Was the scenery too repetitive?", ix) "Was the level too long?", x) "Do you think the walker's movement translates well into the game's?", and xi) "Did it feel natural to control the game with the IMU (back sensor)?".

Q1: Was the game fun?

Starting with the most crucial question, participants unanimously answered yes. The reason for its importance is the same as the previous game, with fun driving motivation.

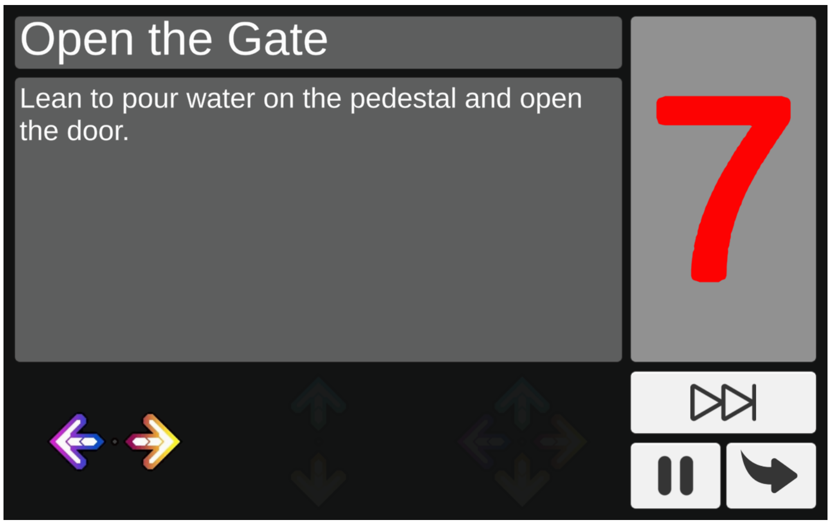
Q2: How challenging was the game?

There were three options: Easy, Adequate, and Hard. Every participant picked Adequate, meaning no one found the game too easy or overly difficult.

Q3: What could be improved?

Two participants suggested the implementation of a countdown at the start of events since they began as soon as the player character touched the anchor. When an event starts, the player has to let go of the handlebars and ready themselves; meanwhile, the timer is already counting. This leads to inflated completion times, which can be frustrating for someone trying to improve their time. The same participants also indicated the need for instructions before starting the events, especially for the first couple of times.

In light of this feedback, both suggestions have made it into the game, as shown in Figure 62. Now, whenever the player touches the event anchor, they first see a screen with a countdown timer and a description of the upcoming event. This screen also introduces additional features like the choice to skip events altogether.


Open the Gate

Lean to pour water on the pedestal and open the door.

Figure 62: Event Info and Timer GUI.

The speed of the Rug Event was found to be too fast and challenging to control by some participants. This was also observed during testing, where players frequently overshot the rings and had to reposition back. In response, the rug's speed was reduced, and the number of rings was increased, which now move towards the player if they are near. Lastly, a participant suggested background music and a friendlier player model. These are planned as future work, but the implementation has been delayed due to time constraints.

Q4: Did walking and playing simultaneously make the game harder?

The responses to the open-ended question were interpreted as yes or no. One participant said the experience added cognitive load, while the rest did not perceive any. Another stated that during the Movement phase, they were able to focus on maintaining correct posture, making the experience less about playing and walking and more about playing through walking.

Likewise, events also avoid dual-tasking, although they do so by requiring the player to stop moving, allowing them to focus solely on performing the event itself.

Q5: In your opinion, do you think the game can be useful for rehabilitative purposes?

Like with the previous game, all participants answered positively. Out of the four that gave further reasoning, three attributed it to the physical quality of performing these exercises, while the other mentioned the motivation from playing a fun game.

Q6: Which one (event) was your favourite?

Out of every question, this is arguably the one with the most disagreement. Two events stand out as the favourites: rug and mole, with the rug event winning by two votes. Their reasons were: the challenge, fun, and dual-axes controls.

Simply put, this event appears to be the most fun to play due to its simple objective of collecting rings and the responsiveness from leaning to either side and having the character move accordingly. However, it is possible that as the sole event controlled on both axes, there is an unfair advantage over the other three.

However, that does not apply to the mole event, which competes directly with the door and fire events. Once again, it appears that the mole event won by virtue of being fun. The two last events had realism as their end goal rather than fun, which might have resulted in bland gameplay.

Additionally, another question asked whether the rug event's esotericism had a negative impact on the participants. Every response stated that it had not, further supporting the hypothesis that fun is more important than realism.

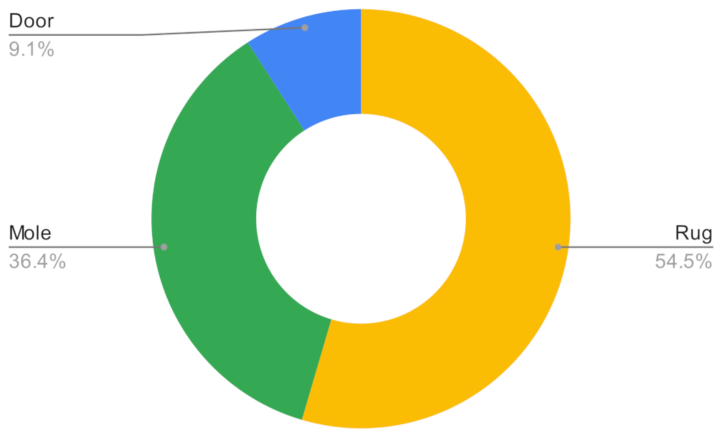


Figure 63: Pie Chart of event popularity distribution.

Q7: Did you consider the minigames to be: easy, adequate, or hard?

The goal of the question was to determine if any events were excessively difficult. Despite the previously reported Rug Event speed issue, no participants found the events hard, with 70% finding them challenging and the remaining 30% stating they were easy.

Q8: Was the scenery too repetitive?

One participant indicated so. That could be addressed by: creating more nodes for each level, creating more shorter levels, or even both.

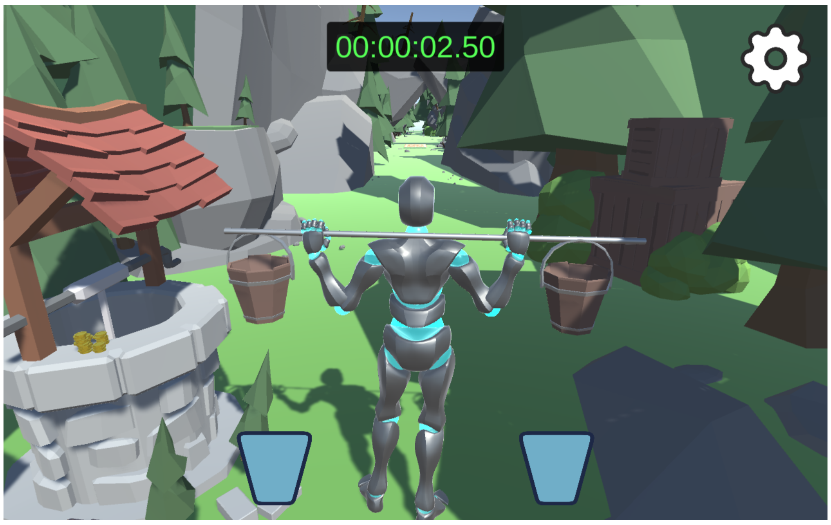


Figure 64: Example of Forest scenery, as seen during testing.

Lastly, there were several yes-no questions, where the response was unanimously in favour of the intended outcome. These are:

Q9: Was the level too long?

It was unanimous that the 5 minutes played during the test was not too long for a single level.

Q10: Do you think the walker's movement translates well into the game's?

All participants agreed that the walker's movement reflected accurately in the on-screen character and, as such, did not require adjustment.

Q11: Did it feel natural to control the game with the IMU (back sensor)?

All participants agreed that the walker's movement felt natural, implying that the sensor calibration was accurate and the configuration values were appropriate.

5.6.3 *Dynamic Game Specific Questions*

Finally, participants went through various features from a list, rating them from 1 to 5 based on how much they would like to see them in a future version of the game. The goal was to gauge player interest and subsequently organise future development of said features.

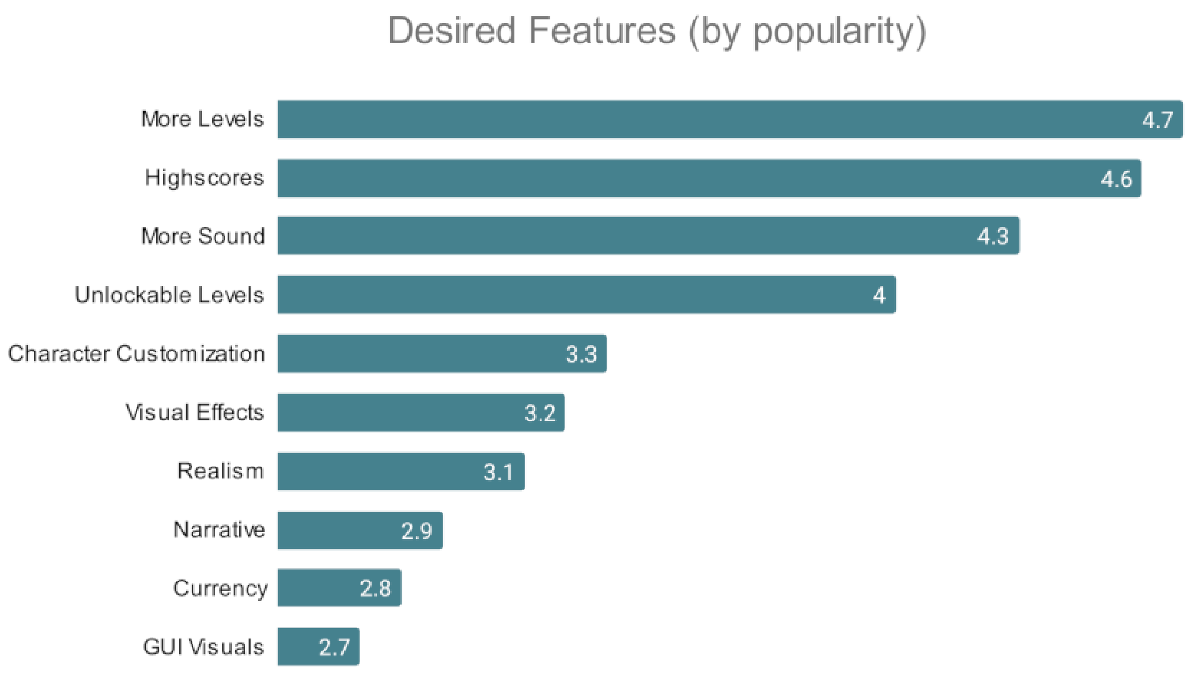


Figure 65: Future Features, ordered by popularity.

Based on this graph, it is clear that the most desired feature, almost unanimously, was the addition of new levels. Fortunately, their creation requires very little technical skill, so with enough time, many can get added to the game.

Next are high scores, just slightly below first place. Scores are clearly great motivators and excellent indicators of improvement. During testing, participants were often delighted when achieving better scores than before. Additionally, they were frequently curious about how their peers performed in these same events.

In third place: is the addition of sound. While the game technically has sound, it is very sparse, almost as a proof of concept; at the moment, there is sound only in the main menu as ambient sound, the desert level's wind, and the event score Sound Effects (SFX). It is noteworthy that, during testing, the speakers were unavailable, so no sound was heard by any participant, which might have influenced their decision.

Future recommendations include: i) background sound, music, or otherwise for each level; ii) footsteps, synced to the player model, that account for the terrain type; and iii) sound effects, e.g., water sloshing, fire crackling, game overs, among others.

Fourth, unlockable levels. The idea behind these is that only some paths will be available when playing for the first time. As the player progresses, more options become available in the following sessions, incentivizing the patient to return.

The remaining suggestions appear to be far less desirable; they primarily focus on the visual aspects of the game, contrary to the gameplay and QoL offered by the above four.

5.7 GENERAL CONCLUSIONS

This chapter presented the design process, development, and validation of the second game of this dissertation, the dynamic game, whose main goal was to control an avatar considering the user's weight distribution if using the load cells, or the user's torso orientation if using the inertial sensor. The player character holds two buckets full of water that, according to the user's input, can stay full or start to empty. This game was designed specifically for balance rehabilitation while walking with WALKit Smart Walker, with specific events that exercise the user's balance control. Furthermore, if using the load cells configuration, this game may also give biofeedback to users regarding their posture in a fun and intuitive way.

The results from the validation were very insightful, although some questions could have been worded differently to get more specific answers. Overall the game seems to be satisfactory, with no explicitly negative opinions towards it. Still, this testing phase revealed many of the game's areas for improvement. Some got addressed as of writing this dissertation, while others got left for future development. These include the previously mentioned event countdown and instructions.

The questionnaire also gathered valuable information on which features should be implemented in the future or at least the order in which they should get implemented. Still, these conclusions can change once proper clinical testing is conducted, which was unfortunately not possible in the span of this dissertation.

In conclusion, according to the literature, the Dynamic Game currently fits the ideal characteristics of a Serious Game: it has a range of features, and enjoyable gameplay, while also being customizable and accommodating to the patients' physical capabilities. However, there is room for improvement with the addition of new levels, events, and features, which can easily be implemented by future developers focusing on specific aspects of the game.

6

CONCLUSION

In this dissertation, two serious games were developed to explore the use of serious video games in a robotic walker to improve ataxic gait. The work started with the first objective, which was to research the state-of-the-art of not only ataxia-focused serious games but most balance-focused serious games. This research was the key for identifying the most important aspects of serious games, as outlined in objective two and three of this dissertation, and drafting several ideas for the games.

Before development could start, it was conducted a research into all the development variables, including QML, Game Engines, how to deploy the games from the walker's application and possible alternatives if that proved impossible. Once these frameworks were in place, the work began on the WALKit's application algorithm of the cognitive game, which allows the user to select levels and launch the game. Afterwards, the game itself was developed, taking approximately two months, and fulfilling part of objective four. Afterwards was the dynamic game's algorithm, which was considerably faster to develop, given that it was based on the first one. This algorithm allows the user to configure some settings before launching the game. The game itself was made from scratch, and was the longest part of the dissertation, taking an estimated 5 to 6 months, and finalising the fourth objective, with work still in progress.

Once both games were in a suitable state, validation began, which was the fifth and final objective of the dissertation. This involved testing the games with several participants and having them answer a questionnaire afterwards. The feedback was overall positive, with the cognitive game attaining a SUS score of 92 ±6, or "Excelent", while the dynamic game received a slightly lower score of 89.5 ±7.4, or between "Excelent" and "Good". Unfortunately, this validation was not accomplished with target end-users, i.e., persons with ataxia, due to time constraints and the lack of patients available to participate in the clinical study. Still, the conclusion of the validation phase also meant that the final objective, Objective 5, was accomplished, although with healthy volunteers. With this, the three research questions posed in Chapter 1 could be answered.

RQ1: "How can serious games be helpful in motor rehabilitation?"

Chapter 2 answers this by examining the current state-of-the-art regarding serious games in not just the field of ataxia therapy but balance therapy as a whole. It was concluded that serious games could be categorised as either custom or commercial. It further emphasised some aspects of serious games: customisation, replay value, and aesthetics. According to the research gathered, the ideal game would be a custom game, with some of these better aspects from commercial games mixed in. This study further concluded that serious games are beneficial for rehabilitation as the main drivers or complements of conventional therapy. This derives from the fact that most studies reported increased test results when introducing serious games.

RQ2: "How can serious games be included in WALKit?"

The third chapter answers this by analysing all the available walker components, including hardware controllers, such as the load cells and IMUs. It also reviewed the software, such as the Linux OS, and the QML and C++ languages used to write the UI application. Lastly, this chapter explored various game development frameworks, from low-level frameworks to fully fleshed game engines, ultimately deciding on the Unity game engine. This research facilitated the subsequent planning phase of both games.

RQ3: "How could the addition of SGs to WALKit improve the rehabilitation of ataxic patients?"

Chapters 4 and 5 tackle the same research question, although from different angles. They share a similar structure, but the difference lies in the type and scope of the games discussed. Chapter 4 focuses on the cognitive game, the first game developed, while Chapter 5 focuses into the dynamic game.

Both games were tested with healthy participants and the results indicated that they were enjoyable to play and, according to the participants, could aid in the rehabilitation process for ataxic patients.

6.1 FUTURE WORK

This dissertation aimed to develop two fully functional serious games, the cognitive game and the dynamic game. While both games are fully functional, they have ample room for further improvement and expansion. Due to the time-consuming nature of game development and the limited duration of the dissertation, the focus was on critical game aspects, resulting in many desired features being left unimplemented.

These features fall into two categories: concrete and abstract. Concrete features include core game systems, such as character customisation in the dynamic game or the inclusion of high scores in both games. The abstract features, on the other hand, include the creation of new levels in both games, which could be endless, and therefore work could never truly be completed.

To guide future development, the dynamic game test questionnaire was conducted, asking users to indicate their preference for several features. The results, shown in Figure 65, provide a clear roadmap for future development. It would be advisable to prioritise the development of new levels, followed by the features with the highest demand according to the survey results.

Despite having validated both games, this was not a true clinical validation which was a significant focus in most of the literature reviewed for this dissertation. Therefore, a complete validation of both games with patients should be conducted since, as the project stands, the games are only hypothetically beneficial for rehabilitation.

The creation of the games has resulted in the development of a number of tools, such as Unity objects that are able to read sensor values, which automate communication between the walker and the game, which can be used in any Unity project that interacts with WALKit, not just games. Another byproduct of these games was a template for future QML game algorithms. These tools not only have the potential to support the development of future serious games but also to be further improved, as they greatly accelerate the development process.

Lastly, it would also be possible to adapt these games to other devices and create additional controllers if necessary, making the games accessible to a wider range of users.

BIBLIOGRAPHY

[1] Jon Marsden and Chris Harris. Cerebellar ataxia: pathophysiology and rehabilitation. *Clinical Rehabilitation*, 25(3):195–216, February 2011.

[2] Winfried Ilg and Dagmar Timmann. Gait ataxia-specific cerebellar influences and their rehabilitation. *Movement Disorders*, 28(11):1566–1575, September 2013.

[3] Tadeusz Mikolajczyk, Ileana Ciobanu, Doina Ioana Badea, Alina Iliescu, Sara Pizzamiglio, Thomas Schauer, Thomas Seel, Petre Lucian Seiciu, Duncan L Turner, and Mihai Berteanu. Advanced technology for gait rehabilitation: An overview. *Advances in Mechanical Engineering*, 10(7):168781401878362, July 2018.

[4] Geneieve Tai, Louise A Corben, Eppie M Yiu, Sarah C Milne, and Martin B Delatycki. Progress in the treatment of friedreich ataxia. *Neurologia i neurochirurgia polska*, 52(2):129–139, 2018.

[5] Bruce H Dobkin and Andrew Dorsch. New evidence for therapies in stroke rehabilitation. *Current atherosclerosis reports*, 15(6):1–9, 2013.

[6] CL Martin, D. Tan, P. Bragge, and A. Bialocerkowski. Effectiveness of physiotherapy for adults with cerebellar dysfunction: a systematic review. *Clinical Rehabilitation*, 23(1):15–26, January 2009.

[7] K M Gill-Body, R A Popat, S W Parker, and D E Krebs. Rehabilitation of balance in two patients with cerebellar dysfunction. *Phys. Ther.*, 77(5):534–552, May 1997.

[8] Niall Maclean and Pandora Pound. Discussion. *Social Science & Medicine*, 50(4):495–506, February 2000.

[9] Gianfranco Lamberti, Gianluca Sesenna, Martina Marina, Emanuela Ricci, and Gianluca Ciardi. Robot assisted gait training in a patient with ataxia. *Neurology International*, 14(3):561–573, 2022.

[10] World Health Organization. *World Report on Disability 2011*, pages 93–133. World Health Organization, Genève, Switzerland, May 2011.

[11] Maria M. Martins, Cristina P. Santos, Anselmo Frizera-Neto, and Ramón Ceres. Assistive mobility devices focusing on smart walkers: Classification and review. *Robotics and Autonomous Systems*, 60(4):548–562, April 2012.

[12] Maria Martins, Cristina P Santos, Anselmo Frizera, Ana Matias, Tânia Pereira, Maria Cotter, and Fátima Pereira. Smart walker use for ataxia's rehabilitation: Case study. In *2015 IEEE International Conference on Rehabilitation Robotics (ICORR)*, pages 852–857, 2015.

[13] Serhii Shapoval, Begoña García Zapirain, Amaia Mendez Zorrilla, and Iranzu Mugueta-Aguinaga. Biofeedback applied to interactive serious games to monitor frailty in an elderly population. *Applied Sciences*, 11(8):3502, April 2021.

[14] Rizzo Albert A. and Buckwalter J. Galen. Virtual reality and cognitive assessment and rehabilitation: The state of the art. *Studies in Health Technology and Informatics*, 44(Virtual Reality in Neuro-Psycho-Physiology):123–145, 1997.

[15] Maria Martins, Cristina P Santos, Anselmo Frizera, Ana Matias, Tania Pereira, Maria Cotter, and Fatima Pereira. Smart walker use for ataxia's rehabilitation: Case study. In *2015 IEEE International Conference on Rehabilitation Robotics (ICORR)*. IEEE, August 2015.

[16] D. Jack, R. Boian, A.S. Merians, M. Tremaine, G.C. Burdea, S.V. Adamovich, M. Recce, and H. Poizner. Virtual reality-enhanced stroke rehabilitation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 9(3):308–318, September 2001.

[17] Biel Moyà Alcover, Antoni Jaume i Capó, Javier Varona, Pau Martinez-Bueso, and Alejandro Mesejo Chiong. Use of serious games for motivational balance rehabilitation of cerebral palsy patients. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*. ACM, October 2011.

[18] Sílvia Lopes, Paula Magalhães, Armanda Pereira, Juliana Martins, Carla Magalhães, Elisa Chaleta, and Pedro Rosário. Games used with serious purposes: A systematic review of interventions in patients with cerebral palsy. *Frontiers in Psychology*, 9, September 2018.

[19] Sruti Subramanian, Yngve Dahl, Nina Skjæret Maroni, Beatrix Vereijken, and Dag Svanæs. Assessing motivational differences between young and older adults when playing an exergame. *Games for Health Journal*, 9(1):24–30, February 2020.

[20] Gabriel Moyà-Alcover, Ines Ayed, Javier Varona, and Antoni Jaume i Capó. Rgb-d interactive systems on serious games for motor rehabilitation therapy and therapeutic measurements. *Advances in Computer Vision and Pattern Recognition*, pages 335–353, 2019.

[21] Jen-Wen Hung, Chiung-Xia Chou, Hsueh-Feng Chang, Wen-Chi Wu, Yen-Wei Hsieh, Po-Chih Chen, Min-Yuan Yu, Chiung-Chih Chang, and Jr-Rung Lin. Cognitive effects of weight-shifting controlled exergames in patients with chronic stroke: a pilot randomized comparison trial. *European Journal of Physical and Rehabilitation Medicine*, 53(5), October 2017.

[22] Emmanuel Bonney, Lemke Dorothee Jelsma, Gillian D. Ferguson, and Bouwien C.M. Smits-Engelsman. Learning better by repetition or variation? is transfer at odds with task specific training? *PLoS ONE*, 12, 3 2017.

[23] Oleh Kachmar, Anna Kushnir, Bohdana Fedchyshyn, Julián Cristiano, John O'Flaherty, Kjetil Helland, Gordon Johnson, and Domenec Puig. Personalized balance games for children with cerebral palsy: A pilot study. *Journal of Pediatric Rehabilitation Medicine*, 14(2):237–245, June 2021.

[24] Kazuo Hemmi, Yuki Kondo, Takuro Tobina, and Takeshi Nishimura. Floor projection type serious game system for lower limb rehabilitation using image processing. In *Recent Advances in Information and Communication Technology 2019*, pages 119–128. Springer International Publishing, May 2019.

[25] Judith E. Deutsch, Brittany Hoehlein, Marisa Priolo, Joshua Pacifico, Harish Damodaran, and Urska Puh. Custom game paced video games played by persons post-stroke have comparable exercise intensity but higher accuracy, greater enjoyment and less effort than off-the-shelf game. In *2019 International Conference on Virtual Rehabilitation (ICVR)*. IEEE, July 2019.

[26] Thomas Matheve, Guido Claes, Enzo Olivieri, and Annick Timmermans. Serious gaming to support exercise therapy for patients with chronic nonspecific low back pain: A feasibility study. *Games for Health Journal*, 7(4):262–270, August 2018.

[27] Zahra Amiri, Yoones A. Sekhavat, and Sakineh Goljaryan. A framework for rehabilitation games to improve balance in people with multiple sclerosis (MS). In *2018 2nd National and 1st International Digital Games Research Conference: Trends, Technologies, and Applications (DGRC)*. IEEE, November 2018.

[28] Bruno Bonnechère, Lubos Omelina, Bart Jansen, and Serge Van Sint Jan. Balance improvement after physical therapy training using specially developed serious games for cerebral palsy children: preliminary results. *Disability and Rehabilitation*, 39:403–406, 2 2017.

[29] A V Soares, N G Borges Júnior, M S Hounsell, E Marcelino, G M Rossito, and Y Sagawa Júnior. A serious game developed for physical rehabilitation of frail elderly. *Eur. res. telemed.*, 5(2):45–53, June 2016.

[30] Terigi Augusto Scardovelli and Annie France Frère. The design and evaluation of a peripheral device for use with a computer game intended for children with motor disabilities. *Computer Methods and Programs in Biomedicine*, 118(1):44–58, January 2015.

[31] Jose-Antonio Lozano-Quilis, Hermenegildo Gil-Gómez, Jose-Antonio Gil-Gómez, Sergio Albiol-Pérez, Guillermo Palacios-Navarro, Habib M Fardoun, and Abdulfattah S Mashat. Virtual rehabilitation for multiple sclerosis using a kinect-based system: Randomized controlled trial. *JMIR Serious Games*, 2(2):e12, November 2014.

[32] Yi-Wen Wang, Chien-Hsu Chen, and Yang-Cheng Lin. Balance rehabilitation system for parkinson's disease patients based on augmented reality. In *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*. IEEE, October 2020.

[33] Philippe Charbonneau, Mikael Dallaire-Cote, Sara Saint-Pierre Cote, David R. Labbe, Neila Mezghani, Sharif Shahnewaz, Imtiaz Arafat, Tanvir Irfan, Gayani Samaraweera, and John Quarles. Gaitzilla: Exploring the effect of embodying a giant monster on lower limb kinematics and time perception. In *2017 International Conference on Virtual Rehabilitation (ICVR)*. IEEE, June 2017.

[34] Shih-Ching Chen, Chueh-Ho Lin, Sheng-Wen Su, Yu-Tai Chang, and Chien-Hung Lai. Feasibility and effect of interactive telerehabilitation on balance in individuals with chronic stroke: a pilot study. *Journal of NeuroEngineering and Rehabilitation*, 18(1), April 2021.

[35] Asiye Tuba Ozdogar, Ozge Ertekin, Turhan Kahraman, Pinar Yigit, and Serkan Ozakbas. Effect of video-based exergaming on arm and cognitive function in persons with multiple sclerosis: A randomized controlled trial. *Multiple Sclerosis and Related Disorders*, 40:101966, May 2020.

[36] F. Noveletto, A. V. Soares, B. A. Mello, C. N. Sevegnani, F. L.F. Eichinger, M. Da S. Hounsell, and P. Bertemes-Filho. Biomedical serious game system for balance rehabilitation of hemiparetic stroke patients. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 26:2179–2188, 11 2018.

[37] The art of game polish: Developers speak, 2021. Accessed on: 01/22/2023, URL: https://www.gamedeveloper.com/design/the-art-of-game-polish-developers-speak.

[38] Asiye Tuba Ozdogar, Ozge Ertekin, Turhan Kahraman, Pinar Yigit, and Serkan Ozakbas. Effect of video-based exergaming on arm and cognitive function in persons with multiple sclerosis: A randomized controlled trial. *Mult. Scler. Relat. Disord.*, 40(101966):101966, May 2020.

[39] Keyte Guedes Silva, Tatiana Beline De Freitas, Flávia Doná, Fernando Freitas Ganança, Henrique Ballalai Ferraz, Camila Torriani-Pasin, and José Eduardo Pompeu. Effects of virtual rehabilitation versus conventional physical therapy on postural control, gait, and cognition of patients with parkinson's disease: study protocol for a randomized controlled feasibility trial. *Pilot Feasibility Stud.*, 3(1), December 2017.

[40] Valentin Bégel, Ines Di Loreto, Antoine Seilles, and Simone Dalla Bella. Music games: Potential application and considerations for rhythmic training. *Front. Hum. Neurosci.*, 11:273, May 2017.

[41] María José Cano-Mañas, Susana Collado-Vázquez, Javier Rodríguez Hernández, Antonio Jesús Muñoz Villena, and Roberto Cano-De-La-Cuerda. Effects of video-game based therapy on balance, postural control, functionality, and quality of life of patients with subacute stroke: A randomized controlled trial. *Journal of Healthcare Engineering*, 2020, 2020.

[42] Ki Yeun Nam, Hyun Jung Kim, Bum Sun Kwon, Jin-Woo Park, Ho Jun Lee, and Aeri Yoo. Robot-assisted gait training (lokomat) improves walking function and activity in people with spinal cord injury: a systematic review. *J. Neuroeng. Rehabil.*, 14(1), December 2017.

[43] Stefanie T L Pöhlmann, Elaine F Harkness, Christopher J Taylor, and Susan M Astley. Evaluation of kinect 3D sensor for healthcare imaging. *J. Med. Biol. Eng.*, 36(6):857–870, December 2016.

[44] H. Tannous, D. Istrate, M. C. Ho Ba Tho, and T. T. Dao. Étude de faisabilité d'un système de jeux sérieux basé sur le système kinect pour la rééducation fonctionnelle des membres inférieurs. *European Research in Telemedicine*, 5:97–104, 9 2016.

[45] A.V. Soares, N.G. Borges Júnior, M.S. Hounsell, E. Marcelino, G.M. Rossito, and Y. Sagawa Júnior. Un jeu sérieux développé pour la réhabilitation physique des personnes âgées atteintes de syndrome de fragilité. *European Research in Telemedicine*, 5:45–53, 6 2016.

[46] Prashant K Jamwal, Shahid Hussain, and Mergen H Ghayesh. Robotic orthoses for gait rehabilitation: An overview of mechanical design and control strategies. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 234(5):444–457, January 2020.

[47] Nathalie Swinnen, Mathieu Vandenbulcke, Eling D. de Bruin, Riekje Akkerman, Brendon Stubbs, and Davy Vancampfort. Exergaming for people with major neurocognitive disorder: a qualitative study. *Disability and Rehabilitation*, pages 1–9, September 2020.

[48] Jane Case-Smith and Jane Clifford O'Brien. *Occupational therapy for children*. Case Review. Mosby, St. Louis, MO, 6 edition, December 2009.

[49] Iuliana Chiuchisan, Oana Geman, and Octavian Postolache. Future trends in exergaming using MS kinect for medical rehabilitation. In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*. IEEE, October 2018.

[50] Peter Kokol, Helena Blažun Vošner, Jernej Završnik, Joeri Vermeulen, Samaa Shohieb, and Frank Peinemann. Serious game-based intervention for children with developmental disabilities. *Current Pediatric Reviews*, 16(1):26–32, April 2020.

[51] B Bonnechère, B Jansen, L Omelina, M Degelaen, V Wermenbol, M Rooze, and S Van Sint Jan. Can serious games be incorporated with conventional treatment of children with cerebral palsy? a review. *Res. Dev. Disabil.*, 35(8):1899–1913, August 2014.

[52] Terigi Augusto Scardovelli and Annie France Frère. The design and evaluation of a peripheral device for use with a computer game intended for children with motor disabilities. *Comput. Methods Programs Biomed.*, 118(1):44–58, January 2015.

[53] Re-Logic. Terraria on steam. Accessed: 2022-10-02, Available at: `https://store.steampowered.com/app/105600/Terraria/`.

[54] Supergiant Games. Hades presskit. Accessed: 2022-10-02, Available at: `https://www.igdb.com/games/hades--1/presskit`.

[55] id Software. Doom eternal presskit. Accessed: 2022-10-02, Available at: `https://www.igdb.com/games/doom-eternal/presskit`.

[56] Namco Bandai. Katamari damacy mini. Accessed: 2022-10-02, Available at: `https://phaser.io/news/2021/04/katamari-damacy-mini`.

[57] VoxelSrv. Voxelsrv. Accessed: 2022-10-02, Available at: `https://github.com/VoxelSrv/voxelsrv`.

[58] Plasma Starfish. Spellcraft. Accessed: 2022-10-02, Available at: https://plasmastarfish.itch.io/spellcraft.

[59] AnythingTechPro. A3p. Accessed: 2022-10-02, Available at: https://anythingtechpro.itch.io/a3p.

[60] Unity Technologies. Unity, Oct 2022. Available at: https://polycount.com/discussion/209380/unity-editor-theme-survey-second-iteration.

[61] Epic Games. Unreal engine, Oct 2022. Available at: https://ultragames786.blogspot.com/2019/04/unreal-engine-4190-free-download.html.

[62] Playtonic Games. Yooka-laylee, Apr 2017. Available at: https://ultragames786.blogspot.com/2019/04/unreal-engine-4190-free-download.html.

[63] Blizzard Entertainment. Hearthstone, Mar 2014. Available at: https://ultragames786.blogspot.com/2019/04/unreal-engine-4190-free-download.html.

[64] Moon Studios. Ori and the will of the wisps, Mar 2020. Available at: https://polycount.com/discussion/209380/unity-editor-theme-survey-second-iteration.

[65] Gearbox Software. Borderlands: The pre-sequel, Oct 2014. Available at: https://borderlands.com/en-US/picture.

[66] Tarsier Studios. Little nightmares ii, Feb 2021. Available at: https://www.igdb.com/games/little-nightmares-ii/presskit.

[67] Jon Favreau. The mandalorian, Nov 2019. Available at: https://www.kotaku.com.au/2020/02/the-mandalorian-unreal-engine-4-ilm/.

[68] Jared Couto. Primal light, Apr 2021. Available at: https://godotengine.org/showcase/primal-light.

[69] Andres Ortiz. Until then, Feb 2021. Available at: https://godotengine.org/showcase/until-then.

[70] Oli Latham. Hive time, Feb 2021. Available at: https://godotengine.org/showcase/hive-time.

[71] Unity Technologies. Scripting - unity manual, 2023. Available at: https://docs.unity3d.com/Manual/ScriptingImportantClasses.html.

[72] dhbaird. easywsclient, n.d. https://github.com/dhbaird/easywsclient (accessed Jan 28, 2023).

[73] Candida S. Punla, https://orcid.org/ 0000-0002-1094-0018, cspunla@bpsu.edu.ph, Rosemarie C. Farro, https://orcid.org/0000-0002-3571-2716, rcfarro@bpsu.edu.ph, and Bataan Peninsula State University Dinalupihan, Bataan, Philippines. Are we there yet?: An analysis of the competencies of BEED graduates of BPSU-DC. *International Multidisciplinary Research Journal*, 4(3):50–59, September 2022.

[74] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Studies*, 4(3):114–123, May 2009.

[75] Diganta Saha. Kalman filter lite, 2021. https://github.com/diganta162004/KalmanFilterLite.