Haptic Collaborative Games

COMP2002 / G52GRP: Final Report

Project Information:

Project title: UoN-HapticCollaborativeGames
Project sponsor: Ayse Kucukyilmaz
Academic supervisor: Ayse Kucukyilmaz

Team information:

Team number: 21

Salaar, Mir, 20275881, psysm13
Daniel, Lambert, 14328665, ppydl3
Tianfeng, Chen, 20215546, scytc1
Jiahao, Wang, 20217166, scyjw5,
Ibrahim, Atomanson, 20265875, psyia3
Lok Him, Lam, 20219626, efyll1

Documentation (links):

Trello board: https://trello.com/b/ZC3MWaD7/haptic-collaborative-games

Code repository: https://projects.cs.nott.ac.uk/comp2002/2021-2022/team21_project

Document repository: https://projects.cs.nott.ac.uk/comp2002/2021-2022/team21_project/-/tree/main/Documentation

Contents

1 Interim Report Changelog **3**

2 Introduction **3**
2.1 Background . . . 3
2.1.1 Haptic Technology . . . 3
2.1.2 Collaboration Based Research . . . 4
2.1.3 The CHAI3D Framework . . . 5
2.2 Main Objectives . . . 5

3 Problem Analysis **5**
3.1 Analysing the Brief . . . 5
3.1.1 Requirement Analysis . . . 6

4 Implementation **7**
4.1 AI Design . . . 7
4.2 Potential Field . . . 8
4.3 Mouse Controls . . . 9
4.4 Haptic controls . . . 10
4.5 Activity Diagram . . . 11
4.5.1 Explaining the Activity Diagram . . . 11
4.6 Testing . . . 12
4.6.1 Unit Testing . . . 12
4.6.2 Integration Testing . . . 12
4.6.3 System Testing . . . 12

5 Management **12**
5.1 Our Initial Agile-Based Approach . . . 12
5.2 Time Management . . . 13
5.3 A Change in Methodology . . . 13
5.4 Our Meeting Format . . . 13
5.5 Git usage . . . 14

6 Successes and Achievements **14**

7 Summary **15**

8 Future Roadmap **15**

1 Interim Report Changelog

Modifications to specific sections of report from Interim:

1 Introduction:	Heavily modified — Combined with background
2 Background Information and Research:	Removed
3 Requirements Analysis:	Heavily modified
4 Management:	Heavily modified
5 UML Diagrams:	Integrated with implementation
6 Prototype:	Removed
7 Reflections:	All subsections combined into their relevant sections
8 Roadmap:	Heavily modified
9 Summary:	Heavily modified

Table 1: Changelog of the report

2 Introduction

2.1 Background

2.1.1 Haptic Technology

Haptic technology applies a combination of forces, vibrations and motions to a user in order to create an experience of touch. Devices employing haptic technologies boast a wide variety of applications. Some of the most common and simple haptic technologies come in the form of tiny motors that users would find in any smartphone that spins and creates vibrations when, for example, the user receives a notification. Motors are also found in game controllers that might vibrate when something such as an explosion occurs in a game, with the objective of increasing immersion. Taking a step up, some more sophisticated haptic devices like racing game steering wheels use force feedback to manipulate the movement of the wheel held by a user. Racing wheels specifically attempt to simulate real-life racing conditions by applying forces to turn the wheel when driving over rough terrain or turning a corner.

The Haption Virtuose 6D [1] is one of the most sophisticated haptic devices to date, combining high force feedback in six degrees of freedom (see Figure 1) with a large workspace. Force feedback on this sort of free movement device unlocks many possibilities. Contact between a haptic device cursor with a static object in virtual space causes forces to be applied to the arm in such a way that it creates the illusion that users are feeling the shape or textures of the object in real space.

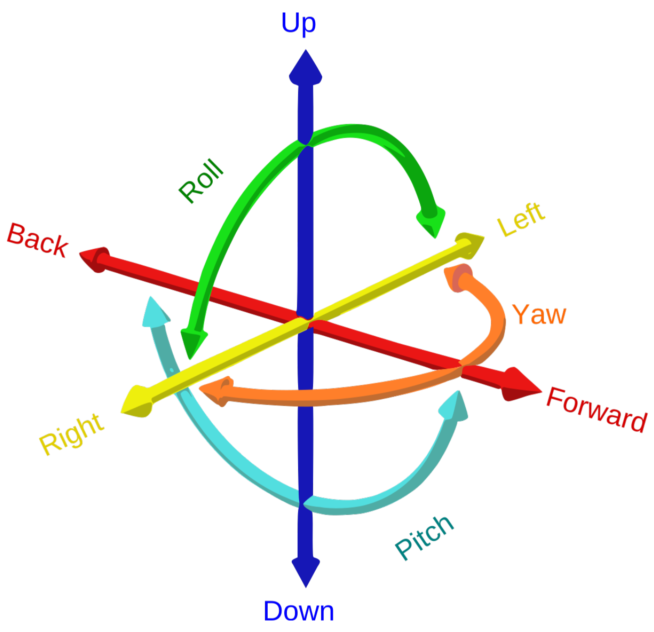


Figure 1: A diagram showing the six degrees of freedom [2]

2.1.2 Collaboration Based Research

A key consideration of our project is how to implement variable collaboration in a game with inputs from two agents. The following paper discusses this topic and forms the basis of our project.

Intention Recognition for Dynamic Role Exchange in Haptic Collaboration [3]
This paper proposes that a computer's ability to improve efficiency and effectiveness when aiding humans in dynamic tasks is limited because they lack a human's adaptability, versatility and awareness. If there exists a computer that can infer its human equivalent's intentions and adjust their respective control levels accordingly, it might facilitate a more intuitive interaction setup.

To investigate this, the paper proposes a "dynamic role exchange mechanism" where two partners negotiate their control over a task through a haptic channel that may increase their joint efficiency. A haptic negotiation model is used where force negotiation is achieved by setting two stiffness constants, $K_{p,H,N}$, $K_{p,C,N}$, one for each agent. These constants set the control levels for each input. The computer infers a user's intentions by his/her movements: a human is assumed to take control of movements requiring a large amount of force, such as large open areas, but when the user slows down, the computer infers that control is being surrendered so that it can provide more precise movements. The board game built by the authors provides large spaces for greater force to be applied and also narrows channels intended to be more efficient for computer dominated movement.

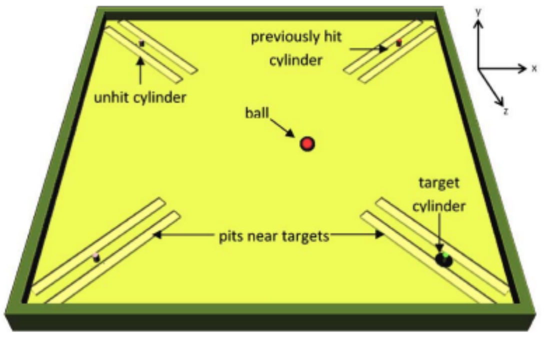


Figure 2: The paper's proposed haptic board game

The authors construct the board game (see Figure 2) which contains four narrow channels for the ball to move towards several goals, with pits on either side of each where the player has to start the entire task over if they fall in. The player has to hit all four goals before the game ends. This design is chosen to create an environment where the AI and the player are more effective than each other at different points during the task. The authors show that this role exchange mechanism improves task performance and the joint efficiency of the partners.

2.1.3 The CHAI3D Framework

Our work is built on top of the CHAI3D[4] (Computer Haptics and Active Interface) framework - an open-source set of C++ libraries for computer haptics, visualisation and interactive real-time simulation. CHAI3D was developed to provide a simple and compact framework for introducing experimental haptics to the research community. Written entirely in C++, CHAI3D was designed to make it easier and more intuitive for developers to create applications that combine 3D modelling with force-feedback rendering capabilities.

The framework has multiple advantages specific to our requirements:

- An abundance of force rendering algorithms can be utilised to compute the interaction forces among objects within an environment.
- Full integration with a lightweight OpenGL-based graphics engine which provides the foundations to easily render Mesh Objects, Line Segments and Volume Objects with surface materials and texture properties.
- Full support for integration with a range of haptic devices with six degrees of freedom and high-level force feedback.
- Full integration with third-party dynamics engines like Bullet and ODE, making it possible to simulate rigid and deformable bodies in real-time to help imitate a physical task.

2.2 Main Objectives

In this project, we aimed to create a logical software architecture for implementing 3D haptic games for use in research. These games could be controlled by both normal input devices, and a more sophisticated haptic force feedback device. With the help of this device, users would be able to feel forces due to game dynamics and the input from the AI trying to collaborate.

Therefore, our most important objectives for the project were as follows:

- Create an easy-to-understand API that simplifies the creation of haptic games.
  This meant our API had to have a modular structure and strong documentation so that whoever might use our framework to create a game to their own specifications can do so with ease.
- Build convincing motion planning for an AI that allows collaboration with a human user.
  This meant we had to carefully consider how to handle two agents providing inputs to an object and decide on the right algorithm to balance processing time with the precision of the optimal path.
- Build the project using stellar software engineering principles to facilitate productivity.
  To accomplish this we needed from the beginning to keep our directory structure clean and readable, make use of existing tools like Trello to keep track of all tasks, have consistent meetings after development including with our supervisor, and reflect on our successes and shortcomings during such meetings.

3 Problem Analysis

3.1 Analysing the Brief

We got a list of requirements from the brief and chose suitable tools to achieve them. For the non-functional requirements, several aspects needed to be valued. For proper functionality, this software must run on Linux and screens would be adapted to the screen size. For performance, the software will be able to give a quick response (no more than one second for AI to compute the optimal path). When users interact with our software, the force feedback that the user feels cannot exceed the maximum force feedback that the haptic device can provide. For maintainability, modular programming has been adopted to facilitate maintenance, and the naming of source files and variables is clear and understandable.

3.1.1 Requirement Analysis

For functional requirements, several parts are analyzed in detail.

Firstly, we are supposed to complete at least one game with game graphics, physics simulation and haptic feedback. This was a very basic requirement, and we implemented a 3D haptic game architecture to create a basic example we called "The Maze Game". We decided on a maze-based board game architecture because it could be based on prior research. CHAI3D could easily create exactly the objects we required in a game world. The Bullet engine could also be applied to them to realize realistic physics simulations such as collision detection and applied force vectors. The user can feel haptic force feedback when the ball moves on the maze map. In addition, a review of literature inspired the path finding algorithm, we designed a similar mechanism.

Secondly, we were supposed to have three interaction settings, including Human vs. Human, Human vs. AI, AI vs. AI, but considering that there is only one haptic device, the requirement of making the game multiplayer is unachievable and would have added too much scope. Instead, we decided that having the user collaborate with an AI would be achievable. Therefore, we decided not to make the game competitive but instead teamed up the user and AI, where the AI uses artificial guidance to keep the user on the right track, a change in requirements which was agreed upon with our supervisor. However, if more than one device becomes available later, users can do simple modifications to allow the game to be compatible with two players.

Thirdly, it is required to be possible to run the game without connecting a haptic joystick (i.e. play should be possible using a game-pad without haptic rendering capabilities). Hence, we planned to create two versions, one is controlled by a haptic device and the other one is controlled by a mouse because users have mouse equipment available at any time.

Fourthly, we were supposed to construct the modular software structure to separate game graphics and logic. At first, we used a singleton type design pattern to accomplish the logical software architecture, whilst when getting familiar with the purpose of each function and having a better understanding of the whole structure, we were then able to implement a more procedural programming approach, creating header and source files for different elements of the game separately.

Fifthly, we are supposed to allow easy modification of the system in terms of changing the game logic and AI behaviour. We understood that as long as the previous modular software structure has been made accurately with separating game graphics (each element of the game is in different files) and logic and clear extension instructions are provided, this requirement could be met easily.

A comprehensive list of detailed requirements and specifications that we collated when we analysed the brief can be found in Table 2.

Requirements	Specifications
Users should be able to play the game with a mouse and keyboard, as well as a haptic joystick	Give the user the choice of input device they wish to use.
Users should be assisted by an AI when required, which can provide more precise movements.	The AI must consist of simple motion planning algorithms that move the agent in the correct direction towards the goal.
Users should be able to play the game with the Haption Virtuose 6D Device.	The API needs to be intertwined with the CHAI3D library that the device can use to operate as well as the Bullet physics engine to allow for collision detection and applied forces.
Users should have a game to play that is easily modifiable into other games that still implements the Haption Virtuose 6D Device.	These games should use the same basic architecture to allow for easy modifiability.
Users should be able to browse through different interfaces of the game to allow for better interactivity.	The game should have a variety of different interfaces such as a start and pause screen for users to interact with.
Previous game times of users should be stored for the possibility of later analysis and viewing.	The game will store its data in a database to be viewed by users on request.
Modular software structure should separate the game graphics and logic from haptics and physics-based simulation.	Keep code organised in many aptly named source files in logically built functions to separate different aspects.
Users should be able to play the game with other users or with an implemented AI.	Human + Human game-play would be unachievable provided only one haptic device. Human + AI can be added through the inclusion of an AI algorithm that can identify and successfully reach a goal state.

Table 2: Functional Requirements and Specifications

4 Implementation

4.1 AI Design

In our game, an AI collaborates with the user to reach a goal state. We have accomplished this using an implementation of the A* search algorithm to plan an optimal path based on current positions. The reason why we utilised this algorithm is that it can find the shortest path of state space search in a short period of time with a high search space. We defined evaluation functions ($f(n) = g(n) + h(n)$) to increase the possibility of finding the correct path. Specifically, $f(n)$ is the comprehensive priority of node n. When we choose the next node to traverse, we always choose the node with the highest comprehensive priority (the lowest value). $g(n)$ is the cost of node n from the starting point. $h(n)$ is the estimated cost of node n from the end point, which is the heuristic function of astar algorithm. We will explain the heuristic function in detail below. In the operation process of astar algorithm, the node with the lowest $f(n)$ value (the highest priority) is selected from the priority queue every time as the next node to be traversed. In addition, the astar algorithm uses two sets to represent the nodes to be traversed and the nodes that have been traversed, which is usually called open set and close set.

To implement the algorithm, we supposed the board was fixed and could not rotate, so we only required the x-coordinate to correspond to the row of the board array and the y-coordinate to correspond to the column of the array. We then predefined a hard-coded 2D array of size (25x25) representing the game board filled with zeroes and ones: zero corresponding to a space on the board covered by an obstacle and one referring to an open space.

Particularly, the point of origin in the real game board is the center of geometry and a coordinate system is created based on this point (The right direction is the positive direction of the x-axis, and the inward direction is the positive direction of the y-axis).

We also wrote functions to convert local coordinates of the ball in the real game board to the index of the 2D array and vice versa. Moreover, when the user controls the ball on the board, the A* algorithm will be applied periodically to update optimal paths towards the goal due to dynamic changes in the position of the ball, so we designed functions to find the next optimal position based on the current one and allow the AI to create corresponding forces between them.

For example, if the ball is located at a certain point on the board, the current real position is firstly converted to the corresponding index of 2D array and is regarded as the start point and the next optimal position (the new index of the 2D array where the ball needs to move to according to the A* algorithm) can be obtained. Then, this new index of the 2D array will be converted back to the real position on the board. Then a new force will be created based on these two real points.

When the ball moves to the next position (whether it is optimal or not), this position is now regarded as the start point and the rest can be done in the same manner. The newly created force is utilised in different versions (mouse version and haptic version). If the user chooses the mouse version, the ball will automatically move to the goal through forces guided by AI even if the user does not click. If the user chooses the haptic version, corresponding forces created by AI will be added to the haptic devices directly. So, the user will feel a stronger force feedback if the ball is closer to the goal, compared to if it was farther away. In the meantime, the visualised road map will be displayed in the command line to instruct the user on how to reach the final goal.

Although the A* search algorithm can obtain the optimal path with a bigger search space and in less time than other algorithms, there are some limitations to our implementation of the algorithm. Firstly, the index of the 2D array is discrete while the position of the ball is continuous. To illustrate, during conversion of coordinates, it is a many-to-one relationship which means several positions in the area correspond to one index of the 2D array. So if we intend to gain the real position of the ball in the game board only based on index of the 2D array, the gained real position is not very accurate and in this case, we just return the central point of the area that index of the 2D array corresponds to. Secondly, all created objects (like the ball and the obstacles) are three-dimensional, and the volume cannot be ignored but we have to use its centre of geometry to symbolise the whole object. Therefore, errors may be produced during navigation, leading to inaccurate directions of applied force. We have tried to minimise such errors in two ways. One is to divide the map more accurately, which means several zeroes in the 2D array represent one obstacle to decrease the influence of its volume. The other one is to utilise haptic devices to adjust parameters, such as original location of the ball, size of the ball, board or obstacles, the placement location of obstacles and so on, to increase the actual effects of the AI algorithm.

4.2 Potential Field

All forces in CHAI3D are represented as three-dimensional vectors (cVector3d class in CHAI3D library). The direction of the vector determines the direction of the force, while the length of the vector determines the magnitude of the force.

In our project, potential fields are used as a basic building block for generating forces. To simplify the problem, a simple linear force model is applied to the potential fields. It acts as a linear spring in three-dimensional space, which obeys Hooke's law (i.e., the force applied to the object is proportional to the distance between the object and the centre of the potential field):

$$F = -k(x - x') \tag{1}$$

Where:
k - A constant value symbolising the stiffness of the spring
x - The position of the object

x' - The position of the centre of the potential field

Multiple potential fields can coexist in the virtual environment, and the net force applied to the object is the vector addition of all forces on the object applied by different potential fields.

4.3 Mouse Controls

As for the mouse version of the game, the movement of the ball is mainly influenced by two potential fields: a mouse click potential field and astar potential field. The former is controlled by the user, which has the greater impact on the force applied to the ball. Whenever the user clicks the mouse button on the board, it will create a permanent potential field at the clicking position, attracting the ball to move in that direction. Since the mouse click potential field obeys Hooke's Law, the further the distance between the ball and the centre of the potential field, the greater the force will be. The second one (astar potential field) is controlled by the AI agent without any human interference. The main purpose of the A* potential field is to guide the ball through the right path to the final destination. In our updateHaptics function, the A* algorithm is invoked periodically, calculating the shortest path from the current ball position to the destination point, and then creating a potential field dragging the ball to the next position of the optimal path. Both the mouse click potential field and A* potential field can be enabled and disabled manually by pressing certain keys on the keyboard for testing purposes.

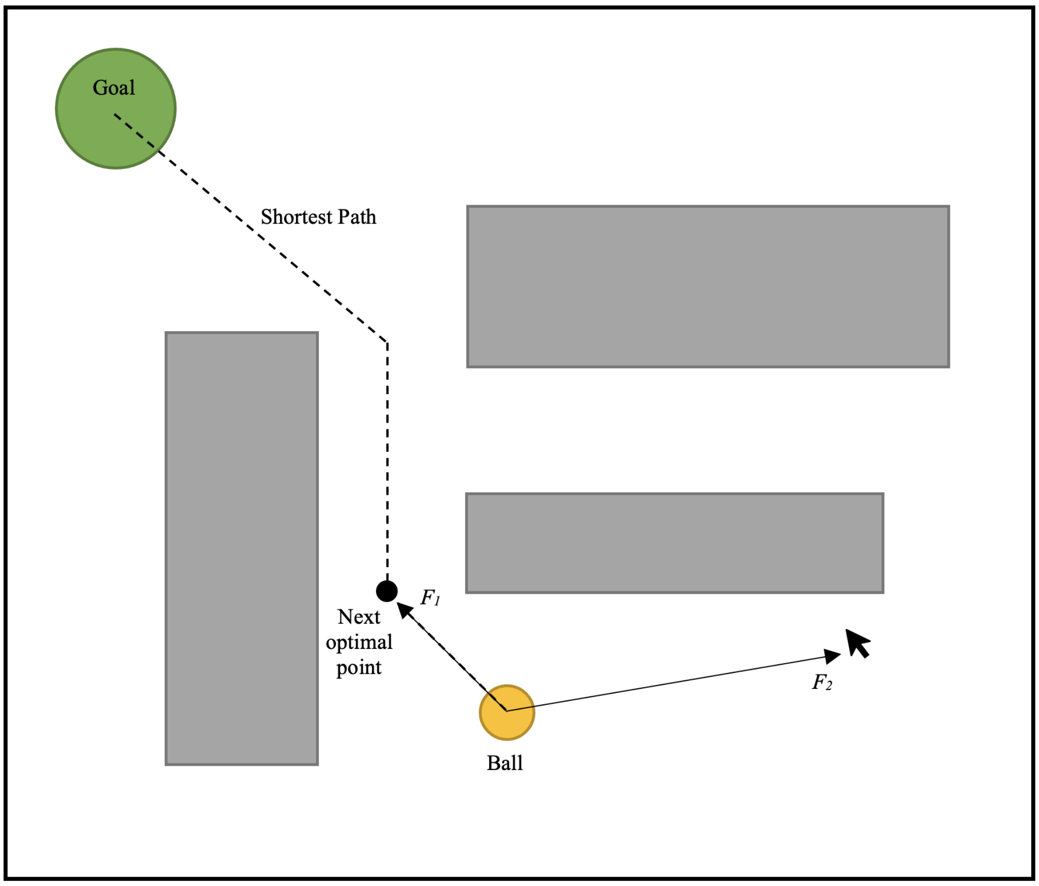


Figure 3: Free body diagram (Mouse Version)

The most distinct advantage of this mouse version is that the ball can automatically reach the goal using AI collaboration. So even if the user clicks in the wrong direction to create force, the ball will also be controlled by the correct direction of force created by AI and be back to the right path. However, there is a limitation. There will be stuttering or lag when the ball moves because the A* algorithm needs more time to obtain the optimal path when the map becomes larger. This may have a bad effect on the user's experience.

4.4 Haptic controls

As for the haptic version of the game, we decided to keep the haptic cursor as a moving potential field to affect the movement of the ball when rolling on the board, which is also the only outer force applied to the ball (except for preset environment forces such as gravity and friction). The haptic cursor (avatar) is represented as a black sphere in the virtual environment, whose position is directly controlled by the haptic device.

Similar to the mouse version of the game, the A* algorithm is invoked periodically in the game loop to find the optimal path when the ball is moving on the board. However, the force generated by A* algorithm is not exerted on the ball this time, but on the haptic device such that the user can feel the force directly. By calculating the optimal path, the next point on the optimal path is obtained, and a force pointing to the next point is applied to the haptic device in order to guide the user in the right direction.

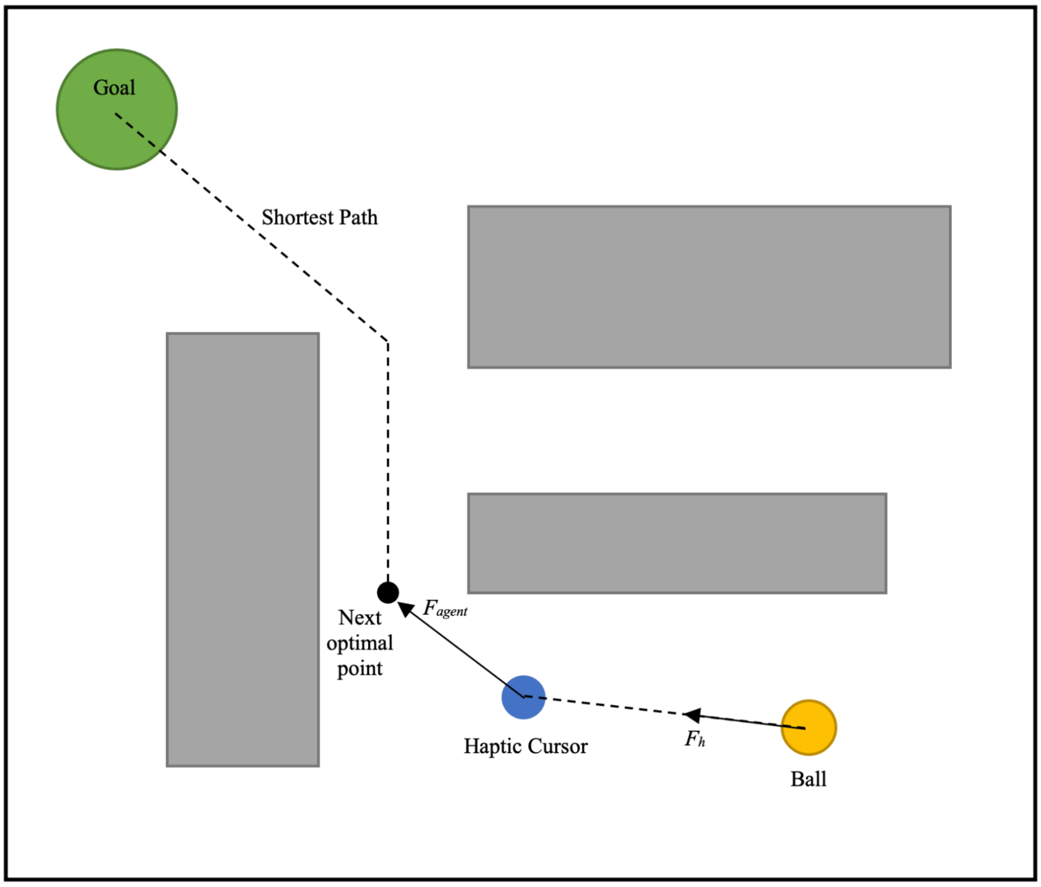


Figure 4: Free body diagram (Haptic Version)

The most distinct advantage of this haptic version is that users can feel strong force feedback when the ball moves towards the goal. Also, when it impacts obstacles, users feel collision forces, simulating a real experience. However, there are some limitations to this also. Firstly, we currently utilise haptic devices to control the ball, but it would be better to control the board. However, it would involve conversion of three-dimensional coordinates considering angle of inclination of the board, allowing the obstacles and the plate to have the same rotation angle and so on, which, when we attempted it, caused too many bugs that would have taken too long to correct. Secondly, the haptic version is independent from the mouse version, which means the user can only select one version, and if they need to select the other, they need to re-compile the API. The ideal way is to choose versions in the interface of the start menu, and the user can make conversion at any time, which would be convenient. However, we were not able to accomplish this in the time we were given.

4.5 Activity Diagram

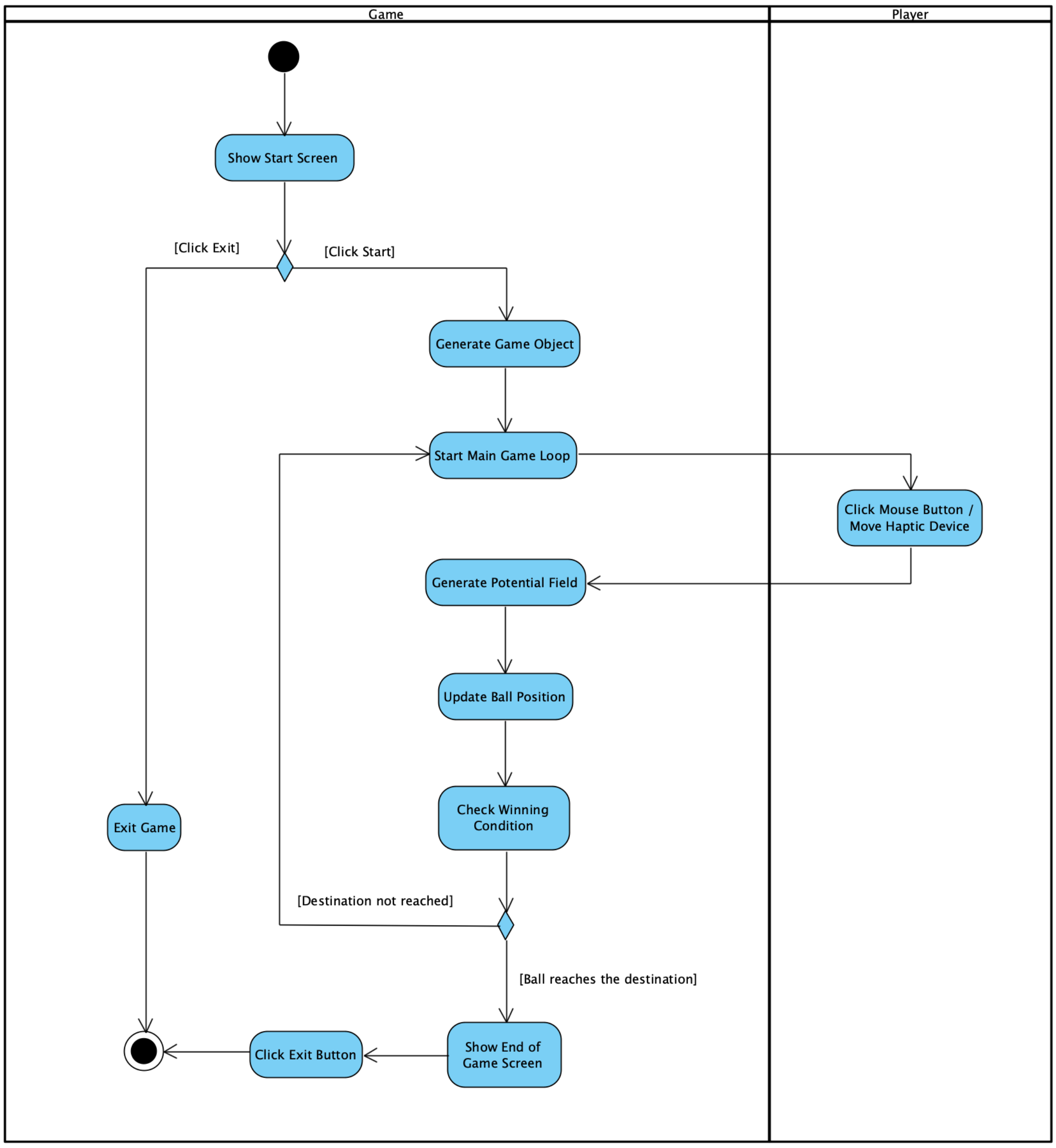


Figure 5: Pipeline of the game program

4.5.1 Explaining the Activity Diagram

When the program is launched, a start screen is displayed on the screen with a default background with an interface containing the "start" and "exit" buttons. If the user clicks the exit button, the program would terminate immediately. If the user clicks the start game button, the game would create a virtual environment (a cBulletWorld object) and all related game objects in the world. After the world is fully initialised, the program will execute the main game loop, waiting for user input. When the user clicks the mouse or moves the haptic device handle, the program will receive the input signal and create / modify the corresponding potential field in the world. The physics engine will sense the change of forces and move the object in the world, updating the ball position at the same time. At the end of the game loop, the program will check whether the ball has reached the goal. If the current ball position falls inside the preset destination range, the end game screen will be displayed and the user can click the exit button to terminate the program. If the winning condition is not met, the program will return back to the start of the game loop and wait for the user input repeatedly (see Figure 5).

4.6 Testing

Software testing plays a key role in validating whether the working software meets the requirements and works as expected. However, due to the attributes of our final project, we modified the testing plan that had been created before. Therefore, in this section, we discuss the details of our new testing plan.

4.6.1 Unit Testing

In our project, we used automated testing tools to test applicable method via a C++ Testing Framework called Catch because it takes a different approach (to both NUnit and xUnit) that is a more natural fit for C++ and it can be perfectly integrated into IDEs (Visual Studio / CLion) and is easy to implement. We introduced two test cases of the AI algorithm with the macro. One is for the simple map and the other one is for the complex one. Then we wrote the individual test assertions using the REQUIRE macro to check whether each method is carried out logically, such as whether obstacles are set correctly, whether the size of map is correct, whether the path from source point and destination point is accessible and so on. In addition, we also took the coverage of testing inputs into account, and checked memory leaks using Valgrind.

4.6.2 Integration Testing

In the phase of integration testing, the interfaces between components is verified, and the integrity of a single class should be examined. This makes automated testing difficult to implement. Moreover, integration tests usually involve abundant codes, which has an impact on the ease of localising the fault when failures happen. Hence, to mitigate this issue, it is expected to divide the larger tests into smaller pieces, which means manually testing each single object created in the game world through haptic devices. For example, to test the nature or behavior of “Board”, we just see whether its nature behaves as expected in the GUI (like sizes, textures and colors) and utilise output parameters of haptic devices to check whether the return values correspond to outputs.

4.6.3 System Testing

The main job in the phase of system testing is to check whether the complete software configuration items can be correctly connected with the system, and find out the nonconformity or contradiction between the software and the requirement documents in the real system working environment. Particularly, manual testing is more practical in our project. We used the laboratory PC in the Cobot Maker Space with haptic devices to play the whole game. Meanwhile, we confirmed the maintainability and error recovery functions of the software.

However, there are some limitations to our testing. Firstly, we manually tested frequently, which takes a lot of time. Instead of manually testing, we could have used methods such as writing testing scripts to save time and to follow the guidelines of Test-Driven Development (TDD), all testing scripts should be done before the real implementation of methods. Secondly, we mainly do tests for the AI part, but only simple and tiny ones for game interface and objects. We should have done tests more in those aspects, like using automated/visualised GUI Testing Framework.

5 Management

5.1 Our Initial Agile-Based Approach

Agile development takes an iterative, step-by-step approach to software development, with the evolution of the client’s needs at its core. In Agile development, a software project is initially built and divided into sub-projects, the results of which are tested, visualised, integrated and ready for use. In our project, we tried initially to pair scrum and paired programming with fortnightly sprints. Paired programming proved to be very effective both in terms of motivation to get through the workload, and pooling knowledge to come up with solutions more effectively to solve complex problems – a crucial aspect given the trouble we had learning the CHAI3D framework.

5.2 Time Management

Initial progress with implementation was slow. We had never seen C++ used at this level before and we wanted to be sure that we learned how to use it properly before we began with implementation. This resulted in much wasted time where we tried to learn tools that were out of scope, such as OpenGL and the lower level structure of the CHAI3D framework and its modules, whereas instead we should have been learning how to use it to create our own code. Once this became apparent we shifted our focus to CHAI3D's documentation and began learning how to create our own example programs. Regretfully, due to this poor time management, some of the smaller, less vital requirements gradually grew out of scope and we were no longer able to accomplish them.

5.3 A Change in Methodology

Despite our successful plan to utilise paired programming, we did find that after we started coding, we had some problems following the methodology. Daily scrums as described did not work given the part-time nature of the project. We also found that some tasks we set were far more difficult than others and they ended up having to be delayed by multiple sprints. This sub-optimal approach continued until near the end of March, but eventually, we realised that because of our lack of experience programming with professional level C++, how we functioned as a group and the fact that it was part-time, we were unable to keep to scrum's key values. We discussed this problem in an emergency meeting and unanimously decided to change methodology in the hopes that we might be more productive for the rest of the project duration.

After that discussion, we found Kanban to be much more effective in terms of productivity. After the switch, our Trello boards that we had been neglecting before began to be used consistently and we made sure to be thorough in detailing all tasks and their paths through the boards. Once we had completed a Trello task, we would also leave a comment under it showing the commit we made on Git where said task was accomplished. Please follow our Trello board link to see how we utilised boards and tasks.

5.4 Our Meeting Format

Prior to the change, our meetings consisted of all of us deciding to meet up on a day and then all work together on the project at the same time. This was a seriously inefficient way to manage our time spent on the project and required immediate correction. Alongside our change in methodology, we also altered our meeting format.

Our new meeting structure was as such:

- Start with a recap of the tasks we had assigned ourselves in the meeting before, how we got on with them and what trouble we had.
- We then all collaborated to work on problems that were crucial to solve before we could continue.
- When the meetings ended we would organise a time to meet next and assign ourselves new or existing tasks either as individuals or in groups to complete before we met next. We would always arrange meeting times dynamically based on the difficulty of our new tasks.

We would hold these meetings frequently at an average of two to three times per week to stay on top of progress and workload separation. Our team had near-perfect attendance at these meetings. Except for a low number of outliers, all team members were always present at every meeting we held. They would always be held in person whenever possible and those who were not able to attend in person would be highly encouraged to participate through Microsoft Teams.

This facilitated a high level of participation and communication between team members. For more informal conversation, a popular live group chatting service was used to encourage frequent further discussion.

As a result of changing methodology and our meeting format, our work output drastically increased. This is evident by the density of our commits on our repository page (see Figure 6).

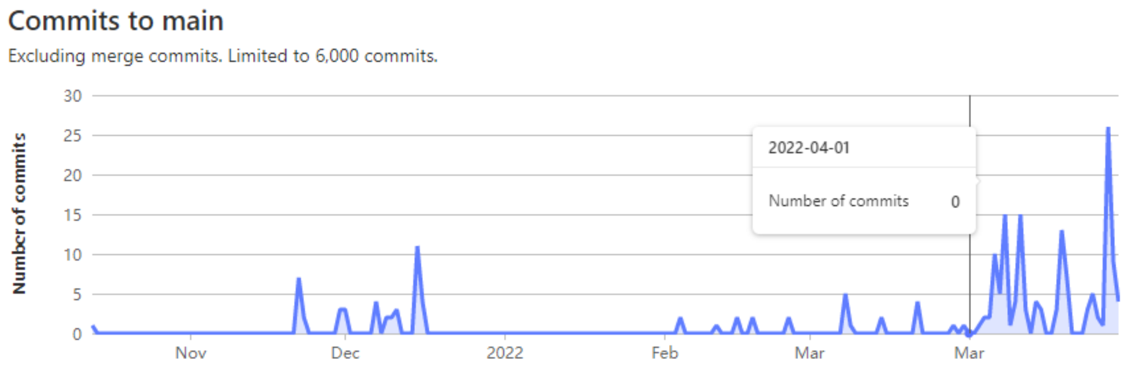


Figure 6: A graph of our commits to Git throughout the project

5.5 Git usage

We found Git to be a vital part of the organisation of our project. Keeping a master branch where a stable version of the project was crucial to our success. We would rarely commit code changes directly to this branch, and instead create branches with a logical and consistent naming scheme, corresponding to the user's university name and the purpose of the branch:

psyabc/branch

We would create a branch from main, implement our additions or changes to the codebase, make sure that the changes would compile correctly, utilise user testing and our testing framework to ensure that our changes did not cause any instability in the running program, and then request a merge from the Git Master back into the master branch. Git also allowed us to keep our codebase and progress with documentation in the same place, accessible by every member. After switching from Scrum to Kanban, we found ourselves using Git issues, milestones and labels less and eventually phased them out completely. We felt that the job meant for management tools on the repository page was eclipsed by Trello cards and the only major milestone we had left was the project deadline. In many projects all of these tools can be very useful for organisation, but given that we were a small team on a relatively small project, we made a conscious choice to keep it simple and use only Trello.

6 Successes and Achievements

Overall, despite certain struggles we faced throughout the project timeline, the process in its entirety has been very successful in teaching us the procedures and level of organisation required to complete such a project. Although at times we struggled with time management and task organisation, it was a good learning experience for us to gain perspective on how most software engineering projects will look post-graduation. Our organisation skills have improved as we are now more accustomed to long-term time-planning and task delegation. We now feel much more comfortable coding in a group environment and using resources such as GitLab and Trello to keep our work organised. Having to work in a group with other peers we were not familiar with was also very effective in preparing us for future projects in the real-world, where we won't necessarily always get a choice on who we work with.

In terms of the end product, although there were certain brief requirements we were not able to successfully implement, the final deliverable is complete and very similar to the product we initially envisioned. All of the high priority requirements we set out to finish were successfully included into the game. The final game architecture has full functionality, including a number of interfaces for users to interact with. We were also able to incorporate the Haption Virtuose Device into our product and program it to provide the users with force feedback to assist them in reaching the goal. The AI algorithm is also effective in correctly identifying the optimal path to the game goal and correctly modifies this path depending on the location of the ball on the board.

7 Summary

This project was a great hands-on experience with coding in a team environment and we found out that we were still making many mistakes throughout semester two. We had understood first-hand that communication skills are very important within a team environment and it is important to encourage everyone to speak up and discuss opinions to make better decisions as a team. We also learnt that implementing code in a team environment is very different from writing code individually and tools such as Git help with keeping our code in order. Having more developers does not necessarily mean the project will get completed sooner, instead, code quality will be higher due to the use of paired programming and work morale will be improved over programming individually.

8 Future Roadmap

If we were able to further develop this product, there are certain improvements that could be made for better haptic implementation as well as overall game functionality.

In terms of the haptic feedback, our game could be better developed by fine tuning the role exchange mechanism. Although our current AI interaction is designed to allow for force feedback to guide users to the goal if they slow down movement, it could be improved by allowing for more smooth role exchange so as to not obstruct user experience.

In terms of game functionality, one aspect that could be refined is our AI algorithm. Currently, the astar algorithm we implemented has been hard coded to recognise the different obstacles on the map. So given more time, we would also improve upon this to allow for the algorithm to independently recognise the obstacles on the board and adjust itself to find the fastest path to the end. The current game GUI could also be further developed as right now it is quite simple in its design. It could be made to look more aesthetically pleasing and detailed. A database to show previous completion times would also be effective in making the game more suitable for research. Currently, our example game only has one playable level, so another improvement would be to create more levels, ranging in difficulty, to make the game more challenging and competitive between users.

References

[1] Utilisateur, S., 2022. Virtuose™ 6D - HAPTION SA. [online] Haption.com. Available at: https://www.haption.com/en/products-en/virtuose-6d-en.html [Accessed 1 May 2022].

[2] GregorDS, 2015. Six degrees of freedom. [image] Available at: https://en.wikipedia.org/wiki/Sixdegreesoffreedom#/media/File:6DOF.svg [Accessed 1 May 2022].

[3] Ayse Kucukyilmaz, Tevfik Metin Sezgin, and Cagatay Basdogan. "Intention recognition for dynamic role exchange in haptic collaboration". In: IEEE transactions on haptics 6.1 (2012), pp. 58–68.

[4] Chai3d.org. 2022. CHAI3D - Home. [online] Available at: https://www.chai3d.org/ [Accessed 1 May 2022].

Software and User Manual

Note: *The target audience for our project is those who would use the architecture to construct their own haptic games according to their needs. They need the knowledge of the code structure in order to create the games to their specifications. As such, since they serve the same purpose, the software manual and user manual have been combined into one.*

Introduction

This manual was written for developers with instructions on how to work with our API. The haptic device related aspects of our project have only been tested with Ubuntu with the Virtuose 6D from Haption in the Cobot Maker Space at the University of Nottingham. The device has its software which works in tandem with the customised CHAI3D library in our repository. If our software is being used with any haptic devices or environments apart from the ones we tested it on, we cannot guarantee any haptic functionality. The repository will however work outside the Cobot Maker Space using the vanilla CHAI3D library if the user only wishes to use mouse controls.

If working in the Cobot Maker space:
The Virtuose 6D is a piece of fragile and expensive hardware and we would suggest that only a seasoned developer with robotics experience work with it. Online support for this device is limited.

This software may not be portable and has been designed to only work in an Ubuntu environment.

CHAI3D Documentation: https://www.chai3d.org/documentation/getting-started

CHAI3D documentation also comes packaged in the library itself:
https://projects.cs.nott.ac.uk/comp2002/2021-2022/team21_project/-/tree/main/chai3d-3.2.0/doc

BULLET Documentation comes packaged within the module's main directory:
https://projects.cs.nott.ac.uk/comp2002/2021-2022/team21_project/-/tree/main/chai3d-3.2.0/modules/BULLET/doc

Contents:

Installing required packages and tools	1
Cloning the repository	2
Building the CHAI3D libraries	2
Building the board game architecture API files	3
Running the provided example	3
The Haption Virtuose 6D	4
Overview of code structure	6
The CHAI3D and Bullet libraries	7
The Board Game Architecture	8
Building your own game	14
Coding conventions and commenting	17
Testing	18
Glossary	19

Installing required packages and tools

Assuming you are starting from a fresh install of Ubuntu:

Begin by running this command:

```
sudo apt update
```

This will download information on all installed packages from the internet in order to get information about updated versions of packages or their dependencies.

Git, Make and g++

```
sudo apt install Git
sudo apt install build-essential
```

Make and g++ are included in the build-essential package.

CHAI3D requires the following packages:

libusb-1.0 development package:

```
sudo apt-get install libusb-1.0-0-dev
```

ALSA (Advanced Linux Sound Architecture) development package:

```
sudo apt-get install libasound2-dev
```

FreeGLUT development package:

```
sudo apt-get install freeglut3-dev
```

It is not stated in the documentation, but we found that we also required the following packages from the **xorg-dev** package:

```
libxcursor-dev
```

```
libxrandr-dev
```

```
Libxinerama-dev
```

These can be acquired individually, or all at once:

```
sudo apt install xorg-dev
```

Cloning the repository

Once all the required packages are required, the building of the repository can begin.

Using a terminal in Ubuntu, clone the project from Gitlab

```
git clone
https://projects.cs.nott.ac.uk/comp2002/2021-2022/team21_project.git
```

You are cloning over 1GB of files from the repository, this should take a while.

Building the CHAI3D libraries

Once cloning has finished, build the CHAI3D library or libraries you require

Building the **customised CHAI3D** library for haptic device integration, assuming you start at the top-level directory of the repository (/team21_project/)

```
cd chai3d

make
```

This should build the main library files, however, you still need to build the BULLET module files.

```
cd modules/BULLET

make
```

This should build all necessary files for the Bullet module.

Building the **vanilla CHAI3D** library for simple mouse interaction, assuming you start at the top-level directory of the repository (/team21_project/)

```
cd chai3d-3.2.0

make

cd modules/BULLET

make
```

After everything you require is built, you can move on and build the board game architecture API files.

Building the board game architecture API files

Assuming you start from the top-level directory (/team21_project/)

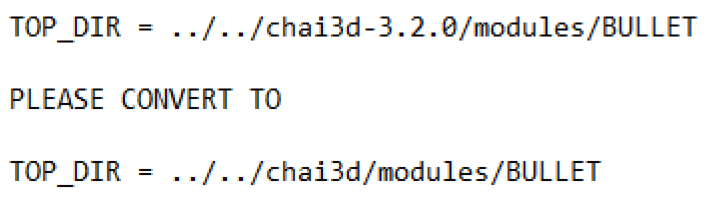
Navigate to the example game directory

```
cd BoardGameArchitecture/haptic\ games/maze
```

Please consider which CHAI3D library you wish to build with. By default, both makefiles are configured to use the **vanilla CHAI3D** library.

If you wish to compile the API with the **customised CHAI3D** library please change these two lines at the top of each makefile:

team21_project/BoardGameArchitecture/src/Makefile
team21_project/BoardGameArchitecture/haptic games/maze/Makefile



```
TOP_DIR = ../../chai3d-3.2.0/modules/BULLET

PLEASE CONVERT TO

TOP_DIR = ../../chai3d/modules/BULLET
```

Provided the makefiles are in the configuration you wish, please run this command to build.

```
make output
```

This command should run the makefile provided in the /maze/ directory and also compile all the files in /src/ and link with the correct CHAI3D library.

Running the provided example

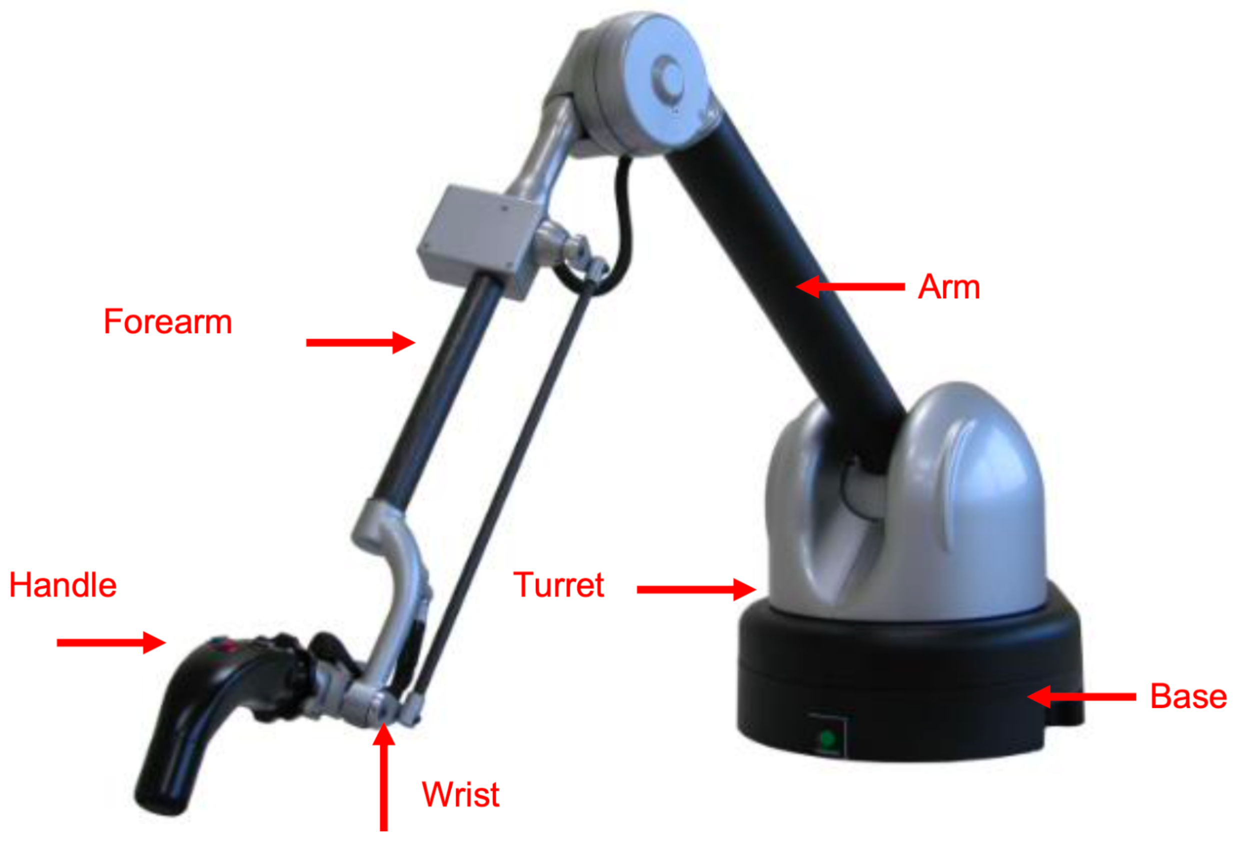
Assuming you start at the top-level directory (/team21_project/) To run the game example enter the following commands:

```
cd BoardGameArchitecture/bin

./mazeGame
```

The Haption Virtuose 6D

Introduction

Virtuose 6D is a haptic device manufactured by Haption, a company located in France. The Virtuose 6D is made up of two main articulated segments fixed on a rotating base. The second segment concludes with an articulated wrist, rotating around three concurrent axes. Hence, the haptic interface is a six degrees-of-freedom device and provides force feedback in every direction.



The **base** contains one motor and encoder, and provides the signals interface to the control unit. The motor torque is transmitted to the turret rotation using one capstan drive.

The **turret** contains two motors and encoders, and the weight compensation system. The motor torques are transmitted to the arm and forearm joints using two capstan drives and one connecting rod.

The **arm** contains only the electric connection for the signals of the forearm and the connecting rod for the forearm joint.

The **forearm** contains two motors and encoders. The motor torques are transmitted to the forearm and wrist using harmonic drives and a connecting rod to the wrist.

The **wrist** contains one motor and encoder and the tool changer for the handle. The motor torque is transmitted using one harmonic drive.

Technical Specifications

Motors: Type DC - Output power 150 W (axes 1 to 3) and 20 W (axes 4 to 6) in 48V

Power supply: 240 VAC one-phase

Operational workspace:

- 450mm x 450 mm x 450 mm ; 300° - 100°- 250°

Translation force:

- 31 N (maximum), 8.5 N (continuous)

- for High Force configuration 70 N (maximum) 25 N (continuous)

Rotation torque (6D configuration) :

- 3.1 Nm (maximum), 1.0 Nm (continuous)

Control stiffness:

- 1800 N/m (translation), 10000 N/m (High Force configuration translation)

- 4 Nm/rad (6D configuration rotation)

Apparent inertia: 1400 g

External dimensions: H 1080 x L 1300 x l 658 mm

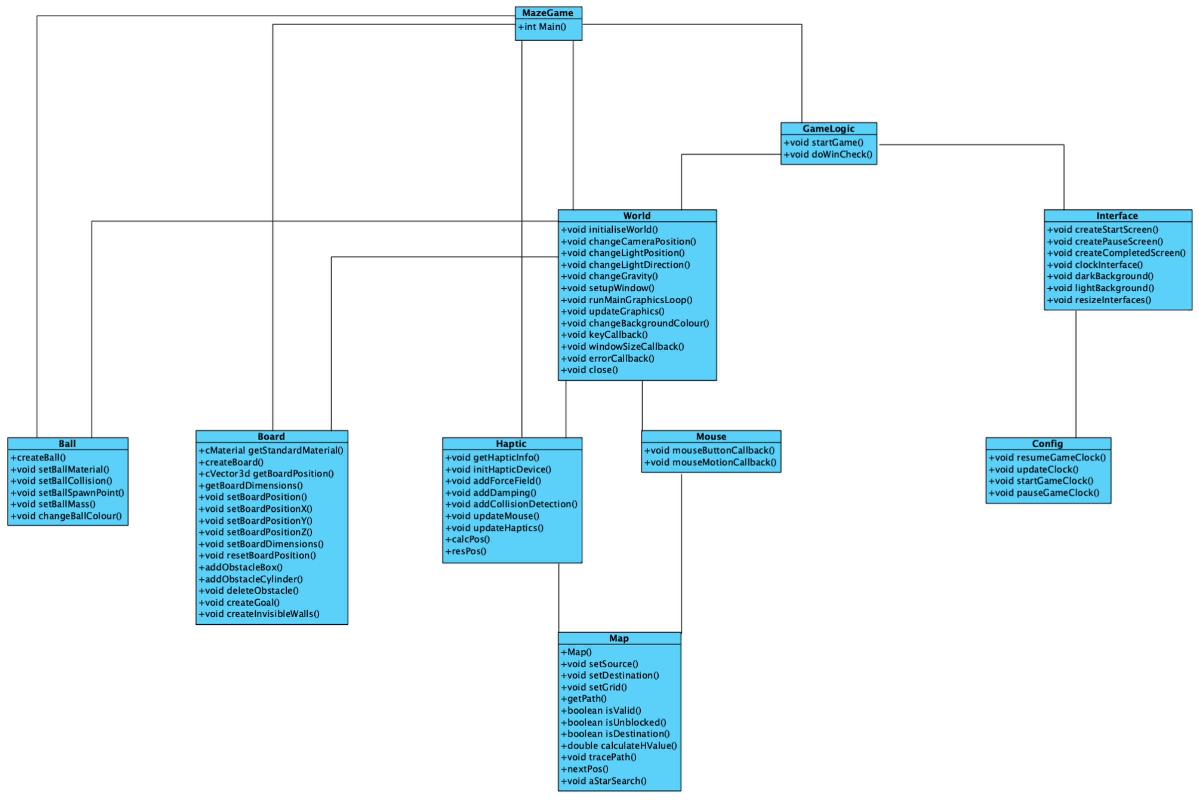
External dimensions of the power supply box: H 50 x L 290 x l 205 mm

External Links

CHAI3D Haptic Devices Introduction:
https://www.chai3d.org/download/doc/html/chapter6-haptic-devices.html#section6-1

Virtuose 6D brochure: https://cdn.shopify.com/s/files/1/1750/5061/files/Datasheet-Virtuose6D-2019.pdf?109

Overview of Code Structure



Above is a diagram displaying the hierarchical structure of the game code and how the different classes interact with each other. The MazeGame, GameLogic and World files are the ones that require a variety of functions and variables from the other files in the architecture.

The MazeGame file is the file that runs the example game, so it needs a range of functions from the different API source files to actually set up the game with its individual objects. The World file contains the bulk of the code to initialise the game world, as well as set up the window and visuals required to play the game, thus requiring functions and variables from a majority of different files. The GameLogic file handles the actual graphics and functionality, and thus needs to incorporate the World and Interface files to set up the gameplay and interactive screens.

It is also important to note that some of the function return types are not included within the diagram as they would cause a syntax error on Visual Paradigms. Refer to the *Board Game Architecture* section of the manual for complete function definitions.

CHAI3D and Bullet libraries

The framework used is CHAI3D which is an open-source set of C++ libraries for computer haptics, visualisation and interactive real-time simulation. The framework has many benefits such as abundant force rendering algorithms, which are utilised to compute the interaction forces among objects, for instance, the ball is moved by the mouse. It also provides the necessary data structures required to realise static, dynamic and articulated bodies in different scenarios. It also supports the lightweight OpenGL-based graphics engine which can easily create Mesh Objects, Line Segments and Volume Objects with surface materials and texture properties. Furthermore, it supports many physics engines such as ODE (Open Dynamics Engine) or Bullet physics Engine.

Bullet is a physics engine that is compatible with both Python and C++ which simulates collision detection as well as soft and rigid body dynamics. Physics engines are beneficial as they save developers the stress of having to code collisions between objects and the forces that occur during this using mathematical equations.

CHAI3D (and the Bullet library stored in the CHAI3D folder) are used to build our applications, in our makefile certain directories are targeted in order to compile and run our program/game. The main chai3d.h is the main header file used to access all chai3d objects and classes, without this, main components for the game cannot be built such as a world, panel or even colour.

The Board Game Architecture

The architecture we used to design this game consists of a series of source files that build up the different objects needed to create, modify and run the game. These are not class files but rather individual source files with their corresponding header files. They are made to contain global variables so their separate functions can then be combined to create the interface and game functionality. Below is a comprehensive list of the different files within the architecture, as well as the different functions within each of them:

Game

```
Sets up the game logic and screen interfaces
void startGame();

Constantly checks whether the ball has reached the goal to end the game
void doWinCheck();
```

World

```
Initialise the world and all its default settings
void initialiseWorld();

Change camera angle to a different set of vectors
void changeCameraPosition(cCamera* camera, cVector3d cameraPos, cVector3d
lookatPos, cVector3d upVector);

Change position of the light
void changeLightPosition(cVector3d newLightPosition);

Change direction of the light
void changeLightDirection(cVector3d newLightDirection);

Change gravity of the world
void changeGravity(cVector3d newGravity);

Sets up the window to view the game on
void setupWindow();

Runs the main graphics loop to keep game visible
void runMainGraphicsLoop();

Updates the graphics of the game
void updateGraphics();

Changes worlds background colour to black
void changeBackgroundColor(cBulletWorld* world1);
```

Callback when a key is pressed
void keyCallback(GLFWwindow* a_window, int a_key, int a_scancode, int a_action, int a_mods);

Callback when the window display is resized
void windowSizeCallback(GLFWwindow* a_window, int a_width, int a_height);

Callback when an error occurs
void errorCallback(int a_error, const char* a_description);

Deletes all game objects and closes the game window
void close();

Haptic

Reads the info of the Haptic device
void getHapticInfo(void);

Initialises the haptic device
void initHapticDevice(chai3d::cBulletWorld *world);

Adds a force field to the device
void addForceField(chai3d::cVector3d& force, chai3d::cVector3d desiredPos, chai3d::cVector3d position, double Kp);

Computes and adds damping force to the device
void addDamping(chai3d::cVector3d& force, chai3d::cVector3d linearVelocity);

Adds collision detection to the BulletMesh object
void addCollisionDetector(chai3d::cBulletMesh *obj, double toolRadius);

Updates the mouse position
void updateMouse(void);

Updates the devices position
void updateHaptics(void);

Maps (x, y) coordinate to index of 2d array
pair<int, int> calcPos(double x, double y)

Maps index of 2D array to (x, y) coordinate
pair<double, double> resPos(int row, int col);

Mouse

Callback to handle mouse click
void mouseButtonCallback(GLFWwindow* a_window, int a_button, int a_action, int a_mods);

Callback to handle mouse motion
void mouseMotionCallback(GLFWwindow* a_window, double a_posX, double a_posY);

Map

Sets the start point
void setSource(int row, int col);

Sets the destination point
void setDestination(int row, int col);

Sets the grid of the map
void setGrid(int **grids);

Gets the optimal path
stack<Pair> getPath();

Checks whether given cell (row, col) is a valid cell or not
bool isValid(int row, int col) const;

Checks whether the given cell is blocked or not
bool isUnBlocked(int row, int col);

Checks whether destination cell has been reached or not
bool isDestination(int row, int col) const;

Calculates the 'h' heuristics
double calculateHValue(int row, int col) const;

Traces the path from the source to the destination
void tracePath(cell **cellDetails);

Finds the next position based on the given start point
pair<int, int> nextPos(int src_r, int src_c);

Finds the shortest path between source cell to destination cell based on A* Search Algorithm
void aStarSearch();

Interface

Creates start screen
void createStartscreen();

Create pause screen
void createPausescreen();

Creates goal screen
void createCompletedScreen(int timetaken, int level);

Adds clock panel
void clockInterface();

Changes interface screen to dark mode
void darkBackground ();

Changes interface screen to light mode
void lightBackground ();

Updates interface sizes when window changes
void resizeInterfaces();

Config

Resumes clock after pause
void resumeGameClock();

Shows clock counting
void updateClock();

Start the clock at the beginning of the game
void startGameClock();

Pauses game clock
void pauseGameClock();

Board

Retrieve a standard material to apply to an object
cMaterial getStandardMaterial();

Creates a board with set dimensions (2.5, 2.5, 0.05) at (0,0,0)
cBulletBox* createBoard(cBulletWorld* container);

Creates a board with a set thickness (0.05) at (0,0,0)
cBulletBox* createBoard(cBulletWorld* container, double length, double width);

```
Creates a board at global position (0,0,0) in a world
BulletBox* createBoard(cBulletWorld* container, double length, double width,
double depth);

Get the position of the centre of a board in the world.
cVector3d getBoardPosition(cBulletBox* board);

Get dimensions of a board. Returns a 3-tuple <x,y,z>
std::tuple<double, double, double> getBoardDimensions(cBulletBox* board);

Set the position of the centre of the board in a world
void setBoardPosition (cBulletBox* board, double posX, double posY, double
posZ);
void setBoardPositionX(cBulletBox* board, double pos);
void setBoardPositionY(cBulletBox* board, double pos);
void setBoardPositionZ(cBulletBox* board, double pos);

Sets dimensions of the board
void setBoardDimensions(cBulletBox* board, double length, double width, double
depth);

Resets the board position to its initial position
void resetBoardPosition();

Adds a CBulletBox obstacle onto the board
cBulletBox* addObstacleBox(cBulletWorld* container, double sizeX, double sizeY,
double sizeZ, double X, double Y, double Z);

Adds a CBulletCylinder obstacle onto the board
cBulletCylinder* addObstacleCylinder(cBulletWorld* container, double height,
double radius, double X, double Y, double Z);

Deletes passed in obstacle from the board
void deleteObstacle(cBulletWorld* container, cBulletBox* object);

Creates a goal for ball to reach on the board
void createGoal(cBulletWorld* container, cVector3d position, double height,
double radius);

Creates invisible walls around the board
void createInvisibleWalls(cBulletWorld* container);
```

Ball

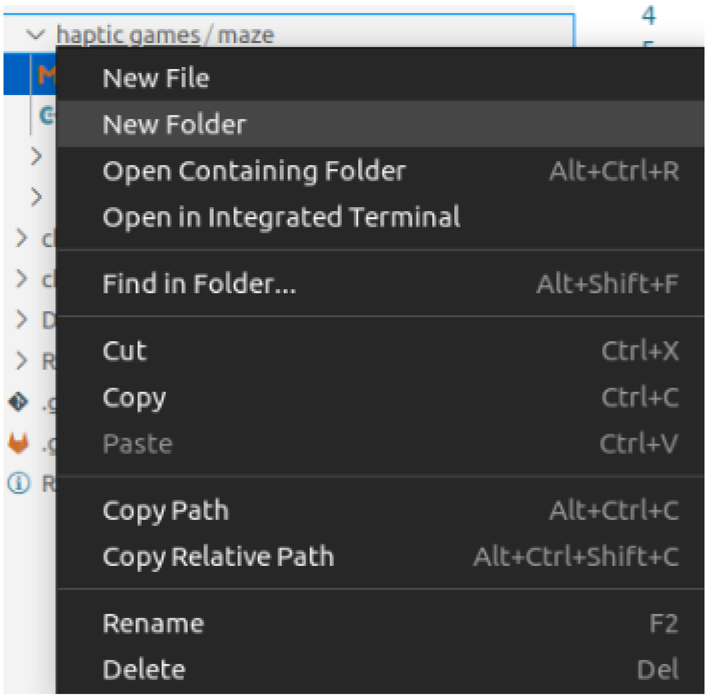
```
Creates the ball
cBulletSphere* createBall(cBulletWorld* world, double radius);

Sets material of the ball
void setBallMaterial(cBulletSphere* ball);

Sets collision detector for the ball
void setBallCollision(cBulletSphere* ball, double toolRadius);

Sets the spawn location for the ball
void setBallSpawnPoint(cBulletSphere* ball, double x, double y, double z);

Sets the mass of the ball
void setBallMass(cBulletSphere* ball, double mass);

Changes colour of the ball to blue
void changeBallColor(cBulletSphere* ball, cMaterial mat);
```

Building your own game

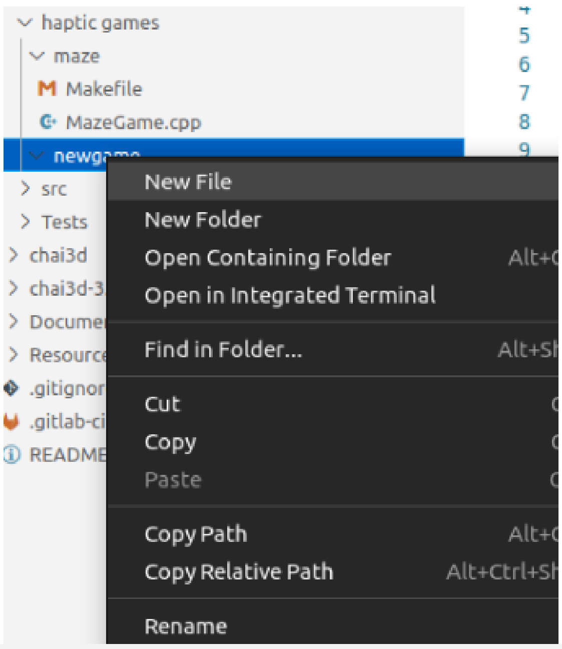
This is a step-by-step tutorial on how to construct your own game using the architecture. This tutorial uses Visual Studio Code.

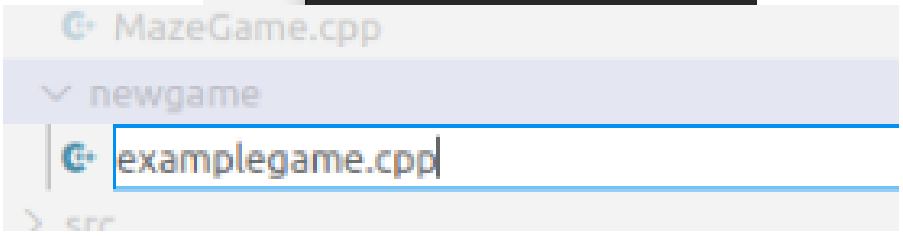
1. Open the top level directory of the repository in Visual Studio Code

2. Create a new folder by right clicking on “haptic games”

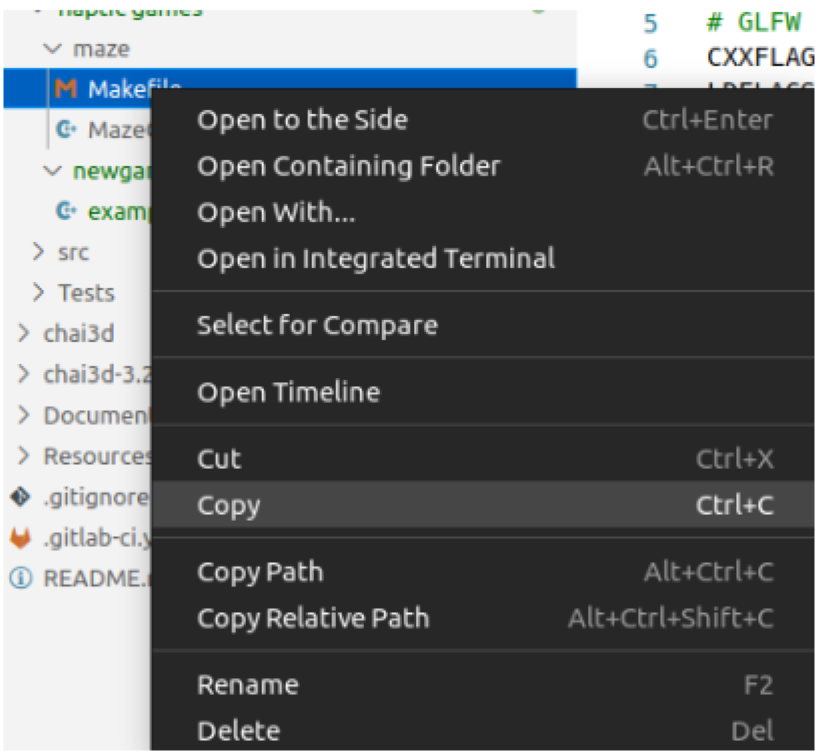

haptic games / maze
New File
New Folder
Open Containing Folder Alt+Ctrl+R
Open in Integrated Terminal
Find in Folder... Alt+Shift+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy Path Alt+Ctrl+C
Copy Relative Path Alt+Ctrl+Shift+C
Rename F2
Delete Del

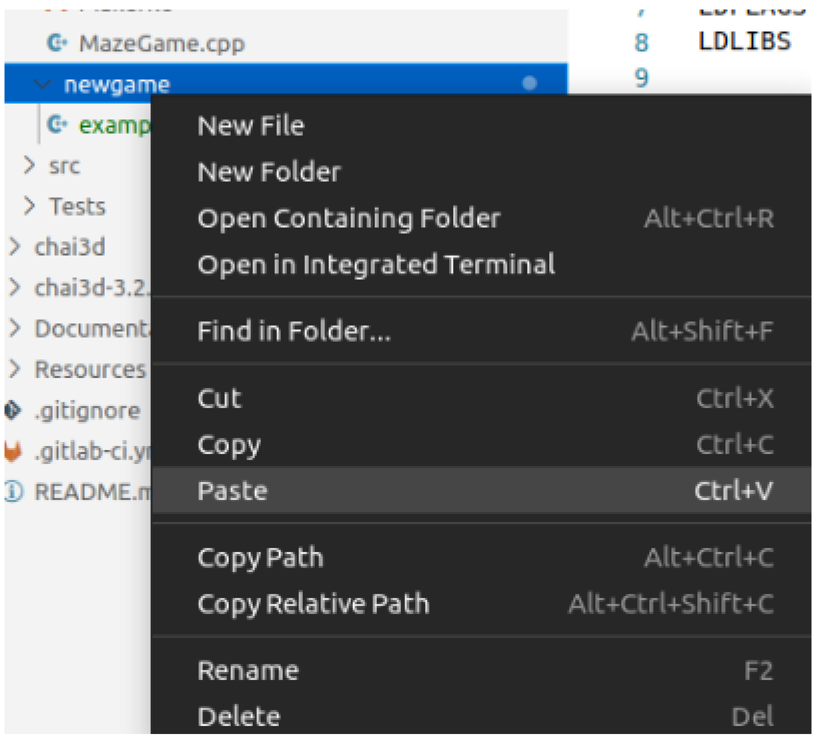
3. Create a new .cpp file to house your example



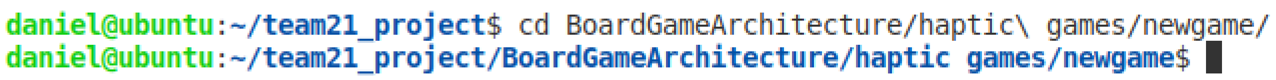


4. Copy and paste the Makefile over from the provided example game



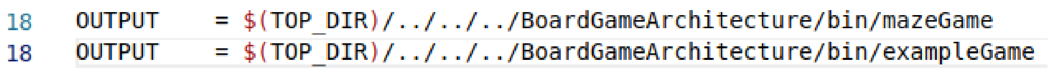


5. Navigate to this new directory using a terminal



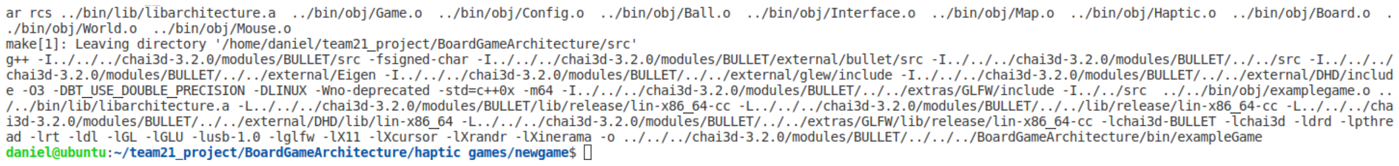
```
daniel@ubuntu:~/team21_project$ cd BoardGameArchitecture/haptic\ games/newgame/
daniel@ubuntu:~/team21_project/BoardGameArchitecture/haptic games/newgame$ 
```

6. Modify line 18 of the new Makefile so that the output executable will have the correct name:



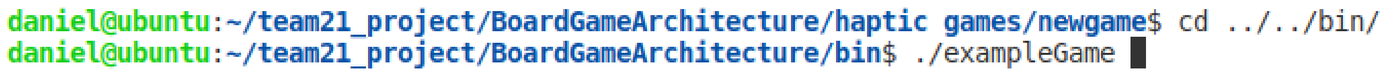
```
Old    18  OUTPUT      = $(TOP_DIR)/../../../BoardGameArchitecture/bin/mazeGame
New    18  OUTPUT      = $(TOP_DIR)/../../../BoardGameArchitecture/bin/exampleGame
```

7. Save your changes to the makefile and type “make output” in the terminal to ensure that this new makefile is functional. You should see this as the last few lines of the output:



```
ar rcs ../bin/lib/libarchitecture.a  ../bin/obj/Game.o  ../bin/obj/Config.o  ../bin/obj/Ball.o  ../bin/obj/Interface.o  ../bin/obj/Map.o  ../bin/obj/Haptic.o  ../bin/obj/Board.o  .
./bin/obj/World.o  ../bin/obj/Mouse.o
make[1]: Leaving directory '/home/daniel/team21_project/BoardGameArchitecture/src'
g++ -I../../../chai3d-3.2.0/modules/BULLET/src -fsigned-char -I../../../chai3d-3.2.0/modules/BULLET/external/bullet/src -I../../../chai3d-3.2.0/modules/BULLET/../../src -I../../../
chai3d-3.2.0/modules/BULLET/../../external/Eigen -I../../../chai3d-3.2.0/modules/BULLET/../../external/glew/include -I../../../chai3d-3.2.0/modules/BULLET/../../external/DHD/includ
e -O3 -DBT_USE_DOUBLE_PRECISION -DLINUX -Wno-deprecated -std=c++0x -m64 -I../../../chai3d-3.2.0/modules/BULLET/../../extras/GLFW/include -I../../src  ../../bin/obj/examplegame.o ..
/../bin/lib/libarchitecture.a -L../../../chai3d-3.2.0/modules/BULLET/lib/release/lin-x86_64-cc -L../../../chai3d-3.2.0/modules/BULLET/../../lib/release/lin-x86_64-cc -L../../../cha
i3d-3.2.0/modules/BULLET/../../external/DHD/lib/lin-x86_64 -L../../../chai3d-3.2.0/modules/BULLET/../../extras/GLFW/lib/release/lin-x86_64-cc -lchai3d-BULLET -lchai3d -ldrd -lpthre
ad -lrt -ldl -lGL -lGLU -lusb-1.0 -lglfw -lX11 -lXcursor -lXrandr -lXinerama -o ../../../chai3d-3.2.0/modules/BULLET/../../../BoardGameArchitecture/bin/exampleGame
daniel@ubuntu:~/team21_project/BoardGameArchitecture/haptic games/newgame$ 
```

8. You can now add features to your new game source file to create a board game. “Make output” will generate your executable and you should navigate to the bin folder to run it:

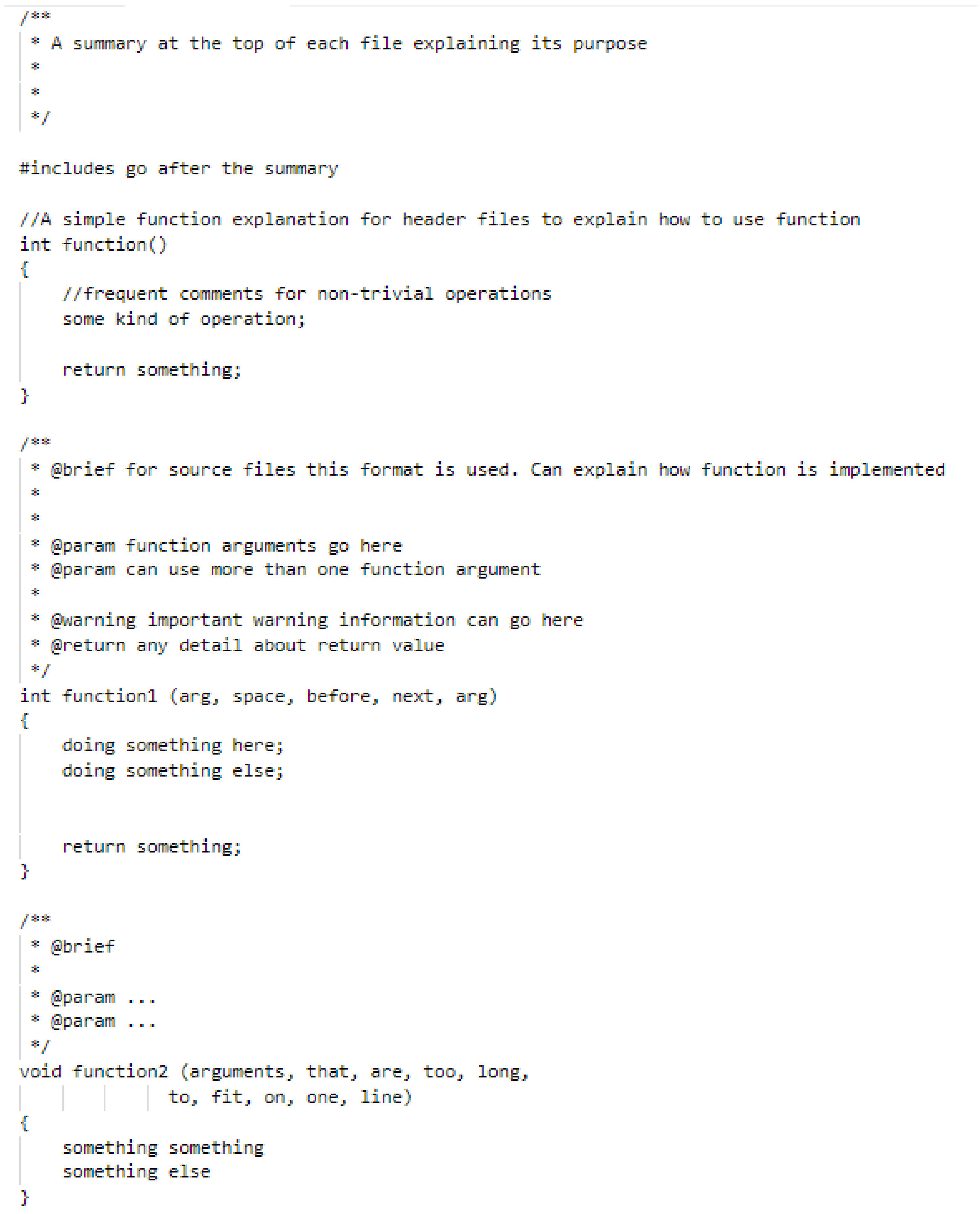


```
daniel@ubuntu:~/team21_project/BoardGameArchitecture/haptic games/newgame$ cd ../../bin/
daniel@ubuntu:~/team21_project/BoardGameArchitecture/bin$ ./exampleGame
```

Coding conventions and commenting

In order to make the code more organised and easy to understand, some coding conventions have been followed throughout the coding process. This has been done to improve the readability of the software and allow for further developers of the game to understand the code more quickly and thoroughly. The coding conventions we used include:

- Use tabs over spaces
- A Maximum of 80 characters per line
- Use only standard english alphabet characters when naming files
- Keep file names simple with first letter capitalised: Board.cpp
- Keeping to established directory structure
- Avoid code duplication when possible



```
/**
 * A summary at the top of each file explaining its purpose
 *
 *
 */

#includes go after the summary

//A simple function explanation for header files to explain how to use function
int function()
{
    //frequent comments for non-trivial operations
    some kind of operation;

    return something;
}

/**
 * @brief for source files this format is used. Can explain how function is implemented
 *
 *
 * @param function arguments go here
 * @param can use more than one function argument
 *
 * @warning important warning information can go here
 * @return any detail about return value
 */
int function1 (arg, space, before, next, arg)
{
    doing something here;
    doing something else;


    return something;
}

/**
 * @brief
 *
 * @param ...
 * @param ...
 */
void function2 (arguments, that, are, too, long,
              to, fit, on, one, line)
{
    something something
    something else
}
```

Testing

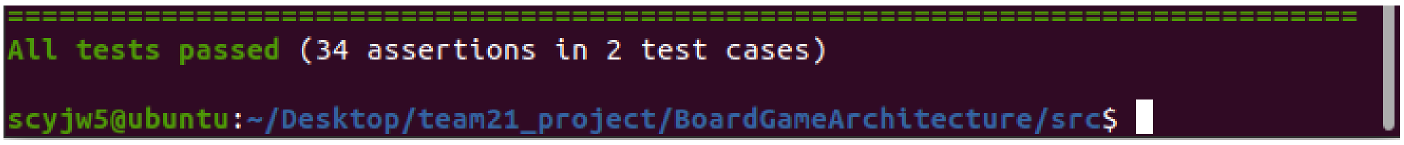
User Testing
A very important purpose of user experience testing is to determine whether our products can be quickly accepted and used by users. If there are problems, such as finding that the page structure is not in line with the user's operation habits, or some functions need to be strengthened for the user, or the operation steps are too complicated, we can modify and optimise the code in time. Specifically, we test whether or not the start menu appears properly, the ball moves correctly, AI applies to the haptic device or mouse successfully and so on.

A* Algorithm Test Framework
In our project, we use automated testing tools to test each method via a C++ Testing Framework called Catch, which has the following advantages:
1) Ease of use: You need only go to the website and download catch.hpp, and then just include it in your project with your test files.
2) Independence from external libraries: As long as you can have C++ standard libraries, you can use the test framework without dependencies.
3) Test cases can be divided into sections: Each section is an independent running unit.
4) Test names can be any form of string: You do not need to be concerned whether the name is legal or not.

A* Testing

To compile test files, assuming you start at the top-level directory of the repository (/team21_project/)
```
cd BoardGameArchitecture/src
```
please type this command to compile:
```
make tests
```

To run test files, assuming you are at the directory (team21_project/BoardGameArchitecture/src)
please type this command to run:
```
../Tests/bin/testMap
```

Then you can see all tests have passed

```
===============================================================================
All tests passed (34 assertions in 2 test cases)

scyjw5@ubuntu:~/Desktop/team21_project/BoardGameArchitecture/src$ 
```

Glossary

CHAI3D - The open-source haptic framework we built our architecture on top of. More information: chai3d.org

Vanilla CHAI3D - A version of CHAI3Dpulled from the chai3d.org website that is compatible with mouse-based controls.

Customised CHAI3D - The version of CHAI3D taken from the Cobot Maker Space that has been customised to function properly with the Haption Virtuose

Haptic device - refers to Virtuose 6D produced by Haption in France.

aStar (A*) algorithm - A route optimization algorithm technique used to compute the optimal path

Cobot Maker Space -The Cobot Maker Space is a 99m² facility that offers ultramodern robots and equipment to explore and design human-robot interaction. Located on the Jubilee campus at the University of Nottingham.