

**ICT 312 Physics Simulation**

**Milestone Two**

**Design Document**

**By Arran Ford, Hamish Carrier and Timothy Veletta**

Contents

[Program Design Philosophy 3](#_Toc371061360)

[Model Information 3](#_Toc371061361)

[Program information 4](#_Toc371061362)

[Plans and Measurements 4](#_Toc371061363)

[Software Quality 5](#_Toc371061364)

[Issues 5](#_Toc371061365)

[Testing Details 5](#_Toc371061366)

[Realism 5](#_Toc371061367)

[NPC Design 5](#_Toc371061368)

[Physics 5](#_Toc371061369)

[AI Used 5](#_Toc371061370)

[Overall Appeal 5](#_Toc371061371)

[Special Features 6](#_Toc371061372)

[References 6](#_Toc371061373)

# Program Design Philosophy

Our approach to the design and subsequent creation of this project was a 3 step procedure. We initially started out by getting together and thoroughly planningout how we were going to approach the project. It was at this stage that we decided on what we wanted to get out of the project and what would be the best way to do it, as we were all also doing ICT313 (and in two different groups) we decided that the ability to be able to work independently was one of the most important goals. This way depending on the varying group schedules we could each work on the project at times that best fit into our own schedules. To facilitate the ability to work independently we split the first major milestone of the project up into major three categories, graphics development, collision detection and resolution, and asset creation. This kept the amount of points of overlap to a minimum, which minimised the amount of conflict we encountered when merging / sharing each of the three categories.

Once we finished the planning stage we came together as a group to measure out and record the physical design specifications of the area we planned on simulating. In addition to physical measurements we also photographed all the models we planned on creating, and any textures that we would need to replicate / make use of.

After all the physical specifications were recorded we each moved onto the category we had decided on earlier. An advantage of splitting up the tasks into three categories with as few dependencies on the other tasks as possible, was that we were able to enter the development stage and work concurrently. Without ever having to halt development while waiting on another party to finish a part of their section.

As a result of our design philosophy we were able to develop much quicker than we had done in previous group projects, and the number of problems we encountered was also much lower than previously experienced.

## Model Information

All of the assets were created in two programs, the 3D models in 3DS Max, and the textures in Photoshop. This was done due to the ease of use of the programs, and due to the ability of Ogre3D to easily render pre-made 3D files. The models were exported from 3DS Max to Ogre3D scene, mesh and material files using the OgreMax plugin for 3DS Max.

The models are loaded into our program using the ‘SceneLoader’ class which loads in the .scene format produced by OgreMax. The scene file is essentially an XML file that defines each of the models in the scene, their position, orientation and scale and also the material associated with that model.

The scene loader then creates new ‘GenericObjects’ from each of the models loaded in and adds them to the current scene.

## Program information

Our program consists of 5 main parts, classes associated with running the game, classes associated with rendering, the collision system, the physics system and the AI system.

The ‘Game’ class brings all the other sections together and contains the program loop for updating the program.

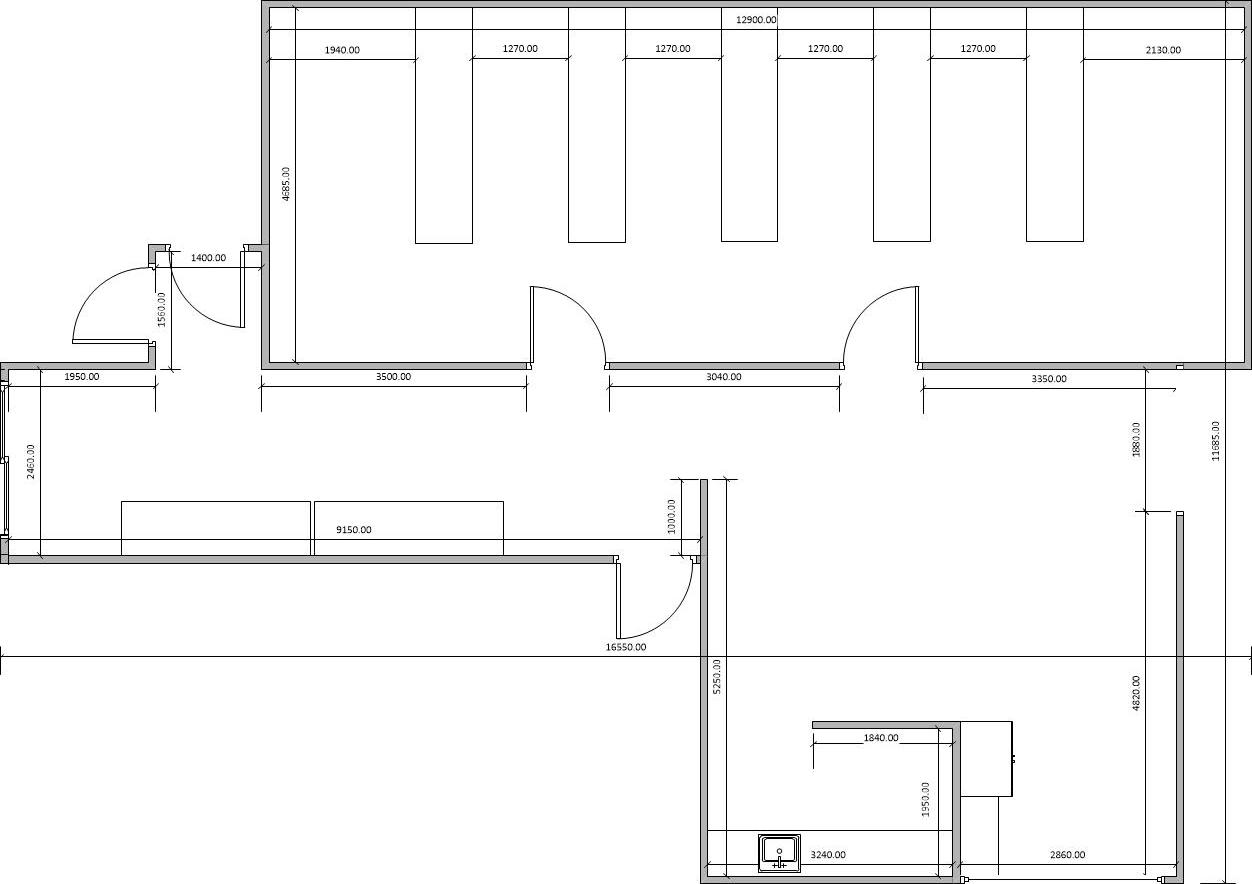
For rendering we use the Graphics API ‘Ogre3D’ with all the functionality we require defined in functions in the ‘OgreGraphics’ class. All of the components in the Graphics namespace can be used independent of the other parts of the system so they can be reused in latter projects.

The collision system uses the API “Bullet Physics” to detect collisions, each collision object in the world has a void pointer that points to a custom object we create that is associated with the collision object, allowing the collision system to trigger collision resolution in the objects that have collided.

The physics system uses an impulse based system for resolving collisions and uses Verlet integration for determining an objects change in position and velocity over time. The reasoning for using Verlet integration over Euler integration for determining object positioning is because Verlet offers a greater level of accuracy since it uses the average acceleration between frames to determine the objects velocity. The impulse based physics system is detailed in a latter section.

The AI system…

## Plans and Measurements



# Software Quality

## Issues

## Testing Details

# Realism

## NPC Design

## Physics

## AI Used

# Overall Appeal

Explain how the software presents when running. Describe how stable is the software.

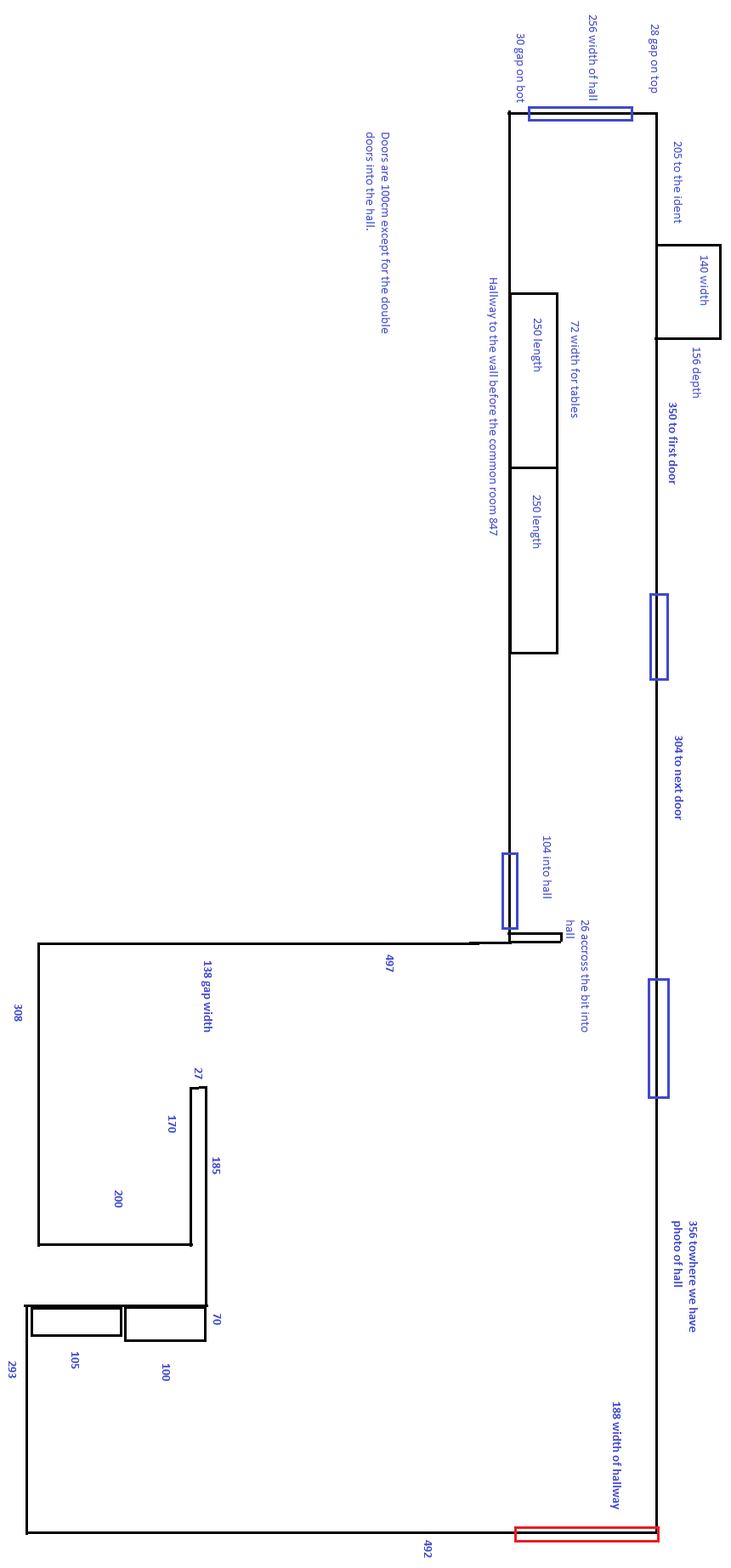
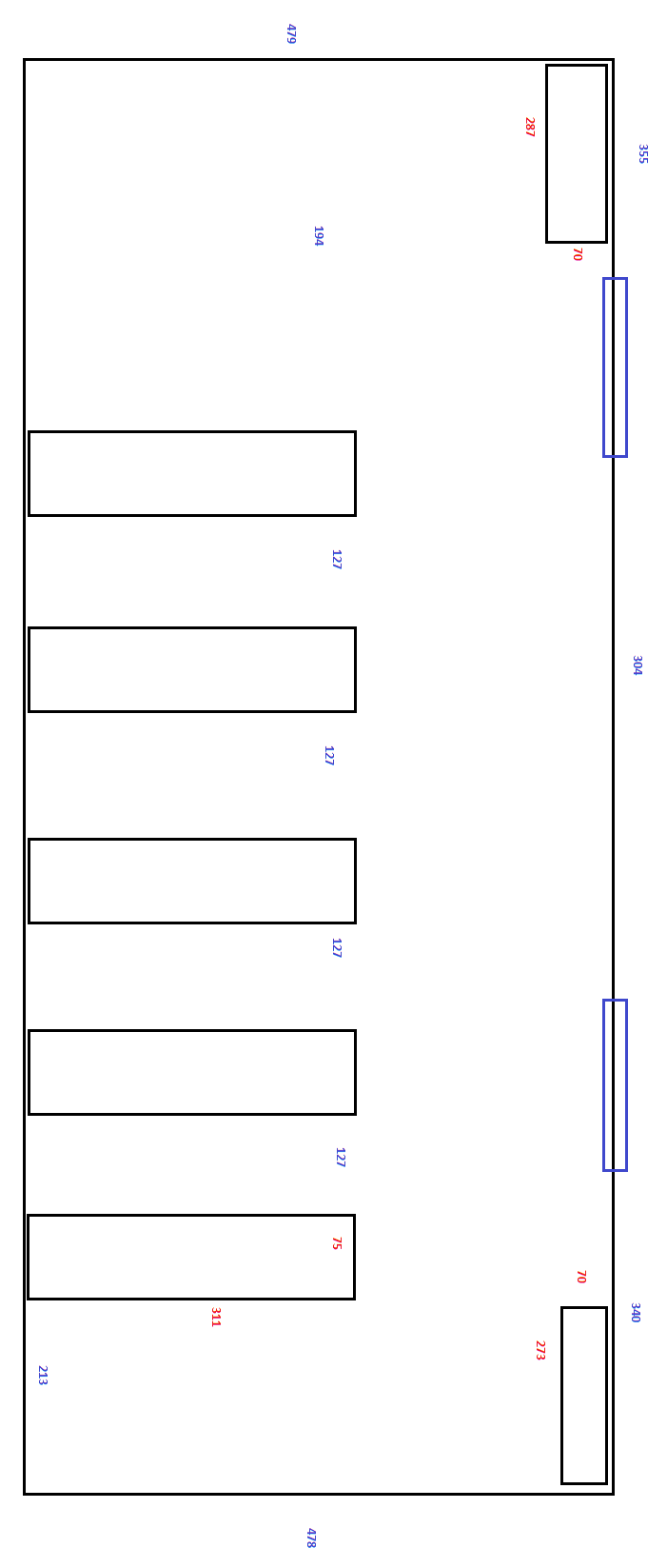
# Special Features

Special features3 – like networking, Player – NPC dialog or anything else you put in. Explain these features.

Illustrate does not mean that you just copy and paste code. An explanation is needed and the code is to backup the explanation.

# References

Contain reference to Debug Drawers here



For larger versions of images see:

<http://i.imgur.com/H0yi7Zr.png>

and

<http://i.imgur.com/I7tqwMN.png>

respectively.