KINGSTON UNIVERSITY

Stephen Davies
Computer Science Games Programming
Puzzle Game: Tessera

Individual Project

CI3330-A

K0905143

03/05/13

Supervisor – Vasileios Argyriou
2nd Marker – Andreas Hoppe

Contents

1. Introduction

1.1 Overview

Puzzle games are becoming a more and more prominent genre in the current games industry, through breakthrough games such as Portal. This project will aim to create a fully working puzzle game with an up to date game engine.

1.2 Idea

The idea of the game comes from playing the 2010 puzzle game Portal and the 1997 film Cube. The game will be a cross between the film and the game, using the fluidity of play that comes with playing Portal, with the complexity and paranoia of being trapped from Cube. The reason for the game is that not many students attempt a puzzle game for their final year project. The game will be from the first person point of view and the game will start, with the player in the middle of the cube rooms. The aim of the game is for the player to escape the cubes via a doorway hidden somewhere in one of the other rooms. Some of the cubes will contain mazes and rooms to traverse. The player will also have to deal with the rooms moving. The player will be able to see the total time they have spent in the game and will try and gain a high score with the quickest time.

1.3 Scope

The scope of this project will be creating a puzzle game through a defined and up to date game engine using 3rd party physics.

1.4 Aims and Objectives

High level aims and objectives of the Project.

1. The game should be able to be played on an up to date medium.
2. The game should be able to be completed.
3. The game should be able to be played by anyone above the age of 5

1.5 Player and Personal Objectives

High level Player and Personal objectives

1. To be able to implement from a design a completed game
2. To be able to interact with the game in the best way.
3. To be able to get lost in the game
4. To be able to successfully complete the game
5. To be able to reset the game and try again.
6. To be able to register a high score.

1.6 Gameplay and Mechanics

The gameplay of the game should resemble the fluidity of playing Portal, by means of moving around the cubes through running and jumping, by the player using keyboard to move forward and backward and the mouse to move the camera.

The two main Gameplay mechanics are: Running and Jumping.

2. Literature Review

2.1 Introduction

The literature review will be the starting point for the project, taking in the current development methodologies, approaches and possible tools and critically reviewing each.

2.2 Development Methodologies

2.2.1 Waterfall Model

The waterfall model is a development methodology that completes its phases flowing downwards, thus the name waterfall (See Fig.1). This model is mainly used for manufacturing and construction industries but was used by software engineers in a time where no real development methodology existed. The structure of the waterfall method is such that each phase has to be completed before the next can be started. This means that going back to another phase because a problem has occurred is often very costly. (Boehm, November 1995)

2.2.2 Spiral Model

The Spiral Model was created by Barry Boehm in 1986, to try and bridge the gap between an iterative process that allows development phases to be cycled and the controlled aspects of the waterfall model. The Spiral Model (See Fig.2) uses iterations of a set period of time, usually between 6 months and 2 years and allows for continuous refinement of the key phases seen in the Waterfall Model. Each cycle would be a release of a part of or an extension of the software. The general idea behind the approach is to constantly have software available to test by the user. The Spiral has 4 sections that each phase should go through before a release, which are: Determine Objectives, Identify and Resolve Risks, Development and Test and Plan the next iteration. (Boehm, November 1995)

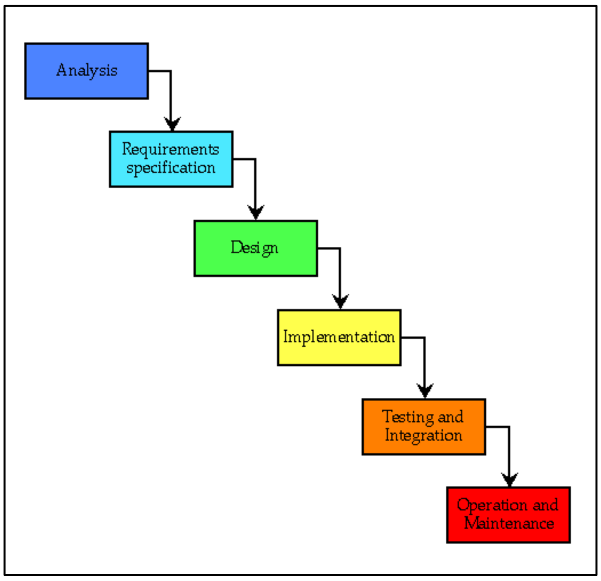


Fig.1 (McCormack & Conway, n.d.)

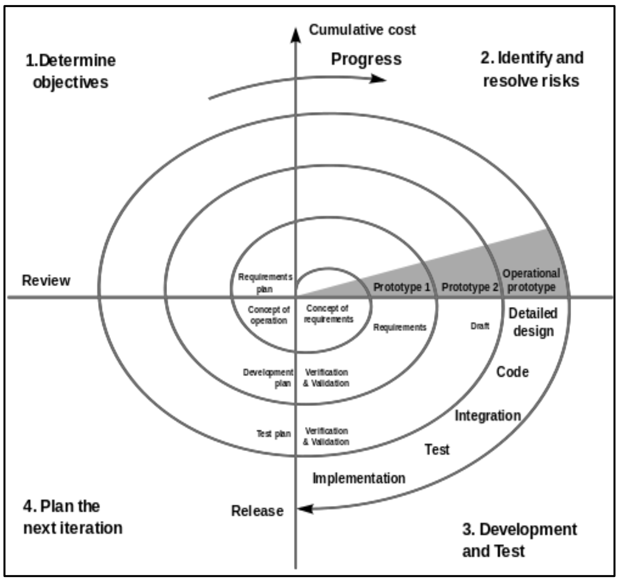


Fig.2 (Boehm, 1986)

2.2.3 Agile Method

The Agile Method is an iterative method that has in recent times become more active in software engineering (See Fig.3). The Method is based upon iterations and increments much like the Spiral Model above, except that this time the requirements of the system would evolve through collaboration of different teams and the promotion of adaptive plans, development and delivery. The most compelling part of Agile is the speed at which the method can adapt to sudden change, however it has been seen in the past that the Agile Method does not work well with project based teams. There are various versions of Agile that have been created for focus on different parts of the development lifecycle, including: Agile Unified Process, Extreme Programming, Kanban Development and SCRUM. The Agile Method includes "Stand-up" meetings with the members of the team to continually see where each team member is on their tasks to do, what they did yesterday, what they plan to do today and whether or not they face an problems.

2.2.4 Agile – SCRUM

The SCRUM method is a development framework of the Agile Method. The method is very similar to that of Agile except is focus is on speed, flexibility and a "holistic" or "rugby" approach, meaning that the method is performed by one functional team of different sections that continually pass the each other the work in order to reach the final goal (See Fig.4). One benefit of this method is that it takes in to account if the customer changes there mind. The SCRUM Method uses the iterations from the agile method along with sprints. Sprints are the normal unit of development in SCRUM and the act in the same way that iterations do in Agile except sprints happen inside an iteration and are often no longer than 2 weeks in length. The tasks that go into a sprint are from the product backlog, which is an ordered list of the product requirements.

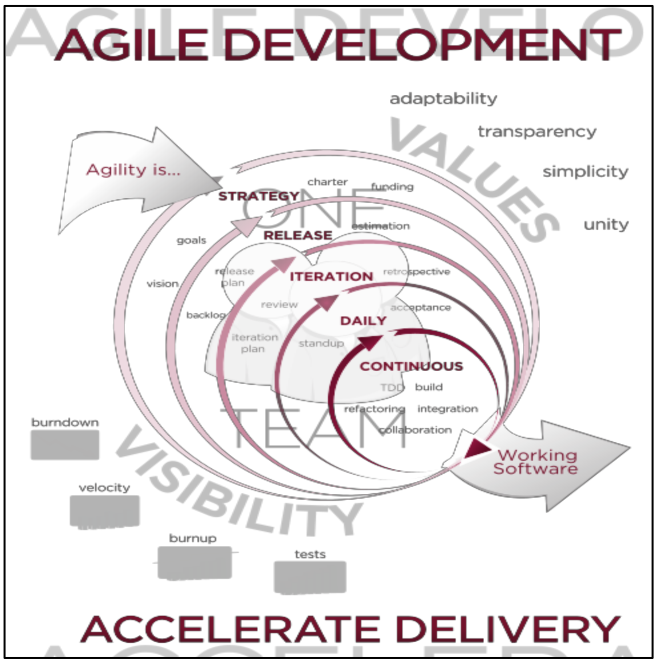


Fig.4 (Schwaber & Beedle, n.d.)

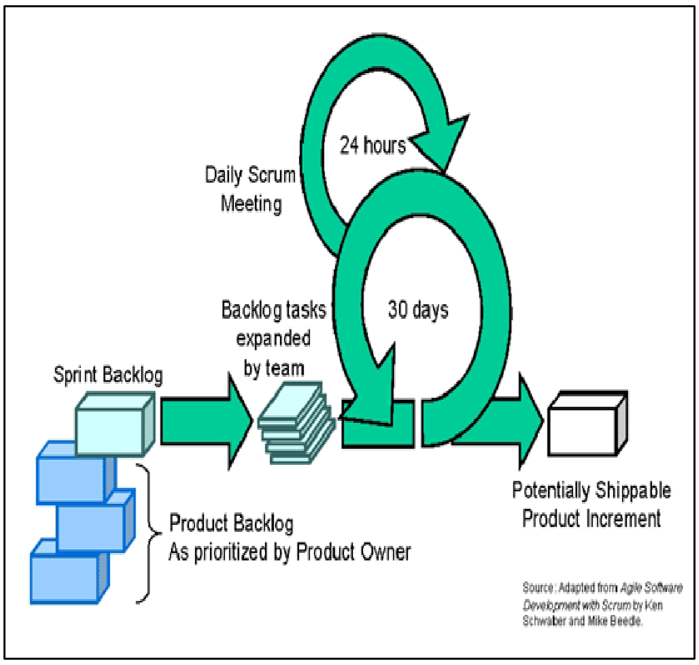

24 hours
Daily Scrum Meeting
Backlog tasks expanded by team
30 days
Sprint Backlog
Potentially Shippable Product Increment
Product Backlog As prioritized by Product Owner
Source: Adapted from Agile Software Development with Scrum by Ken Schwaber and Mike Beedle.

Fig.3 (benson, 2010)

2.2.5 Critical Review

The 4 possible methods detailed above all have their own positives and negatives when applied to making a game. In the industry it the most popular methods used are the spiral method and the Agile-SCRUM method. REF The Spiral Method is popular because when in the first stages of development, a game may not have a solid design specification, so taking advantage of the spiral methods recursive test and develop phases the design and implementation can have constant releases based upon design changes at the moment of change. (Boehm, November 1995). The Agile- SCRUM method also has this iteration cycle but changes would have to be put on a backlog. SCRUM is a more organized method, however this and the SCRUM agile method are slowly being taken up by games companies because both in one way or another allow for changes to be made in the design and implementation phases at nearly any point. This is in a stark contrast to the amount of companies that use the waterfall method. (Zynga On Game Development Tools, March 2010). The waterfall method does not fit the way that games are created. Very rarely do games stay the same from design to final product and because of the way the waterfall method works, it would mean that a company that did use the waterfall method would spend time and resources iterating whole phases for 1 design change.

2.3 Game Design Approaches

2.3.1 Technical Design

Technical deign involves the outlaying of the technical parts of the project. There are various ways to do this including using the Unified Modelling Language. UML as it is better known is the tool of which a software engineer would use to help design a system. UML has many different ways of specifying different parts of the system, usually each corresponding to each other.

2.3.2 Use Cases

"Use cases are descriptions of the functionality of the system from the users' perspective" (Bennett, et al., 2006). What usually happens is an engineer is given a long user story of how a particular problem is solved at the current moment in time with the old system they are trying to replace. From this document the engineer would then pull out the main high level requirements that the new system has to meet. The requirements can then be placed in Use Case Diagram. The Use Case Diagram is then used to accurately display and documents the scope of the system and to also make sure a developer understands what is required (Bennett, et al., 2006). Fig.6 shows a use case diagram that has been created to show the requirements of an ATM.

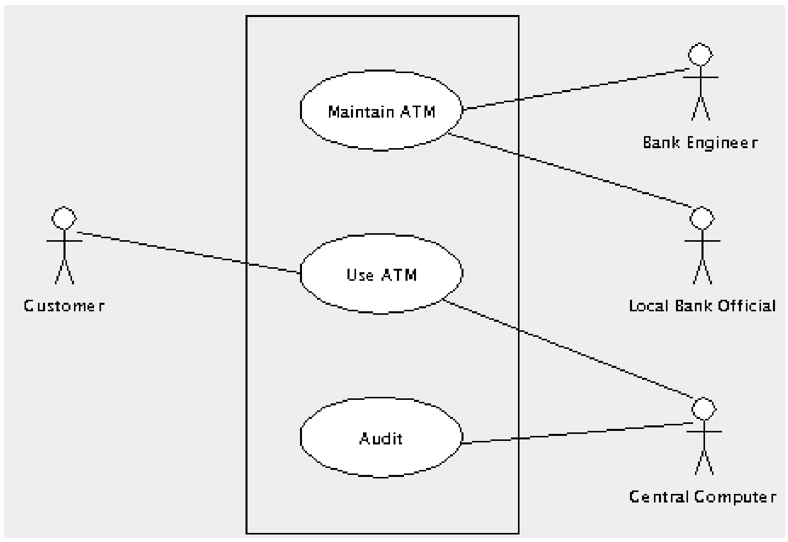


Fig.6 (CollabNet, 2009)

2.3.3 Class Diagrams

Class Diagrams are the next step forward from the Use Case Diagram. Class diagrams are often derived from the Use Case Diagram by taking those requirements and creating object orientated classes for the proposed new system. The class diagram is then used to implement the new system. Each class will have the name of the class, the attributes associated with the class and the possible methods that are going to be used. For instance a class called ATM may hold attributes of ATM_ID, max_amount and location, whereas the method may be like deposit_money and retrieve_money. Each class would then be connected to another by association. An association is a "logical connection between two or more objects" (Bennett, et al., 2006). The Class Diagram Below is an example of a Flight class association with the Plane class (Fig.7).

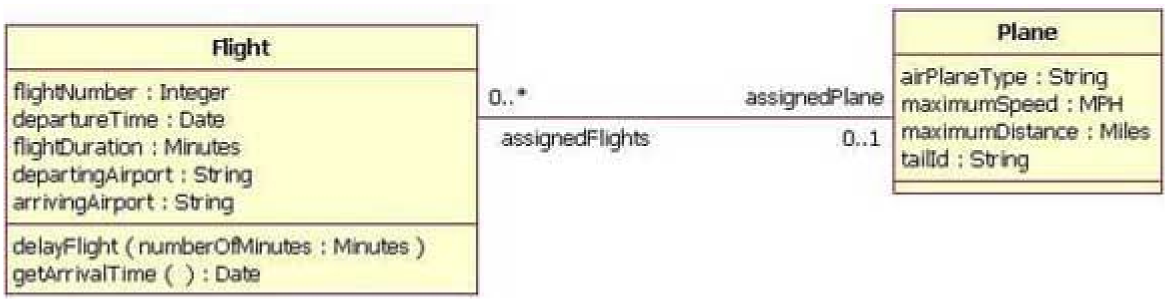


Fig.7 (Bell, 15 Sept 2004)

2.3.4 Activity Diagrams/Flow Diagrams

"Activity Diagrams can be used to model different aspects of a system" (Bennett, et al., 2006)At a high level view, like a Use Case Diagram, the activity diagram can be used to model activities in an already existing system. At a low level an Activity Diagram can be used to model possible activities and how they will be carried out in the new system. Because an Activity Diagram can be used in both the high and low levels it is common that a design document will contain both (Bennett, et al., 2006). Below is an example of an activity diagram that passes through states (Fig.8).

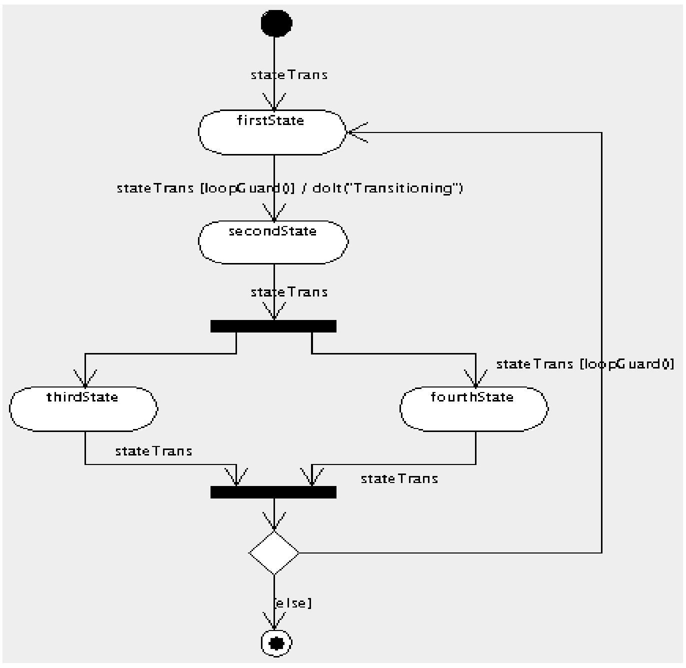


Fig.8 (CollabNet, 2009)

2.3.5 Sequence/Interaction Overview Diagrams

“An interaction overview diagram focuses on the overview of flow of control in an interaction where the nodes in the diagram are interactions of interaction occurrences”. (Bennett, et al., 2006) In the Interaction diagram the same syntax as an activity diagram is used in order to model a broken down interaction in to its key elements. A sequence diagram is a modelled part of the interaction and the interaction overview is the result of the combined sequence diagrams making up a full interaction. This is often used to model complex interactions because they have to be broken down. A games orientated example of this is the interaction between the user (keyboard, Mouse) and the result of those actions at a given time. Below is an example of a sequence diagram of a balance lookup of an ATM(Fig.9).

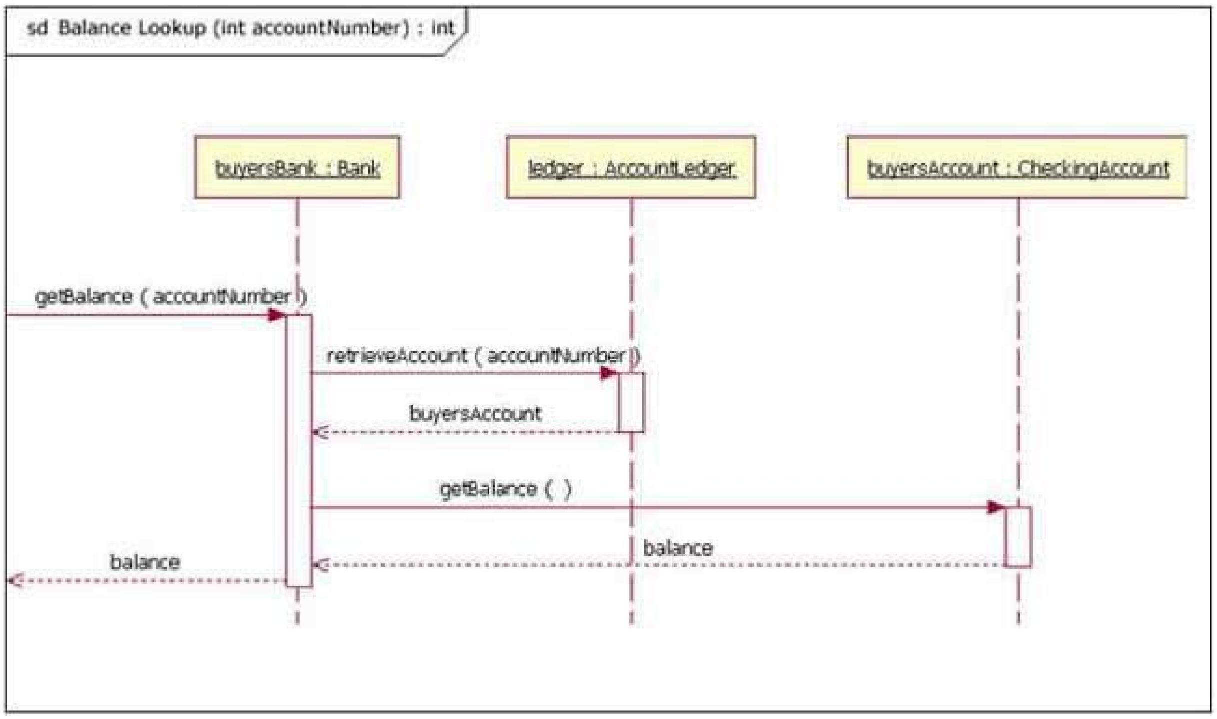

sd Balance Lookup (int accountNumber) : int
buyersBank : Bank
ledger : AccountLedger
buyersAccount : CheckingAccount
getBalance (accountNumber)
retrieveAccount (accountNumber)
buyersAccount
getBalance ( )
balance
balance

Fig.9 (Bell, 16 Feb 2004)

2.3.6 Conceptual Design

Conceptual Design is the design of features of a program through mediums such as concept art. Concept art is used to display possible designs or a particular part of a piece of software before it is implemented. Often there are many styles of design created for one design and the best one is then chosen. Concept art helps the engineer to create the software to the expectations of the design by using them as guidelines. Other Conceptual Designs could be the drawings of movement or particular styles of actions.

2.3.7 Critical Review

Both the technical and the conceptual design are needed when creating a game. A technical design is always needed when professionally creating software; however a conceptual design may be rarely used in this case. When talking about games production, concept design is always used and needed for the programmers, modellers, and texture artists. When using technical design, games production very rarely has a full unchanging design to follow and therefore has to continually update its technical designs.

2.4 Game Engines

2.4.1 XNA Game Studio

XNA Game studio is Microsoft’s framework for games development on the PC, Xbox 360 and windows phone. (Reed, December 2010)The framework is much like .NET framework but for DirectX development. The main programming language is C# and the platform has been the basis of many if not all Xbox Live Indie Games. The framework also has a large following online with a large amount of websites dedicated to developing with the studio and giving out support via code samples and answering questions. The SDK comes with support for sound (using XACT) graphics, basic collision detection and 3D content modelling pipeline. However as of 1st February 2013, “Microsoft has confirmed that it does not plan to release future versions of the XNA development toolset” (Rose, 2013).

2.4.2 PlayStation Mobile/Suite

PlayStation mobile is software development kit for Sony’s PlayStation Vita. The Open Beta was released on April 19 2012, coinciding with the release for the PS Vita. The development kit comes with a studio for building the game, much like XNA’s framework, and a support studio for creating GUI’s. The main programming language is C# but the kit does have a support for Android applications in Java.

2.4.3 Critical Review

Both of the game engines mentioned above are for use with either recent consoles or PC. XNA Game studio allows for the game to be written for PC and then with a few tweaks can be ported over to an Xbox 360. PlayStation mobile also allows the user to plug in a PS Vita and play there created games on their own hardware. However because of PlayStation Mobiles release date there are little to no sources of examples online, and the user has to go through the examples provided with comments in Japanese. For a first time developer on the platform this is not a good starting point, it is also a stark contrast to the amount of help found online and in literature for XNA development.

2.5 Physics Engines

2.5.1 JigLibx

The JigLibx Physics engine is an open source physics engine built in C# for use with Microsoft’s XNA framework (Linneweber, May 2011).The engine has been built by Thorben Linneweber, a physics student from Germany, who first started to port JigLibx from JIgLIb, a C++ Physics Engine. The engine comes packed with ready to use collision systems and rigid body physics. For each object that needs to be simulated in the game, one or more bodies are defined and attached. Each of which also has one or more primitive shapes or collision skins. For instance a model that has independent moving parts will have at least two collision skins, one for each part. The primitive skins are the normal cubes, spheres and capsules, but the engine also has a mesh skin that is made from the list of indices and vertices of the model. The get the proper physics to work, the user will need to have an instance of a ‘Physics System’ and ‘Collision System’. With both of these implemented the game tells the engine to look out for collision bodies and work out how each one will interact with each other.

"Having a collision system and a rigid body physics engine makes JigLibx one of the favoured free open-source physics engines designed for use with XNA. " (Linneweber, 2011)

2.5.2 FarSeer Physics

FarSeer Physics works in the same way that JigLibx physics does above, with the exception that the engine works with both XNA Game Studio and PlayStation Mobile and is based upon Box2D. The engine works by using a World, a Body, a Shape and a Fixture. The World, the body and the shape can all be related to the Physics System/Collision System, The Body and the Collision Skin of JigLibx. The fixture however is used to attach the shape and body to the model's central position.

2.5.3 Critical Review

There isn't a huge amount of open source physics libraries available for use with XNA Game Studio, of which the ones that are free aren't always easy to use. JigLibx's open source content allows the user to get to grips with JigLibx easily through its wikidot page, which features easy to do tutorials in order for the user to implement the physics in their own game. The FarSeer engine is also only 2D.

2.6 Development Tools

2.6.1 Microsoft Visual Studio 2010

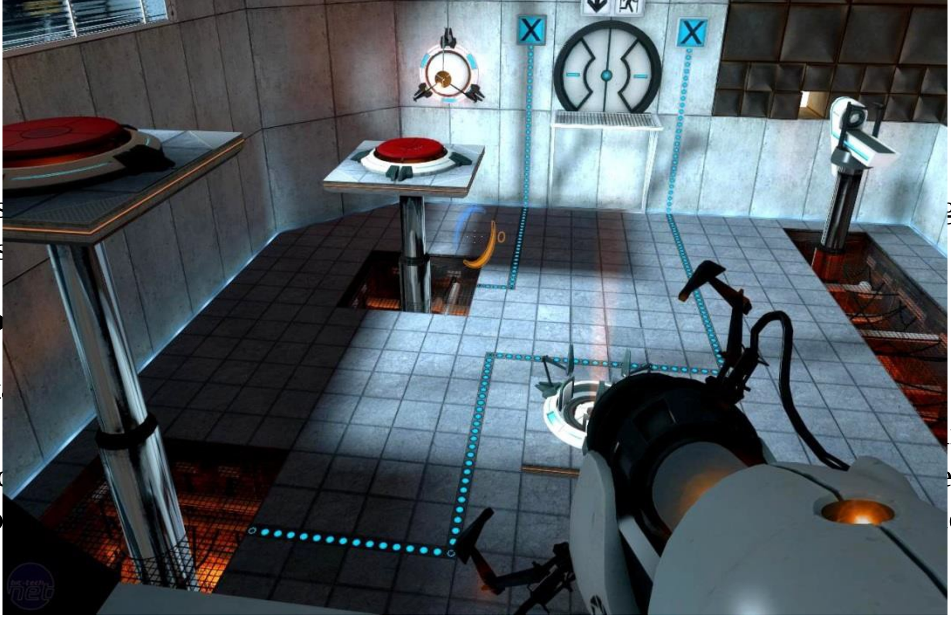
Microsoft Visual Studio 2010 is an Integrated Development Environment (IDE) that is used to program applications and software programs through a code editor and there other features like intelliSense. The IDE supports various languages including C++ and C# for all recent platforms by Microsoft. The XNA framework works as an add-on to visual studio. The Games Industry mainly uses visual studio for the multiple of different platforms they program for if they use Windows. REFERENCE NEEDED. This is mainly because the majority of games are written in C++ or C# for which visual studio has really good support for both.

2.6.2 Sony PSM Studio

The PSM Studio comes free when sign up as a new PlayStation Mobile Developer. The Suite consists of an editor and intelliSense as well as a standalone GUI creator. The Studio is very similar to that of visual studio but has fewer features and was created specifically for use with Sony PlayStation programming.

2.6.3 Critical Review

There are fair few forums and general questions on the internet about which IDE to use and most of them including (ApochPiQ, 2012) say it depends on what you are programming. So based upon this and the two IDE's above, a choice would be made for the user by choosing which game engine to program in. However that's not say that that decision wouldn't be affected by this one. Visual Studio for instance has been around for 18 years (at time of writing), and has been used by not only game programmers but normal software engineers as well (Blow, February 2004). With this backing, visual studio is the first port of call for



developers g
distinctly s

2.7 Previo

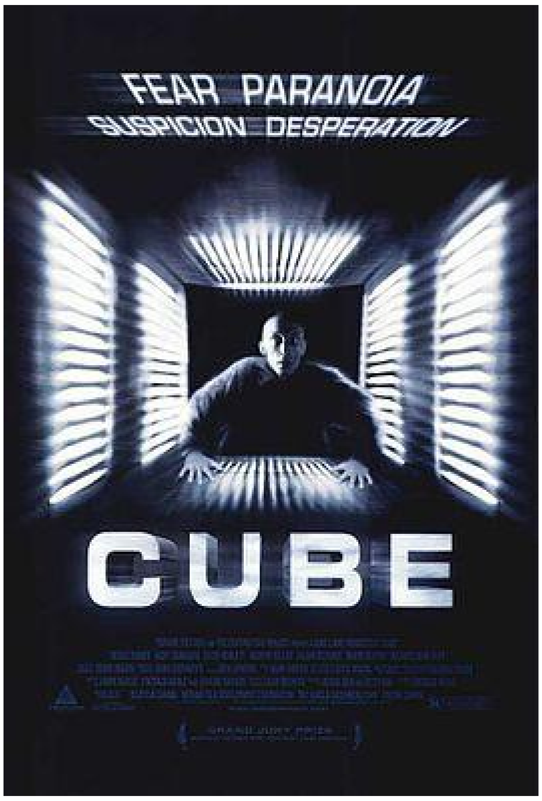
2.7.1 Port

Portal is a nd an
unknown c s both
'in' and 'o omplete
the game
(See Fig.5).

Fig.5 (Watters, April 2008)

The game achieved critical acclaim when it was released and won a number of awards including: Game of the Year – Game Developers Choice Awards and Most Memorable Villain – GamePro Editors Choice Awards. The game follows the player as he/she completes each puzzle room using physics of inertia and momentum. The storyline takes a turn for the worse in the middle when the controller of the program attempts to kill you. The game does a good job of making the player think in not only a puzzle sense but also taking into account the laws of physics (Watters, April 2008).

2.7.2 Cube 1997 Film



The film Cube starts with a group of strangers being locked inside a cube. The plot thickens when they find out that in fact there are many cube rooms and that they are all moving at one time or another. On top of this predicament is the threat of trap rooms that try to kill the strangers so they don't get out. No back story is given to why they are trapped inside or even who the strangers are. Each room they are in is a different colour to the last and each hatch has different numbers relating to the cubes location inside the larger maze of cubes. In order to get out they have to work out the location of the cube that leaves the larger mass of cubes.

2.7.3 Critical Review

The game Portal introduced the games industry to a puzzle game that can be played from the first person view. The game has fluid mechanics and takes advantage of real world physics. A good example of this is when the player has to jump a gap by using the inertia gained from falling through one portal and exiting through another higher up. The player then has enough power to fly over a gap. Another feature of Portal is the experience of the player. The player will often go into rooms and look around and on first look find nothing he or she can do. It is only on looking further and thinking about how the player can cross the problem with the tools at hand can the player realise how to progress.

The same kind of mechanics can be applied to the film Cube. The main characters are stuck in a room, and they have to find their way out through the tools they have. The tools in this case being there brains. Each room would have a code which gave a clue to where they were in relation to the larger picture and also gave a clue to which of the rooms where trapped. In the end it was the characters math skills that got the character out.

3. Methodology and Analysis

3.1 Introduction

The methodology and analysis section will be used to provide the explicit requirements of the game, using applicable requirements analysis methods to analyse the previous higher level game objectives and produce a list of lower level objectives.

3.2 MoSCow Analysis

MoSCow Analysis is an analysis technique that defines the importance of each requirement. The term is an anagram of Must, Should, Could, Would. MoSCow analysis will be used to generate the low level requirements of the game.

3.3 Detailed Game Proposal

3.4 Explicit/Low level Requirements of the Game

The lower level requirements of the game are the requirements that are development level and are derived from the original higher level objectives defined in the introduction.

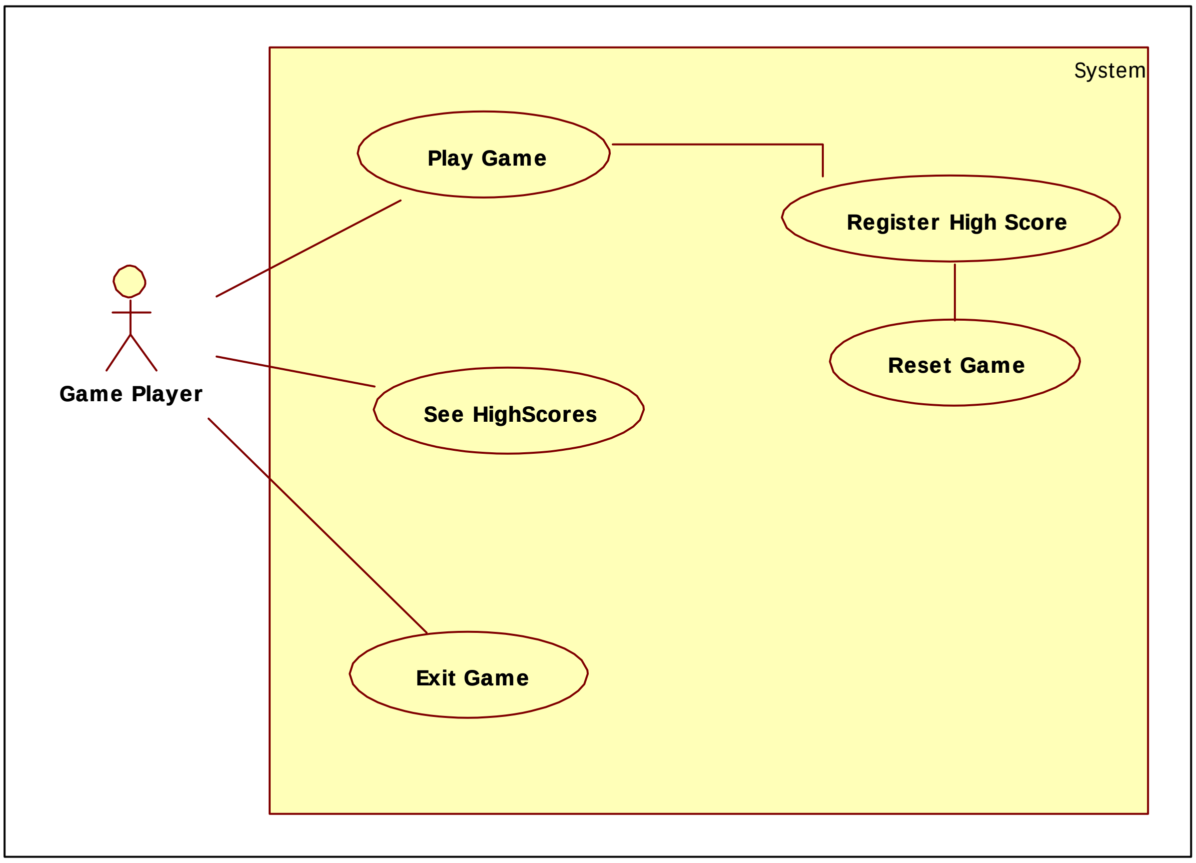
High Level Objectives

- The game should be able to be played on an up to date medium.
- The game should be able to be completed.
- The game should be able to be played by anyone above the age of 5
- To be able to implement from a design a completed game
- To be able to interact with the game in the best way.
- To be able to get lost in the game
- To be able to successfully complete the game
- To be able to reset the game and try again.
- To be able to register a high score.

Use Cases

- Play the Game
- Exit the Game
- Reset the Game
- See the High scores
- Register a High Score

Use Case Diagram



Low Level Objectives

1. The game should be developed for an up to date medium
2. The game should have a specific start menu
3. The player should be able to navigate the start menu
4. The Buttons of the start menu should be readable
5. The game should be able to exit from the start menu
6. The game should be able to start from the start menu
7. The game should have appropriate sound
8. The game should be able to be ended.
9. The game should be able to be played with the keyboard and mouse
10. The player should be able to jump
11. The player should be able to move
12. The player should be able to move from room to room
13. The player should stop when a room is moving
14. The camera should change when the room that the player is in is moving
15. The player should be able to see the time
16. The players time should be printed at the end
17. The player should be able to reset the game at the end
18. The player should be able to collide with the walls of the room
19. The player should be able to experience realistic gravity.
20. The player should be able to record a high score

MoSCoW Analysis

- MUST have requirements – 8, 9, 10,11,12,14,15, 18
  These requirements must be satisfied in order for the game to work to a specific standard. The requirements include basic movement and what is needed to play the game and succeed and to be able to program a happy path.

- SHOULD have requirements – 1,2,3, 6, 7, 13, 17, 19
  These requirements should be satisfied to produce a good game. They include GUI basics like menus and sound for the look and feel of the game. These requirements are as important as the MUST requirements but are not as time critical.

- COULD have requirements – 4, 5, 16
  The COULD requirements are nice to have if they can be done within the time limit. The few could requirements there are, are to make the GUI variables readable

- WOULD have requirements – 20
  The WOULD have requirements are the least time critical and would be implemented if there isn't anything to do once everything else has been completed.

3.5 Critical Review

The new low level requirements have formed the basis of what is needed development wise from the game. With this data and the methodologies described in the literature review, an Agile methodology is the best way forward, as the project can be properly organised but also, using Agile means that changes can be successfully implemented during iterations.

3.6 Specific Platform Requirements

The specific platform requirements are the requirements needed for the current game objectives to run. The platform requirements of the game are:

1. The platform needs to be an up to date medium
2. The platform needs to be able to handle a 3D physics engine
3. The platform needs to be able to handle 3D graphics
4. The platform needs to be able to handle sound

MoSCow Analysis

- MUST have requirements – the platform needs to be able to handle 3D graphics, the platform needs to be able to handle a 3D Physics Engine
- SHOULD have requirements – The platform needs to be an up to date medium.
- COULD have requirements – The Platform needs to be able to handle sound
- WOULD have requirements –

3.7 Critical Review

The Must have platform requirements mean that the platform for this project will be XNA Game Studio. This decision is based upon the need for 3D graphics and physics requirements, for which the PlayStation Mobile Suite cannot, at this current time, deal with.

3.8 Specific Physics Requirements

The Specific Physics requirements are the requirements needed by the Game of a physics library.

1. The physics engine needs to be able to supply gravity
2. The physics engine needs to be able to supply realistic collision detection
3. The physics engine needs to be able to create 3D physics

MoSCow Analysis

- MUST have requirements – The physics engine needs to be able to supply gravity, The physics engine needs to be able to supply realistic collision detection, The physics engine needs to be able to create 3D physics
- SHOULD have requirements
- COULD have requirements
- WOULD have requirements

3.9 Critical Review

All of the physics requirements are MUST have requirements because these are the requirements that are needed to play the game properly. The game MUST have 3D collision detection and gravity, therefore the only choice for this would the JigLibx physics engine as the FarSeer physics is only 2D.

3.10 Specific Development Requirements

The Specific Development Requirements are the requirements of the development kit that will be used to implement the design into the final product.

- The Development kit should be supplied with an up to date editor
- The Development kit should be able to compile the code
- The Development kit should be able to port the code over to an up to date medium.
- The language and syntax of the development kit should be able relatively easy to understand

MoSCoW Analysis

- MUST have requirements - The Development kit should be able to compile the code, The Development kit should be able to port the code over to an up to date medium, The language and syntax of the development kit should be able relatively easy to understand
- SHOULD have requirements - The Development kit should be supplied with an up to date editor
- COULD have requirements
- WOULD have requirements

3.11 Critical Review

The MUST have requirements of the development kit define a package that can compile the code, can port to an up to date medium and the code is relatively easy to understand. These requirements mean that Visual Studio 2010 will be the development kit that is used because of its support within the community and the industry over PlayStation mobile, which is new and therefore hasn't got the same backing.

3.12 Conclusion

In conclusion, the results of MoSCow analysis and the critical analysis of each section mean that the following choices are the best for the must have requirements of the system.

- Game Engine – XNA Game Studio
- Platform – PC
- Development Kit – Microsoft Visual Studio 2010
- Development Methodology – Agile
- Physics Engine – JigLibX
- Language - C#
- Modelling – Maya 2013

The final decision was based upon the analysis of requirements above, however it can be seen that from some of the choices would influence the decision of another requirement. For instance the choice to use XNA Game Studio as the Game Engine often goes hand in hand with the choice of Microsoft Visual Studio as the development kit as both are provided by Microsoft. Also the choice of language as being C# was made already as both the Game Engines are written in C#. All models will be created using Maya 2013.

Agile methodology will be implemented with month long iterations. Which can be referenced in Appendix 3 – Log Book Entries.

4. Design

4.1 Introduction

The design phase of development is split into two sections. The first section, Conceptual Design, will cover the basic design of the visual elements of the game and the second section, Technical Design, will cover the technically elements like object orientation of code.

4.2 Technical Design

4.2.1 Introduction

The technical Design of the game will be split up into sections, first defining how the game engine and the physics engine will interact with each other, then how rendering and graphics are produced. The end of the technical design will provide detailed diagrams explaining how object orientation is defined in the game.

4.2.2 Game Components

Game components are the classes that will be used to implement the game in XNA game studio. The list of classes and there small descriptions are below.

- Character_Controller - Is used to control the movement of the character object
- CharacterObject - Is be used to define a character object
- CollisionDetection - Is used to test collision detection between objects that cannot be attached to the physics engines collision detection.
- CubeManager - Is used to organize the technical movements of the CubeRooms
- CubeRoom - Is used to define the Cube Rooms and there individual functionality
- DebugDrawer - Is used to provide real time drawings of collision skins for debugging
- DoorManager - Is used to organise and implement the door functionality
- Doors - Is used to define the doors and their functionoailty.
- Game1 - The main class that is the starting point for the game.
- GUI - Is used to define the GUI elements of the game and there functionality
- HUD - Is used to define the heads up display and debug elements
- New_Camera - The camera for the game
- PhysicObject - Defines a Physics object
- Program - The entrance point for the game
- SkyBox - Defines the skybox for the game
- Timer - Defines the timer functionality for the game

4.2.3 Game Engine and Physics Engine

The physics engine is defined in the Game1 class and is integrated with each loop of the Update function in Game1. The project will then define parts of the game as being a physics object by attaching a body and a collision skin to the required object. The physics system will then, at every loop, find the physics objects and calculate what the behaviour of each object will be, i.e. if two are colliding then the required velocity and direction will have to change to simulate this. The following diagram shows how the body of the object interacts with the collision system of the physics system (See Fig.10).

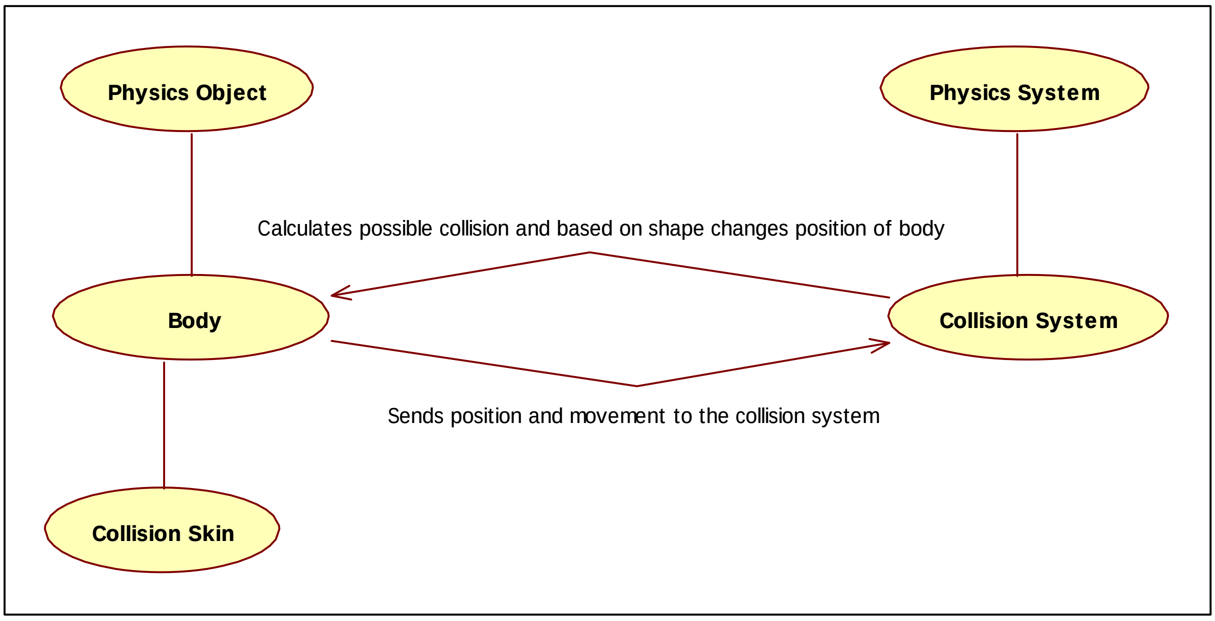

Physics Object
Physics System
Calculates possible collision and based on shape changes position of body
Body
Collision System
Sends position and movement to the collision system
Collision Skin

Fig.10 The relationship between a physics body of an object and the physics systems collision system.

4.2.4 Rendering and Graphics

The drawing of graphics happens when the Draw function is called at the end of Game1 and all drawable Game components. The following diagram best describes how and when draw calls are made (See Fig. 11). The first step is that Game1’s Update and Draw Methods are called and executed. Next any components that are attacthed to Game1 are ordered and called one at a time, so therefore component1’s update method will be called next and then component 2’s Update and Draw methods, respectively.

There are two ways to render graphics and obejects to the screen. For 2D objects like texture2D’s using a spriteBatch variable is enough which will be used in the GUI and HUD classes to display the total time and Menu System.

The other way is to render 3D graphics by using a camera. Most of the draw functions in the 3D models involve the use of the cameras Projection, View and World Matrices, which are used to properly draw the models on the screen and also to draw them occurding the to World axis.

4.2.5 Object Orientation

Object orientation is a software design approach that uses objects that contain attributes and methods and how they interact with each other. This approach being taken by this project and has already been shown with a Use Case diagram in the Methodology and Analysis section. The class diagram that follows is the full class diagram design for the Game (see Fig.12).

.

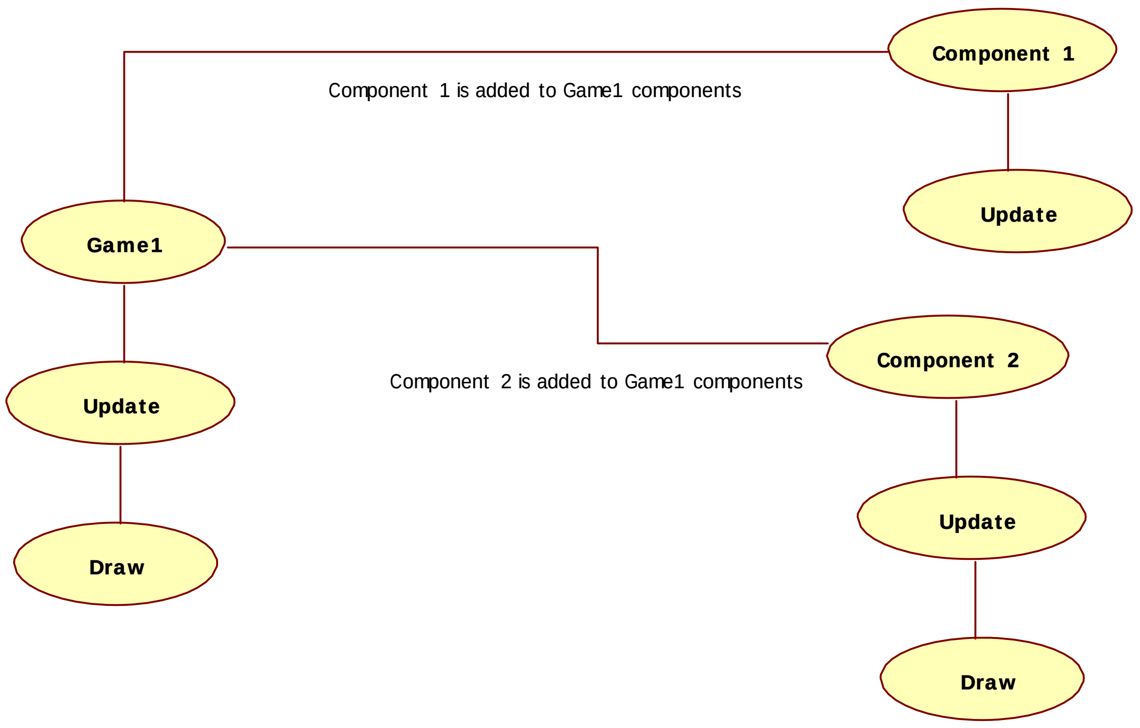

Component 1
Component 1 is added to Game1 components
Update
Game1
Component 2
Component 2 is added to Game1 components
Update
Update
Draw
Draw

Fig.11 The example attachment of game components and the Game1 Class

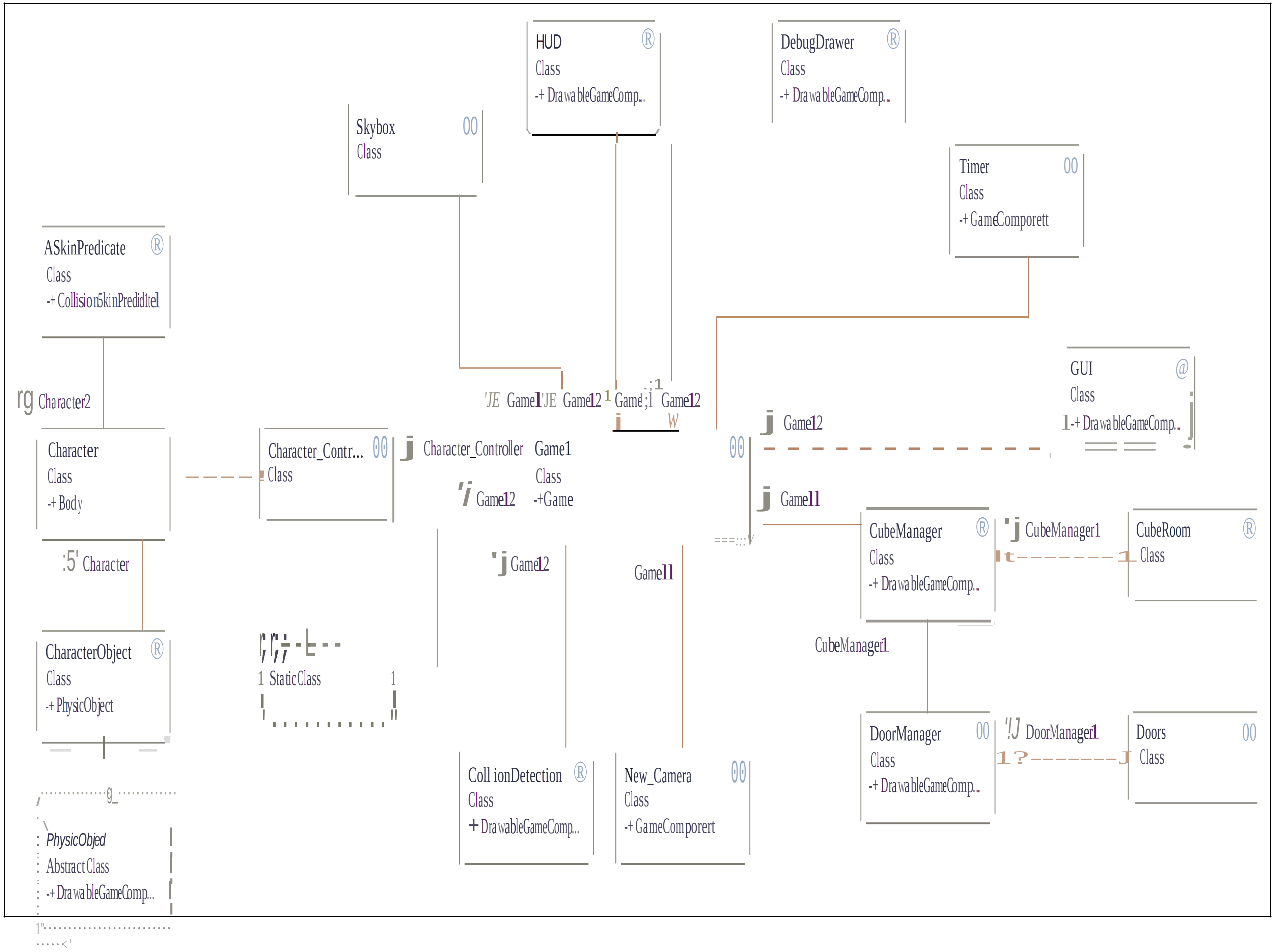
4.2.6 Overview of class diagram

The class Diagram in Fig.12 is a temporary design for the class diagram of the whole system. There are parts however that can already be permanent. These classes include, the Game1 class to the Program Class, as these classes are automatically generated on creation of a new project and there is no reason why this should be changed and the Character class and all affiliated classes. The character class is made up of character object which in its own right inherits from PhysicObject.

The timer and GUI Classes are self-explanatory but there has been an attempt to encapsulate most functionality into the right classes. Other classes like this include Skybox and HUD. New_Camera is the normal camera class that needs to be implemented in a 3D XNA game in order for 3D Assets to be drawn to the screen.

The Cube Manager and Door Manager take care of the functionality for the Cubes and Doors Respectively.

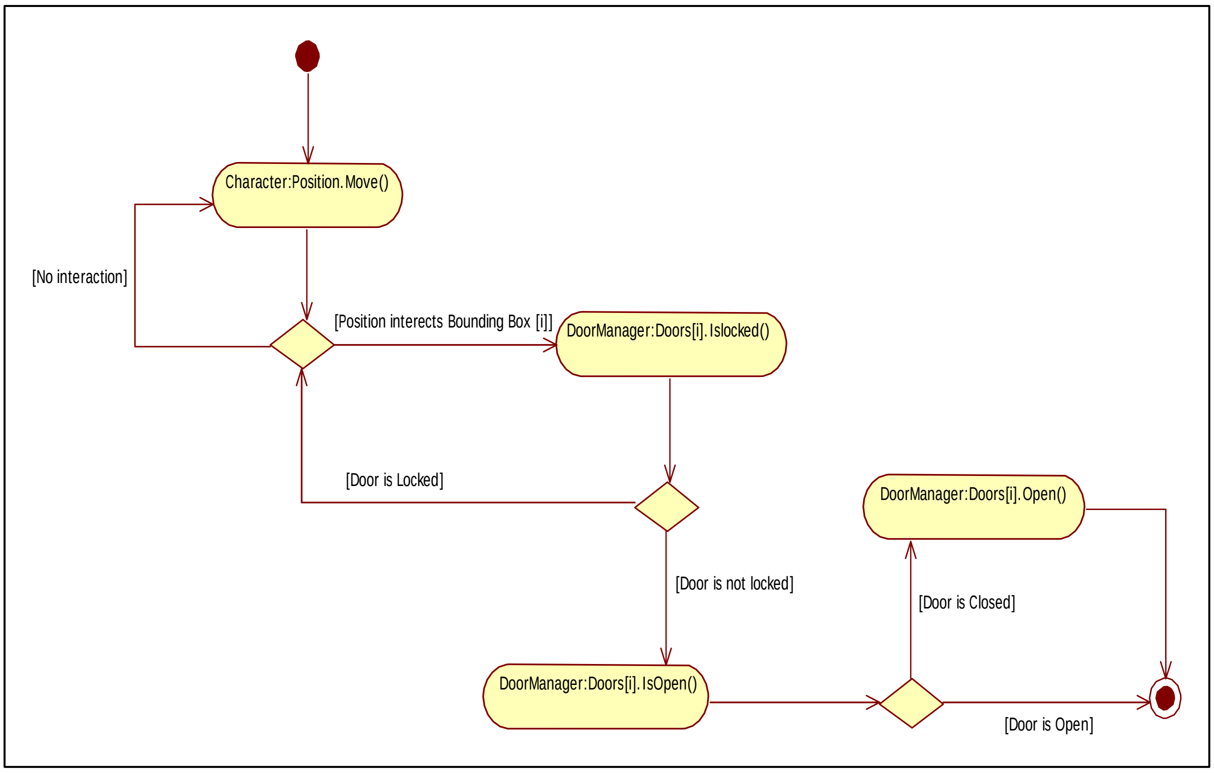
Finally the Character Controller will help bridge the gap between the Character object and the camera, as the camera will be attached to the position of the character object.


HUD
Class
+ DrawableGameComp...
DebugDrawer
Class
+ DrawableGameComp...
Skybox
Class
Timer
Class
+ GameComponent
ASkinPredicate
Class
+ CollisionSkinPredicate
Character2
GUI
Class
+ DrawableGameComp...
Character
Class
+ Body
Character_Contr...
Class
Character_Controller
Game1
Class
+ Game
Game1
Game12
Game12
Game12
Game11
Character
CubeManager
Class
+ DrawableGameComp...
CubeManager1
CubeRoom
Class
CharacterObject
Class
+ PhysicObject
StaticClass
CubeManager1
DoorManager
Class
+ DrawableGameComp...
DoorManager1
Doors
Class
CollisionDetection
Class
+ DrawableGameComp...
New_Camera
Class
+ GameComponent
PhysicObject
Abstract Class
+ DrawableGameComp...

4.2.6 Data Flow Diagrams

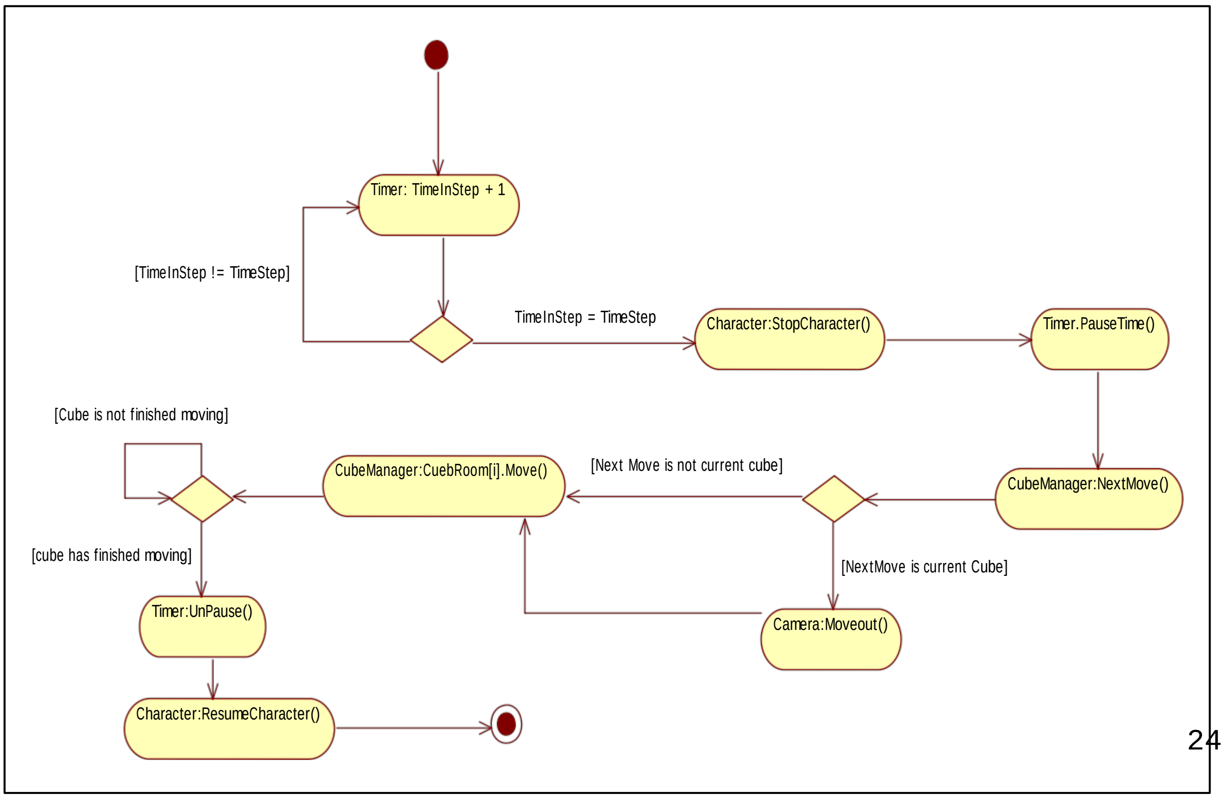
This section will look at the activity diagrams for the Doors and Cubes Moving.

Doors opening and Closing



Character:Position.Move()
[No interaction]
[Position interects Bounding Box [i]]
DoorManager:Doors[i].Islocked()
[Door is Locked]
DoorManager:Doors[i].Open()
[Door is not locked]
[Door is Closed]
DoorManager:Doors[i].IsOpen()
[Door is Open]

The diagram above shows how the implementation of doors opening and closing will be designed. First the characters position will constantly be updated in the update loop of the game. Then the position will be tested against a list of Bounding Boxes. If the door relating to the bounding box is not locked then the system will test if the door is open. The final part of the algorithm will open the door if the door is not open.



Timer: TimeInStep + 1
[TimeInStep != TimeStep]
TimeInStep = TimeStep
Character:StopCharacter()
Timer.PauseTime()
[Cube is not finished moving]
CubeManager:CuebRoom[i].Move()
[Next Move is not current cube]
CubeManager:NextMove()
[cube has finished moving]
[NextMove is current Cube]
Timer:UnPause()
Camera:Moveout()
Character:ResumeCharacter()

Cubes Moving

The diagram above shows the possible implementation of the cubes moving. Every time increment, the time in step is tested to see if a fixed time has been reached. If it has the player is stopped and so it's the time. The function for next move is then called to find the next cube that will move. If the cube to move next is the cube that the player is standing in, then the camera will pan out. The cube will then move. Once the cube has finished moving the time is un-paused and the character can resume.

4.2.7 Critical Review

The technical design is the design based upon the current design specification and objectives. If the design needs to be changed, which is likely, there will be a new iteration of the agile method. The sequence diagrams for the cubes moving and the doors opening and closing are likely to be changed as they are only the rough designs. However they have the core path defined.

The cubes moving algorithm will be more complex than the sequence diagram above shows. This is because the algorithm will take into account more information than can be placed on a sequence diagram. The design of the camera has not been created because a simple camera needs to be implemented and changed to accommodate other classes in the system.

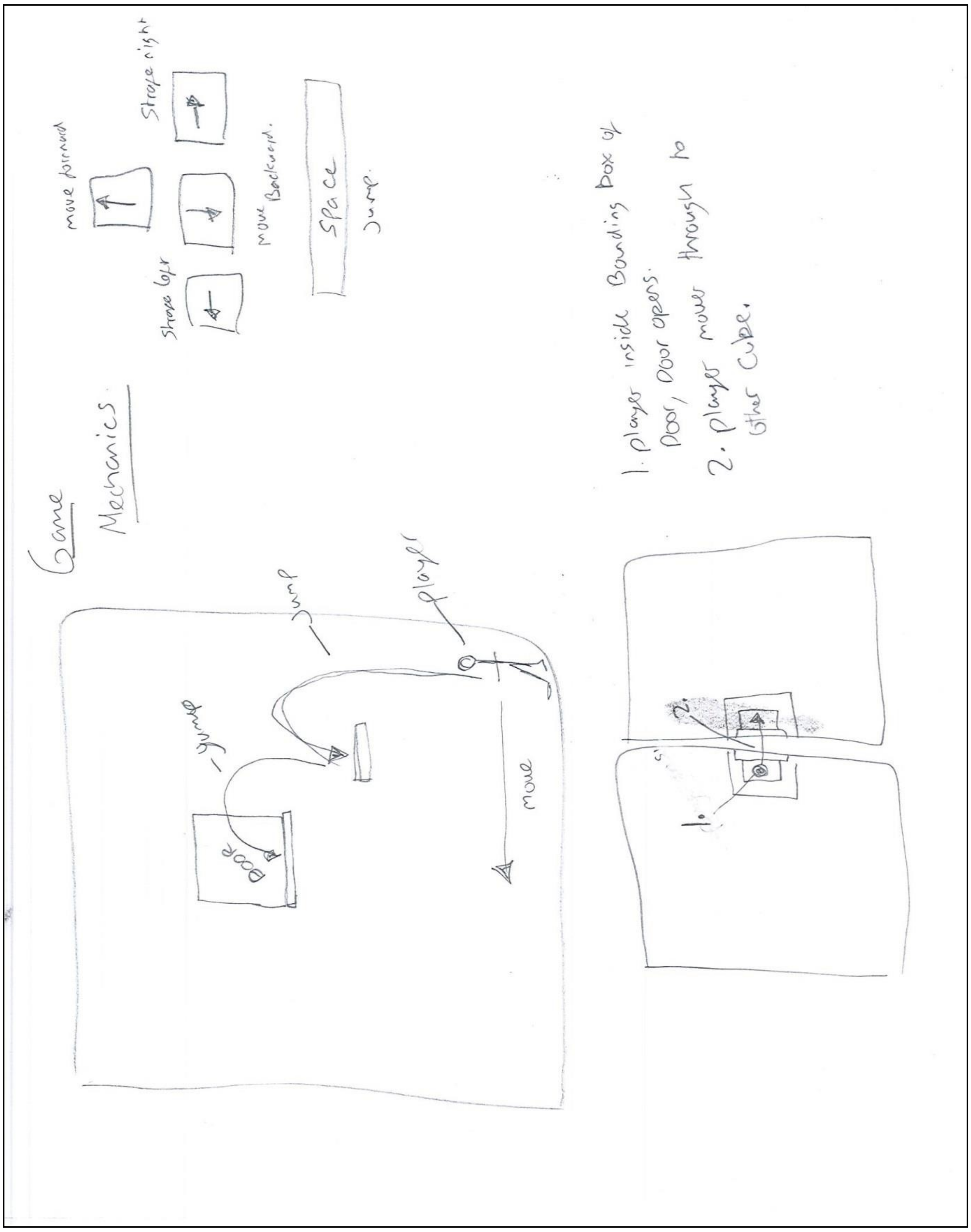
The character controller also will take into account code from other classes and the best implementation will be designed once these other classes have been implemented.

4.3 Conceptual Design

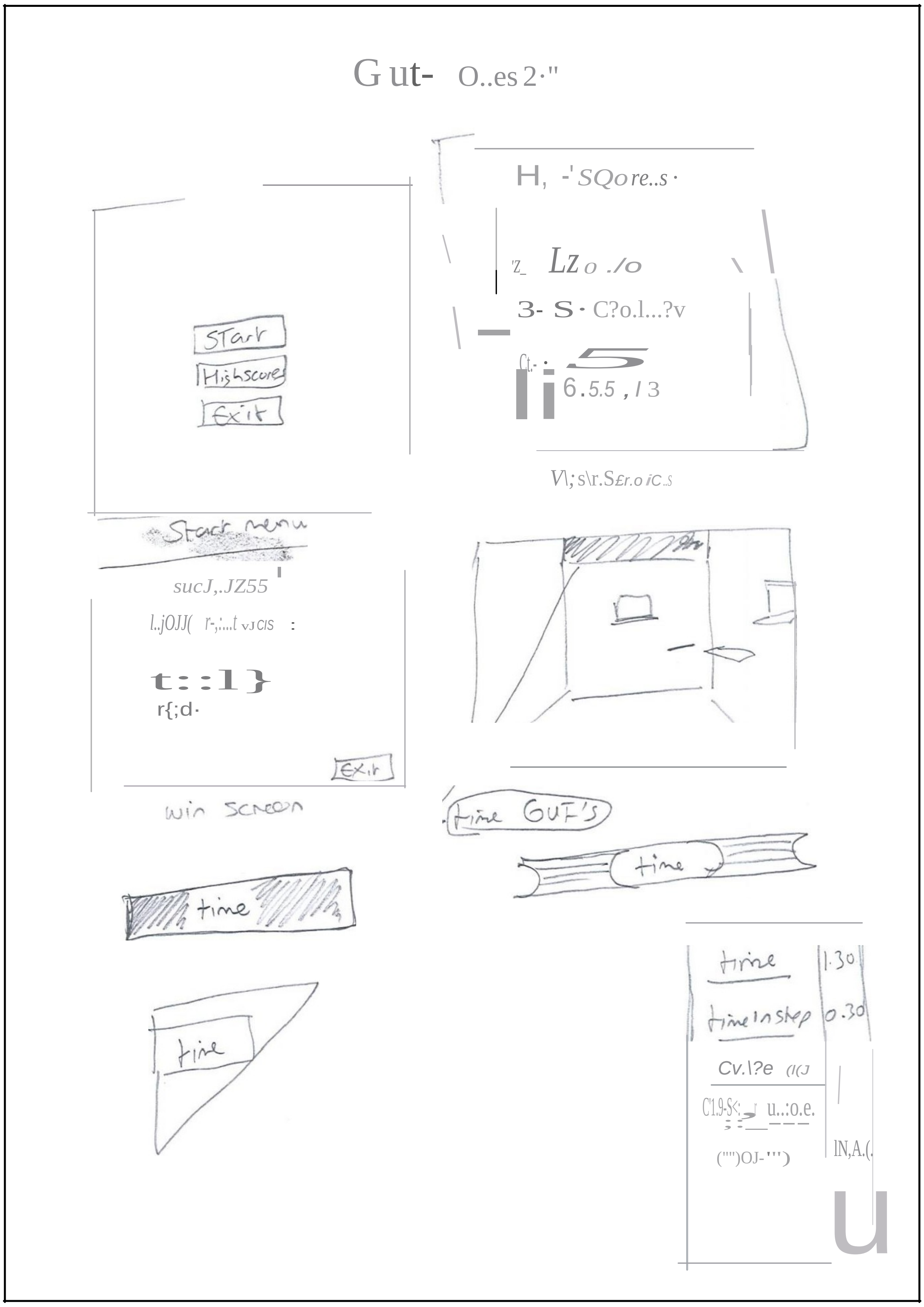
4.3.1 Introduction

Conceptual Design is the design of the visual aspects of the game through drawings and concept art.

4.3.2 Game mechanics

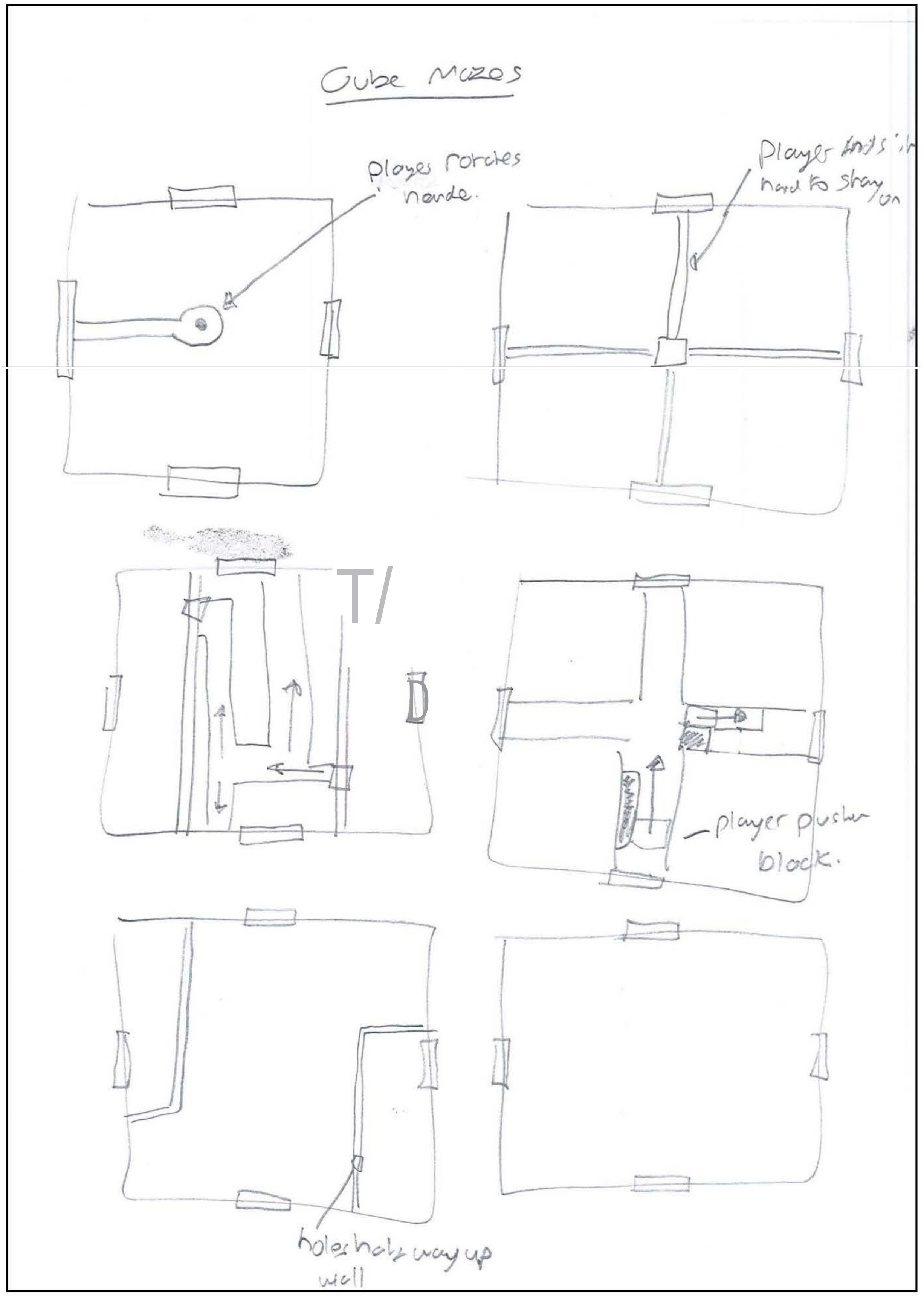


4.3.3 GUI Design



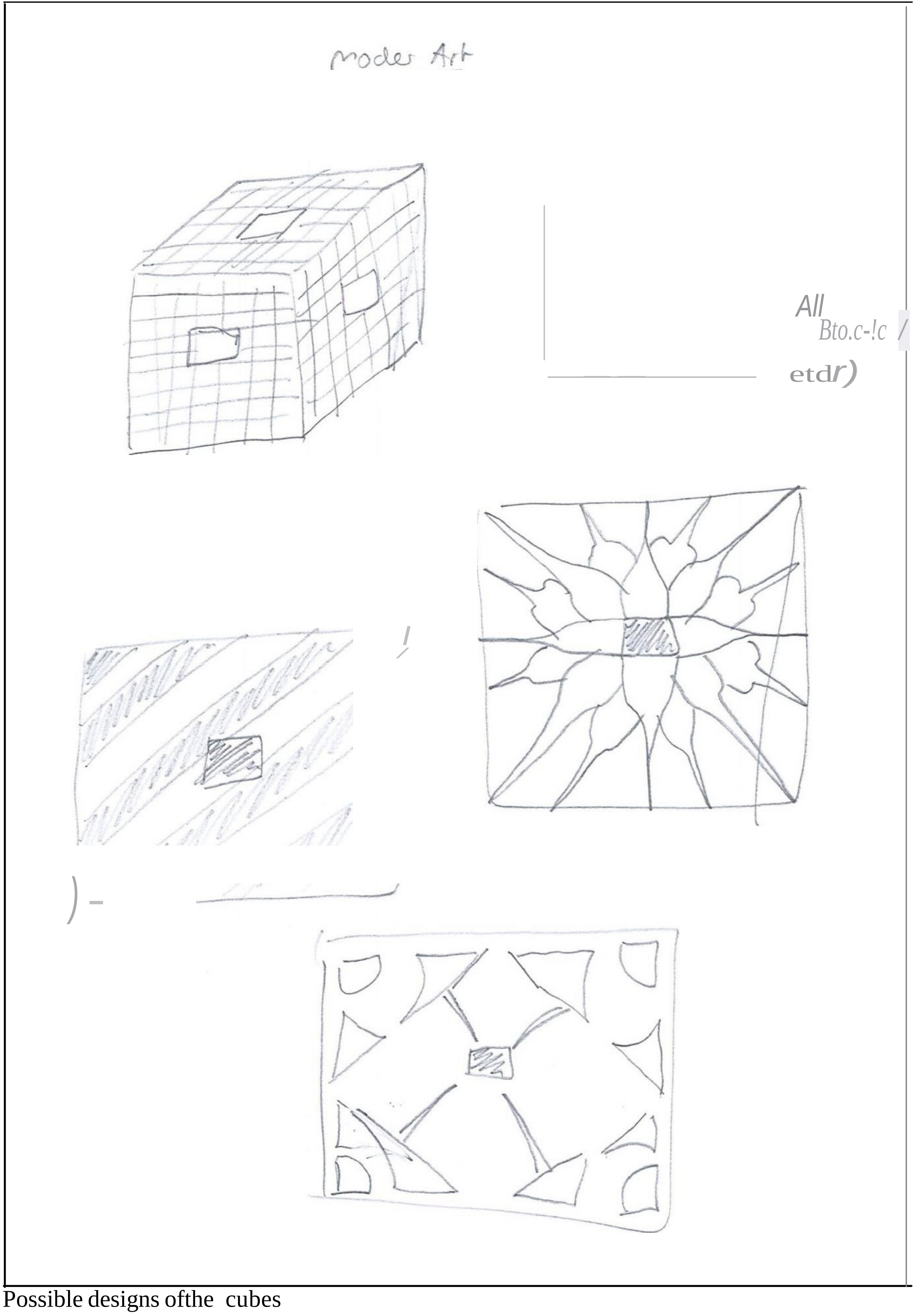
GUI Design- both menus and in-game GUI

4.3.4 Level Des1gn



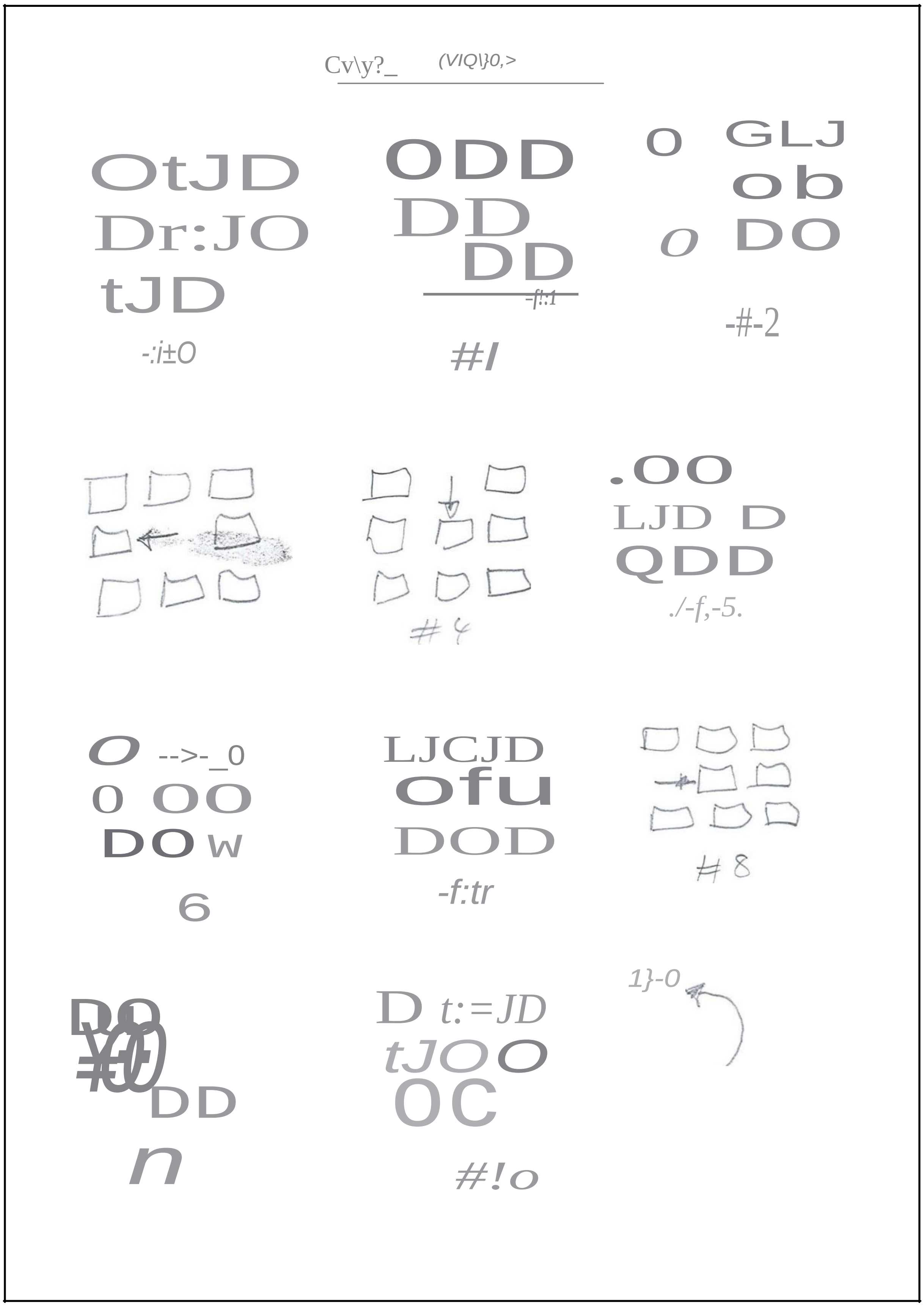
Cube Maze Designs

4.3.5 Model Concept Art



Possible designs ofthe cubes

4.3.6 Cube Movements



The Movements of the first run through of the cubes.

5. Implementation

5.1 Introduction

The implementation section will look at the actual code and how it was implemented. It will then describe in detail the structure of the project and then look at the most important features.

5.2 Technical Implementation and Structure

During the implementation stage of the project the decision was made to split the functionality of different parts into separate classes. For instance, there was a point where the functionality for the timer was encased inside the HUD class, purely because the HUD class is where the debug information was printed out. This was changed to have the time functionality inside a Timer Class and the HUD class would 'get' the Timer class through the Game1 Class.

A lot of the variables in the game have taken advantage of the getters and setters provided by the C# language. This allows the actual variables of the class to be set to private and they can only be accessed via a 'getter'. For an example of this see Fig.13.

```
        //Timer
        private Timer timer;
        public Timer Timer_Class { get { return timer; } }
```

Fig.13 example of getters in C#

There were some problems with using the physics library, which were encountered late on in the development process when attaching collision skins to doors. The physics library could not handle having so many collision skins at one time, therefore the result was that the collision detection for the doors were implemented differently to that of a room. See Collision Detection below.

The modelling of the cube rooms left the model not being able to have a primitive as its collision skin. This gave a major problem as there were only two choices available. The first choice was to separate the bodies and collision skins manually and attach primitive cubes at the points that they could fit. This would lead to having a model that had more than one collision skin, which by having more than one cube would increase quickly how little the physics system could already take. Not only this, but the time spent aligning the many different primitives would have been large. Instead the second option was used, which was using the triangular mesh primitive. The triangular mesh primitive creates a triangular mesh that is the shape of the model using the models indices and vertices.

A problem did arise with this method however, as a non-perfect cube would create a non-perfect floor. The way to solve this problem was to use Maya's Boolean values. This allowed primitives to be taken away from other primitives and leave a hole that was the exact shape. Using this method proves not only useful for having a straight floor (and thus a flat run) but also improves the poly count of the model so that it does take up a lot of memory and slow the game down.

5.3 Character Controller

The character controller is the class that holds the functionality for the character. The character controller has an instance of CharacterObject, which is the object that holds the body and collision skin of the character. In this case the primitive attached to the body is a capsule primitive, which is because a capsule has a variable height rather than a sphere primitive with a fixed height. The character class also has the definition for the character class, which sets the functionality of movement of a CharacterObject. The CharacterObject also inherits from PhysicObject which is an abstract class that sets the variables for a Physics based object, in this case this is needed for the character to interact with the environment and also to be affected by gravity. See Fig.14 for the class diagram.

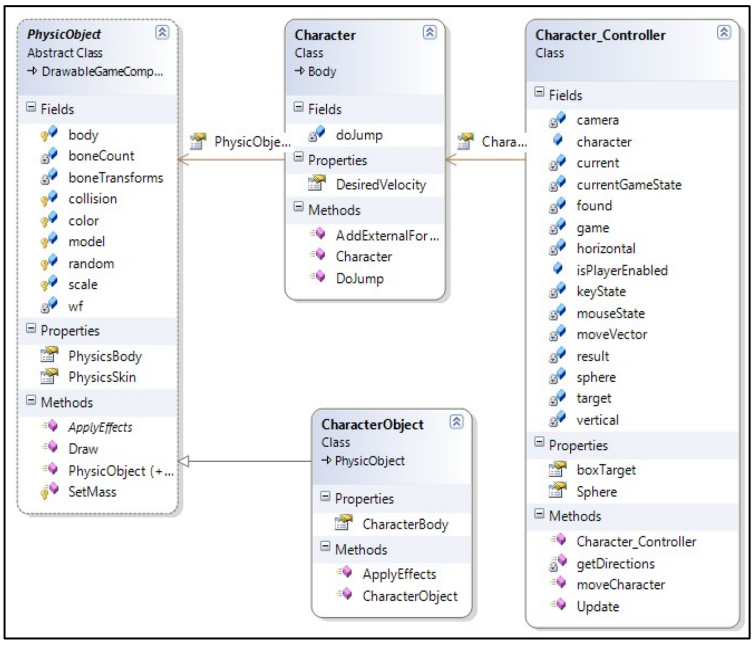


Fig.14 class diagram for character controller

The Character Controller class is also where the movement of the character takes place (See Fig.15,16,17). The movement of the character is a result of both the forward keys on the keyboard and the movement of the mouse for the camera. This is split up into 3 states, if the player is enabled and can move, if the player is not enabled and the cube that moves is where the character is, and if the player is not enabled and any random cube is moving.

The first state is if the player can move. If the player can move then the mouse pointer is pinned and camera position is set to the character object position. The keyboard is then tested for input from the player and the velocity of movement is set based upon this. The camera rotation is also configured here based upon the movement from the mouse (See Fig.15).

```
if (isPlayerEnabled)
            {
                found = false;
                vertical = false;
                horizontal = false;
                current = -1;
                result = Vector3.Zero;

                camera.IsMousePinned = true;
                camera.EnableKeyboardInput = false;
                camera.Position = character.PhysicsBody.Position + Vector3.Up;

                Vector3 moveVector = new Vector3();
                float amountOfMovement = 50.0f;

                if (keyState.IsKeyDown(Keys.Right))
                    moveVector += Vector3.Right;
                if (keyState.IsKeyDown(Keys.Left))
                    moveVector += Vector3.Left;
                if (keyState.IsKeyDown(Keys.Down))
                    moveVector += Vector3.Backward;
                if (keyState.IsKeyDown(Keys.Up))
                    moveVector += Vector3.Forward;

                Matrix cameraRotation = Matrix.CreateRotationX(camera.Angles.X) *
                    Matrix.CreateRotationY(camera.Angles.Y);

                moveVector = Vector3.Transform(moveVector, cameraRotation);

                JiggleMath.NormalizeSafe(ref moveVector);
                moveVector *= amountOfMovement;

                character.CharacterBody.DesiredVelocity = moveVector;

                if (keyState.IsKeyDown(Keys.Space))
                    character.CharacterBody.DoJump();
            }
```

Fig.15 the first state

The second state is when the player isn’t enabled and the current state is inside the cube that moves next (See Fig.16). This means that the camera should pan out and to a distance where the player can see the movement of that cube. This is produced by not allowing the player to enter input, and setting the camera position and target. Also inside this state is the code for moving the player if the cube has moved. The code will get the characters current cube and using the function getDirection() (See Fig.17) will work out the direction of movement. With this data, the player is then moved with the cube.

There originally was a problem with this implementation. The character object would still move, even without the input from the player, because the object has been set on a velocity and direction and will slow down gradually. The stop this happening the code sets the character body’s desired velocity to zero.

```
else if (!isPlayerEnabled && (currentGameState == "Inside"))
            {
                //character.CharacterBody.DesiredVelocity = Vector3.Zero;

                camera.EnableKeyboardInput = false;
                camera.IsMousePinned = false;

                camera.Position = new Vector3(0.0f, 400.0f, 0.5f);
                camera.Target = new Vector3(0, 500, 0);

                if (!found)
                {
                    current = ((Game1)this.game).Manager.currentCharacterLocation();
                    getDirections();
                    found = true;
                }

                if (horizontal)
                {
                    moveVector = new
Vector3(((Game1)this.game).Manager.CurrentPositions[current-1].X,
character.CharacterBody.Position.Y, character.CharacterBody.Position.Z);
                }

                if (vertical)
                {
                    moveVector = new Vector3(character.CharacterBody.Position.X,
character.CharacterBody.Position.Y, ((Game1)this.game).Manager.CurrentPositions[current -
1].Z);
                }

                moveCharacter(moveVector);
            }
```

Fig.16 the second state

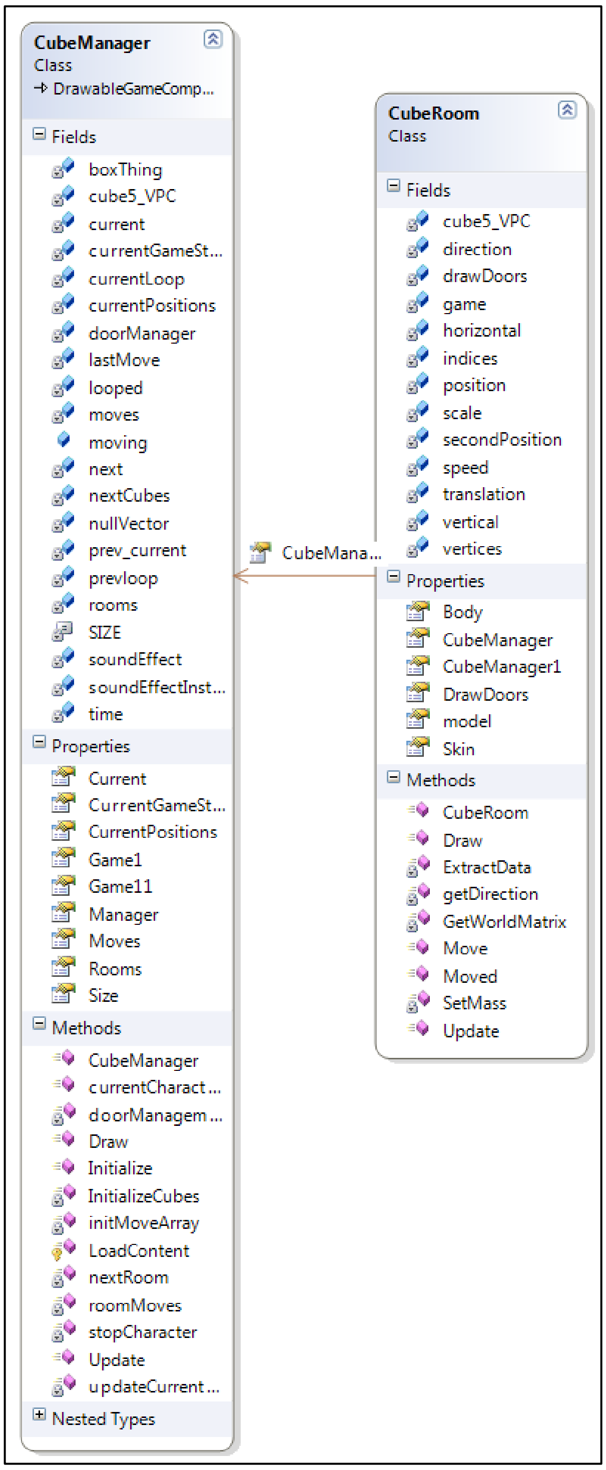
```
private void getDirections()
        {

            Vector3 newPosition =
((Game1)this.game).Manager.Moves[((Game1)this.game).Timer_Class.NumLoops + 1, current -
1];
            Vector3 oldPosition = ((Game1)this.game).Manager.Rooms[current  -
1].Body.Position;

            result = Vector3.Zero;

            if ((newPosition.X  - oldPosition.X)  == 0)
                result.X  = 0;
            else if ((newPosition.X  - oldPosition.X)  > 0)
                result.X  = 1;
            else
                result.X  = -1;

            if ((newPosition.Y  - oldPosition.Y)  == 0)
                result.Y  = 0;
            else if ((newPosition.Y  - oldPosition.Y)  > 0)
                result.Y  = 1;
            else
                result.Y  = -1;


            if (result.X  > 0 || result.X  < 0)
                horizontal  = true;
            else
                vertical  = true;
        }
```

Fig.17 The getDirection Function.

The third and final state is when a cube is moving but the player is not in the cube, if this happens then the player must stop whatever it is doing. For this to work the desired velocity of the character is set to down because the velocity will cancel all other direction other than downward. For screenshots, see Appendix 5 – Screenshots Fig.1-4.

5.4 CubeManager & CubeRoom

The Cube manager and Cube Room classes are complex classes designed to handle the automatic movement of the rooms. The overall algorithm of the cubes moving is pre-defined



and will loop around to the start again when it has finished. The class diagram for the relationship between the CubeManager and CubeRooms is Fig.18.

The CubeManager class starts by initializing the Cubes that will take part in the movements. Each Cube has a starting position and is added to a List. At the same time the 2D array for the moves is also initialized, which is made up of destination for each cube at a given loop.

The Update Loop is the busiest place in the CubeManager class. First the loop updates the current positions of the cubes. Then the current cube that the player is in is updated. Next is the update call for the rooms and afterward is the setting the looped Boolean to false. This happens because the code tests whether or not the number of loops has changed, if so then the next stage of the update call is called, which pauses the timer, sets the current state to whether a random cube is moving or if the player is inside the cube, sets the player to false, so they can't move and moves the room (See Fig.19).

The last part of the update method is testing whether or not the last cube to move has moved all the way to its destination, if so then the timer is unpaused, the player is enabled and the doors are moved to be in the same position they were before they were moved (See Fig.20).

Fig.18 CubeManager and CubeRooms

```
            updateCurrentPositions();
            current = currentCharacterLocation();

            //((Game1)Game).Hud.PrintOut("Current Cube: " + current);
            time += gameTime.ElapsedGameTime.TotalSeconds;

            //Loop through all models and call update
            for (int i = 0; i < rooms.Count; i++)
            {
                rooms[i].Update();
            }

            //Set the current loop
            currentLoop = ((Game1)Game).Timer_Class.NumLoops;

            //Set prevloop to 0
            if (currentLoop == 0)
                prevloop = 0;

            //Only loop when the currentloop is not equal to itself
            if (currentLoop - prevloop > 0 || currentLoop > 10)
                looped = false;

            //pause the timer if looped is false and move the rooms
            if (!looped)
            {
                ((Game1)Game).Timer_Class.Pause = true;
                if (currentGameState != GameState.Inside)
                    currentGameState = GameState.Moving;
                ((Game1)Game).Character.isPlayerEnabled = false;
                soundEffectInstance.Play();
                roomMoves();
            }
```

Fig.19 The Start of the update loop CubeManager

```
// check if the last room has moved all the way, if so then enable the player, set the
next room and un pause the timer
            if (lastMove != -1)
            {
                if (rooms[lastMove].Moved())
                {
                    //((Game1)Game).Hud.PrintOut("");
                    soundEffectInstance.Stop();
                    currentGameState = GameState.Not_Moving;
                    ((Game1)Game).Character.isPlayerEnabled = true;
                    nextRoom();
                    ((Game1)Game).Timer_Class.Pause = false;

                    if (current > -1)
                        doorManager.ChangeCentre(rooms[current - 1].Body.Position);
                }
            }

            //Check for stopping the character if the room is moving
            stopCharacter();
            //((Game1)Game).Hud.PrintOut("GameState: " + currentGameState);

            doorManagement();
```

Fig.20

The functions used by the update loop include:

- roomMoves() – calculates the which cube needs to move and where too
- updateCurrentPositions – updates the current position of the character via the character object
- currentCharacterLocation – base upon the position of the character, the cube that the player is in is then calculated using the update positions of the cubes (See Fig.21).
- nextRoom() – The next room to move is calculated by taking the current loop and adding one. Then the result is tested against all records in the 2D array that don't have a vector3 of (-1,-1,-1)
- stopCharacter() – stop the character moving if the character is in the cube to move, also set the state to 'inside'
- doorManagement() – Changes the centre of the range of doors so that they can move the new position of the current cube if the cube has moved recently.

```
public int currentCharacterLocation()
        {
            for (int i = 0; i < currentPositions.Length; i++)
            {
                if ((((Game1)Game).Character.character.CharacterBody.Position.X >
(currentPositions[i].X - SIZE / 2)) &&
(((Game1)Game).Character.character.CharacterBody.Position.X < (currentPositions[i].X +
SIZE / 2)))
                {
                    if ((((Game1)Game).Character.character.CharacterBody.Position.Y >
(currentPositions[i].Y - SIZE / 2)) &&
(((Game1)Game).Character.character.CharacterBody.Position.Y < (currentPositions[i].Y +
SIZE / 2)))
                    {
                        if ((((Game1)Game).Character.character.CharacterBody.Position.Z >
(currentPositions[i].Z - SIZE / 2)) &&
(((Game1)Game).Character.character.CharacterBody.Position.Z < (currentPositions[i].Z +
SIZE / 2)))
                        {
                            return i + 1;
                        }
                    }
                }
            }

            return -1;
        }
```

Fig.21 – the currentCharacterLocation() function

CubeRooms

The CubeRoom class uses the same principles of other physics objects, however as said above this model for the room is using a triangular mesh collision skin. The rules for using the triangular mesh over a primitive mesh are that the model should then not move using physics, because it could end up being too much for the physics system to handle.

The Class has a function called Move that takes the new position as a parameter (See Fig.22). The direction is then calculated in the same way that getDirection does for the Character Controller (See Fig.17). With the result of the direction, the cube is then moved in the update function by increased or decreasing the position by the direction multiplied by the speed. Then a new translation matrix is calculated and used in the draw function (See Fig.23).

```
public void Move(Vector3 newPosition)
        {
            secondPosition = newPosition;
            direction = getDirection(newPosition);

            if (direction.X > 0 || direction.X < 0)
                horizontal = true;
            else
                vertical = true;
        }
```

Fig.22 the CubeRoom:Move function

```
if (vertical)
            {
                if (this.Body.Position.Z != secondPosition.Z)
                {
                    this.Body.Position += direction * speed;
                    translation = Matrix.CreateTranslation(this.Body.Position);
                }
                else
                {
                    vertical = false;
                }
            }

            if (horizontal)
            {
                if (this.Body.Position.X != secondPosition.X)
                {
                    this.Body.Position += direction * speed;
                    translation = Matrix.CreateTranslation(this.Body.Position);
                }
                else
                {
                    horizontal = false;
                }
            }
```

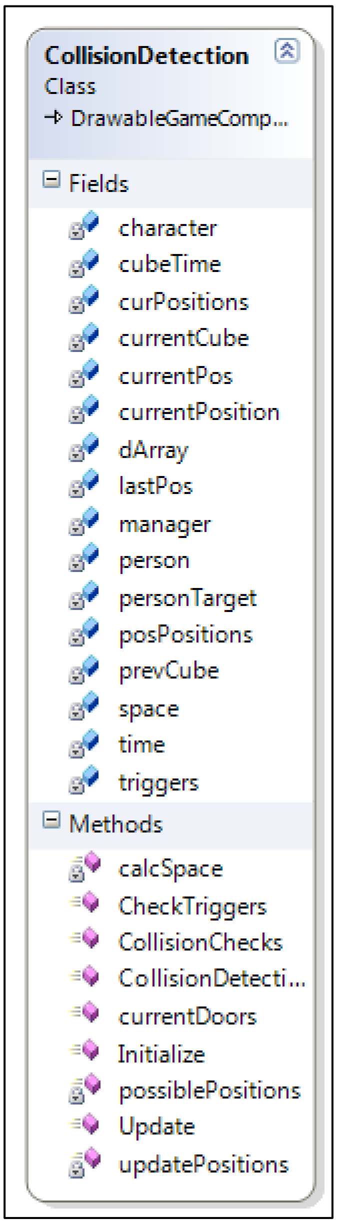
Fig.23 moving the cube

5.5 Collision Detection

The collision detection class is a by-product of a problem with the excess of collision skins. The collision detection class holds the code for testing the collision between the doors and the character (See Fig.24). To do this both the doors and the character have bounding boxes. There are also bounding boxes that protrude further to act like triggers.

The main part of the class involves the positions of the triggers and bounding boxes of the doors and the character. The code is designed to update the current positions of the character and check against the updated positions of the bounding boxes. When the characters bounding sphere intersects with the bounding box of the door, the system tests whether the door is locked. If the door is locked, then when the character hits the door, the player will be forced back. If the door is open then the code calls the function to open the door (See Fig.25).

The class also holds the functionality for locking the doors. The current implementation means that there is only one set of doors that change position to the cube that the player is in. When this happens a switch statement defines whether or not a door is locked (See Fig.26). This same code also defines what door is the finishing door. See also Appendix 5 –ScreenShots Fig 5-6.



CollisionDetection
Class
→ DrawableGameComp...

Fields
- character
- cubeTime
- curPositions
- currentCube
- currentPos
- currentPosition
- dArray
- lastPos
- manager
- person
- personTarget
- posPositions
- prevCube
- space
- time
- triggers

Methods
- calcSpace
- CheckTriggers
- CollisionChecks
- CollisionDetecti...
- currentDoors
- Initialize
- possiblePositions
- Update
- updatePositions

Fig.24 the class for Collision detection.

```
public void CheckTriggers()
        {
            //Collision between trigger to open doors that arent locked
            for (int i = 1; i < triggers.Length; i++)
            {
                if (person.Intersects(triggers[i-1]))
                {
                    //((Game1)Game).Hud.PrintOut2("Trigger");
                    this.manager.openDoor(i);
                }
                else
                {
                    //((Game1)Game).Hud.PrintOut2("");
                    this.manager.closeDoor(i);
                }
            }
        }

        public void CollisionChecks()
        {
            //Collision between camera and door
            for (int i = 0; i < dArray.Length; i++)
            {
                if (person.Intersects(dArray[i].BoundingBox1))
                    if (dArray[i].Locked == true)
                    {
                        ((Game1)Game).Hud.PrintOut2("Locked");
                        character.character.CharacterBody.Position = lastPos;
                    }
                    else
                    {
                        if (dArray[i].Finish)
                        {
                            //change game state to finsihed screen
                            //((Game1)Game).Hud.PrintOut2("FINSIHED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
                            ((Game1)Game).CurrentGameState = Tessera.Game1.GameState.END;
                        }
                        else
                            ((Game1)Game).Hud.PrintOut2("UnLocked");
                    }

            }
        }
```

Fig.25 testing if doors are locked

```
public void currentDoors()
        {

            switch (currentCube)
            {
                case 1:
                    dArray[0].frame =
((Game1)Game).Content.Load<Model>(@"Models/New_Door");
                    dArray[0].Locked = false;
                    dArray[1].Locked = true;
                    dArray[2].Locked = false;
                    dArray[3].Locked = true;
                    dArray[0].Finish = false;
                    break;
                case 2:
                    dArray[0].frame =
((Game1)Game).Content.Load<Model>(@"Models/New_Door");
                    dArray[0].Locked = false;
                    dArray[1].Locked = false;
                    dArray[2].Locked = false;
                    dArray[3].Locked = true;
                    dArray[0].Finish = false;
                    break;
                case 3:
                    dArray[1].Locked  =  false;
                    dArray[2].Locked  =  false;
                    dArray[3].Locked = true;

                    //Case 3 (cube 3) will have the door to escape.
                    dArray[0].frame =
 ((Game1)Game).Content.Load<Model>(@"Models/Finish_Door");
                    //therefore if it is the escape door then the door will be unlocked
                    dArray[0].Locked = false;
                    dArray[0].Finish = true;
                    break;
                case 4:
                    dArray[0].frame =
((Game1)Game).Content.Load<Model>(@"Models/New_Door");
                    dArray[0].Locked = false;
                    dArray[1].Locked = true;
                    dArray[2].Locked = false;
                    dArray[3].Locked = false;
                    dArray[0].Finish = false;
                    break;
                case 5:
                    dArray[0].frame =
((Game1)Game).Content.Load<Model>(@"Models/New_Door");
                    dArray[0].Locked = false;
                    dArray[1].Locked = false;
                    dArray[2].Locked = false;
                    dArray[3].Locked = false;
                    dArray[0].Finish = false;
                    break;
```

Fig.26 example of locked doors switch statement

6. Testing and Evaluation

6.1 Introduction

The testing and evaluation will be split into two sections. The first will test to see how close the implementation has come to the low level objectives defined in the methodology and analysis and the second will see 10 different users test the system and answer a questionnaire.

6.2 Integration Testing

The integration testing is how well the implementation has fared against the low level objectives defined in the earlier sections. The objectives are assessed on a pass or fail basis with comments.

Objective	Test	Outcome	Comments
The game should be developed for an up to date medium	Is the game on an up to date medium?	Pass	The Game is played on the PC.
The game should have a specific start menu	Has the game got a start menu?	Pass	The game has a basic start menu
The player should be able to navigate the start menu	Is the start menu easy to navigate?	Pass	The start menu has simple easy to use buttons for navigation
The Buttons of the start menu should be readable	Are the buttons on the start menu readable?	Pass	The buttons are basic but readable.
The game should be able to exit from the start menu	Can the game exit from the start menu?	Pass	Clicking the exit button exits the game.
The game should be able to start from the start menu	Can the game start from the start menu?	Pass	The game starts by clicking the start button.
The game should have appropriate sound	Does the game have appropriate sound?	Pass	The game has sound, on the menu and in the game.
The game should be able to be ended.	Can you finish the game?	Pass	The game can be finished by passing through a door.
The game should be able to be played with the keyboard and mouse	Can the game be played with the keyboard and mouse?	Pass	You play the game with the keyboard and mouse.
The player should be able to jump	Can the player jump?	Pass	The player jumps with the space bar.
The player should be able to move	Can the player move?	Pass	The player can move with the arrow keys and the mouse.
The player should	Can the player	Pass	The player jumps through a door to

be able to move from room to room	move from room to room?		enter another room.
The player should stop when a room is moving	Does the player stop when a room is moving?	Pass	The player stops when the room is moving and there is a sound played.
The camera should change when the room that the player is in is moving	Does the camera change when the room you're in is moving?	Fail	The Camera does not zoom out enough to see the whole of the cubes.
The player should be able to see the time	Can the player see the time?	Pass	The time is displayed on the HUD.
The players time should be printed at the end	Is the player's time printed at the end?	Pass	The player's time is printed with the success screen.
The player should be able to reset the game at the end	Can the player reset the game at the end?	Pass	The player can reset the game or exit to the main menu.
The player should be able to collide with the walls of the room	Does the player have collision detection?	Pass	The player has perfect collision with the rooms but not so much with the doors
The player should be able to experience realistic gravity.	Does the game have realistic gravity?	Pass	The game has gravity
The player should be able to record a high score	Can the player record a high score?	Fail	The player cannot record a high score.

6.3 Critical Review

The results of the integration testing showed a 90% pass outcome. These tests are however, to test the final implementation with the low level objectives, which is backed up by the evidence given in Appendix 3 – Log Book Entries, namely the agile tasks to complete at the start of each iteration.

6.4 User Testing

User testing will be completed by a group of 10 different users, who will play the game and then answer a questionnaire (for the questionnaire see appendix 1) based upon their experience. The results of the test are below. The questions for the questionnaire are based upon the low level objectives seen above.

Results

The results of the questionnaire have been collated and then the mean value of each question has been outlined below.

- Was the game Enjoyable? – Result 2/3
- Did you get Lost? – Result – 2/3
- Was the Game Hard? – Result – 3/4
- Was collision accurate? – Result – 2
- Were you able to hear and enjoy the sound being played? – Result - 4
- Was the menu system easy to navigate? – Result - 5
- Were you able to reset the game easily? – Result – 3/4
- Would you play the game again? – Result – 2/3

6.5 Critical Review

The results from above showed that the menu system and GUI were the most popular part of the game and the main parts of the game, collision, sound etc. were satisfactory. The comments left by the people who took the questionnaire included the following main points:

- Camera needs to pan out more when the cube is moving – when the camera changes to show the cube moving, the camera has not moved out far enough, therefore all of the cubes cannot be seen in full.
- The mazes can be more symmetrical – The current mazes inside the cubes are not symmetrical enough so that the player gets more disorientated.
- Player stops in mid-air when cubes move – When the player jumps in mid-air and a cube is moving, the player stops and then falls after the cube has moved.
- The last main problem covered by the comments section was that a room in particular (pedestal in the middle) does not collide properly.

These problems are problems that arise from constant player use and cannot necessarily be found by the developer because the developer is constantly working on it. Using this type of testing is good for the end of each iteration, usually completed by a client. For games however it requires testers. These testers unfortunately cannot be introduced until there is a game to test. This means that until a playable game is created, which can take 2-3 iterations, the game will be tested only by integration.

6.6 Evaluation

Both the integration and user testing have given the project a success rate, which is defined by the results of both cases. Unfortunately the User testing does not have a big enough test group for a substantial analysis to be made. There were only 10 people to test the project, which was partly due to time limitations and is a small group. Nevertheless the group produced results which can be evaluated, although a larger group should be consulted. Integration too, has its negatives since the testing has been done by the developer of this project and can be said that the developer would be bias for the success of the project.

From the tests a possible new iteration can be formed, that involves adding a high score list, changing the camera angle, adding more detail to the mazes and allowing the player to keep moving if there in the air.

7. Critical Review

7.1 Introduction

The critical review will look at the project in a wider context and will discuss what was a achieved and what could have been done better. The game has been produced by extensive analysis of possible methods, a design based upon these methods and implemented to the best of those designs.

7.2 What was achieved?

The game was nearly 100% completed to the design specification and low level requirements, which can be seen in the testing and evaluation section. The game was hard to design technically as my previous experience creating games so far, was to create them how I saw them in the moment and now with more emphasis on design before you start was a new experience for me. However this is the reason I chose the agile method as my project methodology because the design and other phases can be changed after each iteration.

One of the features I was trying to get across with this game is the emotion of feeling lost. I think I have successfully implemented this in the game through the use of symmetry of the rooms and not letting the player see the shape of the positions of the rooms before they play. This works so much so that if a player has worked out the shape of the rooms and they then turn the camera quickly, the chances are they don't know where they are anymore. For this reason the sensitivity of the mouse will be kept to the default values in the game.

The main motivations for the game were the film Cube and the game Portal, as discussed in the literature review. The game does the films concept justice, even though there are significant changes from it i.e. mazes in the rooms.

One of the hardest parts of the project was the movement of rooms at fixed intervals. A lot of time was spent on figuring out a possible automatic algorithm for the cube moves but in the end, the easiest way was to pre define the movements. The end classes might be extensive but they are also some of the best work of this project.

The modelling included in the project went well, and what I had to do in order to create some models increased my knowledge of using Maya. For instance, when creating a material in Maya, the material will not export properly to XNA if the material has been changed to effect. The way around this problem was to convert the material to texture, which I had not known about previously.

Overall the project went smoothly even though there were problems at the beginning and near the end of the project.

7.3 Problems

At the start of the project there were two possible methods of developing the game. One method was the final XNA Game Studio implementation and the other was development in the PS Vita. The initial idea was to develop for the PS Vita as it was a new console just out. However due to problems with time scale, lack of support because the development kit was just out of beta stage and proper documentation, the idea was stopped in favour of the XNA

Game Studio, a development kit with plenty of support. For more detailed information see appendix 3 – Log Book Entries.

The other main problem the project faced was in the late stages of development. The physics library, JigLibx, did not work well with so many collision skins of different types at one point. This came to attention during the stage of attaching door to rooms. The solution was to implement a piece of code that would use XNA's Bounding boxes to test collision and set what happens when the player collides, manually. In the end this was created by constantly getting the players previous position by time I.e. every 2 frames. Then using this constant previous position, when the player hits the bounding box the player is placed back in the previous position.

7.4 What I would do differently

Looking back at the project, there are a few things that could have been done differently. First and foremost, the approach that was taken to creating the game would be changed. Unfortunately there was a problem early on with trying to use a particular game engine and because of this there was a lot of project time spent on figuring out how to develop for the engine. A solution to this would have been to pick an engine that had been out in the community for a while, that way if difficulties arise, there is a bigger chance of getting help or at the very least a starting point. Another solution would be to pick an engine that I, myself, know some of, that way there will be less of large transition in implementing a simple game.

During the implementation of the game, I had the opportunity in other modules to try other game engines I had not had the chance of using before. At the fore front of this was the Unity Engine. A drag and drop environment that uses scripting to program functionality. Looking back, I would have created a better more polished game in this engine than in XNA Game Studio.

If I were to get a project that I would have to implement the full development lifecycle again, I would also consider using a non-open source physics library. The reason being is that while JigLibx is fine for most small projects, I did come across some problems, problems that I don't think I'd find in a supported system.

7.5 Improvements

There are many future improvements I would like to make to this game. First is making sure the player can register and save high scores. This can be implemented using C#'s file input output stream to hold high scores in a text file. I believe these needs to be done to add a competitive edge to the game.

The next improvement would to add two higher layers of rooms to the one layer that is already there, each being attached to the other via ladders. This would increase the complexity tenfold, but would also be closer to the film concept. If this was to be implemented then the game needs to have new maze structure sin the rooms and possibly the addition of key items to be collected in order to get out.

Another addition to this improvement would be the automatic and random movement of the cubes, now moving on 3 axis. While making the game I had made it possible, through code, to make the rooms move on 3 axis, however what would be really amazing to implement would be an algorithm that would move rooms randomly based upon the progress of the players journey and also where they are in the rooms. This improvement would increase the games difficulty rating but would also increase the emotion of feeling lost, which is what the original game is aiming to do.

7.6 Ethical, legal and social perspectives

This project focuses on the emotion the player will feel if they are trapped and for that reason there can be the threat of someone taking offensive to this. However because the game is a puzzle game there aren't any social or legal perspectives involved. It can be said however that if the game was turned into a multiplayer game then social aspects would become involved.

8. Bibliography

ApochPiQ, 2012. *The Bag Of Holding.* [Online]
Available at: http://www.gamedev.net/blog/355/entry-2254835-which-x-should-i-use/
[Accessed 27 April 2013].

Bell, D., 15 Sept 2004. *Developer Works.* [Online] Available at:
http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/
[Accessed 28 April 2013].

Bell, D., 16 Feb 2004. *Developer Works.* [Online]
Available at: http://www.ibm.com/developerworks/rational/library/3101.html
[Accessed 28 April 2013].

Bennett, S., McRobb, S. & Farmer, R., 2006. *Object-Orientated Systems Analysis and Design Using UML.* 3rd ed. London: McGraw-Hill.

benson, D., 2010. *Wikipedia.* [Online]
Available at: http://en.wikipedia.org/wiki/File:Agile_Software_Development_methodology.svg
[Accessed 26 April 2013].

Blow, J., February 2004. Game Development: Harder Than You Think. *Game Development: Harder Than You Think,* 1(10), p. 28.

Boehm, B., 1986. *A Spiral Model of Software Development and Enhancement.* s.l.:s.n.

Boehm, B., November 1995. Anchoring the Software Process. *Anchoring the Software Process,* p. 10.

CollabNet, 2009. *Tigris.org.* [Online]
Available at: http://argouml-stats.tigris.org/documentation/manual-0.32/ch04s03.html
[Accessed 28 Arpil 2013].

Linneweber, T., 2011. *JigLibX WikiDot.* [Online]
Available at: http://jiglibx.wikidot.com/
[Accessed 27 April 2013].

Linneweber, T., May 2011. *JigLibX.* [Online]
Available at: http://jiglibx.wikidot.com/
[Accessed 2 May 2013].

McCormack, J. & Conway, D., n.d. [Online] Available at:
http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/07.13.SWEng1/html/text.html
[Accessed 26 April 2013].

MoriBunt, 27 August 2005. *Wikipedia Commons.* [Online]
Available at: http://commons.wikimedia.org/wiki/File:UML_Sequence_diagram.JPG
[Accessed 28 April 2013].

Reed, A., December 2010. *Learning XNA 4.0.* 1 ed. s.l.:O'Reilly Media.

Rose, M., 2013. *GamaSutra.* [Online]
Available at: http://gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php
[Accessed 27 April 2013].

Schwaber, K. & Beedle, M., n.d. *Methods and Tools.* [Online]
Available at: http://www.methodsandtools.com/archive/archive.php?id=18
[Accessed 26 April 2013].

Watters, C., April 2008. *Portal Review.* [Online]
Available at: http://uk.gamespot.com/portal/reviews/portal-review-6190126/
[Accessed 2 May 2013].

Zynga On Game Development Tools (March 2010) Friberg, Morgan.

8.1 Game References

8.1.1 Textures

- gold texture - www.123rf.com
- ladder texture - www.texturelib.com
- rock_texture1 - www.desizntech.info
- crate_texture - www.roblox.com
- Small_Crate_texture - www.spiralforums.biz
- End Screen texture - http://wakpaper.com/id159089/cube-metalic-texture-background-gris-wallpaper-download-1200x800-pixel.html
- Game_GUI - http://all-hdwallpapers.com/textures-hdwallpapers/wallpapers-background-textures-ha/
- tessera background texture - http://www.psdgraphics.com/backgrounds/
- highscores texture - http://metrochessla.com/underconstruction/

8.1.2 Code

- Reed, A., December 2010. *Learning XNA 4.0.* 1 ed. s.l.:O'Reilly Media.
- Linneweber, T., May 2011. *JigLibX.* [Online] http://jiglibx.wikidot.com/

8.1.3 Sounds

- 172937__setuniman__creepy-0v55m2.wav
  http://www.freesound.org/people/Setuniman/sounds/172937/
- 27218__mitchlee83__cogs.wav
  http://www.freesound.org/people/mitchlee83/sounds/27218/
- 31601__dj-chronos__dark-drone.wav
  http://www.freesound.org/people/DJ%20Chronos/sounds/31601/

9. Appendices

9.1 Appendix 1 – Questionnaire

Questionnaire

Please fill out this form at the end of playing my game. Please mark each question out of 5, 5 being strongly agree and 1 being, strongly disagree. Your input is welcome thank you.

Was the game Enjoyable?	1	2	3	4	5
Did you get Lost?	1	2	3	4	5
Was the Game Hard?	1	2	3	4	5
Was collision accurate?	1	2	3	4	5
Were you able to hear and enjoy the sound being played?	1	2	3	4	5
Was the menu system easy to navigate?	1	2	3	4	5
Were you able to reset the game easily?	1	2	3	4	5
Would you play the game again?	1	2	3	4	5

Comments/Improvements

9.2 Appendix 2 – Log Book Template

Log Book

Entry No: **Dates Between:**

Tasks Completed:

Task to do this iteration:

Bugs Found:

Fixed Bugs:

Signed –

Date -

9.3 Appendix 3– Log Book Entries

Log Book

Entry No:	1	Dates Between:	1/10/12 – 1/11/12

Tasks Completed:

Task to do this iteration:

- Get used to working in the PS Vita Environment
- Find Tutorial for PS Vita Development
- Add a 3D Model to the screen
- Add a 3D Cube to Screen

Bugs Found:

Fixed Bugs:

Log Book

Entry No: 2 Dates Between: 1/11/12 – 1/12/12

Tasks Completed:

- Download PS Vita Environment – 7/11/12
    - Downloaded the PlayStation Mobile Suite from the game developers' website.
- Find a Tutorial for Ps Vita Development – 7/11/12
    - Found a tutorial that goes through game building from the beginning.

Task to do this iteration:

- Add a 3D Model to the screen
- Add a 3D Cube to Screen
- Add a Cube to the screen via primitives from PS Vita.
- Get the Cube to Move.

Bugs Found:

1. Can't load a 3D model in the game. Comes up with Error. Possibly because my laptop cannot handle the graphics, but I can't find this out because there is no extensive documentation.

Fixed Bugs:

Log Book

Entry No: 3 Dates Between: 1/12/12 – 1/01/13

Tasks Completed:

- Download PS Vita Environment – 7/11/12
- Find a Tutorial for Ps Vita Development – 7/11/12

Task to do this iteration:

This iteration has changed to now develop in XNA Game Studio

- Create a 3D Cube in Maya
- Export the Cube into XNA Game Studio
- Create the camera to show the cube
- Create the CubeRoom Class to hold the functionality for the cube.
- Create the Cube Manager class to the handle the setup of the cubes.
- Make the cube move.

Bugs Found:

1. Can't load a 3D model in the game. Comes up with Error. Possibly because my laptop cannot handle the graphics, but I can't find this out because there is no extensive documentation.

Fixed Bugs:

Log Book

Entry No: 4 Dates Between: 1/01/13 – 1/02/13

Tasks Completed:

- Download PS Vita Environment – 7/11/12
- Find a Tutorial for Ps Vita Development – 7/11/12
- Create a 3D Cube in Maya
    - Created a simple 3D Cube in Maya.
- Export the Cube into XNA Game Studio
    - Exported the Cube from Maya into an FBX file Format.
- Create the camera to show the cube
    - Created a simple camera to use its View, Projection and World matrices to draw the model on the screen.
- Create the CubeRoom Class to hold the functionality for the cube.
    - Created a class for the cube that draws and updates the cubes position.
- Create the Cube Manager class to the handle the setup of the cubes.
    - Created a CubeManager that holds multiple instances of the CubeRoom class to draw out to the screen in different positions. The Cube Manager will also hold the data for moving.
- Make the cube move.
    - The Cube can move with the input of user.

Task to do this iteration:

This iteration has changed to now develop in XNA Game Studio

- Model New Cube.
- Move the cubes automatically in a predefined pattern
- Print the time to the screen.

Bugs Found:

2. The FBX file format will not export a Maya texture.
3. The Cube Model had separate parts and so when rotated or moved, the parts moved separately.

Fixed Bugs:

1. Fixed the cube models pieces moving separately by combining the mesh in Maya.

Log Book

Entry No: 5 Dates Between: 1/02/13 – 1/03/13

Tasks Completed:

- Download PS Vita Environment – 7/11/12
- Find a Tutorial for Ps Vita Development – 7/11/12
- Create a 3D Cube in Maya
- Export the Cube into XNA Game Studio
- Create the camera to show the cube
- Create the CubeRoom Class to hold the functionality for the cube.
- Create the Cube Manager class to the handle the setup of the cubes.
- Make the cube move.
- Model New Cube.
    - New Cube Modelled
- Move the cubes automatically in a predefined pattern
    - The cubes move in a predefined pattern, through the use of a 2D array and a time step, and the functions required in the cube rooms class.
- Print the time to the screen.
    - The time is printed through to the screen through the HUD Class.

Task to do this iteration:

This iteration has changed to now develop in XNA Game Studio

- Add a physics Library
- Polish off the cubes automatically moving, to a time step
- Produce collision detection with the physics library

Bugs Found:

4. The FBX file format will not export a Maya texture.
5. The Cube Model had separate parts and so when rotated or moved, the parts moved separately.

Fixed Bugs:

1. Fixed the cube models pieces moving separately by combining the mesh in Maya.
2. Fixed the material problem in Maya, by exporting the material as a texture.

Log Book

Entry No: 6 Dates Between: 1/03/13 – 1/04/13

Tasks Completed:

- Download PS Vita Environment – 7/11/12
- Find a Tutorial for Ps Vita Development – 7/11/12
- Create a 3D Cube in Maya
- Export the Cube into XNA Game Studio
- Create the camera to show the cube
- Create the CubeRoom Class to hold the functionality for the cube.
- Create the Cube Manager class to the handle the setup of the cubes.
- Make the cube move.
- Model New Cube.
- Move the cubes automatically in a predefined pattern
- Print the time to the screen.
- Add a physics Library
    - Added the JigLibX Physics Library
- Polish off the cubes automatically moving, to a time step
    - Made the cubes move with a time in step fashion perfectly
- Produce collision detection with the physics library
    - There is minor collision detection implemented.

Task to do this iteration:

This iteration has changed to now develop in XNA Game Studio

- Program way of telling what cube your in
- also what cube will move next
- -SkyBox
- -Add Timer Class
- -Camera to span out and disable input?
- -move characterObject if cube is moving
- -Need capsule slightly higher up
- -Need to have two states (if the cube is moving or not and whether your inside or not)
- -need to stop if ANY cube is moving
- -test against more than one cube when finding next

Bugs Found:

6. The FBX file format will not export a Maya texture.
7. The Cube Model had separate parts and so when rotated or moved, the parts moved separately.

Fixed Bugs:

3. Fixed the cube models pieces moving separately by combining the mesh in Maya.
4. Fixed the material problem in Maya, by exporting the material as a texture.

Log Book

Entry No: 7 Dates Between: 1/04/13 – 1/05/13

Tasks Completed:

- Move the cubes automatically in a predefined pattern
- Print the time to the screen.
- Add a physics Library
- Polish off the cubes automatically moving, to a time step
- Produce collision detection with the physics library
- Program way of telling what cube your in 15/03/2013
- also what cube will move next 15/03/2013
- -SkyBox 15/03/2013
- -Add Timer Class 15/03/2013
- -Camera to span out and disable input? 21/03/2013
- -move characterObject if cube is moving 27/03/2013
- -Need capsule slightly higher up 27/03/2013
- -Need to have two states (if the cube is moving or not and whether your inside or not) 27/03/2013
- -need to stop if ANY cube is moving 27/03/2013
- -test against more than one cube when finding next 27/03/2013

Task to do this iteration:

This iteration has changed to now develop in XNA Game Studio

- -Add restrictions to doors that can't open
- -Remodel doors to one door
- -change layout to 2d
- -Attach doors to rooms (make appear on other side)
- -Create Bounding boxes for doors
- -don't draw doors for rooms that you aren't in
- -Open certain doors
- -bounding box for doors so they open/triggers
- -push the player back when they hit a wall/ collision detection
- -Model new cube with no doors at the bottom
- -work out algorithm for locked doors
- -Program simple happy path that ends with leaving the cubes - specific door
- -Program new cube model in
- -Need to remodel cube so I can't fall through bottom
- -Get GUI textures
- -Create own GUI texture for time
- -GUI - only need a rectangle to hold value in.
- -Simple GUI nearly done with start menu
- -Need to ba able to reset the game

- -need to stop objects when ended
- - Need to add functionality and scripts to end screen and high scores
- -Rework Camera
- -Refine door algorithm for cube free space
- -Global time as high score at the end
- -Sound
- -Need to try and add cube steps to model
- -need to be able to drop y axis when cube is moving
- -GUI buttons need to be disabled
- -Design mazes
- -Model Mazes

Bugs Found:

8. The FBX file format will not export a Maya texture.
9. The Cube Model had separate parts and so when rotated or moved, the parts moved separately.

Fixed Bugs:

5. Fixed the cube models pieces moving separately by combining the mesh in Maya.
6. Fixed the material problem in Maya, by exporting the material as a texture.

Log Book

Entry No: 8 Dates Between: 1/05/13

Tasks Completed:

- -Add restrictions to doors that can’t open
- -Remodel doors to one door
- -change layout to 2d
- -Attach doors to rooms (make appear on other side)
- -Create Bounding boxes for doors
- -don’t draw doors for rooms that you aren’t in
- -Open certain doors
- -bounding box for doors so they open/triggers
- -push the player back when they hit a wall/ collision detection
- -Model new cube with no doors at the bottom
- -work out algorithm for locked doors
- -Program simple happy path that ends with leaving the cubes - specific door
- -Program new cube model in
- -Need to remodel cube so I can’t fall through bottom
- -Get GUI textures
- -Create own GUI texture for time
- -GUI - only need a rectangle to hold value in.
- -Simple GUI nearly done with start menu
- -Need to ba able to reset the game
- -need to stop objects when ended
- - Need to add functionality and scripts to end screen and high scores
- -Rework Camera
- -Refine door algorithm for cube free space
- -Global time as high score at the end
- -Sound
- -Need to try and add cube steps to model
- -need to be able to drop y axis when cube is moving
- -GUI buttons need to be disabled
- -Design mazes
- -Model Mazes

Task to do this iteration:

This iteration has changed to now develop in XNA Game Studio

Bugs Found:

7. Need to maybe pull camera out every time a move happens because you stop in mid-air.
8. Had a problem with floating point variables and moving, game needs whole integer values to test against

9. not using ladders anymore
10. problem with trigger bounding box and door 3
11. character stops in mid-air, this is not good

Fixed Bugs:

- Need to maybe pull camera out every time a move happens because you stop in mid-air. – fixed.

9.4 Appendix 4 – Rough Log Entries

FYP

TO DO

-Title design

****

-Program Movable Blocks

-Screen Rumble

IN PROGRESS

DONE

-Get used to working in the PS Vita Environment 15/10/2012

-Find Tutorial for PS Vita Development 15/10/2012

-Create a 3D Cube in Maya 07/12/2012

-Export the Cube into XNA Game Studio 07/12/2012

-Create the camera to show the cube 07/12/2012

-Create the CubeRoom Class to hold the functionality for the cube. 21/12/2012

-Create the Cube Manager class to the handle the setup of the cubes. 21/12/2012

-Make the cube move. 27/12/2012

-Model New Cube.16/01/2013

-Move the cubes automatically in a predefined pattern 20/01/2013

-Print the time to the screen. 16/01/2013

-Add a physics Library 2/2/13

-Polish off the cubes automatically moving, to a time step 12/2/13

-Produce collision detection with the physics library 23/2/13

-Model new cube - Done // Need to design texture 1/3/13

-Program way of telling what cube your in - Done // But needs going over - odd stuff happens

-Also what cube will move next - Done

- doesnt work for doubles 13/03/2013

-SkyBox - Done 13/03/2013

-Add Timer Class - Done 13/03/2013

-Camera to span out and disable input? - Done - but rotate camera doesnt will always face forward on the way back 11:42 21/03/2013

-move characterObject if cube is moving - DONE - moved with current position instead of moveto positions - had a problem with vertical, because i was getting the wrong loop num and had to +1 - 12:24 27/03/2013

-Need capsule slightly higher up - DONE 12:24 27/03/2013

-Need to have two states (if the cube is moving or not and whether your inside or not) -Done 12:57 27/03/2013

-need to stop if ANY cube is moving - Done 12:57 27/03/2013

-test against more than one cube when finding next - Done 13:34 27/03/2013

-Model Doors/frames & textured - Done 19:36 08/04/2013

-Model Ladder - need to texture 19:45 08/04/2013 - DONE 10:52 09/04/2013

-Model Cubes for maze - 3 types/ big, small, immovable - need to texture 19:45 08/04/2013 - DONE 10:52 09/04/2013

-Add restrictions to doors that cant open 16:09 11/04/2013

-Remodel doors to one door - DONE 18:31 11/04/2013

-change layout to 2d - DONE 11:19 13/04/2013
-Attach doors to rooms (make appear on other side) - DONE 12:00 13/04/2013
-Create Bounding boxes for doors - DONE 16:14 13/04/2013
-dont draw doors for rooms that you arent in - DONE 10:40 14/04/2013
-Open certain doors - DONE 10:57 14/04/2013
-bounding box for doors so they open/triggers - DONE 11:43 14/04/2013
-push the player back when they hit a wall/ collision detection - DONE 10:42 16/04/2013
-Model new cube with no doors at the bottom - DONE 19:43 17/04/2013
-work out algorithm for locked doors - DONE 11:17 19/04/2013
-Program simple happy path that ends with leaving the cubes - specific door - DONE need to hook up to gamestate 12:55 19/04/2013
-Program new cube model in DONE 12:56 19/04/2013
-Need to remodel cube so i cant fall through bottom - DONE 11:35 20/04/2013
-Get GUI textures DNE 11:55 20/04/2013
-Create own GUI texture for time - DONE 11:55 20/04/2013
-GUI - only need a rectangle to hold value in. - DONE 11:55 20/04/2013
-Simple GUI nearly done with start menu 21:19 13/04/2013 - Start menu DONE, Exit screen needed DONE 12:46 20/04/2013
-Need to ba able to reset the game - DONE 12:46 20/04/2013
-need to stop objects when ended - DONE 13:04 20/04/2013
- Need to add functionality and scripts to end screen and highscores - DONE 14:01 20/04/2013
-ReWork Camera - Cant do and have decided that it is more complex as is 15:31 20/04/2013
-Refine door algorithm for cube free space - Done ish 15:04 20/04/2013
-Global time as highscore at the end
-Sound - DONE 12:58 21/04/2013
-Need to try and add cube steps to model - DONE 13:36 21/04/2013
-need to be able to drop y axis when cube is moving - DONE but not perfect 13:40 21/04/2013
-GUI buttons need to be disabled - DONE 14:09 21/04/2013
-Design mazes - DONE 21:17 21/04/2013
-Model Mazes - DONE 21:17 21/04/2013
-Input Cube Mazes 09:26 03/05/2013

BUGS
Need to maybe pull camera out everytime a move happens because you stop in mid air. - DONE 16:20 11/04/2013
Had a problem with floating point variables and moving, game needs whole interger values to test agianst
not using ladders anymore
problem with trigger boundign box and door 3
character stops in mid air, this is not good

9.5 Appendix 5 - Screenshots

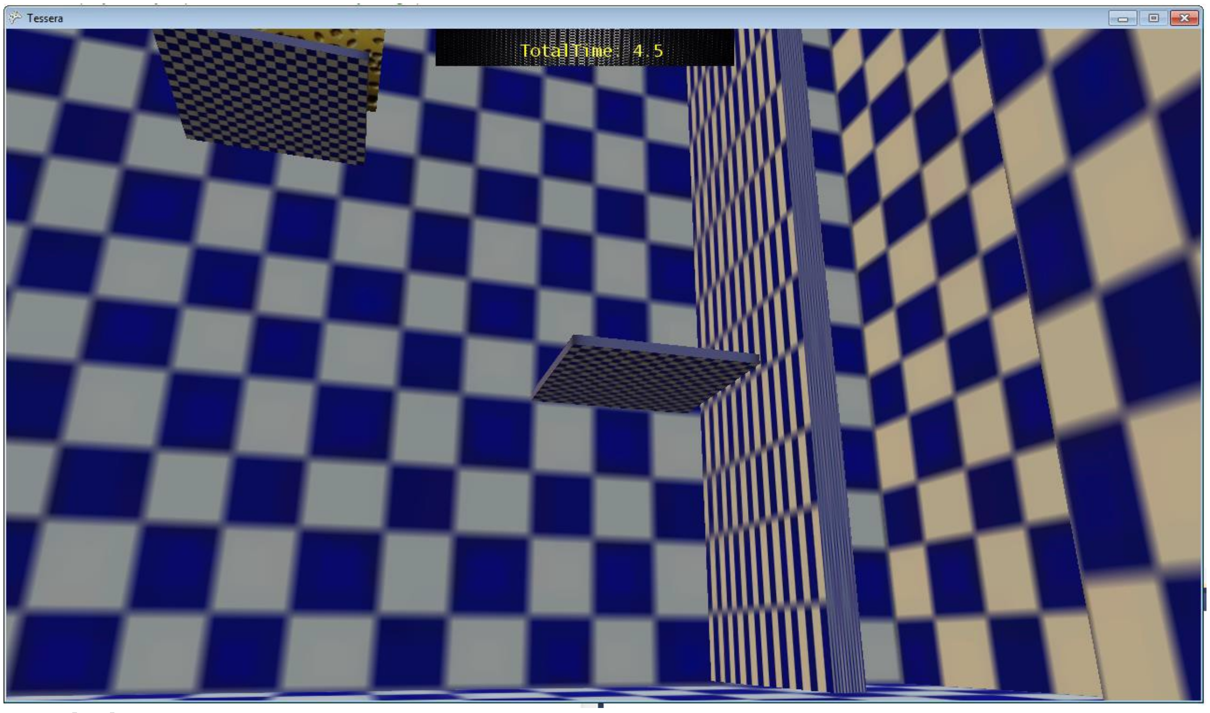


Fig.1 looking to jump

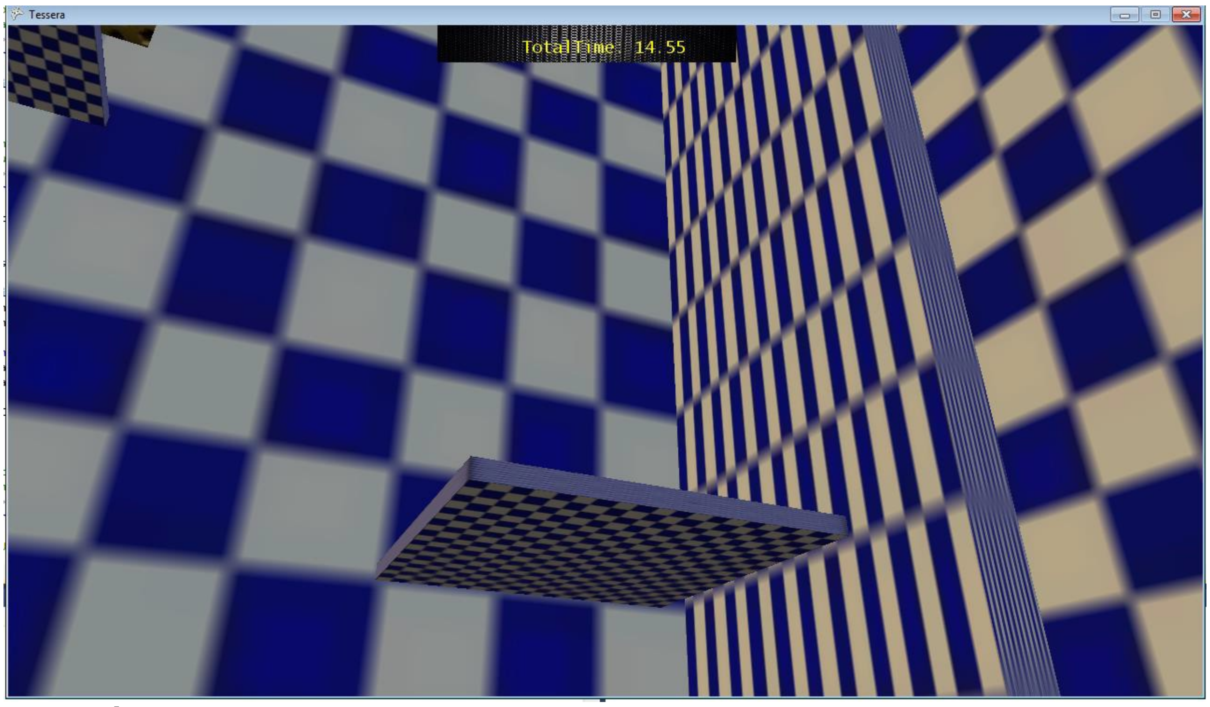


Fig.2 Mid-Jump to step.

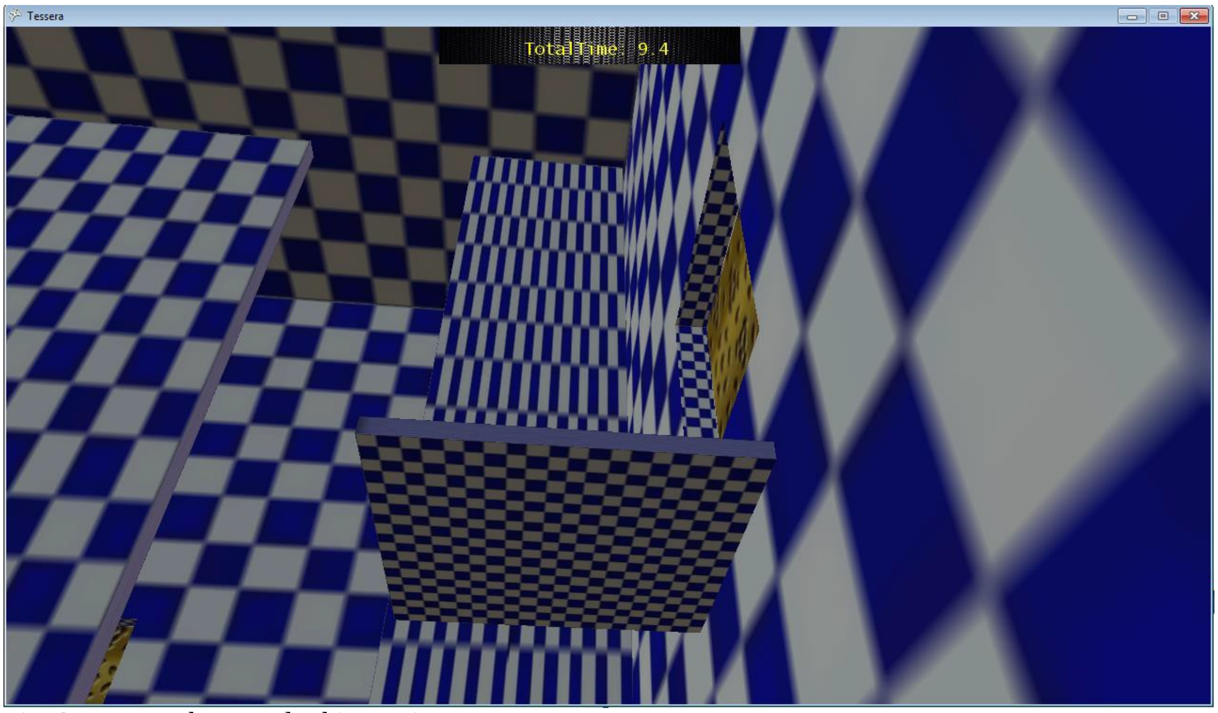


Fig. 3 Now on the step, looking to jump to next step

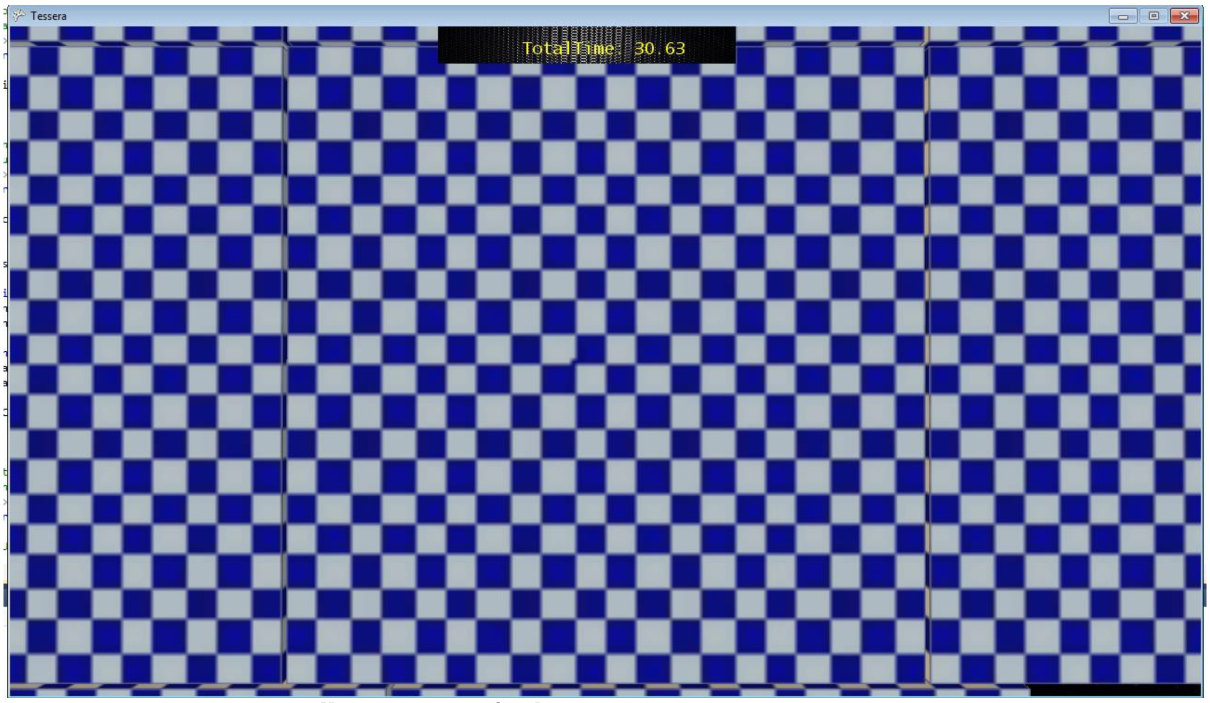


Fig.4 Camera Pans out to allow moving of cube.

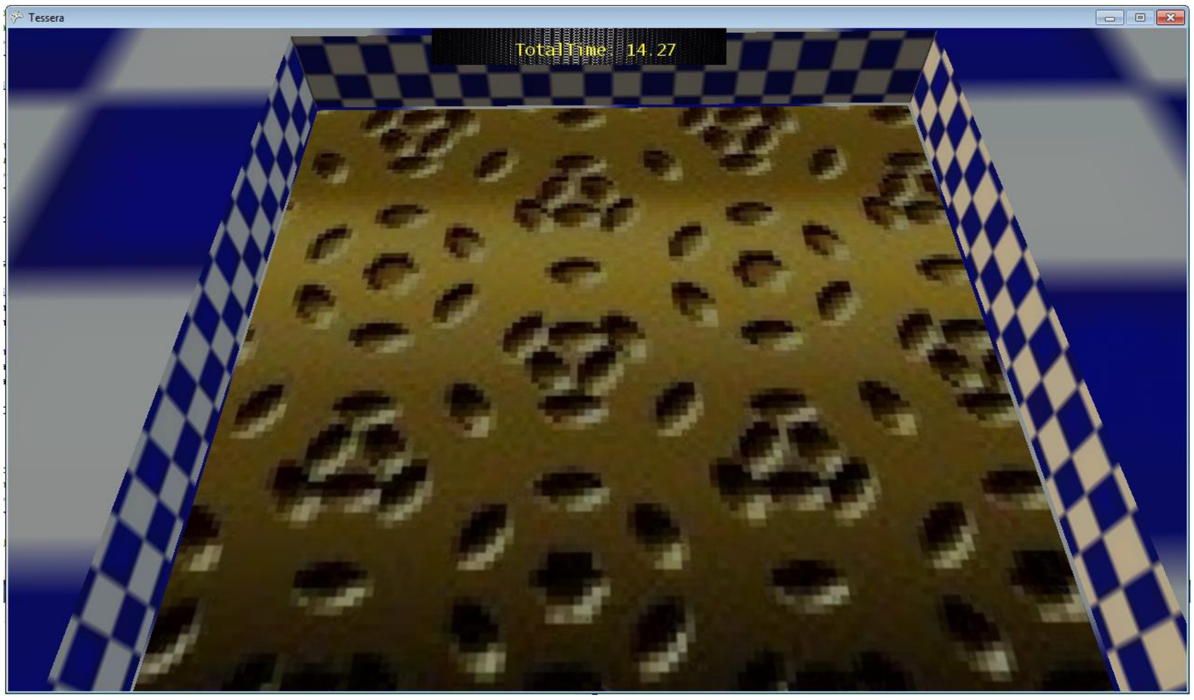


Fig.5 Door to go through

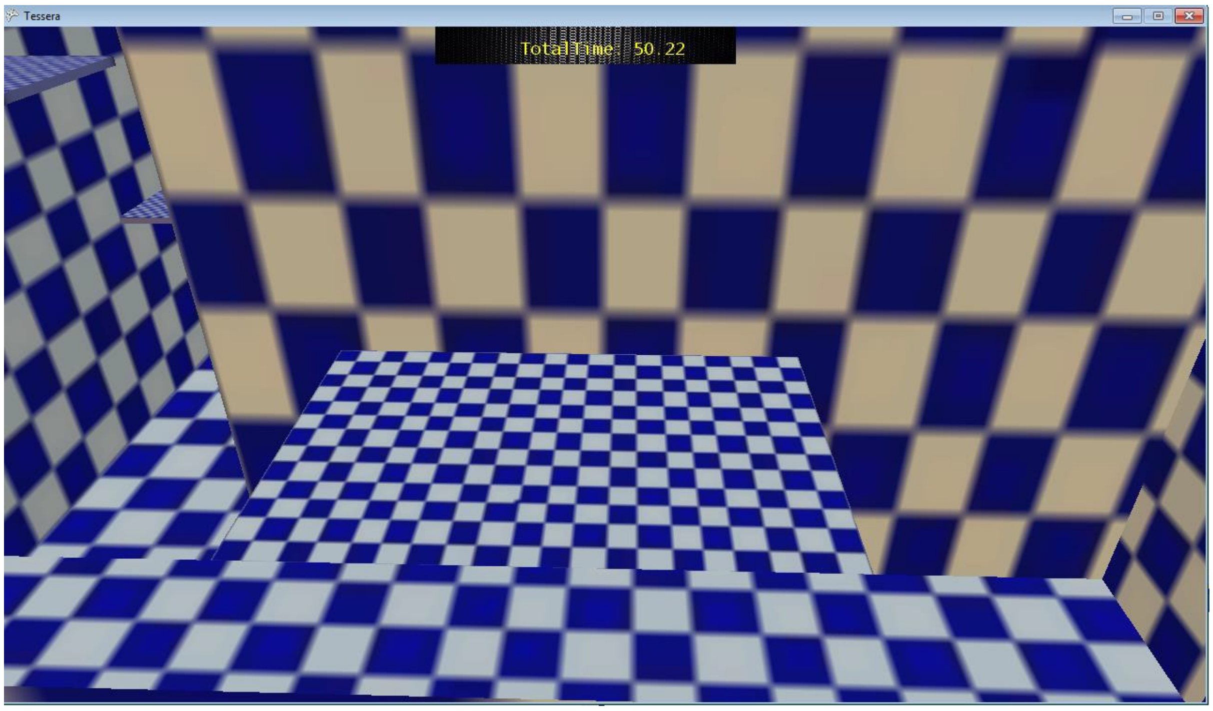


Fig.6 Door Now Open

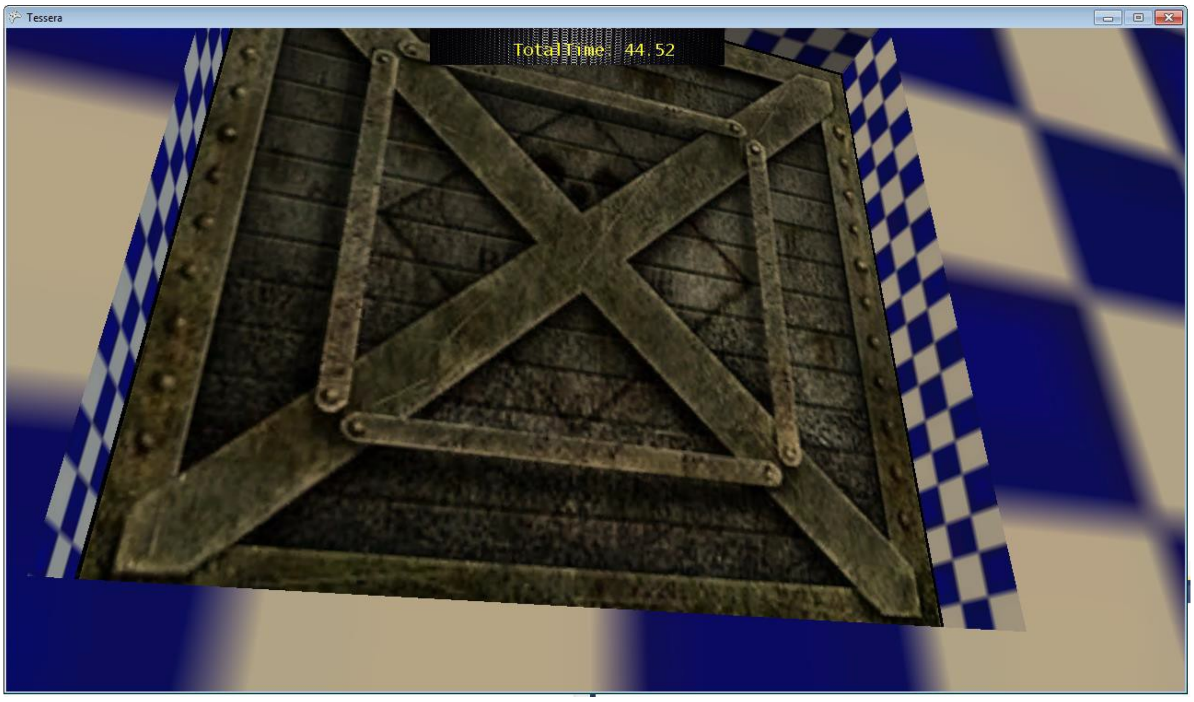
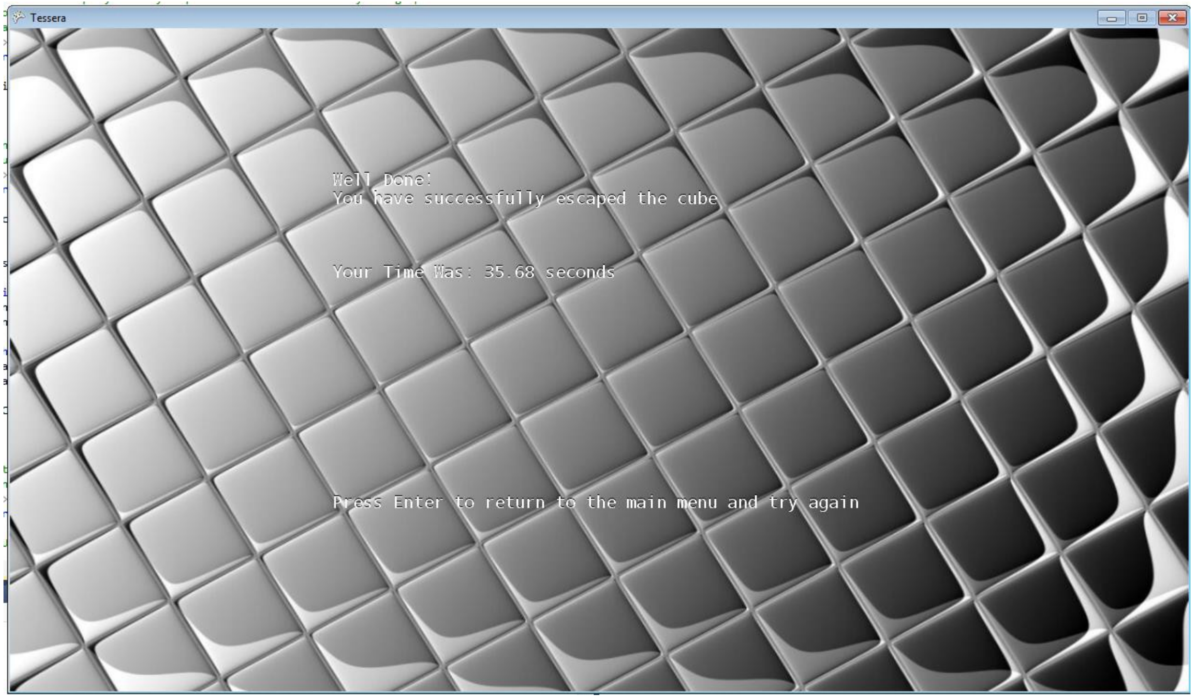


Fig.7 Ending Door



Well Done!
You have successfully escaped the cube

Your Time Was: 35.68 seconds

Press Enter to return to the main menu and try again

Fig.8 Ending Screen

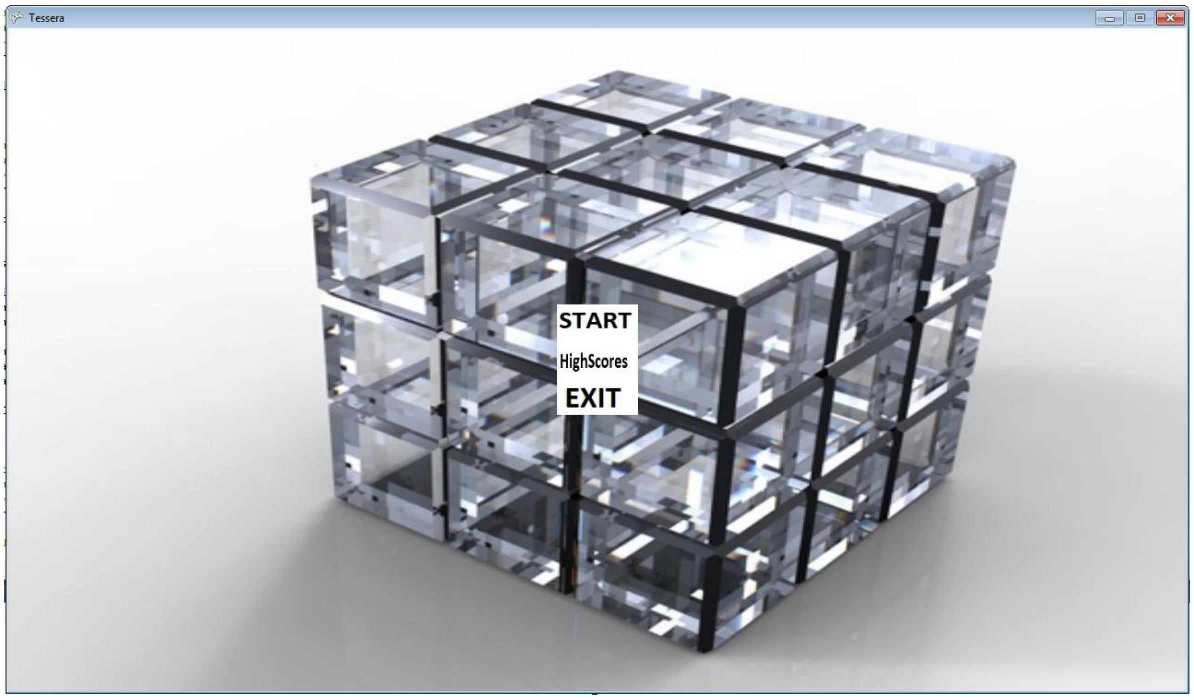


Fig.9 Start Screen