Final Year Project Report

1. Abstract

The purpose of this dissertation is to explore the capabilities of HTML 5 technology. Especially its two features: WebGL and WebSockets. The project involves combination of the mentioned above two technologies with the purpose of developing new game engine. The study is to show that modern browsers are able to compete with other game platforms and consoles. Project will show how modern browsers such as Google Chrome are able to effectively perform 3D graphics rendering and smooth communication with server over the network. This document will compare different browsers and their ability to utilise HTML5 features without third party plug-ins and add-ons. In the lifetime of the project a real game is being built.

2. Acknowledgement

First and foremost I would like to express my sincere gratitude to Dr. James Denholm-Price for his valuable guidance and advice.

I am also grateful for love, understanding and support I received from my family and especially my wife Olesia Grigel. Without them this project would not have been possible.

3. Table of Contents

4. Introduction

Computer Technologies are developing with enormous speed and latest trends show that more and more activities are brought to the web, including social activities, gaming, shopping and many others. Games were in the web for years now, but the experience was never nearly as good as on personal computers or dedicated game consoles. With all modern web technologies available it is now the time to reveal the full power of browser based gaming experience. HTML5 technology has enabled browsers to render graphics, play video and audio files, receive asynchronous messages from a server and do many other things without such plugins as Adobe Flash or Microsoft Silverlight. Browsers implementing HTML5 specification are able to take advantage of graphical processing unit on the machine they are running on.

4.1. Project Aims

Aim of the HelWars project is to deliver a new trend in computer games development by using cutting edge web technologies. The game is to be developed that doesn't require plugins or add-ons to be played right inside the browser window. With no installation required and with a multiplayer enabled. This project aims to combine two features of HTML 5 with the purpose of developing a new trend in games engines. Those features are WebGL renderer and WebSockets protocol. WebGL is responsible for 3D rendering in game, while WebSocket is to be used as client-server communication protocol.

4.2. Project Objectives

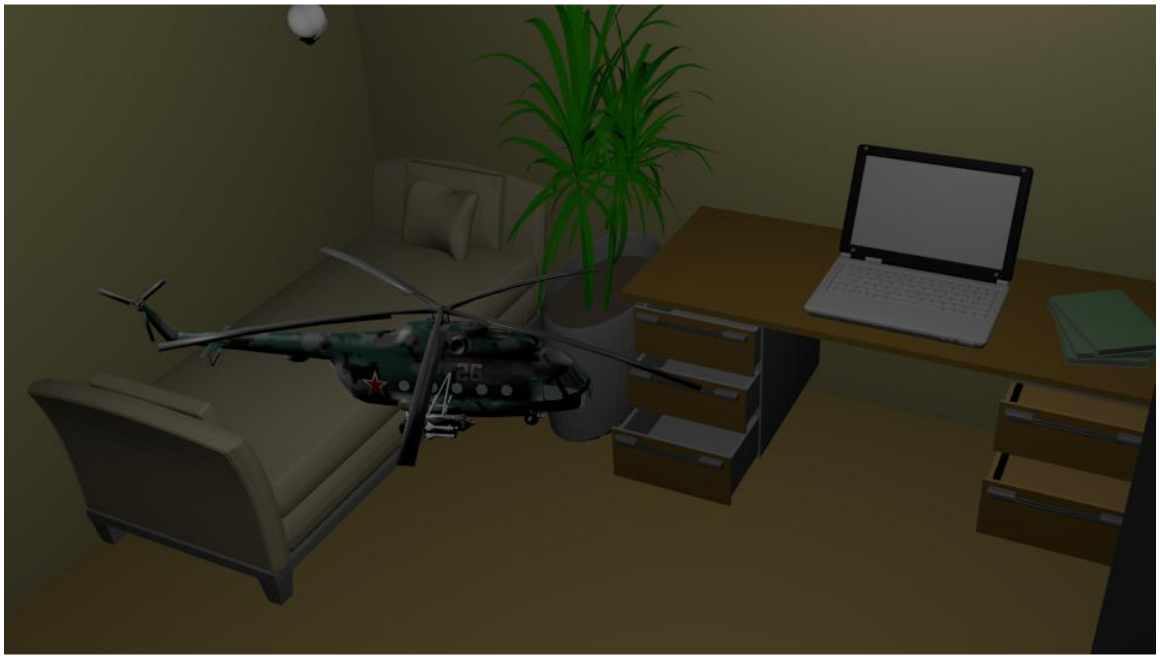
Project can be broken down into series of Objectives.

- Develop 3D engine capable of effectively render Scene, multiple models, missiles and other environment objects.
- Create Server to handle player communications rapidly.
- Server must be able to handle multiple player connections through WebSockets protocol.

- Players must be able to control Helicopter models simulating real flying physics.
- Players must be able to shoot other players in game.
- Chat room functionality needs to be implemented.
- Scores are to be tracked accurately for each player.

4.3. Game Scenario

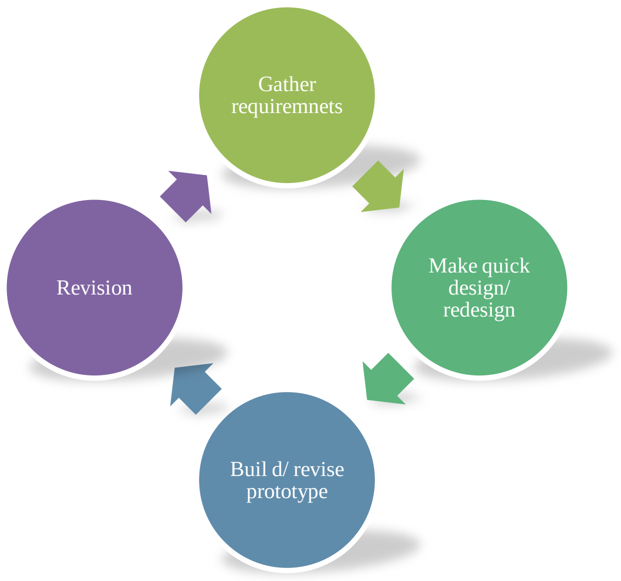
HelWars is a world of plastic helicopter models placed in the environment of the student room. When the player is logged in and connected to the server he is able to control a helicopter. Model is controlled in manner that simulates real helicopter flying physics. Player is able to shoot at the opponent model. Scenario for helicopter wars seems to be not unique since there are games like Helicopter Wars (2010). However this particular game has a little twist. The helicopters are going to be placed in the environment of a student room, with various obstacles common for studio flat. Plastic helicopter models flying in a student room, trying to shoot each other.



5. Methodologies

5.1. Project Management Methodology

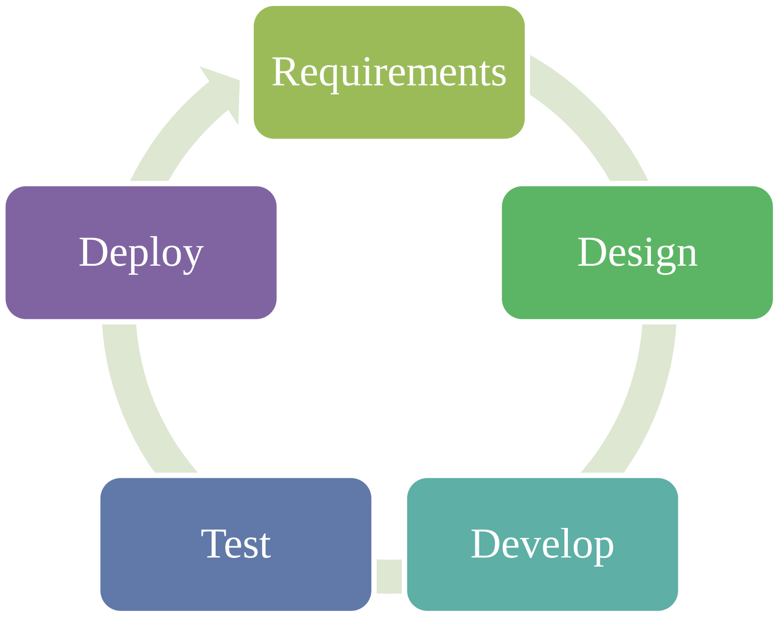
In order manage the project though its lifecycle effectively the DSDM (Dynamic systems development method) framework has been applied. DSDM involves iterative development and software prototyping which works so well with RAD (Rapid application Development) software development methodology. DSDM emphasise the prototyping. The idea is to go through each stage of software development life-cycle quickly to build draft version of the system that can be reviewed with aim to better understand and identify new requirements of the system being built. The process is iterative. First, the requirement gathering occurs. (Not all system requirements are gathered at this stage). A quick system design is then outlined which is followed up by "dirty" build. At this point the system will not be optimised for speed and memory efficiency and will be lacking some functionality. However it will be a working version which will enable development team to find addition requirements. This is called a prototype. The prototype then will be deployed. Feedback will be provided based on the first prototype and the system. At this stage new requirements will be revealed. The iteration will then occur and the new round will start. [PRESSMAN 05]



5.2. Software Development Methodology

RAD (Rapid Application Development) has been chosen as development strategy for the HelWars project. Main reasons for choosing RAD development methodology are very well described in "System Analysis & Design Methods" (2002) by Whitten Bentley. It allows to:

- Organise system development into a series of focused, intense workshops
- Accelerate the requirement analysis and design phases through an iterative construction approach
- The reduce the amount of time that passes before stakeholders can see a working systems
- Reduce time in developing applications and systems; therefore, the initial problem analysis, requirements analysis and decision analysis are consolidated and accelerated



6. State-of-the-Art review

6.1. Demand and requirements

Online Multiplayer games were around the many years including such well-known MMOPRGs (Massively-Multiplayer Online Role Playing Games) as Ultima Online (released 1997) and much newer, more graphically advanced, World of Warcraft and Lineage2, popular first-person shooters games such as Counter-Strike, Battlefield and Call of Duty, and of course a whole lot of browser based online games.

So what makes player to spend days and night playing them? A survey completed by Nick Yee (2006) suggests that typical online gamer is 26 years old and spends on average 22 hours. In his study, “Motivations for Play in Online Games” he subdivided player motivation on three categories called components. Those Components are as follow:

- *Achievement component*
  - *Advancement—The desire to gain power, progress rapidly, and accumulate in-game symbols of wealth or status*
  - *Mechanics—Having an interest in analyzing the underlying rules and system in order to optimize character performance*
  - *Competition—The desire to challenge and compete with others*
- *Social component*
  - *Socializing—Having an interest in helping and chatting with other players*
  - *Relationship—The desire to form long-term meaningful relationships with others*
  - *Teamwork—Deriving satisfaction from being part of a group effort*
- *Immersion component*
  - *Discovery—Finding and knowing things that most other players don’t know about*
  - *Role-Playing—Creating a persona with a background story and interacting with other players to create an improvised story*
  - *Customization—Having an interest in customizing the appearance of their character*

Nick Yee (2006)

From this we can conclude that multiplayer online game needs to have three factors: it needs to be social; Gamers should be able to communicate with each other and being able to unite in groups for achieving goals. There must be a progress line in the game so player can advance their avatars over time. This also adds reward for gamers playing the game for longer time. Similarly people should be rewarded by discovering new bonuses in the game as they progress

6.2. System Components

6.2.1. Graphics Renderer (WebGL)

One of the major browser based 3D game component, is its renderer. Renderer is a rather important part of the game as it has influence on the performance, level of details and rendering speed. WebGL is the renderer chosen for this project. According to Khronos Group (2011)“WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0, exposed through the HTML5 Canvas element as Document Object Model interfaces”. Khronos Group is a consortium that develops standards and API specifications for rich media playback including WebGL.

6.2.2. Graphics Library (Three.js)

In order to simplify the development of the game a lightweight 3D library can be used. There is a library that suits our needs. It is called Three.js. Three.js library is a JavaScript 3D Engine which simplicity allows building 3D apps in a very robust way. This engine can be rendered using WebGL. Three.js library is widely used on Chrome Experiments, for example on brilliant demo called "Ginger" Facial Rigging by David Steele at http://stickmanventures.com.

6.2.3. Server Communication (WebSockets)

WebSockets protocol was developed as a part of the HTML5 specification. WebSocket bring a whole new trend in client-server interaction by introducing bi-directional communication. Before WebSockets the way to communicate with server without reloading webpage was AJAX (Asynchronous JavaScript and XML), however the problem with this communication is that server was unable to send data to client without client triggering communication. Other workaround for this problem was Comet model also called AJAX Push. The problem with those two technologies is well defined by Nikolai Qveflander (2010): “Technologies that enable pushing of data from a server to a subscribing client are not using true asynchronous communication. Instead they emulate this using long polling where the client polls the server for data. If no data exists for the client, the server does not immediately respond but rather waits for data to be available and then sends it to the client. This technique of delayed responses relies on always having an outstanding client request that the server can respond to. What might look like a server initiated communication actually relies on the client making requests to the server." So what we need is a true asynchronous bi-directional communication and HTML5 is able to provide this for us.

6.2.4.Front-end

Front-end of the HelWars project is going to be implemented in HTML5 and JavaScript. To simplify the development HelWars project is going to incorporate various open source JavaScript Libraries such as jQuery, jQuery UI and Three.js

6.2.5.Back-end

ASP.NET is used for building the back end of this project for numerous reasons, firstly because Visual Studio 2010 provides many tools for building web applications. Anything from creating database to member authentication is done easy with Visual Studio 2010. The other reason for choosing ASP.NET is a new version of .NET is being released this year. In this new version, 4.5, framework will provide native support for WebSockets, Microsoft ASP.NET Team (2011)

6.2.6.Database

Since back-end of the application is planned to be implemented in .NET framework it does make sense to use a database from same technology provider. As .NET and SQL Server are both developed by Microsoft they ought to work together well. Since our project is small we can use a free lightweight version of this database – SQL Server Express. Newer version of database is available from http://www.microsoft.com/sqlserver. However the 2012 version is not a final release yet, but the release candidate. According to sqlexpress blog at MSDN (2011) 2012 version was released late November 2011, which makes its stability hardly predictable. For this sole reason we are going to use SQL Server 2008 R2 which has plenty of documentation in such books as Accelerated SQL Server 2008 by Robert E. Walters et al (2008) and MSDN libraries.

6.3. Review of existing systems

6.3.1. WebGL Games

Wide range of WebGL games and application are presented in Chrome Experiments. However most of them do not bring high level of interaction with user. Those applications are experimental and main interaction is input from user of various characteristics like level of details and speed of the animation. X-Wing game by OutsideOfSociety (2011) offers best practices of a 3D game, being very interactive and requiring a lot of concentration and action from user yet being very simple. Not much physics however is involved in this game.

6.3.2. WebSockets Games

Rawkets is 2D online multiplayer space shooter game build using HTML5, WebSockets, and Node.js by Rob Hawkes, Technical Evangelist at Mozilla. This experimental game proves that WebSockets can allow smooth bi-directional communication between server and client. HelWars project is going to be using WebSockets as well however as back end being built in ASP.NET it makes sense to use .NET implementation of WebSockets instead of Node.js. Stefan Schackow and Paul Batum (2011) describe how easy it will be to build a WebSockets application in .NET4.5 when it will be released later in 2012. However .NET4.5 is unavailable at the moment hence we need non-native implementation of WebSockets on .NET such as project Fleck by Jason Staten (2011) which is freely available via Github.

6.3.3. Helicopter Games

Helicopter control is available in many games such as Battlefield and Call OF Duty, however in those games helicopter control is just a small bit of the game. Let's look at the Helicopter Wars (2010) game that is all about helicopters. The game is very easy to control, there is no learning curve to be able to control the helicopter. By achieving the ease of controlling helicopter developers had sacrificed on detail of physics. A game dedicated to a very narrow interest should implement realistic physics.

6.4. Other Areas

6.4.1. Security

There are many rumors on the web regarding security matters of the WebGL. The main issues mentioned in WebGL security issues are Denial of Service and Cross-Origin Media. Those issues are explained in details in WebGL security whitepaper by Khronos Group (2011).

6.4.2. Denial of Service

Denial of Service is a scenario when Graphics Processing Unit is loaded with a large mesh. The computational cost for such operations are high, hence GPU takes a lot of time to process given operation and is unable to respond to other requests. Since some components of the operating system require some job done by GPU system appears to be frozen and may crash. Microsoft starting from Windows Vista has overcame this issue by setting the time limited for which GPU can work on one task.

6.4.3. Cross-Origin Media

Cross-Origin Media was a security issue brought from original WebGL specification. Shaders could have been used to indirectly gather the contents of textures uploaded to the GPU. “For security reasons, JavaScript code on a web page is not allowed to read the pixels of an image or video which comes from a different site.” Khronos (2011), however as WebGL can directly access GPU it was possible to steal images and videos from other sites. Eric Bidelman (2011) states at Chromium Blog that both Google Chrome starting from version 13 and Firefox from version 5 will not be allowing cross-domain media on WebGL.

6.4.4. Authentication

ASP.NET has a build-in support for authentication and Membership. Visual studio 2010 is able to automatically create a database for users, Mathew MacDonald et al (2010)

6.5. State-of-the-Art Conclusion

After reviewing available literature and web articles as well as existing games and web application we can conclude that project HelWars is achievable within given time frame. Current technologies suffice to meet the requirements. However it will be easier to build the multiplayer game when .NET 4.5 framework has been released with native support for WebSockets.

7. Analysis

7.1. SWOT Analysis

7.1.1. Internal Factors

- Strengths

  The main strength of the HelWars project is in the technology being used to develop it. Modern browsers do not require plug-ins or add-ons to run the application written in HTML5.

  Another advantage of the HelWars project is its Authoritative Client-Server communication model. This architecture ensures no cheating is possible on the client side. This is a rather important aspect as being a browser based game, part of the application source code is not compiled and available to any user though the development tools. Authoritative client-server model takes most of the processing away from client machine and puts it on remote server code that is not accessible by user. This also helps to keep the code secure from client with aim to prevent plagiarism.
- Weakness

  Since the game is rendered though the Web Browser it is more likely to be vulnerable to a players cheats. By altering the Source code and running JavaScript queries users can potentially cheat in the game. This issue however was resolved by Authoritative client-server communication model. (See section 6.1).

7.1.2. External Factors

- Opportunities

  HelWars application is written using cutting edge technologies and libraries. It utilises a very new WebSockets protocol. 3D rendering is drawn using THREE.JS library whose popularity grows with enormous speeds. Server side of the application is written in an extensible Object-Oriented manner. All this makes project open to the future development. Game engine developed during this project can be easily utilised for development of other kinds of games, not necessarily 3D or multiplayer games. Part of the engine could be utilised to make simple 2D game with multiplayer or advanced 3D shooter with no multiplayer option.

- Threats

  One of the main threats to the projects is browser compatibility. Currently not all major browsers are supporting HTML5 fully. The table below shows browser compatibility*.

Browser/ Feature	Chrome	Firefox	Safari	IE9	Opera
WebGL	Yes	Yes	No	No	No
WebSockets	Yes	Yes	Yes	No	No

* Following tools were used for browser compatibility testing:

For WebGL - http://get.webgl.org/

For WebSockets - http://websocket.org/echo.html

It can be seen that only two browsers are supporting both of the HTML5 features that are required for the project. This is a potential threat as not all users are wishing to change their browser preference just to play one game. This could potentially turn the clients and investors away. Although not all major browsers currently supporting all HTML5 features, we can assume newer releases will add the support over time. There is an unofficial WebGL plugin available for Internet Explorer here: http://iewebgl.com.

Despite the fact that most calculations are brought to the server some code is still available to the client. This is not sufficient for users to cheat or hack the game. However this enables them to view the structure of rendering techniques. Fortunately this part of the application is a standard way of rendering graphics through THREE.JS library. So this does not affect application security.

7.2. Risks

There are several risks concerned with this project. Main ones are: Would there be enough time to complete project? Would there be enough documentation on WebSockets to achieve multiplayer? There is a hardware risk: Would the server respond time be sufficient to handle real-time multiplayer game? Unfortunately it is hardly possible to answer those questions so early in the project.

7.3. Requirements Analysis

7.3.1.Market Analysis

HelWars game is oriented on students because of its environment - student's room. There are almost 2 million students in UK (BBC 2009) which makes 3.15 percent of UK population. It is a huge market, and this is only the higher education students and only in UK.

7.3.2.Technical Analysis

One thing to consider when creating something completely new using cutting edge technologies is to check whether this can be built at all. The easiest way to do so is to look at what is currently available.

There are number of WebSockets demos available on the web. Rawkets (online 2011) and RUMPETROLL (online 2011) are 2 WebSockets games that match our needs quite closely. They both are using web sockets to create a multiplayer game. This would mean that WebSockets multiplayer game is achievable.

There are many WebGL demos available at Chrome experiments (2011) which prove the 3D in browser is achievable. Another interesting 3D demo is available of rendering Quake 3 maps with WebGL (TojiCode 2010)

All above demos prove it is possible to make multiplayer games and it is possible to make a 3D game. However what about 3D multiplayer game? On the web there seem to be no such thing available. Although Google claims they have created a browser based Quake (Quake II GWT Port Google Code 2010), the game itself however is only available through building on local machine. There is a video of the gameplay available online (GWT Quake YouTube 2010)

Importing models to WebGL can be a bit tricky task. But with the Help of Three.js library that can be achieved. The problem is that Three.js can only understand model mesh in JSON format. However Three.js library provides an add-on for Blender that can help exporting most popular 3D model formats like .OBJ . SVG .3DS. to JSON format .JS. The only obstacle that can be foreseen right now is converting .MD2 models to JSON mesh. It is not vital however as there are other options available.

Another option is creating models in open source 3D modeling SDK Blender. Models created in blender can be easily exported to JSON mesh file.

7.3.3.Timescale Analysis

It might seem very challenging to complete this project by the deadline because of its complexity.

Hence this project can be broken down into 2 parts:

1. Browser Based 3D game.
2. Use of WebSocket for creation a non-3D multiplayer game

Development team (consisted of one person) already has experience in implementing computer vision applications although using different technologies – OSG and OpenGL. This should ensure fewer problems during creation of the first part of the project.

Development team does not currently have skills using WebSocket and there are not so many books available. However the range of options for WebSockets server ensures plenty of online documentation. WebSocket API (2011) also became recently available.

Another issue is the amount of time required for creating 3D models. This can be solved by using models freely obtainable on the web.

7.4. Requirements Specification

7.4.1. Functional Requirements

- Model Controls Simulation
  Model controls are to be developed in a way that they simulate a real helicopter flying experience to a certain extent. This requirement shall not downgrade the usability in the game. The learning curve should be as smooth as possible.
- Smooth graphics
  Graphics in the game should be smooth without too high hardware requirements. Because the application will be able to take advantage not only of a computer processor but a graphics processing unit as well, the minimal hardware requirements are to be outlined. Those however should not be set too high.
- Fast client-server communication
  Server shall be sending the data to the client at about the same rate as rendering fps (frames per second).Even though it is difficult to predict what will be fps as it will depend on factors that are specific to users' hardware and browser choice. Smooth rendering normally starts at 30 fps which means the server should send new data to the client every 30 milliseconds.

- Object-Oriented server

Server must be written in an Object-Oriented manner. This not only means an object oriented language to be used, but that architecture to be organized to ensure smart pattern of communication between classes.

7.4.2.Non-functional Requirements

- **Usability**
  Application controls were built with usability matters in mind. Helicopter controls built similarly to other popular games that have helicopters simulations.
- **Portability**
  Key feature of the application is that it should not require an installation except for browser which is a pre-requisite. This makes the application portable. Users must be able to launch the game whenever they have fast and reliable access to the internet.
- **Stability**
  In the first version of the game it might not be very stable, so it is going to be constrained to 2 players connected to the server. This will help to test the server and client part of the application. Once completely stable the constraint can be removed
- **Accessibility**
  Accessibility is hardly achievable by HelWars projects. Reasons for that are dictated by the technology being used to develop the application. Unfortunately the HTML5 is currently not fully supported by all browsers (see section 4.1.2 for browser compatibility) this mean that it will not be accessible by number of people who are still using older browsers. Browser detection is however built in to the application so it will notify the user that his browser is incompatible and it will direct user to the relevant upgrade page.

7.5. Prioritisation

Following the DSDM framework, MoSCoW prioritisation technique was utilised in the project. MoCSoW prioritisation allowed placing importance breakdown on the list of requirements and features. According to "A Guide to the Business Analysis Body of Knowledge" (BABOK Guide) (2009) MoSCoW analysis divides requirements into four categories: Must, Should, Could, and Won't. Category descriptions are as follows:

- Must: Describes a requirement that must be satisfied in the final solution for the solution to be considered a success.

- Should: Represents a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary.

- Could: Describes a requirement which is considered desirable but not necessary. This will be included if time and resources permit.

- Won't: Represents a requirement that stakeholders have agreed will not be implemented in a given release, but may be considered for the future.

Following is the list of requirements, features and options organised using MoSCoW technique:

- Must
  - Rendering in Browser
    It must be possible to render the graphics in the browser without installing addition plug-ins or add-ons. At the time of project imitation only two browsers are supporting all required elements of HTML5. (See section 4.1.2 for more details)
  - WebSockets

WebSockets protocol is to be used as the main protocol for client-server communication.

- Multiplayer

  Multiplayer is essential part of the HelWars game. WebSocket protocol of the HTML5 specification is to be used for smooth bi-direction communication between client and server

- Realistic Model Controls
  - Tilts

    Models are to be tilting during forward and backward moves to simulate real helicopter flying physics
  - Smooth acceleration

    Models should respond to user input in smooth way slowly accelerating and tilting with the aim to make impression of realistic physics.
  - Smooth camera

    Camera is to follow helicopter slowly with a delay allowing user to see the model being controlled from sides. Such delay in the camera movement would reveal the beauty of 3D rendering.
- Fast client-server communication

  Client-Server communication needs to be fast and reliable. WebSockets protocol is to be used in order to achieve this requirement.
- Collision Detection

  For simplicity the Collision Detection that is to be used for the HelWars project is a simple parallelepiped box that will detect any collisions inside. (See section 9.2.3 Collision Detection)

- Should
  - Use Three.js library

    Three.js library provides easy to use functionality out of the box. This library is to be used for 3D rendering part of the game.

- Use Fleck WebSockets

  Fleck WebSockets is a simple open-source solution for WebSockets protocol written in C# .NET
- Object-Oriented Server

  Game entities should be related in an object-oriented manner. Server required to be written in extensible way, so that more types of objects could be written in future to improve the game. (See section 12. Future Enhancements)
- Animated models

  In order to improve realistic gaming experience model are to be animated. Helicopter propeller shall rotate creating the effect of flying helicopter.
- Textures on models

  Textures and materials of the models should render in way that models look realistic.

- Could
  - Terrain/Environment

    Environment of the HelWars game should look like a student room with various common objects for such location. Objects like desk, books and laptop should present in the game environment.
- Won't
  - Advanced Collision Detection

    Advanced collision detection is not in the scope of current project. Only simple square box collision detection is to be implemented as described above. Although Collision detection is not going to be implemented it would be possible to add more advanced detection in future using JigLibJS (2011). (See section 12. Future development)

8. Design

8.1. Client-Server Communication

Unity3D game engine documentation describes 2 types for client-server communications: Authoritative and Non-Authoritative. According to this documentation "The authoritative server approach requires the server to perform all world simulation, application of game rules and processing of input from the player clients. Each client sends their input (in the form of keystrokes or requested actions) to the server and continuously receives the current state of the game from the server. The client never makes any changes to the game state itself. Instead, it tells the server what it wants to do, and the server then handles the request and replies to the client to explain what happened as a result." and "A non-authoritative server does not control the outcome of every user input. The clients themselves process user input and game logic locally, then send the result of any determined actions to the server. The server then synchronizes all actions with the world state. This is easier to implement from a design perspective, as the server really just relays messages between the clients and does no extra processing beyond what the clients do."

Because of the issue described in section 4.1.1 it is necessary to minimize chance of players cheating. Hence it would be more sensible to Use an Authoritative client-server communication model.

I game client will be sending to the server the intention to make certain movements e.g. after hitting button "UP" or "DOWN" server will decide whether the model can be moved in the requested direction. This way server will be calculating the collision detection as well as the missile locations. This concept disallows player to cheat by running JavaScript queries altering his locations or Scores.

Here is the code snippet that shows how the client intention to move or shoot a missile is sent to server by browser.

```
//Sending shots intention to the server
   container.addEventListener('click', function (e) {
      //Senging message to server
      ws.send("{\"type\":\"shot\"}");
   }, false);

//Tracking key hits and sending intention to move to server:
   $(window).keydown(function (e) {
      //key press Q for turning left
      if (e.keyCode == 81) {
         modelLeftTry = true;
         ws.send("{\"type\":\"moves\",\"moveType\":\"modelLeftTry\", \"value\":\"true\"}");
      }
```

The server receives the message and assigns the appropriate values to the ***Player*** object.

```
socket.OnMessage = message =>
            {
//messages from clients handled here
//Reading message
dynamic msg = DynamicDeserialize(message);

if (msg["type"] == "moves")
               {
                  Player tempPlayer = players.Find(p => p.wsId == socket.WsId);
                  switch ((String)msg["moveType"])
                  {
                     case "modelLeftTry":
                        players.Find(p => p.wsId == socket.WsId).modelLeft =
Boolean.Parse(msg["value"]);
                        break;
}
```

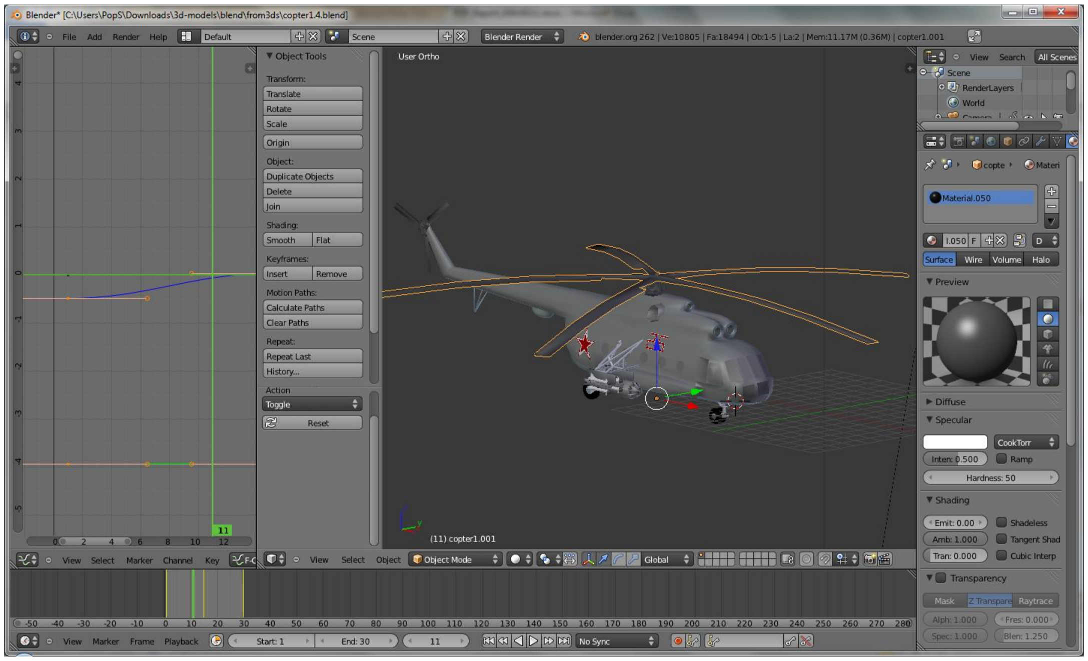
After the server updates the locations for each player it sends the message back to all clients with new locations of each player in game:

```
//updating players positions on server
players.ForEach(p => p.updateLocation());

//updating clients
allSockets.ForEach(s => s.Send(ManualSerialize(typeof(Player))));
```

8.2. Game Model Design

It was difficult to find suitable open-source models for HelWars project. Models requirements were quite specific. The solution was to take open-source 3D models from http://archive3d.net/ which license allows modification of the models for non-commercial purposes. Import them to Blender, add animation and alter shapes and materials. The THREE.JS plug-in for Blender was used to export models in appropriate format.



9. Implementation

9.1. Client-side Implementation

9.1.1. Communication with server

Client Communication with server has been established through WebSockets protocol. It is written using JavaScript code. WebSocket implementation on the client side is fairly straightforward. Three events can occur on client side:

- OnOpen and OnClose

  Those two events are not having any significant callback. They only inform user that the connection has been established or terminated:

```
// when the connection is established, this method is called
ws.onopen = function () {
        log.innerHTML += '.. connection open<br/>';
};
// when the connection is terminated, this method is called
ws.onclose = function () {
        log.innerHTML += '.. connection closed<br/>';
}
```

- OnMessage

  This event handles all important messages that come from server. Data comes from server is encoded in a JSON string that is simply decoded using jQuery.parseJSON method:

```
//parse incoming message to JSON object
var JInfo = jQuery.parseJSON(evt.data);
```

  Once message is decoded script can decide how to handle the message depending on its type:

  - Receiving own ID

```
case "id":
        //Update users wsId - will receive once connection is opened
        myWsId = JInfo.wsId;
        log.innerHTML += "You were assigned id " + myWsId + "<br/>";
```

  - Receiving message about new player in game:

```
case "newOpponent":
        //Update users wsId - will receive once connection is opened
        opWsId = JInfo.wsId;
        log.innerHTML += "New player in game, id " + opWsId + "<br/>";
```

  - On message about hit, check who got hit and ouput this to user:

```
case "hit":
         //Server will tell who got hit.
        if (myWsId == JInfo.wsId)
                log.innerHTML += "You got hit!<br/>";
        else
                log.innerHTML += "Player with id " + JInfo.wsId + " got hit!<br/>";
```

  - On any text message output it to user:

```
case "message":
        //Any message from server to be displaed on screen
        log.innerHTML += JInfo.message + "<br/>";
```

- On player locations message script stores all the position and rotation parameter in global variables that are user for model rendering. This message brings not only the players coordinates but coordinates for the opponents as well:

```
case "playerLocations":
        //Server will send all players locations
        for (var l in JInfo.locations) {
         if (myWsId == JInfo.locations[l].wsId) {
                playerPositionX = parseFloat(JInfo.locations[l].playerPositionX);
                playerPositionY = parseFloat(JInfo.locations[l].playerPositionY);
                playerPositionZ = parseFloat(JInfo.locations[l].playerPositionZ);
                playerRotationX = parseFloat(JInfo.locations[l].playerRotationX);
                playerRotationY = parseFloat(JInfo.locations[l].playerRotationY);
                playerRotationZ = parseFloat(JInfo.locations[l].playerRotationZ);
        }
        else {
                opponentPositionX = parseFloat(JInfo.locations[l].playerPositionX);
                opponentPositionY = parseFloat(JInfo.locations[l].playerPositionY);
                opponentPositionZ = parseFloat(JInfo.locations[l].playerPositionZ);
                opponentRotationX = parseFloat(JInfo.locations[l].playerRotationX);
                opponentRotationY = parseFloat(JInfo.locations[l].playerRotationY);
                opponentRotationZ = parseFloat(JInfo.locations[l].playerRotationZ);
        }
}
```

- Similar to player locations, missile location message update coordinates to all missiles:

```
case "missileLocations":
        gloD = evt.data;
        glo = JInfo;
        //Server will send all bullets locations
        for (var m in JInfo.missileLocations) {
                missiles[m].position.set(JInfo.missileLocations[m].shotPositionX,
                JInfo.missileLocations[m].shotPositionY,
                JInfo.missileLocations[m].shotPositionZ);
        }
        console.log(missiles.length);
```

- Add missile message creates new mesh and pushes it into array that stores all bullets in game:

```
case "addMissile":
        var missile = new THREE.Mesh(new THREE.SphereGeometry(0.05, 2, 2), nev
THREE.MeshLambertMaterial({ color: 0xCC0000 }));
        missile.position.set(JInfo.shotPositionX,
                JInfo.shotPositionY,
                JInfo.shotPositionZ);
        missiles.push(missile)
         //missile.add(new THREE.AxisHelper());
         scene.add(missile);
        console.log("Missiles: " + missiles.length);
```

- When any of the missiles is outside of the scope of the game server will send the message to remove it:

```
case "removeMissile":
            scene.remove(missiles[0]);
            missiles.splice(0, 1);
            console.log(missiles.length);
```

9.1.2.User input handling

Client script is not processing user input itself. Instead it composes JSON messages and sends them to server. For example when user hits left mouse button script will catch the event and send the intention of the user to shoot to the server:

```
//Sending shots to the server
container.addEventListener('click', function (e) {
        //Senging message to server
        ws.send("{\"type\":\"shot\"}");
}, false);
```

9.1.3.Rendering World

The world of the HelWars game is a JavaScript file in JSON format that stores vertices for all objects in the world. This Scene has been generated using Blender. Once the model is loaded callback function set such properties as position, size and shading.

```
var loader = new THREE.JSONLoader();
loader.load("models/js/room.js", function doRoomModel(geometry) {
        var material = new THREE.MeshLambertMaterial({ color: 0x606060, morphTargets: true
vertexColors: THREE.FaceColors });
        var material = new THREE.MeshFaceMaterial({ color: 0x606060, morphTargets: true,
vertexColors: THREE.FaceColors });
        roomModel = new THREE.Mesh(geometry, material);
        roomModel.scale.set(2, 2, 2);
        roomModel.castShadow = true;
        roomModel.receiveShadow = true;
        roomModel.position.set(5, 0, 5);
        roomModelLoaded = true;
        if (playerModelLoaded && opponentModelLoaded && roomModelLoaded) {
                init();
                animate();
        }
});
```

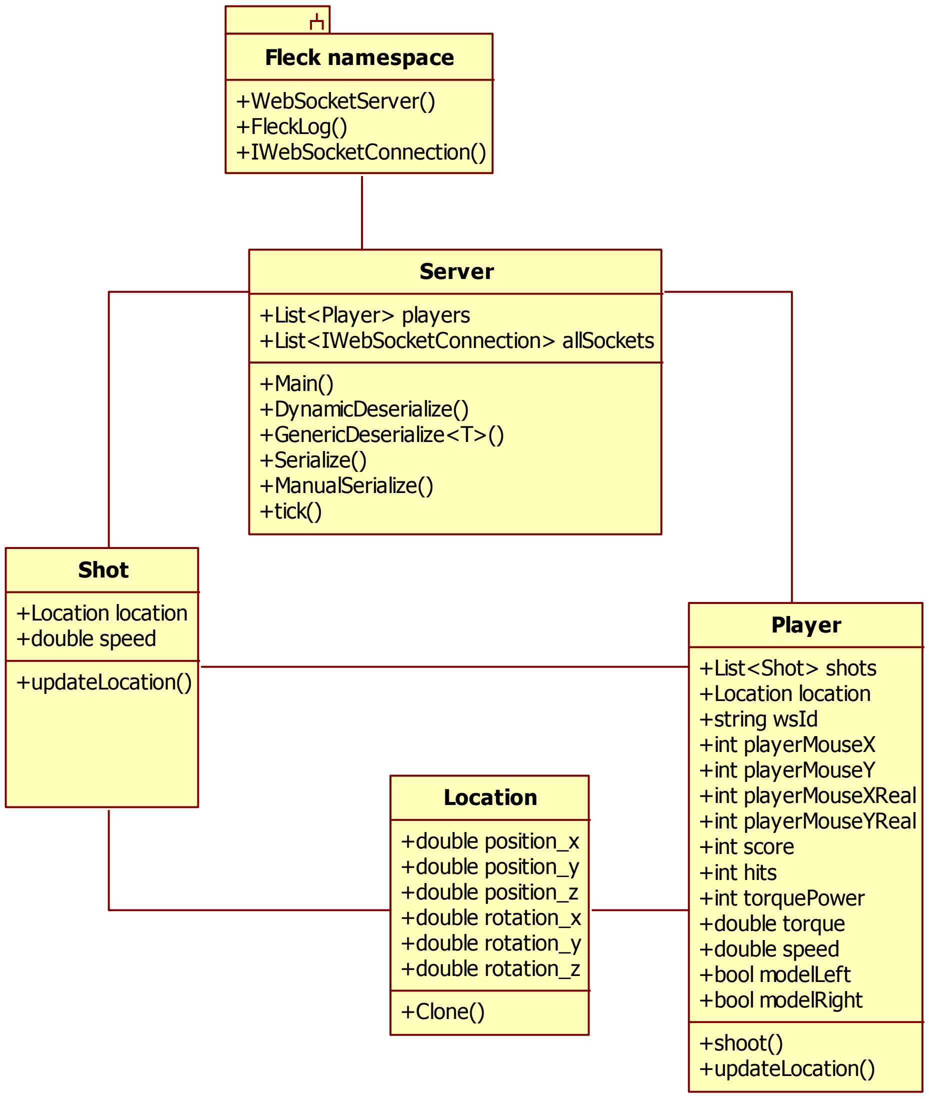
9.1.4.Loading and Rendering Models

Loading and rendering of the player models is achieved in almost the same way as room loading. Although it has one significant difference - models are animated. In order to animate the mesh it is required to set few additional properties. Morph targets and morph normal have to be enabled as well as morph computation has to be initiated:

```
for (var i = 0; i < geometry.materials.length; i++) {
        var material = geometry.materials[i];
        material.morphTargets = true;
        material.morphNormals = true;
        material.shading = THREE.SmoothShading;
}
geometry.computeMorphNormals();
```

9.2. Server-side Implementation

Server has been implemented in an Object-Oriented manner. Next figure shows how server classes are organised.



9.2.1.Game Entities

Games Entities namespace is holding 3 active classes in software. Those are as follows:

- Location

  Location Class is used for storing six variables of double type. Those are players' position and rotation in 3D space on axis x y z. Location Class has method ***clone ()*** used to copying Location to other objects with creating new object in memory instead of referencing old one.

- Shot

When Shot object is created it expects a Location in its constructor.

Server is sending Players current location, so the new shot automatically knows its direction of flying and position. Server is calling ***updateLocation ()*** method of each shot to change the shots location. Shot direction cannot change.

```
//updating shot' position
    public void updateLocation()
    {
        this.location.position_x += Math.Cos(this.location.rotation_y) * speed;
        this.location.position_z += Math.Sin(-this.location.rotation_y) * speed;
        this.location.position_y += Math.Sin(this.location.rotation_z) * speed;
    }
```

Shots' speed is hard coded and stored in ***speed*** variable. In future game development this variable could become changeable by player through collection of various bonuses in game.

- Player

  Player class is the most sophisticated among game entities.

  It stores the location for calculating Player position and rotation. Player object stores the list of shots which getting its own Location cloned every time player shoots.

  updateLocation () method is updating players' location and rotation smoothly though changing mouse position variable by 3 every frame depending on what is the real mouse location:

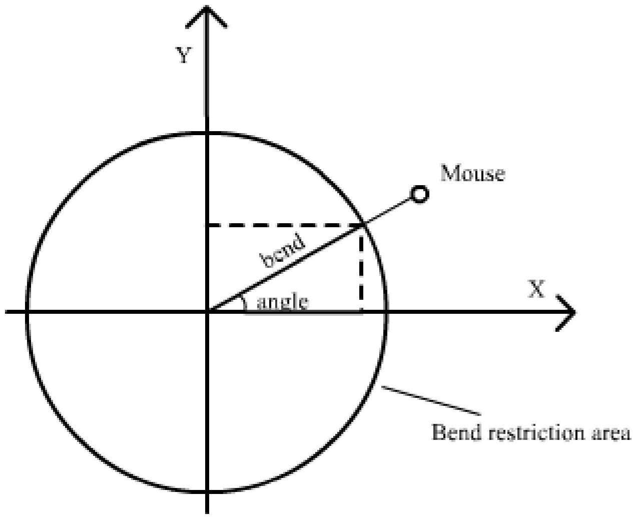
```
//smooth torque controll
        if (playerMouseX < playerMouseXReal - 3 && playerMouseX<30)
            playerMouseX += 1;
        else if (playerMouseX > playerMouseXReal + 3 && playerMouseX > -30)
            playerMouseX -= 1;
        if (playerMouseY < playerMouseYReal - 3 && playerMouseY < 30)
            playerMouseY += 1;
        else if (playerMouseY > playerMouseYReal + 3 && playerMouseY > -30)
            playerMouseY -= 1;
```

Mouse locations on server are restricted to 30 pixels from screen center point.

Depending on mouse location variables ***bend*** and ***angle*** are calculated. ***bend*** stores distance between the mouse and screen center point. ***angle*** stores angle of the bend.

```
        //bend is a distance between mouse and center screen (No more then 50)
        var bend = Math.Round(Math.Sqrt(playerMouseX * playerMouseX + playerMous
playerMouseY));
        //angle rounded to 2 decimal points
        var angle = Math.Round(Math.Atan2(playerMouseY, playerMouseX) * 100) / 100
this.location.rotation_y;
```

Next figure illustrates what the angle and bend are:



Those variables are used to calculate the X and Z rotation change of position of the model.

```
//rotation on Z axis
this.location.rotation_z =  - Math.Sin(this.location.rotation_y - angle) * Math.PI * (bend *

//speed
if (speed < bend / 500)
   speed += 0.001;
else if (speed > bend / 500)
   speed -= 0.001;
//movement
this.location.position_x -= sn * speed;
this.location.position_z -= cs * speed;
```

Rotation on Y axis is controlled by players’ intention to rotate left or right:

```
//rotating player
if (modelRight)
   this.location.rotation_y -= 0.02;
else if (modelLeft)
   this.location.rotation_y += 0.02;
```

Height changes smoothly

9.2.2.Handling messages

In order to process user input server is required to handle messages received over WebSocket communication with client. Similarly to client server needs to recognise 3 types of events:

- OnOpen

  This event is triggered when client initiates connection with the server. Several important things are happening at this stage. First of all a new Globally Unique Identifier (GUID) is created and gets assigned to the new socket connection. Program then checks if the server is full, and if it is the connection is getting closed and user receives message informing him that the server is full:

```
if (players.Count() < 2)
                {
{
          //server not full code here
}
else
{
          socket.Send("{\"type\":\"message\", \"message\":\"Sorry, Server is full.\"}");
          Console.WriteLine("Connection declined, Server is full!");
          socket.Close();
}
```

  If the server is not full server creates random location which is sent to the new Player created here as well:

```
if (players.Count() < 2)
{
             Console.WriteLine("New connection is open!");
            allSockets.Add(socket);

            //Random location for new player
            Location newPlayerLocation = new Location();
            newPlayerLocation.position_x = new Random().Next(-10, 10);
            newPlayerLocation.position_z = new Random().Next(-10, 10);
            newPlayerLocation.position_y = 2;

            newPlayerLocation.rotation_x = 0;
            newPlayerLocation.rotation_y = 0;
            newPlayerLocation.rotation_z = 0;

            players.Add(new Player(socket.WsId, newPlayerLocation));
            //sending ID to the client
            socket.Send("{\"type\":\"id\", \"wsId\":\"" + socket.WsId + "\"}");

            //let all other player know that there is a new player in game.
            allSockets.ForEach(s => s.Send("{\"type\":\"newOpponent\", \"wsId\":\"" + socket
"\"}"));
}
```

Server then sends two messages to the clients. First is message to the client who just connected to the server letting him know his GUID. Second message is to all players informing them that new player has successfully joined the game.

- OnClose

  When the client shuts down the connection, OnClose event fires on server. The callback handles the closing. In order to avoid memory leaks and also to free the place for the new connection on server code removes the player from memory. To avoid the scoring confusion code also removes all shots that belong to the player who just left the game:

```
socket.OnClose = () =>
{
        //Inform other players that player has left the game
        Console.WriteLine("Player " + socket.WsId + "  has left the game!");
        //Remove shots that belong to that player
        foreach (Shot s in players.Find(p => p.wsId == socket.WsId).shots)
        {
                Console.Write("Shot " + s.location.position_x.ToString() + " ");
        }
        Console.WriteLine("");
        //Remove player
        players.Remove(players.Find(p => p.wsId == socket.WsId));
        //Remove socket
        allSockets.Remove(socket);
};
```

- OnMessage

  This message is a custom JSON string that server receives from client. Server need to decode the message, check the message type and then handle message appropriately. First Deserialisation of the method occurs

```
socket.OnMessage = message =>
{
        //messages from clients handled here
        dynamic msg = DynamicDeserialize(message);
```

DynamicDeserialize method desterilises message into a dynamic Object that doesn’t have properties specified:

```
//converting JSON to Object
private static dynamic DynamicDeserialize(string json)
{
        var s = new System.Web.Script.Serialization.JavaScriptSerializer();
        dynamic obj = s.DeserializeObject(json);
        return obj;
}
```

Once this is complete it becomes possible to recognise what kind of message the client has sent. If the server message was about player moves intensions server will decompose it accordingly:

```
if (msg["type"] == "moves")
{
        Player tempPlayer = players.Find(p => p.wsId == socket.WsId);
        switch ((String)msg["moveType"])
        {
                case "modelLeftTry":
                players.Find(p => p.wsId == socket.WsId).modelLeft =
        Boolean.Parse(msg["value"]);
                break;
                case "modelRightTry":
                players.Find(p => p.wsId == socket.WsId).modelRight =
        Boolean.Parse(msg["value"]);
                break;
                case "torquePowerTry":
                players.Find(p => p.wsId == socket.WsId).torquePower = Int16.Parse(msg
                break;
                case "mouseMoveX":
                players.Find(p => p.wsId == socket.WsId).playerMouseXReal =
        Int16.Parse(msg["value"]);
                break;
                case "mouseMoveY":
                players.Find(p => p.wsId == socket.WsId).playerMouseYReal =
        Int16.Parse(msg["value"]);
                break;
        }
}
```

If the message type was “shots” server will create a new shot and assign it the player. It will also instruct all players to render additional missile:

```
else if (msg["type"] == "shot")
{
        Player tempPlayer = players.Find(p => p.wsId == socket.WsId);
        tempPlayer.shoot();
        allSockets.ForEach(s => s.Send("{\"type\":\"addMissile\", \"shotPositionX\":\"" +
tempPlayer.location.position_x + "\", \"shotPositionY\":\"" + tempPlayer.location.position
\"shotPositionZ\":\"" + tempPlayer.location.position_z + "\"}"));
}
```

9.2.3.Sending Messages

Server needs to send various messages to clients. Those are missile location, player locations, opponent location etc. In order to send complex data to client such as location of players this data needs to be converted to JSON format. This is achieved through ManualSerialisation method:

```
private static string ManualSerialize(Type type)
{
        string json = "";
        if (type == typeof(Player))
        {
                json = "{\"type\":\"playerLocations\", \"locations\":[";
                foreach (Player player in players)
                {
                        json += "{\"wsId\":\"" + player.wsId + "\",\"playerPositionX\":\"" +
                player.location.position_x + "\",\"playerPositionY\":\"" + player.location.position_
                + "\",\"playerPositionZ\":\"" + player.location.position_z +
                "\",\"playerRotationX\":\"" + player.location.rotation_x + "\",\"playerRotationY\":\
                + player.location.rotation_y + "\",\"playerRotationZ\":\"" + player.location.rotation
                + "\"},";
                }
                 json = json.TrimEnd(',');
                json += "]}";
        }
        else if(type == typeof(Shot))
        {
                //simple missile structure
                json = "{\"type\":\"missileLocations\", \"missileLocations\":[";
                foreach(Player player in players)
                {
                        foreach(Shot shot in player.shots)
                        {
                                json += "{\"shotPositionX\":\"" + shot.location.position_x +
"\",\"shotPositionY\":\"" + shot.location.position_y + "\",\"shotPositionZ\":\"" +
shot.location.position_z + "\",\"shotRotationX\":\"" + shot.location.rotation_x +
"\",\"shotRotationY\":\"" + shot.location.rotation_y + "\",\"shotRotationZ\":\"" +
shot.location.rotation_z + "\"},";
                        }
                }
                json = json.TrimEnd(',');
                json += "]}";
        }
        return json;
}
```

In order to ensure smooth gameplay server is required to send updated location to client at the minimum rate of 16 times per second (Which would result in 15 Frames per second render) This is achieved though tick method which is currently set to be called every 60 milliseconds:

```
private static void tick(object source, ElapsedEventArgs e)
{
        //updating players positions
        players.ForEach(p => p.updateLocation());

        //updating shots positions
        players.ForEach(p => p.shots.ForEach(s => s.updateLocation()));
        //updating clients
        allSockets.ForEach(s => s.Send(ManualSerialize(typeof(Player))));
        allSockets.ForEach(s => s.Send(ManualSerialize(typeof(Shot))));

}


Timer timer = new Timer(60);
timer.Elapsed += new ElapsedEventHandler(tick);
timer.Enabled = true;
```

9.2.4.Collision Detection

Collision detection in HelWars game is a simple parallelepiped box around helicopter models that collides with center of missile.

Below is the snipped of code that does the collision calculation. It also removes the missiles that already collided with objects in order to avoid the situation when a missile hits more than one object, or hits same object twice. Code also checks if the missile on collision belongs to the player that it hits in order to avoid self-hitting.

```
if(players.Count()>1)
            {
               List<int> removal0 = new List<int>();
               List<int> removal1 = new List<int>();
               for (int shot = 0; shot < players[0].shots.Count;shot++ )
               {
                  if (Math.Abs(players[0].shots[shot].location.position_x - 
players[1].location.position_x) < 1 &&
                     Math.Abs(players[0].shots[shot].location.position_y - 
players[1].location.position_y) < 1 &&
                     Math.Abs(players[0].shots[shot].location.position_z - 
players[1].location.position_z) < 1)
                  {
                     Console.WriteLine("Player " + players[0].wsId + "shot Player " + players[1]);
                     players[0].score++;
                     players[1].hits++;
                     allSockets.Where(socket => socket.WsId == 
players[0].wsId).FirstOrDefault().Send("{\"type\":\"message\", \"message\":\"You hit opponent(" + 
players[0].score + " times)\"}");
                     allSockets.Where(socket => socket.WsId == 
players[1].wsId).FirstOrDefault().Send("{\"type\":\"message\", \"message\":\"You got hit(" + 
players[1].hits + "times)\"}");
                     removal0.Add(shot);
                  }
               }
```

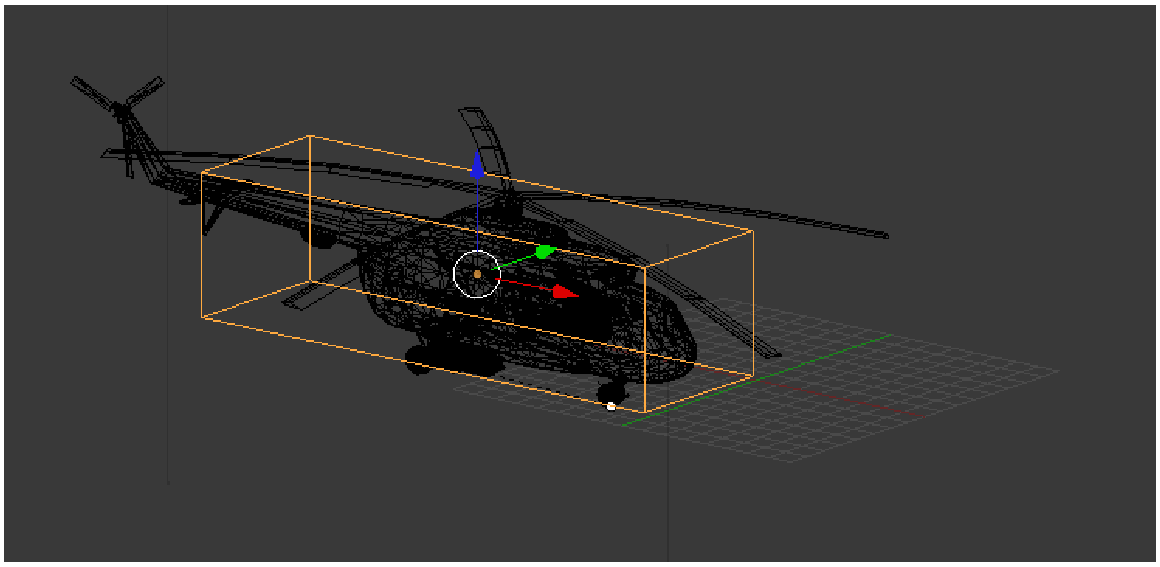
```
                for (int shot = 0; shot < players[1].shots.Count(); shot++)
                {
                    if (Math.Abs(players[1].shots[shot].location.position_x - 
players[0].location.position_x) < 1 &&
                        Math.Abs(players[1].shots[shot].location.position_y - 
players[0].location.position_y) < 1 &&
                        Math.Abs(players[1].shots[shot].location.position_z - 
players[0].location.position_z) < 1)
                    {
                        Console.WriteLine("Player " + players[1].wsId + "shot Player " + players[0]);
                        players[1].score++;
                        players[0].hits++;
                        allSockets.Where(socket => socket.WsId == 
players[1].wsId).FirstOrDefault().Send("{\"type\":\"message\", \"message\":\"You hit opponent(" + 
players[1].score + " times)\"}");
                        allSockets.Where(socket => socket.WsId == 
players[0].wsId).FirstOrDefault().Send("{\"type\":\"message\", \"message\":\"You got hit(" + 
players[0].hits + "times)\"}");
                        removal1.Add(shot);
                    }
                }


                //Remove the missile that hit the target
                foreach (int r in removal0)
                {
                    players[0].shots.RemoveAt(r);
                    allSockets.ForEach(s => s.Send("{\"type\":\"removeMissile\"}"));
                }
                foreach (int r in removal1)
                {
                    players[1].shots.RemoveAt(r);
                    allSockets.ForEach(s => s.Send("{\"type\":\"removeMissile\"}"));
                }
            }

            //updating clients
            allSockets.ForEach(s => s.Send(ManualSerialize(typeof(Player))));
            allSockets.ForEach(s => s.Send(ManualSerialize(typeof(Shot))));

        }
```

Next figure is representation of the parallelepiped box around helicopter. This is for demonstration purposes only. Actual box is not drawn in the game - calculations are complete through set of coordinates as explained in the above code.



10.Testing

10.1. Black-box testing

Test Case	Expected Result	Actual result
Leave application running to see if it will crash.	Application can be left running without crashing for any amount of time.	Once out of ten fifteen minute tests the application has crashed.
Fire a missile at an opponent	Application is to provide feedback, confirming the opponent got hit	20 out 20 missiles sent to target reported by the application as hits.
Use controls to displace helicopter model in 3D space	Model is to be moving around in space.	Model successfully reacts to mouse keyboard controls
Try to fly outside of the room	Helicopter model should stop moving once reached the wall, celling or floor.	Helicopter model cannot fly outside of the room
Check if helicopter propeller is rotating	Propeller is rotating	Propeller rotates as expected

11.Review

11.1. Critical

Although HelWars project has been successful it is surely missing some features. Let’s compare HelWars to Helicopter Wars (2010). Helicopter Wars game is a very simple 3D shooter that is unfortunately missing multiplayer functionality. Another downside of Helicopter Wars compared with HelWars is that it requires installation. HelWars on the opposite has multiplayer and does not require installation. Yet it still does not have few features that Helicopter Wars has. Those options in the game could greatly enhance gaming experience and should be considered for future development. One important downside on HelWars is that it does not provide any options to the player, such as selecting helicopter models or selecting world. HelWars also has only one kind of missiles. And even though right click has been disabled in browser window it has not been utilised for any other functionality. Other aspect to notice is that HelWars has a simple round mesh for missiles, while Helicopter Wars has nicely draw rockets and bullets. Helicopter Wars looks more like a finished product for the market while HelWars at its current state shall not be released. When comparing with Helicopter Wars, HelWars game has a nicer physics engine that looks more realistic. We cannot say this however if we will compare HelWars with Battlefield 2. Battlefield 2 helicopter mode benefits from extremely realistic physics and graphics that are impressive. This means HelWars engine still could benefit from those improvements.

Overall HelWars game development went smoothly and was successful (met the requirements). Nevertheless having all the experience now, certain things would have been done in a different way, if I was to write similar project again. First of all a deeper research at the chosen technology would have been done. All the limitations of the libraries explored and decision made whether to proceed with the current libraries or look for an alternative. May be a wider range of technologies considered at the first place. It is now possible to say that real user involvement could be very helpful during development process. I now realise that constant feedback from users is important during Rapid Application Development. Methodologies chosen for this project proved to be useful so this would not change for a similar project. For a larger project however different Software development life-cycle could be considered.

Important thing to acknowledge is the legal aspect of the project. During the project there were difficulties with finding suitable open-source 3D models. Designer skills could be helpful during the project to develop unique helicopter models. Since HelWars is a one person project some 3D modeling skill had to be developed. This activity did not have time allocated at the start of the project and caused delay.

I cannot claim the HelWars engine is revolution, but I can and I will commit further to this project with aim to make changes in the gaming industry. I hope HelWars project will develop into something bigger and will affect how people play the games. In ideal scenario I want to see major games development studios are switching to the web revealing the truly powerful capabilities of browser-based gaming. Being built with cutting-edge technologies this project has a great potential to a make a significant social impact.

11.2. Acquired knowledge

A lot can be told about the experience gained though the life-time of the HelWars project. It would be quite impossible task to mention all the valuable skills and knowledge acquired. Here is the list of major skills developed and material learned that I did not have at the start of the project:

- Technology
  HelWars project was very technology oriented. Its main purpose was exploring HTML5 technology features and trying to combine them into a new innovative game engine. Due to this reason main expertise acquired was technology oriented. Through building complex solutions required for the game to run I became proficient in computer vision, 3D modeling. I understood client-server communication principles and various options available for designing architecture for this type of communications.
- Project Management

DSDM project management framework has been explored and utilised in the project through which project management knowledge has been acquired.

- Software Development
  Rapid Application Development (RAD) has been used throughout the life-time of the project. Game server and front-end was developed in small cycles introducing prototypes on every iteration.

12.Future Enhancement

Although current release of the game is fully functional and meets the initial requirements (see section 7.3) it is still missing a number of features that could greatly enhance gaming experience.

- Remote server
  Currently server is running as local console application. Although it is enough to run the multiplayer on a local network there are disadvantages in this approach. First problem is that a firewall on the computer running server has to be configured to allow local network connections from other computers. Secondly it is not possible to bring connection from around the World Wide Web. This constraint makes the game limited to the local server. With the release of the .NET version 4.5 this constraint will be easy to overcome. New version of .NET will contain native support (built-in libraries) for WebSockets. This makes it much easier to bring the WebSocket server to a WCF service. This will have its own limitation documented here http://forums.asp.net/t/1732788.aspx/1. WebSockets in .NET 4.5 unfortunately will only be supported on Windows 8 OS. This potentially could be a problem, as might take long time for hosting providers to update servers.
- Register and Login

Registration and Login feature could increase the appeal of the game as this allows users to store their result in the database on server. It will also enable the statistics and top players list features. Such improvement might attract more users and will make players tempted to come back to play the game to improve their scores.

- Server capacity

  Current release of the game has a player capacity limited to two at the same time. This limitation was brought for the reason of testing and development. This However is not suitable for an application on the web. Server capacity limitation needs to be disabled before real deployment. This however pulls out more requirements. For example the gaming world needs to be increased. Current environment is a student room which is quite small and filled with reasonable amount of obstacles. If there will be many more helicopter models loaded the world will become overcrowded.

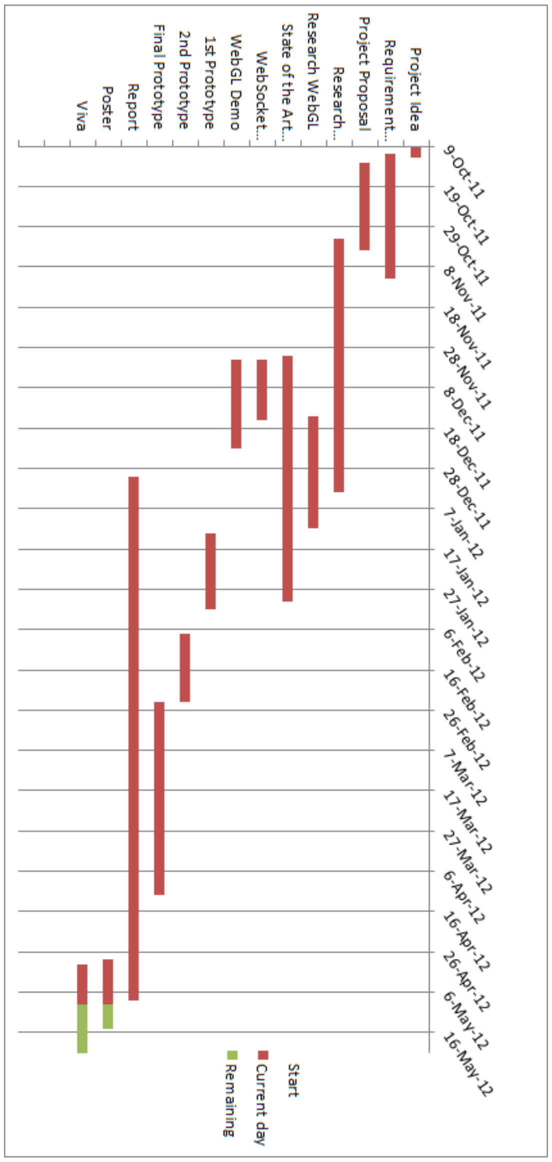
Listed improvement will greatly enrich gaming experience and could be vital on the way of the game development. It is however important to mention that those are not the only possible improvements that can be brought to the game. I acknowledge that HelWars game will also benefit from other features such as improved controls, enhanced physics engine, model optimisation and many others.

13.Conclusion

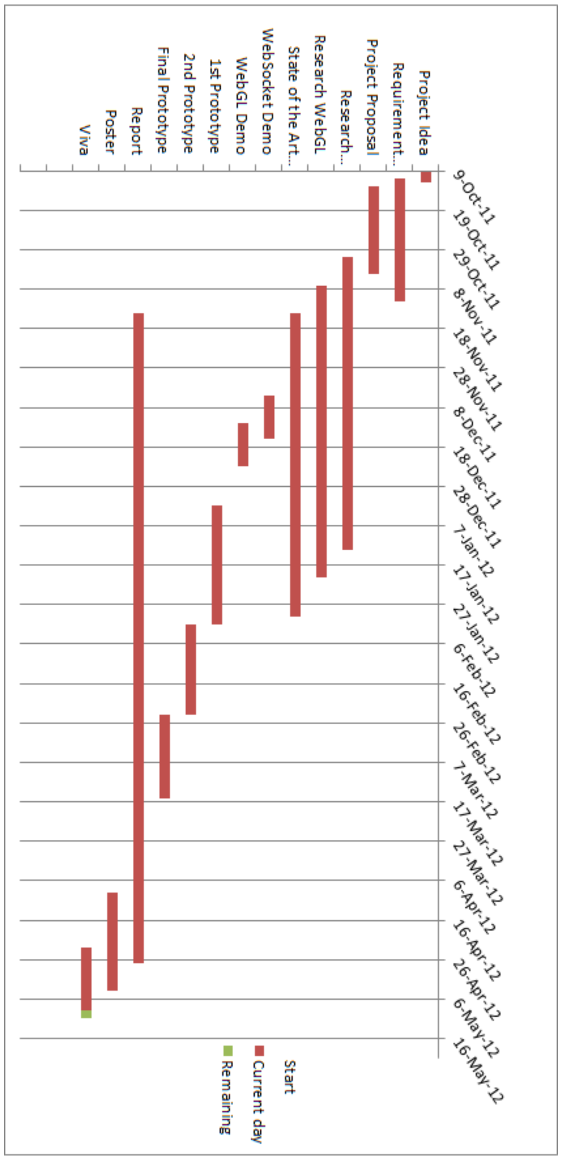
Looking at the initial specification of the project it would be possible to conclude that the project has been successful. All major requirements have been effectively implemented. The game is fully functional and allows multiplayer. Realistic rendering has been implemented. Simple collision detection has been applied. HTML 5 WebSockets and WebGL have been successfully combined into a robust game engine. Developed application could be used as a starting point in development of many kinds of games both multiplayer and singles player. Various 3D and 2D games can be built on top of this engine.

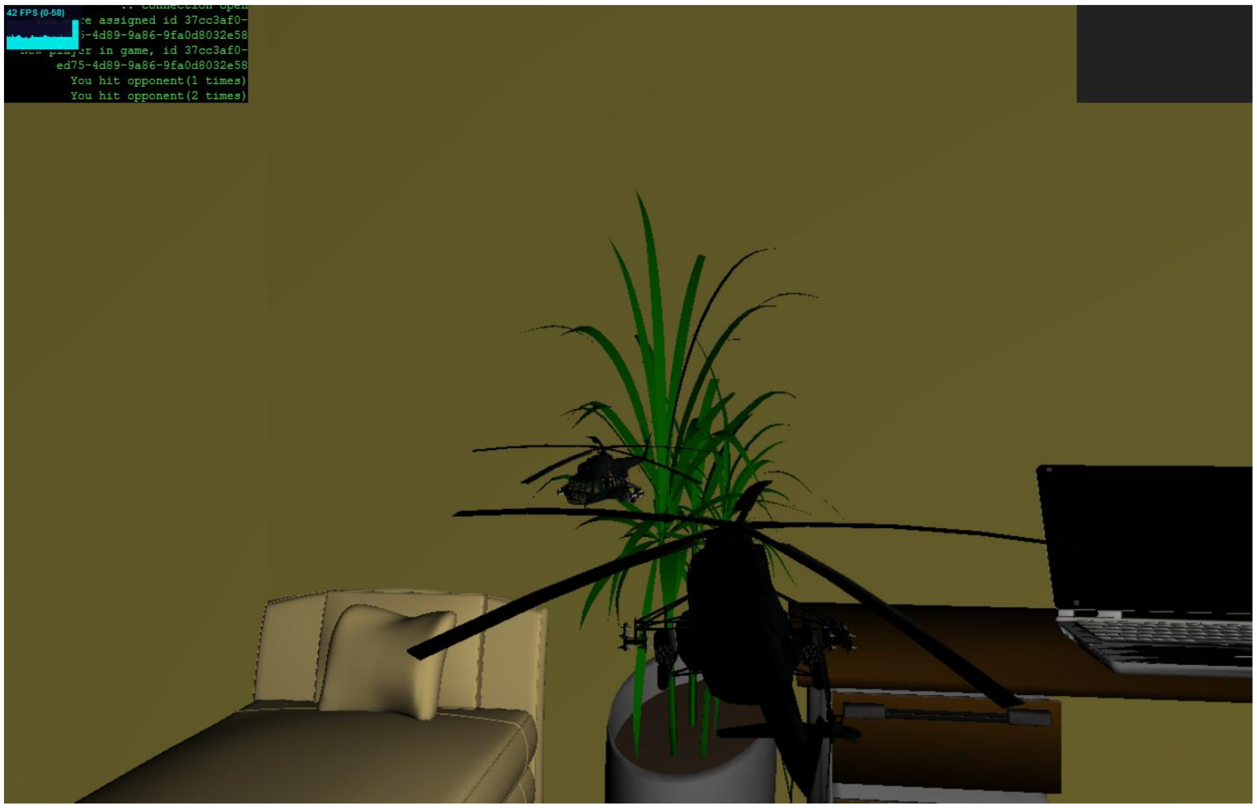
Certain circumstances affected the timespan of the project however project was still delivered with a small delay. Two charts below are the comparison of the planned schedule and actual time breakdown of the project. It can be seen that actual activities were performed almost as planned.

Actual



Initial





```
        .. connection open
     ce assigned id 37cc3af0-
     5-4d89-9a86-9fa0d8032e58
  New player in game, id 37cc3af0-
     ed75-4d89-9a86-9fa0d8032e58
       You hit opponent(1 times)
       You hit opponent(2 times)
```

References

Eric Bidelman (2011), Using Cross-domain images in WebGL and Chrome 13, *Available at:* http://blog.chromium.org/2011/07/using-cross-domain-images-in-webgl-and.html *[Accessed on 14-December-2011]*

Khronos Group, WebGL Security Whitepaper, *Available at:* http://www.khronos.org/webgl/security/ *[Accessed on 20- January -2012]*

Khronos Releases Final WebGL 1.0 Specification, *Available at* http://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification *[Accessed on 8-January-2012]*

Matthew MacDonald; Adam Freeman; Mario Szpuszta (2010), Pro ASP.NET 4 in C# 2010, 4th Edition

Microsoft ASP.NET Team (2011), What's New in ASP.NET 4.5 and Visual Web Developer 11 Developer Preview, *Available at* http://www.asp.net/vnext/overview/whitepapers/whats-new *[Accessed on 29-December-2011]*

Nick Yee. CyberPsychology & Behavior. December 2006, 9(6): 772-775. doi:10.1089/cpb.2006.9.772.

Stefan Schackow ; Paul Batum (2011), Building real-time web apps with WebSockets using IIS, ASP.NET and WCF *Available at:* http://channel9.msdn.com/Events/BUILD/BUILD2011/SAC-807T *[accessed 15 January 2012]*

Walters, Rob; Coles, Michael; Dewson, Robin; Ferracchiati, Fabio Claudio; Narkiewicz, Jan; Rae, Robert (2008), Accelerated SQL Server 2008

X-Wing by Outsite OfSocienty (2011), *Available at:* http://oos.moxiecode.com/js_webgl/xwing/index.html*[accessed 24 January 2012]*

Helicopter Wars (2010) *Available at: http://www.myplaycity.com/helicopter_wars/ [accessed 31 October 2011]*

Rawkets (2011) *Available at:* http://rawkets.com/ *[accessed 12 October 2011]*

Chrome Experiments (2011) Available at http://www.chromeexperiments.com/webgl *[accessed 7 October 2011]*

The WebSocket API (2011) *Available at:* http://dev.w3.org/html5/websockets/ *[accessed 7 November 2011]*

Problem with self hosted WCF WebSocket *Available at:* http://forums.asp.net/t/1732788.aspx/1 *[accessed 15 February 2012]*

JigLibJS, rigid body physics for the GL generation *Available at:* http://www.jiglibjs.org/ *[accessed 21April 2012]*

Archive 3D, free 3D models *Available at:* http://archive3d.net/ *[accessed 15 February 2012]*

WampServer, web development platform on Windows models *Available at:* http://www.wampserver.com/en/ *[accessed 12 November 2011]*

Appendices

Deployment Instructions

This project comes with a CD that has the files required for a local deployment. There are few pre-requisites to run the project locally:

- Visual Studio 2010 or later
- WampServer
- Browser with full HTML 5 support (at the moment of writing this dissertation such browser are Google Chrome and Mozilla Firefox. See section 7.1.2 for details)

Above software has to be installed on the machine that is going to run the project.

- Place the HelWars folder (located inside Client folder) from CD to www directory of the WampServer. By default this is "C:\wamp\www",
- In file js/WebGL.js on line 50 change the IP address to IP of the machine the server is going to run on.
- Launch WampServer
  - You might need to change the apache default port:
    - Click on WampServer icon->Apache->httpd.config
    - Change port 80 to unused port, e.g. 8080
- Launch HelWars.sln file (From Server/HelWars) folder run the server without debugging (Ctrl+F5)
- In browser (Google Chrome recommended) open the localhost location of the client application. By default: http://localhost:8080/HelWars
- To enable local network multiplayer you need to disable firewall for home networks.
- Second player will need to open game using server IP address instead of local host. E.g.: http://192.168.0.5:8080/HelWars

Project Proposal

Contents

HelWars - Web-based 3D Game

Project description

Computer Technologies are developing with enormous speed and latest trends show that more and more activities are brought to the web, including social activities, gaming, shops and many others. Games were in the web for years now but the experience was never nearly as good as on personal computers or dedicated game consoles. Now with the all modern web technologies available it is now the time to reveal the full power of browser based gaming experience.

Purpose and Objectives

This project is to develop a new trend in web based gaming experience. 3D gaming is rarely available on the web, and it is almost an impossible task to find a web based multiplayer game.

Main Objective of this project is to combine two cutting edge technologies to produce an outstanding experience in the web that has not yet been available. A Browser based 3D game is something unusual by itself. But in this project it's going to be enhanced with multiplayer which would make it absolutely exclusive.

It is also important that the end product will not be relying on any plugins, browser add-ons or 3rd party players. The goal is to create an application that can run using only the web browser on client side. The only requisite is that user uses modern browser with full support to HTML5 including WebGL.

Scenario

Scenario for helicopter wars may seem to be not unique – there are games like Helicopter Wars (2010). However this particular game has a little twist. The helicopters are going to be placed in the environment of a student room, with various obstacles common for studio flat. Plastic helicopter models flying in a student room, trying to shoot each other and bonus items like stationaries on the desk, books, mobile phones, empty beer bottles.

Justification of the project

Market Feasibility

HelWars game is oriented on students because of its environment - student's room. There are almost 2 million students in UK (BBC 2009) which makes 3.15 percent of UK population. It is a huge market, and this is only the Higher education students.

Technical Feasibility

One thing to consider when creating something completely new using cutting edge technologies is to check whether this can be built at all. The easiest way to do so is to look at what is currently available. So let's look at some demos.

There are number of WebSockets demos available on the web. Rawkets (online 2011) and RUMPETROLL (online 2011) are 2 WebSockets games that match our needs quite closely. They both are using web sockets to create a multiplayer game. This would mean that WebSockets multiplayer game is achievable.

There are many WebGL demos available at Chrome experiments (2011) which prove the 3D in browser is achievable. Another interesting 3D demo is available of rendering Quake 3 maps with WebGL (TojiCode 2010)

All above demos prove it is possible to make multiplayer games and it is possible to make a 3D game. However what about 3D multiplayer game? On the web there seem to be no such thing available. Although Google claims they have created a browser based Quake (Quake II GWT Port Google Code 2010), the game itself however is only available through building on local machine. There is a video of the gameplay available online (GWT Quake YouTube 2010)

Importing models to WebGL can be a bit tricky task. But with the Help of Three.js library we can achieve that. The problem is that Three.js can only understand model mesh in JSON format. Luckily Three.js library provides an add-on for Blender that can help exporting most popular 3D model formats like .OBJ . SVG .3DS. to JSON format .JS. The only obstacle that can be foreseen right now is converting .MD2 models to JSON mesh. It is not vital however as there are other options available.

Another option is creating models in open source 3D modeling SDK Blender. Models created in blender can be easily exported to JSON mesh file.

Timescale feasibility

It might seem very challenging to complete this project by the deadline because of its complexity.

Hence this project can be broken down into 2 parts:

3. Browser Based 3D game.
4. Use of WebSocket for creation a non-3D multiplayer game

Development team (consisted of one person) already has experience in implementing computer vision applications although using different technologies – OSG and OpenGL. This should ensure fewer problems during creation of the first part of the project.

Development team does not currently have skills using WebSocket and there are not so many books available. However the range of options for WebSockets server ensures plenty of online documentation. WebSocket API (2011) also became recently available.

Another issue concerning time available is the amount of time needed for creating 3D models. This can be solved by using models freely obtainable on the web.

Technology

Server side

The server needs to be implement that has full support for Web Sockets. The decision on server is yet to be made. There is number of options available for WebSockets server including those built in Java, Python and Node.js. Here is the list of options to consider:

1. Socket IO-Node (Node.js)
2. jWebSocket (Java)
3. Kaazing WebSocket Gateway (Java)
4. Netty (Java)
5. web socket ruby (Ruby)
6. websocket (Python)

7. websocketify (Python, Node.js, C)

Client Side

There are two options for rendering the 3D through the browser, the Canvas and WebGL elements. Both are part of the HTML5 and both having advantages over each other. On Curved Cat Games there is a Post about Canvas and WebGL (2011). Author concludes multiple advantages of Canvas and WebGL over each other. According to the post Canvas is supported by most of the modern browsers, WebGL is on the other side currently is fully supported only by Google Chrome and Firefox. Although it is not as popular yet it has a number of advantages which lack in Canvas. It is faster – WebGL is able to use users Graphic Card to boost rendering of the 3D which makes a huge difference in performance and offloads the main processor.

Even though the WebGL seem to be a bit more complicated, it is a much better option for HelWars project as performance is more crucial then complexity.

Methodology

What's available?

There are multiple methodologies available however far not all of them suitable for HelWars project. Some most popular models can be considered Linear and Iterative. Let consider them. Linear Waterfall model often called classic Lifecycle model according to Roger Pressman (2001) requires to state all requirements clearly long before implementation stage, and will provide a first working model very late in the project time-span. This is not suitable for HelWars project as early prototype will need to be demonstrated. In the case of HelWars we need to employ a shorter development cycle. Among the iterative and incremental models there are Agile and RAD.

Agile vs. RAD

Considering Simon Evans' (2006) principles on differences between Agile and Rad we can conclude that Agile development doesn't allow prototypes but enforces breaking down the project into features and then incrementally adding them. Agile development lifecycle requires a team inclusive of testers, analysts and UX specialists, while Rapid Application Development is able to produce early prototypes by just having technical team members.

Decision

Considering the fact that HelWars is a one-person project, some principles of Agile development are impossible to incorporate. For example the It is not possible to make regular meetings. It’s also impossible to make dedicated roles for Testing, UX and Analysis. So because of those constraints RAD development is making much more sense.

Risks

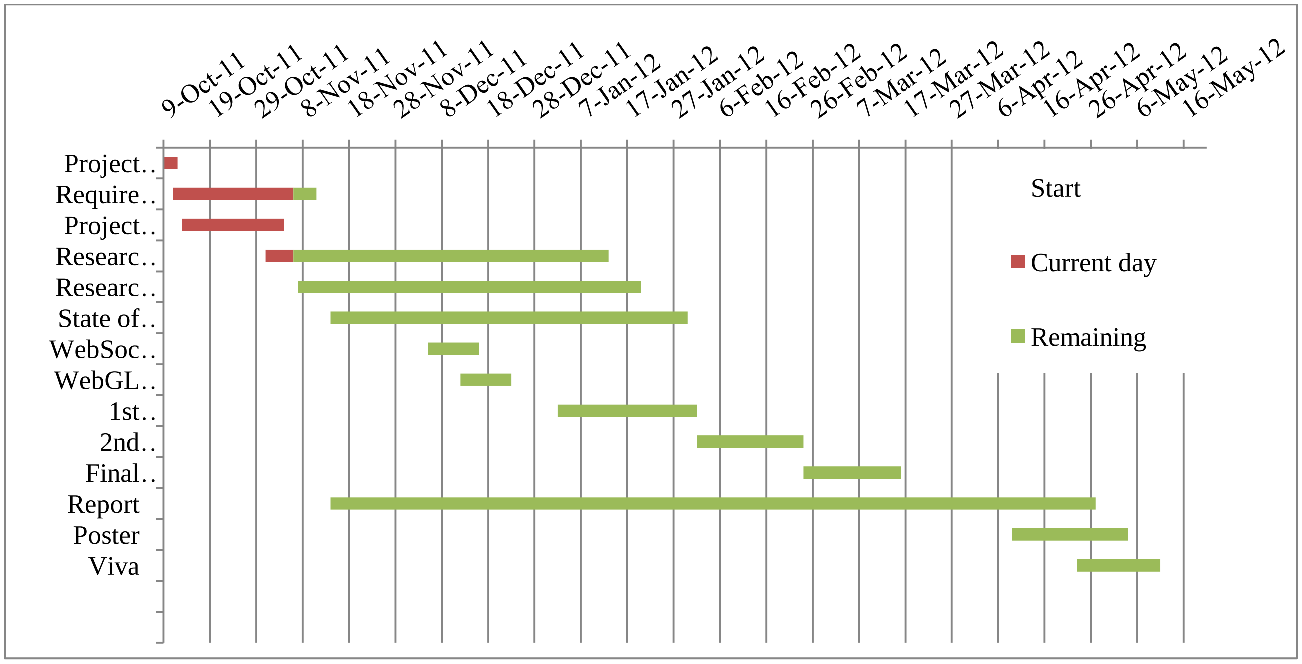
There are several risks concerned with this project. Main ones are: Would it be enough time to complete project? Would it be enough documentation on WebSockets to achieve multiplayer? There is a hardware risk: Would the server respond time be sufficient to handle real-time multiplayer game? Unfortunately it is hardly possible to answer those questions so early in the project.

Activities and implementation timeline

Activities and timeline for the project are as follows:

ID	Task	Start	End	Duration
1	Project Idea	9-Oct-11	12-Oct-11	3
2	Requirement Gathering	11-Oct-11	11-Nov-11	31
3	Project Proposal	13-Oct-11	4-Nov-11	22
4	Research WebSockets	31-Oct-11	13-Jan-12	74
5	Research WebGL	7-Nov-11	20-Jan-12	74
6	State of the Art Report	14-Nov-11	30-Jan-12	77
7	WebSocket Demo	5-Dec-11	16-Dec-11	11
8	WebGL Demo	12-Dec-11	23-Dec-11	11
9	1st Prototype	2-Jan-12	1-Feb-12	30
10	2nd Prototype	1-Feb-12	24-Feb-12	23
11	Final Prototype	24-Feb-12	16-Mar-12	21
12	Report	14-Nov-11	27-Apr-12	165
13	Poster	9-Apr-12	4-May-12	25
14	Viva	23-Apr-12	11-May-12	18

The above can be represented in Gant chart:



References

Helicopter Wars (2010) *Available at: http://www.myplaycity.com/helicopter_wars/ [accessed 31 October 2011]*

BBC 2009 Available at: http://news.bbc.co.uk/1/hi/education/7859034.stm [27 October 2011]

Rawkets (2011) Available at: http://rawkets.com/ [accessed 12 October 2011]

RUMETROLL (2011) Available at: http://rumpetroll.com/ [accessed 12 October 2011]

Chrome Experiments (2011) Available at http://www.chromeexperiments.com/webgl [accessed 7 October 2011]

Rendering Quake 3 maps with WebGL (2010) Available at http://media.tojicode.com/q3bsp/ [accessed 7 October 2011]

Quake II GWT Port Google Code (2010) Available at: http://code.google.com/p/quake2-gwt-port/ [accessed 14 October 2011]

GWT Quake YouTube (2010) Available at: http://www.youtube.com/watch?v=XhMN0wlITLk [accessed 14 October 2011]

The WebSocket API (2011) Available at: http://dev.w3.org/html5/websockets/ [accessed 7 November 2011]

Curved Cat Games, Html5 Canvas vs. WebGL (2011) Available at: http://curvedcatgames.wordpress.com/2011/08/27/html5-canvas-vs-webgl/ [accessed 7 November 2011]

Roger S. Pressman (2001) Software Engineering: A Practitioner's Approach 5th Edition

Simon Evans (2006) 10 Reasons why Agile is not Rapid Application Development (RAD) Available at: http://consultingblogs.emc.com/simonevans/archive/2006/04/18/10-Reasons-why-Agile-is-not-Rapid-Application-Development-_2800_RAD_2900_.aspx [accessed 12 October 2011]

State-of-the-Art – first submission

Contents

State-of-the-Art-Review

Demand and requirements

Online Multiplayer games were around the many years including such well-known MMOPRGs (Massively-Multiplayer Online Role Playing Games) as Ultima Online (released 1997) and much newer, more graphically advanced, World of Warcraft and Lineage2, popular first-person shooters games such as Counter-Strike, Battlefield and Call of Duty, and of course a whole lot of browser based online games.

So what makes player to spend days and night playing them? A survey completed by Nick Yee (2006) suggests that typical online gamer is 26 years old and spends on average 22 hours. In his study, “Motivations for Play in Online Games” he subdivided player motivation on three categories called components. Those Components are as follow:

- ❖ *Achievement component*
  - ➢ *Advancement—The desire to gain power, progress rapidly, and accumulate in-game symbols of wealth or status*
  - ➢ *Mechanics—Having an interest in analyzing the underlying rules and system in order to optimize character performance*
  - ➢ *Competition—The desire to challenge and compete with others*
- ❖ *Social component*
  - ➢ *Socializing—Having an interest in helping and chatting with other players*
  - ➢ *Relationship—The desire to form long-term meaningful relationships with others*
  - ➢ *Teamwork—Deriving satisfaction from being part of a group effort*
- ❖ *Immersion component*
  - ➢ *Discovery—Finding and knowing things that most other players don’t know about*
  - ➢ *Role-Playing—Creating a persona with a background story and interacting with other players to create an improvised story*
  - ➢ *Customization—Having an interest in customizing the appearance of their character*

Nick Yee (2006)

Agreeing to this we can conclude that multiplayer online game needs to have three factors. It needs to be social: Gamers should be able to communicate to each other and being able to unite in groups for achieving goals. There must be a progress line in the game so player can advance their avatars over the time. This also adds reward for gamers playing the game for longer time. Similarly people should be rewarded by discovering new bonuses in the game as they progress.

System Components

Graphics Renderer (WebGL)

One of the major browser based 3D game component is its renderer. Renderer is a rather important part of the game as it has influence on the performance: level of details and rendering speed. WebGL is there renderer chosen for this project. According to Khronos Group (2011)“WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0, exposed through the HTML5 Canvas element as Document Object Model interfaces”. Khronos Group is a consortium that develops standards and API specifications for rich media playback including WebGL.

Graphics Library (Three.js)

In order to simplify the development of the game a lightweight 3D library can be used. There is a library that suits our needs. It is called Three.js. Three.js library is a JavaScript 3D Engine which simplicity allows building 3D apps in a very robust way. This engine can be rendered using WebGL. Three.js library is widely used on Chrome Experiments, for example on brilliant demo called "Ginger" Facial Rigging by David Steele at http://stickmanventures.com.

Server Communication (WebSockets)

WebSockets protocol was developed as a part of the HTML5 specification. WebSocket bring a whole new trend in client-server interaction by introducing a bi-directional communication. Before WebSockets the way to communicate to server without reloading webpage was AJAX (Asynchronous JavaScript and XML), however the problem with this communication is that server was unable to send data to client without client triggering communication. Other workaround for this problem was Comet model also called AJAX Push. The problem with those two technologies is well defined by Nikolai Qveflander (2010). “Technologies that enable pushing of data from a server to a subscribing client are not using true asynchronous communication. Instead they emulate this using long polling where the client polls the server for data. If no data exists for the client, the server does not immediately respond but rather waits for data to be available and then sends it to the client. This technique of delayed responses relies on always having an outstanding client request that the server can respond to. What might look like a server initiated communication actually relies on the client making requests to the server." So what we need is a true asynchronous bi-directional communication and HTML5 is able to provide this for us.

Front-end

Front-end of the HelWars project is going to be implemented in HTML5 and JavaScript. To simplify the development HelWars project is going to incorporate various open source JavaScript Libraries such as jQuery, jQuery UI and Three.js

Back-end

ASP.NET is used for building the back end of this project for numerous reasons, firstly because Visual Studio 2010 provides many tools for building web applications. Anything from creating database to member authentication is done easy with Visual Studio 2010. The other reason for choosing ASP.NET is a new version of .NET is being released this year. In this new version, 4.5, framework will provide native support for WebSockets, Microsoft ASP.NET Team (2011)

Database

Since back-end of the application is planned to be implemented in .NET framework it does make sense to use a database from same technology provider. As .NET and SQL Server are both developed by Microsoft they ought to work together well. Since our project can be treated is minor we can use a free lightweight version of this database – SQL Server Express. Newer version of database is available from http://www.microsoft.com/sqlserver. However the 2012 version is not a final release yet, but the release candidate. According to sqlexpress blog at MSDN (2011) 2012 version was released late November 2011, which makes its stability hardly predictable. For this sole reason we are going to use SQL Server 2008 R2 which has plenty of documentation in such books as Accelerated SQL Server 2008 by Robert E. Walters et al (2008) and MSDN libraries.

Review of existing systems

WebGL Games

Wide range of WebGL games and application are presented in Chrome Experiments. However most of them do not bring high level of interaction with user. Those applications are experimental and main interaction is input from user of various characteristics like level of details and speed of the animation. X-Wing game by OutsideOfSociety (2011) offers best practices of a 3D game, being very interactive and requiring a lot of concentration and action from user yet being very simple. Not much physics however is involved in this game.

WebSockets Games

Rawkets is 2D online multiplayer space shooter game build using HTML5, WebSockets, and Node.js by Rob Hawkes, Technical Evangelist at Mozilla. This experimental game proves that WebSockets can allow smooth bi-directional communication between server and client. HelWars project is going to be using WebSockets as well however as back end being built in ASP.NET it makes sense to use .NET implementation of WebSockets instead of Node.js. Stefan Schackow and Paul Batum (2011) describe how easy it will be to build a WebSockets application in .NET4.5 when it will be released later in 2012. However .NET4.5 is unavailable at the moment hence we need non-native implementation of WebSockets on .NET such as project Fleck by Jason Staten (2011) which is freely available via Github.

Helicopter Games

Helicopter control is available in many games such as Battlefield and Call OF Duty, however in those games helicopter control is just a small bit of the game. Let's look at the Helicopter Wars (2010) game that is all about helicopters. The game is very easy to control, there is no learning curve to be able to control the helicopter. By achieving the ease of controlling helicopter developers had sacrificed on detail of physics. A game dedicated to a very narrow interest should implement realistic physics.

Other Areas

Security

There are many rumors on the web regarding security matters of the WebGL. The main issues mentioned in WebGL security issues are Denial of Service and Cross-Origin

Media. Those issues are explained in details in WebGL security whitepaper by Khronos Group (2011).

Denial of Service

Denial of Service is a scenario when Graphics Processing Unit is loaded with a large mesh. The computational cost for such operations are high, hence GPU takes a lot of time to process given operation and is unable to respond to other requests. Since some components of the operating system require some job done by GPU system appears to be frozen and may crash. Microsoft starting from Windows Vista has overcame this issue by setting the time limited for which GPU can work on one task.

Cross-Origin Media

Cross-Origin Media was a security issue brought from original WebGL specification. Shaders could have been used to indirectly gather the contents of textures uploaded to the GPU. “For security reasons, JavaScript code on a web page is not allowed to read the pixels of an image or video which comes from a different site.” Khronos (2011), however as WebGL can directly access GPU it was possible to steal images and videos from other sites. Eric Bidelman (2011) states at Chromium Blog that both Google Chrome starting from version 13 and Firefox from version 5 will not be allowing cross-domain media on WebGL.

Authentication

ASP.NET has a build-in support for authentication and Membership. Visual studio 2010 is able to automatically create a database for users, Mathew MacDonald et al (2010)

Conclusion

After reviewing available literature and web articles as well as existing games and web application we can conclude that project HelWars is achievable within given time frame. Current technologies suffice the requirements however is will be easier to build the multiplayer game when .NET 4.5 framework will be released with native support for WebSockets.

References

Eric Bidelman (2011), Using Cross-domain images in WebGL and Chrome 13, *Available at:* http://blog.chromium.org/2011/07/using-cross-domain-images-in-webgl-and.html *[Accessed on 14-December-2011]*

Khronos Group, WebGL Security Whitepaper, *Available at:* http://www.khronos.org/webgl/security/ *[Accessed on 20- January -2012]*

Khronos Releases Final WebGL 1.0 Specification, *Available at* http://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification *[Accessed on 8-January-2012]*

Matthew MacDonald; Adam Freeman; Mario Szpuszta (2010), Pro ASP.NET 4 in C# 2010, 4th Edition

Microsoft ASP.NET Team (2011), What's New in ASP.NET 4.5 and Visual Web Developer 11 Developer Preview, *Available at* http://www.asp.net/vnext/overview/whitepapers/whats-new *[Accessed on 29-December-2011]*

Nick Yee. CyberPsychology & Behavior. December 2006, 9(6): 772-775. doi:10.1089/cpb.2006.9.772.

Stefan Schackow ; Paul Batum (2011), Building real-time web apps with WebSockets using IIS, ASP.NET and WCF *Available at:* http://channel9.msdn.com/Events/BUILD/BUILD2011/SAC-807T *[accessed 15 January 2012]*

Walters, Rob; Coles, Michael; Dewson, Robin; Ferracchiati, Fabio Claudio; Narkiewicz, Jan; Rae, Robert (2008), Accelerated SQL Server 2008

X-Wing by Outsite OfSocienty (2011), *Available at:* http://oos.moxiecode.com/js_webgl/xwing/index.html*[accessed 24 January 2012]*

Helicopter Wars (2010) *Available at: http://www.myplaycity.com/helicopter_wars/ [accessed 31 October 2011]*

Rawkets (2011) *Available at:* http://rawkets.com/ *[accessed 12 October 2011]*

Chrome Experiments (2011) Available at http://www.chromeexperiments.com/webgl *[accessed 7 October 2011]*

The WebSocket API (2011) *Available at:* http://dev.w3.org/html5/websockets/ *[accessed 7 November 2011]*