Project Hellknöw

Group 3

Henrik Sandström

Jonas Lindmark

Carl-Fredrik Sundlöf

Tim Hao Li

Table of Contents

1 Preface

1.1 Expected readership

This document is written for the stakeholders of this project and includes the following people:

- The system end-users
- The developers

1.2 Version history

Version:	*Date:*	*Rational*	*Summary of changes*
1.0	2008-01-02	Final version	Corrected minor mistakes.
0.5	2007-12-28	Second draft	Added UCs, corrected POD, etc.
0.1	2007-12-20	First draft	

2 Introduction

2.1 Who are the users and what problem does the system solve for them?

Hellknöw is going to be a 2D shooter game that is mainly focused on multiplayer gaming. The main difference between our game and the games that are currently on the market is a so called Fog of War feature. Fog of War is a feature that is most commonly used in strategy games to hide parts of the course from the player. In our game the player will have a visual field that he can perceive, and the rest of the map will be blurred out or hidden in some way. This is revolutionary in the 2D-shooter game because usually the player can see things all around the avatar on the screen, which is not realistic since one would not be able to see what is behind if one does not turn its head. One of the successful aspects of the 3D-shooter genre is the realness in perception. Often times in a 3D-shooter game, players slaughter other players from behind or above. Things like that happens only when the agent in the game sees only what a human is capable of seeing. And this attribute makes the shooter game much more exciting and challenging. Therefore, our main goal is to implement the property of Fog of War in order to achieve this purpose.

Our main target audiences are gamers, people who play a lot of games, in the age range of 14 to 27. The users should have previous experience with similar types of games, with the ability to maneuver proficiently with a mouse and keyboard simultaneously. However, this will not be a necessary requirement. A person with less experience with the computers might still enjoy the game, and become more skilled as he or she plays the game. The game should also offer something new and inventive to the 2D shooter genre, so it will appeal to gamers who enjoy 2D shooters but are looking for something new.

The game will help with the problem of boredom and help the user to relax. The aim is to keep the user entertained for a moderate amount of time, while the user is able to interact with friends. It allows friends to spend some quality time together, while doing something that they (hopefully) enjoy.

The game will be a alternative to the general 3D-shooter/FPS games that are on the market. Hellknöw will not require as much time or concentration as the 3D-shooters. The game will also be possibility for people who might not have a good enough computer to play the newest 3D-shooter.

2.2 The main uses of the system.

The system is generally going to be used for entertainment. It will give friends something to do together that they both can enjoy and talk about afterwards.

Scenario 1:

Martin is a 16 year old gamer who just arrived home after a long exhausting day at school. Martin does not want to do any more school work today. He sits down at his computer and logs on to his favorite chat client. He engages in a conversation with his good friend Åsa, who also is a gamer of age 17. Martin needs something more than just a simple conversation to get his mind off the school work. He decides to challenge Åsa to a game of Hellknöw. He tells Åsa that he wants to play and she accepts his challenge. Both Åsa and Martin have Hellknöw previously installed on their computers. They both start the game. Since there is nothing to load for previous sessions they can easily start a game. They decide that Martin will initiate the game and act as server. Martin gets his IP from the game, starts the server, and gives the IP to Åsa. Upon receiving the IP, Åsa enters it into the game and connects. The first round starts with Martin's character spawning at one side of the course and Åsa's character spawning on the other side. When they spawn, both players have randomly been assigned a couple of weapons from the collection of weapons in the game. Both players can freely move around the map. Martin starts exploring the stage, because of the fog of war feature he can only see part of the map at the time, so he carefully moves his character around, ready to shoot if Åsa suddenly appear. Åsa, on the other hand, moves around quickly and manages to sneak up on Martin from behind. Since Martin only can see what is in front of him, he does not notice Åsa. Åsa realizes this and uses the knife weapon that she has been assign to brutally stab Martin from behind. When Martin's character dies the round is over and a new round is started. While playing, Martin uses the chat function to comment on Åsa's skill with the pizza-weapon. They both have a good laugh about the comment. They play Hellknöw for about half an hour before crowning Åsa as the winner. They both disconnect by closing the game down. Martin feel refreshed and really enjoyed the playing, now he will have something to talk about with Åsa at school tomorrow. He is now ready to resume his studies on the computer.

Scenario 2:

At home Misse, a 22 year old gamer, just got beaten by his dear friend Milan, a gamer in the same age range. So he decided to practice his skills in just the practice mode. He was not very sure how the practice mode would differ from the multiplayer. Therefore, he opened the game, selects the practice mode option, and finds himself in the same environment as he was beaten by Milan except that he is alone this time. "This way," he thinks to himself, "I will be able to get really familiar with the environment without being blown to pieces by my best friend." "And Milan will be so impressed by my tremendous improvement in the game of Hellknöw!" Armed with only a pizza for weapon he strolls around the level discovering an excellent spot for killing Milan, a large pit. Satisfied with his devious plan he ends the practice mode. He feels excited when he thinks about the potential victory during the next game.

2.3 The context/environment in which the system is to be used

Hellknöw can be played while sitting in the same room as the opponent and it can also be played against an opponent at the other side of the world. It is mainly supposed for home use, but it could also be played at LAN-parties (Local Area Network-parties) or at schools, workplaces and such (where gaming is allowed). The game will be developed for Windows XP. To play the multiplayer mode the users must be connected on a network, local or over the Internet. The single player mode will only require that the computer can run the game.

2.4 The scope of the system

Since the game aims for people who want to pick up a small game during their free time, the game will not have too many or complicated features. However, it need enough material to keep the player entertained.

Topic	**in**	**out**
Game intro	X	
About Hellknöw...	X	
Help/ instructions	X	
Menu	X	
Submenu for practice	X	
Submenu for multiplayer	X	
More than 2 players		X
Sound	X	
Control modifiability		X
Connection notification	X	
Life/Hitpoints	X	
Multiple Weapons	X	
Display user IP	X	

Multiple Stages		X
Background music		X
Full screen mode		X
Support for patching		X
Chat	X	
Other communication types (than chat)		X
Artificial Opponents		X
Save Games		X
High Score		X

2.5 The main factors that need to be taken into account when designing and building the system

To be able to implement a game with multiple programmers developing different features we need to agree on a good base structure to build upon. We also need to develop a protocol for network communication between the client/server side and the client side, which is effective and tolerant to errors like lag.

- If one of the players loses his or her connection to the server, the game should be paused and all the players should be notified about this. The player who is hosting the game will then receive a message about the connection loss and gets the option of either try to reestablish the connection, or to close the game done.
- If one player has a slow or lossy connection, he should still be able to play the game. This means that the client must be able to manage updates when not receiving all the packages.
- It should be possible for an enthusiast to either buy or download the source code several years from now and be able to add features without understand the whole code.
- The game needs to have realistic system and network requirements to be played. A realistic system is a system that is not more than 4 years old, and is able to run Windows XP efficiently. A realistic network connection is a connection of at least 0,5 Mbit/s in upload and download

speed.

- We need to implement a working Fog of War feature.

2. 6 Technologies and Risks

We plan to implement the game in Java 6. For the graphic we are planning on using The Lightweight Java Game Library (LWJGL) . All in the group are experienced Java programmers, however, one risk is that we all lack experience with the 3D libraries. Even though the game is going to be in 2D, it would be beneficial to use a 3D library since it is usually faster than the 2D libraries. It also gives us the possibilities to improve on the visual effects. By choosing LWJGL there is also the risk that a 3rd-party library might not be fully documented and it might be hard to get support from. Another problem with that visual part of the game is that none of us are graphical artists.

For the network interaction we are planning to use regular Java-sockets (TCP/IP protocol). The risks with this is that we lack experience with creating protocols and effectively transmitting them.

We need to be able to implement the Fog of War feature by relating it to ray tracing. ”Ray tracing is a general technique from geometrical optics of modeling the path taken by light by following rays of light as they interact with optical surfaces.” (From: Wikipedia http://en.wikipedia.org/wiki/Ray_tracing). The risk with this is that do not at the moment know how to implement this.

3. Glossary

Adversary	The opponents player.
API (application programming interface)	An API is a source code interface
Fog of War	Gaming term used for the bluring out/hiding things that the player should not be able to see.

FPS games	First Person Shooter. An FPS is a game where the user's point of view is through the eyes of his avatar, also the core of the game revolves around the user using weapons to shoot.
Java 6	Java is a programming language that will be used in this project. The 6 denotes the version of the programming language.
Java-sockets	A part of the Java programming language that provides a way to communicate over the TCP/IP-protocol.
Kinetic Energy	A measure of how much energy a moving object has, calculated as one half the mass divided by the velocity squared.
Level	A 2 dimensional display where the player can move around and act.
Lightweight Java Game Library (LWJGL)	A game library that has code for handling OpenGL graphics.
Momentum	A measure of motion calculated as mass times velocity.
Movable Objects	Objects that can move or be moved e.g. bullets and crates.
OpenGL (Open Graphics Library)	OpenGL is a standard specification defining an API for writing applications that produce 2D and 3D computer graphics
Player	The avatar that the user controls.
Ray tracing	Ray tracing is a general technique from geometrical optics of modeling the path taken by light by following rays of light as they interact with optical surfaces.
Stationary Objects	Objects that cannot move or be moved e.g. walls and ceiling.
TCP/IP protocol	A protocol for communicating over the Internet.
User	The person playing the game.

4. User requirements definition

4.1 Functional requirements

4.1.1 Objects:

4.1.1.1 Changes in movement

Players shall act/react as movable objects to change in movement.

a. A player should be able to push another player who is not moving by running into him.

Rationale: To make the game feel more realistic, all objects should interact with each other in the same way.

Test: A player pushes another player and a movable object. If both the player and the movable object reacts in the same way the test succeeds.

4.1.1.2 Interaction

Objects shall be able to interact with other movable objects.

a. When two movable objects collide, total momentum should be preserved.

Rationale: By interacting with movable objects one player can hurt another player.

Test: Try pushing a movable object towards another movable object which is standing still. When they collide the object that stands still should gain momentum and the other object should lose momentum.

4.1.1.3 Frictional force

All moving objects are subject to preset frictional force.

a. A object that is moving will eventually stop if not affected by a external force.

Rationale: To provide a more realistic environment, all objects should eventually stop.

Test: Try pushing a movable object towards another movable object which is standing still. When they collide the object that stands still should gain momentum and the other object should lose momentum. Both objects shall eventually stop.

4.1.1.4 Gravitational force

All movable object are subject to a preset gravitational force.

a. All objects will eventually hit the ground if not affected by an external force.

Rationale: To provide a more realistic environment, all objects should be pulled downwards.

Test: A player should jump up in the air. If he comes down the test succeeds.

4.1.2 Obstacles

4.1.2.1

There will be obstacles in the game preventing the player to move in certain directions.

Rationale: This allows the levels to be more versatile.

Test: Try moving against an obstacle. If it isn't possible to continue moving in the direction the test is successful.

4.1.2.2

Different obstacles have different properties.

Rationale: This allows the levels to be more versatile.

Test: Try moving against the specific obstacle and see if the specific property hold.

Table of different obstacles and their properties:

Obstacle:	*Property:*
Ladder	The player is able to climb upwards or downwards.
Crate	A movable object.
Pit	A hole in the ground resulting in death if fallen into.

4.1.3 Weapons:

4.1.3.1 Assigned weapons

All players are automatically assigned two weapons at the start of each round.

Rationale: To be able to hurt the adversary, each player is automatically assigned two weapons at the start of every round.

Test: Start the game and see if all players have two weapons.

List of weapons that the players may be assigned

Name:	Description:
Knife	A short blade used for stabbing the adversary in close combat.
Katana	A sharp blade used in close combat. Inflicts more damage than the knife.
Baseball bat	Swing the bat in close combat to inflict the damage. Does less damage than the katana and the knife.
Pizza	Delicious food used do hurt the adversary. Throw the sticky pizza slices at the adversary to inflict damage.
BS Pistol	Shots bullets that inflicts more damage than the pizza slices.
EMO Laser	Shoots laser beams. Causes sever damage to the player. The weapon in the game that inflects most damage.

4.1.3.2 Cool down

All weapons are subject to a preset cool down time that vary from weapon to weapon.

a. When a player has fired a weapon he has to wait a time of greater than or equal to the cool down time before he can fire the weapon again.

Rationale: Different weapons have different strengths and to make the game more fair, it should be possible to fire a weak weapon faster then a strong weapon.

Test: Try to fire different weapons rapidly and see if there is a difference in how fast they can be fired.

4.1.3.3 Gravity affect bullets.

Bullets from a weapon is not affected by gravity.

Rationale: Bullets need to move slower than in real life to make them even visible to the user. If they were to be affected by gravity, they would fall to the ground too quickly.

Test: Shoot and see if the bullets are affected by gravity. If the bullets are not affected by gravity the test is successful.

4.1.4 Player:

4.1.4.1 Move horizontally

A player shall be able to move horizontally.

a. There is a fixed rate of acceleration and deceleration.

b. There is a maximum velocity.

Rationale: By affecting horizontal movement the player is able to explore the level.

Test: Try to move the player and see if he can move horizontally.

4.1.4.2 Jump

A player shall be able to jump.

a. When starting a jump, the player gets an initial upward speed.

b. A player can not jump while in the air or on a ladder.

Rationale: Jumping gives the player the ability to move vertically to pass objects.

Test: The player should jump and if the player moves up in the air the test succeeds.

4.1.4.3 Crouch

A player shall be able to crouch.

a. When crouching, the height of the player should be reduced by half.

b. When crouching, the players' ability to affect horizontal movement is decreased by a preset value.

Rationale: Crouching gives a player the ability to hide behind objects and dodge objects.

Test: Make a player crouch and see if his height is decreased and that the player can't move as fast as before.

4.1.4.4 Climb ladders

A player shall be able to climb ladders.

a. When on a ladder, a player is able to affect his vertical momentum.

Rationale: Ladders gives a player access to more of the map.

Test: Let a player stand on a ladder and see if he can move vertically.

4.1.4.5 Pass through objects

A player shall not be able to pass through stationary objects or movable objects.

Rationale: To make the game feel more realistic, stationary objects or movable objects should not pass through each other.

Test: Try to make objects pass through each other.

4.1.4.6 Changing line of sight

A player shall be able to change his line of sight.

Rationale: To be able to explore the map and find his/her adversary, the player needs to change his line of sight and thus chaining his field of view.

Test: Make the player look around and see if the field of view changes.

4.1.4.7 Use assigned weapons

A player shall be able to use any of his assigned weapon.

Rationale: The player must be able to use all weapons assigned to him for the game to be functional.

Test: See if anything changes when a player uses his weapon.

4.1.4.8 Switch weapons

A player shall be able to switch between assigned weapons.

Rationale: Since a player will be assigned multiple weapons, the player should be able to switch between these.

Test: Try the first weapon, change weapon, and see if the player can use the other weapon.

4.1.4.9 Lose health points

A player shall lose health points when hit by a moving object. The amount of health lost is depending on the object's momentum.

*Rational*e: A player's goal is to kill the other player by hitting him/her with movable objects.

Test: Throw different objects at a player and see if his health points change.

4.1.4.9 Losing health points when hit by a weapon

A player that is hit by a weapon or bullet from a weapon will loose additional health points compared to being hit by a movable object

*Rational*e: In real life bullets and other weapons have a piercing effect that make them deadly. To reproduce this in the game, we will give them additional damage.

Test: Throw an object at a player and then shoot at the player. If the weapons do an additional damage compared to the object the test is successful.

4.1.5 Network

4.1.5.1

Upon loss of connection during a game players will be notified.

Rationale: The players will want to know if connectivity is the fault if their program breaks up.

Test: Start up a game with two players and physically disconnect them.

4.2 Non-functional requirements

4.2.1.1 Minimum requirements

The minimum requirements to run the game shall be a computer with Windows XP, 1.6 GHz processor, 512 MB RAM and a graphic card that supports OpenGL (Open Graphics Library).

Rationale: These specifications reflect an ordinary computer which the game is supposed to run on.

Test: Run the game on a computer with the specified specifications and see if it works.

4.2.1.2 Size of the game

The size of the game shall not be greater than 100 MB.

Rationale: The user should be able to download the game from a homepage.

Test: Validate the size of the game and see if it is less than 100 MB.

4.2.1.3 Frame rate

The game shall update the frames with a frequency of at least 20 frames per second.

Rationale: 20 frames per second update rate is required for the game experience to feel smooth.

Test: Run the game on a machine that has our minimum requirements specifics and check the frame rate during gameplay.

4.3 Use Cases

4.3.1 Use a weapon

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – wants to control the player to use a weapon.

Preconditions: The player is in a game and has been assigned two weapons.

Minimal Guarantee: The player stays in the game.

Success Guarantee: The weapon is successfully operated.

Trigger: Player has weapons and wants to use it.

Main Success Scenario:

1. The user aims at his desired target.
2. The user fires the weapon.
3. The weapon's effect is executed.

Extensions:

3a The weapon's effect is not executed.

3a1: The weapon is subject to a cool down: The user waits a moment then fires his weapon again.

4.3.2 Interfere with the adversary

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – wants to use the player to interfere with the adversary.

Opponent user - wants to use the adversary to hurt or interfere with the adversary.

Preconditions: The player and the opponent are in a multiplayer game together.

Minimal Guarantee: The players stay in the game.

Success Guarantee: The adversary gets affected by the interference or losing health.

Trigger: User wants to use the player to affect the adversary by moving an object.

Main Success Scenario:

1. The user uses the player to move an object.
2. The adversary collides with the moving object.
3. The adversary loses health and gains momentum accordingly.

Extensions:

3a. The Adversary's health loss is dependent of the velocities of the adversary and the moving object. The greater the relative velocity, the greater the loss.

3b. The Adversary's detour is dependent of the velocities and weights of the adversary and the moving object. The kinetic energy and the momentum are conserved throughout the collision.

4.3.3 Send a text message to the opponent

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – wants to send a message to the opponent

Opponent – wants to receive a text message from the opponent.

Preconditions: The player and the adversary are in a multiplayer game together.

Minimal Guarantee: The players stay in the game.

Success Guarantee: A text message is sent to the opponent.

Trigger: The user wants to send a text message to his opponent.

Main Success Scenario:

1. The user starts to write a message.
2. The user sends the message.
3. The message is sent to the opponent and is displayed for the user and the opponent.

Extensions:

3a: The message is too long to be sent.

3a1: The user starts over from the beginning using less words.

4.3.4 Hurt the adversary

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – Wants to reduce his adversary's health by the correct amount.

Opponent - has the same interest as the user.

Preconditions: The user and his opponent are in a multiplayer game together. The player is equipped with a weapon.

Minimal Guarantee: The user and the opponent stay in the game.

Success Guarantee: The amount of health points that the adversary has is reduced by the amount specified for the user's weapon.

Trigger: User wants to fire his weapon aimed at the adversary.

Main Success Scenario:

1. The user fires the weapon.
2. The opponent is hit.
3. The opponent loses health points.

Extensions:

1a. User is unable to fire his weapon because of a cool down:

1a1: The user waits for the cool down to end and then tries to fire his weapon.

2a. Opponent is not hit:

2a1: The opponent receives zero reduction in health points.

4.3.5 Kill the opponent

Primary Actor: User.

Scope: Hellknöw.

Stakeholders and interests:

User – wants to kill the adversary.

Opponent - wants his player to die.

Preconditions: The user and the opponent are in a multiplayer game together. The user is equipped with a weapon that has a specific range.

Minimal Guarantee: The user and the adversary stay in the game.

Success Guarantee: The adversary dies.

Trigger: User wants to fire his weapon aimed at the adversary.

Main Success Scenario:

1. The user hurts the opponent's player.
2. The opponent's player amount of health is zero or negative.
3. The opponent's player dies.

Extensions:

2a. The opponent's player's amount of health points is positive:

2a1: The user hurts the opponent's player again.

4.3.6 Start up a single player game

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – wants to initiate a game of Hellknöw.

Preconditions: The user has the game Hellknöw installed on his computer. The users computer meets our minimal requirements for the game to function properly.

Minimal Guarantee: The users computer still functions properly.

Success Guarantee: A singleplayer game of Hellknöw is initiated.

Trigger: User wants to play a game of Hellknöw by himself.

Main Success Scenario:

1. The user starts the Hellknöw application.
2. The user initiates the singleplayer mode.
3. The single player mode is loaded by the system.

Extensions:

1a: The application does not start:

1a1: The application is somehow faulty: The user uninstall and then reinstall Hellknöw.

4.3.7 Start up a multiplayer game and acting as server

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – wants to host a multiplayer game of Hellknöw.

Preconditions: The user has the game Hellknöw installed on his computer. The users computer meets our minimal requirements for the game to function properly. The user has a pre-established means of communications with his friend whom he intends to play with.

Minimal Guarantee: The users computer still functions properly.

Success Guarantee: A multiplayer game of Hellknöw is initiated.

Trigger: User wants to play and host a multiplayer game of Hellknöw which he wants to play with a friend.

Main Success Scenario:

1. The user starts the Hellknöw application.
2. The user initiates the multiplayer mode and indicates that he wants to be the host of the game.
3. The user notes and sends his IP address to his friend.
4. The friend connects to the user's game.
5. The multiplayer game loads and is initiated.

Extensions:

1a: The application does not start:

1a1: The application is somehow faulty: The user uninstalls and then reinstalls Hellknöw.

4a: The connection cannot be established:

4a1: The connection is refused to the friend: The friend evaluates his connection to the user. If the friend does not find any fault, he tries to connect to users IP address again through the game.

4a2: The connection is refused to the friend: The user examines the connectivity of his IP address and restarts the multiplayer connection.

4-5a: The connection is lost after a successful connection

4-5a1: An error message is sent to both the user and friend indicating the loss of connection.

4-5a2: Repeat steps 2-5.

1-5a: The game crashes.

1-5a1: When the connection has already established; an error message is sent to both users, and the game is aborted for both.

1-5a2: When the connection is not yet established; an error message is sent to the user who uses the crashing computer, and the game is aborted on that computer.

1-5a3: Repeat steps 1-5.

4.3.8 Join a multiplayer game

Primary Actor: User

Scope: Hellknöw

Stakeholders and interests:

User – wants to join a game of Hellknöw.

Preconditions: The user has the game Hellknöw installed on his computer. The user's computer meets our minimal requirements for the game to function properly. The user has a means of communications with his friend. His friend has successfully hosted a multiplayer game of Hellknöw.

Minimal Guarantee: The user's computer still functions properly.

Success Guarantee: A multiplayer game starts with the user and his friend.

Trigger: User wants to play a game of Hellknöw with a friend.

Main Success Scenario:

1. The user starts the Hellknöw application.
2. The user initiates the multiplayer mode and indicates that he wants to join an existing game.
3. The user connects to the IP address given by his friend through the multiplayer connection mode.
4. The game connects the user to his friend and starts up the game.

Extensions:

1a: The application does not start:

1a1: The application is somehow faulty: The user uninstalls and then reinstalls Hellknöw.

3a: The connection cannot be established:

3a1: The connection is refused to the friend: The friend evaluates his connection to the user. If the friend does not find any fault, he tries to connect to users IP address again through the game.

3a2: The connection is refused to the friend: The user examines the connectivity of his IP address and restarts the multiplayer connection.

3-4a: The connection is lost after a successful connection

3-4a1: An error message is sent to both the user and friend indicating the loss of connection.

3-4a2: Repeat steps 2-4.

1-4a: The game crashes.

1-4a1: When the connection has already established; an error message is sent to both users, and the game is aborted for both.

1-4a2: When the connection is not yet established; an error message is sent to the user who uses the crashing computer, and the game is aborted on that computer.

1-4a3: Repeat steps 1-4.

5. System architecture

Java, that we are going to use, is an object oriented programming language. We will try to divide the system into independent parts. Our goal is it to be able to implement for example a new weapon by just adding a new weapon object without the need of changing anything else. To be able to do this we need to predefine what every type of object needs to contain, both in terms of variables and methods. So we need to construct a centralized model with one main system and the sub-modules that the main system can use. The sub-modules need to be independent.

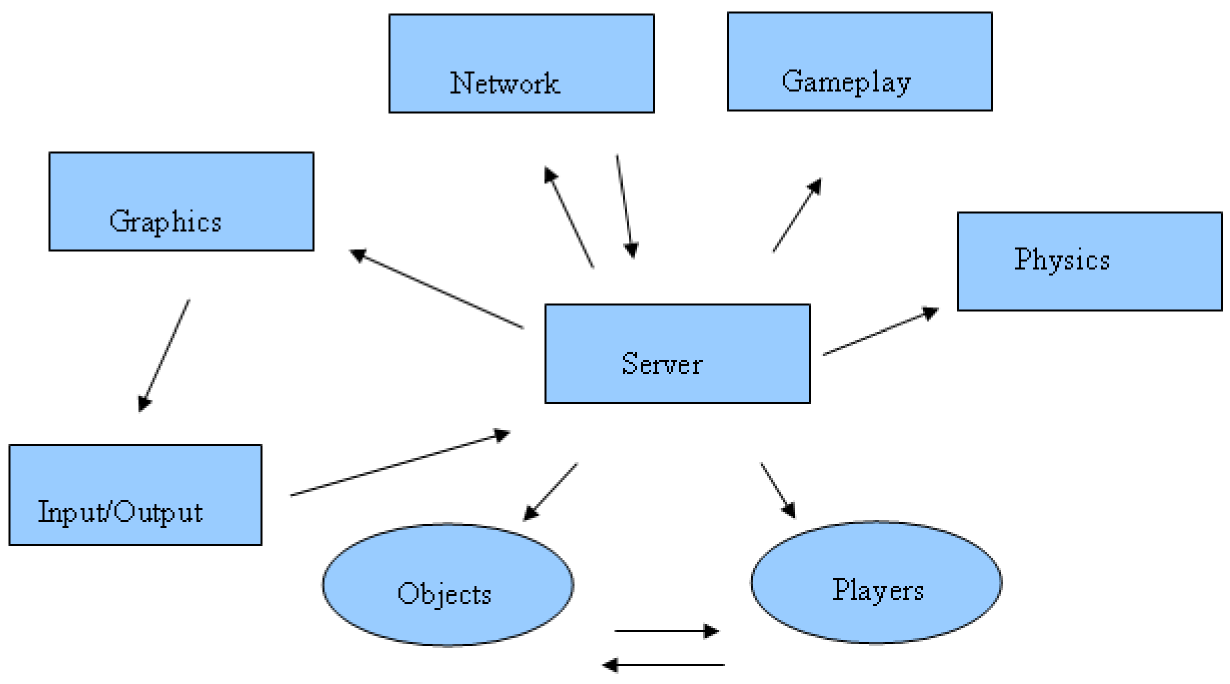

figure 1

This is the dependency chart that we will strive to maintain. The rectangular boxes are regular classes and the ellipse shaped boxes are abstract classes. An abstract class is like a layout that all sibling classes need to follow. This means that there can be many different instances of objects or players, but they all need to have the same standard functions so that the server knows how to use them. The main parts of the system, graphics, network, gameplay, physics and server will all be coded individually. The main "brain" of the system is the server that handles the communication and interactions between all the other classes.

The main game flow will consist of the user starting a game and thereby creating a server. This server will then as the user for input about what kind of session is going to be played. Say that the user chooses multiplayer. The server will then use the network part of the game to establish a connection with another user. When the connection is established the server will use gameplay to start a new game, and create objects and players. The user feeds the server with input about how he wants to move his avatar and the network gives the server information about the other user. The server takes this information, uses physics to update it and then sends everything to graphics, that will produce a visual image that then will be displayed for the user. The server will continue taking information from the user and the network until the game is over.

A main factor in the multiplayer case is updating objects, players, etc, at both ends; these updates need to remain identical. To acquire this property we are going to have something called a master server and a slave server. The master server will act as in the example above (figure 1). The slave server is going to work a little bit differently. Instead of sending its user input though server and to physics, it's going to go through server and to network. This means that the master server does all the calculations and then sends it to the slave server. This results in a system structure that looks like the figure below (figure 2).

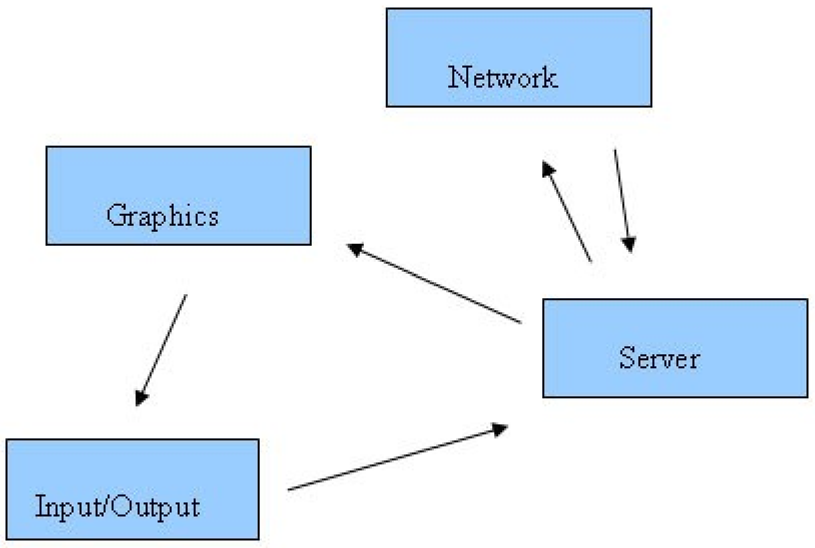

figure 2- slave server

6. System requirements specification

6.1 Functional requirements

6.1.1 Calculate new speeds of two objects that have collided

Function: Calculate the new speed of an object after a collision.

Description: Calculates the new velocity in vertical and horizontal speed.

Inputs: Two objects initial vertical and horizontal speed and weight.

Source: The game system.

Outputs: New vertical and horizontal speeds for both objects.

Destination: Main control loop.

Action: Using these formulas speeds are calculated:

$$v_{1xn}=\frac{(m_1-m_2)}{(m_1+m_2)}*v_{1xb}+\frac{2*m_2}{(m_1+m_2)}*v_{2xb}$$

$$v_{2xn}=\frac{2*m_1}{(m_1+m_2)}*v_{1xb}+\frac{(m_1-m_2)}{(m_1+m_2)}*v_{2xb}$$

Where v_{1xn} is the horizontal speed of object 1 and similarly v_{2xn} is the horizontal speed of object 2. The 'n' marks 'new' and the 'b' marks 'before'. The vertical speeds are calculated in the same way with the exception of the vertical initial speeds are used instead of the horizontal.

Requires: Nothing.

Preconditions: The masses are non zero and positive.

Postconditions: None.

Side effects: None.

6.1.2 Determine correct frictional force type

Function: Determine the correct type of frictional force.

Description: Calculates which force applies for the surface and the object.

Inputs: The two frictional coefficients, the alignment phi of the surface and the objects weight.

Source: The game system.

Outputs: New vertical and horizontal speeds for the object.

Destination: Main control loop.

Action: First calculate the gravitational force.

$f_g = Fg * \cos(phi)$

Then if this condition is satisfied.

$f_{smax} < my_s * f_g$

It means that the object will slide along the surface and the resulting force will be f_k , and if its not satisfied then the force will be f_s which is calculated through these formulas.

$f_s = my_s * f_g$

$f_k = my_k * f_g$

The acceleration, a, is then determined as $\frac{f}{m}$ where m is the mass of the object and f is the calculated frictional force. Considering the effect of the acceleration in one instant would give the speed increment of a. The vertical and horizontal speed differences are now given by.

$v_{vertical} = a * \cos(fi)$

$v_{horizontal} = a * \sin(fi)$

Requires: Nothing.

Preconditions: The masses are non zero and positive.

Postconditions: None.

Side effects: None.

6.1.3 Exercise Gravitational force on objects

Function: Exercise gravitational force on an objects.

Description: Exercise gravitational force on an objects.

Inputs : Information that an object have moved.

Source: The game engine.

Outputs: New velocities for the object. My be the same as before the function.

Destination: The game engine.

Action: Checks that the object is on an unmovable object. If it is then no changes are done to the objects velocity. If the player is not on an unmovable object, then the object is in the air. The objects vertical velocity is now changed. It is changed according to this formula:

$V_f = V_i + g*t$ (Positive is downward, negative downward)

Where V_f is the final velocity, V_i is the initial velocity, t is the time elapsed, g is the gravitational acceleration constant.

Requires: Nothing

Pre-condition: An object has been moved

Post-condition: The velocities for the object has been updated.

Side effects: None

6.1.4 Weapon cool down

Function: Start the cool down for a weapon.

Description: Start the cool down for a weapon so that the player has to wait before he can fire again.

Inputs : Information that the weapon has fired.

Source: The game engine

Outputs: None

Destination: Nowhere

Action: Starts a timer with the initial value defined for the weapon. See chart below for cool downs for all the weapons. The timer then decreases until it reaches zero. When the timer reaches zero the player can fire the weapon again.

Requires: That the player is equipped with a weapon.

Pre-condition: That the player has fired a weapon.

Post-condition: None

Side effects: None

Cool down for the weapons.

Weapon	*Cool down (seconds)*
Knife	0.1
Katana	0.5
Baseball bat	0.6
Pizza	0.8
BS Pistol	0.5
EMO Laser	2

6.1.5 Verify weapon cool down.

Function: Verify that the weapon can shoot and is not subject to cool down.

Description: Verifies that the cool down has reached zero.

Inputs : The cool down of the weapon that the avatar is currently equipped with.

Source: The game system.

Outputs: True if the player can fire the equipped weapon, otherwise false.

Destination: The game system.

Action : Receive the current cool down. If the cooled own is greater than zero the weapon has been fired recently and can not be fired again until the cool down has reached zero. The function returns false. If the cool down is zero the weapon can fire again and the function returns true.

Requires: Nothing.

Pre-condition: The player is equipped with a weapon

Post-condition: The game system knows if the weapon the player is equipped with can fire again or not.

Side effects: None

6.1.6 Assign weapons

Function: Assign weapons to the player.

Description: At the start of each round both players are assigned with two weapons. One weapon is for close combat and one weapon is for long distance. Both weapons are chosen by random.

Inputs: Information about a new round of the game is to start.

Source: The game engine.

Outputs: Information that both players have been assigned weapons.

Destination: The game engine.

Action : First choose a close combat weapon randomly for player A. Then choose a long distance weapon randomly for player A. Player A has now been assigned his weapons.

Requires: Nothing

Pre-condition : A new round is about to start.

Post-condition: The player is assigned two weapons.

Side effects: None

6.1.7 Move horizontally

Function: Move a player horizontally on the level.

Description: Moves the player in the the desired direction.

Inputs : The direction the player desires to move.

Source: The game engine.

Outputs: The new horizontal velocity.

Destination: The game engine.

Action : The players horizontal velocity is updated by a preset constant.

Requires: Nothing

Pre-condition: An update in which direction the player is traveling

Post-condition: The players horizontal velocity is updated.

Side effects: None

6.1.8 Jump

Function: Move the player upwards into the air.

Description: Moves the player into the air.

Inputs : The information that the player wants to jump.

Source: The game engine.

Outputs: The new velocities for the player.

Destination: The game engine.

Action : If the player is in the air or on a ladder the vertical velocity is not zero. If it is not zero it is not possible to jump. The function returns same velocities the player had before trying to jump. If the vertical velocity is zero a preset constant is added to the vertical velocity.

Requires: Nothing

Pre-condition: That the player wants to jump.
Post-condition: The velocities of the player is updated.
Side effects: None.

6.1.9 Crouch

Function: Change the velocities for the player as he is crouching.
Description: Changes the velocities for the player as he is crouching.
Inputs : Information that the player is crouching.
Source: The game engine.
Outputs: The new velocities and the new height of the player.
Destination: The game engine.
Action : The height of the player is decreased by half. The vertical velocity is decreased by a preset value.
Requires: Nothing
Pre-condition: That the player wants to crouch.
Post-condition: The velocities and the height of the player is updated.
Side effects: None.

6.1.10 Climb ladders

Function: Changes the velocities of the player.
Description: Changes the velocities of the player so that he is moving upwards or downwards on the ladder.
Inputs: Information that the player wants to climb a ladder.
Source: The game engine.
Outputs: The new velocities of the player.
Destination: The game engine.
Action : The velocities of the player is changed so that the horizontal velocity is zero. The player is not able to move horizontally. The vertical velocity is changed by a preset. This value will be added to the vertical velocity if the player wants to climb up. If the player wants to climb down the preset value will be subtracted from the vertical velocity.
Requires: Nothing
Pre-condition : The player wants to climb a ladder.

Post-condition: The players velocities are updated.
Side effects: None.

6.1.11 Collide with stationary objects

Function: Check if the players path is not occupied by an object.
Description: Check if the players path is not occupied by an object. If an object is in the players way he should not be able to pass through it.
Inputs: Surrounding all objects is a rectangle, called a hit box. The coordinates for all hit boxes are the input.
Source: The game engine
Outputs: Information if the player is able to move in the direction he is traveling.
Destination: The game engine.
Action: Check if any of the hit boxes intersect with the players hit box. If a hit box intersects with the players hit box he should not be able to move in that direction.
Requires: That the player is traveling in a direction.
Pre-condition: That the player wants to move in a direction.
Post-condition: Information whether the player can move in the desired direction or not.
Side effects: None

6.1.12 Change line of sight

Function: Changes the visible area on the screen
Description: Changes the visible area of the screen so that the visible area from the players perspective is highlighted and the invisible area is shaded. The level is shown in both visible and invisible areas but not the objects located on the level.
Inputs: Information that the player is changing sight
Source: The game engine
Outputs: New visible area of the player
Destination: The game engine
Action: When changing the line of sight one shall see the current status of objects shown in the visible area, which is determined by user control; whereas the shaded area shows only the contour of the map but not the objects. What part of the map that is visible to the player is determined by tracing a number of rays from the head of the avatar to a object. The avatar will have a 80 degree field of vision meaning that the rays furthest to the ends will be 80 degrees apart (40 degrees from

the center of vision). These rays will establish what the user is able to see, everything else will be shaded out.

Requires: Nothing

Pre-condition: Player wants to change the line of sight

Post-condition: Player's line of sight is changed

Side effects: None

6.1.13 Switch weapons

Function: Switch weapon.

Description: A player changes his current weapon to another.

Inputs: Available weapons and current weapon.

Source: The game system.

Outputs: New current weapon.

Destination: The game system.

Action: The current weapon will be changed to the next one in the list of available weapons.

Requires: None.

Preconditions: The list of available weapons is not empty, the current weapon is among the available weapons.

Postconditions: The new current weapon is one of the players weapons. The current weapon has changed to another that is also part of the available weapons list.

Side effects: None

6.1.14 Lose health points due to weapon

Function: Reduce health points from a player.

Description: A player that is hit by a weapon or a bullet from a weapon will have his health points reduced.

Inputs : Information that the player was hit by a weapon or a bullet from a weapon and what kind of weapon it is.

Source: The game engine.

Outputs: The health points of the player

Destination: The game engine.

Action : The players health points is reduced by the amount that would be reduced when hit by a movable object (6.1.15). Additional damage will be subtracted. The amount subtracted depends on what weapon the player was hit by. A list of weapons and their damage is located below.

Requires: A player to deduct health points from.

Pre-condition: A player has been hit by a weapon or a bullet from a weapon and is therefore subject to health points deduction.

Post-condition: The players health point amount has been reduced by the amount of damage the weapon caused.

Side effects: None

List of weapons and their damage.

Weapon:	*Damage:*
Knife	25
Katana	45
Baseball bat	20
Pizza	10
BS Pistol	25
EMO Laser	55

6.1.15 Lose health points

Function: Reduce health points from a player.

Description: A player that is hit by a moving object will have his health points reduced.

Inputs: The weight and velocity of the object that hits the player.

Source: The game system.

Output: None.

Action: The momentum, p, of the object is calculated. The amount of damage the impact will cause is determined by the formula:

$$Damage = p * 0.1$$

Which will then be deducted from the players health point amount (note that since the momentum by definition is always positive the damage deducted is also always positive).

Requires: A player to deduct health points from

Preconditions: A player has been hit by a moving object and is therefore subject to health points deduction.

Postcondition: The players health point amount has been reduced by the amount of damage the moving object causes.

Side effects: None

6.1.16 Notify players upon loss of connection during a game

Function: Inform players that the connection between them has been lost.

Description: Inform the players that the connection between them has been lost.

Inputs : Error message that the connection has timed out.

Source: The client.

Outputs: Information to the user.

Destination: The client.

Action : If the connection has timed out, that is the client is unable to connect to the server within a preset amount of time the client will inform the user of this.

Requires: Nothing

Pre-condition : That the client has timed out.

Post-condition: Information is displayed to the user

Side effects: None

6.2 Use Cases

6.2.1 Two movable objects interact with each other

Primary Actor: Movable object.

Scope: Hellknöw

Stakeholders and interests:

System – wants the interaction to be resolved in accordance to the physical laws implemented in the system.

Preconditions: In an established game two movable objects are just about to collide. Both objects are in motion before the collision.

Minimal Guarantee: The established game does not crash.

Success Guarantee:The two objects are in turn affected by the correct forces that are the result of the collision. The kinetic energy and momentum is preserved through the collision.

Trigger: The system wants to know what will happen to the two objects that collide.

Main Success Scenario:

1. Calculate kinetic energy and momentum of the objects and store them.
2. The new speed is calculated separately for both objects divided into horizontal and vertical speed.
3. Calculate the new kinetic energy and momentum of the objects.

4. Validate that the sum of kinetic energy of the objects before collision equals that of the objects after the collision.
5. Validate that the sum of momentum of the objects before collision equals that of the objects after the collision.

Extensions:

4-5a Validation fails:

4a1: For some reason the two sums differ albeit only by a small margin: Small differences are acceptable, nothing will be done to rectify.

4a2: The difference is major: The formulas are wrong or there have been interference somewhere along the line. The game will abort.

6.2.2 Calculate frictional force between object and surface

Primary Actor: Movable object

Scope: Hellknöw

Stakeholders and interests:

Object – needs to know on what surface it is moving, so the frictional force is determined.

Preconditions: A movable object is moving along a surface. The surface has defined the two frictional constants my_s and my_k .

Minimal Guarantee:

The established game does not crash.

Success Guarantee: The frictional force is calculated and applied to the object.

Trigger: The system wants to know how an object is to be affected by the surface it is moving on.

Main Success Scenario:

1. The gravitational force of the object is calculated.
2. The correct frictional force type is determined.
3. The frictional force is calculated.
4. The acceleration of the object to which the frictional force applies is calculated.
5. The acceleration is applied to the objects speed.

Extensions: None

7 System evolution

7.1 Fundamentals that the System is Based On

To run Hellknöw the user will be required to have a computer that runs Windows XP efficiently combined with an installation of the Java Runtime Environment version 6. This means that the computer should have at least 512 MB of ram with a processor of 1.8 GHz or more. The computer should also at least have 1 GB of free hard drive space.

To be able to play the multilayer mode, the computer will need an Ethernet card that supports transfer speeds of at least 0.5 Mbit/second. The user will of course also need and opponent user with the same setup. The users computer also needs to be able to understand the TCP protocol as all communication within the game will run on that protocol.

7.2 Changes in the environment or needs

As long as the user's computer can run, or even simulate/emulate, Windows XP and the Java runtime environment version 6, the user should be able to run Hellknöw.

Hardware changed that might effect Hellknöw is in the relatively unlikely event the TCP protocol is changed into something else, then Hellknöw won't be able to run. The solution to this problem is to reprogram the communication part of the program to use the new standard.

The controls in Hellknöw will be predefined and the user will not be able to change these unless he/she changes the source code and recompiles the game. This will only be a problem if the I/O (Input/Output) standard for computers change. This is very unlikely to occur and nothing we anticipate.

Hellknöw will not have an integrated patching system, so if any of the features need to be changed, the source code will need to be rewritten, recompiled and then be completely re-downloaded by the user. We do not plan for the game to have a lifespan of longer than maybe a year and then it might be used occasionally for retro gaming. During this use the game should not need any updates or reconfigurations.

8. Appendices

8.1 Organization and Structure of Data

In a folder-tree view, the main files of the system that the user need to execute to start the game, will be located in the root folder. All other complementary files will be organized in sub-folders. There will be separate sub-folders for all the objects and weapons. All sprites will be located in one folder. All sprite filenames should start with a prefix describing what the sprite is for. The system will not store any information about the player or any state of a previous game so we do not need to take that into account.

Alphabetical Index