Project: Jarl
Design document version 1

Group Number: 18

Magnus Andermo
Mattias Frånberg
Carl Johan Gustavsson
Pontus Stenetorp

Contents

1 Introduction

This document is intended to describe all implementiation aspects for the game Jarl. The document is intended for readers with a technical background who are to implement the game or to review the game and give their opinions.

Before reading this document the reader shall be familiar with the requirements document for Jarl. For a general overview without indepth details the reader shall see the project overview document for Jarl.

1.1 Summary

Here is a brief summary of the document.

The first section is the introduction which you are currently reading. The second section contains an overview of the system. All parts are handled, server, client and all their respective subcomponents. First there is an overall description and then a detailed description. The third section discusses design considerations. That is, constraints, assumptions, dependencies and how they will affect the game. Section four describes the graphical user interface, functionality and graphically. The fifth section is the most detailed and largest section, it covers implementation details, class hierarchies, state charts and all other elements needed in order to implement the system. The sixth and last section covers test cases in order to test the functional requirements found in the requirements document, these are intended for testers and implementers.

2 System Overview

2.1 General Description

Jarl is a strategy game with a variety of game elements not commonly associated with other strategy games. Instead of offering full 3D or a 2D view from above, Jarl has 2D segments in which units move much like in a sidescrolling platform game. It is designed to be played against human players and offers various strategies, a player can foucus on improving his units, his hero or just to spend resources on more units. Jarl offers a pecking order, each player has to fend off the foe behind him while attempting to defeat the foe in front of him. Each player therefore has a predator and a prey and battle each other in a circle.

Using the game lobby players can join up in battles with other players from all over the world. A players results are stored and gives each player a ranking to compare himself against other players.

2.2 Overall Architecture Description

2.2.1 Lobby server

The lobby server is responsible for keeping data about games that not yet have been started. It's also responsible for delivering data to the lobby clients. When a new game session is created the lobby server is responsible for spawning a new game server for that session.

The lobby server consists of a network module, a game server handler, a database module and the lobby handler.

2.2.2 Game server

The *Game server* consists of a network module, a client handler and a proxy module.

Client handler
The *client handler* filters game commands arriving from the clients and forwards them to the *client controller* or the *proxy*. Client control commands are commands regarding the overall game, for example leaving, winning and loosing. Proxy commands are commands broadcasted to *all* clients for example commands that the user issues in the GUI that changes the game world.

2.2.3 Lobby client

The lobby client is responsible for connecting a user to the lobby server. Through the lobby client the user is able to communicate and join non-started game sessions and then be able to join game sessions.

2.2.4 Game client

The game client is responsible for holding a current game state and present it to the user. The user can affect the game state by giving the game client input which is then sent to the game server. The game client will continuously receive game state updates from the game server and apply the changes to it's game state.

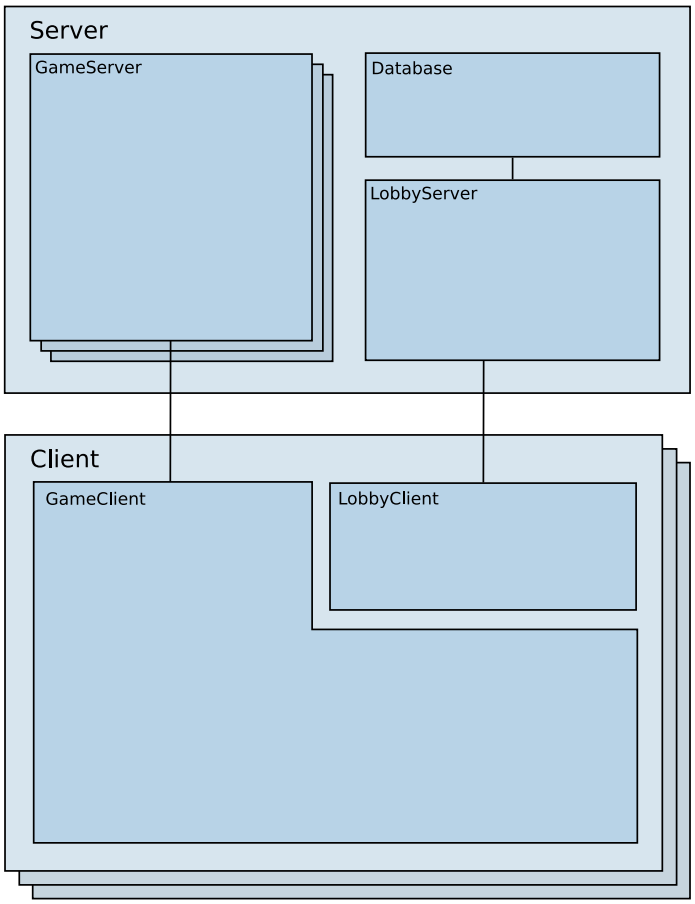


Figure 1: Diagram decribing the full system.

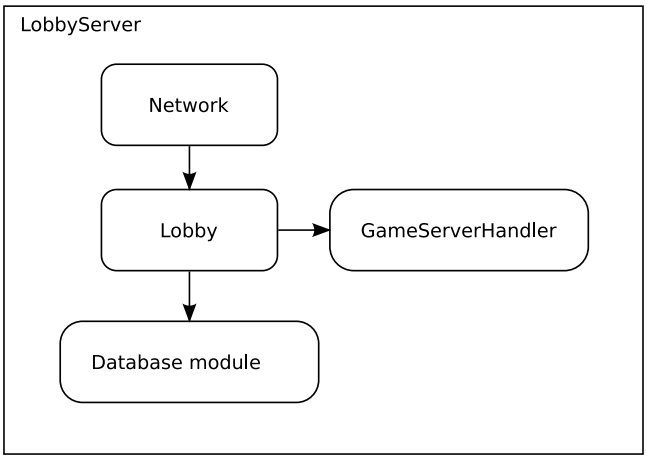


Figure 2: Picture describing how the Lobby server subsystems interact.

2.3 Detailed Architecture

2.3.1 Lobby server

Lobby
The Lobby handles requests from the client regarding game session states. It provides the client with data about the current game sessions. When a game session should start the Lobby mediates between clients and the game server handler and is responsible for that the game server handler has all the data required to start the game session.

Game server handler
The game server handles the interaction between the Lobby Server and the Game Servers. The Game server handler is responsible for starting up Game Servers. The game server handler is part of the Lobby Server.

User database
Contains data about all users and their wins and losses.

2.3.2 Game server

Network
The commands coming through the external networks are either Game commands or commands that handles interaction between the client and the server, for example a client disconnecting.

When a client sends a game command that command will get executed at the same frame on all clients, because all clients have the same state and random seed the command will result in the same new state for all clients. The Artificial Intelligence will act based on the same state and random seed on all clients, so the AI decisions will be the same for all clients.

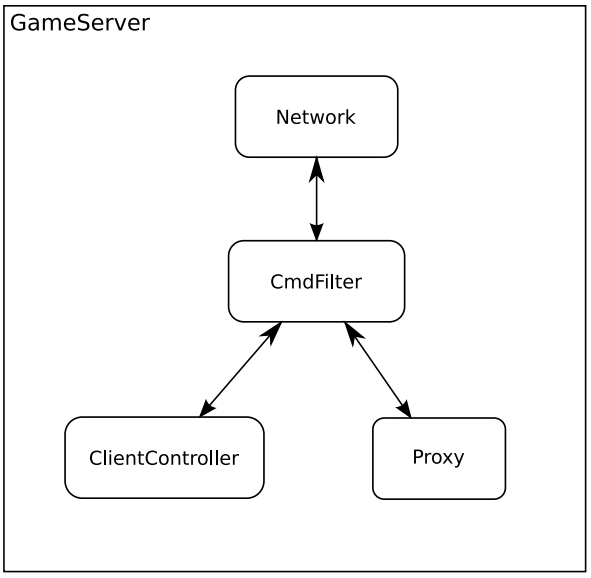


Figure 3: Picture describing how the Game server subsystems interact.

Command filter
The CmdFilter distinguishes between interaction and game commands and forwards them to either the ClientController or the Proxy.

Client controller
The ClientController handles all the management of the clients.

Proxy
The proxy module of the game server is responsible for distributing game commands to the clients. The clients send commands that represent user commands. The server runs frames with a fixed frequency. All commands that where received during a server frame should be executed at the same frame on all game clients. A frame is an interation of the game loop including logic and rendering. The server signals the end of a server frame by sending a frame end command. Consequently the frame end commands will be sent from the server at equally spaced intervals, but the commands will not arrive at equally spaced intervals, due to difference in network latency. If we execute the frames the moment we receive them from the server, the time the frames will be shown to the user will variate. If that variation in frame time will be clearly observable and affects the gameplay experience, we will compensate for that by having a buffer of frames and use heuristic that will decide when to execute the next frame.

2.3.3 Lobby client

Lobby
The lobby client presents a lobby which consists of data received from the lobby server. The data is presented using a GUI. The data presented is the current

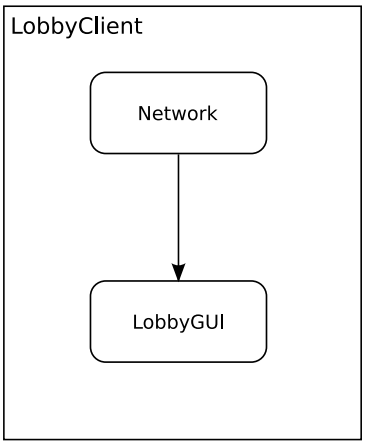


Figure 4: Picture describing how the Lobby client sub systems interact.

joinable games and the players currently connected to the server. It also allows the user to create a new game using the GUI.

2.3.4 Game client

IO-client
The IO-client takes input from the user via the keyboard or mouse and transforms them into game commands.

Game
Game forwards commands coming from the IO-client to the Network. Game also keeps track of which commands that should be executed in their respective rendering frame. Each frame is a set of commands that shall be executed. Game commands that changes the game world will alter the world repository.

Network
Handles game commands coming from the server and appends them to the frame which they belong to.

World
World is a data repository that holds all the objects in the world and the data corresponding to them.

Renderer
The renderer will on request read the world repository and create an image of the current game state based on the data from the View. The data about the clients current view determines which part of the game world that should be rendered.

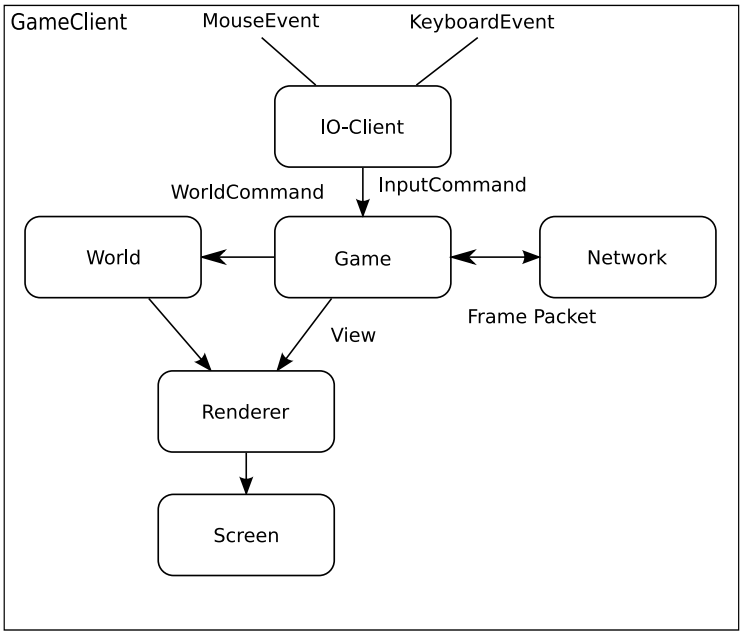


Figure 5: Picture describing how the Game client subsystems interact.

3 Design Considerations

3.1 Assumptions and Dependencies

The system assumes to be running under Windows XP, Ubuntu Gutsy Gibbon or Mac OS X. The system does not restrict which platforms to run on, but dependent libraries do. Light-Weight Java Game Library is used by the system and currently supports Windows, Linux and Mac OS X. The system also depends on Java, Java is at the time of writing supporting numerous platforms, which includes many more than the ones supported by LWJGL.

It is probably that future Java versions will change it's requirements and features on Java code. The system will in that case require patches accordingly. Another factor of change is LWJGL, we already know that version 2.0 will loose support for texture loading and audio loading. These will how ever be supported through external libraries supplied by the makers of LWJGL. These external libraries will use the same syntax and the change shall therefore be seemless.

3.2 General Constraints

The main issue for the system is latency, most game elements are dependent on low latency and will affect all interaction with the user controlled hero. The system therefore has to take into consideration that all network related code might need to be redesigned. The major problem with this is that it will not be possible to decide whether or not a network architecture is good enough will require manual testing. But the maximum latency is approximated to be 100ms in order to not affect the users experience of the game.

Hardware is not a big issue since the game will not demanding when it comes to processing power or hard drive space.

4 Graphical user interface

4.1 Lobby overview

The lobby interface allows interaction with other users. It's main purpose is allowing a user to create games and to join games created by other users. See Figure 6 to see how the different parts of the GUI are related. On some figures there are numbers which indicates which item on the given list that describes the numbered part of the GUI.

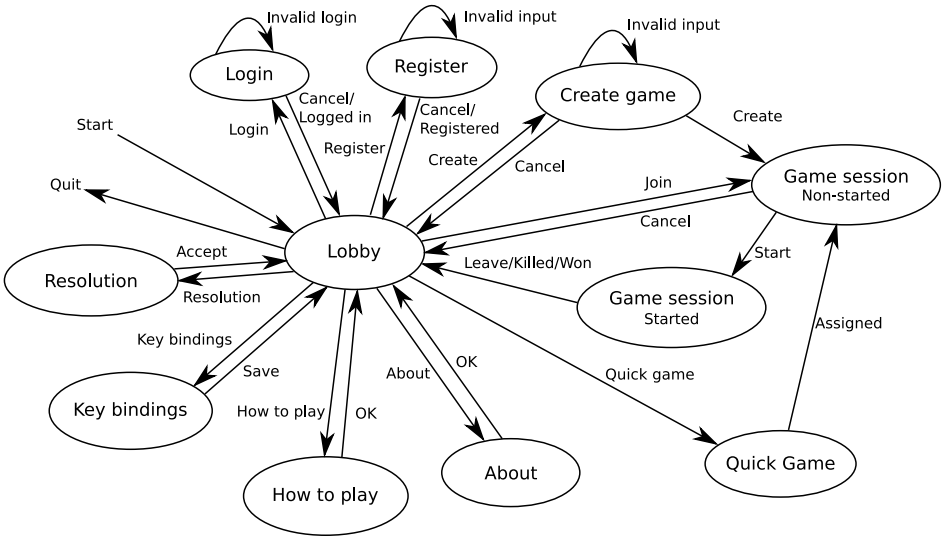


Figure 6: Picture displaying the displaying how the different states in the GUI are related

4.2 Ingame Interface

The in game GUI (Graphical User Interface) will consist of four major components, the canvas, the minimap, the building selection menu and the distribution menu.

The canvas is the part of the GUI that displays the game world. Using the canvas the user can interract with buildings, units and the hero. See Figure 7 for a description of the ingame GUI.

4.3 Requirement correspondance and functional description

1. **Construction menu (reqs. 20, 21)** The construction menu displays the set of buildings that the user can construct. The user can use the construction menu to initiate the construction of a building.

2. **Upgrading menu (req. 24)** Using the upgrading menu the user can upgrade a building that already has been built.

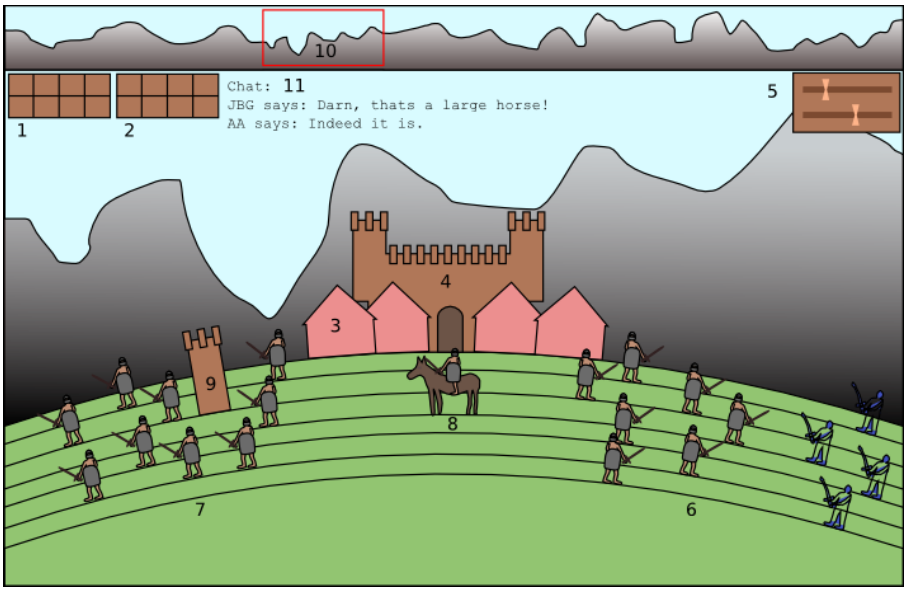


Figure 7: Picture displaying the ingame view

3. **Production building (req. 20)** A production building produces AI controlled units at a fixed rate.
4. **Castle (req. 19)** The castle allows the user to continue playing and resurrects a dead hero after a fixed amount of time.
5. **Distribution control (req. 31)** The distribution controls consists of a set of sliders, the sliders indicate the ratio of attacking and defending units for the corresponding unit type.
6. **Attacking units (reqs. 24)** AI controlled units that are on their way to attack the player to the left.
7. **Defending units (reqs. 24)** AI controlled units defending against the player to the right.
8. **Hero (req. 23)** The hero will react to user input and the changes will be visible in the canvas.
9. **Defensive tower (req. 21)** The defensive tower attacks hostile units that are within range.
10. **Minimap (req. 26)** The minimap displays a miniturized version of the world and is located at the top of the screen. The rectangular area in the minimap displays what part of the world that is drawn in the canvas.
11. **Chat (req. 22)** The chat displays messages from other users and can also be used to send messages

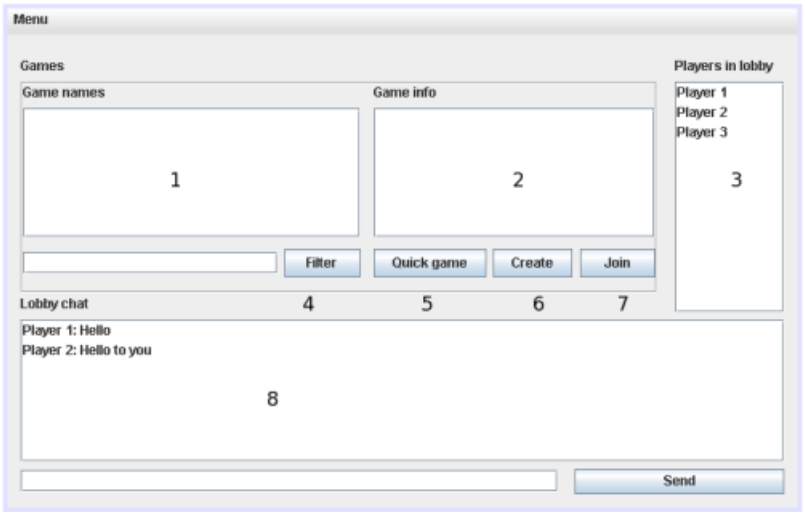


Figure 8: Picture displaying the lobby.

4.4 Lobby Interface

Lobby (usual)
The usual lobby, i.e. the first window the user will see when he starts the application can be seen in Figure 8.

1. **Current games (req. 12)** Lists the games that matches the current filter settings.
2. **Game info (req. 12)** Game information for the game selected in the current games list.
3. **Players in lobby (req. 7)** A list of all current players in the lobby.
4. **Filter (req. 12)** Passes filter settings to the game session filter and upon pressing the filter button the current filter settings from the filter text field that is located to the left of the filter button are applied to the game available.
5. **Quick game (req. 13)** Pressing the quick game button tells the game server to attempt to assign the player to a game session according to the formulas specified in the requiremnts document.
6. **Create (req. 8)** Sends the user to the create game form.
7. **Join (req. 11)** Joins the game selected in the current games list.
8. **Chat (req. 14)** Allows the user to communicate with other users in the lobby.

Lobby (joined game session)
When a user has joined a game session the lobby will be displayed as in Figure 9.

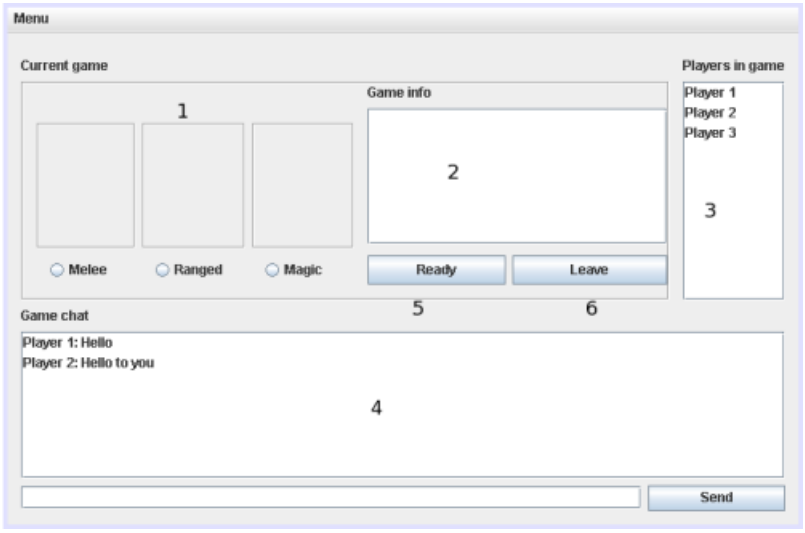


Figure 9: Picture displaying the GUI when a user has joined a game session.

1. **Hero type (req. 23)** By clicking in one of the radio buttons the user can choose between the three hero types.
2. **Game info (req. 12)** The current game settings are displayed to the users.
3. **Chat (req. 14)** The chat allows communication with other users in the game by entering a chat message in the textfield and pressing the send button.
4. **Players in lobby (req. 14)** Diplays the players that are currently in the game.
5. **Ready (req. 11)** The user signals to the others that he is ready to begin a game by pressing this button.
6. **Leave (req. 29)** Allows the user to leave a game session and return to the lobby.

Lobby (created game session)
The Lobby from the perspective of a user who has created a game session is displayed in Figure 10. The only difference from the joined game session is displayed below.

1. **Kick (req. 2)** Using this button the creator of the game kicks the selected player from the game. A player is selected by right clicking players name in the Players in the game list. A kicked player is returned to the lobby.

Create game dialog
The create game dialog is shown when the user wants to create a game and is described in Figure 11.

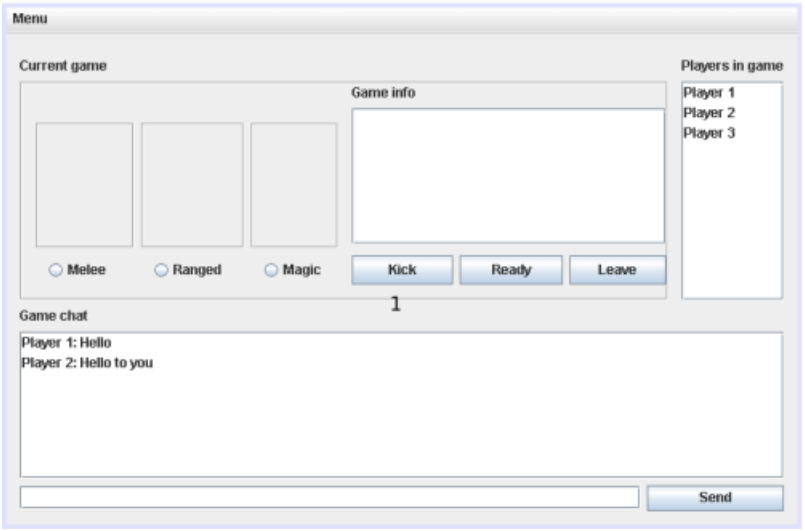


Figure 10: Picture displaying the GUI in a game session from the perspective of a user has created a game session.

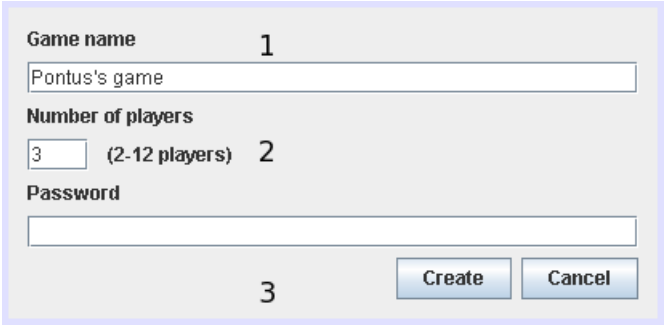


Figure 11: Picture displaying the create game dialog.

1. **Game name (reqs. 8, 12)** Specifies the game name.
2. **Number of players (reqs. 2, 9)** the player can set the amount of players for the game session within the given range using this text field.
3. **Password (req. 10)** the game session password will be the passphrase entered into this field. If left empty the game session will not require any passphrase.

Menu system
The menu system for the lobby will be structured as shown in Figure 12.

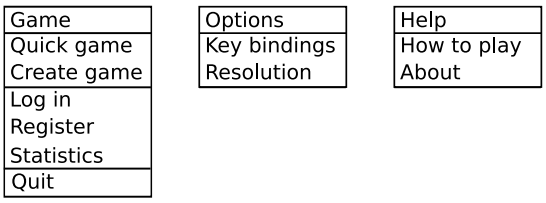


Figure 12: Picture displaying the menu system.

Key bindings dialog
In the key bindings dialog the user can alter his input settings to the game, Figure 13 shows how the dialog is structured.

1. **Key bindings (req. 3)** using this dialogue the user can change the key bindings. A binding is changed by clicking the respective button and pressing the key that the user wants to associate with the command.

How to play
The "How to play"-dialog displays information about how to play the game, the dialog is shown in Figure 14, notice however that the text in the figure is irrelevant. Describes how to play the game.

Login
The login dialog is shown when a user wants to look at his statistics (see Figure 15).

1. **Login name (req. 16)** In this field the user shall provide his username in order to log on to the game network.
2. **Password (req. 16)** The password corresponding to the username provided in the above form.

Register
The register dialog allows a user to keep track of his statistics (see Figure 16).

1. **Username (req. 15)** In this form the user can provide his username.

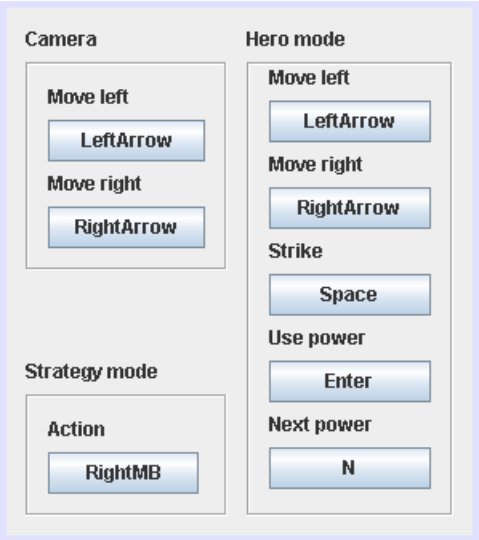


Figure 13: Picture displaying the key bindings dialog.

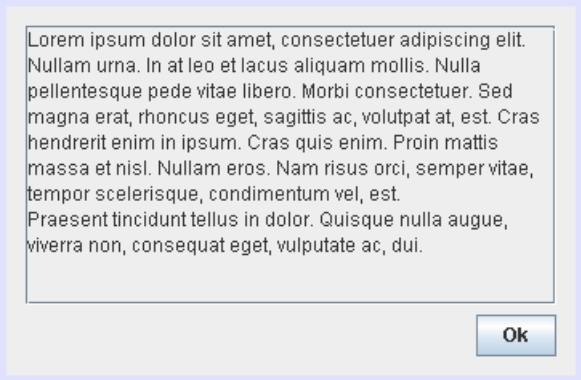


Figure 14: Picture displaying the how to play dialog.

2. **Password (req. 15)** In this form the user can provide his password.
3. **Password again (req. 15)** The user will have to supply his password two times in order to avoid misspelled passwords.

Resolution
The user will be shown a list of game resolutions that he can choose between (see Figure 17). The game resolution will come into effect when a game session is started.

Statistics
The user will be prestented with the statistics of his user in a dialog window (see Figure 18).

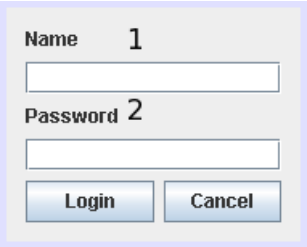


Figure 15: Picture displaying the login dialog.

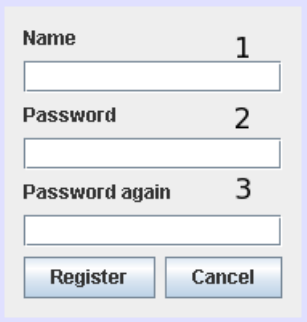


Figure 16: Picture displaying the register dialog.

Figure 17: Picture displaying the resolution dialog.

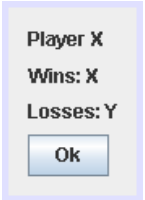


Figure 18: Picture displaying the statistics dialog.

5 Design Details

5.1 Class Responsibility Collaborator (CRC) Cards

5.1.1 Game Client

World	
Responsibilities	**Collaborators**
• Be the central repository for game data • Orders list of game elements depending on position	• Building • Game • Unit

Game	
Responsibilities	**Collaborators**
• Act as a controller for all other classes, tying the game client together • Detect collision • Execute game commands • Perform game logic	• Input • GameClientNetwork • Renderer • World

Input	
Responsibilities	**Collaborators**
• Handle input from user via mouse and keyboard	• Game

GameClientNetwork	
Responsibilities	**Collaborators**
• Act as an interface towards other game clients • Handle frame management	• Game • GameServerNetwork (via external network)

Building (Abstract class)	
Responsibilities	**Collaborators**
• Contain basic properties of a building • Hold hit points • Contain the visual representation of the building	• PolygonFace • Renderer • World

ProductionBuilding	
Responsibilities	**Collaborators**
• Handle the production rate of units	• Unit

DefensiveBuilding	
Responsibilities	**Collaborators**
• Contain logic for shooting at nearby units	• World • Unit

Castle	
Responsibilities	**Collaborators**
• Ensure that upon destruction Game is notified	• Game • World

Unit (Abstract class)	
Responsibilities	**Collaborators**
• Contain base unit properties (hit points etc.) • Contain basic functionality for unit movement • Contain visual representation of the unit • Contain unit AI	• Building • Renderer • PolygonFace • World

Archer	
Responsibilities	**Collaborators**
• Contain logic for shooting at nearby units	• Building • Renderer • PolygonFace • World

Swordsman	
Responsibilities	**Collaborators**
• Contain logic for attacking nearby units	• Building • Renderer • PolygonFace • World

Hero	
Responsibilities	**Collaborators**
• Hero logic for movement and Hero powers	• Building • Texture • PolygonFace • World

Renderer	
Responsibilities	**Collaborators**
• Render game elements to the screen upon request	• Unit • Building • Hero

PolygonFace	
Responsibilities	**Collaborators**
• Act as a self contained polygon with a texture that can be rendered by the renderer	• Texture

Texture	
Responsibilities	**Collaborators**
• Holds a picture that can be rendered onto a polygon.	• Renderer • PolygonFace

5.1.2 Game Server

NetworkCommand	
Responsibilities	**Collaborators**
• Contain basic functionality of a network command	• None

NetworkGameCommand	
Responsibilities	**Collaborators**
• Contain overall game client data, i.e. loosing	• None

NetworkInteractionCommand	
Responsibilities	**Collaborators**
• Contains data about different inputs of the clients	• None

FrameEndCommand	
Responsibilities	**Collaborators**
• Mark the end of a frame	• None

5.1.3 Game Server

CommandFilter	
Responsibilities	**Collaborators**
• Divide incoming commands from the game server network into two categories, NetworkGameCommands and NetworkInteractionCommands	• ClientController • NetworkGameCommand • NetworkInteractionCommand • Proxy

ClientController	
Responsibilities	**Collaborators**
• Handle NetworkGameCommands, i.e. client progress	• NetworkGameCommand

Proxy	
Responsibilities	**Collaborators**
• Handle frame management, associates incomming commands to the correct frame	• CommandFilter • NetworkInteractionCommand

5.1.4 Lobby Client

LobbyGUI	
Responsibilities • Handle input from the user • Chatting • Displaying available games • Logic for joining and creating games	**Collaborators** • GameClientNetwork

GameServerNetwork	
Responsibilities • Handles commands coming from the client and sends commands to the client	**Collaborators** • GameClientNetwork (via external network)

5.1.5 Lobby Server

LobbyServer	
Responsibilities • Handles games and game servers • Handles user that want to join and create games	**Collaborators** • GameServerNetwork

5.2 Class Diagram

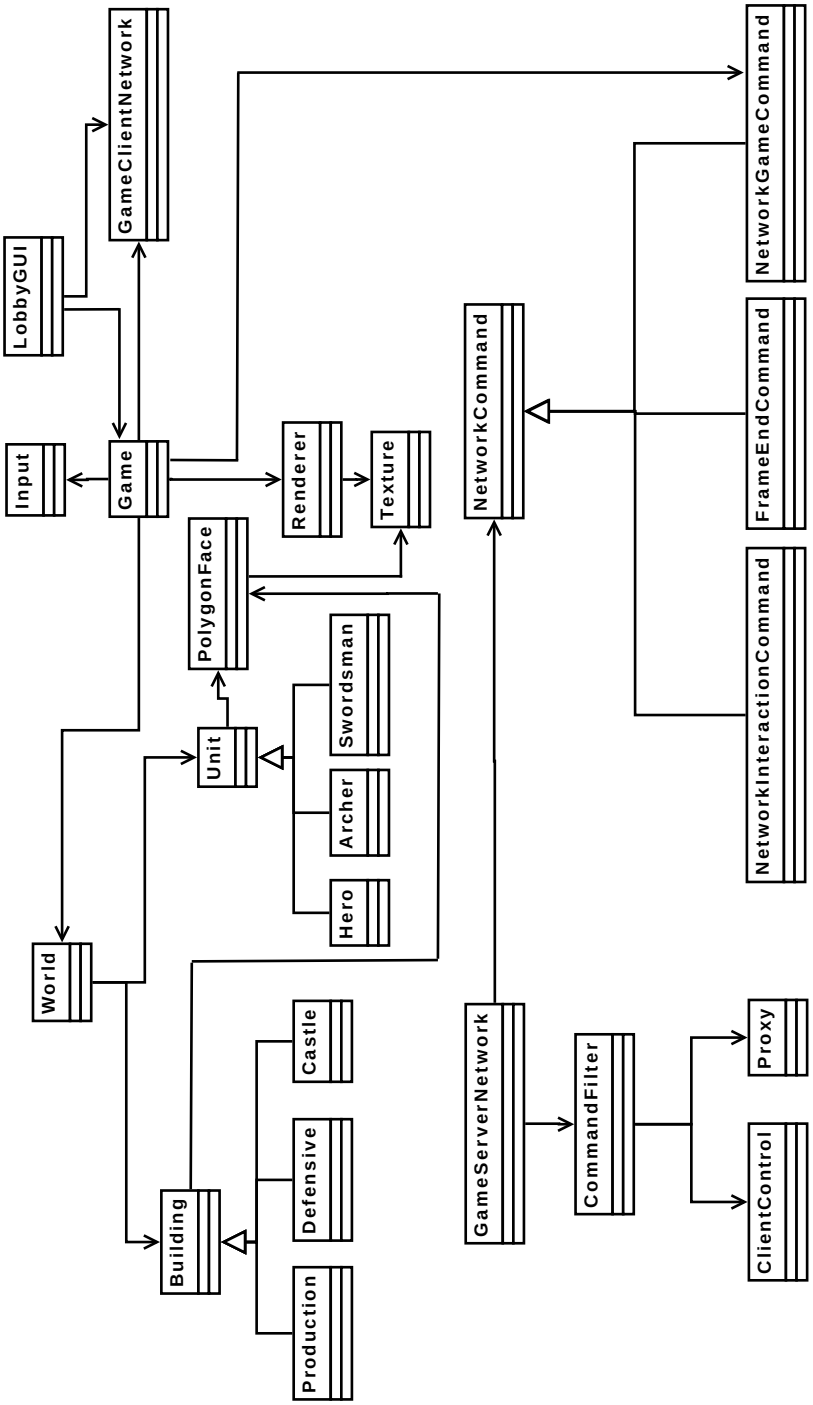


Figure 19: Picture describing the overall class heirarchy.

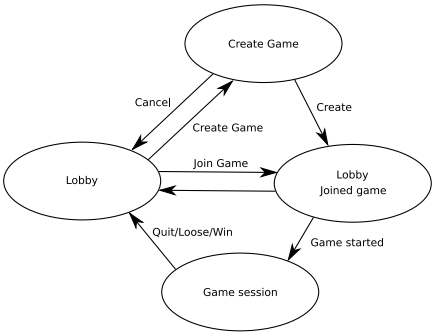


Figure 20: Picture describing the overall transitions in the game.

5.3 State Charts

See figure 20.

5.4 Interaction Diagrams

See figure 21 and 22.

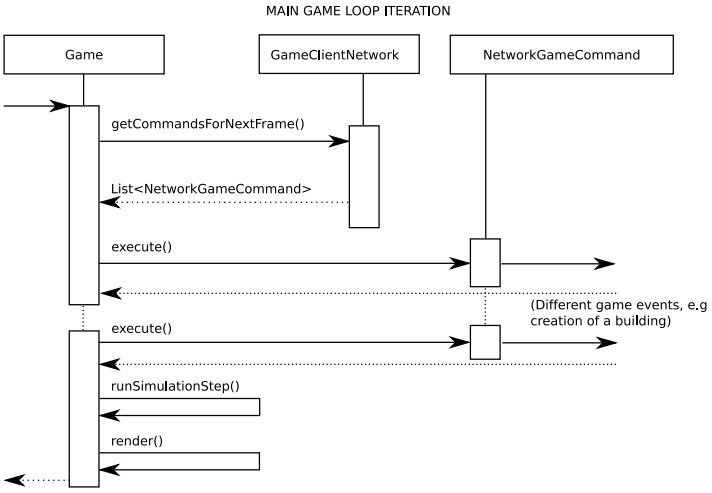


Figure 21: Picture describing the flow in the main game loop.

5.5 Detailed Design

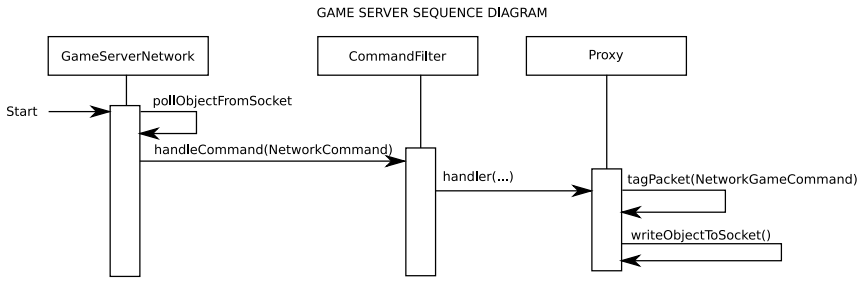
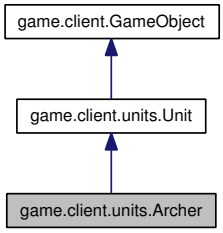


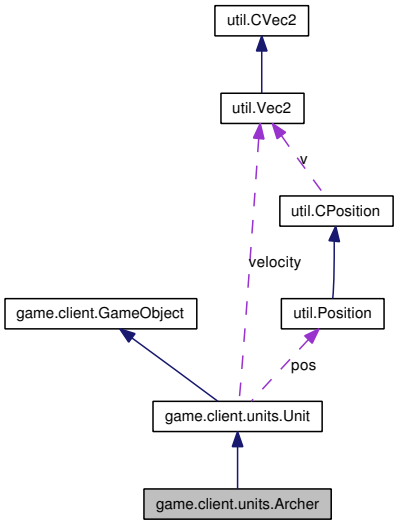
Figure 22: Picture describing the flow in the game server.

game.client.units.Archer Class Reference

Inheritance diagram for game.client.units.Archer:



Collaboration diagram for game.client.units.Archer:



Public Member Functions

- float **getShootingRange** ()
- void **damage** (int amount)
- void **destroy** ()
- boolean **isDead** ()
- boolean **shallBeRemoved** ()
- void **think** ()

Detailed Description

Is a ranged unit that shoots arrows.

Member Function Documentation

float game.client.units.Archer.getShootingRange ()

Returns how far the archer can shoot.

Returns:

The range the archer can shoot.

Precondition:

None.

Postcondition:

None.

void game.client.units.Archer.damage (int *amount*)

Removes the given amount of hitpoints from the unit.

Parameters:

amount Amount of damage to subtract from health.

Precondition:

None

Postcondition:

The entered amount of damage has been subtracted from the objects health or it's health is zero.

Reimplemented from **game.client.units.Unit** (p. 88).

void game.client.units.Archer.destroy ()

Removes the object from the game world.

Precondition:

None.

Postcondition:

The object is removed from the game world.

Reimplemented from **game.client.units.Unit** (p. 88).

boolean game.client.units.Archer.isDead ()

Returns true if the object is considered to be dead.

Precondition:

None

Postcondition:

None

Reimplemented from **game.client.units.Unit** (p. 89).

boolean game.client.units.Archer.shallBeRemoved ()

Returns true if the object should be removed from the game.

Returns:

True if the object is scheduled for removal

Precondition:

None

Postcondition:

None

Reimplemented from **game.client.units.Unit** (p. 89).

void game.client.units.Archer.think ()

Performs the actions of a object that should be executed every frame.

Precondition:

The object is not dead.

Postcondition:

The object has performed it's actions for this frame.

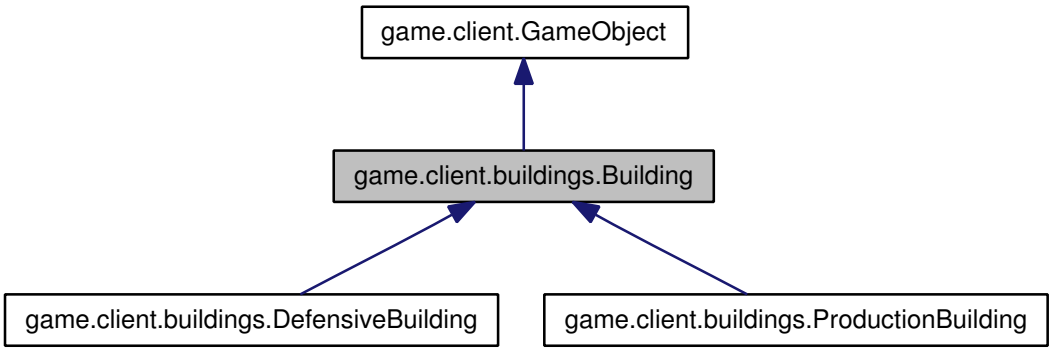
Reimplemented from **game.client.units.Unit** (p. 89).

The documentation for this class was generated from the following file:

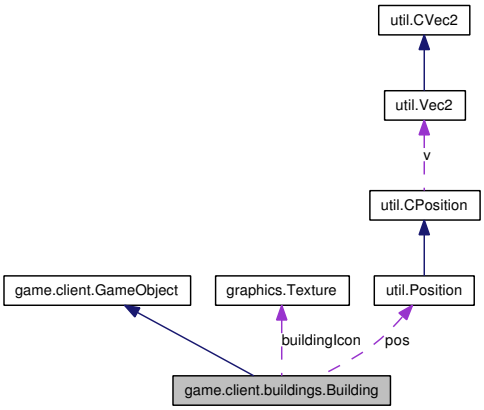
- game/client/units/Archer.java

game.client.buildings.Building Class Reference

Inheritance diagram for game.client.buildings.Building:



Collaboration diagram for game.client.buildings.Building:



Public Member Functions

- final **CPosition getPos** ()
- final void **setPos** (**CPosition** pos)
- **CAxisAlignedRect getBoundingBox** ()
- void **damage** (int amount)
- void **destroy** ()
- boolean **isDead** ()
- boolean **shallBeRemoved** ()
- void **think** ()

Detailed Description

An abstract class describing behaviour of abuilding.

Member Function Documentation

final CPosition game.client.buildings.Building.getPos ()

Returns the position of the unit.

Returns:

The position of the unit.

Precondition:

None.

Postcondition:

None.

Implements **game.client.GameObject** (p. 56).

final void game.client.buildings.Building.setPos (CPosition *pos*)

Sets the position of the unit in the game world.

Parameters:

pos Pos to set.

Precondition:

None.

Postcondition:

The given position has been set.

Implements **game.client.GameObject** (p. 56).

CAxisAlignedRect game.client.buildings.Building.getBoundingBox ()

The building is within the returned bouiding box.

Returns:

A bounding box for the building.

Precondition:

None.

Postcondition:

None.

Implements **game.client.GameObject** (p. 56).

void game.client.buildings.Building.damage (int *amount*)

Removes the given amount of hit points from the building.

Parameters:

amount Amount of damage to subtract from health.

Precondition:

None

Postcondition:

The entered amount of damage has been subtracted from the objects health or it's health is zero.

Implements **game.client.GameObject** (p. 55).

Reimplemented in **game.client.buildings.DefensiveBuilding** (p. 48).

void game.client.buildings.Building.destroy ()

Shall remove the object from the game world.

Precondition:

None.

Postcondition:

The object is removed from the game world.

Implements **game.client.GameObject** (p. 55).

Reimplemented in **game.client.buildings.DefensiveBuilding** (p. 48), and **game.client.buildings.ProductionBuilding** (p. 78).

boolean game.client.buildings.Building.isDead ()

Returns true if the object is considered to be dead.

Precondition:

None

Postcondition:

None

Implements **game.client.GameObject** (p. 55).

Reimplemented in **game.client.buildings.DefensiveBuilding** (p. 48).

boolean game.client.buildings.Building.shallBeRemoved ()

Returns true if the object should be removed from the game.

Returns:

True if the object is scheduled for removal

Precondition:

None

Postcondition:

None

Implements **game.client.GameObject** (p. 54).

Reimplemented in **game.client.buildings.DefensiveBuilding** (p. 49).

void game.client.buildings.Building.think ()

Shall perform the actions of a object that should be executed every frame.

Precondition:

The object is not dead.

Postcondition:

The object has performed it's actions for this frame.

Implements **game.client.GameObject** (p. 55).

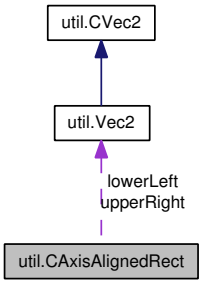
Reimplemented in **game.client.buildings.DefensiveBuilding** (p. 49), and **game.client.buildings.ProductionBuilding** (p. 78).

The documentation for this class was generated from the following file:

- game/client/buildings/Building.java

util.CAxisAlignedRect Class Reference

Collaboration diagram for util.CAxisAlignedRect:



Public Member Functions

- **CAxisAlignedRect** (**Vec2** v, float width, float height)
- boolean **isPointIn** (**Vec2** v)

Detailed Description

A bounding box for a game object.

Constructor & Destructor Documentation

util.CAxisAlignedRect.CAxisAlignedRect (Vec2 *v*, float *width*, float *height*)

Parameters:

v The point of the lower left corner of the bounding box.

width The width of the bounding box.

height The height of the bounding box.

Member Function Documentation

boolean util.CAxisAlignedRect.isPointIn (Vec2 *v*)

Checks wether a point is in the bounding box.

Parameters:

v Any point in the world.

Returns:

true if a point is inside the bounding box.

Precondition:

None

Postcondition:

Class data is unchanged.

The documentation for this class was generated from the following file:

- util/CAxisAlignedRect.java

game.server.Client Class Reference

Public Member Functions

- int **getID** ()

Detailed Description

Holds data related to a **Client** (p. 37).

Member Function Documentation

int game.server.Client.getID ()

Returns the ID of a client.

Precondition:

None.

Postcondition:

None.

The documentation for this class was generated from the following file:

- game/server/Client.java

game.server.ClientController Class Reference

Public Member Functions

- void **processCommand** (**NetworkInteractionCommand** cmd)

Detailed Description

Handles client management.

Author:

ninjin

Member Function Documentation

void game.server.ClientController.processCommand (NetworkInteractionCommand *cmd*)

Processes network interaction commands.

Parameters:

cmd The command to process

Precondition:

None.

Postcondition:

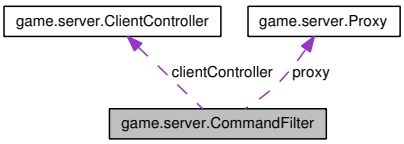
The command has been processed.

The documentation for this class was generated from the following file:

- game/server/ClientController.java

game.server.CommandFilter Class Reference

Collaboration diagram for game.server.CommandFilter:



Public Member Functions

- **CommandFilter** ()
- void **filter** (**NetworkCommand** cmd)

Detailed Description

Holds the client controller and proxy and is also responsible for distributing network commands between them.

Constructor & Destructor Documentation

game.server.CommandFilter.CommandFilter ()

Precondition:

None.

Postcondition:

None.

Member Function Documentation

void game.server.CommandFilter.filter (NetworkCommand *cmd*)

Distributes game commands and network interaction commands to the client controller and proxy respectively.

Parameters:

cmd Command to be filtered.

Precondition:

None.

Postcondition:

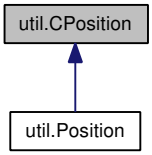
The given command has been handed to the correct handler.

The documentation for this class was generated from the following file:

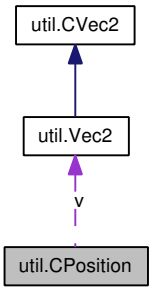
- game/server/CommandFilter.java

util.CPosition Class Reference

Inheritance diagram for util.CPosition:



Collaboration diagram for util.CPosition:



Public Member Functions

- final float **x** ()
- final float **y** ()
- final **CVec2 combatPos** ()
- final int **track** ()
- **Position copy** ()

Package Attributes

- **Vec2 v**
- int **track**

Detailed Description

A constant position in the game.

Member Function Documentation

final float util.CPosition.x ()

Returns the x coordinate.

Precondition:

None

Postcondition:

Class data is unchanged.

final float util.CPosition.y ()

Returns the y coordinate.

Precondition:

None

Postcondition:

Class data is unchanged.

final CVec2 util.CPosition.combatPos ()

Returns the position in the combat plane.

Precondition:

None

Postcondition:

Class data is unchanged.

Returns:

The position in the combat plane.

final int util.CPosition.track ()

Returns the game track of the object.

Precondition:

None

Postcondition:

Class data is unchanged.

Position util.CPosition.copy ()

Returns a copy of the position.

Precondition:

None

Postcondition:

The position reference returned shall not be able to change the class data.

Member Data Documentation

Vec2 util.CPosition.v `[package]`

Position (p. 74) in the 2d plane.

int util.CPosition.track `[package]`

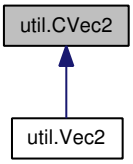
Game track.

The documentation for this class was generated from the following file:

- util/CPosition.java

util.CVec2 Class Reference

Inheritance diagram for util.CVec2:



Public Member Functions

- final float **x** ()
- final float **y** ()
- **Vec2 copy** ()

Detailed Description

A constant vector with two floats. Usualy used to represent a position in the a combat position plane. A combat position plane is a plane that a whole trace lies in.

Member Function Documentation

final float util.CVec2.x ()

Returns the x cordinate.

Precondition:

None

Postcondition:

Class data is unchanged.

final float util.CVec2.y ()

Returns the y cordinate.

Precondition:

None

Postcondition:

Class data is unchanged.

Vec2 util.CVec2.copy ()

Returns a copy of the class.

Precondition:

None

Postcondition:

The position reference returned shall not be able to change the class data.

The documentation for this class was generated from the following file:

- util/CVec2.java

lobby.server.DatabaseConnection Class Reference

Public Member Functions

- **DatabaseConnection** ()
- void **insert** (String query)
- String **select** (String query)

Detailed Description

Handles the **Lobby** (p. 66) connection to the database.

Constructor & Destructor Documentation

lobby.server.DatabaseConnection.DatabaseConnection ()

Precondition:

The database server is online

Postcondition:

The database connection is availible.

Member Function Documentation

void lobby.server.DatabaseConnection.insert (String *query*)

Inserts the given query into the database.

Parameters:

query The query to be executed

Precondition:

The database connection is availible

Postcondition:

The query has been executed on the database server.

String lobby.server.DatabaseConnection.select (String *query*)

Returns the result of the given database query.

Parameters:

query The query to be executed

Returns:

The result of the query

Precondition:

The database connection is availible

Postcondition:

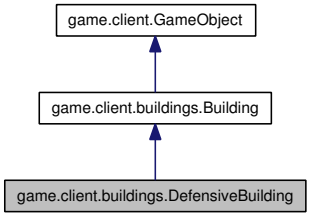
The query has been executed on the database server and the result is returned

The documentation for this class was generated from the following file:

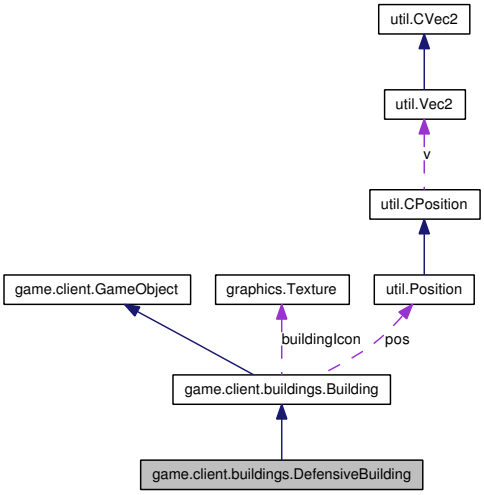
- lobby/server/DatabaseConnection.java

game.client.buildings.DefensiveBuilding Class Reference

Inheritance diagram for game.client.buildings.DefensiveBuilding:



Collaboration diagram for game.client.buildings.DefensiveBuilding:



Public Member Functions

- float **getShootingRange** ()
- void **damage** (int amount)
- void **destroy** ()
- boolean **isDead** ()
- boolean **shallBeRemoved** ()
- void **think** ()

Detailed Description

The super class for the production buildings. A production building produces moving units.

Member Function Documentation

float game.client.buildings.DefensiveBuilding.getShootingRange ()

Returns the distance the defensive building can shoot.

Returns:

The distance the defensive building can shoot.

Precondition:

None.

Postcondition:

None.

void game.client.buildings.DefensiveBuilding.damage (int *amount*)

Removes the given amount of hit points from the building.

Parameters:

amount Amount of damage to subtract from health.

Precondition:

None

Postcondition:

The entered amount of damage has been subtracted from the objects health or it's health is zero.

Reimplemented from **game.client.buildings.Building** (p. 33).

void game.client.buildings.DefensiveBuilding.destroy ()

Shall remove the object from the game world.

Precondition:

None.

Postcondition:

The object is removed from the game world.

Reimplemented from **game.client.buildings.Building** (p. 33).

boolean game.client.buildings.DefensiveBuilding.isDead ()

Returns true if the object is considered to be dead.

Precondition:

None

Postcondition:

None

Reimplemented from **game.client.buildings.Building** (p. 33).

boolean game.client.buildings.DefensiveBuilding.shallBeRemoved ()

Returns true if the object should be removed from the game.

Returns:

True if the object is scheduled for removal

Precondition:

None

Postcondition:

None

Reimplemented from **game.client.buildings.Building** (p. 33).

void game.client.buildings.DefensiveBuilding.think ()

Shall perform the actions of a object that should be executed every frame.

Precondition:

The object is not dead.

Postcondition:

The object has performed it's actions for this frame.

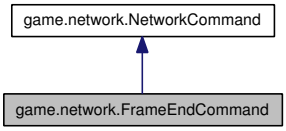
Reimplemented from **game.client.buildings.Building** (p. 34).

The documentation for this class was generated from the following file:

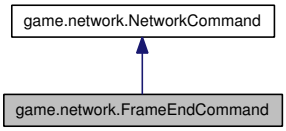
- game/client/buildings/DefensiveBuilding.java

game.network.FrameEndCommand Class Reference

Inheritance diagram for game.network.FrameEndCommand:



Collaboration diagram for game.network.FrameEndCommand:



Public Member Functions

- **FrameEndCommand** (int frameID)
- int **getFrameID** ()

Detailed Description

Represents the end of a frame.

Constructor & Destructor Documentation

game.network.FrameEndCommand.FrameEndCommand (int *frameID*)

See also:

NetworkCommand (p. 70)

Member Function Documentation

int game.network.FrameEndCommand.getFrameID ()

See also:

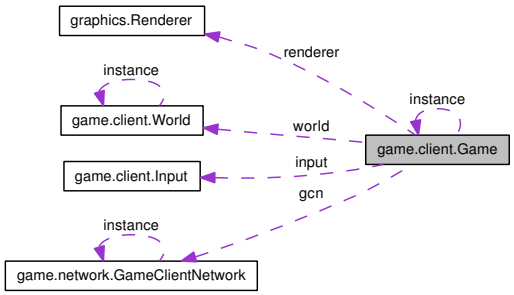
NetworkCommand.getFrameId()

The documentation for this class was generated from the following file:

- game/network/FrameEndCommand.java

game.client.Game Class Reference

Collaboration diagram for game.client.Game:



Static Public Member Functions

- static final **Game ins** ()

Detailed Description

Main class of the client.

Member Function Documentation

static final Game game.client.Game.ins () `[static]`

Creates a singleton instance of game.

Returns:

the static reference to **Game** (p. 51).

Precondition:

none

Postcondition:

An instance of **Game** (p. 51) has been created.

The documentation for this class was generated from the following file:

- game/client/Game.java

game.network.GameClientNetwork Class Reference

Collaboration diagram for game.network.GameClientNetwork:

Public Member Functions

- abstract void **sendGameCommand** (**NetworkGameCommand** c)
- abstract void **sendFrameEndCommand** ()
- abstract Collection< **NetworkGameCommand** > **popCommandsForCurrentFrame** ()

Static Public Member Functions

- static **GameClientNetwork ins** ()
- static void **setInstance** (**GameClientNetwork** instance)

Detailed Description

The client network class. When we test **`ins()`** (p. 52) will probably return a instance of a class that will emulate a GameServerNetwork.

Author:

mandermo

Member Function Documentation

static GameClientNetwork game.network.GameClientNetwork.ins () `[static]`

Return the "singleton instance"

Returns:

static void game.network.GameClientNetwork.setInstance (GameClientNetwork *instance*) `[static]`

Sets the "singleton" instance.

Parameters:

instance

abstract void game.network.GameClientNetwork.sendGameCommand (NetworkGameCommand *c*) `[pure virtual]`

Sends a game command that will be replicated to all clients and scheduled to run at the same frame on all clients.

Parameters:

c The command to be sent.

Precondition:

None

Postcondition:

The command was sent.

abstract void game.network.GameClientNetwork.sendFrameEndCommand () `[pure virtual]`

Called to signal all commands for current frame is sent.

Precondition:

None

Postcondition:

The frame ends.

abstract Collection<NetworkGameCommand> game.network.GameClientNetwork.popCommandsForCurrentFrame () `[pure virtual]`

Called to get all commands that should be executed on this frame. Will pop those commands from the network.

Returns:

The commands to be executed this frame

Precondition:

The frame has ended.

Postcondition:

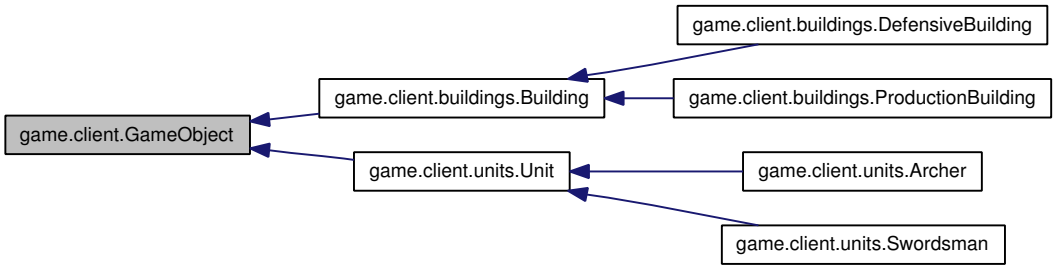
The commands for the current frame has been returned and removed from the queue.

The documentation for this class was generated from the following file:

- game/network/GameClientNetwork.java

game.client.GameObject Interface Reference

Inheritance diagram for game.client.GameObject:



Public Member Functions

- boolean **shallBeRemoved** ()
- void **destroy** ()
- boolean **isDead** ()
- void **think** ()
- void **damage** (int amount)
- **CPosition getPos** ()
- void **setPos** (**CPosition** pos)
- **CAxisAlignedRect getBoundingBox** ()

Detailed Description

An interface that every object that can perform some kind of logic shall implement.

Member Function Documentation

boolean game.client.GameObject.shallBeRemoved ()

Returns true if the object should be removed from the game.

Returns:

True if the object is scheduled for removal

Precondition:

None

Postcondition:

None

Implemented in **game.client.buildings.Building** (p. 33), **game.client.buildings.DefensiveBuilding** (p. 49), **game.client.units.Archer** (p. 30), **game.client.units.Swordsman** (p. 83), and **game.client.units.Unit** (p. 89).

void game.client.GameObject.destroy ()

Shall remove the object from the game world.

Precondition:

None.

Postcondition:

The object is removed from the game world.

Implemented in **game.client.buildings.Building** (p. 33), **game.client.buildings.DefensiveBuilding** (p. 48), **game.client.buildings.ProductionBuilding** (p. 78), **game.client.units.Archer** (p. 29), **game.client.units.Swordsman** (p. 83), and **game.client.units.Unit** (p. 88).

boolean game.client.GameObject.isDead ()

Returns true if the object is considered to be dead.

Precondition:

None

Postcondition:

None

Implemented in **game.client.buildings.Building** (p. 33), **game.client.buildings.DefensiveBuilding** (p. 48), **game.client.units.Archer** (p. 29), **game.client.units.Swordsman** (p. 83), and **game.client.units.Unit** (p. 89).

void game.client.GameObject.think ()

Shall perform the actions of a object that should be executed every frame.

Precondition:

The object is not dead.

Postcondition:

The object has performed it's actions for this frame.

Implemented in **game.client.buildings.Building** (p. 34), **game.client.buildings.DefensiveBuilding** (p. 49), **game.client.buildings.ProductionBuilding** (p. 78), **game.client.units.Archer** (p. 30), **game.client.units.Swordsman** (p. 84), and **game.client.units.Unit** (p. 89).

void game.client.GameObject.damage (int *amount*)

It shall implement a function that removes hit points from the object.

Parameters:

amount Amount of damage to subtract from health.

Precondition:

None

Postcondition:

The entered amount of damage has been subtracted from the objects health or it's health is zero.

Implemented in **game.client.buildings.Building** (p. 33), **game.client.buildings.DefensiveBuilding** (p. 48), **game.client.units.Archer** (p. 29), **game.client.units.Swordsman** (p. 82), and **game.client.units.Unit** (p. 88).

CPosition game.client.GameObject.getPos ()

Returns the position of the unit relative to the lower left corner.

Returns:

The position of the unit relative to the lower left corner.

Precondition:

None

Postcondition:

None

Implemented in **game.client.buildings.Building** (p. 32), and **game.client.units.Unit** (p. 87).

void game.client.GameObject.setPos (CPosition *pos*)

Sets the position of the unit in the game world.

Parameters:

pos The position to set the object to.

Precondition:

None.

Postcondition:

The postion of the unit is now the given position.

Implemented in **game.client.buildings.Building** (p. 32), and **game.client.units.Unit** (p. 87).

CAxisAlignedRect game.client.GameObject.getBoundingBox ()

Returns the bounding box of a game object inside the world.

Returns:

A box that bounds the object.

Precondition:

None.

Postcondition:

None.

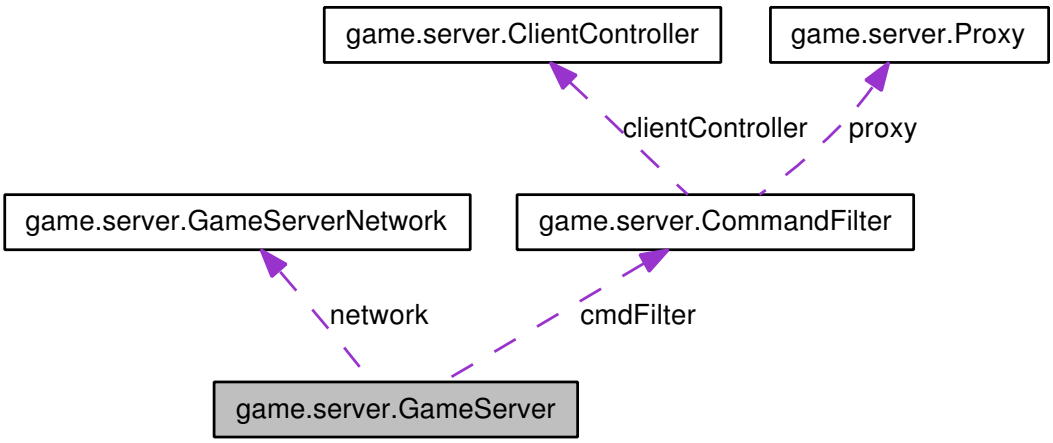
Implemented in **game.client.buildings.Building** (p. 32), and **game.client.units.Unit** (p. 88).

The documentation for this interface was generated from the following file:

- game/client/GameObject.java

game.server.GameServer Class Reference

Collaboration diagram for game.server.GameServer:



Public Member Functions

- **GameServer** (ArrayList< **Client** > clients, int port)
- void **run** ()

Detailed Description

Responsible for setting up and handling the game server.

Constructor & Destructor Documentation

game.server.GameServer.GameServer (ArrayList< Client > *clients*, int *port*)

Parameters:

clients The clients to which this game server will be assigned.

port The port which the game server is to listen to.

Precondition:

None.

Postcondition:

None.

Member Function Documentation

void game.server.GameServer.run ()

Listens on the given port and verifies that the given client id is correct.

Precondition:

None.

Postcondition:

None.

The documentation for this class was generated from the following file:

- game/server/GameServer.java

lobby.server.GameServerHandler Class Reference

Public Member Functions

- **GameServerHandler** ()
- void **startGameServer** (ArrayList< **Client** > clients, int port)
- void **run** ()

Detailed Description

Handles the collection of GameServers that are currently running or that have finished their games and needs to be reaped.

Constructor & Destructor Documentation

lobby.server.GameServerHandler.GameServerHandler ()

Precondition:

None

Postcondition:

The **GameServerHandler** (p. 60) is ready to run.

Member Function Documentation

void lobby.server.GameServerHandler.startGameServer (ArrayList< Client > *clients*, int *port*)

Starts a new game server for the given clients.

Parameters:

clients The clients that should be connected to the server

port The port the server shall listen on

Precondition:

The port is unused.

Postcondition:

The GameServer listens on the specified port and accepts the specified clients.

void lobby.server.GameServerHandler.run ()

Iterates over the game servers and checks their status.

Precondition:

None

Postcondition:

None

The documentation for this class was generated from the following file:

- lobby/server/GameServerHandler.java

game.server.GameServerNetwork Class Reference

Public Member Functions

- **GameServerNetwork** (int port)
- void **send** (**NetworkCommand** cmd)

Package Attributes

- ArrayList< Socket > **clients**

Detailed Description

Handles network connections with the clients.

Constructor & Destructor Documentation

game.server.GameServerNetwork.GameServerNetwork (int *port*)

Parameters:

port The port which this game server network listens to.

Precondition:

None.

Postcondition:

None.

Member Function Documentation

void game.server.GameServerNetwork.send (NetworkCommand *cmd*)

Sends a given command.

Parameters:

cmd The command to send.

Precondition:

None.

Postcondition:

The command has been sent to all connected Clients.

Member Data Documentation

ArrayList<Socket> game.server.GameServerNetwork.clients `[package]`

A list of sockets to the clients.

The documentation for this class was generated from the following file:

- game/server/GameServerNetwork.java

game.client.Input Class Reference

Public Member Functions

- boolean **isLeftMouseButtonClicked** ()
- boolean **isRightMouseButtonClicked** ()
- ArrayList **keysPressed** ()

Detailed Description

Is a wrapper class to JInput devices.

Member Function Documentation

boolean game.client.Input.isLeftMouseButtonClicked ()

Resets event data. The input gotten after All input gotten after the reset is Returns true if the left mouse button is clicked this frame.

Returns:

True if the left mouse button is clicked.

Precondition:

None.

Postcondition:

None.

boolean game.client.Input.isRightMouseButtonClicked ()

Returns true if the right mouse button is clicked this frame.

Returns:

True if the right mouse button is clicked.

Precondition:

None.

Postcondition:

None.

ArrayList game.client.Input.keysPressed ()

Returns a list of keys pressed.

Returns:

An ArrayList containing the keys pressed.

Precondition:

None.

Postcondition:

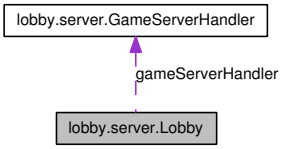
None.

The documentation for this class was generated from the following file:

- game/client/Input.java

lobby.server.Lobby Class Reference

Collaboration diagram for lobby.server.Lobby:



Public Member Functions

- **Lobby** ()
- void **run** ()

Detailed Description

Responsible for managing the game lobby.

Constructor & Destructor Documentation

lobby.server.Lobby.Lobby ()

Precondition:

None

Postcondition:

The lobby server is ready to run.

Member Function Documentation

void lobby.server.Lobby.run ()

Continously pushes and processes commands from the clients. The commands contain the current data for the lobby.

Precondition:

None

Postcondition:

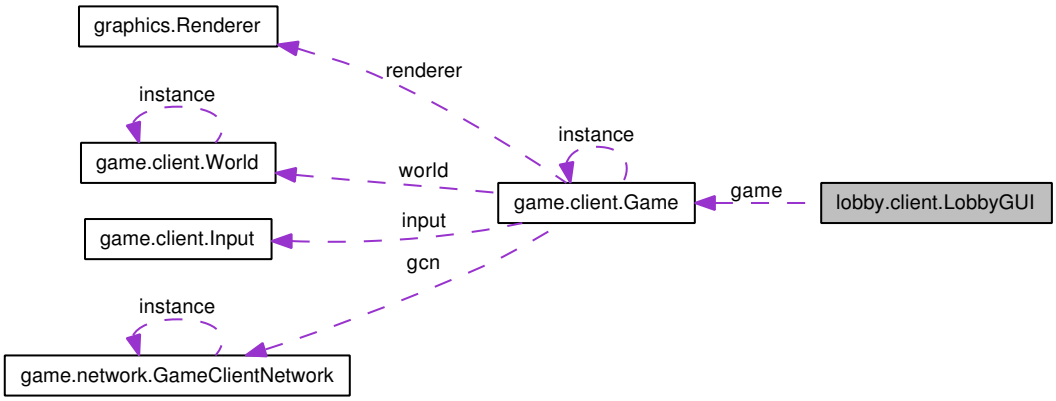
None

The documentation for this class was generated from the following file:

- lobby/server/Lobby.java

lobby.client.LobbyGUI Class Reference

Collaboration diagram for lobby.client.LobbyGUI:



Public Member Functions

- **LobbyGUI** ()

Detailed Description

Handles the **LobbyGUI** (p. 67) using Swing. Also sets up the game object.

Constructor & Destructor Documentation

lobby.client.LobbyGUI.LobbyGUI ()

Sets up the full-fledged GUI using Swing.

Precondition:

None

Postcondition:

The GUI is displayed.

The documentation for this class was generated from the following file:

- lobby/client/LobbyGUI.java

lobby.server.LobbyServerNetwork Class Reference

Public Member Functions

- **LobbyServerNetwork** (int **port**)
- **NetworkCommand recieve** ()
- void **broadcast** (**NetworkCommand** cmd)

Package Attributes

- int **port**
- ArrayList< Socket > **clients**

Detailed Description

Responsible for the lobby network connection.

Constructor & Destructor Documentation

lobby.server.LobbyServerNetwork.LobbyServerNetwork (int *port*)

Parameters:

port The port to listen on

Precondition:

The port to listen on is unused.

Postcondition:

The object is ready to send and recieve messages.

Member Function Documentation

NetworkCommand lobby.server.LobbyServerNetwork.recieve ()

Returns a recieved NetworkCommand

Returns:

A NetworkCommand recieved on the network.

Precondition:

There is a network connection.

Postcondition:

The

void lobby.server.LobbyServerNetwork.broadcast (NetworkCommand *cmd*)

Broadcasts the given command to all connected clients.

Parameters:

cmd The command to send

Precondition:

There is a network connection.

Postcondition:

The message was sent to all clients.

Member Data Documentation

int lobby.server.LobbyServerNetwork.port `[package]`

Connection port number.

ArrayList<Socket> lobby.server.LobbyServerNetwork.clients `[package]`

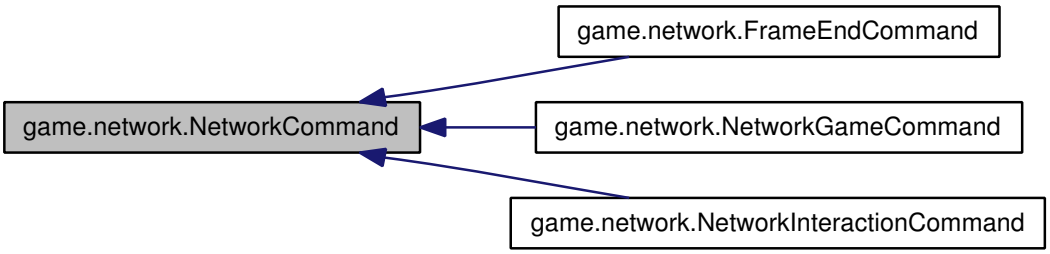
A list of connected clients.

The documentation for this class was generated from the following file:

- lobby/server/LobbyServerNetwork.java

game.network.NetworkCommand Class Reference

Inheritance diagram for game.network.NetworkCommand:



Detailed Description

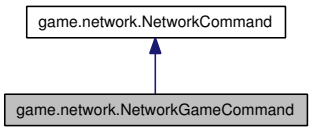
An abstract description of commands sent of the network.

The documentation for this class was generated from the following file:

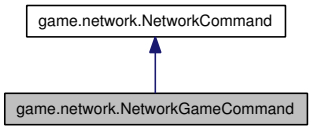
- game/network/NetworkCommand.java

game.network.NetworkGameCommand Class Reference

Inheritance diagram for game.network.NetworkGameCommand:



Collaboration diagram for game.network.NetworkGameCommand:



Public Member Functions

- final int **getSentFrame** ()
- final void **setSentFrame** (int sentFrame)
- final int **getExecuteFrame** ()
- final void **setExecuteFrame** (int x)
- abstract void **execute** ()

Detailed Description

A command representing actions during the game play.

Member Function Documentation

final int game.network.NetworkGameCommand.getSentFrame ()

Returns in which frame the command was sent.

Precondition:

The frame sent id is set

Postcondition:

The frame sent id of the command is returned

final void game.network.NetworkGameCommand.setSentFrame (int *sentFrame*)

Sets the sent frame of the command.

Parameters:

sentFrame The frame sent id to set

Precondition:

None

Postcondition:

The frame sent id is set.

final int game.network.NetworkGameCommand.getExecuteFrame ()

Returns the frame in which the command should be executed.

Precondition:

The frame is set

Postcondition:

The frame id of the command is returned

final void game.network.NetworkGameCommand.setExecuteFrame (int *x*)

Sets the frame in which the command should be executed.

Parameters:

x The frame id to set

Precondition:

None

Postcondition:

The execute frame is set.

abstract void game.network.NetworkGameCommand.execute () `[pure virtual]`

Shall implement the execution of a network command.

Precondition:

The command is to be executed in the correct frame.

Postcondition:

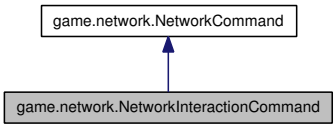
The command was executed.

The documentation for this class was generated from the following file:

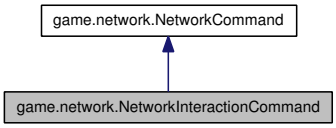
- game/network/NetworkGameCommand.java

game.network.NetworkInteractionCommand Class Reference

Inheritance diagram for game.network.NetworkInteractionCommand:



Collaboration diagram for game.network.NetworkInteractionCommand:



Detailed Description

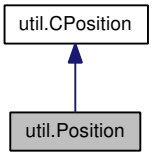
Used to transfer information regarding network activity. Such as time-out and leaving.

The documentation for this class was generated from the following file:

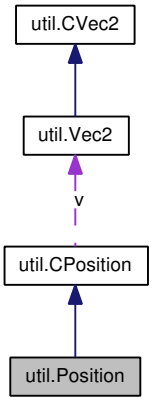
- game/network/NetworkInteractionCommand.java

util.Position Class Reference

Inheritance diagram for util.Position:



Collaboration diagram for util.Position:



Public Member Functions

- **Position** (**CVec2 v**, int track)
- **Position** (float x, float y, int track)
- final void **x** (float xx)
- final void **y** (float yy)
- final void **track** (int t)
- final void **combatPos** (**CVec2 v**)

Detailed Description

The non constant version of **`CPosition`** (p. 40).

Constructor & Destructor Documentation

util.Position.Position (CVec2 *v*, int *track*)

Parameters:

v - **Position** (p. 74) in the 2d plane.

track - The track of the game object.

util.Position.Position (float *x*, float *y*, int *track*)

Parameters:

x - x cordinate of the object.

y - y cordinate of the object.

track - track of the object.

Member Function Documentation

final void util.Position.x (float *xx*)

Returns the x cordinate.

Precondition:

None

Postcondition:

Class data is unchanged.

final void util.Position.y (float *yy*)

Returns the y cordinate.

Precondition:

None

Postcondition:

Class data is unchanged.

final void util.Position.track (int *t*)

Returns the track.

Precondition:

None

Postcondition:

Class data is unchanged.

final void util.Position.combatPos (CVec2 *v*)

Returns the combat position.

Precondition:

None

Postcondition:

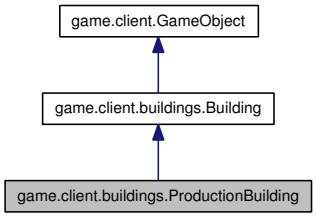
Class data is unchanged.

The documentation for this class was generated from the following file:

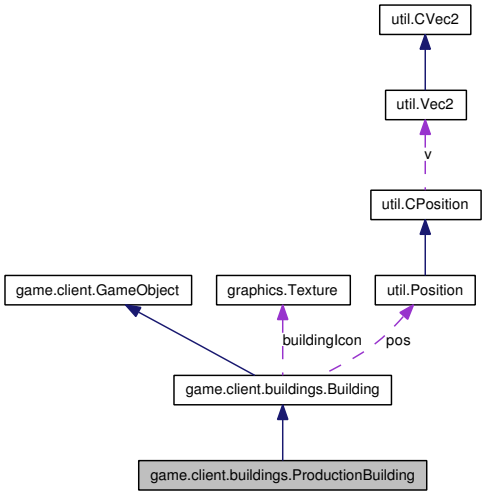
- util/Position.java

game.client.buildings.ProductionBuilding Class Reference

Inheritance diagram for game.client.buildings.ProductionBuilding:



Collaboration diagram for game.client.buildings.ProductionBuilding:



Public Member Functions

- **ProductionBuilding** (**Unit** unitPrototype)
- void **destroy** ()
- void **think** ()

Detailed Description

The class for the production buildings. A production building produces moving units. The game uses the Prototype Design Pattern to create new buildings.

Constructor & Destructor Documentation

game.client.buildings.ProductionBuilding.ProductionBuilding (Unit *unitPrototype*)

Parameters:

unitPrototype The unit prototype for this production building.

Member Function Documentation

void game.client.buildings.ProductionBuilding.destroy ()

See also:

game.client.buildings.Building.destroy() (p. 33)

Reimplemented from **game.client.buildings.Building** (p. 33).

void game.client.buildings.ProductionBuilding.think ()

Spawn a unit depending on a counter state.

See also:

game.client.buildings.Building.think() (p. 34)

Reimplemented from **game.client.buildings.Building** (p. 34).

The documentation for this class was generated from the following file:

- game/client/buildings/ProductionBuilding.java

game.server.Proxy Class Reference

Public Member Functions

- **Proxy** ()
- **Proxy** (int currentFrame)
- void **broadcast** (**NetworkGameCommand** cmd)

Detailed Description

The **Proxy** (p. 79) is responsible for tagging and broadcasting NetworkGameCommands to all connected clients.

Constructor & Destructor Documentation

game.server.Proxy.Proxy ()

Precondition:

None.

Postcondition:

The currentframe is set to 0.

game.server.Proxy.Proxy (int *currentFrame*)

Parameters:

currentFrame The frame to set as current frame.

Precondition:

None.

Postcondition:

The currentframe is set to the given value.

Member Function Documentation

void game.server.Proxy.broadcast (NetworkGameCommand *cmd*)

Tags and broadcasts the given NetworkGameCommand to all connected clients.

Parameters:

cmd The command to broadcast.

Precondition:

None.

Postcondition:

The given command has been broadcasted to all connected clients.

The documentation for this class was generated from the following file:

- game/server/Proxy.java

graphics.Renderer Class Reference

Public Member Functions

- final void **renderTrackAlignedTexture** (**Texture** t, **CAxisAlignedRect** bb, int track)

Detailed Description

Renders objects in the game world.

Member Function Documentation

final void graphics.Renderer.renderTrackAlignedTexture (Texture *t*, CAxisAlignedRect *bb*, int *track*)

The texture is streched to the whole bounding rectangle.

Parameters:

t **Texture** (p. 85) of the mesh.

bb The bounding rectangle.

track The descrete position on the Z-axis.

Precondition:

The screen is ready for rendering.

Postcondition:

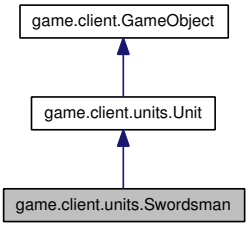
Game element is rendered on screen.

The documentation for this class was generated from the following file:

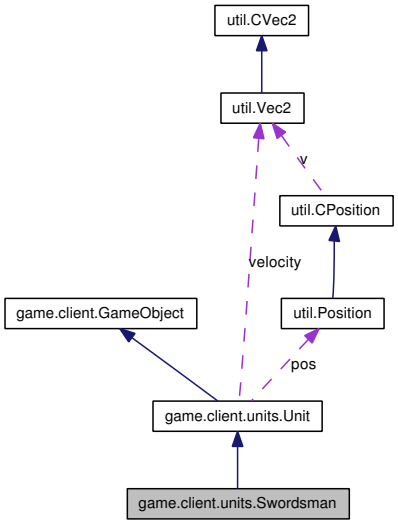
- graphics/Renderer.java

game.client.units.Swordsman Class Reference

Inheritance diagram for game.client.units.Swordsman:



Collaboration diagram for game.client.units.Swordsman:



Public Member Functions

- void **damage** (int amount)
- void **destroy** ()
- boolean **isDead** ()
- boolean **shallBeRemoved** ()
- void **think** ()

Detailed Description

Is a melee unit that attacks units within range.

Member Function Documentation

void game.client.units.Swordsman.damage (int *amount*)

Removes the given amount of hitpoints from the unit.

Parameters:

amount Amount of damage to subtract from health.

Precondition:

None

Postcondition:

The entered amount of damage has been subtracted from the objects health or it's health is zero.

Reimplemented from **game.client.units.Unit** (p. 88).

void game.client.units.Swordsman.destroy ()

Removes the object from the game world.

Precondition:

None.

Postcondition:

The object is removed from the game world.

Reimplemented from **game.client.units.Unit** (p. 88).

boolean game.client.units.Swordsman.isDead ()

Returns true if the object is considered to be dead.

Precondition:

None

Postcondition:

None

Reimplemented from **game.client.units.Unit** (p. 89).

boolean game.client.units.Swordsman.shallBeRemoved ()

Returns true if the object should be removed from the game.

Returns:

True if the object is scheduled for removal

Precondition:

None

Postcondition:

None

Reimplemented from **game.client.units.Unit** (p. 89).

void game.client.units.Swordsman.think ()

Performs the actions of a object that should be executed every frame.

Precondition:

The object is not dead.

Postcondition:

The object has performed it's actions for this frame.

Reimplemented from **game.client.units.Unit** (p. 89).

The documentation for this class was generated from the following file:

- game/client/units/Swordsman.java

graphics.Texture Class Reference

Detailed Description

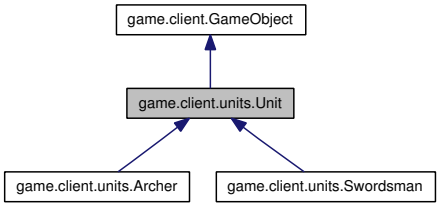
Data container for images.

The documentation for this class was generated from the following file:

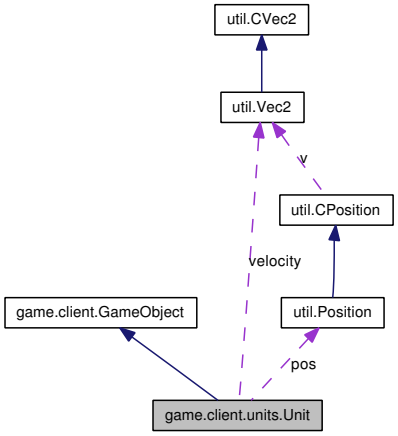
- graphics/Texture.java

game.client.units.Unit Class Reference

Inheritance diagram for game.client.units.Unit:



Collaboration diagram for game.client.units.Unit:



Public Types

- enum **Responsibility**

Public Member Functions

- final **Responsibility getResponsibility** ()
- final void **setResponsibility** (**Responsibility** responsibility)
- final **CPosition getPos** ()
- final void **setPos** (**CPosition** pos)
- **CAxisAlignedRect getBoundingBox** ()
- void **damage** (int amount)
- void **destroy** ()
- boolean **isDead** ()
- boolean **shallBeRemoved** ()
- void **think** ()

Detailed Description

The base class for units.

Member Enumeration Documentation

enum game::client::units::Unit::Responsibility

A unit can be responsible for either attacking or defending.

Author:

mandermo

Member Function Documentation

final Responsibility game.client.units.Unit.getResponsibility ()

See also:

Responsibility (p. 87)

final void game.client.units.Unit.setResponsibility (Responsibility *responsibility*)

Parameters:

responsibility The responsibility to set.

final CPosition game.client.units.Unit.getPos ()

Returns the position of the unit relative to the lower left corner.

Returns:

The position of the unit relative to the lower left corner.

Precondition:

None

Postcondition:

None

Implements **game.client.GameObject** (p. 56).

final void game.client.units.Unit.setPos (CPosition *pos*)

Precondition:

None @ Sets the position of the unit in the game world.

Parameters:

pos The position to set the object to.

Precondition:

None.

Postcondition:

The postion of the unit is now the given position.

Implements **game.client.GameObject** (p. 56).

CAxisAlignedRect game.client.units.Unit.getBoundingBox ()

Returns the bounding box of a game object inside the world.

Returns:

A box that bounds the object.

Precondition:

None.

Postcondition:

None.

Implements **game.client.GameObject** (p. 56).

void game.client.units.Unit.damage (int *amount*)

Removes the given amount of hitpoints from the unit.

Parameters:

amount Amount of damage to subtract from health.

Precondition:

None

Postcondition:

The entered amount of damage has been subtracted from the objects health or it's health is zero.

Implements **game.client.GameObject** (p. 55).

Reimplemented in **game.client.units.Archer** (p. 29), and **game.client.units.Swordsman** (p. 82).

void game.client.units.Unit.destroy ()

Removes the object from the game world.

Precondition:

None.

Postcondition:

The object is removed from the game world.

Implements **game.client.GameObject** (p. 55).

Reimplemented in **game.client.units.Archer** (p. 29), and **game.client.units.Swordsman** (p. 83).

boolean game.client.units.Unit.isDead ()

Returns true if the object is considered to be dead.

Precondition:

None

Postcondition:

None

Implements **game.client.GameObject** (p. 55).

Reimplemented in **game.client.units.Archer** (p. 29), and **game.client.units.Swordsman** (p. 83).

boolean game.client.units.Unit.shallBeRemoved ()

Returns true if the object should be removed from the game.

Returns:

True if the object is scheduled for removal

Precondition:

None

Postcondition:

None

Implements **game.client.GameObject** (p. 54).

Reimplemented in **game.client.units.Archer** (p. 30), and **game.client.units.Swordsman** (p. 83).

void game.client.units.Unit.think ()

Performs the actions of a object that should be executed every frame.

Precondition:

The object is not dead.

Postcondition:

The object has performed it's actions for this frame.

Implements **game.client.GameObject** (p. 55).

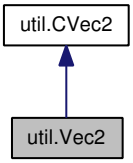
Reimplemented in **game.client.units.Archer** (p. 30), and **game.client.units.Swordsman** (p. 84).

The documentation for this class was generated from the following file:

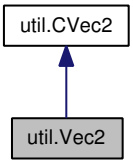
- game/client/units/Unit.java

util.Vec2 Class Reference

Inheritance diagram for util.Vec2:



Collaboration diagram for util.Vec2:



Public Member Functions

- **Vec2** (float xx, float yy)
- final void **x** (float xx)
- final void **y** (float yy)

Detailed Description

A non constant version of **CVec2** (p. 43).

Constructor & Destructor Documentation

util.Vec2.Vec2 (float *xx*, float *yy*)

Creates a **Vec2** (p. 90) instance.

Parameters:

xx - The x cordinate.

yy - The y cordinate.

Member Function Documentation

final void util.Vec2.x (float *xx*)

Sets the x cordinate.

Parameters:

xx - The x cordinate.

Precondition:

None

Postcondition:

this.x has the value of parameter xx.

final void util.Vec2.y (float *yy*)

Sets the y cordinate.

Parameters:

yy - The y cordinate.

Precondition:

None

Postcondition:

this.y has the value of parameter yy.

The documentation for this class was generated from the following file:

- util/Vec2.java

game.client.World Class Reference

Collaboration diagram for game.client.World:

Public Member Functions

- Iterator< TraceInfo > **trace** (**Vec2** combatPos, int track, TraceInfo.Dir dir, float dist, Set< TraceInfo.Attr > attr)

Static Public Member Functions

- static **World ins** ()

Detailed Description

Contains all game objects and can be used to trace units and buildings.

Member Function Documentation

static World game.client.World.ins () `[static]`

Precondition:

None.

Postcondition:

None.

Iterator<TraceInfo> game.client.World.trace (Vec2 *combatPos*, int *track*, TraceInfo.Dir *dir*, float *dist*, Set< TraceInfo.Attr > *attr*)

Returns a iterator that can be traversed to incrementaly get units or buildings infront of `combatPos`.

Parameters:

combatPos The position that the trace starts from.

track The track to trace on or all tracks if `track==-1`.

dir The direction to trace in.

dist The max distance to trace.

attr Attributes for the trace.

Returns:

Precondition:

None.

Postcondition:

None.

The documentation for this class was generated from the following file:

- game/client/World.java

Index

5.6 Package Diagram

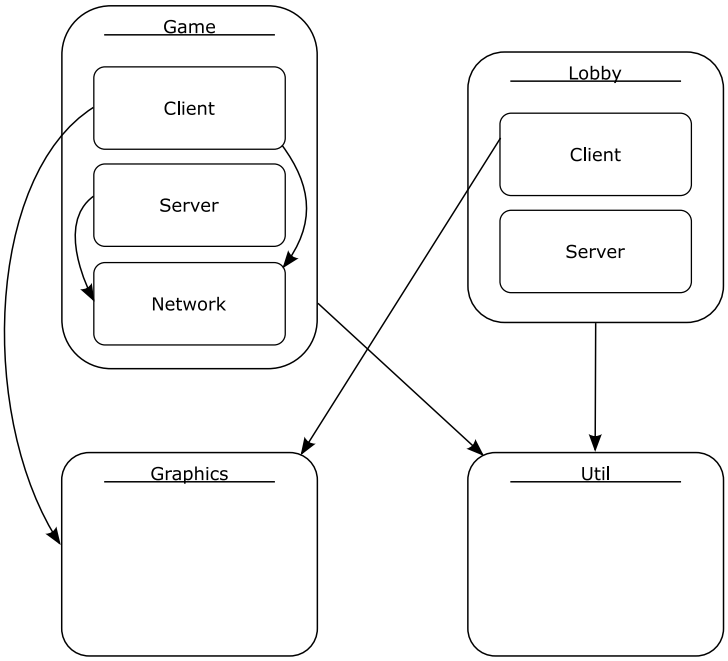


Figure 23: Describes how the different packages are connected.

6 Functional Test Cases

Failing network connections (lobby, client side)

Covering requirement 1

Input None

Observable effects The user shall be presented with a message saying that he does not have a connection with the game server.

Procedure

1. Start the game.
2. Wait for the lobby to appear.
3. Pull the network cord.

Failing network connections (ingame, client side)

Covering requirement 1

Input None

Observable effects The user shall be presented with a message saying that he does not have a connection with the game server.

Procedure

1. Start the game.
2. Wait for the lobby to appear.
3. Join a game.
4. Wait for a game session to start.
5. Pull the network cord.

Failing network connections (ingame, server side)

Covering requirement 1

Input A started game session.

Observable effects The user with a failing network connection shall be removed from the game server.

Procedure

1. Pull the network cord on one of the clients.
2. Look in the network log for taken actions.

Screen resolution

Covering requirement 2

Input The game application is started.

Observable effects The user shall see a change of resolution (if there are more than one to choose).

Procedure

1. Press the options button in the menu.
2. Press the resolution button in the menu.
3. Click on the drop down list in the dialog.
4. Choose a resolution.

Registration

Covering requirement 3

Input The game application is started.

Observable effects There shall be a field in the database with the user's information.

Procedure

1. Press the register button.
2. Fill in a valid username and password (see Table 4 in the requirements document).
3. Press ok.
4. Open the database utility.
5. Search for the user with `"SELECT * FROM users WHERE username='myuser'"`.

User statistics

Covering requirement 4

Input A game result

Observable effects The database row of the account shall be changed according to the result of the game.

Procedure

1. Check the database row representing your account and take note of the columns representing wins and losses.
2. Play a game and win.
3. Check the database row again. The value representing wins shall be the old value incremented by one and the value representing the losses shall be unchanged.

Procedure

1. Check the database row representing your account and take note of the columns representing wins and losses.
2. Play a game and lose.
3. Check the database row again. The value representing losses shall be the old value incremented by one and the value representing the wins shall be unchanged.

Game sessions

Covering requirement 5

Input Data about existing games on the server

Observable effects Each existing game session on the server shall conform to the limits that Requirement 6 implies.

Procedure

1. Join a game with one player missing.
2. Then the game shall start automatically.

Procedure

1. Join a game with more than one player missing
2. Then the game shall start when the other spots are filled with players.

Create game

Covering requirement 6

Input Data to create a game.

Observable effects A new game shall be created.

Procedure

1. Click on create new game.
2. Check that its not possible to create a new game without a Name (3-30 alphanumeric characters or spaces) or password (3-30 alphanumeric characters) or player limit (integer greater or equal to 2).
3. When the game is created you shall automatically be a participant of the new game.

Join game session

Covering requirement 7

Input None.

Observable effects The lobby GUI goes into the GUI state showed in Figure 9 in section 4.

Procedure

1. Go to lobby.
2. Choose a server join.
3. Click the join button in the game GUI, type in the password in the Password field in the dialog.
4. The join should normally be successful but in the rare occurrences when an other client joined the game session and we tried to join it but did so after the other player, but before we got a update about the join event from the server.

Leave game session

Covering requirement 8

Input None.

Observable effects None.

Procedure

1. Start a game session with two clients.
2. The user clicks the leave button.
3. Continues to LOOSING requirement.

Search game session

Covering requirement 9

Input None.

Observable effects The servers in the game list is sorted according to the filter settings.

Procedure

1. Type in filter settings like that he wants the game servers sorted according to.
2. Click on the filter button.

Quick game

Covering requirement 10

Input None.

Observable effects The player that choose quick game joins the server with the player with similar user statistics.

Procedure

1. Start a two game servers and ensure that they are the only game servers in the lobby.
2. Join a player with significantly different statistics on one of the servers and a player with similar statistics on the other server.
3. Start a quick game.

Play order

Covering requirement 11

Input None.

Observable effects Depends on user input.

Procedure

1. Start at least three clients and join a game session.
2. At the beginning of the game, each player should not be able to send units all the way to the players castle which is located clock-wise.
3. All units sent counter-clockwise should stop half way and not approach the castle.
4. When a player is ousted, the units sent by the player ousting the player shall now continue attacking the next player which is located clock-wise if there are still more than one player with a castle.

Distributing attacking and defending units

Covering requirement 12

Input None.

Observable effects Depends on user input.

Procedure

1. Start at least two clients and join a game session.
2. Change the distribution of attacking and defending units.
3. Observe that the amount of attacking and defending units corresponds to the ratio given at the time when a new wave of units spawns.

Spawning units

Covering requirement 13

Input None.

Observable effects Depends on user input.

Procedure

1. Start at least two clients and join a game session.
2. Observe that on given intervals waves of units shall spawn from the players castles.

Hero

Covering requirement 14

Input None.

Observable effects All players has a own hero just as it should be.

Procedure

1. Start a game with any number of players you want.

Hero resurrection

Covering requirement 15

Input None.

Observable effects All players has a own hero just as it should be.

Procedure

1. Start a game.
2. Go with your hero to the enemy base, wait till you gets killed.
3. Wait till your appears again in your base.
4. Immediately go with your hero to the enemy base.
5. Wait till you gets killed, observe that you gets killed more easily this time (see req. 14. Hero).

Units and buildings attacking units, building or heroes

Covering requirement 16

Input None.

Observable effects The hero dies.

Procedure

1. Start a game session with two players, that you control.
2. Build a defense for the first player.
3. Select the hero for the second player, go near the first players defense tower.
4. Verify that the defense tower of the first player starts to shoot at the hero.
5. Stay till your hero dies.

Deal damage

Covering requirement 17

Input None.

Observable effects A unit dies.

Procedure

1. See *Units and buildings attacking units, building or heroes.*

Hostile unit, building and hero

Covering requirement 18

Input None.

Observable effects The enemy two units loses health.

Procedure

1. Start a game with to players.

2. Move the hero of the first player to two enemy units.
3. Utilize a stock wave.

Losing

Covering requirement 19

Input None.

Observable effects Depends on user input.

Procedure

1. Start two clients and join a game session.
2. Let one of the client's castle be destroyed by the other client.
3. The client who got its castle destroyed shall lose the game.
4. Then the statistics shall be updated, according to test case 4.
5. The territory the client had which got its castle destroyed shall be added to the other client that destroyed the castle.
6. Then the client who lost shall have the option to return to the lobby or be able to observe the rest of the game and not be able to interact with the game.

Ingame chat

Covering requirement 20

Input Text message to send.

Observable effects The message shall appear on all players involved in the game session.

Procedure

1. Send a text message using the ingame chat.
2. The message shall appear on all players involved in the game session.

Move camera

Covering requirement 21

Input Move mouse towards the left or right edge of the screen.

Observable effects The camera shall move in that direction

Procedure

1. Move the mouse towards the left or right edge of the screen.
2. Then the camera shall move in that direction.

Minimap

Covering requirement 22

Input None.

Observable effects The minimap shows a miniature of the game world and indicates which area the main view is currently viewing.

Dying and destruction

Covering requirement 23

Input A started game session with two players (both controlled by the tester).

Observable effects The attacked game element shall be inoperable.

Procedure

1. Attack the game element with an attacking unit or hero.
2. Wait for the hit points to reach zero.
3. When the hit points reaches zero the game element shall be removed/inoperable.

Dying and destruction

Covering requirement 23

Input None.

Observable effects A unit is dead and that unit can interact with the world.

Procedure

1. In any game you can wait until an archer dies.
2. That unit will not interact with the world and will not shoot.

Castle

Covering requirement 24

Input None.

Observable effects None.

Procedure

1. Wait in any game till a player loses.
2. See req. 19.

Castle

Covering requirement 24

Input A started game session with two players (both controlled by the tester).

Observable effects One of the players loses.

Procedure

1. Don't build anything with player one.
2. Create attacking units with player two.
3. Attack player two's castle and wait for the hit points to reach zero.

Move hero

Covering requirement 25

Input None.

Observable effects None.

Procedure

1. Build a defensive tower.
2. Move the hero to the same track as the tower.
3. If the hero is at the left side of the tower, move the hero to right, if the hero is at the right side of the tower, move the hero to the left.
4. Continue at 3.

Move hero

Covering requirement 25

Input A started game session with a hero alive.

Observable effects The hero shall be moved in the direction initiated by the player.

Procedure

1. Press one of the keys bound to movement, the hero should move.
2. Try to move to a spot occupied by an other game element, by moving in the respective direction. The hero shall not move in this direction.

Constructing a building

Covering requirement 26

Input A started game session and enough resources to build one building.

Observable effects The building shall be built.

Procedure

1. Select an affordable building by pressing its build button.
2. Build it on a buildable location (see requirement 26).

Observable effects The building shall not be built.

Procedure

1. Select an affordable building by pressing its build button.
2. Try to build it on a location not buildable (see requirement 26).
3. The system shall present information about why it could not be built.

Observable effects The building shall not be built.

Procedure

1. Select a building not affordable.
2. The system shall present information about why it could not be built.

Constructing a building

Covering requirement 26

Input None.

Observable effects None.

Procedure

1. Build buildings a free spots until you get the message not enough resources.

Buildable locations

Covering requirement 27

Input None.

Observable effects A defensive tower should be built.

Procedure

1. Start a game with two players.
2. Try to build a defensive tower on enemy grounds, that attempt will fail.
3. Try to build a defensive tower on a non occupied spot near your own castle.

Attributes

Covering requirement 28

Input None.

Observable effects None.

Procedure

1. See req. 16 for an example on health.

Upgradable attributes

Covering requirement 29

Input None.

Observable effects New archers produced is stronger than they were before.

Procedure

1. Build the production building that enables you to produce archers.
2. Observe how powerful they are by watching how the perform in fight.
3. Upgrade archer.
4. Now all the old archers has the same powerfulness.
5. Observe how powerful the new archers are in battle verify that they are more powerful.

List of Tables

List of Figures