D.U.N.E.

Group 11

Klas Flodin
Kaj Sandberg
Anders Ljungqvist
Erik Nikkola
Mikael Nilsson

Table of contents

1. Introduction

1.1. The purpose and scope of this document

The purpose of this document is to clearly define the integral parts of the game D.U.N.E. and to be used in its development both as a reference for project programmers as well as time planning reference for project coordinators.
The document will provide description of Classes and methods needed to produce a playable alpha release.

1.2. Intended audience

- Project team members
- Anyone who wishes to further develop this game after its alpha release.

1.3. Version History

Version 1.0 – this version

1.4. Related Documents

The reader of this document is assumed to be familiar with the "Requirements document".

1.5. Glossary

AI	Artificial intelligence. A computer opponent.
CamelCase	The practice of writing compound words or phrases in which the words are joined without spaces and are capitalized within the compound. CamelCase is a standard identifier naming convention for several programming languages.
CPU	Central Processing Unit
Disconnection	A user can get disconnected from a network. Disconnection from a network means that communication over the network is no longer possible.
GUI	Graphical User Interface – the main interface through which the user handles the game through various inputs on it.
Host system, multiplayer	The computer which acts as the host in a client-server architecture during a multiplayer game.
J2SE	Java Platform, Standard Edition
LAN	Local Area Connection
Multiplayer	A game mode where several players compete with or against each other in the same game.
OpenAL	**Open A**udio **L**ibrary is a free software cross-platform audio API.
OpenGL	**Open G**raphics **L**ibrary is a standard specification defining a

	cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics.
RTS	Real Time Strategy
RAM	Random Access Memory.
SP2	Service Pack 2. A service pack is a collection of updates, fixes and/or enhancements to a software program.
TCP/IP protocol	Transmission Control Protocol/Internet Protocol is an easy to use protocol for transmitting data over a network
UML	Unified Modeling Language, an object modeling and specification language used in software engineerin.
XML	**Extensible Markup Language**, a general-purpose specification for creating custom markup languages.

1.6. Naming and coding conventions

This project will use CamelCase naming convention for all variables, methods and classes. Hungarian notation is strictly forbidden.

1.7. Abstract

D.U.N.E is a game inspired by Dune II with elements also taken from more modern games such as Space Rangers 2: Rise of the Dominators. It will be a freeware multiplayer game. This document describes, in detail as well as in general, how D.U.N.E. will be implemented.
Section 2 of this document describes the game both in general and an in depth architectural description.
Section 3 describes the design considerations we needed to make.
Section 4 provides a description of the user interface.
Section 5 describes the classes and methods in detail.
Section 6 provides a way of testing the functionality of the game.

2. System Overview

2.1. General Description

D.U.N.E. is a game meant to tickle the nostalgia center of aging gamers. The idea is not to make a graphically stunning game but one that is familiar to the millions of players that have already played Dune II and still present these players with a new experience. The game will primarily focus on multiplayer mode where up to eight players connect to a game host and battle each other.

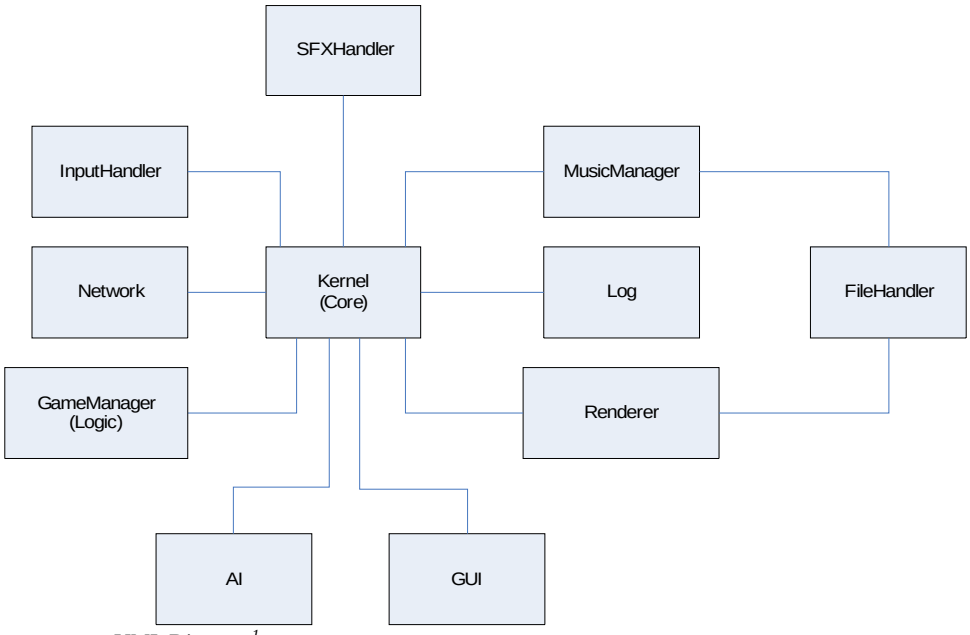
2.1.1. Singleplayer mode

In single player mode one player will play against a rudimentary AI. The option to have multiple AI opponents will not be available. This mode will present the player with an opportunity to familiarize himself with the controls and strategies of the game as well as utilize the custom unit design function which will only be available during Singleplayer mode. The designs saved during Singleplayer will later be available during multiplayer mode.

2.1.2. Multiplayer mode

In multiplayer mode the player will play against up to 7 other human players connected through a local area network. In this mode the player will exercise his strategic expertise again his fellow players and be able to draw upon his unit designs created in single player mode.

2.2. Overall Architecture Description



UML Diagram[1]

The game functions around a single central core called Kernel. The Kernel handles instructions to and between the different functional modules and is designed to be as general as possible so that different parts of the game can be as separate and transparent as possible. The GameManager will manage the maps, players and the different game objects such as buildings and units as well as perform most of the game logic excepts for the pure AI functions of the computer controlled player which are handled in the AI module. All of the in-game data will be read in by a Game Manager support class from XML files to keep this separate from binary files handling in File Handler.

MusicManager (handles the in-game soundtrack) and Renderer (handles the main game rendering) will both be connected to FileHandler in order to fetch data, such as mp3 files, from the secondary memory through filestreams. SFXhandler will not be connected to the FileHandler because OpenAL can handle the sound files for us, however not the compressed music files. The other module with some file I/O, Log, will not use File Handler either since it uses a single file to output the core's status and error logs and will only write to this file, something that can be handled more effectively on its own.

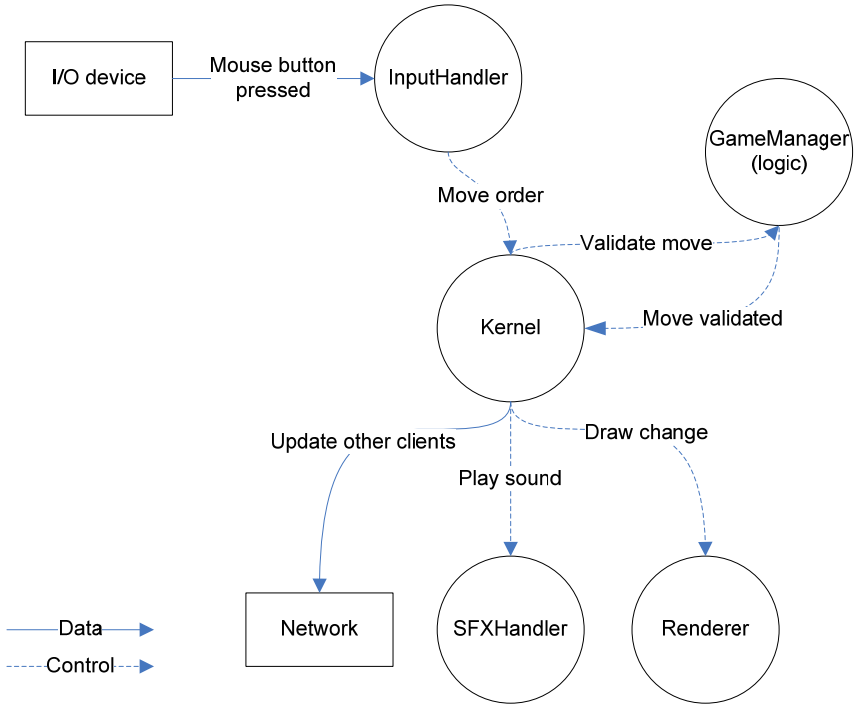
The GUI is separated from the game renderer and will only be used to present the graphical user interface above the actual main game rendering.
All direct user input on the GUI will be handled in the InputHandler and translated to a proper action.

For the network multiplayer games all of the communication will be handle through the Network module, this module will handle both the host and client side of the network game in both the initial set-up face and during the actual game play.

[1] *http://www.uml.org*

2.3. Detailed Architecture

2.3.1.Data flow and control flow diagram



This is a data flow and control flow diagram showing the command to move a unit on the game map. This command employs a very large part the functional modules in the game. The InputHandler listens for input from the supported I/O devices. In this case the right mouse button is clicked somewhere on an empty tile on the map. The input handler receives the event and translates it and sends a move command to the kernel. The kernel in turn confers with the GameManager to validate the move. When the move is validated the Kernel engages the Renderer module to draw the move and SFXHandler to play a relevant sound. When the game is played over a local area network the other players will also be updated with what has moved and where.

3. Design Considerations

3.1. Assumptions and Dependencies

Hardware:
100MB hard disk space
600MHz CPU or better
392MB RAM or more under Windows XP
OpenGL 2.0-compatible graphics card
OpenAL compatible sound card
Network Interface Card with TCP/IP-support
LAN connection for multiplayer
Two button mouse or better
Keyboard

Software:
Windows XP (SP2 or better)
J2SE 5.0 or later
OpenGL 2.0 or later
TCP/IP protocol installed

End user characteristics:
The end user should be an individual with previous experience in the strategic game genre. The experience should be sufficient that the common control subset of the genre is familiar to the user such that selecting units and moving them with the mouse alone is intuitive. Previous game experience should ideally be games such as Command & Conquer series or Starcraft.

Probable changes in functionality:

- The overall user interface is expected to change drastically in design
- Unit and faction design decisions will likely change dependencies between units
- Resource management decisions will likely change unit and structure costs
- Research decisions will affect dependencies between technological research.
- The unit design page is likely to be changed in appearance and options.

3.2. General Constraints

Hardware and driver support for OpenGL 2.0 is required. This will exclude certain computers running with integrated graphics from Intel, to help keep the code clearer no fallback methods will be implemented to support earlier versions. Due to the API chosen for sound OpenAL 1.1 support is also required. Java requires the end user to install Java Runtime Environment 5.
The maximum storage specified keeps a restraint on audio and textures used for the software where higher compression may have to be used.

4. Graphical User Interface

Pre-game menus

When starting the system, the user is presented with the game's main menu. This menu holds the key controls for starting a single player or multi player game, as well as accessing game options and exiting the game. These buttons are always accessible from the sub-menus as well.

In the singleplayer sub-menu, apart from the Main Menu functions, the user is presented with selections for starting a singleplayer game. These parameters include changing map, selecting faction to play as, and changing screen name.

When selecting the Multiplayer button, the user is taken to the Multiplayer sub-menu. This presents the user with an overview of the current available multiplayer games from a list, as well as the option to host a new multiplayer game.
When the user opts to proceed by either hosting or joining a multiplayer game, the user is presented with an overview of the LAN Game. The major difference between Host and Client is that more fields are editable for the Host, such as whether a player slot is open for others to join or not and what map to play on. Both the host and the client may chose what faction they want to play as, as well as their own displayed screen name, but neither may edit another players chosen faction or name.

The Options sub-menu presents the user with relevant system options to customize the user's experience of the game. These options include basic graphical and audio options.

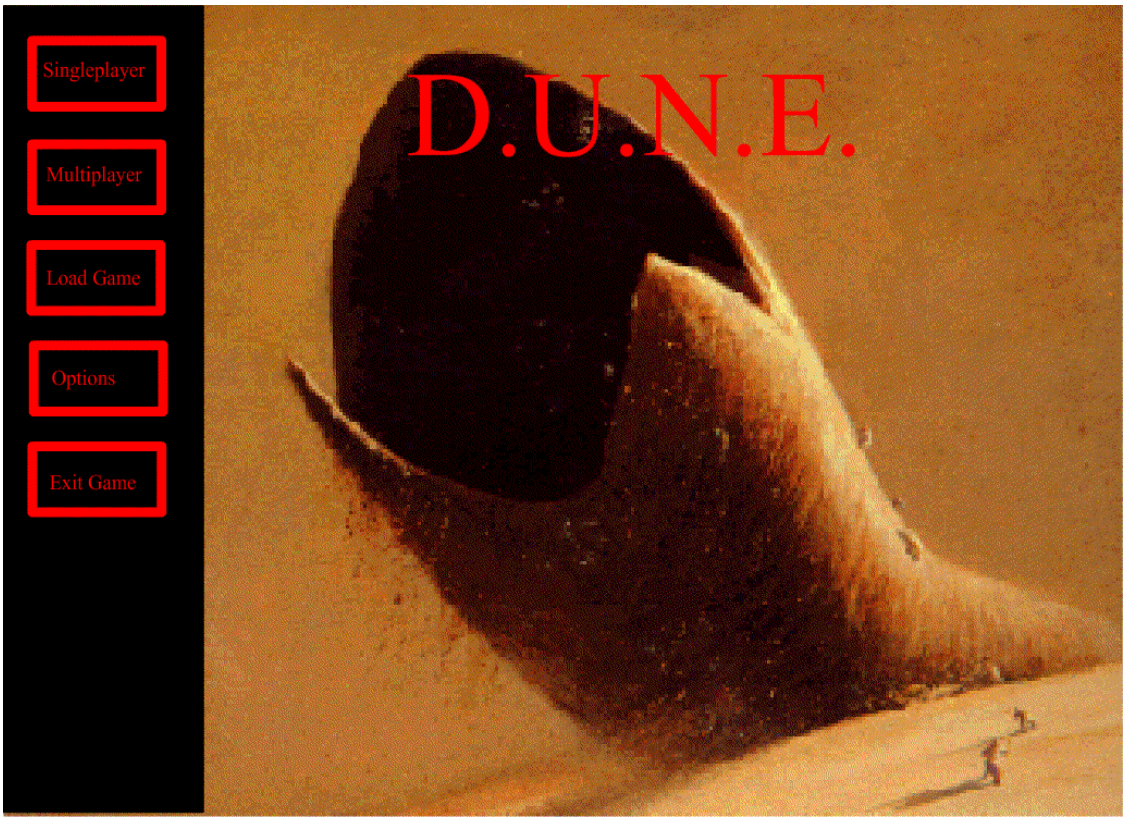
In-game menus

The main in-game heads up display provides the user with all necessary information to command a game session. This information includes a message area displaying a brief message history to the user, a unit information area displaying statistics for the unit(s) or building currently selected and a construction overview area allowing the user access to unit and building construction.
When paused, the heads up display changes to a pause overlay where all heads up display controls are frozen. A pause-menu will display options for resuming the game again or accessing the game main menu.

4.1. Form 1 – Main menu

Functional requirements:

- Start game
- Ending game



The names of the controls and fields:

Singleplayer	Access the singleplayer sub-menu
Multiplayer	Access the multiplayer sub-menu
Load game	Access the load game sub-menu
Options	Access the options sub-menu
Exit Game	Exit the game system to desktop

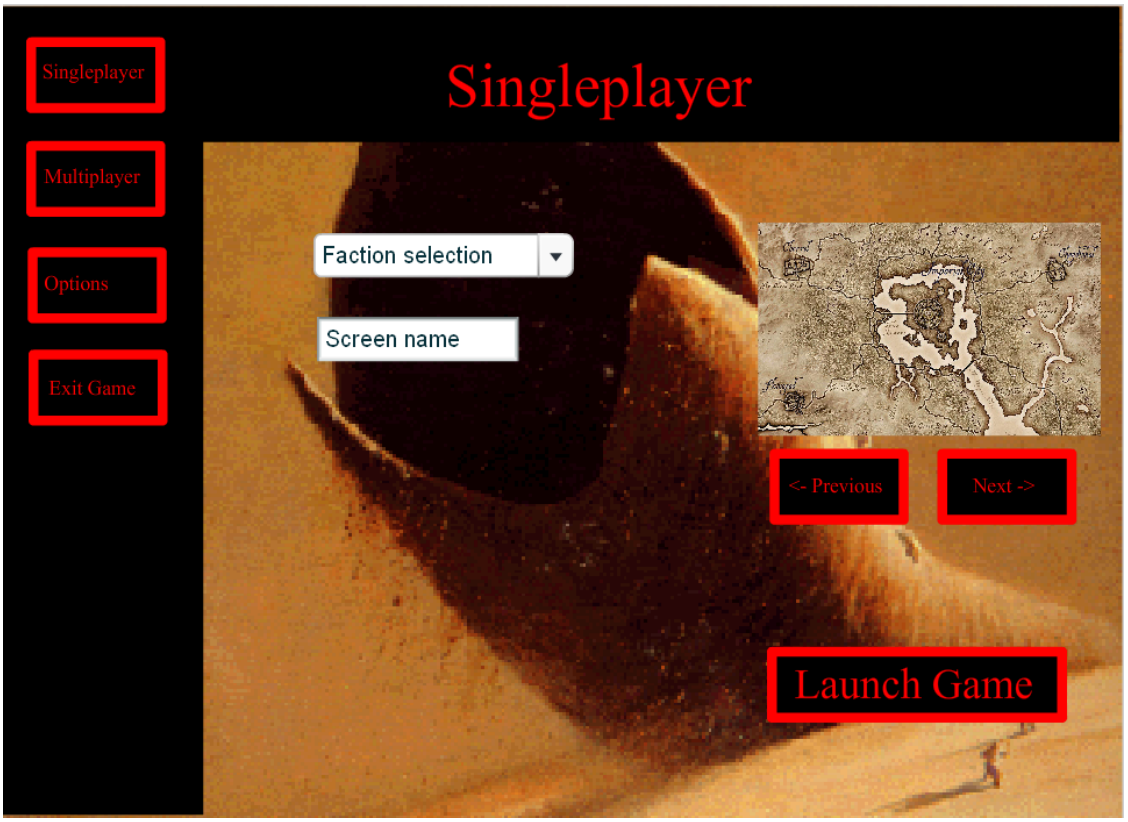
The names of the events, methods, or procedures that cause this form to be displayed:
The leftmost controls are always displayed in the pre-game menus

The names of the events, methods, or procedures triggered by each control:
Singleplayer calls displaySPmenu() to display Form 2
Multiplayer calls displayMPmenu() to display Form 3
Load Game calls calls displayLoadGame() to display Form 11
Options calls displayOptions() to display Form 5
Exit game calls display system.exit()

4.2. Form 2 – Singleplayer menu

Functional requirements:

- Starting a new game
- Factions



The names of the controls and fields:

Map window	Display window for the currently selected map
Next	Select and display the next map
Previous	Select and display the previous map
Faction drop-down	Present a selection of the available factions
Screen Name input	An input field for specifying the name associated with the player
Launch Game	Starts the game with the specified parameters

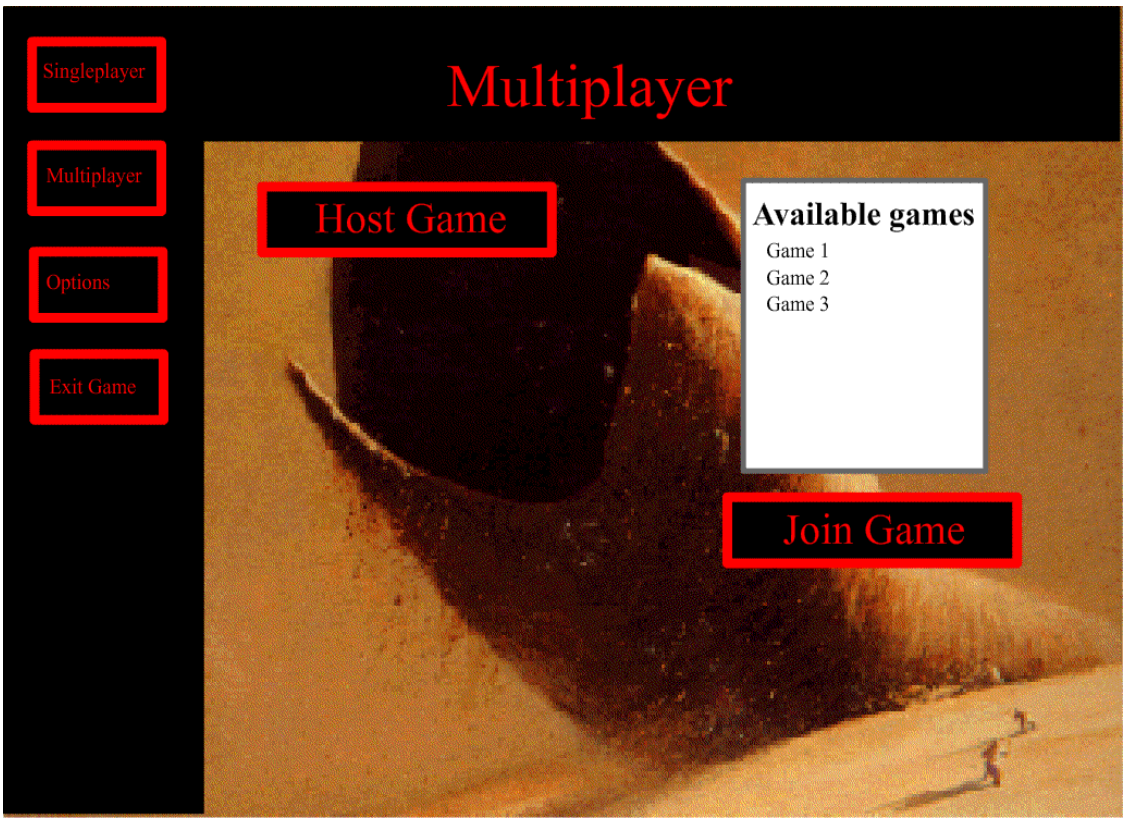
The names of the events, methods, or procedures that cause this form to be displayed:
displaySPmenu ()

The names of the events, methods, or procedures triggered by each control:
Left panel as explained in Form 1
Next calls displayNextMap()
Previous calls displayPreviousMap()
Launch Game calls initiateSpGame()

4.3. Form 3 – Multiplayer Host or Join

Functional requirements

- Network



The names of the controls and fields:

LAN Game window	Provides a selectable overview of currently available LAN games
Host Game	Changes the sub-menu to the Multiplayer (Host) sub-menu
Join Game	Changes the sub-menu to the Multiplayer (Client) sub-menu with input from the LAN Game that is marked in the LAN Game window

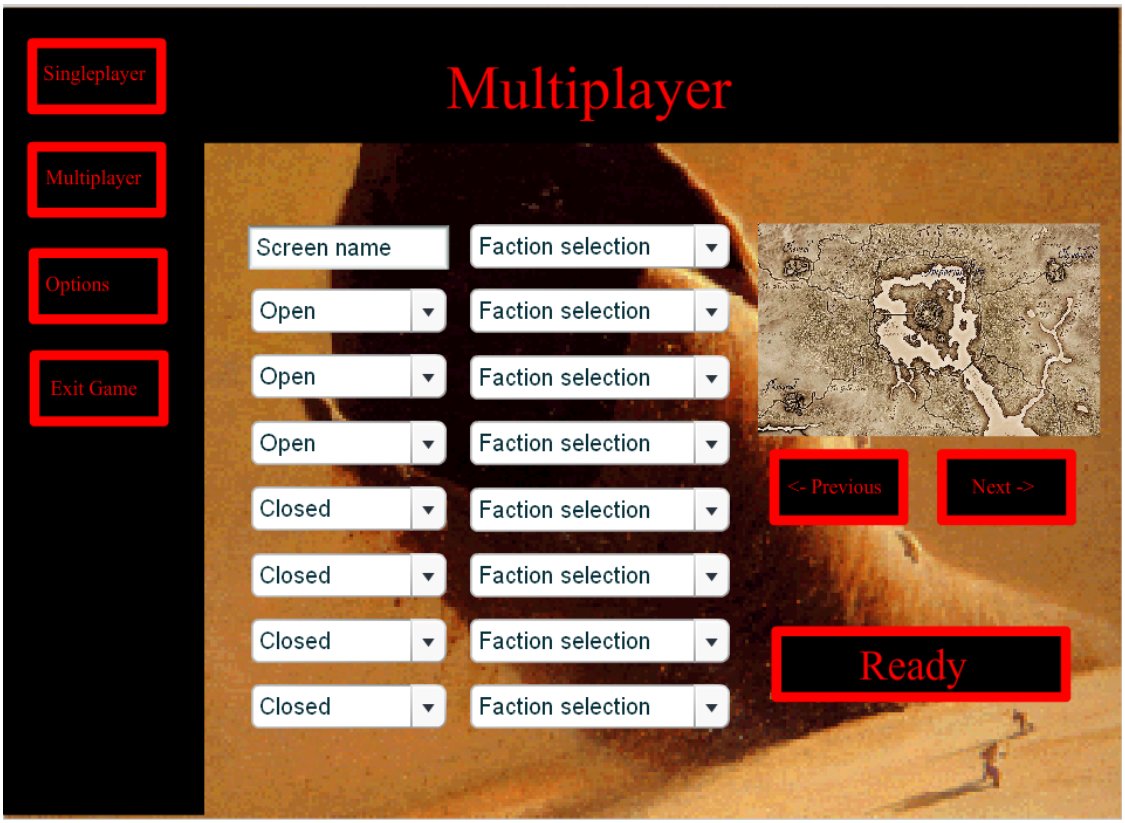
The names of the events, methods, or procedures that cause this form to be displayed:
displayMPmenu ()

The names of the events, methods, or procedures triggered by each control:
Host Game calls initiateHost()
Join Game calls joinMpGame()

4.4. Form 4 – Multiplayer game menu

Functional requirements

- Starting a new game
- Factions
- Network



The names of the controls and fields:

Player Name input	An input field which the current user may input desired screen name
Player X drop-down	Provides a view of Player X´s chosen screen name, or whether the slot is open for new players or closed. The Host may change a slot to be open or closed.
Faction X drop-down	Provides a view of the relevant player's chosen faction. This is displayed as "None" for open or closed slots and is editable only for the user's own faction, which is displayed next to the Player Name input field. Host may affect all players' factions.
Next	Select and display the next map
Previous	Select and display the previous map
Ready	States that the user is ready to start the game
Launch Game	Launches the game provided all players have specified they are ready. This is only accessible to the Host.

The names of the events, methods, or procedures that cause this form to be displayed:
initiateHost()

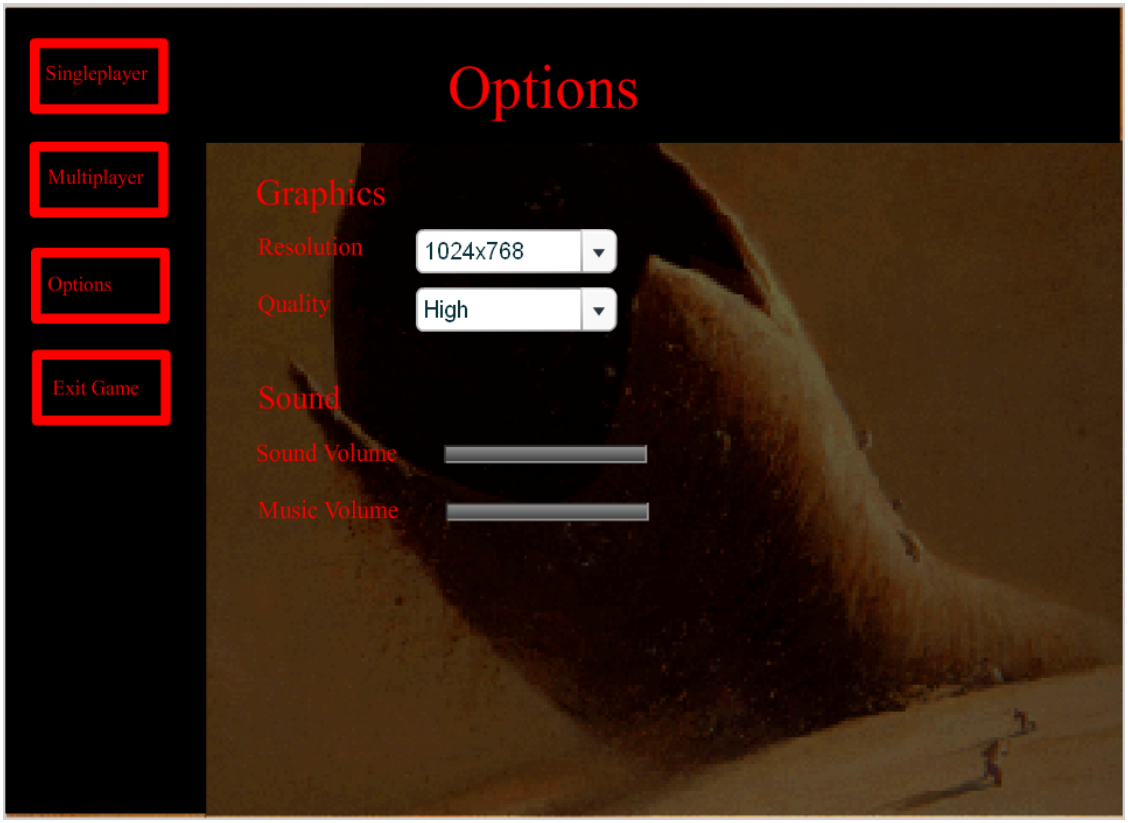
The names of the events, methods, or procedures triggered by each control:
Next calls displayNextMap()
Previous calls displayPreviousMap()
Ready calls setMpReadyState()

For host the Ready-button is instead called Start Game
Start Game calls initiateMpGame()

4.5. Form 5 – Options menu

Functional requirements

- Configuration



The names of the controls and fields:

Resolution	Specifies the desired resolution
Quality	Specifies the desired graphical quality
Sound Volume	Specifies the desired effects volume
Music Volume	Specifies the desired music volume

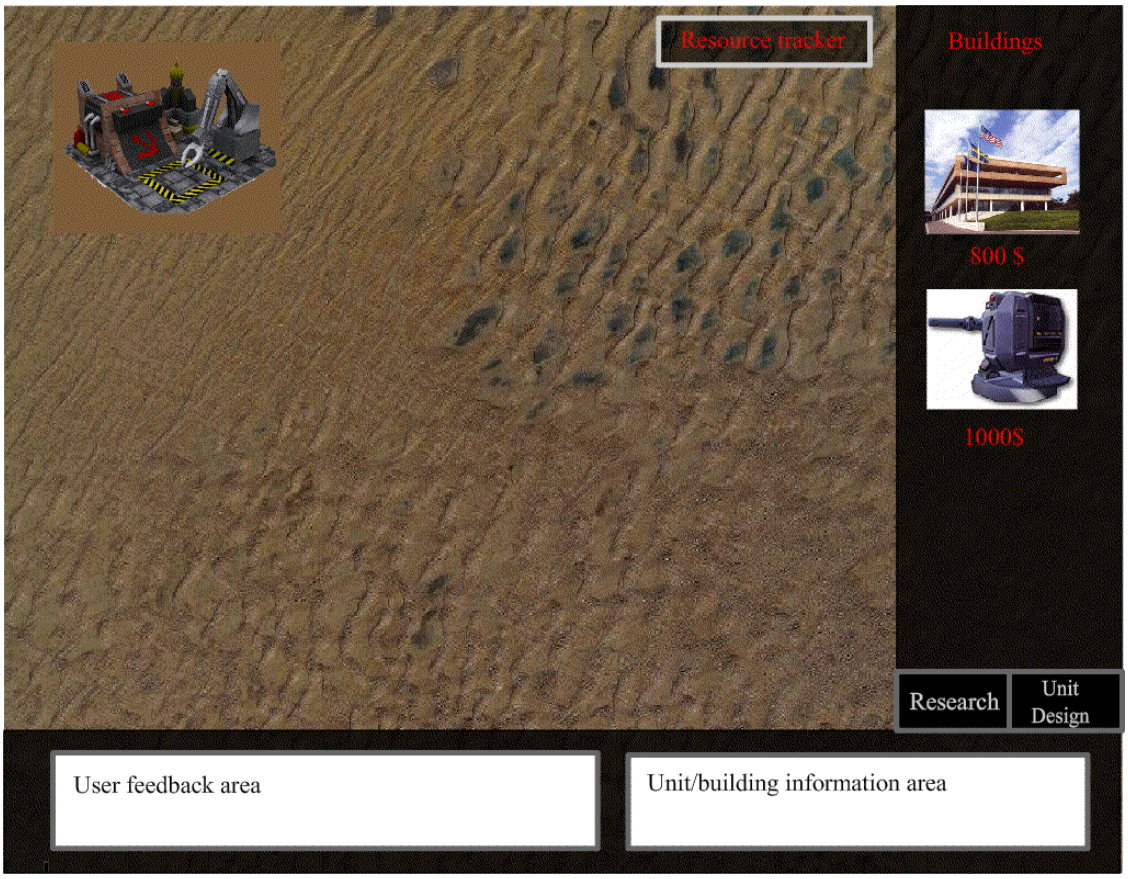
The names of the events, methods, or procedures that cause this form to be displayed:
displayOptions ()

The names of the events, methods, or procedures triggered by each control:
setResolution()
setTextureQuality()
setSoundVolume()
setMusicVolume()

4.6. Form 6 – In-game view

Functional requirements:

- Production
- Economy
- Improvements
- Factions
- Combat
- Unit/building handling



The names of the controls and fields:

User feedback area	Shows a brief chat history of messages to the player
Unit Information	Display information of the current unit(s) or build selected
Constructions	Displays a construction overview of all available constructions
Research	Displays the research menu
Custom Designs	Displays the custom unit design menu
Mouse Right click	Moves units or initiates attack

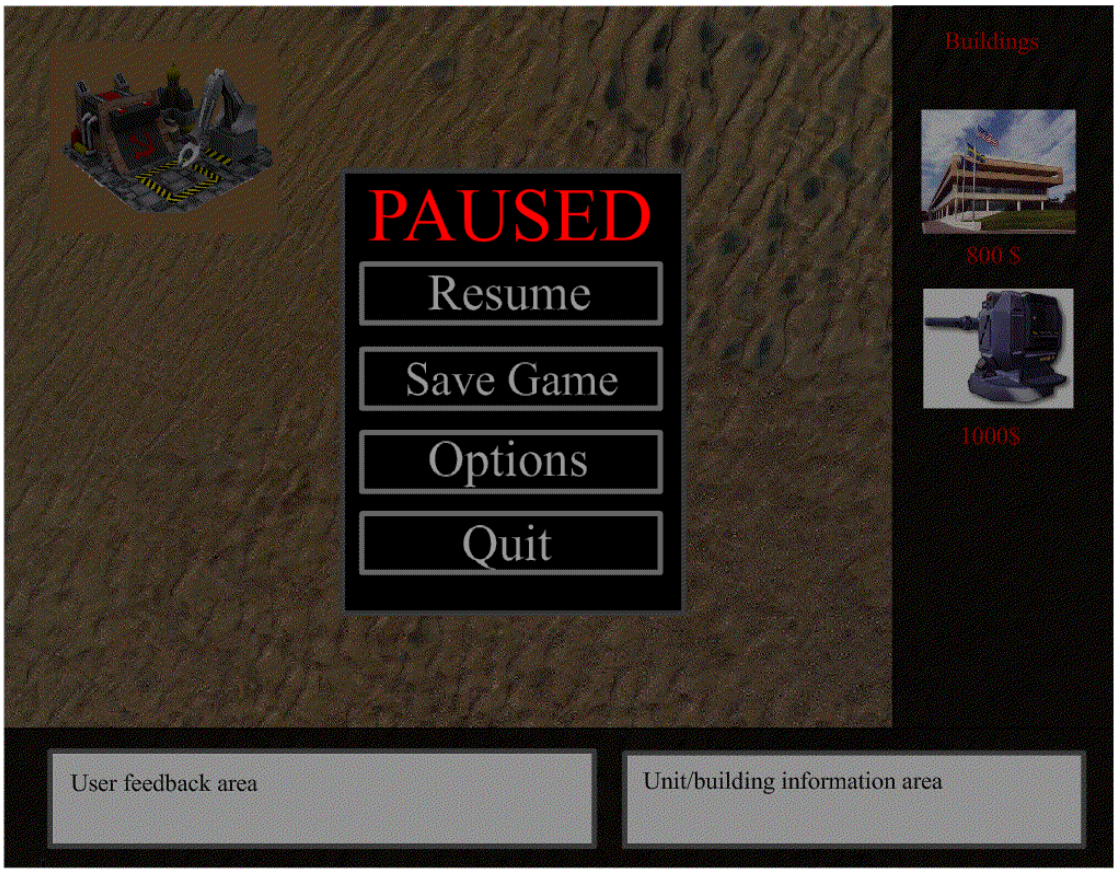
The names of the events, methods, or procedures that cause this form to be displayed:
initiateMpGame()
initiateSpGame()

The names of the events, methods, or procedures triggered by each control:
researchbutton calls displayResearch()
unitDesignbutton calls displayDesign()
buildButton calls addToBuildqueue()
mouse Right Click calls clickInterpret()

4.7. Form 7 – Paused menu

Functional requirements:

- Pausing game
- Resuming an old game
- Ending game



The names of the controls and fields:

Resume	Un-pauses the game
Save Game	Displays the save game menu
Load game	Displays the load game menu
Main menu	Displays the game's main menu

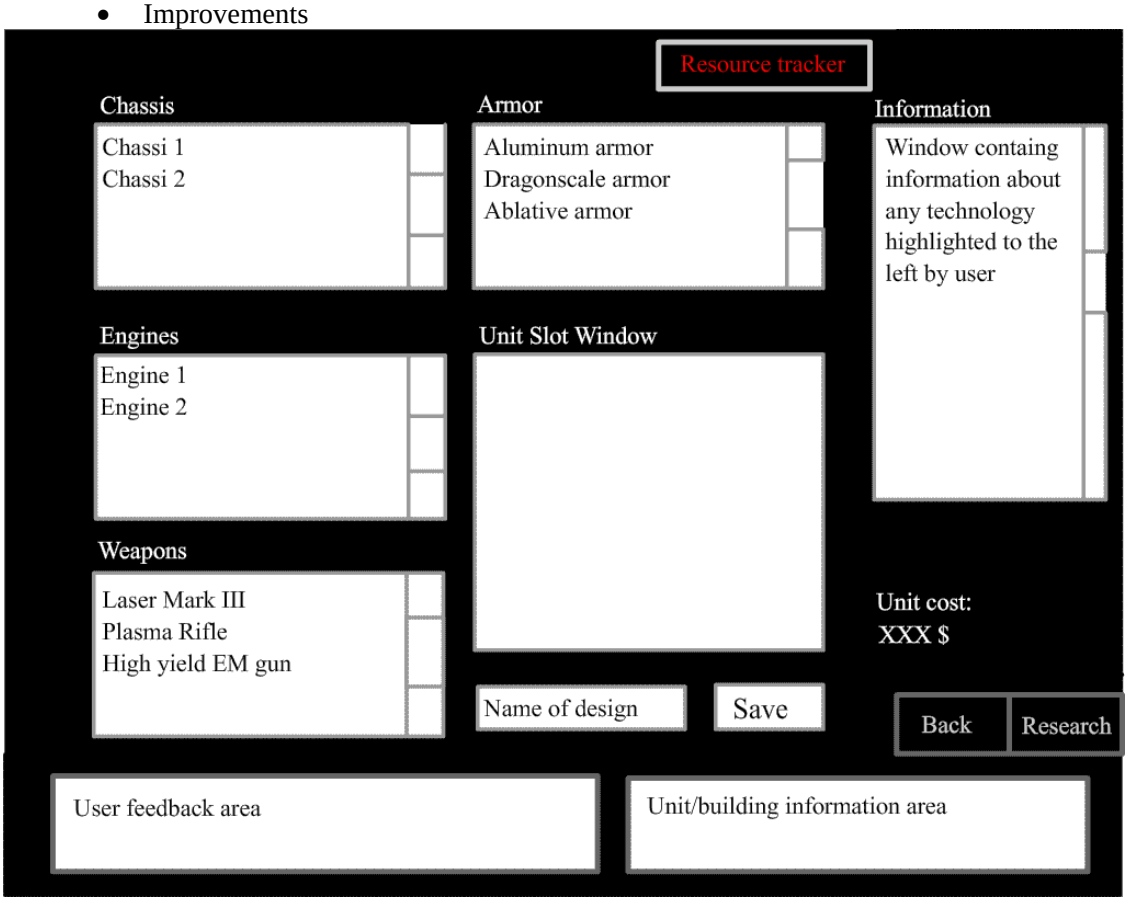
The names of the events, methods, or procedures that cause this form to be displayed:
pauseGame()

The names of the events, methods, or procedures triggered by each control:
Resume calls resumeGame()
Save Game calls displaySaveMenu()
Options calls GUI to display Form 5
Quit calls system.exit()

4.8. Form 8 – Unit design menu

Functional requirements

- Unit design
- Factions
- Improvements



The names of the controls and fields:

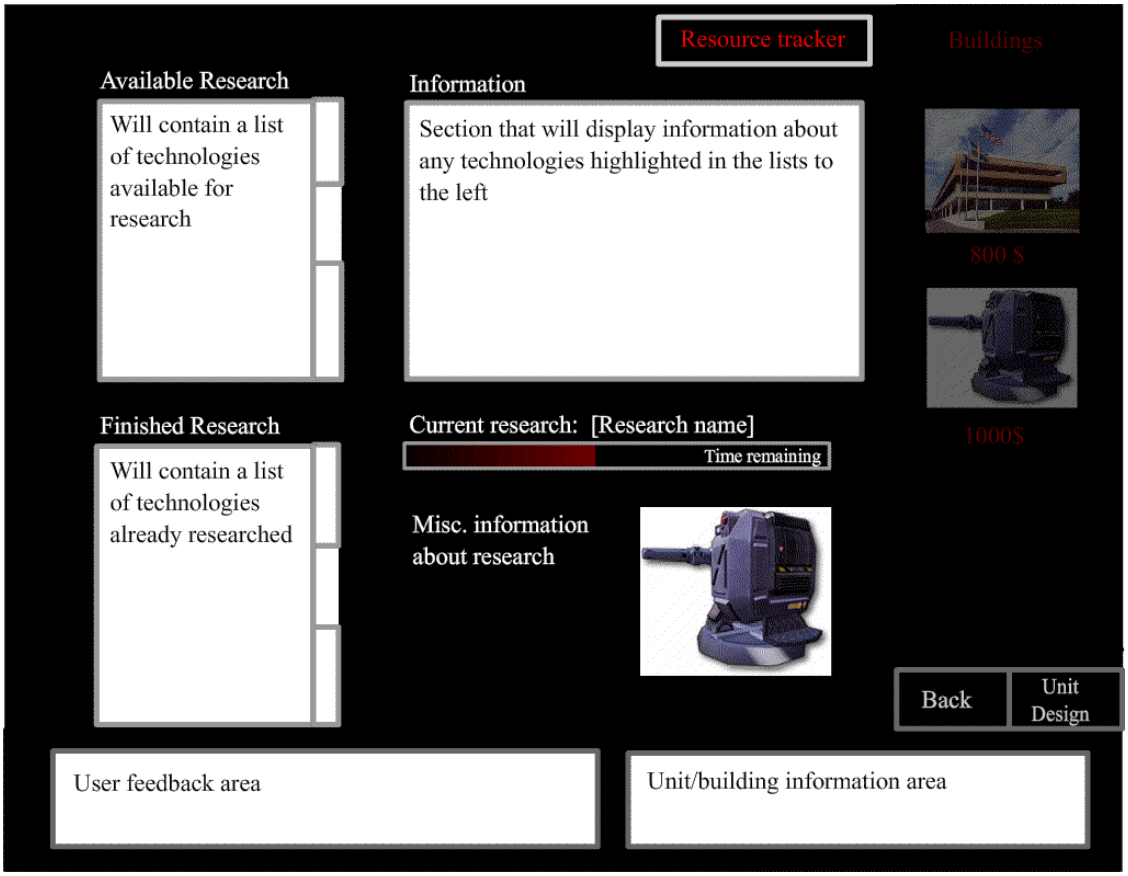
Chassis	Displays list containing available chassis
Engines	Displays list containing available engines
Weapons	Displays list containing available weapons
Armor	Displays list containing available armor
Unit slot window	Displays a summation of added components
Information window	Displays information about the currently selected component
Name of design	Field with name of design
Unit Cost	Displays current cost of design
Save	Button that calls save function to save design
Back	Returns to game
Research	Opens form 9

The names of the events, methods, or procedures that cause this form to be displayed:
displayDesign()
The names of the events, methods, or procedures triggered by each control:
Save calls saveUnitDesign()
Back calls displayGameField()
Research calls displayResearch()

4.9. Form 9 – Research menu

Functional requirements

- Improvements
- Factions



The names of the controls and fields:

Current Research	Displays the current research project.
Available Research	Displays a window with the currently available research projects.
Finished Research	Displays completed research projects.
Information window	Displays information on the most previously marked research project in any of the other windows.
Back	Returns to game
Unit Design	Opens unit design window

The names of the events, methods, or procedures that cause this form to be displayed:
displayResearch()

The names of the events, methods, or procedures triggered by each control:
Back calls displayGameField()
Unit Design calls displayDesign()

4.10. Form 10 – Save game menu

Functional requirements

- Saving game

The names of the controls and fields:

Cancel	Returns the user to the Pause game menu
Saved Games	Displays all previously saved game names
Save Game Name	Input field for the save game name
Save Game Button	Saves the game with the specified name

The names of the events, methods, or procedures that cause this form to be displayed:
displaySaveMenu() in Form 7

The names of the events, methods, or procedures triggered by each control:
Save calls saveGame()
Cancel calls resumeGame()

4.11. Form 11 – Load game menu

Functional requirements

- Loading game

The names of the controls and fields:

Cancel	Returns the user to either the Pause game menu or the system's Main menu
Saved Games	Displays a selectable field with all previously saved game names
Load Game	Loads the selected game

The names of the events, methods, or procedures that cause this form to be displayed:
displayLoadGame()

The names of the events, methods, or procedures triggered by each control:
Load calls loadGame()
Cancel calls resumeGame()

5. Design Details

5.1. CRC cards

InputHandler	
Responsibilities	**Collaborators**
Translates user input to game events	

Kernel	
Responsibilities	**Collaborators**
Main application controller	SFXHandler InputHandler WindowManager Log GUI AI RendererStateManager GameManager Pathfinder NetworkManager NetworkServer MusicManager XMLHandler
TextureManager	
Responsibilities	**Collaborators**
Controls textures.	RendererStateManager FileHandler

MusicManager	
Responsibilities	**Collaborators**
Controls music.	Kernel FileHandler

Log	
Responsibilities	**Collaborators**
Record important events and executions Record errors.	

AI	
Responsibilities	**Collaborators**
Handles player simulations.	GameManager

NetworkManager	
Responsibilities	**Collaborators**
Manages network traffic and connections	Kernel

NetworkHandler	
Responsibilities	**Collaborators**
Parses the messages received by the NetworkManager	

GUI	
Responsibilities	**Collaborators**
Manages the graphical user interface	RendererStateManager

Pathfinder	
Responsibilities	**Collaborators**
Finds optimal paths for unit movement	

RendererStateManager	
Responsibilities	**Collaborators**
Controls the render state	

FileHandler	
Responsibilities	**Collaborators**
Handles the data streaming from file to other classes.	

XMLHandler	
Responsibilities	**Collaborators**
Parses and translates XML to game content (Java objects according to our classes) and game content to XML.	

WindowManager	
Responsibilities	**Collaborators**
Handles the display- and window settings for the render output	

SFXHandler	
Responsibilities	**Collaborators**
Handles SFX sound output, directly access files via use of OpenAL.	

GameManager	
Responsibilities	**Collaborators**
Handles the state of the game. Game logic and communication with the game engine.	MusicManager RendererStateManager AI *GameObject* XMLHandler GUI Map Player

Player	
Responsibilities	**Collaborators**
Contains information of the player	

GameObject	
Responsibilities	**Collaborators**
In-game object superclass	

Unit	
Responsibilities	**Collaborators**
Contains information of an existing unit in the game	Weapon

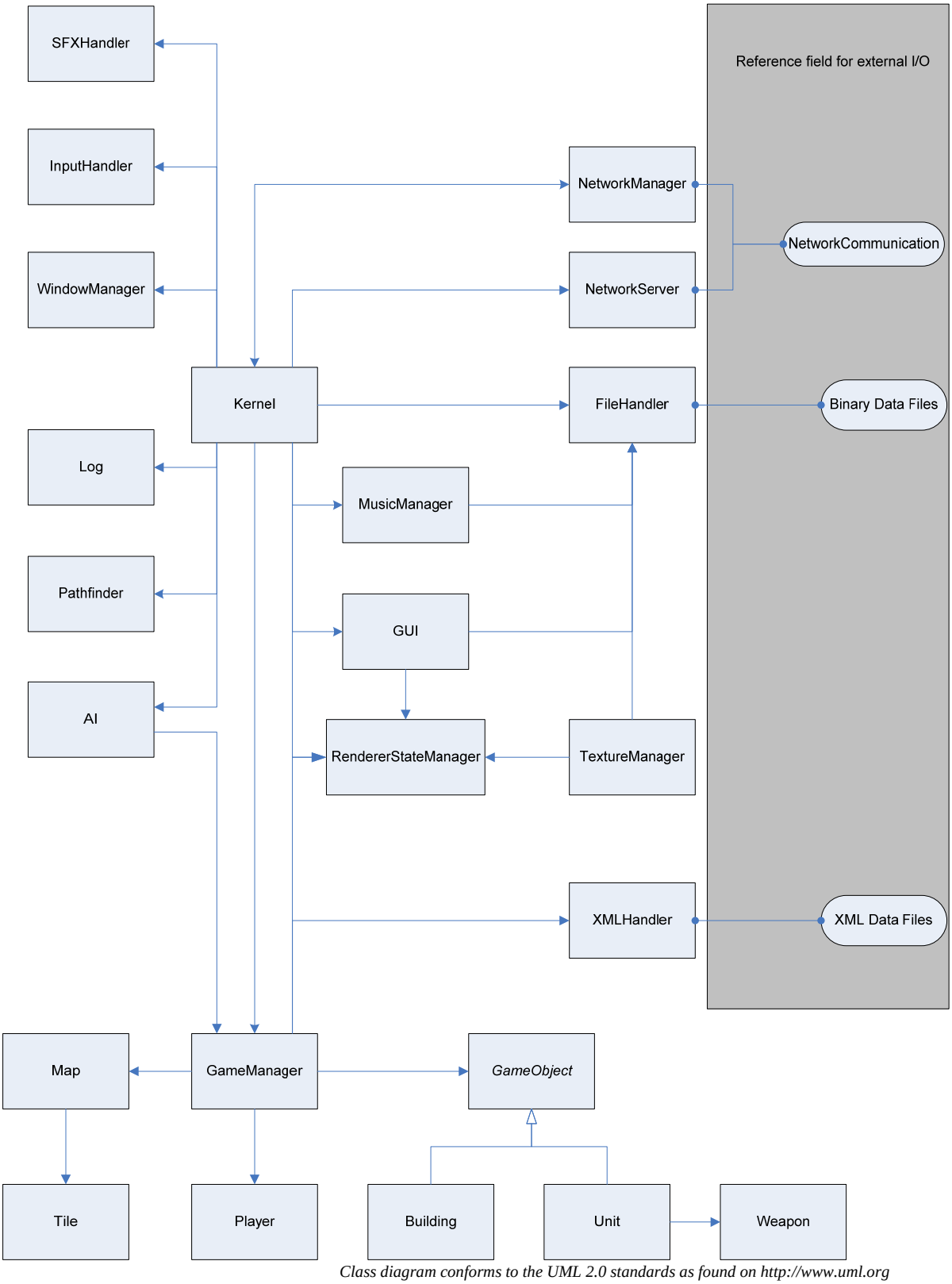
Weapon	
Responsibilities	**Collaborators**
Contains information of a weapon, it's characteristics, and any additional effects associated with it.	

Building	
Responsibilities	**Collaborators**
Contains information of an existing building in the game	

Map	
Responsibilities	**Collaborators**
Contains the collective information of the game map in use	Tile

Tile	
Responsibilities	**Collaborators**
Contains information of a tile on the terrain	

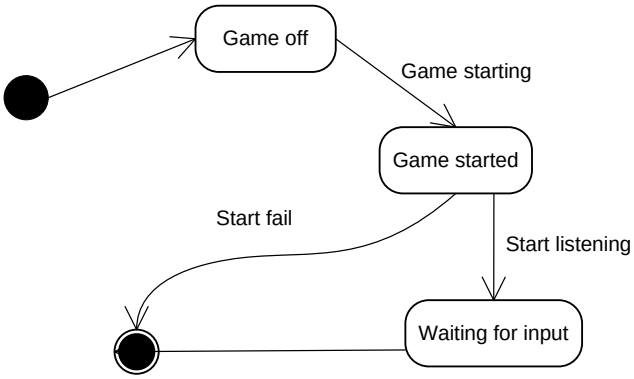
5.2. Class Diagram



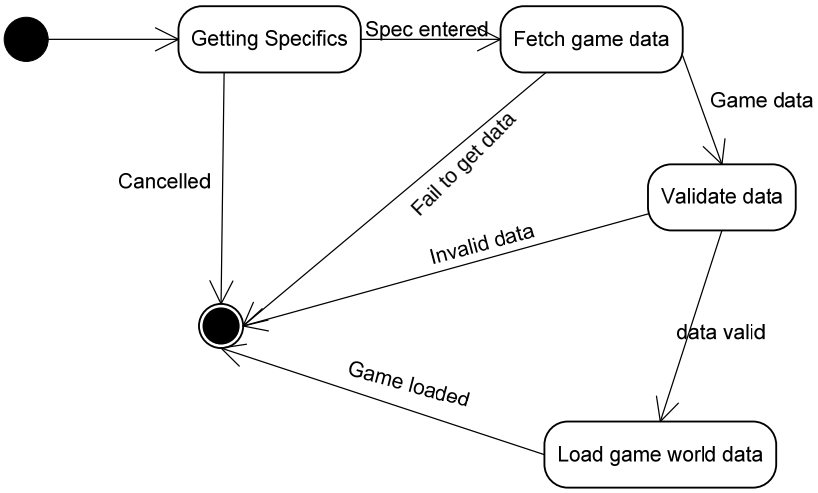
Class diagram conforms to the UML 2.0 standards as found on http://www.uml.org

5.3. State Charts

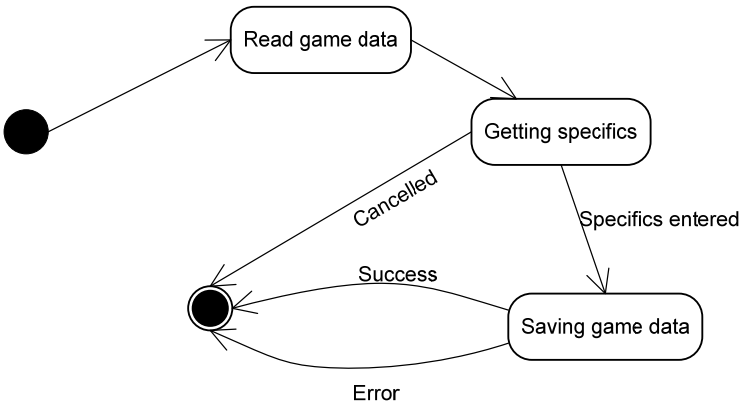
5.3.1.Start game



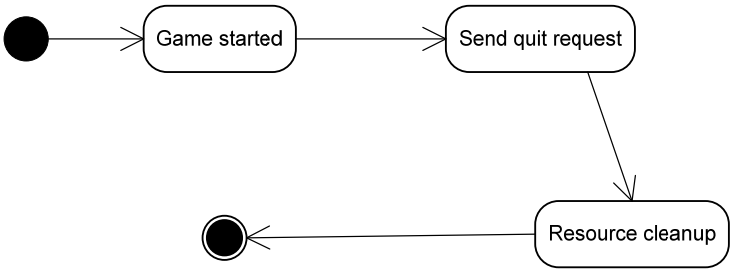
5.3.2.Load game



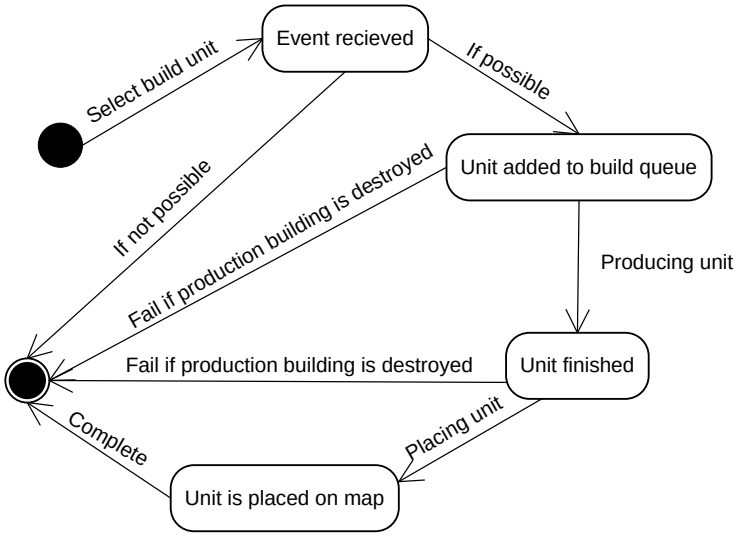
5.3.3.Save game



5.3.4.End game

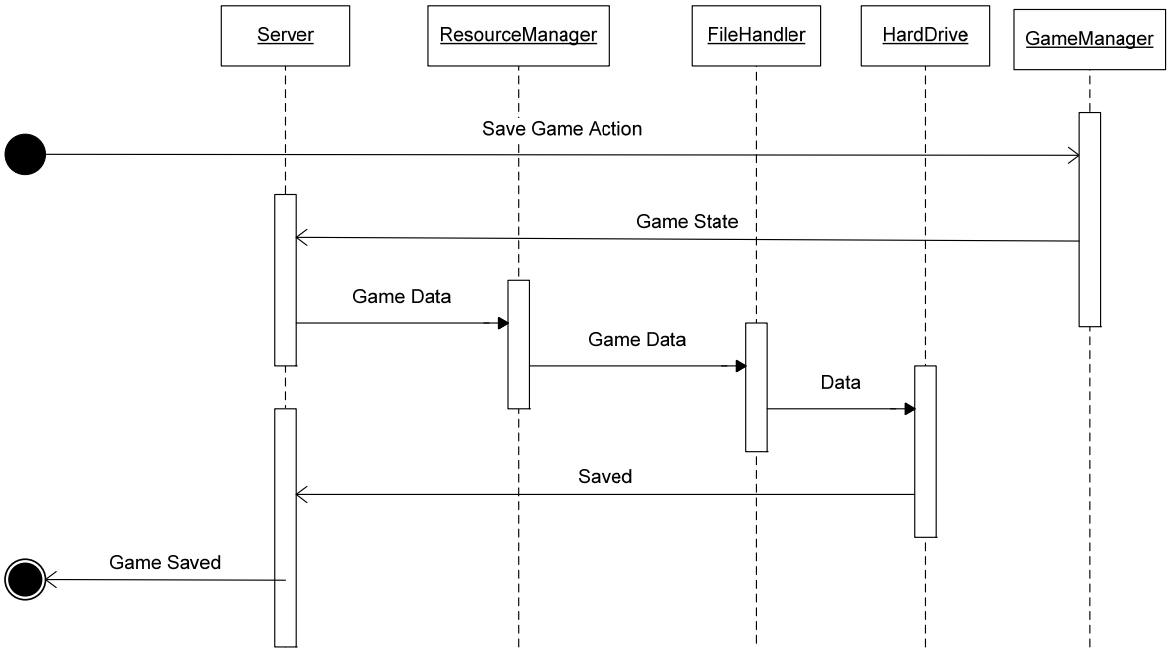


5.3.5.Building a unit

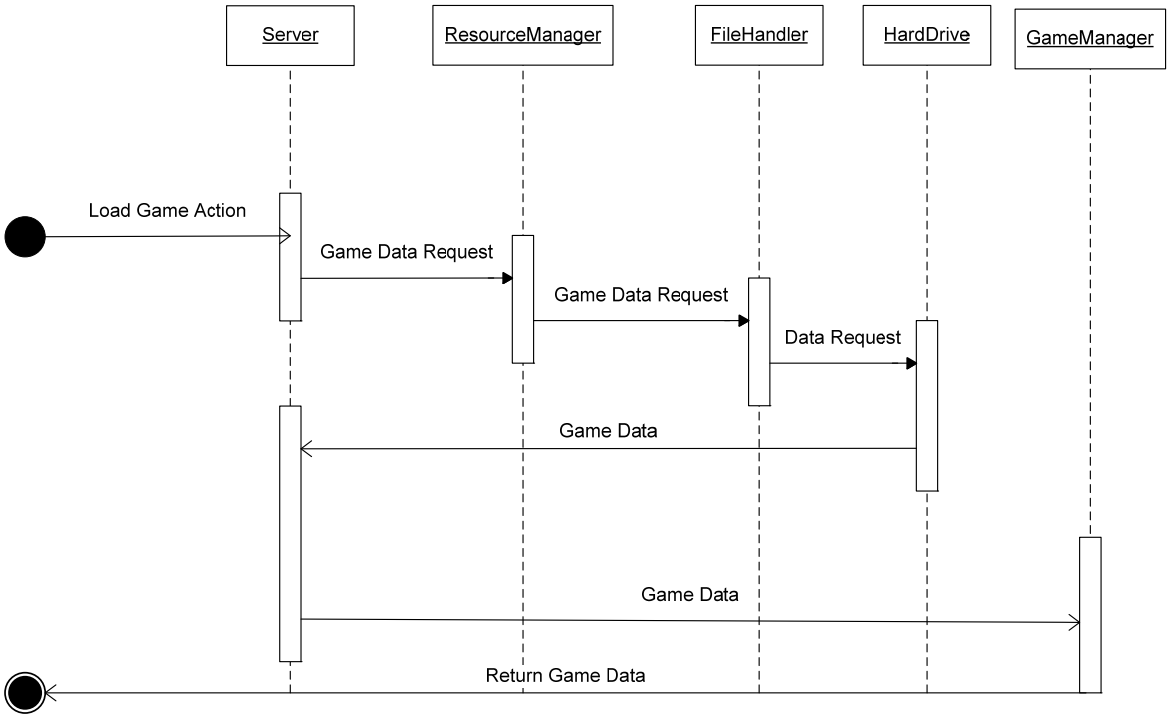


5.4. Interaction Diagrams

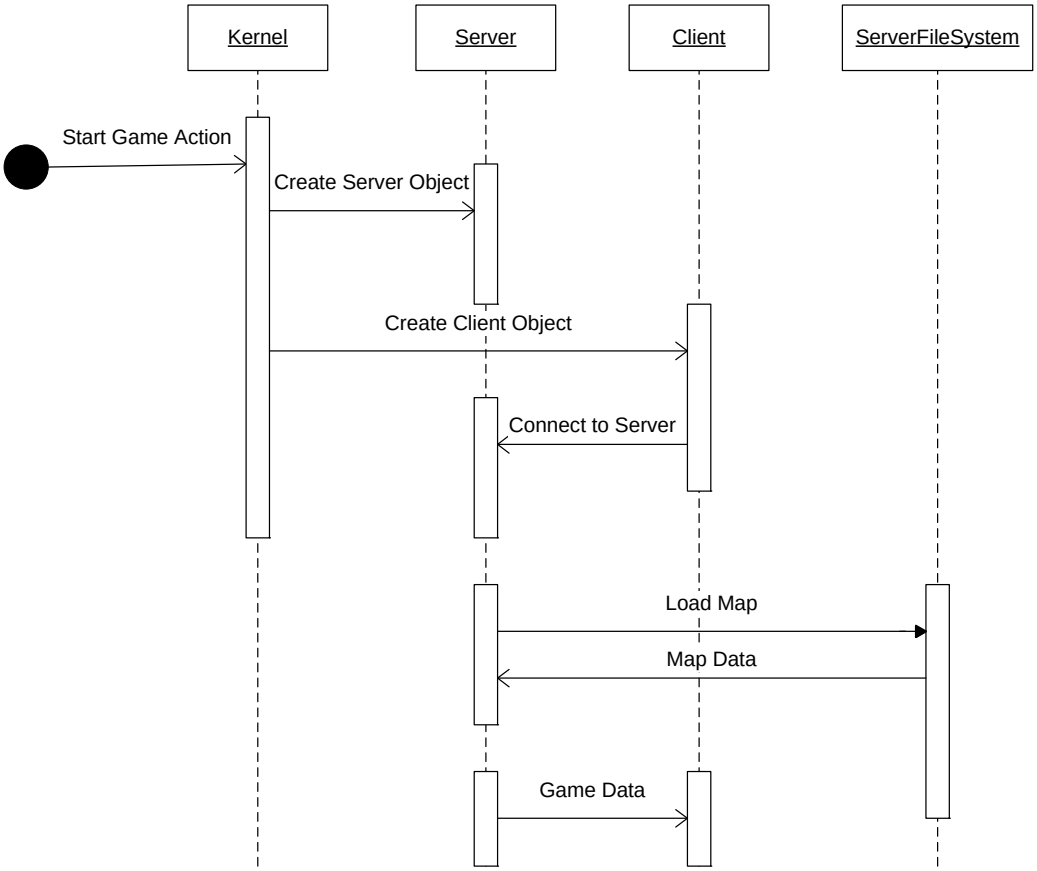
5.4.1.Save game



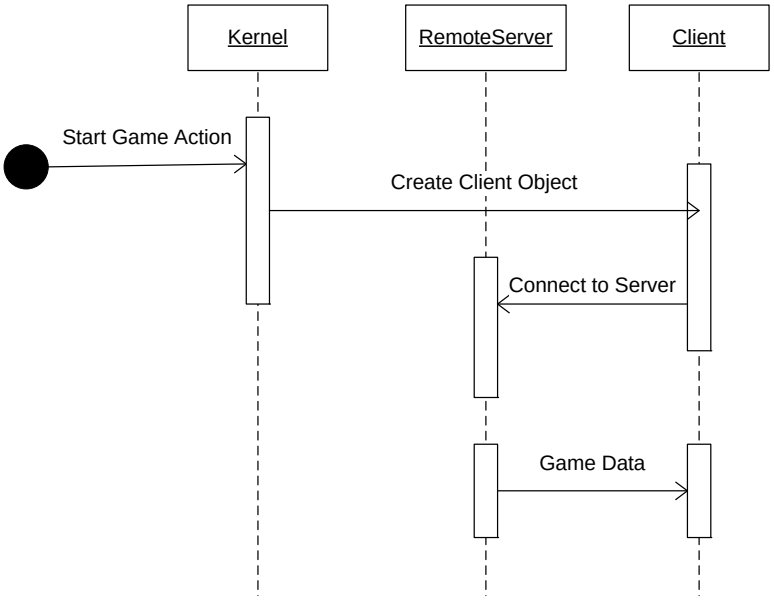
5.4.2.Load game



5.4.3.Host game



5.4.4. Client connect



5.5. Detailed design

5.5.1.Classes

WindowManager

Method summary:
setResolution
toggleFullscreen
setTitle

Functional Requirements:
6.1.10.1 Setting video options

Name: WindowManager(int x, int y, boolean fs)
x specifies width
y specifies height
fs specifies fullscreen
Return value: N/A
Description: Creates a window with the specified values
Pre-conditions:
Validity Checks: Checks if input values is >=1 or else sets it to 1.
Post-conditions: WindowManager is initialized
Called by: Kernel
Calls: N/A

Name: setResolution(int x, int y)
x specifies width
y specifies height
Return value: boolean
Returns true if the resolution change succeeds.
Description: Changes the current in-game resolution to the new specified resolution.
Pre-conditions: WindowManager is initialised
Validity Checks: Checks if input values is >=1 or else sets it to 1.
Post-conditions: New resolution is set.
Called by: Kernel
Calls: N/A

Name: toggleFullscreen()
Return value: void
Description: Toggles fullscreen.
Pre-conditions: WindowManager is initialised
Validity Checks: None
Post-conditions: Sets fullscreen mode if previous condition was window mode.
Called by: Kernel
Calls: N/A

Name: setTitle(string name)
name is a string containing the window title
Return value: void
Description: sets window title
Pre-conditions: WindowManager is initialised
Validity Checks: None
Post-conditions: Title was set.
Called by: Kernel
Calls: N/A

Log

Method summary:
writeLog

Functional Requirements:
N/A

Name: writeLog(String log)
log is written to the log file.
Return value: void
Description: Writes a string to a pre-specified log file.
Pre-conditions: Logger is initialized and has a specified output.
Validity Checks: Target file is specified.
Post-conditions: String is written to file.
Called by: Kernel
Calls: N/A

TextureManager

Method summary:
use

Functional Requirements:
N/A

Name: use(String name)
Name is the texture to be selected.
Return value: boolean
Returns true if the texture is found and selected.
Description: Selects the texture.
Pre-conditions: N/A
Validity Checks: Checks if the texture is loaded into memory, if not the texture manager checks for the texture on the file system and loads it if possible.
Post-conditions: Texture is selected
Called by: Building, Unit, Map, GUI
Calls: N/A

Tile

Method summary:
setTexture
setBlocked
isBlocked
isResource()
setResource()

Functional Requirements:
6.1.3.2 Harvestable resources

Name: Tile(String texture, boolean blocked)
texture is the texture identifier
blocked is a boolean value that indicates if the tile is traversable.
Return value: N/A
Description: Initializes the tile with a texture and sets true if it's blocked
Pre-conditions: Map is initialized
Validity Checks: Checks that the texture exists.
Post-conditions: Tile is initialized.
Called by: Map
Calls: N/A

Name: setTexture(String texture)
texture is the texture identifier
Return value: boolean
Returns true if a new texture is set.
Description: Changes the current tile's texture.
Pre-conditions:
Validity Checks: Checks that the texture exists.
Post-conditions: New texture is set.
Called by: GameManager
Calls: N/A

Name: setBlocked(boolean blocked)
blocked is a boolean value that indicates if the tile is traversable.
Return value: void
Description: Sets the blocked attribute.
Pre-conditions: Map is initialized
Validity Checks: N/A
Post-conditions: New blocked attribute is set.
Called by: GameManager
Calls: N/A

Name: isBlocked()
Return value: boolean
Returns true if blocked attribute is set
Description: Returns true if blocked attribute is set
Pre-conditions: Tile is initialized.
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager, AI, Pathfinder
Calls: N/A

Name: isResource()

Return value: boolean
Returns true if resource attribute is set
Description: Returns true if resource attribute is set
Pre-conditions: Tile is initialized.
Validity Checks: N/A
Post-conditions: N/A
Called by: Map, GameManager, Kernel
Calls: N/A

Name: setResource(boolean resource)
resource is the desired change to resource status
Return value: void
Description: Changes resource status as desired
Pre-conditions: Tile is initialized.
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Map

Method summary:
generateMap
loadMapFromFile
getSize
getTileMatrix
getTile
regenerateResources

Functional Requirements:
6.1.1.1 Starting a pre-made map
6.1.1.2 Starting a randomly generated map
6.1.3.2 Harvestable resources

Name: generateMap()
Return value: void
Description: Initiates a randomly generated map
Pre-conditions: GameManager is initialized.
Validity Checks: N/A
Post-conditions: The Map object is fully initialized
Called by: Kernel
Calls: N/A

Name: loadMapFromFile(String mapName)
mapName is the name of the map file to load
Return value: void
Description: Loads the specified map file into the class, initializing it
Pre-conditions: GameManager is initialized.
Validity Checks: Validates that the mapName is a valid map file
Post-conditions: The Map object is fully initialized
Called by: Kernel
Calls: N/A

Name: getSize()
Return value: int[]
Returns the size of the map
Description: Returns the size of the map, x and y in tiles.
Pre-conditions: Map is fully initialized
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel
Calls: N/A

Name: getTileMatrix()
Return value: Tile[][]
Returns a tile matrix representing the full terrain of the map
Description: Returns an array of Tile objects containing the full map info
Pre-conditions: Map is fully initialized
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GameManager
Calls: N/A

Name: getTile(int[] position)
Return value: Tile
Returns a tile matrix representing the terrain at the given position
Description: Returns a tile matrix representing the terrain at the given position
Pre-conditions: Map is fully initialized
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel
Calls: N/A

Name: regenerateResources()
Return value: void
Description: Regenerates resources in key areas of the map to prevent resource deadlocks
Pre-conditions: Map is fully initialized, game is in progress
Validity Checks: N/A
Post-conditions: Map is reseeded with resource tiles
Called by: GameManager
Calls: N/A

Building

Functional Requirements:
6.1.2.1 Building construction
6.1.3.1 Currency

Method summary:
getBuildingType
setBuildingType

Name: getBuildingType()
Return value: enum
Description: Returns an enum identifying the building type
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: setBuildingType(enum type)
type describes what kind of building it is.
Return value: void
Description: Sets an enum identifying the building type
Pre-conditions: N/A
Validity Checks: type is a valid building
Post-conditions: type is set.
Called by: GameManager
Calls: N/A

GameManager

Method summary:
initiateGame
createUpdate
createFull
updateState
updatePlayer
storeCustomGameObject
setState
getState

Functional Requirements:
6.1.1.1 Starting a pre-made map
6.1.1.2 Starting a randomly generated map
6.1.1.5 Pause
6.1.2.3 Primary production facilities
6.1.6.2 Designing units

Name: initiateGame(GameSettings gameSettings)
gameSettings is the identifier of all gui choices for the current game.
Return value: void
Description: Creates initial GameObject-, Player-, Map- and GameState-objects.
Pre-conditions: A GuiSettings-object must be created.
Validity Checks: Validates the preset GuiSettings.
Post-conditions: Game is initialized.
Called by: Kernel
Calls: N/A

Name: createUpdate()
Return value: Object gameUpdate
Returns a gameUpdate Object containing changes game since last update.
Description: Create the update object to be sent to all players.
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: NetworkServer
Calls: N/A

Name: createFull()
Return value: GameManager fullUpdate
Returns a full gamestate containing a complete game information.
Description: Creates complete update object to be sent to all players.
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: NetworkServer
Calls: N/A

Name: updateState(Event event)
event contains information to update the game.
Return value: void
Description: updates the game information according to the event.
Pre-conditions: An event has occured.

Validity Checks: The update is feasible.
Post-conditions: Update has been applied.
Called by: GameManager
Calls: N/A

Name: updatePlayer(Event playerEvent)
playerEvent contains information to update a player.
Return value: void
Description: updates the player information according to the event.
Pre-conditions: An event has occured.
Validity Checks: The update is feasible.
Post-conditions: The update has been applied.
Called by: GameManager
Calls: N/A

Name: storeCustomGameObject(CustomUnit cu)
cu is the information about the choices made in the GUI in form of a custom unit
Return value: void
Description: Stores custom GameObject.
Pre-conditions: A GuiSettings-object must be created.
Validity Checks: Validates the preset GuiSettings.
Post-conditions: A new custom unit-object is stored
Called by: Kernel
Calls: N/A

Name: setState(int state)
state identifies the state to be set
Return value: void
Description: Changes the current game state
Pre-conditions: A game is running
Validity Checks: N/A
Post-conditions: The game state is changed
Called by: GameManager
Calls: N/A

Name: getState()
Return value: int
returns the current game state
Description: Gets the current game state
Pre-conditions: A game is running
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GameManager
Calls: N/A

Unit

Method summary:
getWeapons
getSpeed
addPath
getNextStep
getMovementSound
getArmor

Functional Requirements:
6.1.2.2 Unit construction
6.1.2.4 Unit types
6.1.3.1 Currency
6.1.6.2 Designing units

Name: getWeapons()
Return value: Weapon
Returns a unit's Weapon
Description: Returns a Weapon Object
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: getSpeed()
Return value: int
Returns an int with the units maximum speed.
Description: Returns an int with the units maximum speed.
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: setPath(int[][] path)
Sets a movement path for a unit
Return value: void
Description: Sets the movement path of a unit to the supplied matrix
Pre-conditions: Unit exists
Validity Checks: N/A
Post-conditions: N/A
Called by: Pathfinder
Calls: N/A

Name: getNextStep()
Return value: int[]
returns the next movement for the unity if any
Description: Returns the next movement position of a unit if there any
Pre-conditions: Unit exists
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: getMovementSound()
Return value: String
Returns the filename of the movement sound
Description: Returns the name of the sound to be played while in movement
Pre-conditions: Unit exists
Validity Checks: N/A
Post-conditions: N/A
Called by: SFXHandler
Calls: N/A

Name: getArmor()
Return value: int
Returns an int with the units armor.
Description: Returns an int with the units armor
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: getType()
Return value: enum
Description: Returns an enum identifying the unit type
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: setType(enum type)
type describes what kind of unit it is.
Return value: void
Description: Sets an enum identifying the unit type
Pre-conditions: N/A
Validity Checks: type is a valid unit
Post-conditions: type is set.
Called by: GameManager
Calls: N/A

Weapon

Method summary:
getFiringSound
getTravelSound
getImpactSound
getDamage
getType

Functional Requirements:
6.1.4.3 Upgrading research
6.1.6.2 Designing units

Name: getFiringSound()
Return value: String
Returns a string with a filename
Description: Returns the weapons firing sound effect
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: SFXHandler
Calls: N/A

Name: getTravelSound()
Return value: String
Returns a string with a filename
Description: Returns the weapons travelling sound effect
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: SFXHandler
Calls: N/A

Name: getImpactSound()
Return value: String
Returns a string with a filename
Description: Returns the weapons impact sound effect
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: SFXHandler
Calls: N/A

Name: getDamage()
Return value: int
Returns an int containing the damage value
Description: Returns the weapons damage
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

Name: getType()
Return value: enum
Returns an enum containing weapon type
Description: Returns the weapons's type
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager
Calls: N/A

GameObject

Method summary:
getPosition
setPosition
render
setTexture
getHealth
setHealth
getDeathSound
getOrientation

Functional Requirements:
N/A

Name: getPosition()
Return value: int[]
Returns an int array with positional information
Description: Returns an int array with positional information of the GameObject
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager, AI, Kernel
Calls: N/A

Name: setPosition(int[] position)
position is an int array with the updated position
Return value: void
Description: Updates the position of the GameObject
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager, AI, Kernel
Calls: N/A

Name: render()
Return value: void
Description: Renders the current GameObject
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: Unit, Building, GUI, Map
Calls: N/A

Name: setTexture(string texture)
texture is the texture identifier
Return value: boolean
Returns true if a new texture is set.
Description: Changes the current GameObject's texture.
Pre-conditions: N/A
Validity Checks: Checks that the texture exists.
Post-conditions: New texture is set.
Called by: GameManager
Calls: N/A

Name: getHealth()
Return value: int
Returns an int with the current health
Description: Returns the GameObject's current health
Pre-conditions:
Validity Checks: N/A.
Post-conditions: N/A.
Called by: GameManager, GUI
Calls: N/A

Name: setHealth(int health)
health is the unit's current health
Return value: void
Description: Sets the GameObject's current health
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: New health is set
Called by: GameManager
Calls: N/A

Name: getDeathSound()
Return value: String
Returns the filename of the death sound
Description: Returns the name of the sound to be played when unit dies
Pre-conditions: Unit exists
Validity Checks: N/A
Post-conditions: N/A
Called by: SFXHandler
Calls: N/A

Name: getOrientation()
Return value: float[]
Returns a float array with the current orientation
Description: Returns a float array with the current orientation
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: GameManager, AI, Kernel
Calls: N/A

Player

Method summary:
getName
getFaction
getColor
getType
getAvailableUnits
getAvailableBuildings
getAvailableResearch
getPerformedResearch
setResearch
setBuildingConstruction
setBuildingConstructionQueue
setUnitConstruction
setUnitConstructionQueue
getAvailableModules
getResources
setResources
setMainBuilding

Functional Requirements:
6.1.2.1 Building construction
6.1.2.2 Unit construction
6.1.2.3 Primary production facilities
6.1.3.1 Currency
6.1.4.2 Research
6.1.4.2 Unlocking research
6.1.4.3 Upgrading research
6.1.5.2 Faction differences
6.1.8.3 Computer controlled opponent
6.1.8.4 Indestructible computer controlled neutral units
6.1.10.4 In-game name

Name: player(String name, int faction, int color, int type)
name specifies the name of the Player
faction identifies what faction the Player belongs to
color specifies what color the Player is
type specifies if the player is a local player, remote player or AI controlled player
Return value: N/A
Description: Creates a Player with the specified value
Pre-conditions:
Validity Checks: Checks if input values are valid.
Post-conditions: A Player object is created
Called by: Kernel
Calls: N/A

Name: getName()
Return value: String
Returns the name of the Player
Description: Returns the name of the Player
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, NetworkManager, GUI
Calls: N/A

Name: getFaction()
Return value: int
Returns the faction of the Player
Description: Returns the faction of the Player
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, NetworkManager, GUI
Calls: N/A

Name: getColor()
Return value: int
Returns the color of the Player
Description: Returns the color of the Player
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, NetworkManager, GUI
Calls: N/A

Name: getType()
Return value: int
Returns the type of the Player
Description: Returns the type of the Player
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, NetworkManager, GUI
Calls: N/A

Name: getAvailableUnits()
Return value: int[]
Returns which units the player can build
Description: Returns an integer array containing unit identifiers
Pre-conditions: Player exists.
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

Name: getAvailableBuildings()
Return value: int[]
Returns which buildings the player can build
Description: Returns an integer array containing building identifiers
Pre-conditions: Player exists.
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

Name: getAvailableResearch()
Return value: int[]
Returns which research the player can build
Description: Returns an integer array containing research identifiers
Pre-conditions: Player exists.
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

Name: setPerformedResearch(int research)
research is used to set what research a player is performing
Return value: void
Description: Adds a researched identifier to the performed research list
Pre-conditions: Player exists.
Validity Checks: Validates that the research exists and can be performed
Post-conditions: Player's research is started
Called by: Kernel
Calls: N/A

Name: setBuildingConstruction(int building)
Sets the players current building construction
Return value: void
Description: Sets a players current construction of a building
Pre-conditions: Player exists.
Validity Checks: Validates that the building exists and construction can be started
Post-conditions: Player's construction is started
Called by: Kernel
Calls: N/A

Name: setBuildingQueue(int[] buildings)
Sets the players current building construction queue
Return value: void
Description: Sets a players current construction queue of buildings
Pre-conditions: Player exists.
Validity Checks: Validates that the building exists and construction can be started
Post-conditions: Player's construction is started
Called by: Kernel
Calls: N/A

Name: setUnitConstruction(int unit)
Sets the players current unit construction
Return value: void
Description: Sets a players current construction of a unit
Pre-conditions: Player exists.
Validity Checks: Validates that the unit exists and construction can be started
Post-conditions: Player's construction is started
Called by: Kernel
Calls: N/A

Name: setUnitQueue(int[] units)
Sets the players current unit construction queue
Return value: void
Description: Sets a players current construction queue of unit
Pre-conditions: Player exists.
Validity Checks: Validates that the unit exists and construction can be started
Post-conditions: Player's construction is started
Called by: Kernel
Calls: N/A

Name: getAvailableCustomUnitParts()
Return value: int[]
Returns what custom unit parts are available to design a custom unit
Description: Returns what custom unit parts a player can use to design a custom unit
Pre-conditions: Player exists.
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

Name: getResources()
Return value: int
Returns the amount of resources player has
Description: Returns the current amount of resources the player controls
Pre-conditions: Player exists.
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

Name: setResources(int resources)
Sets a player's amount of resources
Return value: void
Description: Sets the player's resource amount
Pre-conditions: Player exists.
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GameManager
Calls: N/A

Name: setMainBuilding(int id, int type)
id is the ID of the building to set to main building
type the type of building
Return value: void
Description: Sets the player's main building of this type
Pre-conditions: Player exists
Validity Checks: Building exists
Post-conditions: N/A
Called by: Kernel
Calls: N/A

Pathfinder

Method summary:
calculatePath

Functional Requirements:
N/A

Name: calculatePath(int[] from, int[] to, Tile[][] terrain)
from is the deparature point
to is the destination point
terrain is a representation of the tiles of the map in a matrix
Return value: int[][]
returns an integer matrix containing the calculated path
Description: Creates a new movement path matrix
Pre-conditions: Map loaded and available.
Validity Checks: Validates that the positions are valid
Post-conditions: N/A
Called by: Kernel, AI
Calls: N/A

XMLHandler

Method summary:
saveXML
loadXML

Functional Requirements:
6.1.2.1 Building construction
6.1.2.2 Unit construction
6.1.2.4 Unit types
6.1.5.2 Faction differences
6.1.6.2 Designing units

Name: saveXML(CustomUnit cu, String file)
cu is a custom unit object to be saved
file is the name of the file to save the Object as
Return value: void
Description: Saves a custom unit to an XML file
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: XML file created
Called by: Kernel, GameManager
Calls: N/A

Name: loadXML(String file)
file is the XML file desired to be loaded
Return value: Object
returns the XML parsed as an object
Description: Loads any supported XML data into a data dictionary of the specified class, needing to be casted before use.
Pre-conditions: XML data type supported
Validity Checks: Valid XML supplied
Post-conditions: Object created
Called by: Kernel, GameManager
Calls: N/A

FileHandler

Method summary:
initialize
Read
write
createStream
getFileList
findFile

Functional Requirements:
6.1.1.1 Starting a pre-made map
6.1.1.3 Load
6.1.1.5 Save game state
6.1.10.3 Custom soundtrack folder

Name: initialize()
Return value: void
Description: Parses the filesystem to build an internal list of each supported filetype
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: Internal Collection of supported files is created
Called by: N/A
Calls: N/A

Name: read (String file)
file is the name of the file to load
Return value: Object
Returns the read file in the proper Object
Description: Reads and parses a binary file into an Object according to file extension
Pre-conditions: FileHandler initiated
Validity Checks: File exist and type supported
Post-conditions: Object representing the file data is created
Called by: Kernel, MusicManager, TextureManager, GUI
Calls: N/A

Name: write(Object object)
object is the name of the object to save in binary format
Return value: void
Description: Saves the object into a binary file
Pre-conditions: N/A
Validity Checks: Validates that the filename contains no illegal characters
Post-conditions: A binary file of the object is created in the filesystem
Called by: Kernel
Calls: N/A

Name: createStream(String file)
file is the name of the file to create a read stream to
Return value: InputStream
Returns a inputstream to the specified file
Description: Creates an appropiate InputStream for the specified file
Pre-conditions: FileHandler initiated
Validity Checks: File identifier is correct
Post-conditions: N/A
Called by: Kernel, MusicManager, TextureManager, GUI

Calls: N/A

Name: getFileList(int type)
type is the type of the desired file list
Return value: String[]
Returns an array with the existing files
Description: Returns a list of all the files that can be loaded
Pre-conditions: FileHandler initiated
Validity Checks: File type is correct
Post-conditions: N/A
Called by: MusicManager
Calls: N/A

AI

Method summary:
nextMove

Functional Requirements:
6.1.8.2 Defensive buildings entering combat
6.1.8.3 Computer controlled opponent
6.1.8.4 Indestructible computer controlled neutral units

Name: nextMove()
Return value: void
Description: Initiates the next move by the AI
Pre-conditions: AI Object exists and is initialized
Validity Checks: N/A
Post-conditions: GameManager objects edited according to AI move
Called by: Kernel
Calls: N/A

MusicManager

Method summary:
pause
play
setVolume
getVolume

Functional Requirements:
6.1.10.2 Setting audio volume
6.1.10.3 Custom soundtrack folder

Name: pause()
Return value: void
Description: Pauses the current playback
Pre-conditions: Music is played
Validity Checks: N/A
Post-conditions: Music is paused
Called by: GUI
Calls: N/A

Name: play()
Return value: void
Description: Starts playback of music
Pre-conditions: No music is played
Validity Checks: Validates that no music is played
Post-conditions: Plays music
Called by: GUI
Calls: N/A

Name: setVolume(int volume)
volume is the desired value to change volume to
Return value: void
Description: Changes volume of playback
Pre-conditions: N/A
Validity Checks: Validates that the value is within range
Post-conditions: N/A
Called by: Kernel
Calls: N/A

Name: getVolume()
Return value: int
Returns the current volume
Description: Returns the current value of the volume
Pre-conditions: MusicManager initialized
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

SFXHandler

Method summary:
playSFX
setVolume
getVolume

Functional Requirements:
6.1.10.2 Setting audio volume

Name: playSFX(String filename, int[] objectPosition, int[] cameraPosition)
filename is the name of the sound to be played
objectPosition the absolute position of the unit
cameraPosition is the absolute position of the user camera over the map
Return value: void
Description: Plays the given objects sound using the OpenAL support library
Pre-conditions: Unit exists
Validity Checks: N/A
Post-conditions: Sound played
Called by: Unit, Building, Weapon, GameManager, GUI
Calls: N/A

Name: setVolume(int volume)
volume is the desired value to change volume to
Return value: void
Description: Changes volume of playback
Pre-conditions: N/A
Validity Checks: Validates that the value is within range
Post-conditions: N/A
Called by: Kernel
Calls: N/A

Name: getVolume()
Return value: int
Returns the current volume
Description: Returns the current value of the volume
Pre-conditions: SFXHandler initialized
Validity Checks: N/A
Post-conditions: N/A
Called by: Kernel, GUI
Calls: N/A

NetworkManager

Method summary:
createConnection
sendNetworkObject
getStatus
getUpdateState
getFullState

Functional Requirements:
6.1.9.1 Starting a multiplayer game
6.1.9.3 Multiplayer chat
6.1.9.4 Multiplayer cheat control

Name: createConnection(String address)
address is the IP address of the host
Return value: void
Description: Creates a connection to be used for all network traffic
Pre-conditions: NetworkServer on host is started
Validity Checks: Valid IP address
Post-conditions: Connection established with server
Called by: Kernel
Calls: N/A

Name: sendNetworkObject(Object object)
object is the object to be sent over the network
Return value: void
Description: Sends a network object to the server
Pre-conditions: NetworkServer on host is started
Validity Checks: Valid IP address
Post-conditions: Connection established with server
Called by: Kernel
Calls: N/A

Name: getStatus()
Return value: int
Description: Returns an int with current status
Pre-conditions: Game is up and running
Validity Checks: N/A
Post-conditions: Status-int returned to the calling method
Called by: Kernel
Calls: N/A

Name: getUpdateState()
Return value: Object
Description: Returns the updated state
Pre-conditions: Update is available in NetworkManager buffer
Validity Checks: N/A
Post-conditions: Status-object returned to the calling method
Called by: Kernel, GameManager
Calls: N/A

Name: getFullState()
Return value: GameManager
Description: Returns the complete updated GameManager
Pre-conditions: Update is available in NetworkManager buffer
Validity Checks: N/A
Post-conditions: Complete GameManager-object returned to the calling method
Called by: GameManager
Calls: N/A

NetworkServer

Method summary:
startListener
stopListener
broadcastFullState
broadcastUpdateState

Functional Requirements:
6.1.6.4 Multiplayer designs
6.1.9.1 Starting a multiplayer game
6.1.9.2 Request multiplayer team
6.1.9.3 Multiplayer chat
6.1.9.4 Multiplayer cheat control

Name: startListener()
Return value: void
Description: Starts a listener for incoming network connections
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: Connections can be accepted by clients
Called by: Kernel
Calls: N/A

Name: stopListener()
Return value: void
Description: Stops the listener
Pre-conditions: Listener has to be active
Validity Checks: Validates that the listener is active
Post-conditions: Connections will not be accepted by clients any longer
Called by: Kernel
Calls: N/A

Name: broadcastFullState(GameManager gm)
gm is the current gamestate
Return value: void
Description: Sends the current gamestate to all clients
Pre-conditions: Clients are connected
Validity Checks: Valid IP address and port
Post-conditions: Connection established with all clients
Called by: Kernel
Calls: N/A

Name: broadcastUpdateState(Object update)
update is the updated gamestate
Return value: void
Description: Sends the update gamestate to all clients
Pre-conditions: Clients are connected
Validity Checks: Valid IP address and port
Post-conditions: Connection established with all clients
Called by: Kernel
Calls: N/A

RenderStateManager

Method summary:
setState
getState

Functional Requirements:
N/A

Name: setState(Object rs)
rs is a new render state the renderer is set to.
Return value: void
Description: Sets the render state of the renderer
Pre-conditions: WindowsManager is initialized
Validity Checks: N/A
Post-conditions: Renderer's new state has been set.
Called by: GUI, GameObjects, Map
Calls: N/A

Name: getState()
Return value: Object
Object contains the current render state
Description: Returns and object with the render state
Pre-conditions: WindowManager has been initialized
Validity Checks: N/A
Post-conditions: Connections will not be accepted by clients any longer
Called by: GUI, GameObjects, Map
Calls: N/A

GUI

Method summary:
createSurface
createButton
Render
listenInput

Functional Requirements:

6.1.1.1 Starting a pre-made map
6.1.1.2 Starting a randomly generated map
6.1.1.3 Load
6.1.1.4 Save
6.1.1.5 Pause
6.1.2.1 Building construction
6.1.2.2 Unit construction
6.1.2.3 Primary production facilities
6.1.4.1 Research
6.1.5.1 Faction selection
6.1.5.2 Faction differences
6.1.6.1 Design dialogue access
6.1.6.2 Designing units
6.1.7.1 Selecting a single unit or building
6.1.7.2 Selecting a group of units
6.1.9.1 Starting a multiplayer game
6.1.10.1 Setting video options
6.1.10.2 Setting audio volume
6.1.10.4 In-game name
6.1.11.1 Quit the game

Name: createSurface(int a, int b int x, int y, float r, float g, float b, float a)
a is the screen coordinate x-value
b is the screen coordinate y-value
x is the surface's width
y is the surface's height
r is the color value (red)
g is the color value (green)
b is the color value (blue)
a is the alpha value
Return value: int
Returns a surface identifier
Description: Creates a surface window and returns an identifier to the surface
Pre-conditions: WindowsManager is initialized
Validity Checks: N/A
Post-conditions: A new surface has been created
Called by: Kernel
Calls: N/A

Name: createButton(string text, string texture)
text is a string to be written on the button
texture is a texture identifier to be shown on the button
Return value: int
Returns an identifier to the the button
Description: Create a button and returns an identifier
Pre-conditions: WindowManager has been initialized
Validity Checks: N/A
Post-conditions: A button has been created.
Called by: Kernel
Calls: N/A

Name: render()
Return value: void
Description: Renders the GUI
Pre-conditions: WindowManager has been initialized
Validity Checks: N/A
Post-conditions: The GUI is rendered.
Called by: Kernel
Calls: N/A

Name: input(Object o)
Object takes an input and interprets it and calls the appropriate function
Return value: void
Description: Takes input and calls an appropriate function
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: New function is called
Called by: Kernel
Calls: N/A

Name: getActiveWindow()
Return value: int
Description: Returns the which window is active
Pre-conditions: N/A
Validity Checks: N/A
Post-conditions: Active window is returned
Called by: GameManager
Calls: N/A

5.5.2.Data dictionaries

Research

Field summary:	Type:
id	int
name	String
prerequisite	int[]
time	int
unlocksBuilding	int[]
unlocksUnit	int[]
unlocksCustomUnitPart	int[]
upgradesBuilding	int[][]
upgradesCustomUnitPart	int[][]
upgradesUnit	int[][]
faction	int

Functional Requirements:
6.1.4.1 Research
6.1.4.2 Unlocking research
6.1.4.3 Upgrading research
6.1.5.2 Faction differences
6.1.6.2 Designing units

Name: id
Description: Identifier for this particular research
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: name
Description: Name of the research object
Dependencies: none
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: prerequisite
Description: Previous research needed to perform this research
Dependencies: Value has to be a known research identifier
Integrity: Must fit in a 32bit signed integer

Name: time
Description: Time needed in seconds to perform this research
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: unlocksBuilding
Description: Array of identifiers of buildings that this research unlocks
Dependencies: Must be a known building identifier
Integrity: Must fit in a 32bit signed integer

Name: unlocksUnit
Description: Array of identifiers of units that this research unlocks
Dependencies: Must be a known unit identifier
Integrity: Must fit in a 32bit signed integer

Name: unlocksCustomUnitPart
Description: Array of identifiers of custom unit parts that this research unlocks
Dependencies: Must be a known custom unit part identifier
Integrity: Must fit in a 32bit signed integer

Name: upgradesBuilding
Description: 3 x n matrix, n is the amount of upgraded buildings with this research. First value identifies the buildings that this research upgrades, second value what stat is changed, third value specifies the amount changed of the value.
Dependencies: Must be known building and stats identifiers.
Integrity: All values must fit in 32bit signed integers.

Name: upgradesUnit
Description: 3 x n matrix, n is the amount of upgraded units with this research. First value identifies the units that this research upgrades, second value what stat is changed, third value specifies the amount changed of the value.
Dependencies: Must be known units and stats identifiers.
Integrity: All values must fit in 32bit signed integers.

Name: upgradesCustomUnitPart
Description: 3 x n matrix, n is the amount of upgraded custom unit parts with this research.. First value identifies the custom unit part that this research upgrades, second value what stat is changed, third value specifies the amount changed of the value.
Dependencies: Must be known custom unit part and stats identifiers.
Integrity: All values must fit in 32bit signed integers.

Name: faction
Description: Defines for what factions this research is available
Dependencies: Must be a known faction identifier.
Integrity: All values must fit in 32bit signed integers.

Units

Field summary:	Type:
id	int
name	String
prerequisite	int[]
time	int
faction	int
type	int
cost	int
health	int
speed	int
weapon	int
armor	int

Functional Requirements:
6.1.2.2 Unit construction
6.1.2.4 Unit types
6.1.4.1 Research
6.1.5.2 Faction differences

Name: id
Description: Identifier for this particular unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: name
Description: Name of this unit
Dependencies: none
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: prerequisite
Description: Specifies what research is needed to construct this unit.
Dependencies: Must be a known research identifier.
Integrity: Must fit in a 32bit signed integer.

Name: time
Description: Specifies the amount of time in seconds to construct one of this unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: faction
Description: Specifies what faction can build this unit.
Dependencies: Must be a known faction identifier.
Integrity: Must fit in a 32bit signed integer.

Name: type
Description: Specifies type of unit
Dependencies: Must be a known type identifier.
Integrity: Must fit in a 32bit signed integer.

Name: cost
Description: Specifies the cost of the unit
Dependencies: none

Integrity: Must fit in a 32bit signed integer.

Name: health
Description: Specifies the maximum health of the unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: speed
Description: Specifies the speed of the unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: weapon
Description: Specifies the weapon of the unit
Dependencies: Must be a known weapon identifier.
Integrity: Must fit in a 32bit signed integer.

Name: armor
Description: Specifies the armor of the unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Buildings

Field summary:	Type:
id	Int
name	String
prerequisite	Int[]
time	Int
faction	Int
cost	Int
health	Int

Functional Requirements:
6.1.2.1 Building construction
6.1.4.1 Research
6.1.5.2 Faction differences

Name: id
Description: Identifier for this particular building
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: name
Description: Name of this building
Dependencies: none
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: prerequisite
Description: Specifies what research is needed to construct this building.
Dependencies: Must be a known research identifier.
Integrity: Must fit in a 32bit signed integer.

Name: time
Description: Specifies the amount of time in seconds to construct one of this building
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: faction
Description: Specifies what faction can build this building.
Dependencies: Must be a known faction identifier.
Integrity: Must fit in a 32bit signed integer.

Name: cost
Description: Specifies the cost of the building
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: health
Description: Specifies the maximum health of the unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Factions

Field summary:	Type:
id	int
name	String

Functional Requirements:
6.1.5.1 Faction selection
6.1.5.2 Faction differences

Name: id
Description: Identifier for this faction
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: name
Description: Name of this faction
Dependencies: none
Integrity: Must fit in a String object and contain only alphanumerical characters

CustomUnitPart

Field summary:	Type:
id	int
name	String
prerequisite	int[]
cost	int
faction	int
value	int
type	int

Functional Requirements:
6.1.5.2 Faction differences
6.1.6.2 Designing units
6.1.6.3 Design budget

Name: id
Description: Identifier for this particular custom unit part
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: name
Description: Name of this custom unit part
Dependencies: none
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: prerequisite
Description: Specifies what research is needed to use this custom unit part
Dependencies: Must be a known research identifier.
Integrity: Must fit in a 32bit signed integer.

Name: cost
Description: Specifies the custom design cost this custom unit part adds
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: faction
Description: Specifies what faction can use this custom unit part.
Dependencies: Must be a known faction identifier.
Integrity: Must fit in a 32bit signed integer.

Name: value
Description: Specifies how much the stat tied to this type of custom unit part is changed
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: type
Description: Specifies what type of custom unit part
Dependencies: Must be a known custom unit part type.
Integrity: Must fit in a 32bit signed integer.

CustomUnit

Field summary:	Type:
id	int
name	String
prerequisite	int[]
cost	int
faction	int
value	int
type	int
health	int
speed	int
weapon	int
armor	int

Functional Requirements:
6.1.2.2 Unit construction
6.1.2.4 Unit types
6.1.4.1 Research
6.1.5.2 Faction differences
6.1.6.2 Designing units

Name: id
Description: Identifier for this particular custom unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer

Name: name
Description: Name of this custom unit
Dependencies: none
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: prerequisite
Description: Specifies what research is needed to construct this custom unit.
Dependencies: Must be a known research identifier.
Integrity: Must fit in a 32bit signed integer.

Name: time
Description: Specifies the amount of time in seconds to construct one of this custom unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: faction
Description: Specifies what faction can build this custom unit.
Dependencies: Must be a known faction identifier.
Integrity: Must fit in a 32bit signed integer.

Name: type
Description: Specifies type of custom unit
Dependencies: Must be a known type identifier.
Integrity: Must fit in a 32bit signed integer.

Name: cost
Description: Specifies the cost of the custom unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: health
Description: Specifies the maximum health of the custom unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: speed
Description: Specifies the speed of the custom unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

Name: weapon
Description: Specifies the weapon of the custom unit
Dependencies: Must be a known weapon identifier.
Integrity: Must fit in a 32bit signed integer.

Name: armor
Description: Specifies the armor of the custom unit
Dependencies: none
Integrity: Must fit in a 32bit signed integer.

GameSettings

Field summary:	Type:
mapname	String
type	bit
players	int
playersName	String[]
playersFaction	int[]
hostAddress	String

Functional Requirements:
6.1.1.1 Starting a pre-made map
6.1.1.2 Starting a randomly generated map
6.1.5.1 Faction Selection
6.1.9.1 Starting a Multiplayer game
6.1.9.2 Request Multiplayer team
6.1.10.4 In-game name

Name: mapname
Description: Name of the map to load, *random* if a random map
Dependencies: Map name must exist as a map file or be *random*
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: type
Description: Specifies type of game
Dependencies: Must be a known game type.
Integrity: Must fit in a bit.

Name: players
Description: Amount of players in the game.
Dependencies: Must be no more than 8 and no less than 2.
Integrity: Must fit in a 32bit signed integer.

Name: playersName
Description: Name of all the players in the game
Dependencies: Must be no more than 8 and no less than 2.
Integrity: Must fit in a String object and contain only alphanumerical characters

Name: playersFaction
Description: Faction of each player in the game
Dependencies: Must be known faction identifier
Integrity: Must fit in a 32bit signed integer.

Name: hostAdress
Description: IP address of the host computer.
Dependencies: Must be a valid IP address representing the host machine
Integrity: Must fit the pattern x.x.x.x, where x is a number 0-255.

5.5.3.Enumerations

Terrain	List of all terrains
Weapons	List of all weapons
Stat	List of all stats
CustomUnitPartType	List of all custom unit part types
UnitType	List of all unit types
BuildingType	List of all buildings
EventType	List of all events
Color	List of player colors
FileType	List of all file types

Functional requirements:
6.1.2.4 Unit types
6.1.3.2 Harvestable resources
6.1.4.1 Research
6.1.6.2 Designing units

5.5.4. Cross reference

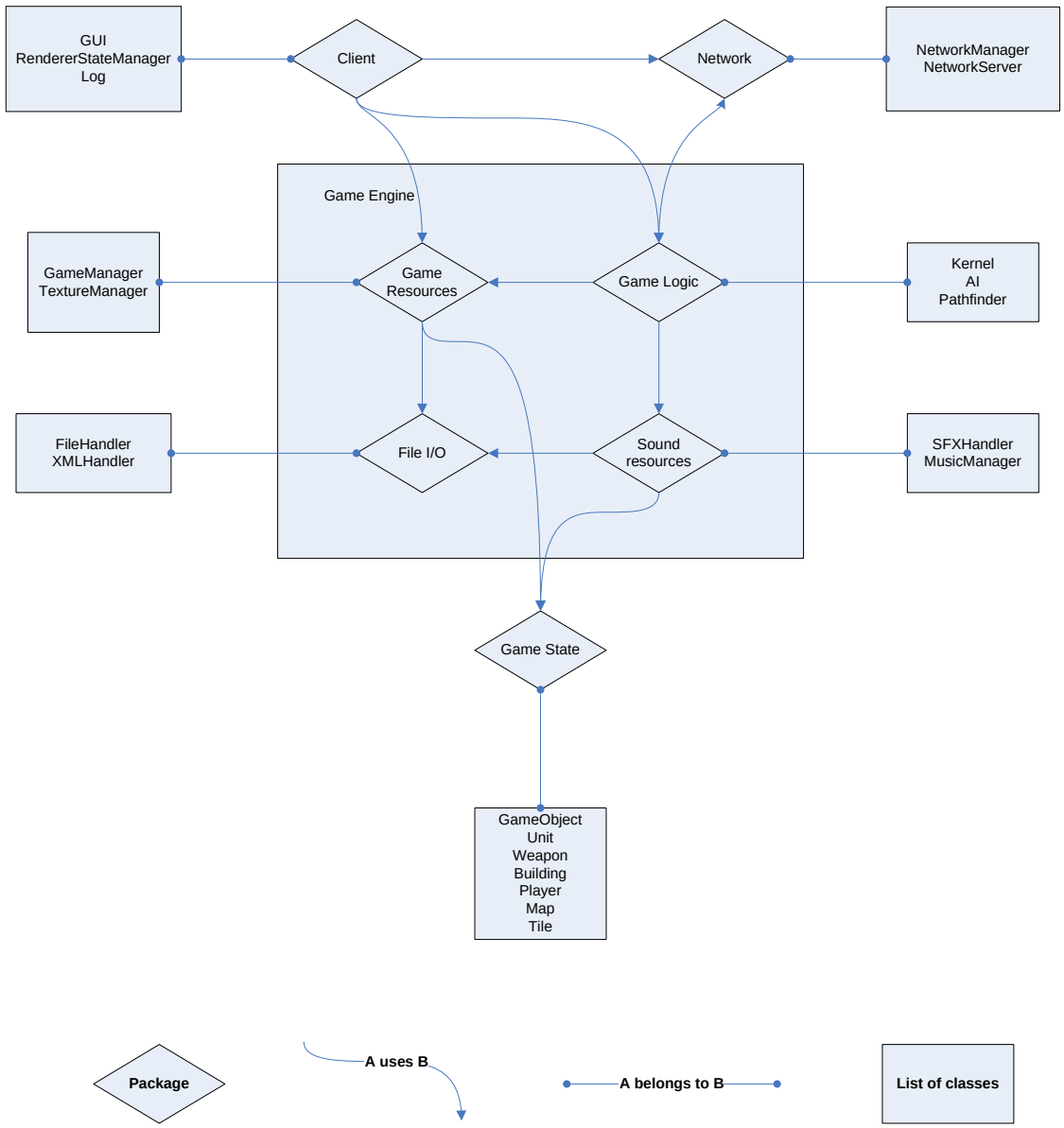
6.1.1.1 Starting a pre-made map	GUI GameSettings FileHandler GameManager Map
6.1.1.2 Starting a randomly generated map	GameSettings GUI GameManager Map
6.1.1.3 Load	GUI FileHandler
6.1.1.4 Save	GUI
6.1.1.5 Pause	GUI GameManager
6.1.1.5 Save game state	FileHandler
6.1.10.2 Setting audio volume	GUI SFXHandler MusicManager WindowManager
6.1.10.3 Custom soundtrack folder	MusicManager FileHandler
6.1.10.4 In-game name	GUI GameSettings Player
6.1.11.1 Quit the game	GUI
6.1.2.1 Building construction	GUI Buildings XMLHandler Player Building
6.1.2.2 Unit construction	GUI CustomUnit Units XMLHandler Player Unit
6.1.2.3 Primary production facilities	GUI Player GameManager
6.1.2.4 Unit types	Enumerations CustomUnit Units XMLHandler Unit
6.1.2.5 Production shortcuts	InputHandler
6.1.3.1 Currency	Player Unit Building
6.1.3.2 Harvestable resources	Enumerations Map Tile
6.1.4.2 Research	Player

	GUI
	Enumerations
	CustomUnit
	Buildings
	Units
	Research
6.1.4.2 Unlocking research	Research
	Player
6.1.4.3 Upgrading research	Weapon
	Research
	Player
6.1.5.1 Faction selection	GUI
	GameSettings
	Factions
6.1.5.2 Faction differences	GUI
	CustomUnit
	CustomUnitPart
	Factions
	Buildings
	Units
	Research
	XMLHandler
	Player
6.1.6.1 Design dialogue access	GUI
6.1.6.2 Designing units	GUI
	Enumerations
	CustomUnit
	CustomUnitPart
	Research
	XMLHandler
	Weapon
	Unit
	GameManager
6.1.6.3 Design budget	CustomUnitPart
6.1.6.4 Multiplayer designs	NetworkServer
6.1.7.1 Selecting a single unit or building	GUI
	InputHandler
6.1.7.2 Selecting a group of units	GUI
	InputHandler
6.1.7.3 Controlling units with the mouse	InputHandler
6.1.7.4 Controlling units with keyboard	InputHandler
6.1.8.1 Controlling units in combat	InputHandler
6.1.8.2 Defensive buildings entering combat	AI
6.1.8.3 Computer controlled opponent	AI
	Player
6.1.8.4 Indestructible computer controlled neutral units	AI
	Player
6.1.9.1 Starting a Multiplayer game	GameSettings
	GUI
	NetworkServer
	NetworkManager
6.1.9.2 Request Multiplayer team	GameSettings
	NetworkServer
6.1.9.3 Multiplayer chat	NetworkServer

6.1.9.4 Multiplayer cheat control

NetworkManager
NetworkServer
NetworkManager

5.6. Package Diagram



6. Functional Test Cases

6.1. Pre-Game tests

6.1.1. Configuring the game

Description: The user shall be able to configure settings for video resolution, sound effects volume, music volume and in-game name. The settings are automatically saved when change is detected.
Reference in RD: 6.1.10.1 Setting video options, 6.1.10.2 Setting audio volume
Initial system state: System is loaded and is in the Options menu
Expected outcome: Settings are saved.
Procedure:

1. Set Video resolution to 1024x768.
2. Set Sound effects volume to 70%
3. Set Music volume to 70%
4. Leave menu by clicking on Singleplayer button
5. Return to Options menu and verify that changes are saved

6.1.2. Starting a pre-made map

Description: Initializing a pre-made map within the game environment
Reference in RD: 6.1.1.1 Starting a pre-made map, 6.1.5.1.Faction selection
Initial system state: System is loaded and is in the system menu
Expected outcome: The game loads a specified map to a playable state.
Procedure:

1. Click on “Singleplayer” button
2. Chose the Atreides faction from the drop-down menu
3. Enter “Paul” in the screen name input field
4. Use the “Next” button to find the “Desert Basin” map
5. Click “Launch Game” button
6. Verify that game has started and chosen map is loaded

6.1.3. Starting a randomly generated map

Description: Initializing a random map within the game environment
Reference in RD: 6.1.1.2 Starting a randomly generated map, 6.1.5.1.Faction selection
Initial system state: System is loaded and is in the system menu
Expected outcome: The game loads a randomly generated map to a playable state.
Procedure:

1. Click on “Singleplayer” button
2. Chose the Atreides faction from the drop-down menu
3. Enter “Paul” in the screen name input field
4. Use the “Next” button to find the “Random” map
5. Click “Launch Game” button
6. Verify that game has started
7. Repeat test and verify that maps are different

6.1.4. Starting a Multiplayer game – Host

Description: Starting a Multiplayer game as a host with one remote player.
Reference in RD: 6.1.9.1 Starting a Multiplayer game, 6.1.5.1.Faction selection
Initial system state: System is loaded and is in the system menu. A client system is performing the test “Starting a Multiplayer – Client” at the same time.
Expected outcome: The game loads a specified map with only one multiplayer client.
Procedure:

1. Click on “Multiplayer” button
2. Click on “Host game”
3. Enter the screen name “Server” in the text field
4. Chose the “Atreides” faction from the drop-down menu
5. Use the “Next” button to find the “Desert Basin” map
6. Verify that clients have connected by checking that a player slot has changed from “Open” to the name chosen by the client.
7. Wait for client to be ready – the “Start game” button will be grayed out until then
8. Press “Start game”
9. Verify that game has started and clients are connected

6.1.5.Starting a Multiplayer game – Client

Description: Starting a Multiplayer game as a client connecting to a host
Reference in RD: 6.1.9.1 Starting a Multiplayer game, 6.1.5.1.Faction selection
Initial system state: System is loaded and is in the system menu. A host system is performing the test "Starting a Multiplayer – Host" at the same time.
Expected outcome: The game loads a specified map connected to a host.
Procedure:

1. Click on "Multiplayer" button
2. Select the game listed as "Server's game"
3. Click on "Join game"
4. Enter the screen name "Client" in the text field
5. Chose the "Harkonnen" faction from the drop-down menu
6. Press "Ready"
7. Wait for Host to start game
8. Verify that game starts

6.1.6.Load a saved game

Description: Resumes a previously saved game within the game environment
Reference in RD: 6.1.1.3 Load
Initial system state: System is loaded and is in the system menu with test save game available.
Expected outcome: The game loads a saved game file to a playable state.
Procedure:

1. Click on "Load Game" button
2. Chose the "Paul's Basin" from the saved game list
3. Click Load button
4. Verify that game has started with all the parameters from the previous game

6.2. In-game tests

6.2.1. Pause the game

Description: Pauses the game session
Reference in RD: 6.1.1.5 Pause
Initial system state: System is loaded and is running a single player game
Expected outcome: The game is paused and displays the pause menu.
Procedure:

1. Press ESC
2. Verify that game has paused

6.2.2. Resume the game

Description: Resumes the game session
Reference in RD: 6.1.1.5 Pause
Initial system state: System is loaded and is running a single player game that is paused
Expected outcome: The game is resumed.
Procedure:

1. Press ESC to resume game
2. Verify that game resumes play

6.2.3. Saving the game

Description: Saves the game to a file on the hard drive
Reference in RD: 6.1.1.4 Save game state
Initial system state: System is running a single player game
Expected outcome: The game saves the current game to be loaded later.
Procedure:

1. Access the pause menu
2. Click Save Game button
3. Enter “Paul’s Basin” in the save game name input field
4. Click “Save” button
5. Perform test 1.f to verify saved game

6.2.4.Produce a building

Description: Produces and place a building on the game map
Reference in RD: 6.1.2.1 Building construction
Initial system state: The system is running a game and the player has enough credits to create “Barracks”.
Expected outcome: A building is constructed and placed on the world map.
Procedure:

1. Click on “Barracks” icon on the right side-panel
2. Wait until the game announces that the building is completed
3. Click on the “Barracks” icon, which should now have changed to display that it is ready.
4. Click on a 2x2 free stone field near the construction yard on the map
5. Verify that building is placed on map

6.2.5.Produce a unit

Description: Produces a unit to the game map
Reference in RD: 6.1.2.2 Unit construction
Initial system state: The game is running and a infantry production structure has been placed on the map, the player has enough credits to create infantry unit.
Expected outcome: A unit is produced and placed near a production facility on the game map.
Procedure:

1. Click on the “Light infantry” icon on the right side-panel
2. Wait for the game to announce that the unit is complete
3. Verify that unit comes out of primary construction facility

6.2.6.Designating primary construction facility

Description: Specifies the building where produced units are placed near.
Reference in RD: 6.1.2.3 Primary production facilities
Initial system state: The game is running and more than one basic infantry production facility of the same type has been produced and placed on the map.
Expected outcome: The game places newly produced units at the most recently specified building.
Procedure:

1. Click on the infantry production building.
2. Right-click on the selected infantry production building
3. Click on the “Light infantry” icon on the right side-panel
4. Wait for the game to announce that the unit is completed
5. Verify that unit comes out of primary construction facility

6.2.7. Shortcut, production

Description: Produce a unit without using the GUI.
Reference in RD: 6.1.2.5 Production shortcuts
Initial system state: The game is running and the player has access to an infantry facility.
Expected outcome: The game produces an infantry unit without direct interaction with the GUI.
Procedure:

1. Press I
2. Press L
3. Verify that a “Light Infantry” is being produced.

6.2.8. Multiplayer chat

Description: Chatting with all other multiplayer players
Reference in RD: 6.1.9.3 Multiplayer chat
Initial system state: Two systems are connected with a multiplayer session and the game is in progress.
Expected outcome: The chat message appears in the user feedback area on all connected player.
Procedure:

1. Press “Return” on the keyboard to initiate chat writing mode.
2. Enter the text “Hello world”
3. Press “Return” on the keyboard to send the message
4. Verify text received on other game connected computer

6.3. In-game Research tests

6.3.1. Research, procedure

Description: Describing the procedure for engaging research.
Reference in RD: 6.1.4.1 Research, 6.1.4.3 Upgrading Research
Initial system state: The game is running, the player’s in-game base meets the sufficient requirements for committing research.
Expected outcome: Research of “Armor Piercing Weapons” is engaged.
Procedure:

1. Click on the “Research” button on the right side-panel
2. Click on “Armor Piercing Weapons” in the “Available Research” list.
3. Click the “Start Research” bar, which will turn into a progress bar.

6.3.2. Credits, unlocking new research

Description: Do research to access new research.
Reference in RD: 6.1.4.1 Research, 6.1.4.2 Unlocking Research, 6.1.4.3 Upgrading Research
Initial system state: The game is running, the player's in-game base meets the sufficient requirements for committing research, i.e. enough resources are available for research and building prerequisites are fulfilled. The player is currently viewing the research menu.
Expected outcome: Research of "Advanced Armor Piercing Weapons" is engaged.
Procedure:

1. Click on "Armor Piercing Weapons" in the "Available Research" list
2. Click the "Start Research" bar
3. When this research is done, it is added to the Finished Research pane. "Advanced Armor Piercing Weapons" is now added to the list of available research.
4. Click on "Advanced Armor Piercing Weapons" in the "Available Research" list
5. Click on the "Start Research" bar

6.3.3. Research, unlocking new building construction alternatives

Description: Researching new technology to advance the construction alternatives in the base.
Reference in RD: 6.1.4.1 Research, 6.1.4.2 Unlocking Research
Initial system state: The game is running, the player's in-game base meets the sufficient requirements for committing research. The player is currently viewing the research menu
Expected outcome: Construction of a land vehicle factory becomes available in the main graphical user interface.
Procedure:

1. Click on "Terrain Vehicle Design" in the "Available Research" list
2. Click the "Start Research" bar, which will turn into a progress bar.
3. Leave the Research menu and await research completion
4. Verify that research has been completed

6.3.4. Research, unlocking new unit construction alternatives

Description: Researching new technology to advance the construction alternatives in the base.
Reference in RD: 6.1.4.1 Research, 6.1.4.2 Unlocking Research
Initial system state: The game is running, the player's in-game base meets the sufficient requirements for committing research. The player is currently viewing the research menu
Expected outcome: Construction of Heavy Infantry becomes available in the main graphical user interface.
Procedure:

1. Click on "Exo-skeleton" in the "Available Research" list
2. Click the "Start Research" bar, which will turn into a progress bar.
3. Leave the Research menu and await research completion
4. Verify that research has been added to the completed research list

6.4. In-game Unit design tests

6.4.1. Opening the design menu

Description: Display the in-game unit design menu.
Reference in RD: 6.1.6.1 Design dialogue access
Initial system state: The system is running a game, and the player is able to interface with the main GUI.
Expected outcome: The design menu is displayed.
Procedure:

1. Click on "Unit Design" button
2. Verify that unit design window opens

6.4.2. Designing a custom unit

Description: Describes the procedure for designing a new unit.
Reference in RD: 6.1.6.2 Designing units, 6.1.6.3 Design budget
Initial system state: The system is running a game, and the player is currently viewing the design menu.
Expected outcome: A land unit design is specified and saved.
Procedure:

1. Click on "Buggy" chassis in the Chassis window, review it's specifications and cost in the information window
2. Click on "Combustion engine mk.I" in the Engine window, review it's specifications and cost in the information window
3. Click on "Light machinegun mk.I" in the Weapon window , review it's specifications and cost in the information window
4. Click on "No extra armor" in the Armor window, review it's specifications and cost in the information window
5. Review the total unit information displayed in the Unit Slot Window window
6. The parts' cost is summarized in the Unit Cost field.
7. Enter the name "Light Buggy" for the design in the text input field.
8. Click "Save" button.
9. Click "Back" button to return to game.

10. Verify design by building it in the in-game build pane

6.4.3.Factional differences

Description: Verifying that there are differences between a sample of factions in the game.
Reference in RD: 6.1.1.1 Starting a premade map, 6.1.1.5 Pause, 6.1.2.1.Building construction, 6.1.2.2.Unit construction, 6.1.2.4.Unit types, 6.1.5.1 Faction selection, 6.1.5.2 Faction differences, 6.1.11.1 Quit the game
Initial system state: The system is running and is currently in the main menu state.
Expected outcome: Visible confirmation that the Atreides faction has access to research alternatives that the Harkonnen faction does not.
Procedure:

1. Click on the “Single Player” button
2. Chose the Atreides faction
3. Enter “Paul” as the screen name
4. Click the “Launch Game” button
5. When the game has loaded, click on the “Barracks” icon on the right side-bar
6. Click on a free 2x2 rock square on the map near the base and wait for the construction to complete
7. Verify that construction of Light Infantry is available
8. Press ESC to access the pause menu
9. Click “Quit” button
10. Start the game
11. When the game has loaded, click on “Singleplayer” button
12. Chose the Harkonnen faction
13. Enter “Feid” as screen name
14. Click “Launch Game” button
15. When the game has loaded, click on the “Barracks” icon on the right side-bar
16. Click on a free 2x2 rock square on the map near the base and wait for the construction to complete
17. Verify that construction of Light Infantry is not available, but that Heavy Infantry is.

6.4.4. Multiplayer designs

Description: During multiplayer custom unit designs are correctly shared.
Reference in RD: 6.1.6.4 Multiplayer designs
Initial system state: Two systems are connected with a multiplayer session and the game is in progress. Each player has only one premade custom unit design.
Expected outcome: The custom designs are merged into one database, however only the creator of each can **construct** the custom unit.
Procedure:

1. Both players create one custom unit
2. Both players send their custom unit to the other players base
3. Both players verify they cannot create any other custom unit designs than theirs.

6.5. In-game unit handling tests

6.5.1. Selecting a single unit or building

Description: Highlights and selects a single unit.
Reference in RD: 6.1.7.1 Selecting a single unit or building
Initial system state: The system is running a game
Expected outcome: The selected unit or building is highlighted with a health bar.
Procedure:

1. Click on a unit or building with the left mouse button
2. Verify that a ring has been presented around the unit

6.5.2. Selecting a group of units

Description: Highlights and selects several units.
Reference in RD: 6.1.7.2 Selecting a group of units
Initial system state: The system is running a game
Expected outcome: The selected units are visually highlighted with health bars.
Procedure:

1. Hold down left mouse button next to a group of units
2. Drag the mouse so that the rectangle drawn by the game encompasses the units
3. Let go of left mouse button.
4. Verify that rings have been presented around the units

6.5.3. Controlling a unit with mouse in combat

Description: Having a unit selected, when clicking upon an enemy unit an attack command shall be issued.
Reference in RD: 6.1.7.3 Controlling units with mouse
Initial system state: The system is running a game and one unit is selected
Expected outcome: The selected unit moves to attack enemy unit.
Procedure:

1. Right click on an enemy unit.
2. Verify that the friendly unit attacks the enemy

6.5.4. Controlling a unit with mouse in non-combat

Description: Having a unit selected, when clicking on an empty non blocked terrain tile the move command shall be issued
Reference in RD: 6.1.7.3 Controlling units with mouse
Initial system state: The system is running a game and one unit is selected
Expected outcome: The selected unit moves to the assigned coordinates.
Procedure:

1. Right click on an empty non blocked terrain tile.
2. Verify that unit moves

6.5.5. Using keyboard to issue defend command

Description: Having a unit selected, when pressing d the defend command shall be issued.
Reference in RD: 6.1.7.4 Controlling units by keyboard
Initial system state: The system is running a game and one unit is selected
Expected outcome: Unit fires upon any enemy unit that comes into range but does not pursue.
Procedure:

1. Press d button
2. Verify that unit stands still when enemy comes within range

6.5.6. Using keyboard to issue attack command

Description: Having a unit selected, when pressing “a” on the keyboard, the attack command shall be issued.
Reference in RD: 6.1.7.4 Controlling units by keyboard
Initial system state: The system is running a game and one unit is selected
Expected outcome: Unit attacks unit even though it is friendly
Procedure:

1. Press “a” on keyboard.
2. Left click upon a unit (not self)
3. Verify that friendly unit attacks target

6.5.7. Using keyboard to issue move command

Description: Having a unit selected, when pressing “m” on the keyboard, the move command shall be issued.
Reference in RD: 6.1.7.4 Controlling units by keyboard
Initial system state: The system is running a game and one unit is selected
Expected outcome: Unit moves to a given location
Procedure:

1. Press “m” on keyboard
2. Left click on ground
3. Verify that unit moves

6.5.8. Using keyboard to issue stop command

Description: Having a unit selected, when pressing “s” on the keyboard, the stop command shall be issued, stopping all move and attack orders.
Reference in RD: 6.1.7.4 Controlling units by keyboard
Initial system state: The system is running a game and one unit is selected
Expected outcome: Unit stops all actions
Procedure:

1. Move unit
2. Press “s” on keyboard when unit is moving
3. Verify that unit stops

6.5.9. Controlling units in combat

Description: When an enemy unit comes within sight range of a unit, this unit will automatically attack. Player can move the unit away from enemy unit to interrupt the fight.
Reference in RD: 6.1.8.1 Controlling units in combat
Initial system state: The system is running a game.
Expected outcome: When an enemy unit approaches the friendly unit shall attack the enemy. When player gives friendly unit a move command it shall cease firing.
Procedure:

1. Place friendly unit close to enemy.
2. Wait for enemy unit to come into range
3. When fight starts move friendly unit away
4. Verify that unit moves away and stops firing

6.5.10. Building and using defensive buildings

Description: Use the build command to create a defensive building on a valid map square. When an enemy unit comes in range of the defensive building it shall attack the enemy.
Reference in RD: 6.1.8.2 Defensive buildings entering combat, 6.1.2.1Building construction
Initial system state: The system is running a game and enough currency is available to create a defensive building and enough space is available on the map.
Expected outcome: When enemy is in range the building fires upon it.
Procedure:

1. Choose a defensive building from the build menu to build
2. Place it on a valid square on the map
3. Wait for an enemy unit to come into range
4. Verify that building fires upon enemy

6.5.11. Firing upon indestructible computer controlled neutral units

Description: Sand Tornados are computer controlled units that are indestructible. These move around the map in a random pattern causing havoc for any players.
Reference in RD: 6.1.8.4 Indestructible computer controlled neutral units
Initial system state: The system is running a game and one unit is selected
Expected outcome: Unit fires upon the tornado but does not do any damage
Procedure:

1. Select unit
2. Issue fire command upon Sand Tornado
3. Verify that Sand Tornado is not damaged by selecting it and viewing its health bar

6.5.12. Gathering Resources

Description: Specific locations on the map will contain resources that a special unit can gather. When these resources have been collected and returned to the base the resources will be converted to currency. This currency can then be used to perform research and buy buildings and units.
Reference in RD: 6.1.3.1 Currency, 6.1.3.2 Harvestable resources, 6.1.7.3 Controlling units with mouse
Initial system state: System is running a game.
Expected outcome: Resources are collected and converted into currency.
Procedure:

1. Select a resource harvester unit.
2. Right click upon a resource filled tile on the map
3. Wait for the unit to complete the gathering process and return to base
4. Observe increase in the Resource Tracker in top right corner of game screen.

6.5.13. Computer controlled opponent

Description: During single player games a computer controlled player should be available
Reference in RD: 6.1.8.3 Computer controlled opponent
Initial system state: User is in the Singleplayer game menu and custom map called “Test AI” is selected
Expected outcome: The game starts with a computer controlled opponent
Procedure:

1. Select “Test AI” map
2. Select the faction “Atreides”
3. Press the “Launch game” button
4. Verify that buildings controlled by the other player are producing

6.6. End of game tests

6.6.1. Victory by mass conquer

Description: When only one player owns units and buildings he/she is declared winner of the game.
Reference in RD: 6.1.11.3 Victorious game by mass conquer, 6.1.8.3 Computer controlled opponent
Initial system state: System is running a game on a special map with 2 units called “Test AI”
Expected outcome: Victory message is displayed and game is ended.
Procedure:

1. Select friendly unit
2. Right click on enemy unit
3. Wait for enemy unit to be destroyed
4. Wait for victory message to be displayed.

6.6.2. Victorious game by disconnection

Description: Upon disconnection of the last client in a multiplayer game the host wins by default
Reference in RD: 6.1.11.2 Victorious game by disconnection
Initial system state: Two systems are connected with a multiplayer session and the game is in progress.
Expected outcome: The game ends and the host is declared victorious
Procedure:

1. Client player disconnect the network cable
2. Wait for 30 seconds in order for connection to time out
3. Verify that proper end of game message is displayed

6.6.3. Lost game by disconnection

Description: Upon disconnection from the host in a multiplayer game the client loses by default
Reference in RD: 6.1.11.4 Lost game by disconnection
Initial system state: Two systems are connected with a multiplayer session and the game is in progress.
Expected outcome: The client loses the game. Game ends for client.
Procedure:

1. Client player disconnect the network cable
2. Wait for 30 seconds in order for connection to time out
3. Verify that proper end of game message is displayed