Requirements and Analysis Document (RAD)

Tansk

Version: 1.0
Date: 2013-05-26
Author: Erik Lundholm, Victor Sandell, Christoffer Henriksson & Tobias Olausson.

Table of Contents

1 Introduction

This section gives a brief overview of the project.

1.1 Purpose of application

The purpose of the application is simply to entertain.

1.2 General characteristics of application

This application will be a standalone desktop, network-based multiplayer game made in Java. It will be a tank shootout game which runs in real-time. The game will be “arena”-based, meaning it takes place in a relatively small closed area which will be entirely visible on the screen at all times. One of the game-modes will be a King of the hill mode where the goal is to hold a certain point for a given time. There will be different setting to make, including limits to limit, score and number of lives.

1.3 Scope of application

This application will not include computer controlled players so it is not possible to play the game actual alone. A playground mode, however, will be available for learning the basics of the game.

1.4 Objectives and success criteria of the project

1. The application should be runnable on Windows & Mac as a multiplayer-only game.
2. The game should feature a deathmatch with four players.

1.5 Definitions, acronyms and abbreviations

- GUI - graphical user interface
- Slick2D - the Java graphics library that is used.
- Kryonet - the Java network library that is used.

2 Requirements

2.1 Functional requirements

- Join/create a “server” (localhost).
- Select “game-mode” (rounds, time, kills)
- Drive around with the tank.
- Rotate turret.
- Fire turret.
- Hit/damage other tanks.
- Pick up powerups.
- Pick up weapons.
- Exit application.

2.2 Non-functional requirements

2.2.1 Usability

(Usability is not the highest priority but the aim is to keep it simple.)
The game should be easy to start and play as soon as possible. There will be no complex controls or mechanics in the game, everything is very straightforward.

2.2.2 Reliability

The game does not hold any vital information and as such there are no fatal effects of crashes, they should however be avoided to give a nice player experience. To do this the game should, among other things, be able to handle errors caused by subsystems.

2.2.3 Performance

The game will react swiftly to player input and interaction and should be fluid at all times.

2.2.4 Supportability

The game will be focused on the Windows and Mac platforms but also run on Linux.

A client-server implementation must be possible. The application will most likely see a lot of rework to its structure once the client-server part will be added.

2.2.5 Implementation

The application will be using the Java environment, and this allows for platform independence. The game simply needs to be downloaded with no installation required.

2.2.6 Packaging and installation

The application will be delivered in a zip-archive with the following:

1. A Java jar-file with the application code.
2. Resources (images, sounds, etc)
3. Readme-file.

2.2.7 Legal

The application contains open-source, free-to-use code, graphical objects and sound.

2.3 Application models

2.3.1 Use case model

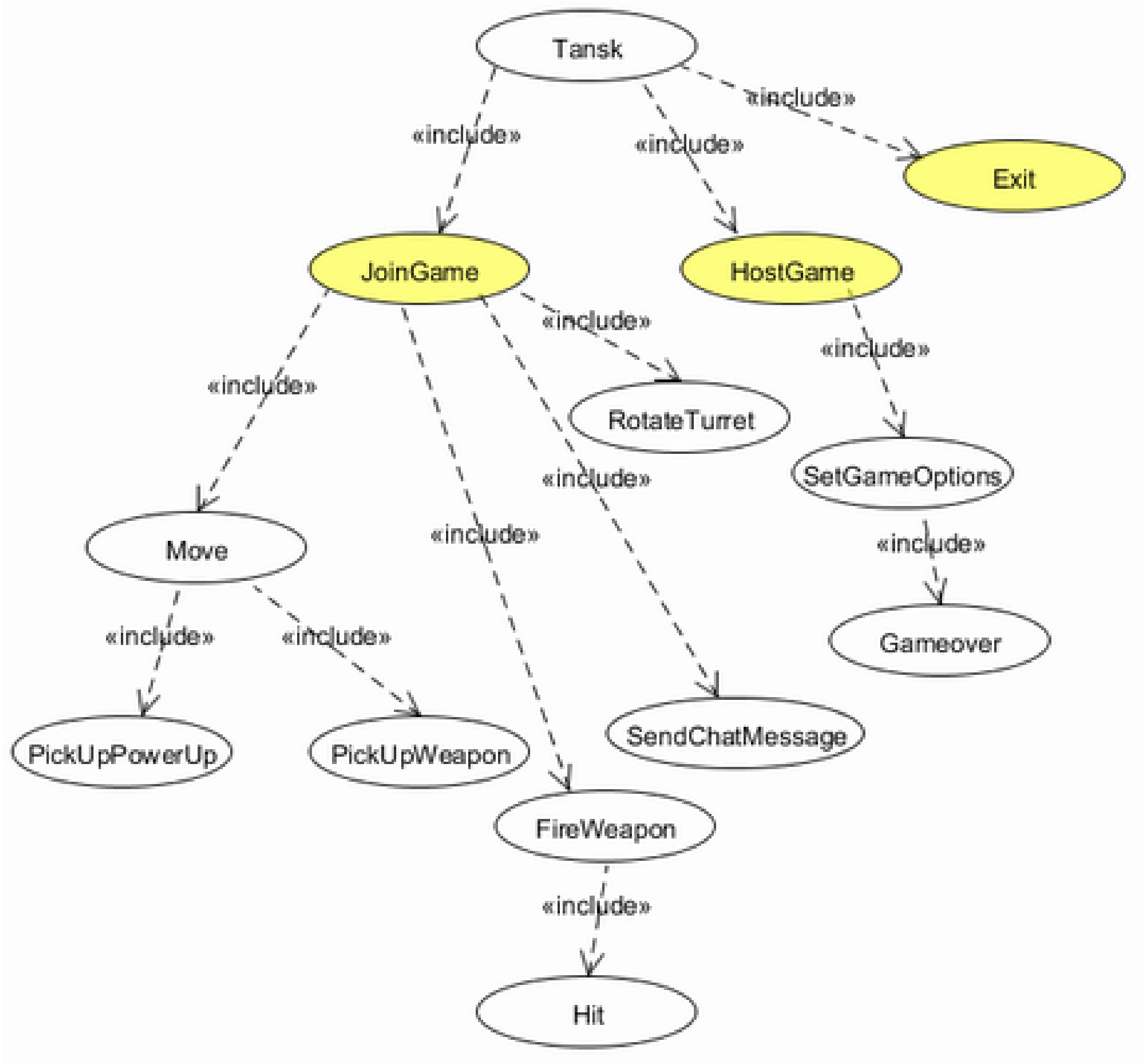


Figure 1: Use case UML

2.3.2 Use cases priority

1. Move
2. Turn
3. Rotate Turret
4. Shoot/fire
5. Hit
6. Spawn/respawn
7. Pick up a power-up

8. Gameover
9. Creating a game server
10. Set game options
11. Joining a game server
12. Pick up a weapon
13. Send a chat message

2.3.3 Domain model

See appendix.

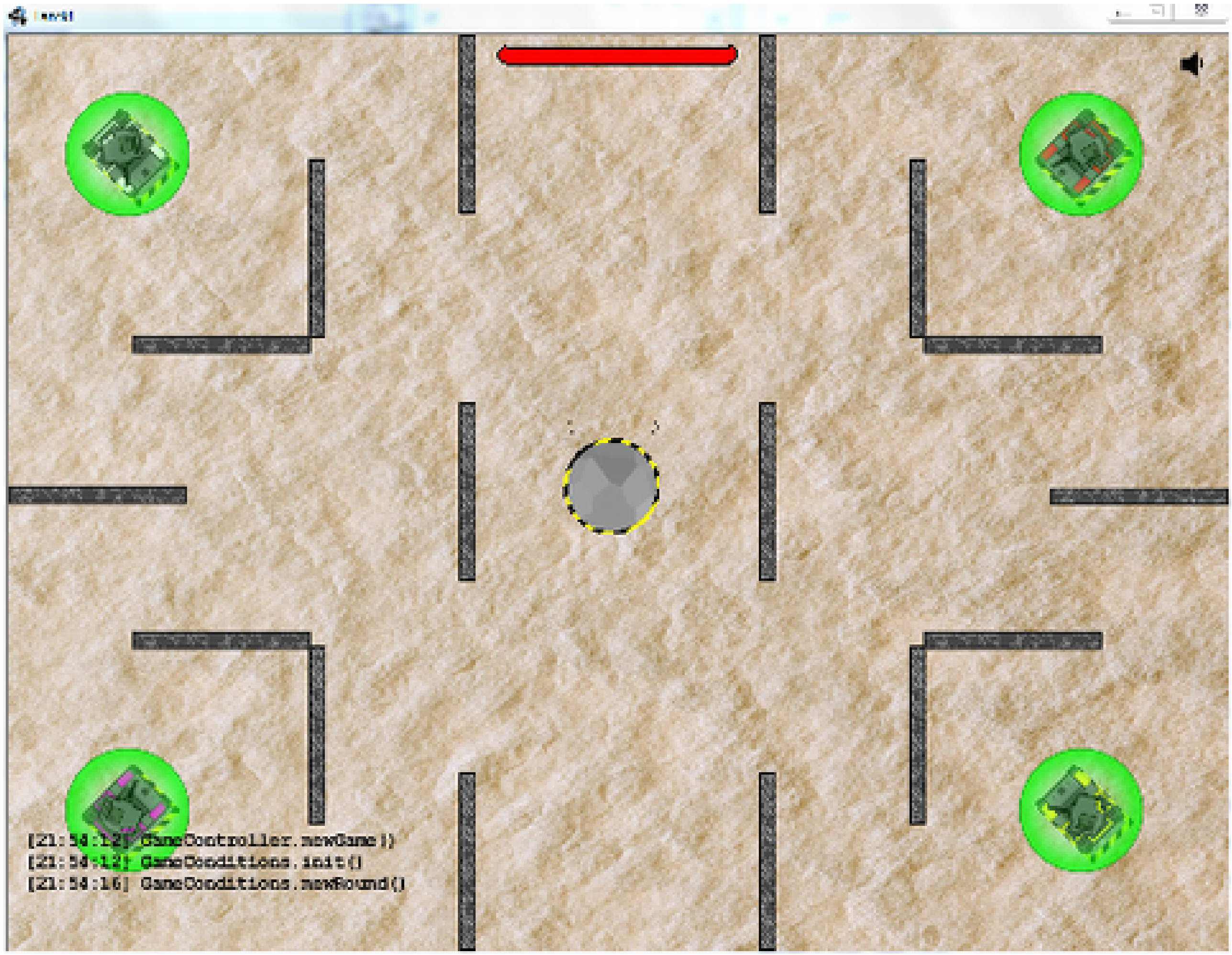
2.3.4 User interface

The user interface will have a fixed size of 1024x768 pixels. The window will not be resizeable and will not take smaller screens into account.

2.4 References

APPENDIX

GUI



```
[21:54:12] GameController.newGame()
[21:54:12] GameConditions.init()
[21:54:16] GameConditions.newRound()
```

Figure 2: Ingame view

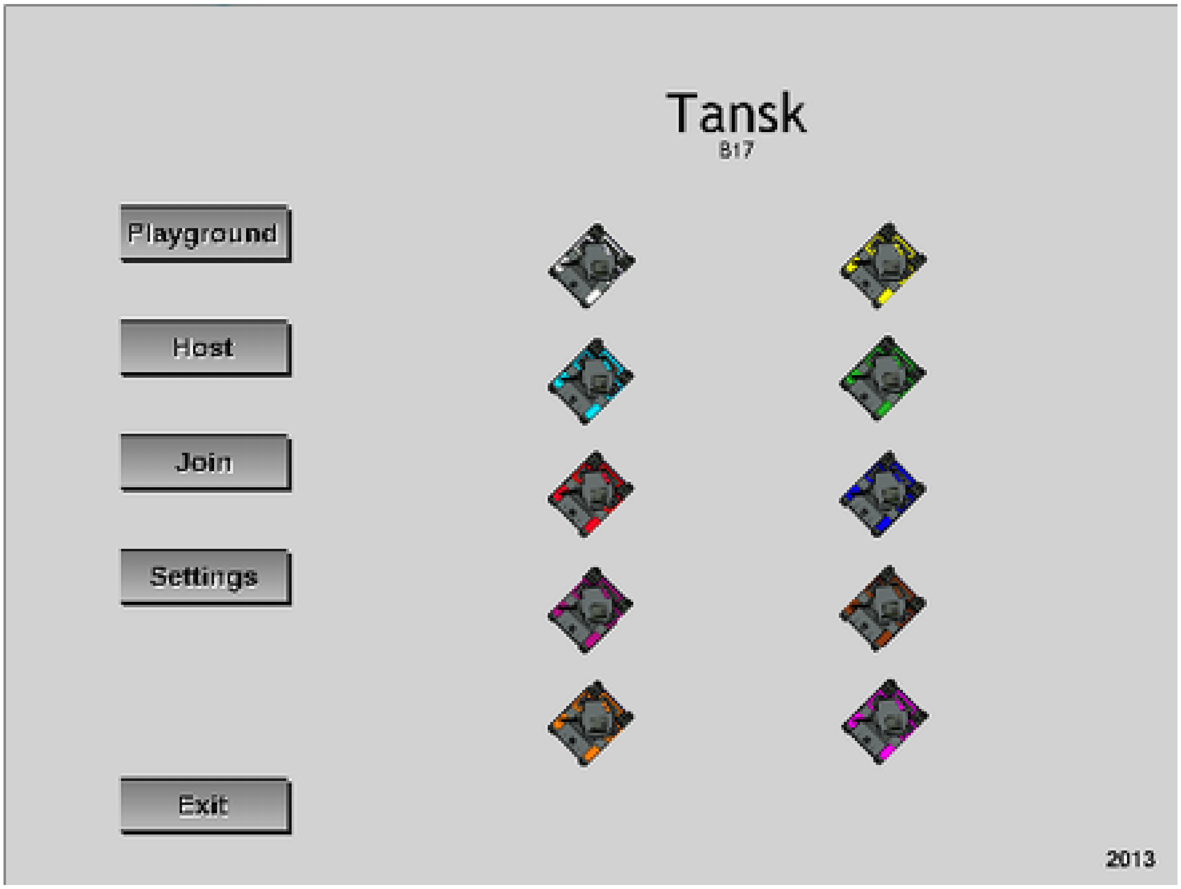


Figure 3: The main-menu

Domain model

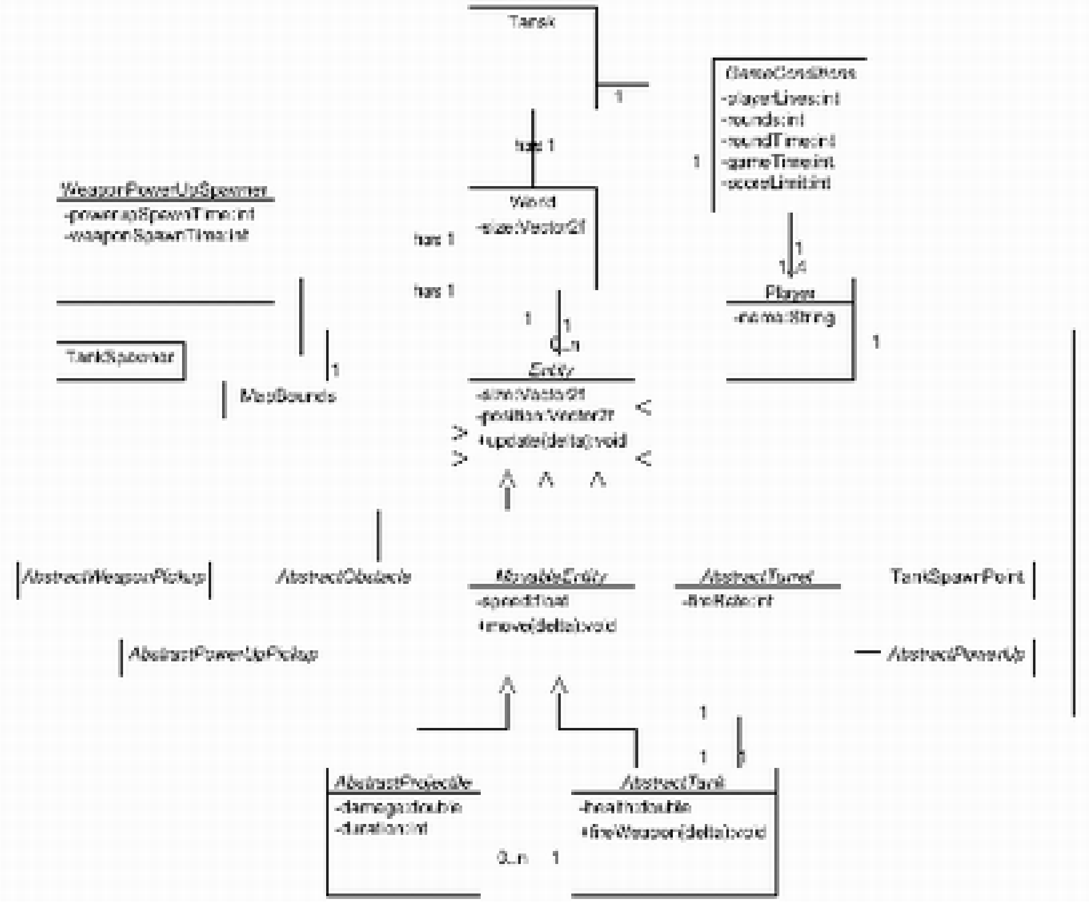


Figure 4: The analysis model

Figure 5: Sequence diagram for usecase FireWeapon

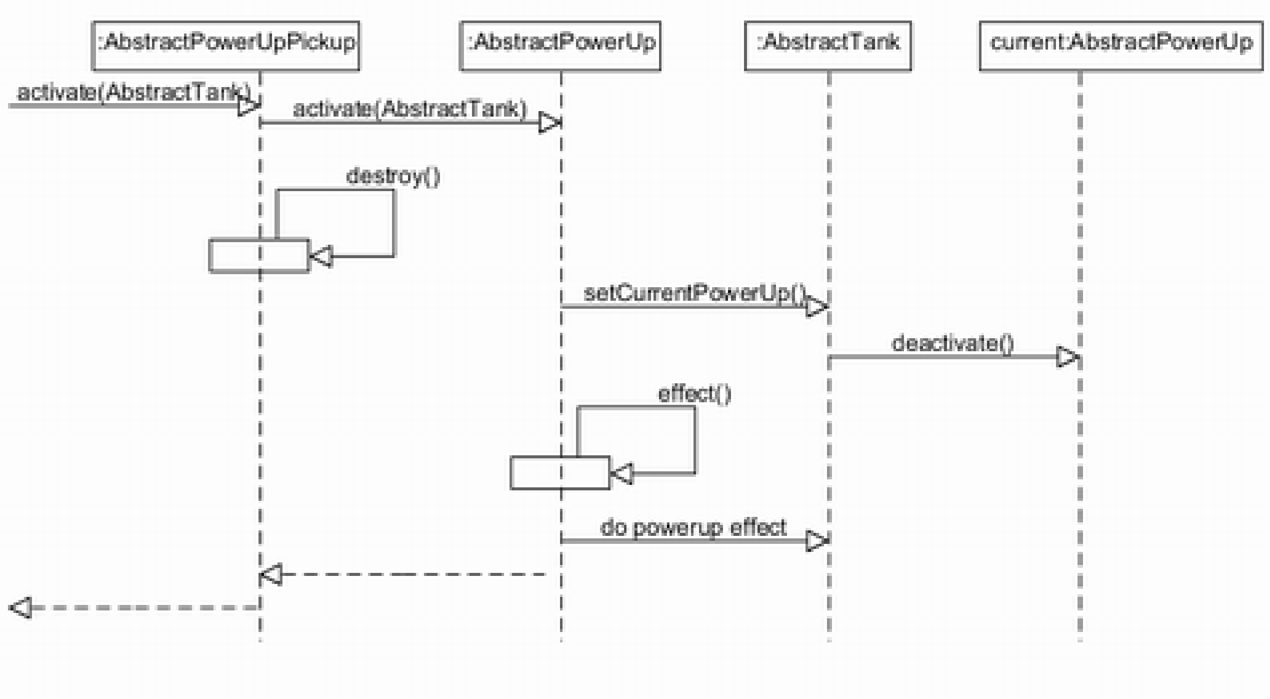


Figure 6: Sequence diagram for usecase Picking up a power-up

Use case texts

Use Case: Move

Summary: This is how the player moves his/her tank.
Priority: *high*
Extends: -
Includes: *Join a game server*
Participators: *Player*

Normal flow of events

Flow 1.1 Player moves

	Action	**System**
1	Player presses movement key (W/S)	
2		Accelerate the tank in the corresponding direction
3	Player releases the movement key	
4		The tank stops accelerating

Alternate flow(s)

Flow 2.1 If blocked by obstacle

	Action	**System**
1	Player presses movement key (W/S)	
2		Stop the tank

Flow 3.1 Multiple buttons

	Action	**System**

1	Player presses S and W at the same time	
2		Continue as if no buttons were pressed

Use Case: Turn

Summary: *This is how the player turns the tank.*
Priority: *high*
Extends: -
Includes: *Join a game server*
Participators: *Player (maybe enemy Player)*

Normal flow of events

Flow 1.1 If unobstructed

	Action	**System**
1	Player presses turn key (A/D)	
2		Turn the tank in the corresponding direction.

Alternate flow(s)

Flow 2.1 If blocked (by obstacle or another tank)

	Action	**System**
1	Player presses turn key (A/D)	
2		Do nothing...

Flow 3.1 Multiple buttons

	Action	**System**
1	Player presses A and D at the same time	
2		Do nothing...

Use Case: Rotate turret

Summary: *A player turns its tank's turret.*
Priority: *high*
Extends: -
Includes: *Join a game server*
Participators: *Player*

Normal flow of events

Flow 1.1

	Action	**System**
1	Player moves his/her mouse.	
2		The tank's turret it rotated, facing the position of the Player's mouse.

Use Case: Fire weapon

Summary: *Player fires the turret on his/her tank.*
Priority: *medium*
Extends: -
Includes: -
Participators: *Player, tank*

Normal flow of events

Flow 1.1

	Action	**System**
1	Player presses the fire key (left mouse button)	
2		The tank’s turret fires projectiles continuously.
3	Player releases the fire key	
4		The tank’s turret ceases firing.

Alternate flow(s)

Flow 2.1 If the tank has fired a shot recently

	Action	**System**
1		Do nothing...

Flow 2.2 If the current weapon have zero ammo left.

	Action	**System**
1		The tank’s weapon is changed to the default weapon.

Use Case: Picking up a power up

Summary: A tank touches and picks up a power up.
Priority: *medium*
Extends: -
Includes: Move
Participators: *Player (and its tank)*

Normal flow of events

Flow 1.1 Picking up a power up.

	Action	**System**
1	A tank comes into range of picking up a power up.	
2		The power up is removed and the power up's effect is applied to the tank.
3		The power up icon out on the map is removed and it can no longer be picked up.

Alternate flow(s)

Flow 2.1 If the player already has a power up active.

	Action	**System**
1		The current power up is deactivated and the new one is activated.

Use Case: Spawn/respawn

Summary: Re-spawning a player’s tank if it dies or spawns a tank if it’s the beginning of a new round.
Priority: *medium*
Extends: -
Includes: -
Participators: *Player (and its tank)*

Normal flow of events

Flow 1.1 The tank dies or a new round begins.

	Action	**System**
1		First a check is made to see if the owner of the tank (the player) has any lives left.
2		If the player infact have a positive amount of lives left a timer starts. At the end of the timer, a new tank is spawned at a random spawn point for the player.

Alternate flow(s)

Flow 2.1 If the player has zero lives left.

	Action	**System**
1		A tank will not be spawned.

Use Case: Picking up a weapon

Summary: A tank comes within range of a weapon and picks up a power up.
Priority: *low*
Extends: -
Includes: Move
Participators: *Tank*

Normal flow of events

Flow 1.1

	Action	**System**
1	A player moves his tank in range of a weapon pickup.	
2		The tank's weapon is changed to the one that has been picked up.
3		The weapon power up on the map is removed and can no longer be picked up.

Alternate flow(s)

Flow 2.1 The tank already has an alternate weapon.

	Action	**System**
1		The current weapon is replaced with the one that has been picked up.

Use Case: Hit

Summary: A projectile hit something.
Priority: *medium*
Extends: -
Includes: Shoot/fire
Participators: *Projectile*

Normal flow of events

Flow 1.1 A projectiles hit something.

	Action	**System**
1		The projectile is always destroyed.

Alternate flow(s)

Flow 2.1 A projectile hits a tank.

	Action	**System**
1		If the tank is a hostile tank, the projectiles damages the tank.

Flow 2.2.1 The tank has a shield.

	Action	**System**
1		The tank's shield takes damage. If the shield is damaged enough to get completely destroyed and any "extra" damage will be given to the tank directly.

Flow 2.2.2 If the tanks HP is now zero or less.

	Action	**System**
1		The tank is destroyed and the owner of the tank (the player) enters the spawn/respawn process (See Use Case: "Spawn/respawn").

Flow 3.1 The projectile hits an obstacle.

	Action	System
1		Nothing happens.

Use Case: Gameover

Summary: A player wins the game, either by achieving the score limit or the time runs out.
Priority: *low*
Extends: -
Includes: Set game options
Participators: *All of the players.*

Normal flow of events

Flow 1.1

	Action	**System**
1		A player is selected as the winner according to game rules.
2		The name of the players are printed along with their score onto the screen.
3		Two buttons appear on the scoreboard, ‘Menu’ and ‘Restart’.

Alternate flow(s)

Flow 2.1 The menu-button

	Action	**System**
1	The player presses the menu-button.	
2		The state is changed to menu and the main menu is displayed.

Flow 2.2 The restart-button

	Action	**System**
1	The player presses the restart-button.	
2		The game restarts.

Use Case: Creating a game server

Summary: A player create a game that others can join.
Priority: *medium*
Extends: -
Includes: Set game options
Participators: *One of the players.*

Normal flow of events

Flow 1.1

	Action	**System**
1	A player selects the “Create game” menu.	The game changes to the Create game-state.
2	The player sets the gameoptions. (See usecase: ‘Set gameopions’)	The game prepares to launch the game with the specified settings.
3	The player waits for other players to join.	
4	The player presses the Create-button.	The game starts the game server.

Use Case: Joining a game server

Summary: A player can join a game that another has created.
Priority: *medium*
Extends: -
Includes: -
Participators: *One of the players.*

Normal flow of events

Flow 1.1

	Action	**System**
1	A player selects the “Join game” menu.	The game changes to the Join game-state.
2	The player enters the IP-address of the server they want to join.	The game displays the IP.
3	The player presses the Join-button.	The game attempts to connect.

Alternate flow(s)

Flow 2.2 Connection success.

	Action	**System**
4	-	The game switches to the ClientLobby-state.
5	The player waits for the game host to start.	

Flow 2.2 Connection failed.

	Action	**System**
4	-	The game outputs an error message.

Use Case: Send chat message

Summary: A player can send a short message to other players.
Priority: *low*
Extends: -
Includes: -
Participators: *One of the players.*

Normal flow of events

Flow 1.1

	Action	**System**
1	A player presses the Enter-button or clicks on the chat field.	The chat window pops up and the chat field begins accepting input.
2	The player enters a string into chat field.	The chat field displays the text.
3	The player presses Enter.	The chat message is sent to the server.
		The server receives the message and relay it to the other clients.

Alternate flow(s)

Flow 2.2 Connection success.

	Action	**System**
4	-	The game switches to the ClientLobby-state.
5	The player waits for the game host to start.	

Flow 2.2 Connection failed.

	Action	**System**
4	-	The game outputs a error message.

Use Case: Set game options

Summary: A player enters the host menu to set the game options
Priority: *medium*
Extends: -
Includes: -
Participators: *One of the players.*

Normal flow of events

Flow 1.1

	Action	**System**
1	The player selects the “Settings” menu	The game changes to the settings state.
2	The player configure the game mode settings.	