Requirements and Analysis Document for Achtung die Klonen (group 7)

Contents

Version

Date

Authors

Jakob Csörgei Gustavsson, Niklas Helmertz, Lucas Persson, Joel Torstensson

This version overrides all previous versions.

1 Introduction

This section gives a brief overview of the project.

Achtung, die Klonen is a game where multiple users compete by controlling their own snake in such a way that it doesn't collide into the wall or a part of another snake's (including itself's) body.

1.1 Purpose of application

Why are we doing this? What are we trying to achieve? Who will use it?

To create a clone of *Achtung, die Kurve* for the desktop. With better performance and fun gameplay than any existing product. It's intended to be used by a group of people gathered around a single computer for pure entertainment purposes.

1.2 General characteristics of application

What kind of application is this? In what environment will it be used?

A stand-alone desktop application with local multiplayer functionality and a graphical user interface made to work on Windows, Mac OS X and Linux.

The game has rounds wherein the game progresses at a constant rate in real-time. The user can control its snake by using keys defined by the user at the start of the game. Once there is only one user left in a round, that user has won that round. Points are awarded linearly from 0 to the one who first died to the amount of players, minus one, to the winner. The rounds continue until one of the players has 10 * (amount of players -1) points.

The style is fairly minimalistic, with pronounced colors and a lack of textures. (See GUI in APPENDIX.)

1.3 Scope of application

What should be in and what should not?

The
Local multiplayer gameplay.
Good and bad power ups. (Positive and negative?)
User defined controls.
User activated power-ups.
(team play?)

Users will not be able to specify power up before game.

1.4 Objectives and success criteria of the project

When are we finished?

Our goal is to create a fun, stable game which supports local multiplayer to some extent. We feel that power-ups are a must to provide variety to the game, which makes it an essential part of the finished product.

Other features that we wish to add, but are not quintessential to the core gameplay, is online play, statistics, persistent user accounts and teamplay. After the product has been “finished” according to our criteria, we will attempt to implement as many of the above features as possible.

1.5 Definitions, acronyms and abbreviations

Achtung, die Kurve - an online multiplayer game developed in 1995 wherein multiple users using the same computer competes by controlling its snake so that it doesn’t collide into something. At start, the snakes are only a point, but as the head moves a trail, referred to as the body, is left behind in every point that the snake visits.

Power-up - an object that can be picked up by a player and causes an effect on something in the game. For example a speed boost which makes the user move at a faster pace than normal.

Lightweight Java Game Library (LWJGL) - a game library for java that amongst other things gives developers the ability to utilize the GPU using OpenGL.

Open graphics Library (OpenGL) - a very common Graphics Library that’s basically an API that has been implemented in most modern graphics hardware.

Graphics processing unit (GPU) - a hardware circuit specialized for graphics rendering.

2 Requirements

2.1 Functional requirements

The players should be able to:

1. Select the amount of players wanted, their colors, names and keys used to control them.
2. Start a new game. Start the first round.
3. Move
   a) Turn right
   b) Turn left
4. Use power-ups.
5. End the current round and move to the next.
6. Exit from round. Takes the user to the main menu.
7. Exit the application.

2.2 Non-functional requirements

Possible NA (not applicable).

2.2.1 Usability

Any user should easily be able to learn the rules of the game and how the application works in a short amount of time.

2.2.2 Reliability

NA

2.2.3 Performance

It should be able to run smoothly on basically any somewhat modern computer. Since the graphical component of the game is fairly simple, it shouldn't be very difficult.

2.2.4 Supportability

The application will support Windows, OS X and Linux without any modifications.

The implementation should allow for easily replacing the existing GUI and rendering components so that different GUI and/or rendering libraries can be used. Porting it to for example the Android platform should be possible without much modification to the non-graphics related code.

The implementation of the power-up system should be sufficiently modular so that a new power-up can be implemented with little or no modification to the existing code.

2.2.5 Implementation

Platform independence will be achieved using the Java Runtime Environment (JRE), which will have to be installed on all host computers running the game. The game will also use the LWJGL-library for rendering, which includes native components for Windows, Linux, Mac OS X and Solaris .

2.2.6 Packaging and installation

Downloadable archive containing a .jar file as well as scripts that can be used for running it on various platforms.

2.2.7 Legal

Will not be covered.

2.3 Application models

2.3.1 Use case model

UML and a list of UC names (text for all in appendix)

See APPENDIX.

2.3.2 Use cases priority

A list

1. Move
2. Turn?
3. Collide
4. EndRound
5. UsePowerup

2.3.3 Domain model (/analysis model)

UML, possible some text.

See APPENDIX.

2.3.4 User interface

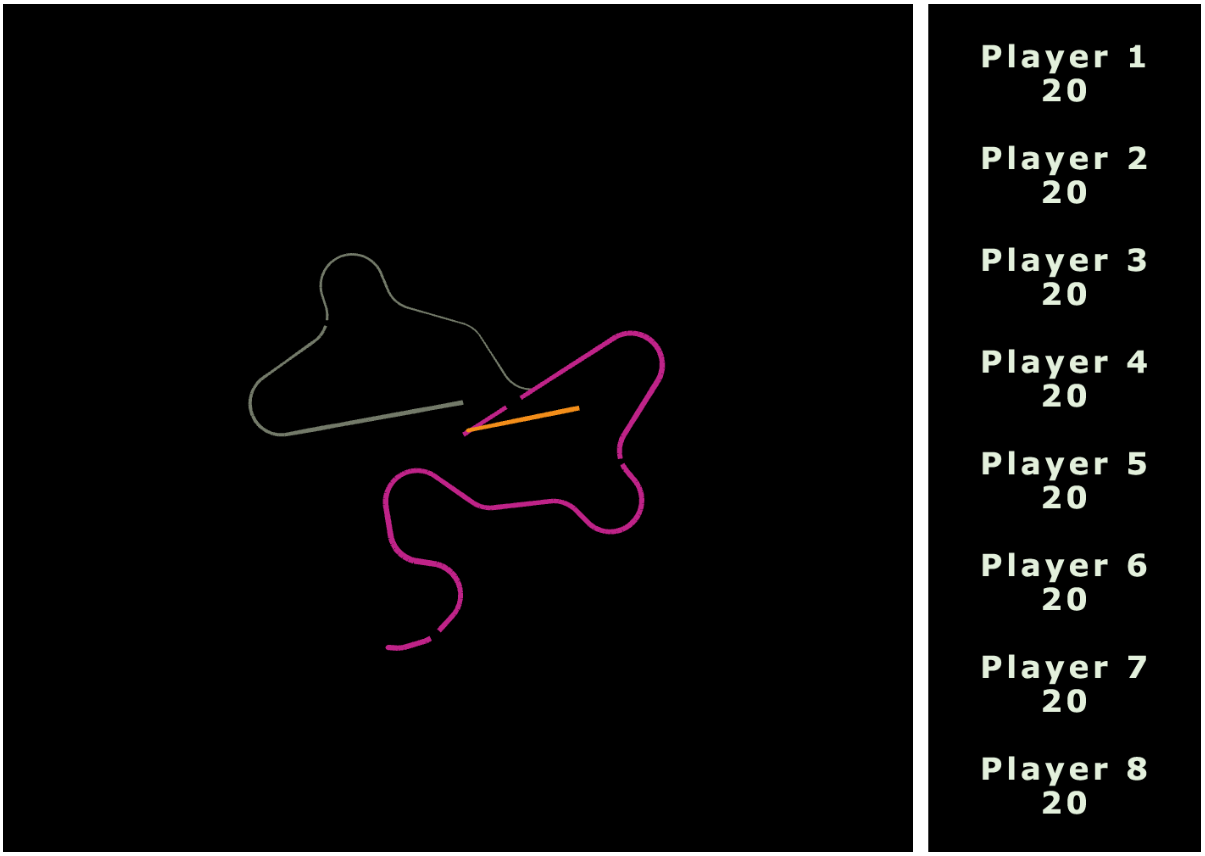
The game consist of a plain black background where the players can move around freely. Somewhere a list of all the players with their respective score can be seen. (See GUI in APPENDIX for an example.)

2.4 References

Popular flash version: http://jesper.nu/spel/achtung-die-kurve
Wikipedia Article: http://en.wikipedia.org/wiki/Achtung,_die_Kurve!

APPENDIX

GUI



Player 1
20

Player 2
20

Player 3
20

Player 4
20

Player 5
20

Player 6
20

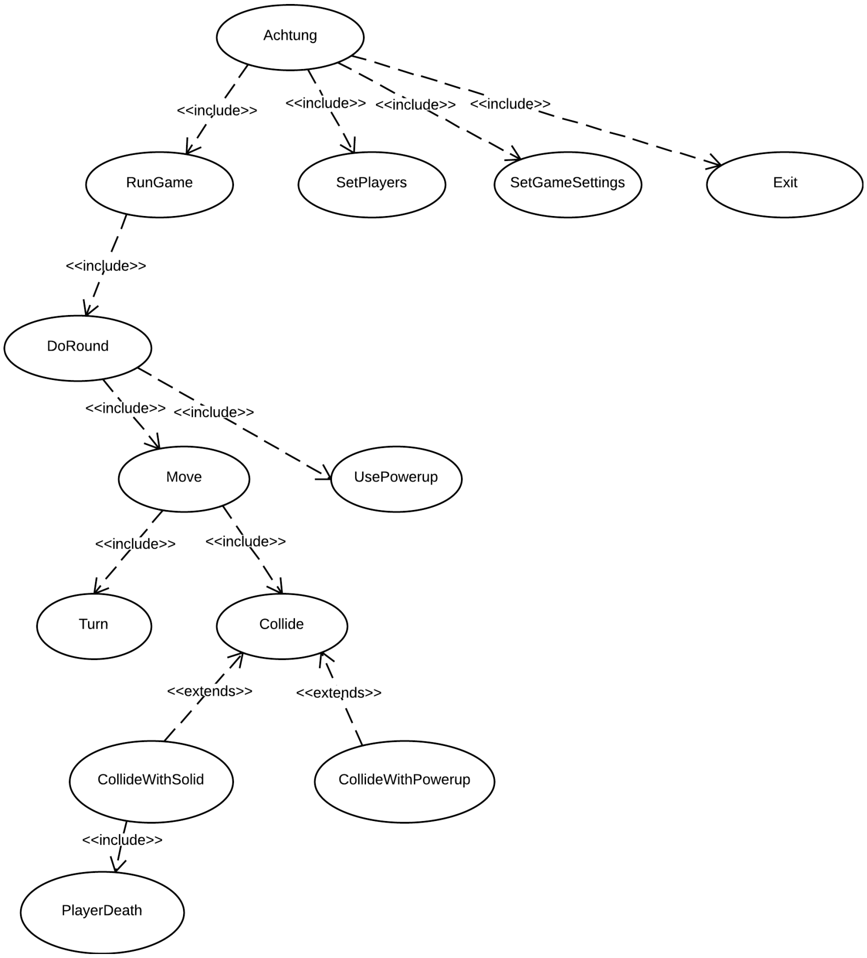
Player 7
20

Player 8
20

Simplest possible GUI that fills the requirements.

Domain model

Use Cases



Use case texts

Use Case: Collide (Abstract)

Summary: A player collides with something

Priority: high

Extends: -

Includes: CollideWithSolid, CollideWithPowerup

Participators: One of the players

Normal flow of events

	Actor	System

1		Responds with the appropriate action depending on what has been collided into (defined in included UCs)

Use Case: CollideWithPowerup

Summary: A player collides with a powerup.

Priority: medium

Extends: -

Includes:

Participators: One of the players

Normal flow of events

	Actor	System
1		Powerup dissapears.
2		The effect of the powerup is applied to the player.

Use Case: CollideWithSolid

Summary: A player collides with something solid (Lika a player’s body or the wall)

Priority: high

Extends: -

Includes: PlayerDeath

Participators: One of the players

Normal flow of events

	Actor	System
1		The player dies

		(PlayerDeath is caused)

Use Case: PlayerDeath

Summary: A player dies.

Priority: high

Extends: -

Includes: EndRound

Participators: One of the players

Normal flow of events

	Actor	System
1		The player's worm is frozen in its current state and the player can no longer control it.
2		One point is awarded to every remaining player.

Alternate flow

Flow 2.1 EndRound (If there is only one player left after death)

	Actor	System
2.1		The round ends (EndRound is caused)

Use Case: EndRound

Summary: The round ends.

Priority: high

Extends: -

Includes:

Participators: All the players.

Normal flow of events

	Actor	System
1		Text is shown displaying who won the round.
2		A keyboard prompt is shown (e.g: “Press space to continue”)
3	Any player presses the requested key	
4		A new round is started

Alternate flow

Flow 2.1 If the win condition set prior to game start is reached (e.g. a user reaches 40 points)

	Actor	System
2.1		Text is shown displaying who won the game.
2.2		Further stats are shown (?, such as gametime etc). Prompt to start a new game with the same settings or return to main menu is displayed.
2.3	User selects an option.	
2.4		The appropriate action is taken.

Use Case: Move

Summary: A player moves.

Priority: high

Extends:

Includes: Collide

Participators: One of the players

Normal flow of events

	Actor	System
1		Extends the player's snake by a given constant in the direction the snake is heading.

Alternate flow

Flow 2.1 Turn

	Actor	System
2.1	Indicates in what direction the snake will turn by holding down the player's left or right key.	
2.2		Extends the player's snake by a given constant in the direction the snake is heading.

Exceptional flow

Collide

	Actor	System
3.1	The snake moves in a direction that makes it collide with another	

	object.	
3.2		The appropriate collision action is taken. (See Collide)