# 'Balls of Steel'

# Group 4

John Laurin

Joakim Åkerlund

Milan Ivanovic

Daniel Öberg

Christoffer Lundell Johansson

# Preface

This document is intended for use in the development of the system described herein. Those that will benefit from using this document are the members of the project-team who will develop the system as well as those who are commissioning the system. This includes developers, project managers and customers.

**Version History:**

| Version | Rationale for new version | Date | Authors |
|---|---|---|---|
| 1.0 | Original Version | 2007-12-20 | John Laurin<br>Joakim Åkerlund<br>Milan Ivanovich<br>Daniel Öberg<br>Christoffer Lundell Johansson |

# Table of Contents:

# Introduction:

## Intended Users and Specification of Purpose

The structure of modern society frequently imposes periods of tediousness on people engaged in occupational, academic and recreational activities. These periods are often distributed into short periods of between 2 and 20 minutes. Examples of these are waiting for a bus/train, breaks in work or academic activities and traveling on public transportation. Often these periods leave the person separated from various means of entertainment, and with little or no means of breaking the tediousness.

The goal of the project is therefore to produce a mobile phone game which is to provide a recreational distraction relieving said tediousness. The game is aimed towards casual gamers and is intended to require only a short time to start up and requiring virtually no configuration, thus enabling the game to be effectively played over intermittent and/or short periods, while still offering an entertainment value.

The intended demographic of users for this project are men between ages 15 and 25, which have access to a cellular or mobile phone supporting the Java Micro Edition Platform for mobile phones (Java ME) and who enjoys playing games on mobile phones.

## System Use Specification

The game shall take the form of an arcade-style scrolling platform game on mobile phones supporting Java Micro Edition. Your avatar in the game takes the shape of a steel ball, which is in a perpetual state of motion. The default state of the ball is that it is in a perpetual bounce with no lateral motion whatsoever. The balls momentum does not decrease as it bounces.

The player can induce lateral motion to move the ball. This motion is imparted on the ball in the form of spin in the appropriate direction. The amount of spin changes the vector of the ball.

The goal of the game is to survive for as long as possible. The game ends when a global countdown reaches zero. Time can gained by collecting power-ups throughout the levels. A level is completed when the player collects a number of keys distributed across the level and reaches the predetermined exit point. On completing a level, an amount of time is added to the global countdown.

In addition to the keys, there are a variety of enemies across the map. These can be both static and dynamic, with the unifying factor that if the player hits one of these, the level restarts and an amount of time from the global countdown is subtracted.

In addition, there are objects placed around the map that alters the position, trajectory and other aspects of the ball, often being combined to create puzzles for the player to solve.

### Usage Narratives:

**1.**

Daniel, 21 years old and Computer Science Student, takes the subway to school. This journey takes him approximately 15 minutes. On this day however, there are major disturbances in the subway-system, resulting in the trains being delayed by 30 minutes or worse.

After 14 minutes of waiting, Daniel finds himself with nothing to do for another 20 minutes. He therefore decides to play some 'Balls of Steel', which he recently obtained for his mobile phone. He starts up the game, arriving at the main menu. Since he hasn't played the game before, there is no option to continue a previous game. As such, he starts a new game.

Daniel performs rather well, and the train arrives before the game ends. He therefor returns to the main menu and quits the game. He is not worried, because he knows he will be able to resume the game at another time.

**2.**

Daniel is now in school. He is now at a lecture which is very tedious. At the break he decides to continue playing 'Balls of Steel'. He starts up the game, arriving at the main menu.

He has his game from earlier that day, so he decides to continue playing that. When he chooses to continue, the game starts in a paused state, allowing Daniel to appraise the situation prior to starting. Daniel then stars he game proper and begins to play.

However, due to the lecturers dry presentation, Daniel's waking state is somewhat precarious. This lowers his performance in the game. Within seconds he hits an enemy, throwing him back to the beginning of the level, and subtracting time from the countdown. This occurs several times, and after a couple of minutes he hits an enemy that results in his countdown reaching zero, whereby the game ends, and his total time is displayed. This time is not high enough to enter the high-score, and he then returns to the main menu. He doesn't feel like playing anymore, so he shuts down the game.

# Contextual and Environmental Specification

The game is meant as a recreational and fun way to spend some extra time and, since the game is to be run on the mobile platform one of the greatest advantages is that it can be accessed anywhere where the user has access to a mobile phone supporting the game, such as a train, a waiting room and similar situations.

The game itself will require that the phone supports Java ME, and meets the following system requirements:

Screen: 65000 Colors

Resolution: at least 128x160

Memory: at least 1 MB free for Java

# The Scope of the System

Since the system we are developing is a game intended for the mobile platform the scope will be fairly limited. The core purpose is to entertain the user of the game, and the game will be quite simple since it is for mobile use and also we don't have that much experience nor time.

| Topic | In | Out |
|---|---|---|
| Single-player | X | |
| Multiple Player Profiles | | X |
| 2D-graphics | X | |
| Sound | X | |
| 3D-graphics | | X |
| Multi Player | | X |
| Save Progress on Exit | X | |
| High score | X | |
| Export High Score to website | | X |
| Different resolutions | X | |

# Design and Building Considerations

'Balls of Steel' is a game developed for the mobile platform using Java ME. It's primary function is to entertain it's users by providing correct output (graphics, sound) on the users phone and to be able to correctly take input that the user provides and implement it as stated in the requirements. The user should be, upon being familiar with the controls, able to play the game directly, without having to refer to any instructions.

- What happens if the game is forcefully exited? When you start the game again you should continue from the last save, but the forceful exit will in all likelihood have bypassed the saving procedure. Therefore the game should not overwrite or remove the save-file when it is loaded.

- Support for different resolutions for different mobiles. Ideally the game should offer an adaptive environment based on the aspect ratio of the current mobile phone.

- If someone calls while you are playing the game it should pause and minimize (if minimize doesn't work it should save the state and then exit).

- Handling of telephone events that are not immediately critical, such as the arrival of an SMS, a battery warning etc. The game should not be exited in such cases, and should simply pause while the user decides what to do about them.

- The game must be efficient as the system resources on mobile phones may be very limited. The resources in question are primarily the computational resources and the storage space. Graphics and game logic is the primary consideration in this regard. The stated systems requirements should be sufficient.

- Handling of phones that do not have standard left, right and accept buttons. Such phones do, however, have designated inputs for those keys, but that will make the game harder if not impossible to play.

# Technologies and Risks

- Programming Language - Java with the Micro Edition API

    - Possible Risks: The Developers are proficient with Java, but not with the ME API. This will result in increased developing time and may result in significant setbacks. Furthermore, Java 2D is somewhat notorious for its inefficiency, and if it is not implemented well enough it could lead to the game being unplayable. The physics component are simply a series of algorithms which may be relatively simple in theory., but implementation on a practical level using as few resources as possible may be an issue.

- Distributed Version Control System - Bazaar

    - Possible Risks: The Developers are not all familiar with the program, which may require some learning time, and it may slow down the implementation somewhat in the beginning.

- Integrated Development Environment - NetBeans

    - Possible Risks: The Developers are not familiar with all the aspects and intricacies of the software, which may make the implementation less efficient than it should be.

- Visual Content Editors - Adobe Photoshop, Autodesk 3D Studio Max 9 (Note: only 2D-content will be used, as stated in section 4)

    - Possible Risks: The visual content produced is too complex for mobile phones to process, resulting in a frame-rate that makes the game unplayable (10 Frames per Second or less).

# Glossary

**Java Micro Edition (ME)** - is a specification of a subset of the Java platform aimed at providing a certified collection of Java APIs for the development of software for small, resource-constrained devices such as cell phones, PDAsset-top boxes. and printers.

**API** - application programming interface is a source code interface providing a library with standard functions.

**Bazaar** - distributed version control system available under the GPL that reduces barriers to participation in your project.

**Version control system** - is the management of multiple revisions of the same unit of information.

**GPL** - The GNU General Public License is a widely used free software license, originally written by Richard Stallman for the GNU project

**NetBeans** - refers to both a platform for the development of Java desktop applications, and an integrated development environment (IDE) developed using the NetBeans Platform.

**IDE** - In computing, an integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development.

**Adobe Photoshop** - a graphics editor developed and published by Adobe Systems.

**3d studio max** - a full-featured 3D graphics application developed by Autodesk Media and Entertainment.

**SMS** - Short message service , a form of text messaging on mobile phones

**MMS** - Multimedia Messaging Service, a standard for a telephony messaging systems

**FPS** - Frames per second, used for measuring the frame speed in a moving image

**A.I.** - artificial intelligence is "the study and design of intelligent agents" where an intelligent agent is a system that perceives its environment and takes actions which maximizes its chances of success

**Repository model** - A central database where the different subsystems can "communicate" with each other by adding information that will be instantly available for all the other systems.

**Centralized control model** - The centralized control model, or call-return model, consists of a main program which calls subroutines which may then call their own subroutines but in the end the desired value is returned back up to the main program.

**Real-time** - The processing of information that returns a result so rapidly that the interaction appears to be instantaneous.

# User requirements definition

## Functional Requirements:

1. The system shall provide a menu-screen on start-up, allowing the user to select to start a new game, to continue one that is in progress, to view the high-score, to make general settings, to view instructions and to exit the program.

   **Test:** Start up the game, and check that the relevant menu items are presented on the screen.

2. The system shall allow users to turn off sound and/or music via the settings-option on the menu-screen.

   **Test:** Turn off game sound, and start playing and confirm the loss of sound. After that restore sound and repeat procedure.

3. The inexperienced user shall be able to gain instructions on gameplay mechanics and goals via the instructions menu.

   **Test:** Test and check if we have access to the instruction screen.

4. The user shall be able to see the 10 highest times, as well as the signature of the user who achieved it via the high-score option on the menu-screen.

   **Test:** Try to add a high score to the list and check if it is represented in the list. Se what happens when we add more scores to fill the table. Test and check if we have access as players to this menu.

5. The user shall be able to return to the main menu at any time during play.

   **Test:** Test from all menus, game settings and levels that we can make it back to the main menu.

6. The user shall be able to control the motion of the ball.

   **Test:** In game, test if ball responds to the given motion controls in the way specified.

7. There shall be a gravitational force that propels the ball, and the default direction is downwards.

   **Test:** Check if the ball bounces opposite the gravity and then returns to the given direction of gravity.

8. The game will provide a number of obstacles that provide special events on collision, including walls, mines, magnets and wormholes.

   **Test:** Test collisions of the ball and all obstacles to observe collision behavior.

9. Mines shall, upon coming in contact with the ball, return the ball to the levels starting point

and subtract a preset amount of time from the global countdown.

**Test:** Collide with a mine, and check if the right amount of time is subtracted, and that we now are at the beginning of the map.

10. Mines can either be static or dynamic. Dynamic mines shall move in a predetermined pattern, static mines shall remain at their original position at all times.

    **Test:** Collide with a mine, and check if the right amount of time is subtracted and that we now are at the beginning of the map.

11. Magnets shall exercise a force upon the ball if it is within a given distance and alter the trajectory of the ball.

    **Test:** Move into the range of the magnet, and check if we are affected as described by the requirements.

12. Wormholes shall, upon contact with the ball, transfer the ball to a corresponding wormhole in another location on the level.

    **Test:** Enter a wormhole, and check that we are transported to the corresponding wormhole as specified.

13. Walls are the standard impediment. Upon contact with the ball, the ball bounces off at a trajectory determined by the angle of the wall and the balls original vector.

    **Test:** Check that walls hinder movement and that the ball bounces off it as specified.

14. The user shall be able to collect keys throughout the levels. Upon collecting all the keys, the exit point for that level shall be unlocked.

    **Test:** Try to collect a key, and check if the exit is opened when all are collected.

15. Upon reaching the exit point for a level, the next level shall begin. Players shall receive additional time for the global countdown, determined by which level they just completed.

    **Test:** Upon connection with the exit check that we are given the extra time specified, and that the next level is loaded as specified.

16. The player shall be able to collect power-ups throughout the level. These shall include time-increases, low gravity, reverse gravity and freeze.

    **Test:** Try to collect the power-up and see if the specified behavior is as expected.

17. Time Increase power-ups shall give the player additional time to the global countdown.

    **Test:** Pick it up (collide with) and check that we are given the right amount of time.

18. Low Gravity power-ups shall lower the gravitational acceleration of the ball, resulting in changes to how the ball behaves when bouncing. This effect ceases after a given time.

**Test:** Pick it up (collide with) and check that the gravity change now and over time reacts in the right way.

19. Reverse Gravity power-ups shall change the direction of the gravitational force, in effect turning the ceiling into the floor and vice versa. This effect ceases after a given time.

    **Test:** Pick it up (collide with) and check if we "fall" in the opposite direction and with the same speed as normal.

20. Freeze power-ups shall stop the motion of all dynamic mines throughout the level. This effect ceases after a given time.

    **Test:** Pick it up (collide with) and check if the mines have stopped in their movement.

21. The game shall provide a number of levels. If the player completes all of these, the player shall return to the first level and shall be subject to increased difficulty. This shall take the shape of increased penalties for hitting mines, less time given upon level completion and less power-ups given across the level. This effect is cumulative until the game ends.

    **Test:** Check that the required changes take effect after the game starts over again.

22. When the global countdown reaches zero, the game shall display a Game Over screen, on which shall be displayed the total time played. It shall also display whether or not the time was sufficiently high to be displayed on the High score.

    **Test:** Let the game time end, and check if the game records and show the items as required.

23. If the player time was sufficient to be displayed on the High Score, the game shall allow the player to input his signature at the specified place.

    **Test:** Enter High Score and check if it is added.

24. If the players time was not sufficient to merit a mention on the High Score, the High Score shall be displayed.

    **Test:** Check if the High Score is presented and that the players score is not present.

25. If the phone receives a call during play, the game shall pause, and the phone shall use its standard procedures for handling incoming calls. Upon terminating the call, the game shall be able to continue.

    **Test:** Send a call to the phone, and check that we can resume the game afterwards.

26. It the phone receives an sms, mms or some other form of message, the game will be paused.

    **Test:** Send an sms and mms to the phone, and check that we can resume the game afterwards.

27. If the phone issues a warning or another kind of system message, the game shall be paused.

**Test:** Force warnings and phone messages to the user, and check if the game is paused and resumable.

# Non-Functional Requirements:

1. The memory usage of the software should not exceed 1MB during run-time.
2. The memory size of the save file should not exceed 50KB.
3. The memory requirements for the level-files should not exceed 100 KB.
4. The response time between choosing a menu option and the screen for that option appearing should not exceed 10 seconds.
5. The response time between user input and game response should not exceed 0.5 seconds.
6. The framerate during gameplay should not drop below 2 frames per second (fps) during gameplay.

# Use Cases:

**Global Extensions:**

*a. The game crashes during play or load times.

1. The player restarts the game. All previous game data is lost and the game begins anew.

*b. The phone receives a call or a message during play.

1. The game minimizes and pauses until the player decides to continue.

## Use Case 1: Turn off sound

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to turn off all sounds.

Minimal guarantee:
The player can check what sound options are on/off.

Precondition:
The game is loaded and the sound is on.

Success guarantee:

All sound in the game is off.

Main Success Scenario:

1. The player enters the settings menu.
2. The player is presented with the sound options which he turns off.
3. The player returns to the main menu and can now start a game without sound.

Extensions:

## Use Case 2: Change the Ball's Movement

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to change the movement of the ball.

Minimal Guarantees:
The player will be able to determine whether the motion of the ball has been altered.

Preconditions:
The player has started a game and is playing it.

Success Guarantee:
The motion of the ball is changed.

Main Success Scenario:

1. The player issues a command for a left or right movement.
2. The spin of the ball in the direction commanded is increased.
3. The system stores the new spin value, using it to calculate the balls motion.
4. The system moves the ball according to the calculations.

Extensions:

## Use Case 3: Collect a Power-up

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to collect a power-up and gain the effect the power-up provides.

Minimal Guarantees:
The player will be able to determine whether or not the power-up has been collected.

Preconditions:
The player has a game in progress and is playing. The player is also in the vicinity of a power-up.

Success Guarantee:

The power-up is collected and the effect is provided to the Player.

Main Success Scenario:
1. The player brings the ball into contact with the power-up.
2. The system removes the power-up from the game-area.
3. The system determines the type of the power-up and applies the effect accordingly.
4. The system records the time of the collection and determines the time at which the effect will be removed.

Extensions:
1.a. The player fails to bring the ball into contact with the power-up.
1. The player changes the movement of the ball, trying to collect the power-up once more.

## Use Case 4: Collect a Key

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to collect a key.

Minimal Guarantees:
The player will be able to determine whether or not the key has been collected.

Preconditions:
The player has a game in progress and is playing. The player is also in the vicinity of a key.

Success Guarantee:
The key is collected and the collected key count is increased with one.

Main Success Scenario:
1. The player brings the ball into contact with the key.
2. The system removes the key from the game-area.
3. The system increases the key count with one.

Extensions
3a. Key count equals the total level key count.
1. Open the exit point.

## Use Case 5: Finish a Level

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to exit the current level and begin the next one.

Minimal Guarantees:
The player is able to determine if he has finished the level.

Preconditions:
The player has gathered all the keys represented on the current level.

Success Guarantee:
The player proceeds to the next level.

Main Success Scenario:
1. The player brings the ball into contact with the exit point.
2. The system loads the next level.
3. The system increases the global time count by the set amount of the finished level.
4. The game starts the the next level.

Extensions:
1a. Key count is less then the level key count.
1. Treat as a wall collision.
3a. There is an error upon loading the next level.
1. Notify the user and try to restore the game session.
2. Terminate the game if restoring doesn't work.

## Use Case 6: Write a signature on the High Score

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to write his signature on the high score list

Minimal Guarantees:
The player is given a change to write a signature on the high score list

Preconditions:
The game is over and the player has collected enough points what will warrant him a high score place.

Success Guarantee:
The player writes his signature on the high score list. The system saves the high score list.

Main Success Scenario:
1. The system prompts the player to write his signature.
2. The player writes his signature by using the input keys predefined by the player's mobile phone manufacturer.
3. The system updates and saves the high score list with the new signature and score.

Extensions:
2.a The player doesn't want to write his signature
1. The player cancels and the high score isn't updated.

## Use Case 7: Start a new Game

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to start a new game.

Minimal Guarantees:
The player get information on what goes wrong on startup.

Preconditions:
The game is loaded.

Success Guarantee:
The game starts and the player can play it.

Main Success Scenario:
1. The system loads the first level.
2. The player gets time equal to the time given for the level.
3. The players key count is set to zero.
4. The player starts playing the new game.

Extensions:

## Use Case 8: Continue a Game in progress

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to continue a game.

Minimal Guarantees:
The player is informed whether he can continue a game or not.

Preconditions:
The player has a game to continue.

Success Guarantee:
The game continues from a saved state.

Main Success Scenario:
1. The player chooses to continue a game.
2. The system loads the save-file.
3. The game starts from the saved position.

Extensions:
2a. The system fails to load the save-file.
1. The system gives an error message and returns the user to the menu.

## Use Case 9: Avoid a Mine

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to avoid colliding with a mine.

Minimal Guarantees:
The player is able to determine if the the ball successfully avoids colliding with the mine or not.

Preconditions:
The player is playing and is in the vicinity of a mine.

Success Guarantee:
The ball avoids colliding with a mine.

Main Success Scenario:
1. The player sees a mine.
2. The player decides to move the ball in a certain direction.
3. The player successfully avoids the mine.

Extensions:
2a. The player fails to avoid the mine.
1. The player moves into contact with the mine.
2. The system subtracts the predetermined amount of time.
3. The ball's position is reset to the start of the level.

## Use Case 10: Enter a Wormhole

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to get the ball into a wormhole.

Minimal Guarantees:
The player is able to determine if the the ball successfully enters the wormhole or not.

Preconditions:
The player is playing and is in the vicinity of a wormhole.

Success Guarantee:
The ball enters a wormhole and exits from the corresponding wormhole.

Main Success Scenario:
1. The player moves the ball into the wormhole.
2. The system determines which is the corresponding wormhole.

3. The ball's position is updated to the new wormhole keeping the current speed and direction.

Extensions:

## Use Case 11: Use a Magnet

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to get the ball affected by a magnet.

Minimal Guarantees:
The player is able to determine if he is affected by a magnet or not.

Preconditions:
The player is playing and is in the vicinity of a magnet.

Success Guarantee:
The ball is affected by a magnet.

Main Success Scenario:
1. The player moves the ball within range of the magnet.
2. The system determines in what way the magnet should affect the ball.
3. The ball is affected by the magnet.

Extensions:

## Use Case 12: Hit a wall

Primary Actor:
The player

Stakeholders and Interests:
The player - Wants to get the ball to bounce off the wall.

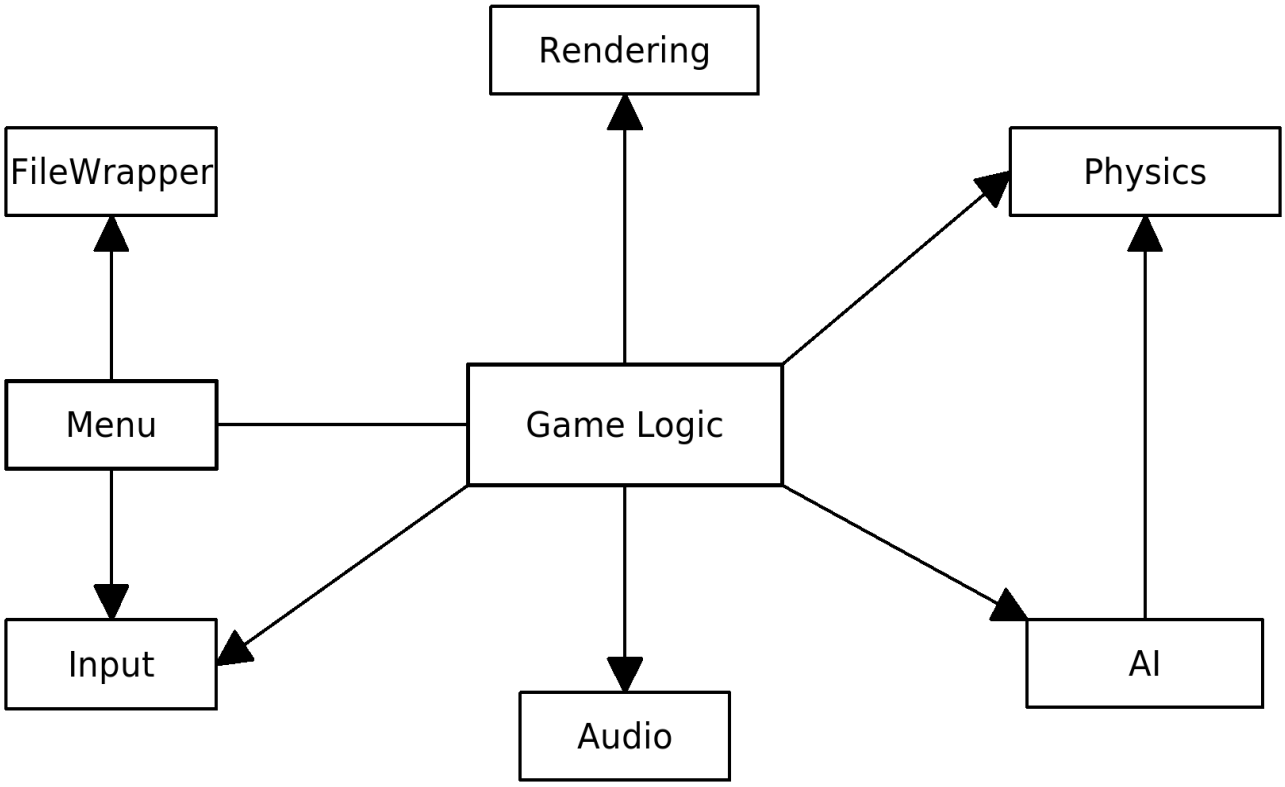
Minimal Guarantees:
The player is able to determine if the ball hits the wall or not.

Preconditions:
The player is playing and is in the vicinity of a wall.

Success Guarantee:
The player manages to hit the wall and bounce in the predetermined direction.

Main Success Scenario:
1. The player brings the ball into contact with the wall.
2. The system determines which direction the ball should bounce.
3. The ball bounces in the determined direction.

# System architecture




The main concern for a real-time game for mobile phones is performance. As such there is a need to localize critical operations within a small number of subsystems with as little communication as possible between these subsystems.

Having that in mind we see that the repository model seems to be the most appropriate. One can argue that the rendering, physics and AI subsystem all need to know the state of the enemies, the position and speed of the ball and so forth.

The centralized control model goes hand in hand with the repository model and our need for performance. Having an interrupt-driven model would make it unnecessary complicated since none of the external API's support it.

## Subsystems

**AI** - Takes care of the enemies AI and their game logic. More specifically in what way the enemies will move and how they will react upon coming into close range of the ball.

**Audio** - This module is called upon when another module want to play a sound clip.

File Wrapper - Handles loading and saving of data into the phones memory card. Loads the data into the phones memory.

**Game Logic** - Handles everything that the other modules don't. That includes the balls motion and calling the other modules for updates.

**Input** - Pulls the state of the input (joystick/keypads) and presents these.

**Menu** - Presents the menu on screen and gives the user the capability to select whether he wants to continue, change settings, see the high score, create a new game or quit.

**Physics** - Controls the players gravity and all collision detection.

**Rendering** - Presents the ball, the enemies, the items and the map onto the screen.

# System requirements specification

## 1. Main Menu

Function: Provide the player with different menu-options.

Description: The menu contains the following options: Start a new game, continue game, view high score, change settings, view instructions, and exit program

Inputs: Button command

Source: Player provides input via key/touch pad

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The menu provides different options that will load if the player chooses one of them.

Precondition: The game is started.

Postcondition: The chosen option is loaded.

## 2. Settings Option

Function: Provide the player with options for changing settings.

Description: Contains option to allow users to turn on/off sound and/or music and returning to main menu.

Inputs: Button command

Source: Player input via key/touch pad

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Provides the player with options to change sound/music settings and to return the game to

the main menu.

Precondition: The settings option is loaded.

Postcondition: New settings are saved and the player is able to return to the main menu.

## 3. Instructions Option

Function: Provide player with instructions for playing the game.

Description: Contains instruction about how to play the game and option to return to the main menu.

Inputs: Button command

Source: Player input via key/touch pad

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Provides the player with instructions about how to play the game, and the ability to return to the main menu.

Precondition: The instructions option is loaded.

Postcondition: Instructions are shown and the player is able to return to the main menu.

## 4. High Score Option

Function: Provide the player with the high score screen.

Description: Contains the 10 highest high scores and their signatures and the option to return to the main menu.

Inputs: Button command

Source: Player input via key/touch pad

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Displays the high score screen and provides the option to return to the main menu.

Precondition: The high score option is loaded.

Postcondition: The high score screen is displayed and the player is able to return to the main menu.

## 5. Return Option

Function: Return the game to the main menu.

Description: Provide the player with the option to return to the main menu while playing a new or continued game.

Inputs: Button command

Source: Player input via key/touch pad

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Returns the game to the main menu and saves the current state, thus allowing the player to continue the game later at will.

Precondition: A game is in progress.

Postcondition: The main menu is displayed and the game state is saved.

## 6. Control Motion

Function: Controls the motion of the ball

Description: The default state of the ball is that it is in a perpetual bounce with no lateral motion whatsoever. The balls momentum does not decrease as it bounces, i.e. an elastic bounce loosing none of its kinetic energy, thus making the ball bounce back towards the same height each time.

Inputs: Button command

Source: Player input via key/touch pad

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The player can induce lateral motion to move the ball. This motion is imparted on the ball in the form of spin in the appropriate direction. The amount of spin changes the vector of the ball in the x-plane, however inducing spin in the air will not change the actual velocity in the x-plane until the ball bounces.

**The system determines the change of position of the ball each "tick" using the following formulas:**

$$\Delta s_x = V_x \cdot t + \frac{a_x \cdot t^2}{2}, \Delta s_y = V_y \cdot t + \frac{a_y \cdot t^2}{2}$$

where t = time in seconds per “tick”, V = the velocity of the ball, a = the acceleration imparted on the ball and s = the change in position.

Precondition: A game is in progress.

Postcondition: The ball is moved in the inputted direction.

## 7. Gravitational Force

Function: Apply gravitational force on the ball.

Description: The gravitational force shall propel the ball in a given direction.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The ball will be propelled in a given direction, which by default is down, by an acceleration g.

Precondition: A game is in progress.

Postcondition: The ball is propelled in the direction of the gravitational force.

## 8. Obstacles

Function: Provide different challenges for the player depending on type.

Description: Different objects including walls, mines, magnets, and wormholes will provide a challenge for the player.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon close proximity and/or contact the different objects will start some special event.

Precondition: A game is in progress.

Postcondition: The special event related to the obstacle is triggered.

## 9. Static Mines

Function: Static mines will hurt the ball reseting its position to the start of the current level and subtracting a set amount of time from the global timer.

Description: Static mines are mines that are non movable objects that will only activate if the ball collides with them.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon contact with the mine, the ball's position is reset to the start of the map and a set amount of time is subtracted from the global timer.

Precondition: A game is in progress and the ball collides with a mine.

Postcondition: The ball's position is reset and the timer reduced according to specification.

## 10. Dynamic Mines

Function: Dynamic mines will hurt the ball reseting its position to the start of the current level and subtracting a set amount of time from the global timer.

Description: Dynamic mines will like static mines only activate if the ball collides with them. The dynamic mines will move in a predetermined pattern, e.g. from point A to point B and back again, and will be totally unaffected by the gravitational force.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon contact with the mine, the ball's position is reset to the start of the map and 20 seconds is subtracted from the global timer.

Precondition: A game is in progress and the ball collides with a mine.

Postcondition: The ball's position is reset and the timer reduced according to specification.

## 11. Magnet

Function: Magnets will attract the ball.

Description: Magnets will act as weak gravitational force acting on the ball attracting it in the direction determined by the position of the ball and the magnet.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon close proximity with a magnet, the ball's trajectory will be affected by another gravitational force generated by that magnet, thus attracting the ball and lessening the normal gravitation's impact on the ball.

**If the ball gets close to a magnet these formulas will apply:**

$$a_x = \frac{k_{magnet}}{-(x_{ball} - x_{magnet})}, a_y = \frac{k_{magnet}}{-(y_{ball} - y_{magnet})} + g$$

where a = the acceleration imparted on the ball, k = the size-koefficient of the magnetic acceleration, x = the x-coordinate, y = the y-coordinate, and g = the gravitational acceleration.

Precondition: A game is in progress and the ball comes into proximity of the magnet.

Postcondition: The ball's trajectory is altered according to specification.

## 12. Wormhole

Function: Transports the ball to another wormhole.

Description: Wormholes will act as a transportation device transporting the ball to the coordinates of a corresponding wormhole.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon contact with a wormhole, the ball will be instantly moved to the corresponding wormhole keeping its current velocity and vector.

Precondition: A game is in progress and the ball comes into contact with a wormhole.

Postcondition: The ball is moved to the new wormhole keeping the current velocity and vector.

## 13. Walls

Function: Walls will act as the standard impediment.

Description: Walls will block the path of the ball.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop.

Action: Upon contact with a wall, the ball bounces off at a trajectory determined by the angle of the wall and the balls original vector(ingoing angle = outgoing angle), e.g. hitting a vertical wall will reverse the direction of the vector in the x-plane.

**Formula:**

$V = V - 2 \cdot Normal_{wall} \cdot (Normal_{wall} \circ V)$ , where V is the vector of the ball.

Precondition: A game is in progress and the ball collides with a wall.

Postcondition: The ball bounces back according to the specification.

## 14. Keys

Function: Keys will be obtainable items that will, when all are acquired, open up the exit to the next level.

Description: Keys will be scattered around the level and the player must reach them and collect all of them before he can progress to the next level.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon contact with a key, the key will disappear and increase the key count by one, thus reducing the amount needed to open the exit to the next level.

Precondition: A game is in progress and the ball comes into contact with a key.

Postcondition: The key disappears and the counter for the obtained keys is increased by one.

## 15. Exit Point

Function: Loads the new level.

Description: When all the keys are obtained the exit point will be open and the new level will be loaded when the player gets there.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Upon contact with the exit point, the new level will be loaded and the player will be awarded time to the global countdown depending on the current level.

Precondition: A game is in progress, all the keys are obtained, and the ball comes into contact with the exit point.

Postcondition: The new level is loaded and the time is added to the timer according to specification.

## 16. Ability to collect power-ups

Function: The player can collect power-ups throughout the level.

Description: The player can collect power-ups during the whole playtime which in turn changes the gameplay.

Inputs: None.

Source: None.

Outputs: None

Destination: None

Action: The player can pick up power-ups which can change the gameplay.

Precondition: The player is playing the game.

Postcondition: The player changes the gameplay by collecting a  power-up.

## 17. The effect of acquiring a time increase power-up

Function: Increases the global countdown.

Description: Changes the global countdown by increasing it and giving a player more time to play the game.

Inputs: None.

Source: None.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The player comes in contact with the time increase power-up. The global countdown is increased by a certain amount.

Precondition: The player comes in contact with the time increase power-up.

Postcondition: The global countdown is increased.

## 18. The effect of acquiring a low gravity power-up

Function: Changes the ball's bouncing behavior.

Description: Once acquired it lowers the gravitational acceleration of the ball, resulting in changes of ball's bouncing behavior. The effect ceases after a given time.

Inputs: None.

Source: None.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The player comes in contact with the low gravity power-up. The gravitational acceleration of the ball is lowered, i.e. the acceleration g is reduced, and this will change the ball's bouncing behavior for the given time, effectively increasing the height of the bounce and the time it takes to get back down again.

Precondition: The player comes in contact with the low gravity power-up.

Postcondition: The ball's bouncing behavior changes.

## 19.The effect of acquiring a reverse gravity power-up

Function: Changes the direction of the gravitation force.

Description: Once acquired it changes the direction of gravitational force, turning the ceiling into the floor and vice versa. The effect ceases after a given time.

Inputs: None.

Source: None.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The player comes in contact with the reverse gravity power-up. The direction of the gravitational force is reversed, i.e. the acceleration g is negated, for a given time.

Precondition: The player comes in contact wit the reverse gravity power-up.

Postcondition: The gravitational force is changed.

## 20. The effect of acquiring a freeze power-up

Function: Stops the motion of dynamic mines.

Description: Stops the motion of all dynamic mines throughout the level. The effect ceases after a given time.

Inputs: None.

Source: None.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The player comes in contact with the power-up. The motion of dynamic mines stops for a given time.

Precondition: The player comes in contact with the power-up.

Postcondition: The  motion of dynamic mines thought the level is stopped.

## 21. Game progress

Function: The player tries to complete the game levels.

Description: The game provides a number of levels. If the player successfully finishes them the

player will be returned to the first level but now with increased difficulty.

Inputs: None.

Source: None.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: Once the player finishes the current level, the system advances to the next level When the player succeeds in finishing the levels, the system restarts the levels but with an increased difficulty.

Precondition: The player has the game installed and is currently playing it.

Postcondition: The player advances to the next level.

## 22. Displaying the game over screen

Function: The game over screen is displayed on the phone.

Description: Once the global countdown reaches zero, the game displays the game over screen.

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The global countdown reaches zero and the game over screen is displayed.

Precondition: The global countdown must reach zero.

Postcondition: The game over screen is displayed.

## 23. Add an entry to the high score

Function: The high score is changed and updated.

Description: When the player loses the game with enough points to replace an already existing entry on the high score, the player will be prompted to enter a signature in order to update the high score.

Inputs: A text representing the signature.

Source: The player

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The system takes input form the player, updates the high score and displays the high score on the phone.

Precondition: The player must have enough points in order to replace an entry on the high score.

Postcondition: The high score is updated and is displayed on the phone.

## 24. The high score

Function: Display the high score.

Description: The high score is displayed when the player loses the game without earning enough points to replace an already existing entry on the high score

Inputs: None

Source: None

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The system displays the high score for the player after the game ends.

Precondition: The game is completed.

Postcondition: The high score is displayed for the player to see

## 25. Receiving an interrupt event triggered by an incoming call

Function: Pausing the game.

Description: The game will enter the paused state when the phone receives a call

Inputs: An interrupt event that triggers the game to pause.

Source: The mobile phone.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The system receives an interrupt event and pauses the game.

Precondition: An interrupt event to occur.

Postcondition: The game is paused.

## 26. Receiving an interrupt event triggered by a sms/mms or some other form of message

Function: Pausing the game.

Description: The game will enter the paused state when the phone receives a sms/mms or some other form of a message

Inputs: An interrupt event that triggers the game to pause.

Source: The mobile phone.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The system receives an interrupt event, and pauses the game.

Precondition: An interrupt event to occur.

Postcondition: The game is paused.

## 27 Receiving an interrupt event triggered by a warning or a system message

Function: Pausing the game

Description: The game will enter the paused state when the phone generates a warning or some other kind of system message.

Inputs: An interrupt event that triggers the game to pause.

Source: The mobile phone.

Outputs: The graphic is updated representing the new system state

Destination: Main control loop

Action: The system receives an interrupt event and pauses the game.

Precondition: An interrupt event to occur.

Postcondition: The game is paused.

# System evolution

**Fundamental assumptions on which the system is based:**

- The platform on which it is used is a mobile phone running Java ME.
- The Java ME version on the phone used is off the same or higher version then the one used in implementing the game.
- The phone used have the standard button setup or equivalent input.

**Anticipated future changes to the system:**

As we start to implement our game there are some points to take in mind. The market of mobile phones is ever changing and new models are released in a never ceasing flow. The essence of mobile game development is for the game to work on as many of the models as possible. This is done to enable players to experience the arcade gameplay on their chosen phone. To this end, we are using the Java ME framework.

The Java ME framework biggest and greatest feature is its platform independence. In theory, all applications made in Java ME should be usable on all platforms having that framework. Even so all phones have some details which makes them different and this must be taken into account during implementation. The NetBeans IDE have specialized sub code to tailor applications to mobile phones. In this way we can focus on the software and only minor changes to the code is needed to adapt it to new phones.

This also takes away the problem of new hardware. As new hardware is released we only need to do small changes to the code to make it work. In this way we provide this gaming experience to future players. On the other hand we can't predict the needs of future users in the same way as the ones that is the target audience as of now. Time is the greatest enemy to overcome for games, as it is a very dynamic market of changing interests of the players. Even so we hope that future users see the charm of the nostalgic arcade action that is "Balls of steel".